

Hoai Le

OPENSTACK AND SOFTWARE- DEFINED NETWORKING

The Enormous Potential of Open Source
Software Collaboration

Bachelor's thesis
Information Technology

2017



South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Hoai Le	Bachelor of Engineering	September 2017
Title		79 pages
OpenStack and Software-Defined Networking The Enormous Potential of Open Source Software Collaboration		
Commissioned by		
XAMK, IT Department		
Supervisor		
Matti Juutilainen		
Abstract		
<p>Twenty-something years ago, we lived in a monolithically licensed software world, but time has flown by, and we are now living in a world of open source innovation. The aim of this work was to gain insights into two of the most popular emerging technologies, namely OpenStack and Software-Defined Networking, and to see how well these two open source solutions could collaborate.</p> <p>Throughout the theoretical part, cloud computing, OpenStack architecture, OpenStack core services, Software-Defined Networking architecture, and SDN-related technologies were researched. The outcome indicated that OpenStack and Software-Defined Networking could play well together. It also showed why people favored OpenStack, and why it has become one of the fastest growing open source communities.</p>		
Keywords		
cloud computing, OpenStack, Software-Defined Networking, OpenDaylight		

CONTENTS

1	INTRODUCTION	7
2	CLOUD COMPUTING	8
2.1	Cloud Computing in a Nutshell	8
2.2	Service Models	10
2.3	Deployment Models	12
2.4	Why Move to the Cloud?	14
3	OPENSTACK OVERVIEW	15
3.1	OpenStack – the Birth.....	15
3.2	What is OpenStack?	16
3.3	OpenStack – the Evolution	17
4	OPENSTACK ARCHITECTURE.....	18
4.1	A Look under the Hood.....	18
4.2	Keystone.....	22
4.3	Glance	23
4.4	Neutron.....	24
4.5	Nova	28
4.6	Horizon	30
4.7	Cinder	31
4.8	Swift.....	33
4.9	Other Projects.....	34
5	SOFTWARE-DEFINED NETWORKING.....	35
5.1	Where were We before SDN?	36
5.2	A Brief History of SDN	38
5.3	What is SDN and How Does It Address Traditional Networking Challenges?	38
5.4	How Does SDN Fit into the OpenStack Big Picture?	41

5.5	OpenFlow	42
5.6	Open vSwitch.....	44
5.7	OpenDaylight and OpenStack	46
6	PRACTICAL IMPLEMENTATION.....	50
6.1	Topology.....	50
6.2	Deploying OpenStack	51
6.2.1	Setting up the Environment.....	52
6.2.2	Installing and Configuring Keystone, Glance, Nova, Neutron, and Horizon	53
6.2.3	Launching an Instance.....	60
6.3	Integrating OpenDaylight into OpenStack.....	66
7	CONCLUSIONS	69
	REFERENCES	74

LIST OF ABBREVIATIONS

AAA	Authentication, Authorization, and Accounting
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
BMP	BGP Monitoring Protocol
BLOB	Binary Large Object
CLI	Command-line Interface
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EMS	Element Management Systems
FaaS	Fabric as a Service
GRE	Generic Routing Encapsulation
IaaS	Infrastructure as a Service
IPFIX	IP Flow Information Export
KVM	Kernel-based Virtual Machine
LACP	Link Aggregation Control Protocol
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
ML2	Modular Layer 2
NASA	National Aeronautics and Space Administration
NETCONF	Network Configuration Protocol
NIC	Network Interface Card
NFV	Network Function Virtualization
NTP	Network Time Protocol
ODL	OpenDaylight
ONF	Open Networking Foundation
OSGi	Open Service Gateway Initiative
OSPF	Open Shortest Path First
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PaaS	Platform as a Service

PAM	Pluggable Authentication Module
PKI	Public Key Infrastructure
RADOS	Reliable Autonomic Distributed Object Store
RBD	RADOS Block Device
REST	Representational State Transfer
RPC	Remote Procedure Call
RSPAN	Remote Switched Port Analyzer
SaaS	Software as a Service
SAL	Service Abstraction Layer
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSH	Secure Shell
STP	Spanning Tree Protocol
UI	User Interface
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
VTN	Virtual Tenant Network
VXLAN	Virtual Extensible LAN
WAN	Wide Area Network

1 INTRODUCTION

Cloud computing is rapidly flourishing and reshaping the academic computing. In the brave new IT world, the infrastructure is moving from the traditional dedicated way to the modern and dynamic cloud-based approach. Server virtualization commenced this move to cloud, and factors like globalization and outsourcing in which diverse teams need to collaborate in real time have accelerated the adoption of cloud-based applications.

Physical servers connect to networking devices like switches and routers to establish network connectivity. Thus, when servers move from physical to virtual, the networking landscape is also required to change. In addition, the traditional way firmly combined applications, servers, and networking, but the present-day IT infrastructure expects servers and networking to gain maximum flexibility so as to support complex applications.

Server virtualization gives cloud infrastructure a part of that flexibility. In order to fully enable the power of cloud computing, networking is required to be dynamic and scalable. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are two emerging technologies that get a lot of hype in the network industry nowadays, especially SDN. And they are said to deliver the flexibility and agility demanded by cloud computing. On the cloud platform side, OpenStack is also prominent. Since its launch in 2010, OpenStack has quietly become one of the fastest growing open source platforms for enterprise cloud infrastructure.

When both OpenStack and SDN are advancing swiftly and becoming platforms for innovation, it is important to understand the key technologies at their intersection. The goal of this project is to gain technology and implementation insights into OpenStack and SDN and to see how well these open source software applications can play together.

2 CLOUD COMPUTING

This chapter provides readers with the essential background to understand what cloud computing means, how it is classified based on service models and deployment models, and why it plays an important role in the IT sector.

2.1 Cloud Computing in a Nutshell

Cloud computing has probably got more attention over the past few years than any data center technology. However, if one ever tries to look for an authoritative definition of cloud computing, he will find out that the term entails lots of different notions. Even experts cannot agree on what constitutes the essence of this fundamental shift in technology. (Rhoton et al. 2014, 6.)

The most commonly utilized definition was articulated by the National Institution of Standards and Technology (NIST 2010):

“Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

According to Kepes (2011), NIST also points out five essential characteristics that define a cloud, including:

- On-demand self-service: End users can quickly sign up and get their wanted services without any postponements that have characterized traditional IT.
- Broad network access: End users can access the service via standard platforms such as desktops, laptops, and mobiles.
- Resource pooling: Multiple customers share a pool of resources.
- Rapid elasticity: Capability to scale up to meet customers' demands.
- Measure Service: Metered billing is delivered to customers.

As stated by Rhoton et al. (2014, 7–8), an informal survey of blogs and tweets as well as published literature on the subject of cloud computing was conducted, and it helped reveal how end users define cloud computing. These keywords include:

- Off-premise: The service is hosted and delivered from somewhere by a service provider, and it crosses both physical and security boundaries.
- Elasticity: Resources can be scaled up and down rapidly as they are required.
- Flexible billing: The charging and billing system must be flexible and vary in options. Customers can pay on a subscription basis, by actual consumption, or by the reservation of resources.
- Virtualization: Cloud computing and virtualization go hand in hand. Cloud computing leverages various virtualization mechanisms and achieves cost optimization.
- Service delivery: Cloud providers offer a great variety of cloud delivery models besides the three most common ones called Infrastructure as a Service, Platform as a Service, and Software as a Service. All of them usually provide programmatic interfaces in addition to user interfaces.
- Universal access: Pool resources with high levels of resilience are available to any end users regardless of their locations.
- Simplified management: Automatic provisioning, user self-service, and programmatically accessible resources help simplify administration to meet scalability requirements, expedite business processes, and facilitate integration into enterprise management frameworks.
- Affordable resources: Most companies cannot afford to build a data center or a private cloud. Therefore, they need an inexpensive cloud service offered by some provider.
- Multi-tenancy: Sharing resources is an important factor in achieving scalability and reducing costs, but cloud providers also need to ensure the isolation among tenants.
- Service-level management: Cloud services that come with only minimal or non-existent commitments are still considered to be cloud services, but

they will typically not be trusted for mission-critical applications to the extent that others might.

Nevertheless, neither the NIST formulation nor any interpretation of what cloud computing means is universally the definition.

2.2 Service Models

The most common classification uses the SPI (Software as a Service, Platform as a Service, and Infrastructure as a Service) model. They are often called the cloud computing stack because they are built on top of one another.

Figure 1 below illustrates a clear distinction among SaaS, PaaS, and IaaS, but in reality, the differences are blurred.

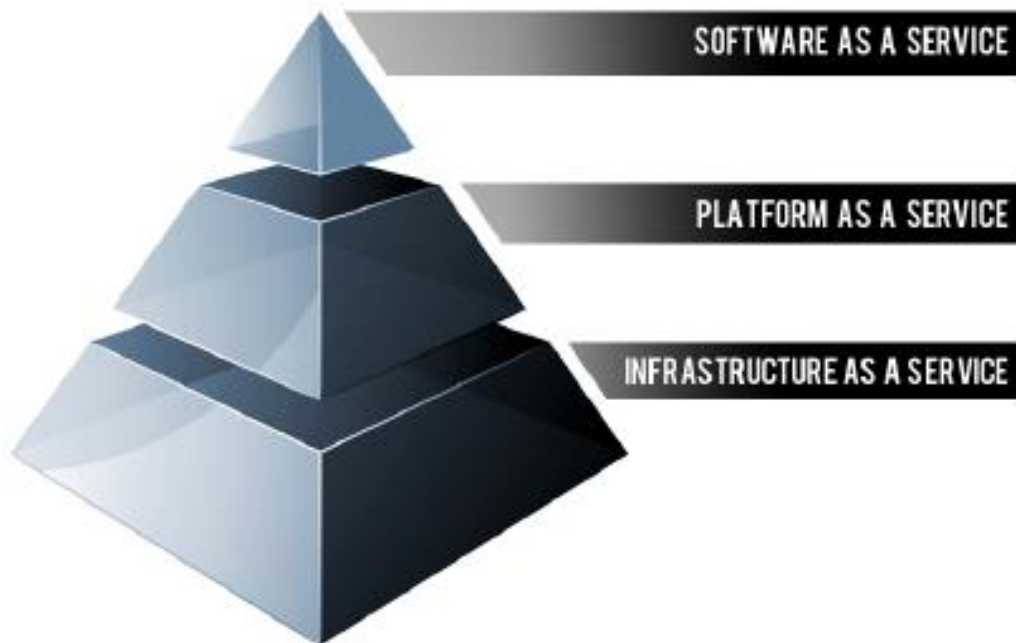


Figure 1. The cloud computing stack (Rackspace Support 2017)

In order to help readers comprehend how these three components are pertinent, Qrimp (2008) utilizes a transportation analogy:

“By itself, infrastructure isn’t useful – it just sits there waiting for someone to make it productive in solving a particular problem. Imagine the Interstate transportation

system in the U.S. Even with all these roads built, they wouldn't be useful without cars and trucks to transport people and goods. In this analogy, the roads are the infrastructure, and the cars and trucks are the platforms that sit on top of the infrastructure and transport the people and goods. These goods and people might be considered the software and information in the technical realm.”

Software as a Service (SaaS) delivers software applications over the Internet. It is said to be arguably the most mature model. All the services run on the cloud provider's infrastructure; licensing, application maintenance, software upgrades, and security patching are all done by the provider. This model entails a significant commitment to a cloud provider compared to IaaS and SaaS because it is not trivial to switch from one SaaS provider to another. (Kepes 2011, 5–13.)

SaaS can be considered as the prime candidate in some cases in which there is a significant interaction between the organization and the outside world, or there is a significant need for web or mobile access, or software is only used for a short time, etc. However, there are also situations where SaaS might not be appropriate, like when extremely fast processing of real-time data is needed, when legislation or other regulation does not allow data to be hosted externally, and so on. Salesforce and Netflix represent some of the best-known SaaS examples. (Sb.)

Platform as a Service (PaaS) is a cloud computing service that provides developers with resources to develop, launch, deliver, and manage apps without the need of buying, setting, and maintaining the underlying infrastructure. PaaS is analogous to SaaS, but rather than providing software delivered over the web, it is the platform in which end users create software applications delivered over the web. (Sb.)

PaaS is extremely useful when a lot of developers will be working together on a development project or there are external parties that need to interact with the development process. However, PaaS might not be ideal when, for example, the application needs to be highly portable with regard to where it is hosted, the

application performance requires customization of the underlying hardware and software, etc. Microsoft Azure, Google App Engine, and Force (delivered by Salesforce) are several well-known examples of PaaS providers. (Sb.)

Infrastructure as a Service (IaaS) is the most common and simplest cloud delivery model. Rather than building a data center, purchasing servers and networking equipment, customers rent virtual machines, storage, networking, and operating systems fully outsourced by a cloud provider. The boundary between IaaS and PaaS is blurred, but PaaS typically offers more programmer-oriented services, like code libraries, development tools, middleware, and so on. Another distinguishing factor is that IaaS can be obtained as public, private, or a combination of the two: hybrid infrastructure. (Sb.)

IaaS is a suitable solution for new organizations that do not have enough capital to invest in hardware, or for a specific line of business when there is a need of trial or temporary infrastructure, or when demand is capricious. There is a plethora of IaaS providers like Amazon Web Services, Google Compute Engine, and Rackspace. (Sb.)

2.3 Deployment Models

The SIP model classifies services based on the type of content that they offer, but the types of service providers can also be used for classification. In an ideal world, the consumer should be independent of the providers, so there is no reason to prefer one to another. That being said, relating to security, governance, invoicing, and settlement, consumers still need to consider what type of deployment model that a cloud provider provides. (Rhoton et al. 2014, 11.)

Figure 2 below depicts four types of cloud computing deployment models including public cloud, private cloud, hybrid cloud, and community cloud.

Cloud Deployment Models

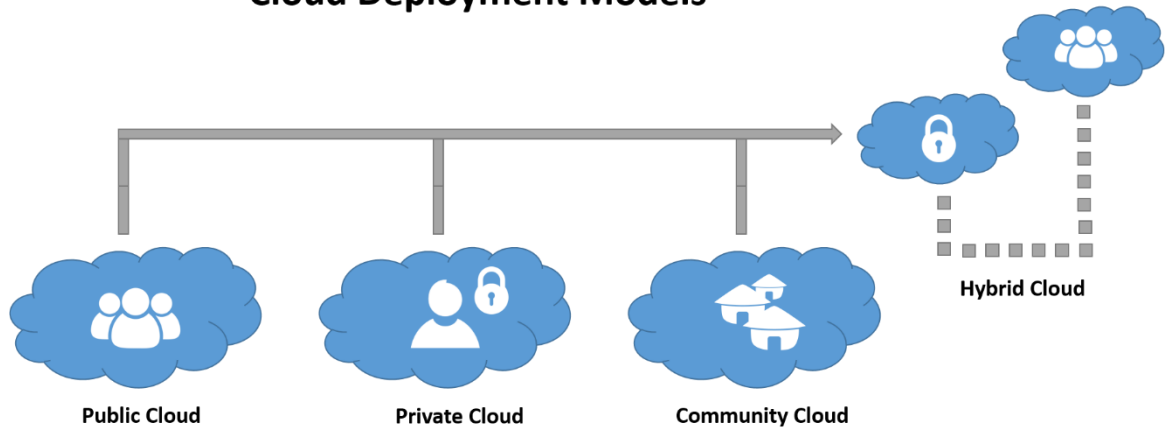


Figure 2. Cloud computing deployment models (Fu 2017)

Public cloud represents true cloud hosting. It delivers cloud services over a network for public usage and renders services and infrastructure to various customers. Public cloud is the best choice for business requirements in which managing load spikes, hosting SaaS-based applications, utilizing interim infrastructure for developing and testing applications are required. This model helps reduce capital overheads and operational costs. (CloudTweaks 2012.)

Private cloud is solely used by one organization since it is safeguarded by a firewall governed by an IT department (Fu 2017). It offers a tremendous level of security and control, but it does not bring much in terms of cost efficiency, because the company still needs to purchase, build, and maintain their own infrastructure. It can be hosted internally or externally as it can be owned, managed, and operated by an organization, a third party, or a combination of them. (Somepalle 2015.) Private cloud is an obvious choice when security, control, and data privacy are paramount to an organization.

Hybrid cloud is the combination of public and private clouds that are bound together but still remain as individual entities. Using this model, customers can take full advantage of security and data privacy inherited from a private cloud while still making good use of cost benefits by keeping shared applications and data on a public cloud. One of the use cases of this model is to handle cloud

bursting in which a private cloud cannot handle load spikes and demands a fallback option to support the load. (CloudTweaks 2012.)

A community cloud is shared by several organizations, such as banks and trading firms, with the same policy and compliance considerations but belongs to a particular community (Fu 2017). This model helps to reduce costs further compared to a private cloud because it is shared by a larger group.

2.4 Why Move to the Cloud?

When you are surfing on Facebook, you are using the cloud. When you are uploading photos into pins on Pinterest or browsing what others have pinned, you are in the cloud again. So why is everyone moving to the cloud?

Deshmukh (2016) states: “The expression ‘Being in the Clouds’ brings us comfort and a sense of superiority. Cloud computing is moving increasingly to a destination with no return: the consolidation as an essential for the future existence of the Internet world.”

Salesforce UK (2015) and Simplus (2014) make some good points in an attempt to answer that question:

- Because everyone is moving to the cloud. “But everyone else is doing it” is not a good excuse for a seven-year-old to follow along with the crowd, but it does not hold true for a business owner; in fact, it is one of the best reasons to do something. Cloud computing allows not only businesses to stay competitive against each other in an equal playing field, but also small businesses to compete against large ones.
- Because cloud computing means anytime, anywhere. It allows you to have access to your stored data whenever and wherever as long as you have access to the Internet. And because your data is stored in a cloud, anyone with permissions can access the data and work on the same project simultaneously. Therefore, it helps you to save time.

- Because it is always there. By utilizing cloud computing, companies do not need to worry about designing a disaster backup plan, because cloud providers will take care of it.
- Because it is environmentally friendly. Cloud computing reduces the need for in-house servers as well as the constant climate control involved in maintaining them, so it helps to decrease oversized carbon footprints.
- Because it alleviates security issues. Losing laptops is one of the billion-dollar-business problems because of the sensitive data inside it. With cloud computing, you can access your data no matter what happens to your laptop.

There is a ton of articles on the advantages and disadvantages of cloud computing for people who are considering adopting cloud technologies. However, the benefits that businesses can reap from cloud computing might easily outweigh the drawbacks. While their motivations and aims vary, businesses of all sizes and shapes are turning to cloud computing.

3 OPENSTACK OVERVIEW

This chapter gives readers a brief overview of how the OpenStack project was created, who were involved in the birth of OpenStack, what OpenStack is, and how OpenStack is evolving.

3.1 OpenStack – the Birth

Back in 2009, NASA was approached by the US Government to harness new technologies that would assist in the newly passed Open Government Initiative. NASA had already been developing NASA.net which was a unified technology platform used across all of NASA's Web projects. NASA developers started to create a set of generic, on-demand, API-driven compute and storage systems, namely Infrastructure as a Service, though cloud computing was still in its developmental stage. The NASA.net team, which later called their project NASA Nebula, thought of a significant chance to become a cloud provider in their own right and shifted their focus on creating an open source cloud compute controller.

As IBM decided to outsource its PC operating system to Microsoft, the NASA.net team chose to foster an open-source community around Nebula's code and release the code via a more indulgent license than the traditional and more restrictive NASA Open Source Agreement. (Bloomberg 2012.)

On the other hand, Rackspace, the public cloud provider that was thinking about the same compute controller, was soon attracted to the nascent developer community. After an email exchanged, Rackspace and NASA decided to join forces, and the OpenStack project officially started in October 2010. (Bentley 2015.)

3.2 What is OpenStack?

OpenStack is an open source platform that lets us build an Infrastructure as a Service (IaaS) cloud running on commodity hardware. It is a control layer sitting above all the virtualized layers and provides a consistent way to access resources regardless of whether the hypervisor technology used underneath is KVM, Xen, or VMware. OpenStack combines a large number of interrelated open source tools, which are known as projects, to deliver various components for a cloud infrastructure solution. Six of the projects that handle the core cloud-computing services including compute, networking, storage, identity, and image can be bundled with a dozen of additional ones to deliver a desired cloud. (RedHat 2017.)

OpenStack is written primarily in Python and available freely under an Apache 2.0 license. Each OpenStack service provides a REST API so all the resources (compute, storage, and networking) can be managed through a dashboard. It gives administrators control while empowering users to provision resources through a web interface or a command-line client. (Fifield et al. 2014, Preface XV.)

OpenStack is developed and released around a six-month cycle. Every release is identified by a codename ordered alphabetically. Those codenames are cities or countries near the place where the corresponding OpenStack summit takes place

like Austin, Bexar, Cactus, Diablo, Essex, etc. (OpenStack 2017h.) OpenStack believes in open source, open design, and open development; all in an open community so anyone can participate (Fifield et al. 2014, Preface XV).

3.3 OpenStack – the Evolution

The OpenStack project was closely managed by Rackspace and its 25 initial partners for the first two years. However, Rackspace decided to transfer the intellectual property and governance of the OpenStack project into a non-profit member-run foundation named the OpenStack Foundation. (Bentley 2016.) NASA has also stepped out of the OpenStack limelight because they preferred a role in the shadows for a number of reasons. Nevertheless, there is no reason why OpenStack will not become the core of cloud efforts at NASA and across the federal government. (Bloomberg 2012.)

OpenStack is now backed by a big roster of IT vendors including Rackspace, RedHat, IBM, Intel, SUSE, Huawei, Canonical, Dell, HPE, Mirantis, and so on. In other words, a lot of companies have placed their bets on OpenStack. Furthermore, the OpenStack community included 68837 registered members, 649 supporting organizations, 181 countries represented, 116 global user groups by the end of 2016 (OpenStack Foundation 2016, 8). It has become one of the fastest growing open source communities.

Like Linux, OpenStack is a bit of miracle: not only because it was somewhat tangential to the agency's mission, but the NASA bureaucracy was also unsuited to the creation of something openly shared with the rest of the world. And more remarkably, OpenStack, a project created by NASA, has quickly found its place among the giants in the technology world. "This could have fallen apart in a million different ways, from the beginning. In fact, it all seemed impossible," said Rick Clark, who worked at Rackspace when OpenStack was in its infancy and now helps drive the project at Cisco. (Metz 2012.) Among Eucalyptus, Apache CloudStack, Open Nebula—all the open source cloud platforms that have matured, they chose to call OpenStack "the Linux of the Cloud".

4 OPENSTACK ARCHITECTURE

OpenStack, in short, is awesome, but one might wonder what the technology behind the OpenStack project is. This chapter covers a deep dive into the OpenStack architecture, its core services, what they are, and how they work.

4.1 A Look under the Hood

OpenStack embraces a modular architecture, so there are a lot of moving parts inside the project. The modular nature of OpenStack is highly appreciated by lots of users. “Install what you need. Saves a lot of cost. Tons of documentation available to do it yourself.” (OpenStack 2017a, 13.)

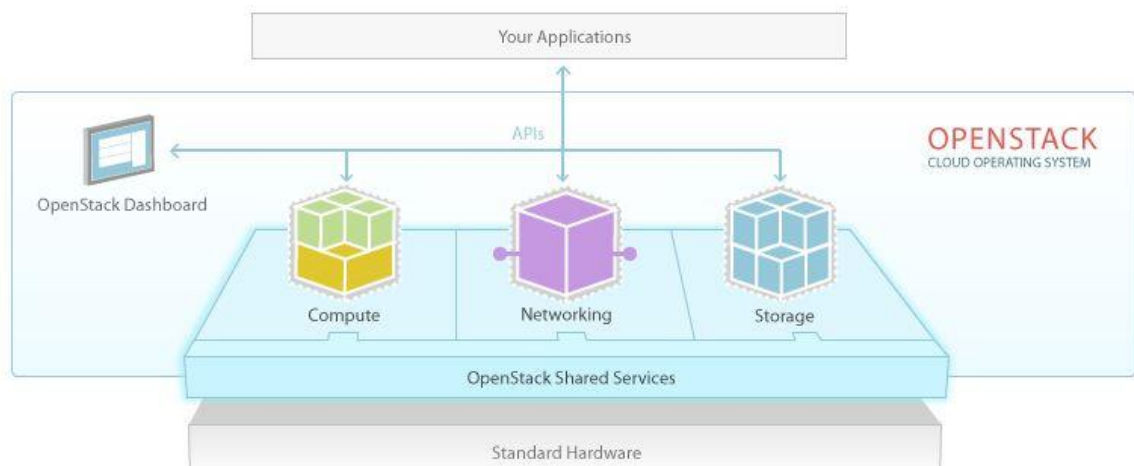


Figure 3. OpenStack architecture (Cfheoh 2011)

Figure 3 illustrates how simple end users can view OpenStack: there are large pools of compute, networking, and storage resources in a datacenter, and OpenStack controls them and gives end users the ability to provision those resources through a web interface. However, in order to provide that elegant look, a lot of services are needed to build up an OpenStack cloud. Figure 4 below shows a higher-level overview of OpenStack core services and their relationship with each other. It also clearly demonstrates what each service provides to boot up a running instance.

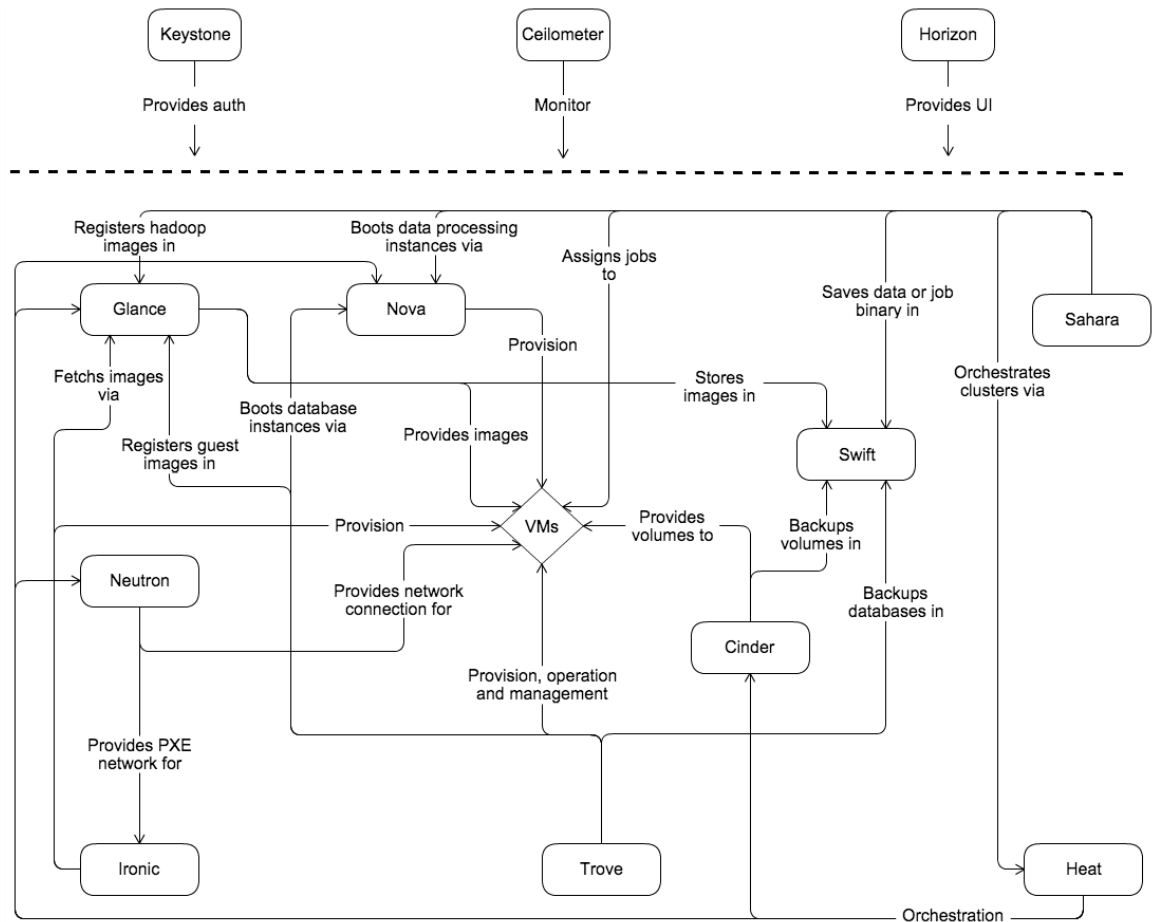


Figure 4. OpenStack conceptual architecture (OpenStack 2016a)

OpenStack consists of dozens of independent parts which are called the OpenStack services. All of them authenticate through a common Identity service. Individual services interact with each other through public APIs, except where privileged administrator commands are needed.

Internally, each OpenStack service comprises of several processes. All services have at least one API process listening, preprocessing, and passing API requests on to other parts of the service. That said, the Identity service is an exception as the actual work is done by distinct processes. An AMQP message broker is utilized for communication between the processes of one service. A service's state is stored in a database. There are several message broker and database solutions, such as RabbitMQ, MySQL, MariaDB, and SQLite. (OpenStack 2017c.)

Users can access OpenStack via the web-based UI implemented by the Horizon Dashboard or via CLIs by issuing API requests. Ultimately, all the access methods issue REST API calls to OpenStack services. (OpenStack 2017c.)

Figure 5 shows the most common logical architecture of an OpenStack cloud.

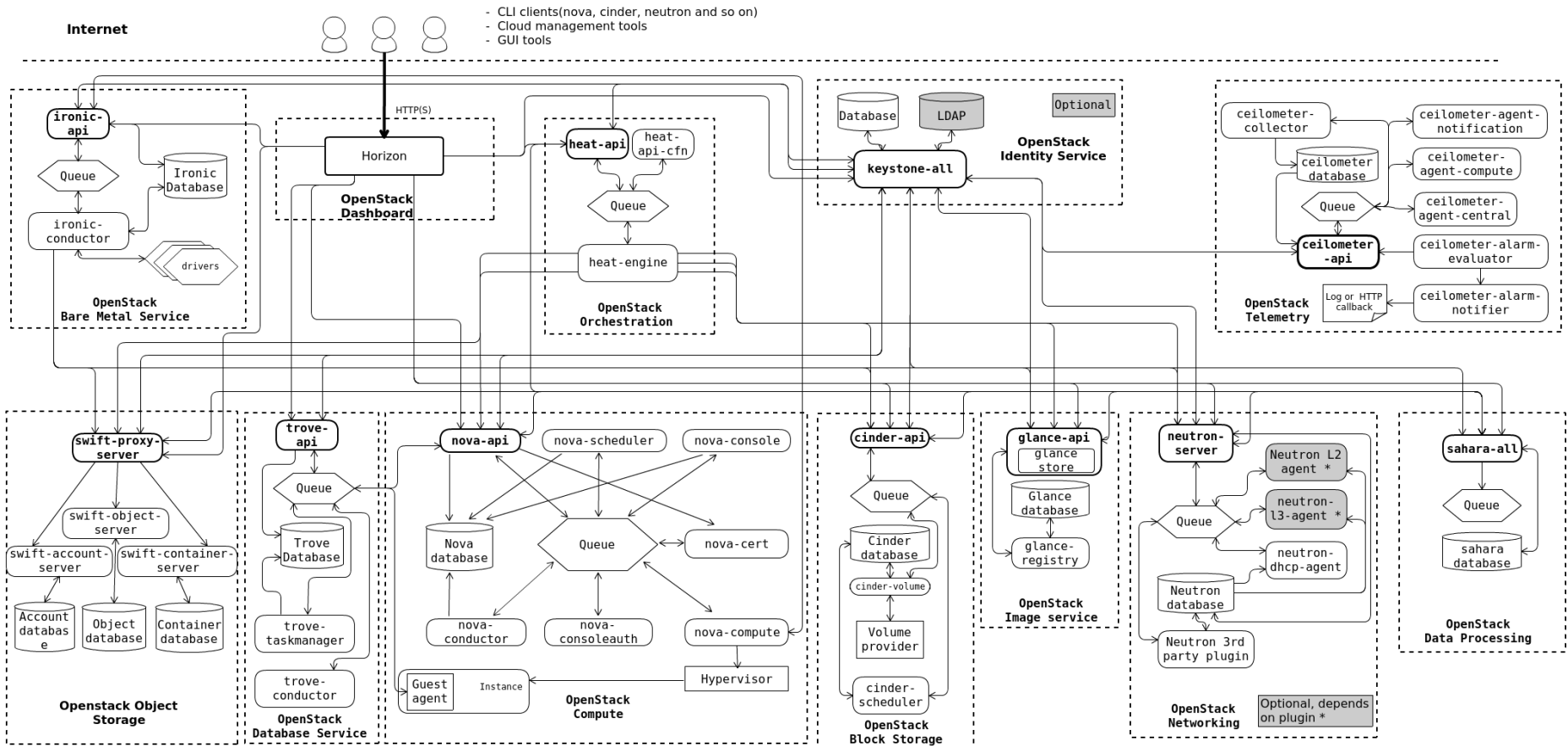


Figure 5. OpenStack logical architecture (OpenStack 2017c)

4.2 Keystone

Keystone is the OpenStack Identity service that provides authentication and authorization services throughout the entire cloud infrastructure. From an architectural perspective, Keystone is the simplest service in the OpenStack composition. (Khedher 2015, 4.)

Keystone utilizes a combination of domains, projects (tenants), users, and roles. Tenant is the old term for a project. Starting from API version 3, the term 'project' is preferred.

- A domain is an Identity API v3 entity. It is a collection of projects, groups, and users, and it defines administrative boundaries for managing OpenStack Identity entities.
- A project is the base unit of ownership in OpenStack. All resources should belong to a specific project, and a project itself should belong to a specific domain.
- A role is composed of a set of rights and privileges.
- A user can be associated with projects, roles, or both.

Keystone keeps a catalog of services and endpoints of all the OpenStack services. All the services have different API endpoints; this is advantageous as an end user only needs to know Keystone's address to interact with the cloud. (Radez 2015, 2.)

```
[root@dgnode3 ~]# source openrc
[root@dgnode3 ~]# openstack endpoint list
```

ID	Region	Service Name	Service Type	Enabled	Interface	URL
ce0f81cdd74341469736f2aa0a53085b	None	keystone	identity	True	public	http://dgnode:5000/v3
0c469862653e46c481afacee66d388d3	None	keystone	identity	True	internal	http://dgnode:35357/v3
a452badcc5714bb4b48c52930361c909	None	keystone	identity	True	admin	http://dgnode:35357/v3
fa3d45ebec4d49e982b3a3220587bb7d	None	swift	object-store	True	public	http://dgnode:8080/v1/AUTH_\$(tenant_id)s
4cef3af47d6f4356a1125a868b6ee96f	None	swift	object-store	True	internal	http://dgnode:8080/v1/AUTH_\$(tenant_id)s
2d6f2c7e0294425dbf6b30224090f73c	None	swift	object-store	True	admin	http://dgnode:8080

Figure 6. Keystone catalog (Deepakrghuge 2015)

Figure 6 is an example of how Keystone itemizes OpenStack services including itself. It knows all the IDs, regions, service names, service types, interfaces, and URLs.

Keystone uses a number of authentication mechanisms including username/password, token-based. Additionally, it also has pluggable support for being integrated with an existing backend directory like LDAP or PAM. (Khedher 2015, 4.)

4.3 Glance

Glance is the OpenStack Image service that serves as an image registry. After we are authenticated and authorized, we first need to have a disk image to launch a virtual machine. However, installing an operating system on every single machine is tedious. Cloud computing has streamlined the procedure by building a registry that contains pre-installed disk images. Glance is that registry in OpenStack. (Radez 2015, 4.) Figure 7 below illustrates Glance architecture.

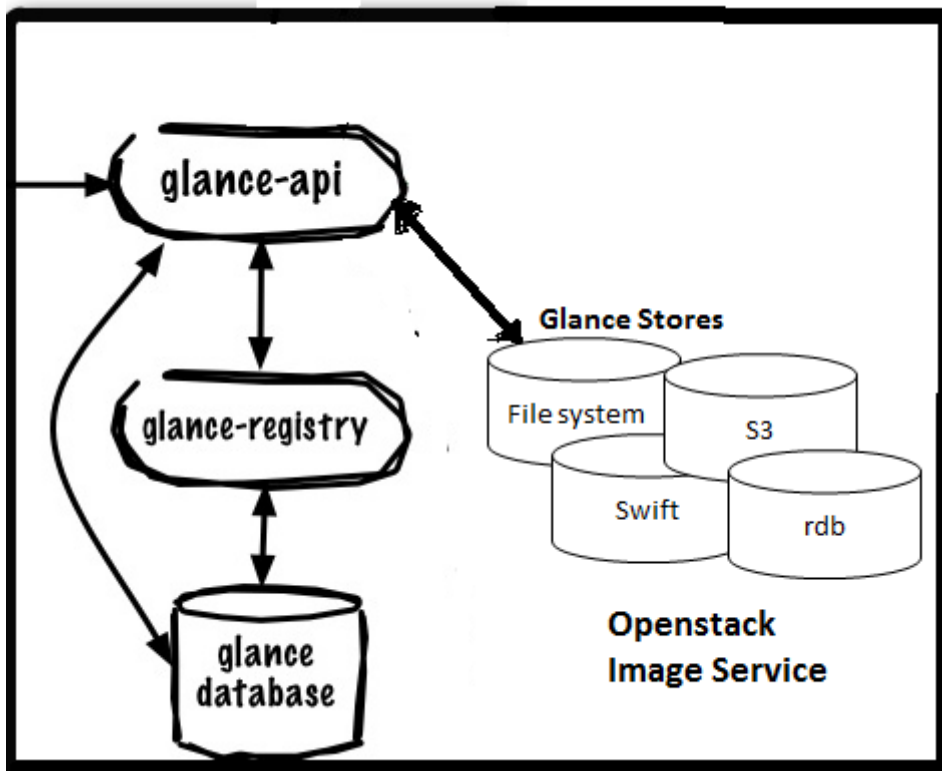


Figure 7. OpenStack Glance (Gupta 2013)

According to Radez (2015, 4), Glance includes the following components:

- glance-api accepts API calls for images from end users or other services.
- glance-registry stores, processes, and retrieves image metadata like size, type, format, and so on.
- Glance database stores image metadata. It supports multiple backends, such as MySQL, SQLite, and MongoDB.
- Glance stores, or Glance storage backends, support normal file systems. They can be Swift or Ceph RBD.

Glance provides discovery, registration, and retrieval services for VM images. (OpenStack 2017i). Images stored in Glance have an operating system installed, but they have ssh host key, MAC address, and so on removed. They are generic disk images that can be used and launched over and over again. Image customization at startup can be done with cloud-init. (Radez 2015, 4.)

4.4 Neutron

Neutron, previously known as Quantum before the Havana release, is an OpenStack project that provides Network as a Service (NaaS). Prior to Quantum, OpenStack had a flat and simple networking architecture controlled by Nova networking, a subcomponent of Nova. Neutron allows tenants to create advanced virtual network topologies that include services like firewalls, load balancers, virtual private networks, etc. It is one of the most complex components among OpenStack projects since it is built around the core networking concepts.

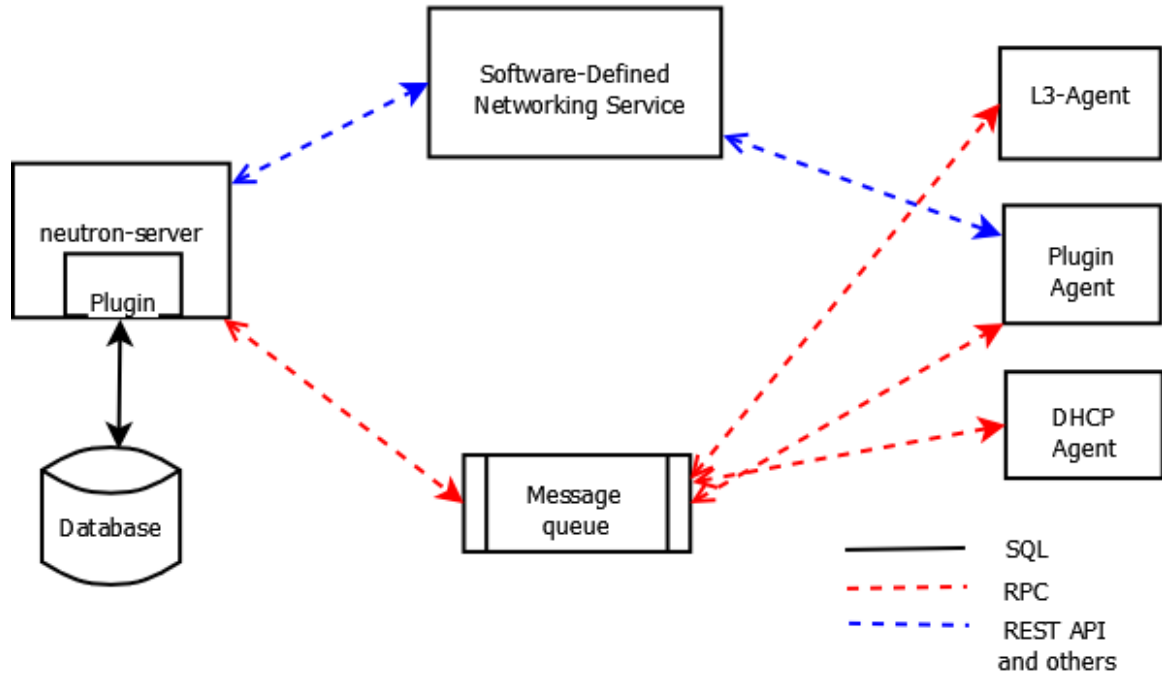


Figure 8. OpenStack Neutron (OpenStack 2017d)

The main process of the OpenStack Networking service is called neutron-server that is a Python daemon exposing Neutron API and passing tenant requests to a suite of plug-ins for additional processing (OpenStack 2017d).

Figure 8 shows the architectural and networking flow diagram of Neutron components. These major components contain:

- Neutron server (including neutron-server and neutron-*-plugin) accepts API requests from other OpenStack components and exposes all the internal networking details in terms of networks, subnets, ports, and so on (Packt 2017). The neutron-server communicates with a persistent database indirectly through plugins. These plugins use AMQP to assist with the communication. (OpenStack 2017d.) A plugin is a collection of Python modules that implement a standard interface. That interface

accepts and receives API calls, and connects with devices downstream. (Packt 2017.)

- Message queue is responsible for routing messages between neutron-server and agents as well as the database in order to store a plugin's state for a particular queue (Khedher 2015, 11).
- Agents: While the Neutron server acts as the centralized controller, all the networking commands and configuration are done on compute and network nodes. Agents are entities that carry out the actual work. They receive messages and instructions from the Neutron server via the message bus. (Sriram 2015.) There are several popular Neutron agents such as neutron-dhcp-agent providing DHCP services to tenant networks, neutron-l3-agent providing L3/NA forwarding for external network access of VMs on tenant networks, neutron-lbaasv2-agent providing load balancing services, and so on. (OpenStack 2017d.)
- There could also be network provider services that provide additional networking services to tenant networks like SDN.

Neutron provides networks, subnets, and routers as object abstractions. Each of them has functionality that mimics its physical counterpart. (OpenStack 2017e.)

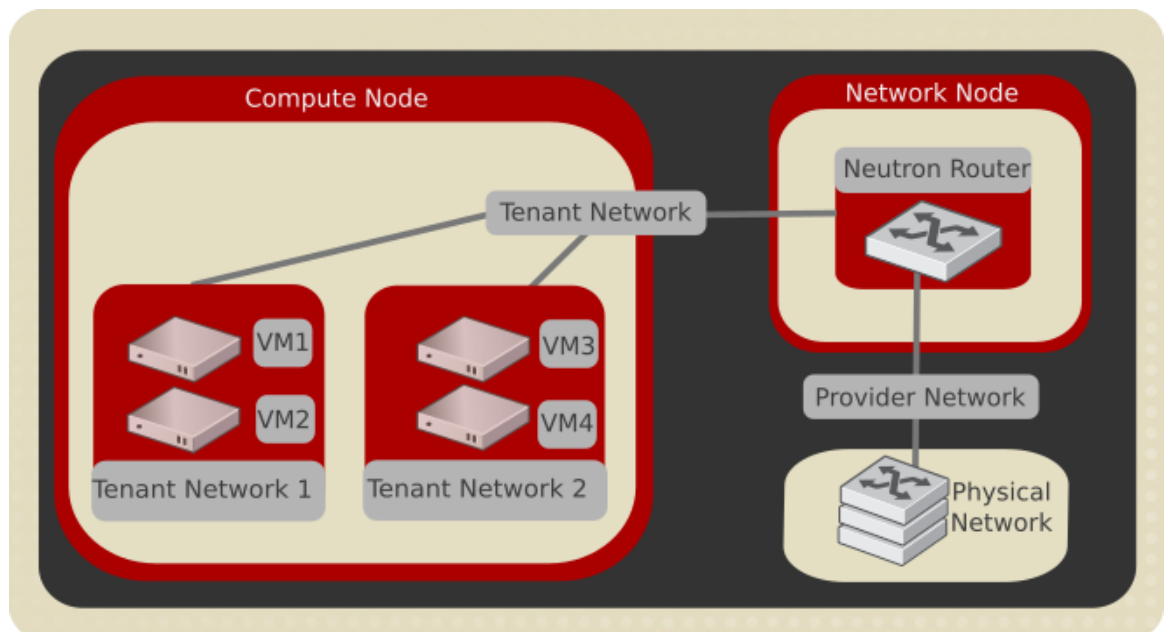


Figure 9. Neutron provider networks and tenant networks (OpenStack 2016b)

There are two network types implemented within Neutron: provider networks and tenant networks as shown in Figure 9. The major difference between them revolves around who provisions them. Users create and configure tenant networks for connectivity within projects, whereas OpenStack administrators create provider networks that are consumed by tenants.

By default, tenant networks are fully isolated and are not shared among projects. Users have full control over networking topology. Virtual routers are responsible for routing traffic among networks within a project or between tenant networks and external networks. Within a project, Neutron fully provides users with self-service subnet, DHCP, DNS, Layer-3 routing, firewall, load balancer, VPN modification. Tenant networks are not routable from the outside world, so users utilize floating IP addresses to access instances. OpenStack supports four types of network isolation and overlay technologies: flat, VLAN, GRE, and VXLAN. (RedHat 2014.) On the other hand, provider networks directly map to existing physical networks in the data center. Their network types are regularly flat or VLAN.

Neutron API is pluggable, and it means users can write and use any plugins and drivers to extend network functionality. Neutron classifies plugins into two categories: core and service. Core plugins implement the core Neutron API. Service plugins provide additional services such as Layer-3 router, load balancer, VPN, firewall, and metering. (Rao 2015.)

ML2 (Modular Layer 2) bundled with OpenStack is the most important core plugin. It supports a wide variety of Layer 2 technologies and allows multiple vendor technologies to coexist. Before ML2 got involved in the OpenStack big picture, Neutron was limited to use a single core plugin at a time. ML2's purpose is to replace and deprecate two monolithic plugins: linuxbridge and openvswitch. Their L2 agents, however, continue to work with ML2. (Denton 2015, 55.)

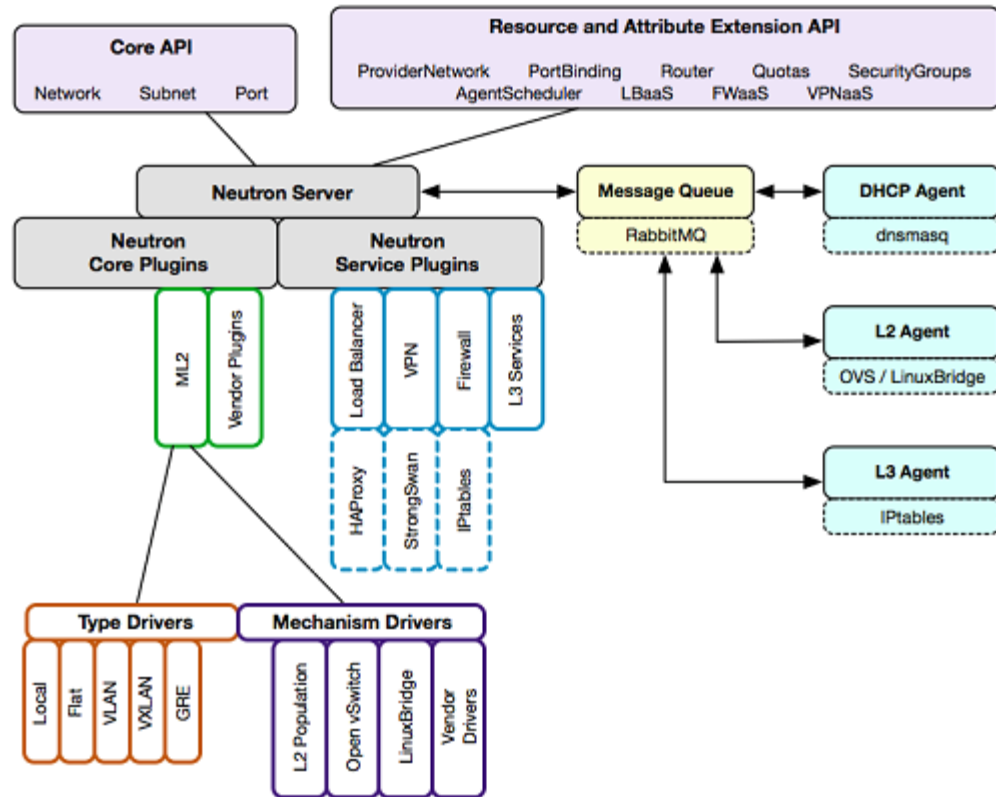


Figure 10. Modular Layer 2 architecture (Denton 2015)

ML2 introduces the concept of drivers. Drivers separate extensible sets of network types being implemented and mechanisms implementing these network types. Drivers are divided into type drivers and mechanism drivers as shown in Figure 10. Type drivers maintain type-specific network state and perform provider network validation and tenant network allocation. Supported network types include local, flat, VLAN, VXLAN, and GRE. Each mechanism driver manages a networking mechanism. A mechanism driver is responsible for taking the information established by the type driver and ensuring this information is properly applied. (OpenStack 2017f.) Multiple mechanism drivers can be simultaneously utilized. There are three mechanism driver types: agent-based, controller-based, and top-of-rack.

4.5 Nova

Nova is the OpenStack compute service and the most original core component of OpenStack. From an architectural perspective, Nova is the most complicated and distributed component of OpenStack.

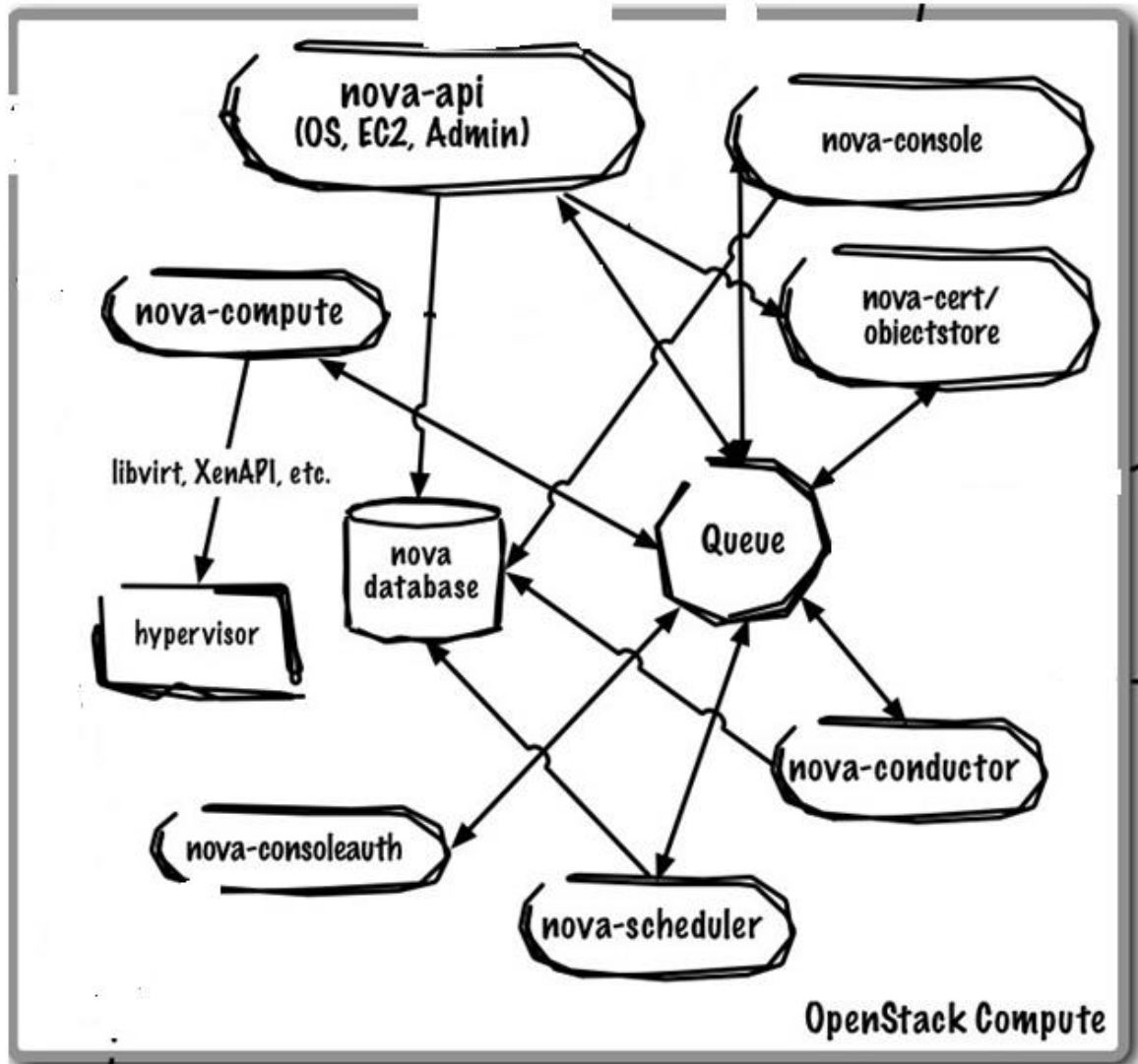


Figure 11. OpenStack Nova (Paternò 2015)

In order to turn an end user's API request into a running VM, a large number of processes, as shown in Figure 11, need to cooperate. As identified by Gupta (2013), these processes are:

- nova-api is a RESTful API service that accepts and responds to end users' requests.
- nova-cert provides the certificate management.
- nova-compute is primarily a worker daemon creating and terminating instances. It ensures that the state of new instances is maintained in the database.

- nova-console allows end users to access their instances console through a proxy.
- nova-consoleauth provides authentication for nova consoles.
- nova-conductor is a server daemon that enables OpenStack to function without compute nodes accessing the database. It conceptually implements a new layer on top of nova-compute.
- nova-scheduler takes an instance's request from the queue and determines where it should run, specifically which compute host it should run on.
- queue provides a central hub for passing messages between daemons. The queue is usually implemented with RabbitMQ.
- Nova database stores most of the build-time and run-time infrastructure state.

Nova encompasses more server processes than any other project in the OpenStack composition, and each process performs a different function. Externally, Nova has a REST API like other OpenStack projects, while Nova components internally communicate via an RPC message passing mechanism. The RPC messaging is carried out by the oslo.messaging library that is an abstraction lies on top of the message queues. Most of the Nova's major components can be distributed among multiple servers, and they have a manager listening for RPC messages. The only exception is nova-compute because it has a single process run on the hypervisor which it is managing. (OpenStack 2017g.)

4.6 Horizon

Horizon is the OpenStack Dashboard service which provides a web-based user interface to OpenStack services, such as Nova, Keystone, Neutron, as shown in Figure 12.

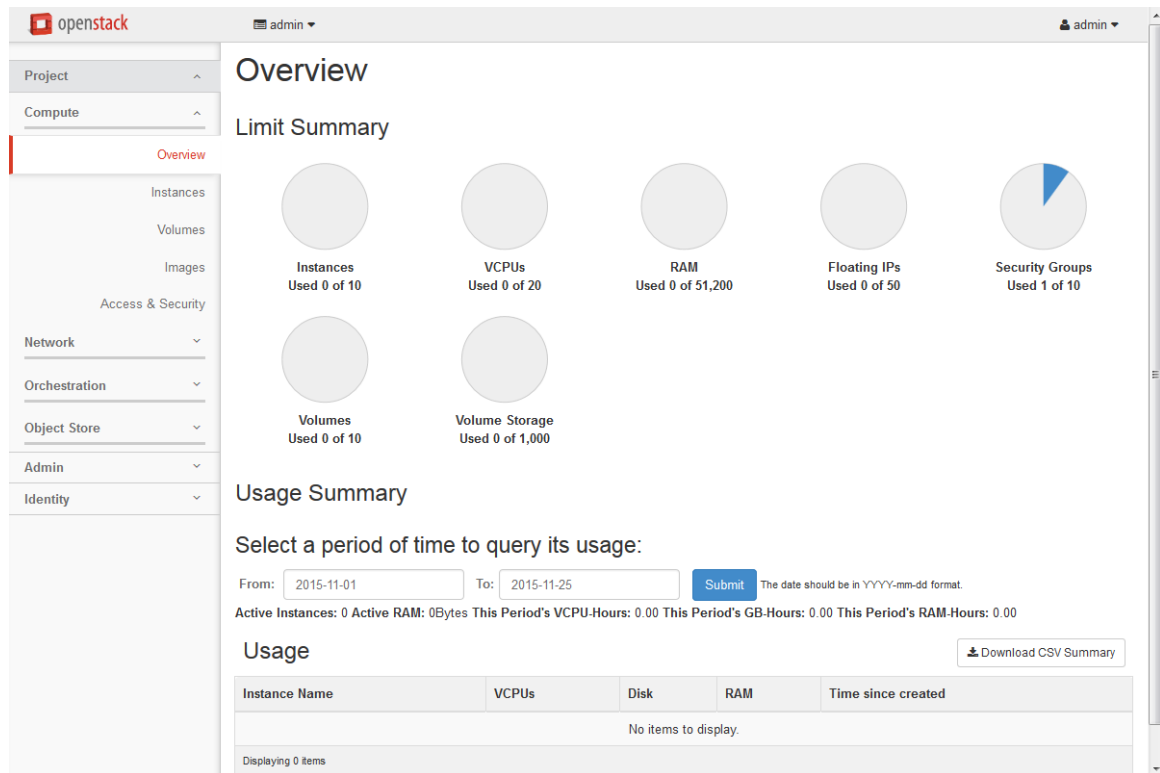


Figure 12. OpenStack Horizon (OpenStack 2017b)

Originally, Horizon was started as a single app to manage OpenStack's compute project. Therefore, all it needed was a set of views, templates, and API calls. After that, it gradually grew to support multiple OpenStack projects and APIs, therefore it was rigidly arranged into dash and syspanel groupings. (OpenStack 2017b.)

4.7 Cinder

Cinder is the OpenStack Block Storage service. It is the replacement of the nova-volume service since the Folsom release.

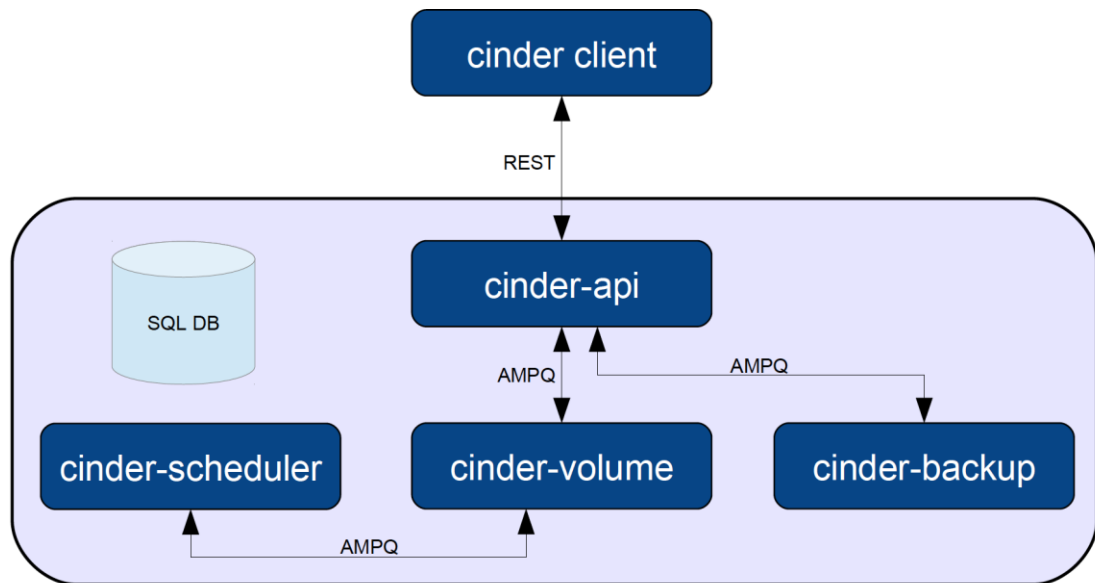


Figure 13. OpenStack Cinder (Hui 2013)

Figure 13 shows Cinder structure. As stated by Hui (2013), Cinder consists of the four following daemons and two other components:

- cinder-api handles, responds, and places requests in the message queue.
- cinder-scheduler reads requests from the message queue, and schedules and routes them to the appropriate volume service. It can be a simple round-robin scheduling or more sophisticated through the use of filter scheduler depending on how it is configured.
- cinder-volume manages the interaction with block storage devices.
- cinder-backup provides the ability to back up a Cinder volume to various backup targets.
- The database provides state information.
- RabbitMQ server provides the AMQP message queue.

Cinder allows users to create and delete block devices as well as to manage the attachment of block devices to VMs. The integration of Cinder with Nova handles the actual attachment and detachment. Cinder is suitable for performance-sensitive scenarios like database storage, expandable file systems, and assisting a VM in getting access to raw block-level storage. (RedHat 2015, 11.)

4.8 Swift

Swift is the OpenStack Object Storage service. It provides redundant and scalable distributed object storage. Distributed means that each piece of data is replicated across a cluster of storage nodes (Ukov 2012). Swift is primarily utilized to store and retrieve BLOBs, it means static data such as VM images, videos, photos, emails, files, backups, and archives.

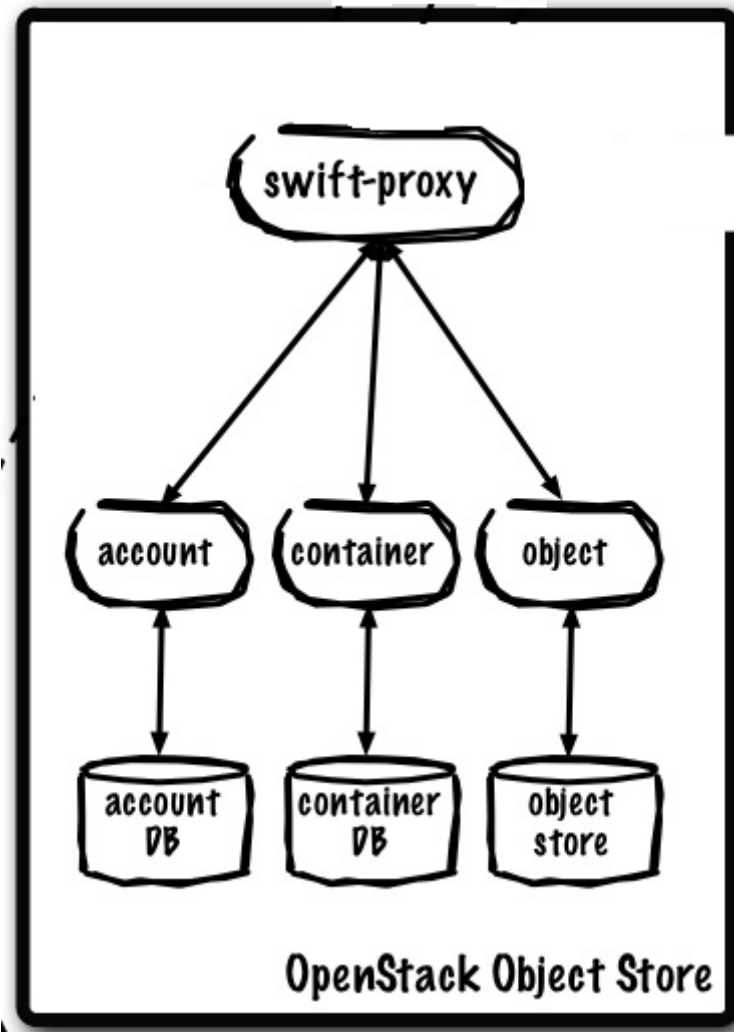


Figure 14. OpenStack Swift (Gupta 2013)

According to Gupta (2013), Swift major components include swift-proxy, swift-account, swift-container, and swift-object as shown in Figure 14.

- swift-proxy exposes the public API, provides authentication, and is responsible for handling and routing incoming requests.
- swift-object stores, retrieves, and deletes objects.
- swift-account is in charge of listings of containers.
- swift-container handles listings of objects.

Swift also provides a REST API to enable access to OpenStack files and organizes data in containers. It utilizes open source programs like Python and Rsync to perform day-to-day tasks, scan files for corruption, and relocate files in the event of node downtime or drive failure. Swift can be used with OpenStack or as a standalone product. (Olow 2017.)

4.9 Other Projects

According to OpenStack (2017a), OpenStack's core services, including Nova, Neutron, Swift, Cinder, Keystone, and Glance, have grown to almost total adoption by all clouds in production. The runner-ups are Heat (Orchestration), Telemetry (Ceilometer), Rally (Benchmark Service), Ironic (Bare Metal), Designate (DNS Service), Manila (Shared File Systems), Trove (Database Service), Kolla (Containerized Deployment), Magnum (Containers Service), Murano (Application Catalog), Sahara (Data Processing), Barbican (Key Management), Mistral (Workflow Service), Zaqr (Message Service), and Congress (Governance Service). Nevertheless, they are only a few names among a great number of OpenStack projects that will be still increasing. Figure 15 is a picture of OpenStack projects and their mascots. It shows us how fast OpenStack is growing, and how awesome the OpenStack community is.

















































 BARBICAN	 NOVA
 CHEF OPENSTACK	 OPENSTACK CHARMS
 CINDER	 OPENSTACK-ANSIBLE
 CLOUDKITTY	 OSLO
 COMMUNITY APP CATALOG	 PACKAGING-DEB
 CONGRESS	 PACKAGING-RPM
 DESIGNATE	 PUPPET OPENSTACK
 DOCUMENTATION	 QUALITY ASSURANCE
 DRAGONFLOW	 RALLY
 EC2-API	 REFSTACK
 FREEZER	 RELEASE MANAGEMENT
 GLANCE	 REQUIREMENTS
 HEAT	 SAHARA
 HORIZON	 SEARCHLIGHT
 I18N	 SECURITY
 INFRASTRUCTURE	 SENLIN
 INTEROP	 STABLE BRANCH MAINTENANCE
 KARBOR	 SWIFT (DRAFT)
 KEYSTONE	 TACKER
 KOLLA	 TELEMETRY
 KURYR	 TROVE
 MAGNUM	 UX
 MANILA	 VITRAGE
 MISTRAL	 WATCHER
 MONASCA	 WINSTACKERS
 MURANO	 ZAQAR
 NEUTRON	

Figure 15. OpenStack services and their mascots (Martinelli 2017)

5 SOFTWARE-DEFINED NETWORKING

This chapter discusses where we were before Software-Defined Networking, how it emerged in the networking industry, what it is, and where it is taking us. It also introduces several SDN-related technologies including OpenFlow, Open vSwitch,

and OpenDaylight. One of the most crucial parts in this chapter is how and where SDN fits into the OpenStack picture.

5.1 Where were We before SDN?

As stated by Stretch (2013), a traditional network is built and operated based on the three networking planes like shown in Figure 16 below:

- Data plane (also known as forwarding plane) is where packets are forwarded from input to output.
- Control plane is where information about the network is learned and gathered by using various protocols, such as STP, OSPF, etc. It determines how packets should be forwarded in the data plane.
- Management plane enables network administrators to manage and configure the control plane using some mechanisms like CLI, SNMP, etc.

The most obvious problem faced in the control plane is interoperability. Cloud operators clearly do not want single vendor lock-in; they want to optimize the cost and achieve flexibility. However, while standards exist for most protocols, each vendor supports the standards in a different way. Even for the same vendor, a protocol's behavior might vary among releases. This challenge leads to incompatibility and limits the intelligence that can be built in a control plane. (Subramanian et al. 2016, 27–28.)

Another issue of the control plane is scalability. Control plane entities (protocols and so on) run on networking devices, but those devices have limited compute resources. Therefore, this challenge might hinder the control plane processing in large-scale networks. Traditional scaling control plane methods were to either do a full hardware upgrade of network devices or to do a partial upgrade on control plane processor cards. In cloud computing, scaling up and down based on demand is one of the most crucial factors, so this issue is also a serious challenge. (Sb.)

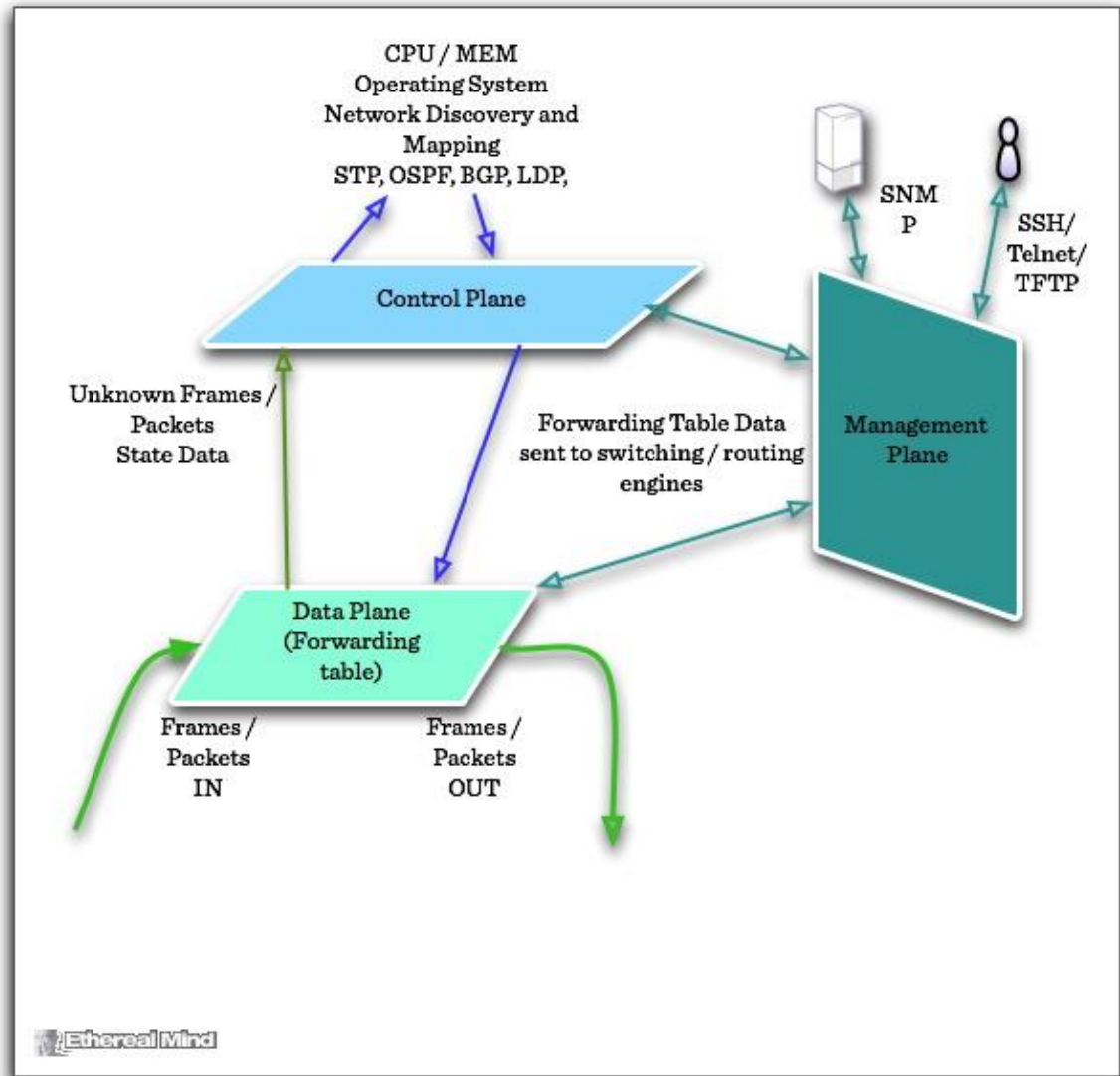


Figure 16. Three networking planes (Ferro 2011)

In the management plane, protocols like SNMP are helpful, but most vendors implement their own CLI and/or EMS for better user experience. For instance, CLI for Cisco IOS is incompatible with Juniper's JUNOS CLI. Consequently, managing multi-vendor networks becomes burdensome due to it. (Sb.)

Another challenge arose when NFV became trendy. Traditional applications require specialized networking hardware like load balancers and firewalls. With NFV, specialized networking functions are also available in virtual form. In other words, networking capabilities can be deployed like software applications regardless of the underlying actual hardware. In order for cloud platforms to

support NFV, networks need to be defined using abstractions which allow entire applications and related resources to be seen as software entities. (Sb.)

5.2 A Brief History of SDN

Software-Defined Networking first emerged from research work performed in 2004 as part of a search for a new network management paradigm. The initial work was built on in 2008 by different groups. The startup Nicira, which was bought by VMWare, created a networking operating system named NOX. At the same time, Nicira worked with teams at Stanford University to create the OpenFlow switch interface. In 2011, the de facto standards body of the SDN space, the Open Networking Foundation, already got a broad support from a lot of big names in the networking industry including Cisco, Juniper Networks, Hewlett-Packard, Dell, Broadcom, IBM, and then later Google, Verizon, Yahoo, Microsoft, Deutsche Telekom, Facebook, and NTT. (Morreale et al. 2015, 28.)

One of the most significant events in the history of the SDN technology occurred in 2012. In the opening keynote speech at the 2012 Open Network Summit, Urs Hölzle, senior vice president of technical infrastructure and Google Fellow at Google, presented how Google was using SDN to over 1,000 networking engineers. Mr. Hölzle informed the audience that Google had already built and used their own switches and SDN controllers in the internal backbone network that interconnected Google's data centers. (Weissberger 2012.) That was how the concept of SDN has captured the attention of network engineers and the trade press.

5.3 What is SDN and How Does It Address Traditional Networking Challenges?

The SDN acronym seems to be everywhere these days. How it is defined may vary depending on the approach. That being said, ONF (2017), the user-driven non-profit organization that focuses on promoting the adoption of SDN through open standards development, defines:

“Software-Defined Networking is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions.”

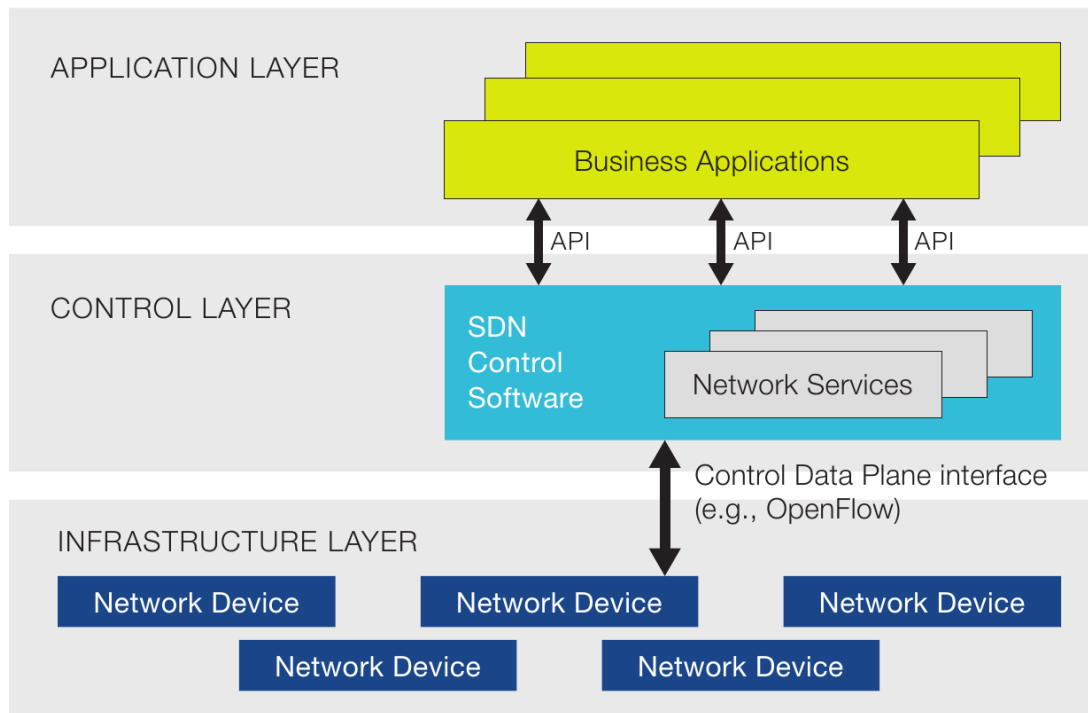


Figure 17. SDN architecture (SDX Central 2017a)

The OpenFlow standard was recognized as the first SDN architecture which defined how the control and data planes could be decoupled and communicate with each other using the OpenFlow protocol (SDX Central 2017a). According to Braun and Menth (2014), the SDN architecture that consists of three layers as shown in Figure 17:

- The infrastructure layer, also called the data plane, is the lowest layer. It is responsible for forwarding data, monitoring local information, and gathering statistics.

- The control layer, or the control plane, is responsible for programming and managing the data plane. It makes use of the information provided by the data plane and gives the data plane instructions. The control layer comprises one or more software controllers that communicate with the data plane through standardized interfaces, which are known as southbound interfaces.
- The application layer contains applications that communicate with the control layer via APIs. The applications collect information from the control layer to build an abstracted and global view of the network for decision-making purposes. These applications could be networking management, analytics, or business used in large data centers. The interface between the application layer and the control layer is referred as the northbound interface.

SDN centralizes the control plane, and it is distinct from how the control plane is distributed and running on each networking device in traditional networks. The centralized control plane is fundamentally a software entity that is usually called the SDN controller. The mentioned interoperability issue is addressed to a large extent thanks to the centralization. The SDN controller programs the device using technologies like OpenFlow, so all devices whose vendors support OpenFlow can easily cooperate. The centralized control plane also addresses scalability challenge. SDN controllers are designed to run on popular hardware platforms or even as virtual machines. Therefore, the scaling of the control plane is independent of the data plane. (Subramanian et al. 2016, 29–30.)

In the traditional network architecture, the management plane is already centralized. Now the SDN architecture whose the control plane is also centralized adds the missing piece to where the traditional network architecture lacks. An SDN controller can directly assist the interaction between the management and control planes. This is very vital to a large and multi-vendor network. And since management software now only needs to deal with a single centralized controller, meaning complexity is hidden from the management plane, more robust programmatic interfaces can be supported. It brings about network abstractions

exposing in the management plane, and cloud platform like OpenStack can make good use of it. (Sb.)

The current landscape of SDN controllers includes a number of commercial products like NSX from VMware, ACI (Application Centric Infrastructure) from Cisco, VSC (Virtualized Service Controller) from Nuage Network, Juniper Contrail from Juniper Networks, and Big Network Controller from Big Switch Networks. There are also a large number of open source controllers including NOX that is the first OpenFlow controllers developed by Nicira, POX that is the Python-only version of NOX, NodeFlow, Floodlight, OpenDaylight, Ryu, OpenContrail, ONOS (Open Network Operating System), MidoNet, NodeFlow, OpenMUL, Beacon, Faucet, etc.

5.4 How Does SDN Fit into the OpenStack Big Picture?

Prior to Neutron, previously known as Quantum, OpenStack had a simple and flat networking environment without L3 or firewall support. The network constructs were baked into Nova which made it difficult to accommodate the changes happening in networking. Neutron was introduced to separate the networking part from other OpenStack service and provide different implementation choices of the abstractions in which Neutron server provides abstraction definition and management, while Neutron plugins do the actual implementation. (Rao 2015.) However, Neutron has been criticized for its complexity and deficiencies on several OpenStack user surveys (Martinelli 2015).

Installing and operationalizing OpenStack is still a constant struggle, especially at scale, and Neutron is a major obstacle to the scalability and resiliency of OpenStack deployments. The reason is that Neutron does not have its own layer 3 routing capability, but it uses a Linux kernel and Linux routing instead. In a large cloud environment with a lot of virtual networks, tenants, and applications, all traffic requiring routing and floating IP services need to be handled by the same Neutron L3 agent. Therefore, the agent becomes the choke point. While it is possible to deploy multiple pairs of L3 agents, it was proven to be very complicated even at moderate scale. Furthermore, there are two networks,

physical and virtual, to manage, which makes correlating issues seen in virtual networks to physical networks more perplexing and time-consuming. (Big Switch 2015.)

SDN solutions can distribute their own L2/L3 agents among OpenStack nodes to help eliminate Neutron L3 agent bottleneck issue. And SDN controllers centralize the management of physical and virtual networks, so it helps simplify managing and monitoring tasks. Additionally, SDN, as discussed above, exposes a myriad of network abstractions thanks to its centralized control plane, which makes it a perfect match for OpenStack. OpenStack supports RESTful APIs for every component. The integration of SDN into OpenStack can result in better networking abstractions and powerful programmatic APIs. The centralized management in SDN architecture also benefits a multi-vendor based cloud infrastructure like OpenStack. (Subramanian et al. 2016, 30–31.)

5.5 OpenFlow

OpenFlow is just an option among several control protocols in SDN, but it the predominant one. OpenFlow is a programmable network protocol designed to manage and direct traffic among routers and switches from multiple vendors. OpenFlow separates the programming of routers and switches from the underlying hardware. (Duffy 2011.)

OpenFlow protocol defines the interface between an OpenFlow controller and an OpenFlow switch like in Figure 18. It allows the OpenFlow controller to instruct the OpenFlow switch on how to handle incoming data packets. (LeClerc 2013.)

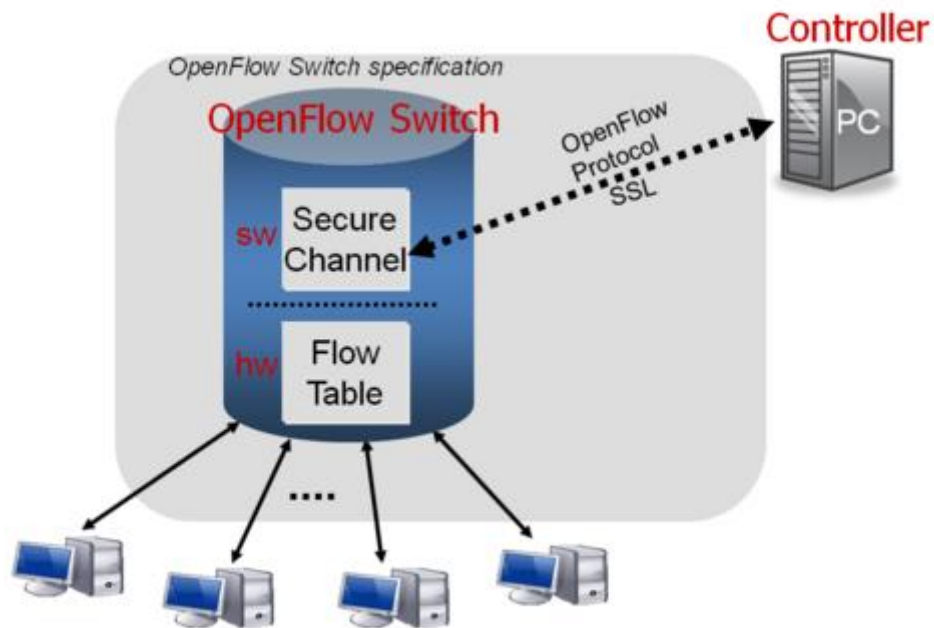


Figure 18. OpenFlow architecture (Stanford 2010)

There are two types of OpenFlow switches: pure (OpenFlow-only) and hybrid (OpenFlow-enabled). Pure switches have no legacy features or on-board control, and they wholly rely on a controller for forwarding decisions. Hybrid switches support OpenFlow in addition to traditional operations and protocols. Commercial switches mostly provide hybrid OpenFlow switches. (Azodolmolky 2013, 10.)

The OpenFlow architecture consists of three parts: the data plane that is built up by OpenFlow switches, the control plane that contains OpenFlow controllers, and a secure control channel that connects these two planes (Braun & Menth 2014).

An OpenFlow switch is a basic forwarding device that forwards packets according to its flow table. The flow table holds a set of flow table entries, each of them consists of rule (match fields), action (instruction), and stats (counters) as shown in figure 19. The match fields define the matching conditions of a flow. The instruction specifies how packets of that flow are handled—forward, drop, and so on. And the counters collect statistics about flows—number of received packets and bytes, as well as the duration of the flow.

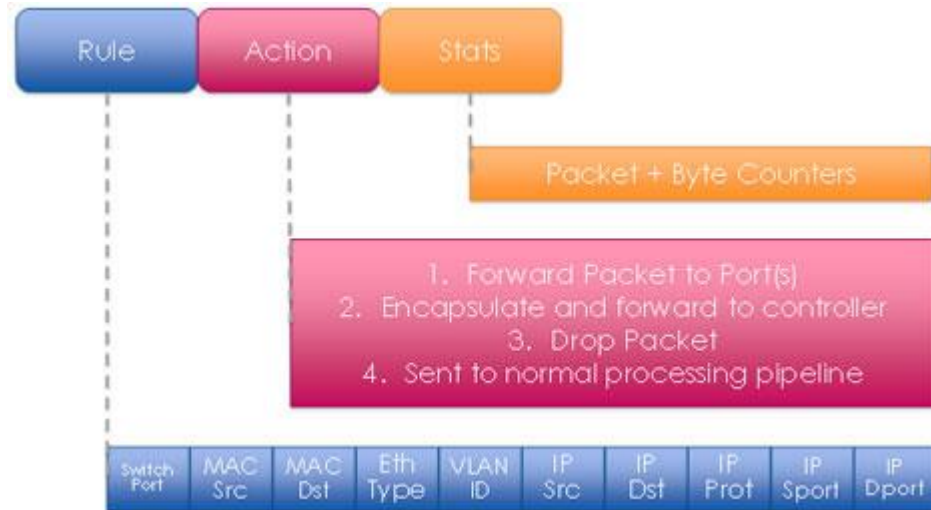


Figure 19. Flow table entry (SDX Central 2017d)

5.6 Open vSwitch

Open vSwitch is one of the three popular virtual switches alongside VMware virtual switch and Cisco Nexus 1000V. Nicira, later bought by VMware, created Open vSwitch to meet the needs of the open source community since there was none feature-rich virtual switch offering designed for a Linux-based hypervisor, such as KVM and XEN. OVS has quickly become the de facto virtual switch for XEN environments, and now it is playing an important part in the OpenStack project as well as a prominent building block in SDN environments. (SDX Central 2017c.)

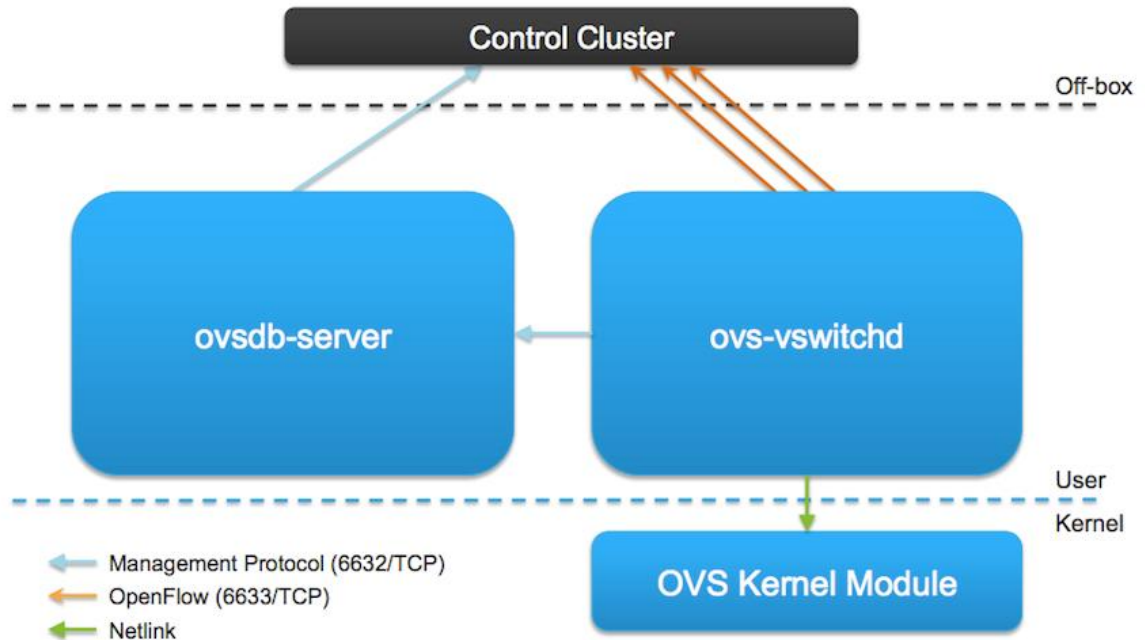


Figure 20. Open vSwitch components (Braun 2014)

Figure 20 illustrates OVS components and how they interact with each other. As stated by Braun (2014), OVS is composed of three main components:

- `ovs-vswitchd` is the Open vSwitch daemon that is responsible for forwarding logic and remote configuration and visibility.
- `ovsdb-server` is the Open vSwitch database server.
- `openvswitch_mod.ko` is the kernel module that is in charge of path lookup, modification, forwarding, and tunnel encapsulation/decapsulation.

The `ovs-vswitchd` daemon communicates with the OpenFlow controller using OpenFlow, with `ovsdb-server` using OVSDDB protocol, with `openvswitch_mod.ko` over netlink, and with the system through the netdev abstract interface.

According to Subramanian et al. (2016, 34–36), OVS provides a rich set of CLI utilities used for configuration, monitoring, and debugging:

- `ovs-vsctl` connects to an `ovsdb-server` process, provides an interface to push the commands, and wait for `ovs-vswitchd` to finish reconfiguring. Its

commands are utilized to create OVS bridges, configure ports and interfaces, and set up the OpenFlow controller.

- `ovs-ofctl` shows the current state of an OpenFlow switch including features, configuration, and OpenFlow table entries for monitoring and administering.
- `ovs-appctl` invokes the commands supported by `ovs-vswitchd` and prints the daemon's response on a standard output.
- `ovs-dpctl` creates, modifies, and deletes OVS data paths implemented outside of `ovs-vswitchd`.

OVS enables massive network automation through programmatic extension and also supports a wide variety of standard management interfaces and protocols, such as NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, and 802.1ag. OVS can operate both as a soft switch running within a hypervisor and as the control stack for switching silicon. (SDX Central 2017c.)

5.7 OpenDaylight and OpenStack

OpenDaylight project which was announced in 2013 is an open source SDN project hosted by the Linux Foundation. The project arose out of the SDN movement, and its purposes are to advance SDN adoption and create the basis for a strong NFV. (SDX Central 2017e.)

The OpenDaylight Controller, which has been renamed the OpenDaylight Platform, supports OpenFlow protocol and other open SDN standards. It exposes open northbound APIs which are used by applications to collect information about the network, run algorithms to conduct analytics, and create new rules throughout the network. The OpenDaylight controller can be deployed on any hardware and operating system platforms that support Java. (SDX Central 2017b.)

Figure 21 below illustrates the architecture of Beryllium—the fourth release of OpenDaylight.



4th Release “Beryllium” Production-Ready Open SDN Platform

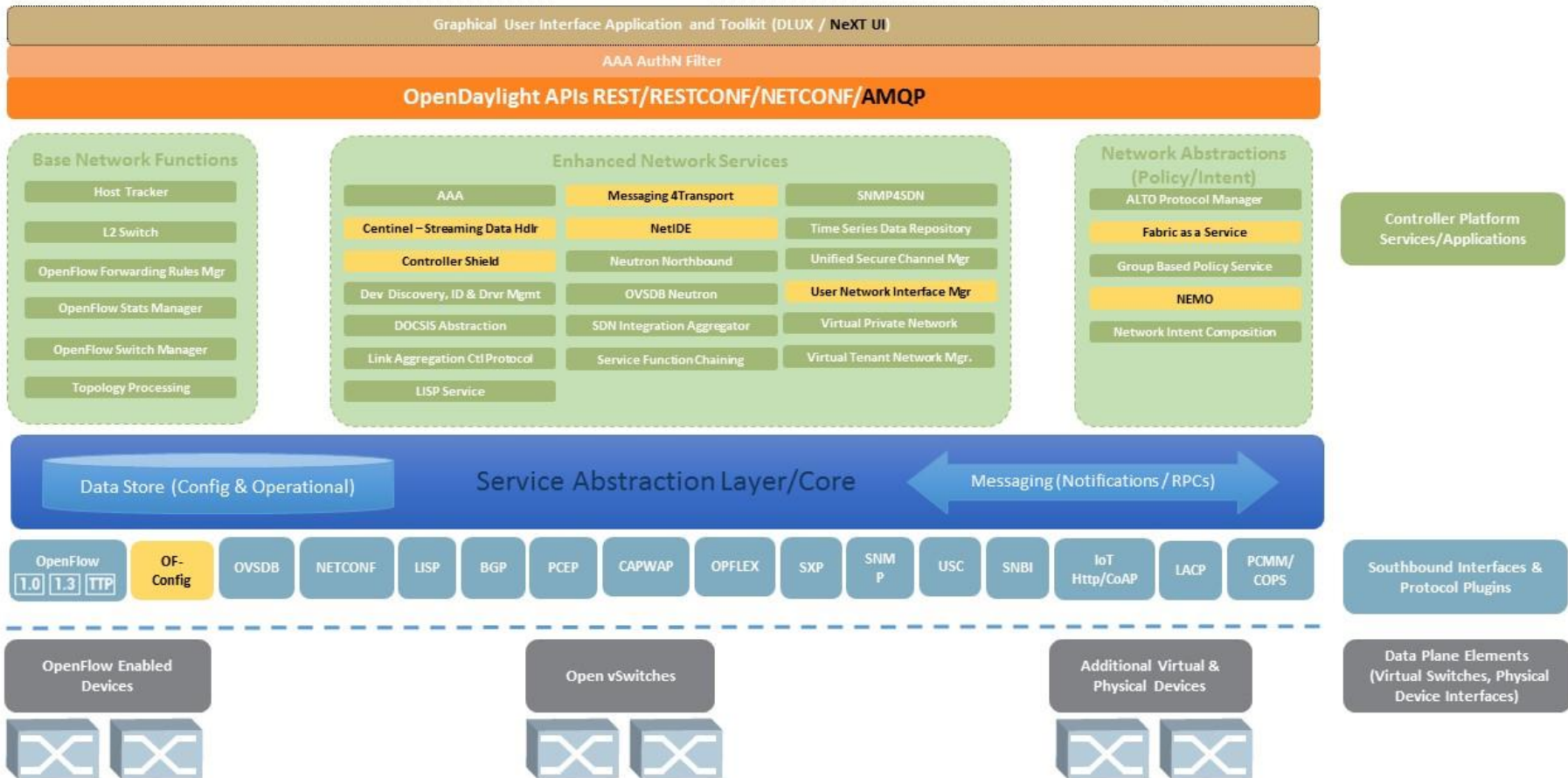


Figure 21. OpenDaylight Beryllium (Ramel 2016)

OpenDaylight (2016a) sums up the fundamental software tools and paradigms that it utilizes, and they include:

- Apache Karaf provides a lightweight runtime to install the Karaf features. OpenDaylight has no pre-installed features by default.
- DLUX (OpenDaylight User Experience) is a web-based GUI that helps to manage networks.
- NeXt (Network embedded Experience) is a developer toolkit that offers visualizations by providing tools to draw network-centric topology UI elements. NeXt can work with DLUX to build OpenDaylight applications.
- MD-SAL (Model-Driven Service Abstraction Layer) is the OpenDaylight frameworks that enables developers to create new Karaf features in form of services and protocol drivers and connect them together.

As seen in Figure 21, there are lots of Karaf features supported in the Beryllium release, such as AAA, BGP, BMP, DLUX, FaaS, LACP, NETCONF, OVSDDB, OF-CONFIG (OpenFlow Configuration Protocol, SNMP, and VTN.

According to Makam (2014), ODL supports a layered architecture:

- The northbound interface provides a rich set of APIs. Those REST APIs are primarily meant for integrating with cloud platforms like OpenStack. They can also be used to build GUI for ODL.
- Controller platform layer is responsible for leveraging the SAL data model and providing fundamental SDN capabilities and networking functions like topology, performance monitoring, physical and virtual switch management, and ARP handling. Controller platform layer links northbound interfaces to southbound interfaces and handles the REST APIs exposed. It also supports use case-specific functionality which turns out to be a useful capability while integrating with OpenStack.
- SAL is the most significant layer in the architecture because its major purpose is to map a diverse set of networking technologies from a

multitude of hardware vendors to a common abstracted data model. All the controller services operate on this abstract data model, thus ODL platform becomes a vendor-neutral controller. SAL and a lot of protocols which are used to communicate with networking devices make up the southbound interface.

Figure 22 below shows a complete diagram of building blocks in the OpenStack and OpenDaylight integration. The idea basically is Neutron's ML2 plugin interacts with ODL's OVSDB Neutron application which in turn commands OVS using OVSDB protocol or OpenFlow protocol. OVS supports both OpenFlow 1.0 and OpenFlow 1.3, and the latter supports multi-table capability that optimizes the number of tunnels needed between Open vSwitches. (Makam 2014.)

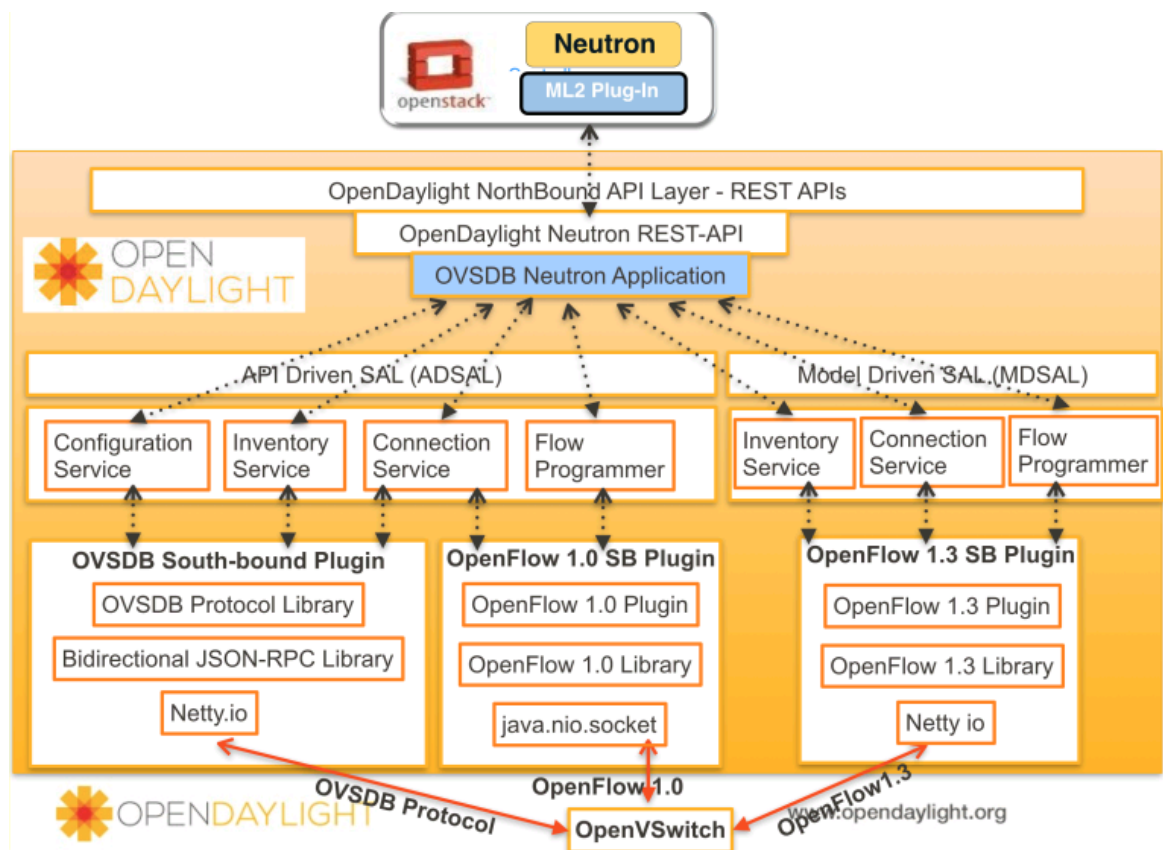


Figure 22. Building blocks involved in OpenStack and OpenDaylight integration (Makam 2014)

6 PRACTICAL IMPLEMENTATION

This chapter demonstrates how a proof-of-concept OpenStack cloud is built with Open vSwitch as the main Neutron ML2 plug-in. And then, OpenDaylight will be integrated into that OpenStack cloud to achieve an SDN-based cloud.

6.1 Topology

For a proof-of-concept environment, I build an OpenStack cloud with four nodes: one controller node, one network node, and two compute nodes. The controller and network nodes are virtual machines running on top of ESXi, while the compute nodes are bare metal servers. All of them use Ubuntu 14.04 as the operating system.

The OpenStack cloud contains most of its core services: Keystone, Glance, Nova, Neutron using the ML2 plug-in with Open vSwitch, and Horizon. Since it is only a prototype, OpenStack is set up to use local disks for instances instead of Swift, Cinder, or Ceph; it also has neither Heat (orchestration) nor Ceilometer (Telemetry).

The controller node runs Keystone and Glance services, management portions of Nova and Neutron, Horizon, as well as supporting services such as an SQL database, message queue, and NTP. The dedicated network node runs Neutron services. The compute nodes run the hypervisor portion of Nova that operates instances and management portions of Neutron, and they use the KVM hypervisor.

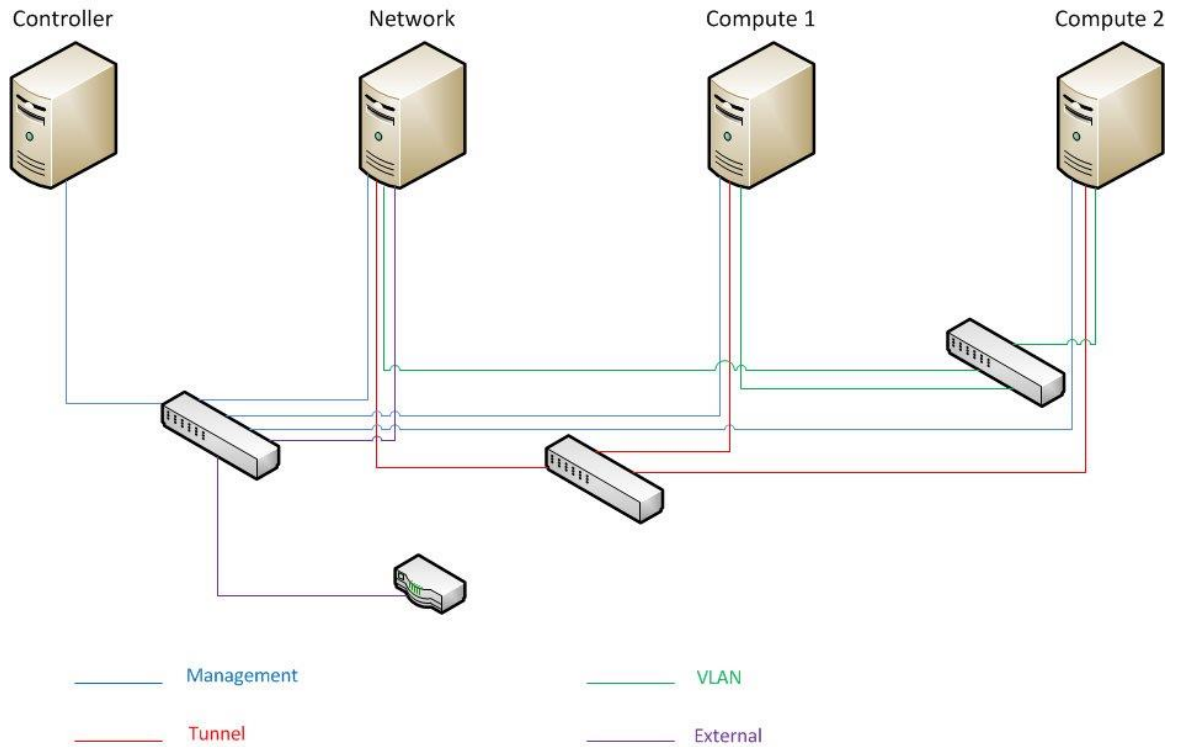


Figure 23. Network topology

There are four simple networks used in the architecture as shown in Figure 23: the management network which is NAT'd for administrative purposes like package installation, security updates, DNS, and NTP, the tunnel network for internal VM traffic type GRE or VXLAN, the VLAN network for internal VM traffic type VLAN, and the external network providing Internet access to instances (the Internet). The management and external networks use 172.16.0.0/21 subnet, the tunnel network uses 10.10.10.0/24 subnet, and the VLAN network does not have any IP address range, because it only handles layer-2 connectivity.

The controller node has only one management interface, the compute nodes have three interfaces for management, tunnel, and VLAN networks, and the network node has four interfaces.

6.2 Deploying OpenStack

This part covers how the OpenStack cloud is built and how to verify OpenStack functionality by launching an instance. Deploying an OpenStack means a lot of

repetition done for each service, so I will only discuss how I installed and configured Keystone and leave out other services for the sake of shortness.

6.2.1 Setting up the Environment

The external interface on each compute node uses a special configuration without any IP address assigned to it, so it needs to be configured to use no IP at all like below:

```
auto eth4
iface eth4 inet manual
up ip link set dev $IFACE up
down ip link set dev $IFACE down
```

The NTP service is also needed on all nodes, so I install and configure chrony so that the `/etc/chrony/chrony.conf` file on each node contains the following line:

```
server <NTP_SERVER/controller> iburst
```

The controller node needs to use a hostname or IP address of an NTP server, whereas other nodes use the controller node as their NTP server.

Then a specific OpenStack repository needs to be enabled. I use the Liberty release, so I add the repository for Liberty and install OpenStack client on all nodes as follows:

```
# apt-get install software-properties-common
# add-apt-repository cloud-archive:liberty
# apt-get install python-openstackclient
```

Most of the OpenStack services use an SQL database to store information. In order for the `/etc/mysql/conf.d/mysqld_openstack.cnf` file to instruct other nodes to access the database via the management network, I install and configure MariaDB on the controller node as well as set the default storage engine and the UTF-8 character set in the following way:

```
[mysqld]
bind-address = <controller-ip>
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

OpenStack also uses a message queue to facilitate inter-process communication among services, so I install RabbitMQ on the controller node. After that, the RabbitMQ broker is told to create a user named `openstack` and set permissions for that user as follows:

```
# rabbitmqctl add_user openstack <rabbitmq_password>
# rabbitmqctl set_permissions openstack \." \." \."
```

At the moment, the environment has host networking, NTP, OpenStack packages and client, SQL database, and message queue set up. And OpenStack core components are ready to be installed.

6.2.2 Installing and Configuring Keystone, Glance, Nova, Neutron, and Horizon

As mentioned above, Keystone is the OpenStack Identity service that provides authentication and authorization services. OpenStack services support multiple security options such as password, policy, and encryption. To ease the installation process in this work, I will only use the password method. Throughout the deployment process, an administrator token will also be needed. The passwords (`GLANCE_DBPASS`, `KEYSTONE_DBPASS`, `NEUTRON_DBPASS`, `NOVA_DBPASS`, etc.) and the token can be manually created or randomly generated by using a tool such as OpenSSL like below:

```
openssl rand -hex 10
```

We need to create a database for Keystone with proper access granted as follows:

```
CREATE DATABASE keystone;
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'
IDENTIFIED BY 'KEYSTONE_DBPASS';
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

Then Keystone package and its dependencies are installed. After that, the Keystone configuration file located at `/etc/keystone/keystone.conf` needs to be edited to contain the admin token generated above, the database access, and some other configurations. The following line includes the names of Keystone package and its dependencies:

```
# apt-get install keystone apache2 libapache2-mod-wsgi
memcached python-memcache
```

The Keystone database is empty at the moment, so we need to populate it with base data with the following command:

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

At this point, a Keystone database with its complete set of tables should be created. By default, Keystone comes with a SQLite database located at `/var/lib/keystone/keystone.db`, but we do not need it, so it can be deleted.

Keystone manages a catalog of OpenStack services, so we need to create service entity and API endpoints for Keystone. Keystone uses port 5000 for public and internal access and port 35357 for admin access.

```
$ openstack service create --name keystone --description
"OpenStack Identity" identity
$ openstack endpoint create --region RegionOne identity
public http://controller:5000/v2.0
$ openstack endpoint create --region RegionOne identity
internal http://controller:5000/v2.0
$ openstack endpoint create --region RegionOne identity
admin http://controller:35357/v2.0
```

Keystone utilizes a combination of domains, projects, users, and roles for authentication as said above, so we need to create them which will be used for administrative operations by the admin user by executing the following commands:

```
$ openstack project create --domain default --description  
"Admin Project" admin  
$ openstack user create --domain default --password-prompt  
admin  
$ openstack role create admin  
$ openstack role add --project admin --user admin admin
```

We also need to create a project called service that consists of a unique user for each OpenStack service like below:

```
$ openstack project create --domain default --description  
"Service Project" service
```

For the following operations, we will need an admin credential file with the content as follows:

```
export OS_PROJECT_DOMAIN_ID=default  
export OS_USER_DOMAIN_ID=default  
export OS_PROJECT_NAME=admin  
export OS_TENANT_NAME=admin  
export OS_USERNAME=admin  
export OS_PASSWORD=<admin_password>  
export OS_AUTH_URL=http://controller:35357/v3  
export OS_IDENTITY_API_VERSION=3
```

For Glance, Nova, and Neutron, the installation and deployment process is almost the same as that of Keystone.

The next step is to create OVS bridges on the network and compute nodes and add the corresponding physical interfaces as ports on the OVS bridges with the following commands:

```
# ovs-vsctl add-br br-tun  
# ovs-vsctl add-port br-tun eth1  
# ovs-vsctl add-br br-vlan  
# ovs-vsctl add-port br-vlan eth2
```

```
# ovs-vsctl add-br br-ex  
# ovs-vsctl add-port br-ex eth3
```

Henceforth, compute and network nodes should have fully functional networking stacks like shown in Figure 24 and Figure 25 below. Besides, each compute node will have an OVS agent running, while the network node will have an OVS agent, an L3 agent, a DHCP agent, and a metadata agent running like described in the OpenStack architecture.

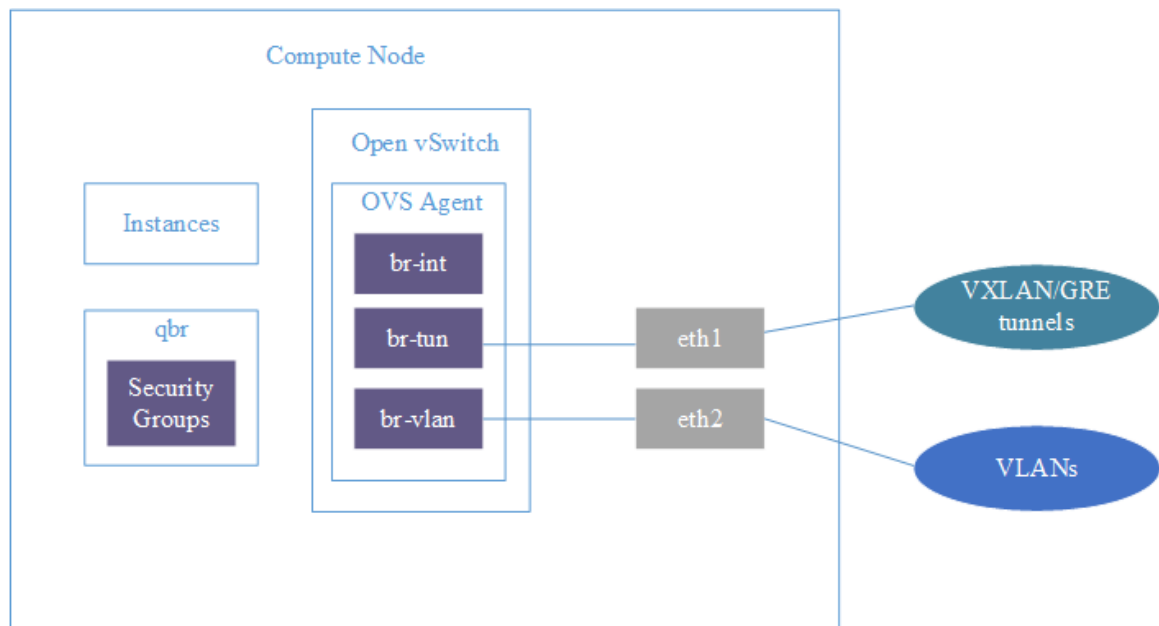


Figure 24. Compute node networking layout

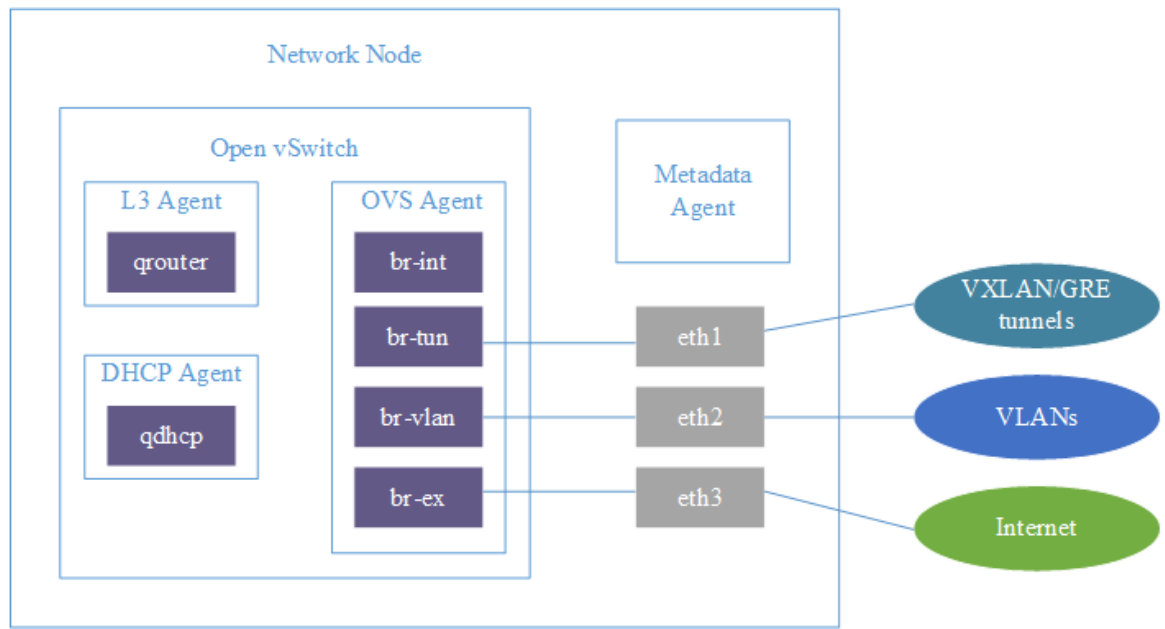


Figure 25. Network node networking layout

We can also obtain a brief overview of the OVS database contents on the compute and network nodes by using `ovs-vsctl show` command as shown in Figure 26 and Figure 27 below.

```

root@network:/home/student# ovs-vsctl show
5a7f6a9e-54d5-45d8-ab72-113e15d6f22d
  Bridge br-tun
    fail_mode: secure
    Port patch-int
      Interface patch-int
        type: patch
        options: {peer=patch-tun}
    Port br-tun
      Interface br-tun
        type: internal
  Bridge br-int
    fail_mode: secure
    Port int-br-vlan
      Interface int-br-vlan
        type: patch
        options: {peer=phy-br-vlan}
    Port br-int
      Interface br-int
        type: internal
    Port int-br-ex
      Interface int-br-ex
        type: patch
        options: {peer=phy-br-ex}
    Port patch-tun
      Interface patch-tun
        type: patch
        options: {peer=patch-int}
  Bridge br-ex
    Port "eth3"
      Interface "eth3"
    Port br-ex
      Interface br-ex
        type: internal
    Port phy-br-ex
      Interface phy-br-ex
        type: patch
        options: {peer=int-br-ex}
  Bridge br-vlan
    Port phy-br-vlan
      Interface phy-br-vlan
        type: patch
        options: {peer=int-br-vlan}
    Port "eth2"
      Interface "eth2"
    Port br-vlan
      Interface br-vlan
        type: internal
  ovs_version: "2.4.1"

```

Figure 26. OVS bridges on network node

```

root@compute1:/home/student# ovs-vsctl show
dd0038ce-e5d1-4312-8aea-fe676e67a840
    Bridge br-tun
        fail_mode: secure
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port br-tun
            Interface br-tun
                type: internal
    Bridge br-vlan
        Port "eth2"
            Interface "eth2"
        Port br-vlan
            Interface br-vlan
                type: internal
        Port phy-br-vlan
            Interface phy-br-vlan
                type: patch
                options: {peer=int-br-vlan}
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port int-br-vlan
            Interface int-br-vlan
                type: patch
                options: {peer=phy-br-vlan}
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
    ovs_version: "2.4.1"

```

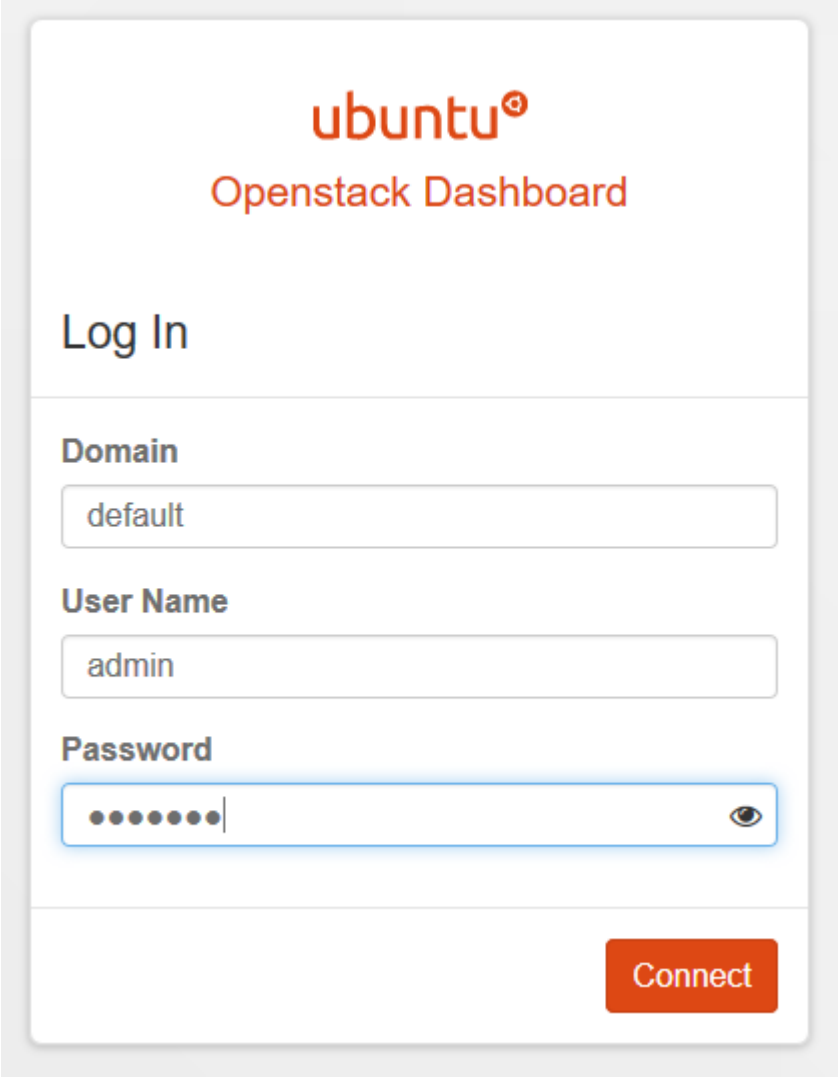
Figure 27. OVS bridges on compute node

For Horizon, we need to install the package that provides the dashboard by using the following command:

```
# apt-get install openstack-dashboard
```

Then the Horizon configuration file located at `/etc/openstack-dashboard/local_settings.py` needs to instruct Horizon to use OpenStack services on the controller node, to allow all hosts to access Horizon, to configure the Keystone API version that we want to use when logging into the dashboard, to choose which Neutron services to enable, and so on. The dashboard is

available to access at `http://<controller-ip>/horizon` afterward. The login screen can be seen in Figure 28 below.



The screenshot shows the OpenStack Horizon login interface. At the top, the Ubuntu logo is displayed in orange, followed by the text "Openstack Dashboard" in a smaller orange font. Below this, the heading "Log In" is centered. The login form consists of three input fields: "Domain" with the value "default", "User Name" with the value "admin", and "Password" with seven dots representing a masked password. A small eye icon is visible to the right of the password field. At the bottom right of the form, there is a red "Connect" button.

Figure 28. OpenStack login screen

6.2.3 Launching an Instance

First, we need to download and upload CirrOS image to Glance like below.

CirrOS is a minimal Linux distribution designed as a test image.

```
$ wget http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img /tmp
```

```
$ glance image-create --name "cirros" --file /tmp/cirros-0.3.4-x86_64-disk.img --disk-format qcow2 --container-format bare --visibility public
```

Then we need to create a flat external network and its subnet by executing the following commands:

```
$ neutron net-create ext-net --router:external True --provider:physical_network external --provider:network_type flat
$ neutron subnet-create ext-net --name ext-subnet --allocation-pool start=172.16.2.50,end=172.16.2.100 --disable-dhcp --gateway 172.16.0.1 172.16.2.0/24
```

A VXLAN network can be created in the same way for a project named `demo` as follows:

```
$ neutron net-create demo-net --tenant-id <demo-project-id> --provider:network_type vxlan
$ neutron subnet-create demo-net --name demo-subnet --gateway 200.200.200.1 200.200.200.0/24
```

After that, a router that connects the `demo` network to the external network is created by using the following commands:

```
$ neutron router-create demo-router
$ neutron router-interface-add demo-router demo-subnet
$ neutron router-gateway-set demo-router ext-net
```

Several `qrouter` and `qdhcp` namespaces will appear on the network node as shown in Figure 29. A network namespace is a logical copy of the networking stack that has its own routers, firewall rules, and network interface devices. A `qdhcp` namespace provides IP addresses to instances via its DHCP service. Every network whose the associated subnets have DHCP enabled has a `qdhcp` namespace. A `qrouter` namespace represents a virtual router and is in charge of routing traffic to and from instances. (Denton 2015, 58.)

```
student@network:~$ ip netns
qrouter-5bd67f38-4612-4a6b-9749-0520fd34134b
qdhcp-66a075f6-f393-4058-91f1-b82664a67a87
```

Figure 29. Neutron network namespaces

There is a security group by default named `default`. We can add additional rules to allow ping and SSH access to the instance like below:

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

A key pair whose public key is injected into a newly created instance is needed so that we can get access to the instances. The public key can be uploaded with the following command:

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key
```

It is now ready to launch an instance like in Figure 30. Available flavors can be listed by using `nova flavor-list` command, and `net-id` is the ID of `demo-net` obtained by using `neutron net-list` command.

```

-----+
student@controller:~$ nova boot --flavor m1.tiny --image cirros --nic net-id=66a
075f6-f393-4058-91f1-b82664a67a87 --security-group default --key-name demo-key v
m1
-----+
-----+
| Property                                     | Value
|-----+-----|
| OS-DCF:diskConfig                           | MANUAL
| OS-EXT-AZ:availability_zone                 |
| OS-EXT-STS:power_state                      | 0
| OS-EXT-STS:task_state                      | scheduling
| OS-EXT-STS:vm_state                         | building
| OS-SRV-USG:launched_at                     | -
| OS-SRV-USG:terminated_at                   | -
| accessIPv4                                  |
| accessIPv6                                  |
| adminPass                                    | u9XBBoifQ74Kv
| config_drive                                |
| created                                      | 2016-07-27T09:39:55Z
| flavor                                       | m1.tiny (1)
| hostId                                       |
| id                                           | 6bf3a5f6-b911-4972-9526-2d1ee6f9b16e
| image                                        | cirros (24be63b4-f45b-4613-9207-75de0f6
9b58f)
| key_name                                    | demo-key
| metadata                                    | {}
| name                                         | vm1
| os-extended-volumes:volumes_attached      | []
| progress                                    | 0
| security_groups                             | default
| status                                       | BUILD
| tenant_id                                   | 536cd333014e4f2bb5b84ccfcfd99d6c
| updated                                      | 2016-07-27T09:39:55Z
| user_id                                     | f8e8a9b93d6e48e18fb4b4e0b949e043
|-----+-----|

```

Figure 30. Launching an OpenStack instance

After associating a floating IP with the running instance, we should be able to access it using SSH. When two instances are created and ping each other, we can see that there are several OVS flows, and the OVS database also gets updated on the network and compute nodes like shown in Figure 31 by using the same `ovs-vsctl show` command. Each OVS database on these nodes will be kept up-to-date with other nodes' information like IP addresses and a lot of virtual network interfaces related to instances (tap devices, veth pairs, Linux bridges, and OVS bridges; their corresponding prefixes are `tap`, `qv`, `qvo`, `qbr`, `qr-`, `qg-`, and `br`). A detailed discussion on what these interfaces are, how they work, and how they connect each other is beyond the scope of this project.


```

Bridge br-tun
  fail_mode: secure
  Port patch-int
    Interface patch-int
      type: patch
      options: {peer=patch-tun}
  Port "vxlan-0a0a141f"
    Interface "vxlan-0a0a141f"
      type: vxlan
      options: {df_default="true", in_key=flow, local_ip="10.10.20.21"
, out_key=flow, remote_ip="10.10.20.31"}
  Port br-tun
    Interface br-tun
      type: internal
Bridge br-int
  fail_mode: secure
  Port "tap1797ce2f-e2"
    tag: 3
    Interface "tap1797ce2f-e2"
      type: internal
  Port int-br-vlan
    Interface int-br-vlan
      type: patch
      options: {peer=phy-br-vlan}
  Port "qg-af31ba5c-68"
    tag: 4
    Interface "qg-af31ba5c-68"
      type: internal
  Port br-int
    Interface br-int
      type: internal
  Port int-br-ex
    Interface int-br-ex
      type: patch
      options: {peer=phy-br-ex}
  Port "qr-ecd9fe4c-78"
    tag: 3
    Interface "qr-ecd9fe4c-78"
      type: internal
  Port patch-tun
    Interface patch-tun
      type: patch
      options: {peer=patch-int}
Bridge br-ex
  Port "eth3"
    Interface "eth3"
  Port br-ex
    Interface br-ex
      type: internal
  Port phy-br-ex
    Interface phy-br-ex
      type: patch
      options: {peer=int-br-ex}
Bridge br-vlan
  Port phy-br-vlan
    Interface phy-br-vlan
      type: patch
      options: {peer=int-br-vlan}
  Port "eth2"
    Interface "eth2"
  Port br-vlan
    Interface br-vlan
      type: internal

```

Figure 31. OVS bridges, ports, and flows on the network node

6.3 Integrating OpenDaylight into OpenStack

ODL node is also a virtual machine like the controller and network nodes, but it has two interfaces: one belongs to the management network so that it can talk to other nodes, the other one belongs to the tunnel network.

After downloading and extracting the latest OpenDaylight release which is Beryllium at the time of writing, we can start OpenDaylight as a server process, then connect to the Karaf shell to install some needed Karaf features and their dependencies by using the following commands:

```
$ ./bin/start
$ ./bin/client
$ feature:install odl-base-all odl-aaa-authn odl-restconf
odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs odl-
ovsdb-openstack odl-ovsdb-northbound odldlux-core
```

Figure 32 illustrates the current architecture of the OpenDaylight and OpenStack integration.

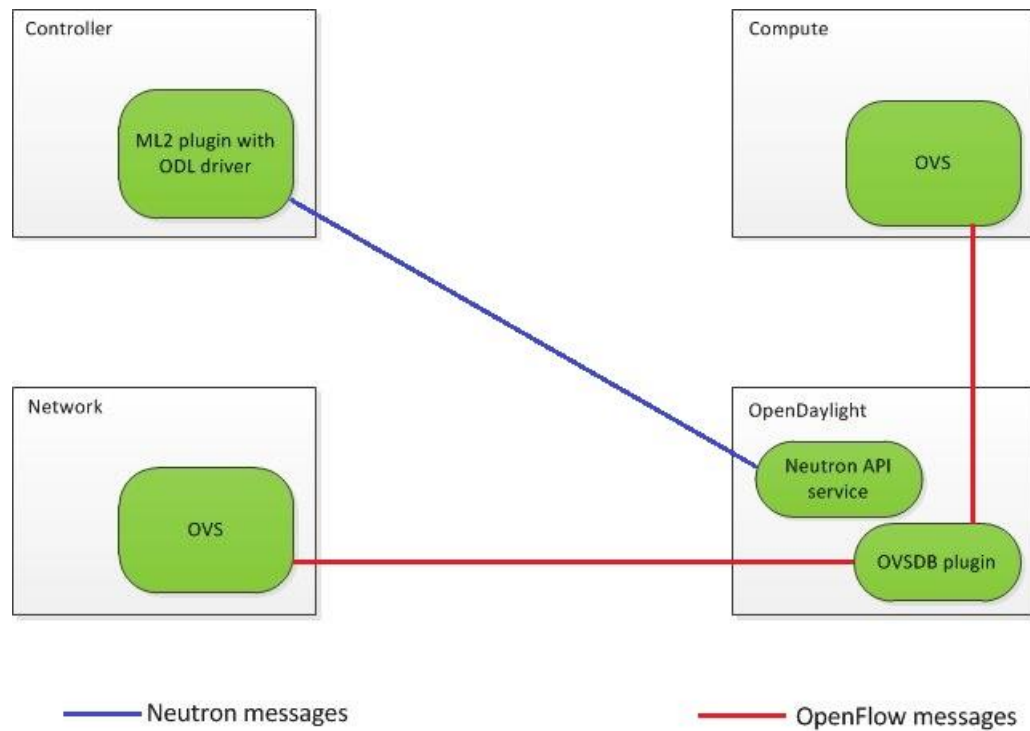


Figure 32. OpenStack and OpenDaylight integration

Since ODL is being utilized as the Neutron backend, it expects to be the only source of truth for OVS configuration, so we need to clean up all the existing OpenStack and OVS configurations to ensure ODL is in a clean state. The neutron-server service on the controller node and the neutron-openvswitch-agent service on all nodes need to be stopped, and the OVS agent also needs disabling so that it will not come back after a reboot as follows:

```
# service neutron-server stop
# service neutron-openvswitch-agent stop
# service neutron-openvswitch-agent disable
```

In order for OpenDaylight to be able to completely manage OVS, the existing OVSDB should be removed, and then we can set up ODL to be the manager of all nodes like below:

```
# service openvswitch-switch stop
# rm -rf /var/log/openvswitch/*
# rm -rf /etc/openvswitch/conf.db
# service openvswitch-switch start
# ovs-vsctl set-manager tcp:<odl_management_ip>:6640
```

All the nodes with running OVS appear in DLUX like in Figure 33.

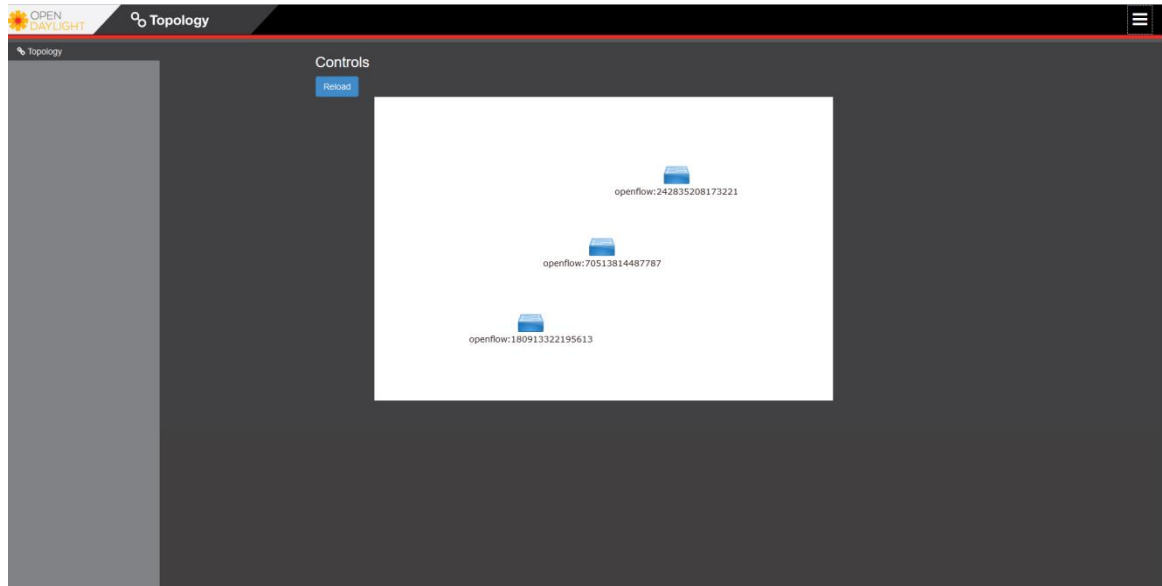


Figure 33. DLUX

We need to configure Neutron to use ODL by adding the `opendaylight` mechanism driver to its existing list. The configuration part for ODL can be seen below:

```
[m12]
mechanism_drivers = opendaylight
[m12_odl]
password = admin
username = admin
url =
http://<odl_management_ip>:8080/controller/nb/v2/neutron
```

The Neutron database needs to be dropped, recreated, and repopulated, too. After recreating networks, subnets, and routers, I create two instances as shown in Figure 34.

```

openstack@controller:~$ nova list
+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State |
+-----+-----+-----+-----+-----+
| 8ca07264-3d85-4387-a323-b39002c73eed | vm1 | ACTIVE | - | Running |
| demo-net=200.200.200.3 |
+-----+-----+-----+-----+-----+
| f02a2753-086d-4301-aa85-47e2cfbeb5eb | vm2 | ACTIVE | - | Running |
| demo-net=200.200.200.4 |
+-----+-----+-----+-----+-----+
openstack@controller:~$ █

```

Figure 34. Running instances

I access one of the instances by using SSH and ping the other, and it works as Figure 35 shows. That means ODL was successfully integrated into the OpenStack cloud.

```

$ ping 200.200.200.3
PING 200.200.200.3 (200.200.200.3): 56 data bytes
64 bytes from 200.200.200.3: seq=0 ttl=64 time=0.846 ms
64 bytes from 200.200.200.3: seq=1 ttl=64 time=0.138 ms
64 bytes from 200.200.200.3: seq=2 ttl=64 time=0.110 ms
64 bytes from 200.200.200.3: seq=3 ttl=64 time=0.109 ms
64 bytes from 200.200.200.3: seq=4 ttl=64 time=0.107 ms
64 bytes from 200.200.200.3: seq=5 ttl=64 time=0.129 ms
64 bytes from 200.200.200.3: seq=6 ttl=64 time=0.114 ms
64 bytes from 200.200.200.3: seq=7 ttl=64 time=0.116 ms

--- 200.200.200.3 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.107/0.208/0.846 ms
$ █

```

Figure 35. Ping check

7 CONCLUSIONS

The goal of the study was to see how the open source software applications that are getting a lot of hype could collaborate, more specifically, to build an SDN-based OpenStack cloud. Overall, that goal was achieved, meaning an SDN controller named OpenDaylight Platform was successfully integrated into a prototypical OpenStack cloud.

Throughout the process, I experienced what people have been saying about OpenStack: deploying an OpenStack cloud is very hard, but scaling the cloud up is much harder without automation tools or support from some vendor. And SDN implementation in this study is still at a basic level because the OpenDaylight platform enables users to write their own applications for controlling networks by exposing RESTful APIs and OSGi interfaces, which is what I did not have enough time to carry out in this study. That being said, working with OpenStack and OpenDaylight showed me the enormous potential of the open source innovation.

OpenStack is undoubtedly an awesome project, but due to its modular architecture, manually deploying OpenStack is time-consuming, error-prone, and tedious. There are dozens of core services that need to come together to get a production-ready OpenStack cloud, and it means a lot of repetition. By making a single mistake, we will have to troubleshoot, investigate log files, and probably end up starting over. However, since OpenStack is an open source project, we can fully and freely patch source code, fix bugs like what we do with a Linux distribution without needing to wait for support from any vendor. Furthermore, OpenStack is growing fast, the document for each release, each networking case is highly detailed and available, and the OpenStack community is extraordinarily large and helpful. And fortunately, there is also a wide range of selection for open source automation tools: Ansible, Chef, Puppet, and SaltStack. All of them are widely used for deploying OpenStack clouds even by commercial vendors: RedHat is using Puppet, HPE is using Ansible, SUSE is using SaltStack, and Rackspace is using Chef. All these great advantages help OpenStack become mature enough for production use at large scale.

Unlike OpenStack, Software-Defined Networking is a bit more like a secret in reality. A great number of research papers, books, and articles on SDN are available for people getting interested in it, but there are hardly any real-life examples on how to implement it, how to use it, or how to make it work, although Mirantis has been adding several SDN plugins to their product for some time. Engineers, specialists, and cloud providers were still wary of this novel technology at first because it is a common case with new technologies: the initial

hype usually exceeds the reality of the situation. Nonetheless, according to a survey done by OpenStack in 2017, SDN/NFV was ranked second among the emerging technologies that interest OpenStack users (Figure 36). That means people are gradually accepting SDN.

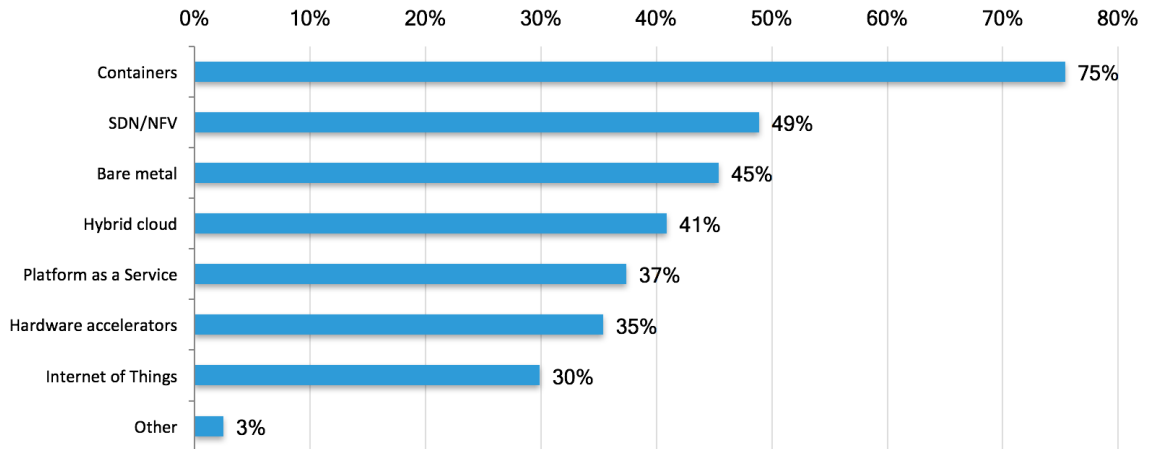


Figure 36. Emerging technologies that interest OpenStack users (OpenStack 2017a)

Moscovici (2015) said, “In the new economy, it’s not the code that matters, it’s how you use it to connect people to things they need. From 3D printers to Docker, open-source-based innovation is fueling some of the hottest digital capabilities of our time. Finally, the golden age of open source has arrived. Companies 20 years ago built monopolies on licensed software; today, free and open-source code fertilizes economic growth. The way to win at tech is no longer to own code, but to serve customers, and service has open source at its roots.” The more people are willing to adopt open source technologies, the more great software like OpenStack will stand out from the rest to give users real benefits.

As previously stated, this is only a proof-of-concept cloud. However, it can be considered as the foundation for further developments to achieve a production-ready OpenStack cloud. There are several ideas that can be taken into account including:

- High availability is one of the foremost factors when building a cloud because it helps reduce system downtime and data loss in the event of a single failure, and it also eliminates single points of failure. High availability

is implemented with redundant hardware. In this project, there is only one virtualized controller node, and there is a standalone network node.

However, in reality, there are usually three bare metal controller nodes in which the networking services are running.

- There are four simple networks used for the prototypical cloud, and the compute nodes use up to four NICs, but it should not occur in a production-ready cloud. A typical real-life OpenStack cloud needs several networks used for management, API, storage, guest, and external traffic. If we want to increase storage and compute capacities, we will not want to spend extra money on extremely expensive servers with that number of network ports, not to mention the high availability of the network. Configuring network bonding and using tagged and untagged VLAN can ease this issue.
- The prototypical cloud also utilizes local disks on the hypervisors for running instances. While it is not a bad option in practice, it is still recommended to use one of the shared storage options including Cinder (block storage), Swift (object storage), Ceph (block, object, and file storage), NFS, ZFS to name a few. One of the benefits of having a shared storage backend is to facilitate the live migration process.
- Adding more OpenStack services is a great idea, too. Heat used for orchestration, Ceilometer used for metering, Logstash used for logging, Monasca, InfluxDB, Icinga, Zabbix, and Nagios used for monitoring and trending, and Kibana used for data virtualization are some external tools that enrich an OpenStack cloud.
- As said above, digging deeper into the SDN real-world use cases, writing some high-level OpenDaylight applications to control the network, and even switching to another open source SDN controller will make a good use of the APIs exposed by OpenStack and OpenDaylight Platform and bring benefits to the cloud.
- Last but not least, there is a crucial fact that nobody wants to build an OpenStack cloud, even one on a small scale, by hand. The deployment process is immensely time-consuming, error-prone, and repetitive. That is the reason why we need an automation tool. Investing time in automating

the deployment process to get a desired cloud takes time, but once it is done, building and rebuilding a cloud will become easier than ever.

REFERENCES

- Azodolmolky, S. 2013. Software-Defined Networking with OpenFlow: Get hands-on with the platforms and development tools used to build OpenFlow network applications. Birmingham: Packt Publishing Ltd.
- Bentley, W. 2015. Evolution of OpenStack: From infancy to enterprise. WWW document. Available at: <https://developer.rackspace.com/blog/evolution-of-openstack-from-infancy-to-enterprise/> [Accessed 15 July 2017].
- Big Switch. 2015. Achieving OpenStack networking nirvana with SDN: Scalability, resiliency, automation. WWW document. Available at: <http://www.bigswitch.com/blog/2015/10/28/achieving-openstack-networking-nirvana-with-sdn-scalability-resiliency-automation> [Accessed 15 July 2017].
- Bloomberg, J. 2012. How NASA helped open-source cloud take off. WWW document. Available at: <http://www.cio.com/article/2394591/developer/how-nasa-helped-open-source-cloud-take-off.html> [Accessed 15 July 2017].
- Braun, S. 2014. Open vSwitch and OpenStack Neutron troubleshooting. WWW document. Available at: <http://www.yet.org/2014/09/openswitch-troubleshooting/> [Accessed 15 July 2017].
- Braun, W. & Menth, M. 2014. Software-Defined Networking using OpenFlow: Protocols, applications and architecture design choices. Ebook. Available at: <http://www.mdpi.com/1999-5903/6/2/302/pdf> [Accessed 15 July 2017].
- Cfheoh. 2011. A cloud economy emerges...somewhat. WWW document. Available at: <http://storagegaga.com/a-cloud-economy-emerges-somewhat/> [Accessed 15 July 2017].
- CloudTweaks. 2012. Cloud deployment models: Read the important differences. WWW document. Available at: <https://cloudtweaks.com/2012/07/4-primary-cloud-deployment-models/> [Accessed 15 July 2017].
- Deepakrghuge. 2015. Cyberduck configuration for spectrum scale for object with Keystone as authentication server. WWW document. Available at: <https://deepakrghuge.wordpress.com/2015/08/03/cyberduck-configuration-for-spectrum-scale-for-object-with-keystone-as-authentication-server/> [Accessed 15 July 2017].
- Denton, J. 2015. Learning OpenStack networking (Neutron): Wield the power of OpenStack Neutron networking to bring network infrastructure and capabilities to your cloud. Second Edition. Birmingham: Packt Publishing Ltd.
- Deskmukh, S. 2016. Importance of cloud computing. WWW document. Available at: <https://www.esds.co.in/blog/importance-of-cloud-computing/> [Accessed 15 July 2017].

Duffy, J. FAQ: What is OpenFlow and why is it needed? WWW document. Available at: <https://www.networkworld.com/article/2202144/data-center/faq--what-is-openflow-and-why-is-it-needed-.html> [Accessed 15 July 2017].

Fifield, T., Fleming, D., Gentle, A., Hochstein, L., Proulx, J., Toews, E. & Topjan, J. 2014. OpenStack operations guide: Set up and manage your OpenStack cloud. California: O'Reilly Media, Inc.

Fu, A. 2017. 7 different types of cloud computing structures that you should know. WWW document. Available at: <https://www.uniprint.net/eng/blog/view/types-cloud-computing-structures-should-know> [Accessed 15 July 2017].

Gupta, R. 2013. Introduction to OpenStack. WWW document. Available at: <https://ilearnstack.com/2013/04/23/introduction-to-openstack-2/> [Accessed 15 July 2017].

Hui, K. 2013. Laying Cinder block (volumes) in OpenStack. Part 1: The basics. WWW document. Available at: <https://cloudarchitectmusings.com/2013/11/18/laying-cinder-block-volumes-in-openstack-part-1-the-basics/> [Accessed 15 July 2017].

Kepes, B. 2011. Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS. WWW document. Available at: <https://support.rackspace.com/whitepapers/understanding-the-cloud-computing-stack-saas-paas-iaas/> [Accessed 15 July 2017].

Khedher, O. 2015. Mastering OpenStack: Design, deploy, and manage a scalable OpenStack infrastructure. Birmingham: Packt Publishing Ltd.

LeClerc, M. 2013. The basics of SDN and the OpenFlow network architecture. WWW document. Available at: <https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/> [Accessed 15 July 2017].

Makam, S. 2014. Network virtualization – OpenStack and ODL integration. WWW document. Available at: <https://sreeninet.wordpress.com/2014/04/19/network-virtualization-and-odlopenstack-integration/> [Accessed 15 July 2017].

Martinelli, N. 2015. Connect the dots on OpenStack Neutron with this updated manual. WWW document. Available at: <http://superuser.openstack.org/articles/connect-the-dots-on-openstack-neutron-with-this-updated-manual/> [Accessed 15 July 2017].

Metz, C. 2012. The secret history of OpenStack, the free cloud software that's changing everything. WWW document. Available at: <https://www.wired.com/2012/04/openstack-3/> [Accessed 15 July 2017].

Morreale, P. & Anderson, J. 2015. Software Defined Networking: Design and deployment. Florida: CRC Press.

Moscovici, M. 2015. The golden age of open source has arrived. WWW document. Available at: <https://techcrunch.com/2015/12/15/the-golden-age-of-open-source-has-arrived/> [Accessed 15 July 2017].

NIST. 2010. NIST Cloud Computing Program – NCCP. WWW document. Available at: <https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp> [Accessed 15 July 2017].

Olow, T. 2017. Diving deep into the world of OpenStack Swift: Part 1. WWW document. Available at: <https://blog.rackspace.com/diving-deep-world-openstack-swift-part-1> [Accessed 15 July 2017].

OpenDaylight. 2016a. OpenDaylight concepts and tools. WWW document. Available at: http://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/concepts_and_tools.html [Accessed 15 July 2017].

OpenDaylight. 2016b. OpenDaylight Karaf features. WWW document. Available at: http://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/karaf_features.html [Accessed 15 July 2017].

Open Networking Foundation. 2017. Software-Defined Networking (SDN) definition. WWW document. Available at: <https://www.opennetworking.org/sdn-definition/> [Accessed 15 July 2017].

OpenStack. 2016a. Conceptual architecture. WWW document. Available at: https://docs.openstack.org/liberty/install-guide-obs/common/get_started_conceptual_architecture.html [Accessed 15 July 2017].

OpenStack 2016b. Overview and components. WWW document. Available at: <https://docs.openstack.org/liberty/networking-guide/intro-os-networking-overview.html> [Accessed 15 July 2017].

OpenStack. 2017a. OpenStack user survey. A snapshot of OpenStack users' perspectives and deployments. Ebook. Available at: <https://www.openstack.org/assets/survey/April2017SurveyReport.pdf> [Accessed 15 July 2017].

OpenStack. 2017b. Horizon Basics. WWW document. Available at: <https://docs.openstack.org/horizon/latest/contributor/intro.html#contributor-intro> [Accessed 15 July 2017].

OpenStack. 2017c. Logical architecture. WWW document. Available at: <https://docs.openstack.org/install-guide/get-started-logical-architecture.html> [Accessed 15 July 2017].

OpenStack. 2017d. Networking architecture. WWW document. Available at: <https://docs.openstack.org/security-guide/networking/architecture.html> [Accessed 15 July 2017].

OpenStack. 2017e. Networking (Neutron) concepts. WWW document. Available at: <https://docs.openstack.org/mitaka/install-guide-rdo/neutron-concepts.html> [Accessed 15 July 2017].

OpenStack. 2017f. Neutron/ML2. WWW document. Available at: <https://wiki.openstack.org/wiki/Neutron/ML2> [Accessed 15 July 2017].

OpenStack. 2017g. Nova system architecture. WWW document. Available at: <https://docs.openstack.org/nova/latest/user/architecture.html> [Accessed 15 July 2017].

OpenStack. 2017h. Release naming. WWW document. Available at: https://wiki.openstack.org/wiki/Release_Naming [Accessed 15 July 2017].

OpenStack. 2017i. Welcome to Glance's documentation! WWW document. Available at: <https://docs.openstack.org/glance/latest/> [Accessed 15 July 2017].

OpenStack Foundation. 2016. OpenStack Foundation 2016 annual report. WWW document. Available at: <https://www.openstack.org/assets/reports/OpenStack-2016-Annual-Report-final-draft.pdf> [Accessed 15 July 2017].

Packt. 2017. OpenStack neutron components and concepts. WWW document. Available at: <https://www.networkcomputing.com/data-centers/openstack-neutron-components-and-concepts/1828660212> [Accessed 15 July 2017].

Paternò, G. 2015. OpenStack – Nova and Glance. WWW document. Available at: <http://www.guruadvisor.net/en/cloud/118-openstack-nova-and-glance> [Accessed 15 July 2017].

Qrimp. 2008. The difference between IaaS and PaaS. WWW document. Available at: <http://www.qrimp.com/blog/blog.The-Difference-between-IaaS-and-PaaS.html> [Accessed 15 July 2017].

Rackspace Support. 2017. Understanding the cloud computing stack: SaaS, PaaS, IaaS. WWW document. Available at: <https://support.rackspace.com/whitepapers/understanding-the-cloud-computing-stack-saas-paas-iaas/> [Accessed 15 July 2017].

Radez, D. 2015. OpenStack essentials: Demystify the cloud by building your own private OpenStack cloud. Birmingham: Packt Publishing Ltd.

Ramel, D. 2016. OpenDaylight cites SDN growth and diversification, releases Beryllium. WWW document. Available at: <https://virtualizationreview.com/articles/2016/02/26/opendaylight-beryllium.aspx> [Accessed 15 July 2017].

Rao, S. 2015. SDN's scale-out effect on OpenStack Neutron. WWW document. Available at: <https://thenewstack.io/sdn-controllers-and-openstack-part1/> [Accessed 15 July 2017].

RedHat. 2014. OpenStack networking (Neutron). Ebook. Available at: http://www.openstackug.org/wp-content/uploads/2014/11/OCF-Red-Hat-OpenStack-UG_Neutron-Introduction.pdf [Accessed 15 July 2017].

RedHat. 2015. RedHat enterprise Linux OpenStack platform 6 component overview: Understanding OpenStack components, their functionality, and their interfaces. Ebook. Available at: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/6/pdf/Component_Overview/Red_Hat_Enterprise_Linux_OpenStack_Platform-6-Component_Overview-en-US.pdf [Accessed 15 July 2017].

RedHat. 2017. Understanding OpenStack. WWW document. Available at: <https://www.redhat.com/en/topics/openstack> [Accessed 15 July 2017].

Rhoton, J., Clercq, J.D. & Novak, F. 2014. OpenStack and cloud computing. Ebook. Available at: <http://docs.huiahoo.com/openstack/mirantis/OpenStack-Cloud-Computing-eBook.pdf> [Accessed 15 July 2017].

Salesforce UK. 2015. Why move to the cloud? 10 benefits of cloud computing. WWW document. Available at: <https://www.salesforce.com/uk/blog/2015/11/why-move-to-the-cloud-10-benefits-of-cloud-computing.html> [Accessed 15 July 2017].

SDX Central. 2017a. Understanding the SDN architecture. WWW document. Available at: <https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/> [Accessed 15 July 2017].

SDX Central. 2017b. What is an OpenDaylight controller? AKA: OpenDaylight platform. WWW document. Available at: <https://www.sdxcentral.com/sdn/definitions/sdn-controllers/.opendaylight-controller/> [Accessed 15 July 2017].

SDX Central. 2017c. What is Open vSwitch (OVS)? WWW document. Available at: <https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch/> [Accessed 15 July 2017].

SDX Central. 2017d. What is OpenFlow? Definition and how it relates to SDN. WWW document. Available at: <https://www.sdxcentral.com/sdn/definitions/what-is-openflow/> [Accessed 15 July 2017].

SDX Central. 2017e. What is the OpenDaylight project (ODL)? WWW document. Available at: <https://www.sdxcentral.com/sdn/definitions/.opendaylight-project/> [Accessed 15 July 2017].

Simplus. 2014. The importance of cloud computing. WWW document. Available at: <https://www.simplus.com/importance-cloud-computing/> [Accessed 15 July 2017].

Somepalle, S. 2015. 3 service and 4 deployment models of cloud computing. WWW document. Available at: <https://www.linkedin.com/pulse/3-service-4-deployment-models-cloud-computing-sankar-somepalle> [Accessed 15 July 2017].

Sriram, S. 2015. OpenStack neutron plugins and agents. WWW document. Available at: <http://www.innervoice.in/blogs/2015/03/31/openstack-neutron-plugins-and-agents/> [Accessed 15 July 2017].

Stretch, J. 2013. What the hell is SDN? WWW document. Available at: <http://packetlife.net/blog/2013/may/2/what-hell-sdn/> [Accessed 15 July 2017].

Subramanian, S. & Voruganti, S. 2016. Software-Defined Networking (SDN) with OpenStack: Leverage the best SDN technologies for your OpenStack-based cloud infrastructure. Birmingham: Packt Publishing Ltd.

Ukov, D. 2012. Object storage approaches for OpenStack cloud: Understanding Swift and Ceph. WWW document. Available at: <https://www.mirantis.com/blog/object-storage-openstack-cloud-swift-ceph/> [Accessed 15 July 2017].

Weissberger, A. 2012. Google's largest internal network interconnects its data centers using Software Defined Network (SDN) in the WAN. WWW document. Available at: <http://techblog.comsoc.org/2012/04/24/googles-largest-internal-network-interconnects-its-data-centers-using-software-defined-network-sdn-in-the-wan/> [Accessed 15 July 2017].