NATIONAL & KAPODISTRIAN UNIVERSITY OF ATHENS

FACULTY OF SCIENCES

DEPARTMENT OF INFORMATICS & TELECOMMUNICATION

INTERDEPARTMENTAL POSTGRADUATE PROGRAMME IN MANAGEMENT & ECONOMICS OF TELECOMMUNICATION NETWORKS

MASTER THESIS

OpenStack

Anargyros.G.KOTZAMANOGLOU

Supervisor: **STATHES HADJIEFTHYMIADES, Associate Professor**

ATHENS

OCTOBER 2015

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΟΙΚΟΝΟΜΙΚΗ ΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

OpenStack

Ανάργυρος Γ.ΚΟΤΖΑΜΑΝΟΓΛΟΥ

Επιβλέπων: **ΕΥΣΤΑΘΙΟΣ ΧΑΤΖΗΕΥΘΥΜΙΑΔΗΣ, Αναπληρωτής Καθηγητής**

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2015

MASTER THESIS

OpenStack

Anargyros.G.KOTZAMANOGLOU
A.M:MOP392

Supervisor: STATHES HADJIEFTHYMIADES, Associate Professor

ATHENS

OCTOBER 2015

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

OpenStack

Ανάργυρος Γ.ΚΟΤΖΑΜΑΝΟΓΛΟΥ
Α.Μ.: ΜΟΠ392

Επιβλέπων: **ΕΥΣΤΑΘΙΟΣ ΧΑΤΖΗΕΥΘΥΜΙΑΔΗΣ, Αναπληρωτής Καθηγητής**

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2015

ABSTRACT

The aim of this thesis is the presentation of OpenStack. An open software management of telecommunications resources in cloud environment. For the preparation of this thesis is first a description of the architecture of cloud environment, and service models used. Architecture Network Function Virtualization occurs then applied to telecommunications in accordance with standards set by the European Telecommunications Standards Institute. The main topic of the thesis is to present the OpenStack software used by the NFV architecture. In these chapters an attempt is as detailed and comprehensive description of the OpenStack functions and parts of the colony consists. Finally there is one techno-economic analysis of the cost of implementing NFV architecture with the current architecture applied to Telecommunication networks. The results derived from this work is a lot of potential implementation and application of the new architecture and the very low operating costs compared with existing technology up to now.

SUBJECT AREA: Architecture Core NetWorks

KEYWORDS: OpenStack, NFV, Cloud computing, Management and Orchestration, Cloud Architecture

ΠΕΡΙΛΗΨΗ

Σκοπός της Διπλωματικής εργασίας είναι η παρουσίαση του OpenStack. Ένα ανοιχτό λογισμικό διαχείρισης των τηλεπικοινωνιακών πόρων σε cloud περιβάλλον. Για την εκπόνηση της Διπλωματικής εργασίας γίνεται αρχικά μία περιγραφή της αρχιτεκτονικής του cloud περιβάλλοντος, και των μοντέλων εξυπηρέτησης που χρησιμοποιούνται. Εν συνεχεία παρουσιάζεται η αρχιτεκτονική Network Function Virtualization που εφαρμόζεται στις τηλεπικοινωνίες σύμφωνα με τα πρότυπα που έχει θέσει ο Ευρωπα'ι'κός Οργανισμός Τηλεπικοινωνιακών Προτύπων. Το κύριο θέμα της Διπλωματικής Εργασίας είναι η παρουσίαση του λογισμικού OpenStack που χρησιμοποιείται από την NFV αρχιτεκτονική. Στα κεφάλαια αυτά γίνεται μία προσπάθεια όσο το δυνατόν λεπτομερέστερης και πληρέστερης περιγραφής των λειτουργιών του OpenStack καθώς και τα μέρη από τα αποία αποτελείται. Τέλος γίνεται μία τεχνοοικονομική ανάλυση του κόστους εφαρμογής της NFV αρχιτεκτονικής με την τωρινή αρχιτεκτονική που εφαρμόζεται στα Τηλεπιοκοινωνιακά δίκτυα. Τα αποτελέσματα τα οποία προκύπτουν από την παρούσα εργασία είναι η πολλές δυνατότητες υλοπόιησης και εφαρμογής της νέας αρχιτεκτονικής καθώς και το πολύ χαμηλό κόστος λειτουργίας της σε σχέση με την υφιστάμενη εώς τώρα τεχνολογία.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αρχιτεκτονική Τηλεπικοινωνιακών Δικτύων Κορμού

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αρχιτεκτονική Σύννεφου, Διαχείριση και Λειτουργία Τηελπικοινωνιακών δικτύων σύννεφου, Εικονικές Δικτυακές Λειτουργίες, Αρχιτεκτονική Τηλεπικοινωνιών

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον Καθηγητή μου Κ. Χατζηευθημιάδη για την βοήθεια της ολοκλήρωσης της εργασίας αυτής μέσα από τις παρατηρήσεις και τα σχολιά του.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ΠΡΟΛΟΓΟΣ

Η παρούσα Διπλωματική Εργασία συντάχθηκε στα πλαίσια του μεταπτυχιακού προγράμματος σπουδών στο Διατμηματικό Τμήμα Οικονομική και Διοίκηση Τηλεπικοινωνιακών Δικτύων, του Εθνικού Καποδιστριακού Πανεπιστημίου Αθηνών, Αθήνα Οκτώβριος 2015.

1 .Cloud Computing

1.1.1 What does the term of Cloud Computing mean

Cloud computing [5] is a computing term or metaphor that evolved in the late 1990s, based on utility and consumption of computer resources. Cloud computing involves application systems which are executed within the cloud and operated through internet enabled devices. Purely cloud computing does not rely on the use of cloud storage as it will be removed upon users download action. Clouds can be classified as public, private and hybrid.

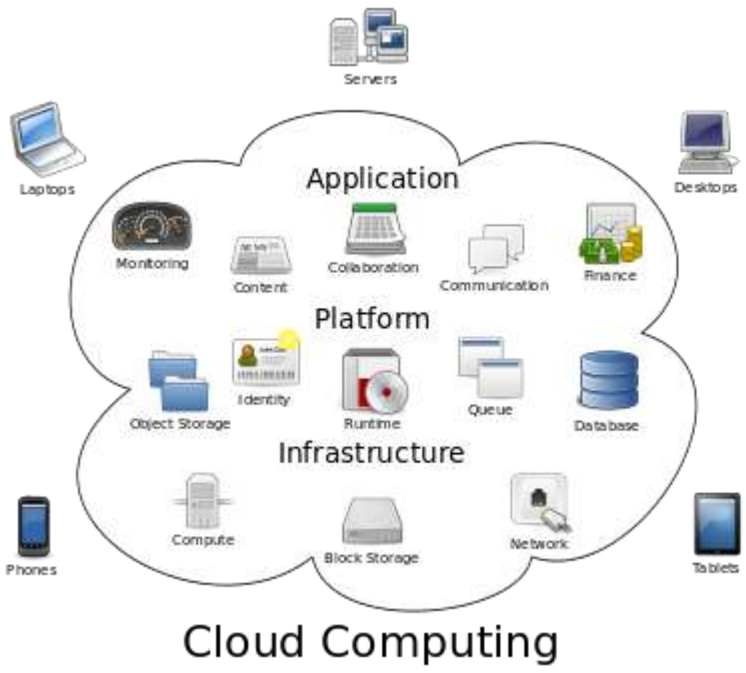


Figure 1: Cloud computing

1.1.2 Characteristics of Cloud

Cloud computing exhibits the following key characteristics: Agility improves with users' ability to re-provision technological infrastructure resources.

Cost reductions claimed by cloud providers. A public-cloud delivery model converts capital expenditure to operational expenditure. This purportedly lowers barriers to entry, as infrastructure is typically provided by a third party and does not need to be purchased for one-time or infrequent intensive computing tasks. Pricing on a utility computing basis is fine-grained, with usage-based options and fewer IT skills are required for implementation (in-house

Device and location independence enable users to access systems using a web browser regardless of their location or what device they use (e.g., PC, mobile phone). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect from anywhere.

Maintenance of cloud computing applications is easier, because they do not need to be installed on each user's computer and can be accessed from different places.

Multitenancy enables sharing of resources and costs across a large pool of users thus allowing for:
- centralization of infrastructure in locations with lower costs (such as real estate, electricity, etc.)
- peak-load capacity increases (users need not engineer for highest possible load-levels)
- utilisation and efficiency improvements for systems that are often only 10-20% utilised.

Performance is monitored, and consistent and loosely coupled architectures are constructed using web services as the system interface.
Productivity may be increased when multiple users can work on the same data simultaneously, rather than waiting for it to be saved and emailed. Time may be saved as information does not need to be re-entered when fields are matched, nor do users need to install application software upgrades to their computer.

Reliability improves with the use of multiple redundant sites, which makes well-designed cloud computing suitable for business continuity and disaster recovery.

Scalability and elasticity via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis in near real-time without users having to engineer for peak loads.

Security can improve due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data, and the lack of security for stored kernels. Security is often as good as or better than other traditional systems, in part because providers are able to devote resources to solving security issues that many customers cannot afford to tackle. However, the complexity of security is greatly increased when data is distributed over a wider area or over a greater number of devices, as well as in multi-tenant systems shared by unrelated users. In addition, user access to security audit logs may be difficult or impossible. Private cloud installations are in part motivated by users' desire to retain control over the infrastructure and avoid losing control of information security.

The National Institute of Standards and Technology's definition of cloud computing identifies "five essential characteristics":

On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear unlimited and can be appropriated in any quantity at any time.

Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

1.2 Service models

Cloud computing providers offer their services according to several fundamental models:

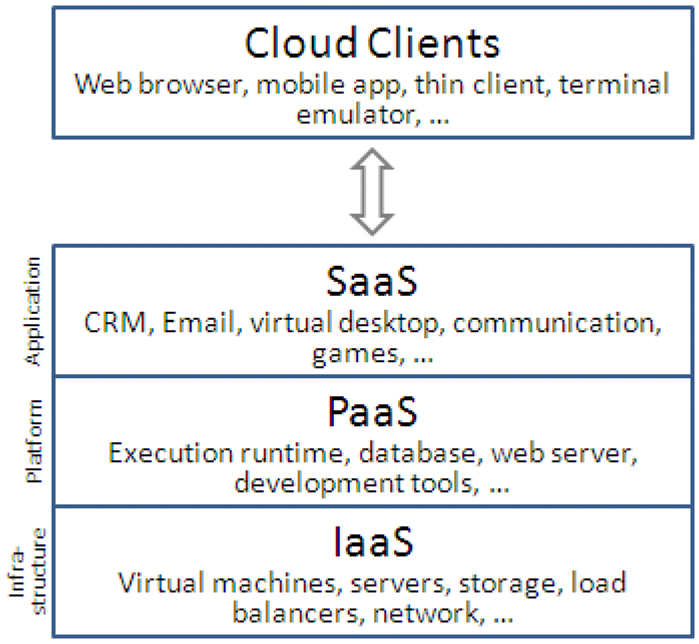

Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
Application
SaaS
CRM, Email, virtual desktop, communication, games, ...
Platform
PaaS
Execution runtime, database, web server, development tools, ...
Infra-structure
IaaS
Virtual machines, servers, storage, load balancers, network, ...

Figure 2: Service Models of Cloud

1.2.1 Infrastructure as a service (IaaS)

In the most basic cloud-service model & according to the IETF (Internet Engineering Task Force), providers of IaaS offer computers – physical or (more often) virtual machines – and other resources. (A hypervisor, such as Xen, Oracle VirtualBox, KVM, VMware ESX/ESXi, or Hyper-V runs the virtual machines as guests. Pools of hypervisors within the cloud operational support-system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements.) IaaS clouds often offer additional resources such as a virtual-machine disk image library, raw block storage, and file or object storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs), and software bundles. IaaS-cloud providers supply these resources on-demand from their large pools installed in data centers. For wide-area connectivity, customers can use either the Internet or carrier clouds (dedicated virtual private networks).

To deploy their applications, cloud users install operating-system images and their application software on the cloud infrastructure. In this model, the cloud user patches and maintains the operating systems and the application software. Cloud providers typically bill IaaS services on a utility computing basis: cost reflects the amount of resources allocated and consumed.

1.2.2 Platform as a service (PaaS)

In the PaaS models, cloud providers deliver a computing platform, typically including operating system, programming language execution environment, database, and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. With some PaaS offers like Microsoft Azure and Google App Engine, the underlying computer and storage resources scale automatically to match application demand so that the cloud user does not have to allocate resources manually. The latter has also been proposed by an architecture aiming to facilitate real-time in cloud environments. Even more specific application types can be provided via PaaS, e.g., such as media encoding as provided by services as bitcoding, transcoding cloud or media.io.

1.2.3 Software as a service (SaaS)

In the business model using software as a service (SaaS), users are provided access to application software and databases. Cloud providers manage the infrastructure and platforms that run the applications. SaaS is sometimes referred to as "on-demand software" and is usually priced on a pay-per-use basis or using a subscription fee.

In the SaaS model, cloud providers install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's own computers, which simplifies maintenance and support. Cloud applications are different from other applications in their scalability—which can be achieved by cloning tasks onto multiple virtual machines at run-time to meet changing work demand. Load balancers distribute the work over the set of virtual machines. This process is transparent to the cloud user, who sees only a single access point. To accommodate a large number of cloud users, cloud applications can be *multitenant*, that is, any machine serves more than one cloud user organization.

The pricing model for SaaS applications is typically a monthly or yearly flat fee per user, so price is scalable and adjustable if users are added or removed at any point.

Proponents claim SaaS allows a business the potential to reduce IT operational costs by outsourcing hardware and software maintenance and support to the cloud provider. This enables the business to reallocate IT operations costs away from hardware/software spending and personnel expenses, towards meeting other goals. In addition, with applications hosted centrally, updates can be released without the need for users to install new software. One drawback of SaaS is that the users' data are stored on the cloud provider's server. As a result, there could be unauthorized access to the data. For this reason, users are increasingly adopting intelligent third-party key management systems to help secure their data.

1.3 Cloud clients

Users access cloud computing using networked client devices, such as desktop computers, laptops, tablets and smartphones. Some of these devices – *cloud clients* – rely on cloud computing for all or a majority of their applications so as to be essentially useless without it. Examples are thin clients and the browser-based Chromebook. Many cloud applications do not require specific software on the client and instead use a web browser to interact with the cloud application. With Ajax and HTML5 these Web user interfaces can achieve a similar, or even better, look and feel to native applications. Some cloud applications, however, support specific client software dedicated to these applications (e.g.,virtual desktop clients and most email clients). Some legacy applications (line of business applications that until now have been prevalent in thin client computing) are delivered via a screen-sharing technology.

1.3.1 Deployment models

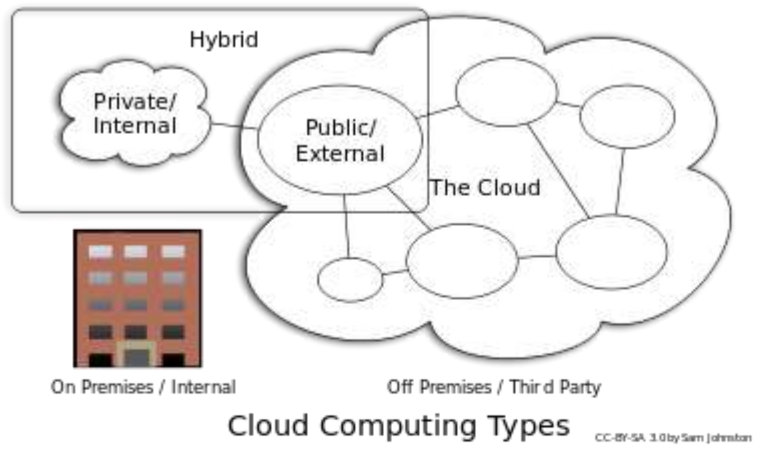


Figure 3: Deployment Models

1.4 Cloud computing types

1.4.1 Private cloud

Private cloud is cloud infrastructure operated solely for a single organization, whether managed internally or by a third-party, and hosted either internally or externally. Undertaking a private cloud project requires a significant level and degree of engagement to virtualize the business environment, and requires the organization to reevaluate decisions about existing resources. When done right, it can improve business, but every step in the project raises security issues that must be addressed to prevent serious vulnerabilities. Self-run data centers are generally capital intensive. They have a significant physical footprint, requiring allocations of space, hardware, and environmental controls. These assets have to be refreshed periodically, resulting in additional capital expenditures. They have attracted criticism because users "still have to buy, build, and manage them" and thus do not benefit from less hands-on management, essentially "lacking the economic model that makes cloud computing such an intriguing concept".

1.4.2 Public cloud

A cloud is called a "public cloud" when the services are rendered over a network that is open for public use. Public cloud services may be free. Technically there may be little or no difference between public and private cloud architecture, however, security consideration may be substantially different for services (applications, storage, and other resources) that are made available by a service provider for a public audience and when communication is effected over a non-trusted network. Saasu is a large public cloud. Generally, public cloud service providers like Amazon AWS, Microsoft and Google own and operate the infrastructure at their data center and access is generally via the Internet. AWS and Microsoft also offer direct connect services called "AWS Direct Connect" and "Azure ExpressRoute" respectively, such connections require customers to purchase or lease a private connection to a peering point offered by the cloud provider.

1.4.3 Hybrid cloud

Hybrid cloud is a composition of two or more clouds (private, community or public) that remain distinct entities but are bound together, offering the benefits of multiple deployment models. Hybrid cloud can also mean the ability to connect collocation, managed and/or dedicated services with cloud resources.

Gartner, Inc. defines a hybrid cloud service as a cloud computing service that is composed of some combination of private, public and community cloud services, from different service providers. A hybrid cloud service crosses isolation and provider boundaries so that it can't be simply put in one category of private, public, or community cloud service. It allows one to extend either the capacity or the capability of a cloud service, by aggregation, integration or customization with another cloud service.

Varied use cases for hybrid cloud composition exist. For example, an organization may store sensitive client data in house on a private cloud application, but interconnect that application to a business intelligence application provided on a public cloud as a software service. This example of hybrid cloud extends the capabilities of the enterprise to deliver a specific business service through the addition of externally available public cloud services. Hybrid cloud adoption depends on a number of factors such as data security and compliance requirements, level of control needed over data, and the applications an organization uses.

Another example of hybrid cloud is one where IT organizations use public cloud computing resources to meet temporary capacity needs that cannot be met by the private cloud. This capability enables hybrid clouds to employ cloud bursting for scaling across clouds. Cloud bursting is an application deployment model in which an application runs in a private cloud or data center and "bursts" to a public cloud when the demand for computing capacity increases. A primary advantage of cloud bursting and a hybrid cloud model is that an organization only pays for extra compute resources when they are needed. Cloud bursting enables data centers to create an in-house IT infrastructure that supports average workloads, and use cloud resources from public or private clouds, during spikes in processing demands.

The specialized model of hybrid cloud, which is built atop heterogeneous hardware, is called "Cross-platform Hybrid Cloud". A cross-platform hybrid cloud is usually powered by different CPU architectures, for example, x86-64 and ARM, underneath. Users can transparently deploy applications without knowledge of the cloud's hardware diversity.This kind of cloud emerges from the raise of ARM-based system-on-chip for server-class computing.

1.4.4 Community cloud

Community cloud shares infrastructure between several organizations from a specific community with common concerns (security, compliance, jurisdiction, etc.), whether managed internally or by a third-party, and either hosted internally or externally. The costs are spread over fewer users than a public cloud (but more than a private cloud), so only some of the cost savings potential of cloud computing are realized.

1.4.5 Distributed cloud

Cloud computing can also be provided by a distributed set of machines that are running at different locations, while still connected to a single network or hub service. Examples of this include distributed computing platforms such as BOINC and Folding@Home. An interesting attempt in such direction is Cloud@Home, aiming at implementing cloud computing provisioning model on top of voluntarily shared resources [76]

Intercloud

The Intercloud is an interconnected global "cloud of clouds and an extension of the Internet "network of networks" on which it is based. The focus is on direct interoperability between public cloud service providers, more so than between providers and consumers (as is the case for hybrid- and multi-cloud).

1.4.6 Multicloud

Multicloud is the use of multiple cloud computing services in a single heterogeneous architecture to reduce reliance on single vendors, increase flexibility through choice, mitigate against disasters, etc. It differs from hybrid cloud in that it refers to multiple cloud services, rather than multiple deployment modes (public, private, legacy).[83][84]

2. Architecture of Cloud

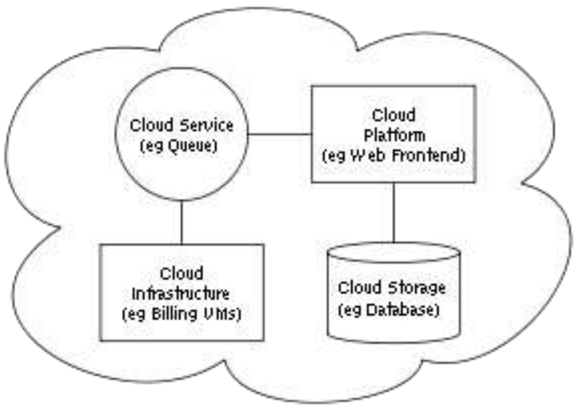


Figure 4: Cloud computing sample architecture

2.1 Cloud Architecture

Cloud architecture [8] , the systems architecture of the software systems involved in the delivery of cloud computing, typically involves multiple *cloud components* communicating with each other over a loose coupling mechanism such as a messaging queue. Elastic provision implies intelligence in the use of tight or loose coupling as applied to mechanisms such as these and others.

2.1.1 Cloud engineering

Cloud engineering is the application of engineering disciplines to cloud computing. It brings a systematic approach to the high-level concerns of commercialization, standardization, and governance in conceiving, developing, operating and maintaining cloud computing systems. It is a multidisciplinary method encompassing contributions from diverse areas such as systems, software, web, performance, information, security, platform, risk, and quality engineering.

2.1.2 Security and privacy

Cloud computing poses privacy concerns because the service provider can access the data that is on the cloud at any time. It could accidentally or deliberately alter or even delete information. Many cloud providers can share information with third parties if necessary for purposes of law and order even without a warrant. That is permitted in their privacy policies which users have to agree to before they start using cloud services. Solutions to privacy include policy and legislation as well as end users' choices for how data is stored. Users can encrypt data that is processed or stored within the cloud to prevent unauthorized access.According to the Cloud Security Alliance, the top three threats in the cloud are "Insecure Interfaces and API's", "Data Loss &

Leakage", and "Hardware Failure" which accounted for 29%, 25% and 10% of all cloud security outages respectively — together these form shared technology vulnerabilities. In a cloud provider platform being shared by different users there may be a possibility that information belonging to different customers resides on same data server. Therefore Information leakage may arise by mistake when information for one customer is given to other.Additionally, Eugene Schultz, chief technology officer at Emagined Security, said that hackers are spending substantial time and effort looking for ways to penetrate the cloud. "There are some real Achilles' heels in the cloud infrastructure that are making big holes for the bad guys to get into". Because data from hundreds or thousands of companies can be stored on large cloud servers, hackers can theoretically gain control of huge stores of information through a single attack — a process he called "hyperjacking".

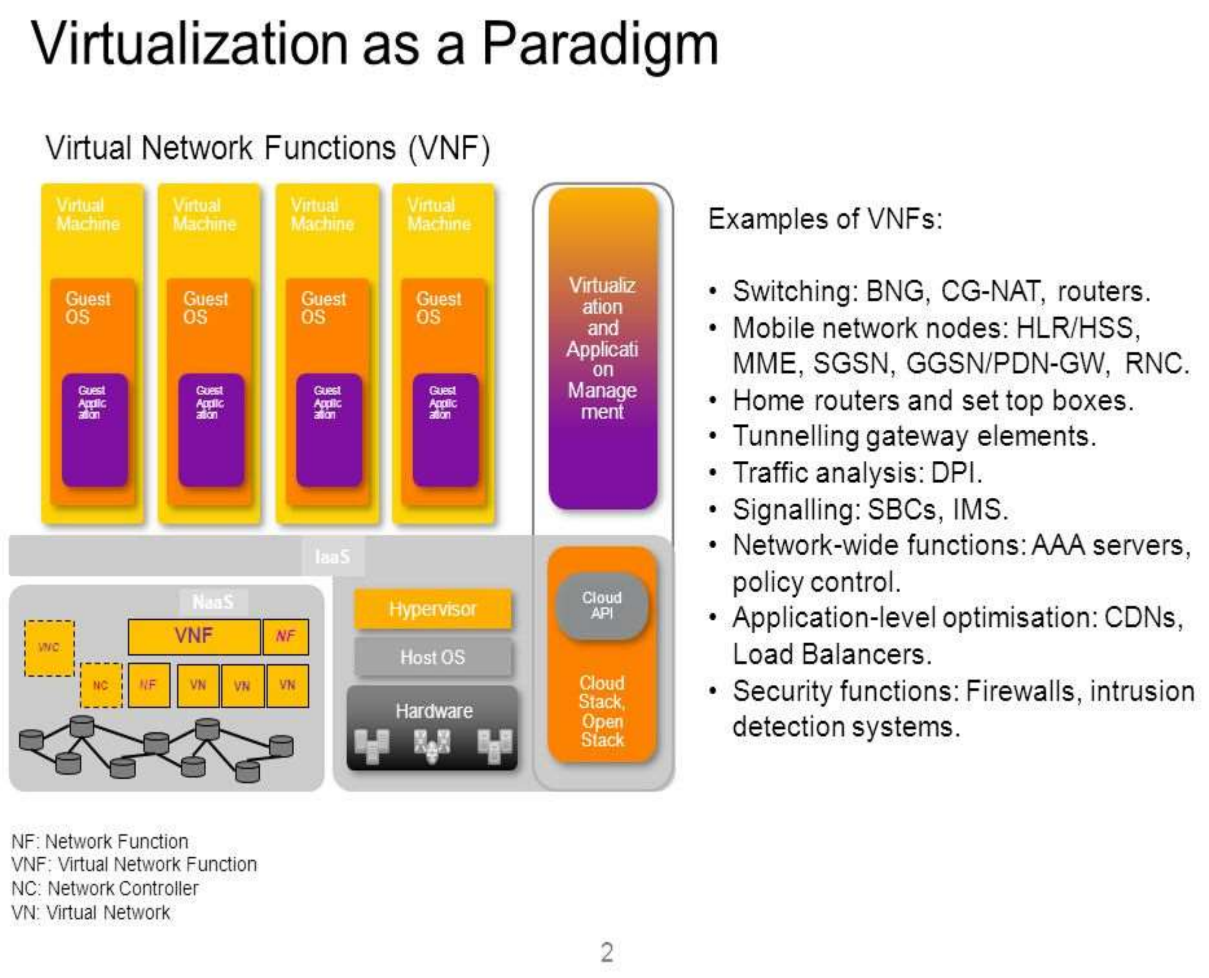
There is the problem of legal ownership of the data (If a user stores some data in the cloud, can the cloud provider profit from it?). Many Terms of Service agreements are silent on the question of ownership.

Physical control of the computer equipment (private cloud) is more secure than having the equipment off site and under someone else's control (public cloud). This delivers great incentive to public cloud computing service providers to prioritize building and maintaining strong management of secure services.Some small businesses that don't have expertise in IT security could find that it's more secure for them to use a public cloud.

There is the risk that end users don't understand the issues involved when signing on to a cloud service (persons sometimes don't read the many pages of the terms of service agreement, and just click "Accept" without reading). This is important now that cloud computing is becoming popular and required for some services to work, for example for an intelligent personal assistant (Apple's Siri or Google Now).

Fundamentally private cloud is seen as more secure with higher levels of control for the owner, however public cloud is seen to be more flexible and requires less time and money investment from the user.

3. NFV Architecture [1]



Virtualization as a Paradigm

Examples of VNFs:

- Switching: BNG, CG-NAT, routers.
- Mobile network nodes: HLR/HSS, MME, SGSN, GGSN/PDN-GW, RNC.
- Home routers and set top boxes.
- Tunnelling gateway elements.
- Traffic analysis: DPI.
- Signalling: SBCs, IMS.
- Network-wide functions: AAA servers, policy control.
- Application-level optimisation: CDNs, Load Balancers.
- Security functions: Firewalls, intrusion detection systems.

NF: Network Function
VNF: Virtual Network Function
NC: Network Controller
VN: Virtual Network

Figure 5: NFV Architecture

3.1 Network Function Virtualization Components [4]

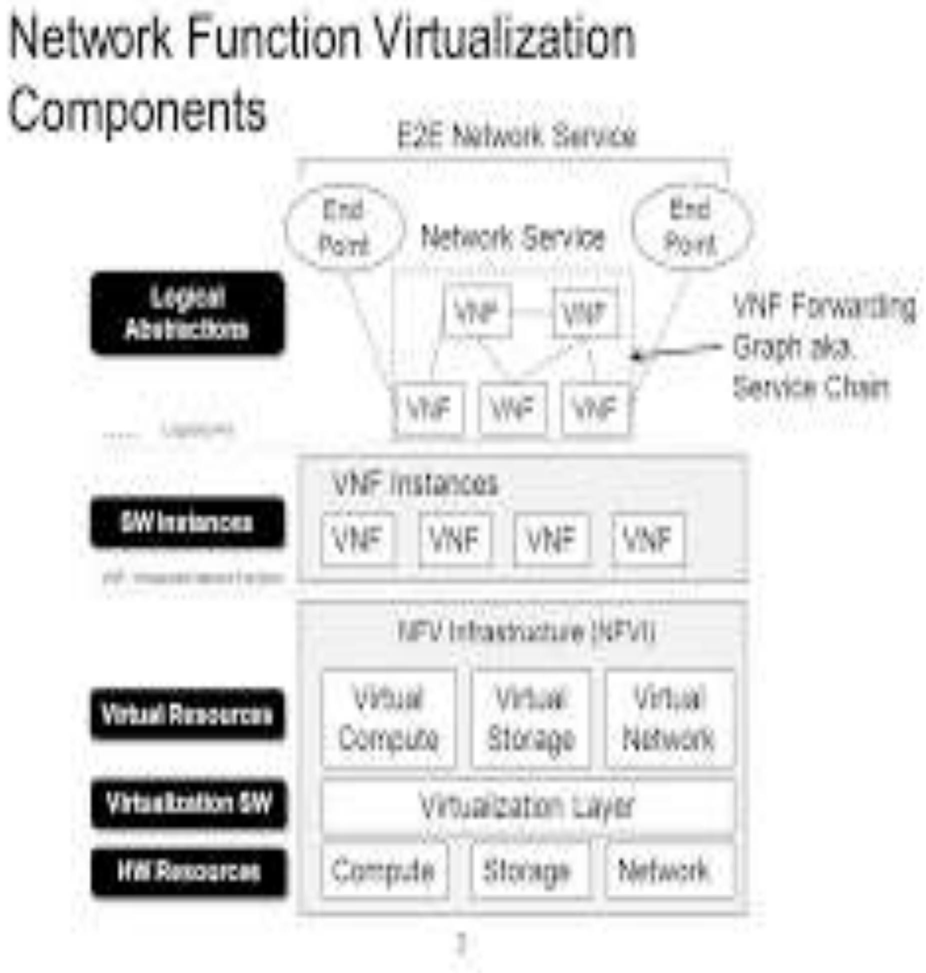


Figure 6: Network Functions Virtualization

3.1.1 MANO Functional Blocks [2]

• NFV Orchestrator: – on-boarding of new Network Service (NS), VNF-FG and VNF Packages – NS lifecycle management (including instantiation, scale-out/in, performance measurements, event correlation, termination) – global resource management, validation and authorization of NFVI resource requests – policy management for NS instances

• VNF Manager: – lifecycle management of VNF instances – overall coordination and adaptation role for configuration and event reporting between NFVI and the E/NMS

• Virtualised Infrastructure Manager (VIM): – controlling and managing the NFVI compute, storage and network resources, within one operator's infrastructure sub-domain – collection and forwarding of performance measurements and events

3.1.2 Hypervisor

In virtualization technology, hypervisor is a software program that manages multiple operating systems (or multiple instances of the same operating system) on a single computer system. The hypervisor manages the system's processor, memory, and other resources to allocate what each operating system requires. Hypervisors are designed for a particular processor architectureand may also be called *virtualization managers*.

A *hypervisor* is an operating system, which means that it knows how to act as a traffic cop to make things happen in an orderly manner. The hypervisor sits at the lowest levels of the hardware environment. Because in cloud computing you need to support many different operating environments, the hypervisor becomes an ideal delivery mechanism.

The hypervisor lets you show the same application on lots of systems without having to physically copy that application onto each system. One twist: Because of the hypervisor architecture, it can load any (or many) different operating system as though it were just another application. Therefore, the hypervisor is a very practical way of getting things virtualized quickly and efficiently.

3.1.3 Scheduling access with the hypervisor

You should understand the nature of the hypervisor. It's designed like a mainframe OS rather than like the Windows operating system. The hypervisor therefore schedules the *amount* of access that guest OSes have to everything from the CPU; to memory; to disk I/O; and to any other I/O mechanisms. With virtualization technology, you can set up the hypervisor to split the physical computer's resources. Resources can be split 50-50 or 80-20 between two guest OSes, for example. Without the hypervisor, you simply can't do that with Windows.

The beauty of this arrangement is that the hypervisor does all the heavy lifting. The guest operating system doesn't care (or have any idea) that it's running in a virtual partition; it thinks that it has a computer all to itself.

3.1.4 Defining types of hypervisors in cloud computing

Different hypervisors support different aspects of the cloud. Hypervisors come in several types:

Native hypervisors**,** which sit directly on the hardware platform are most likely used to gain better performance for individual users.

Embedded hypervisors are integrated into a processor on a separate chip. Using this type of hypervisor is how a service provider gains performance improvements.

Hosted hypervisors run as a distinct software layer above both the hardware and the OS. This type of hypervisor is useful both in private and public clouds to gain performance improvements.

3.1.5 Hypervisor NFV architecture

The Nf-Vi-H [1] is the interface between the hypervisor and the Virtualisation Infrastructure Manager (VIM) . This interface serves multiple purposes below is a description of each purpose:

1) The hypervisor sends monitoring information to the VIM, of the underlying infrastructure. This is currently done through various vendor specific packages. It is a requirement of the VIM to utilize the current vendor specific packages. There may be a gap in this interface with respect to a common standard API requirement in order for the VIM to be able to access various different hypervisor schemes and extend the requirements of the interface.A common standard hypervisor monitoring API has yet to be defined and represents a gap. There are software packages available to implement across different hypervisors. However research is needed and input from across NFV working groups on the gaps with regard to a standard API,what information is transferred (are all the metrics covered) and how the information is transferred (CIM, SNMP, etc.).

2) The VIM is the sole hypervisor controller. All necessary commands, configurations, alerts, policies, responses and updates go through this interface. 5.2.1 Nature of the Interface The nature of this interface is an informational model. There are informational models supporting the data communication between the virtualisation layer and the virtualisation infrastructure manager (VIM) in deployment today. Vmware, Citrix, Redhat, Wind River System., Debian, CentOS all have a VIM. Openstack potentially could be used to be the framework of a VIM that would utilize any VIM thru a standard method. Currently there are software products on the market today that interoperate with the various VIMs in the market place. It is a recommendation that there is a standard, or standard opensource that can be a VIM to interwork with multiple commercial VIMs in deployment today in order to not re-write, re- create current VIMs. Below is a starting point for discussion with regards to virtualisation, BIOS, hypervisors, firmware, networking and hardware. Ultimately there are levels of Metrics, from high level KQIs: (how long does it take to start up the system, how long does it take to delete a system, etc.); all the way down to the hardware or compute domain. Or alternatively from the compute domain, there are hardware capabilities exposed into the software domain via registers, bios, OS, IPMI, drivers, up thru the virtualisation domain, which runs algorithms sent into the VIM for further calculations and from the VIM to the NFV Orchestrator for additional calculations to get to the evaluation of KQIs.Information models are used along the way to gather and communicate these metrics, results and

performance. The next section gives examples of data contained in an information model, followed up by a section that gets into what the hardware provides to software in order for the information model to get the data into feature designed to calculate SLA performance criteria and requirements.

3.2 ETSI RECOMMENDATION FOR HYPERVISOR

ETSI has reported documents for the requirements of Hypervisors. Below we introduce these requirements in general and those which are important for the NFV architecture.

As regards with the definition of Hypervisor is:

• Equivalence: the hypervisor provides an environment for programs which is essentially identical to the original machine.

• Resource control: the hypervisor is in complete control of system resources.

• Efficiency: programs run on this (virtualised) environment show at worst only minor decreases in speed.

3.2.1 Equivalence

The environment provided by a hypervisor is functionally equivalent to the original machine environment. This implies that the same operating systems, tools and application software can be used in the virtual environment. This does not preclude para-virtualisation and other optimization techniques which may require operating systems, tools and application changes.

3.2.2 Resource Control

The hypervisor domain mediates the resources of the computer domain to the virtual machines of the software appliances. Hypervisors as developed for public and enterprise cloud requirements place great value on the abstraction they provide from the actual hardware such that they can achieve very high levels of portability of virtual machines. In essence, the hypervisor can emulate every piece of the hardware platform even in some cases, completely emulating a CPU instruction set such that the VM believes it is running on a completely different CPU architecture from the actual CPU on which it is running. Such emulation, however, has a significant performance cost. The number of actual CPU cycles needed to emulate virtual CPU cycle can be large.

3.2.3 Efficiency

Even when not emulating a complete hardware architecture, there can still be aspects of emulation which cause a significant performance hit. Typically, computer architectures provide means to offload these aspects to hardware, as so called virtualisation extensions, the set of operations that are offloaded and how they are offloaded varies between different hardware architectures and hypervisors as innovation improves virtualisation performance.

example: Intel VT and ARM virtualisation extensions minimise the performance impact of virtualisation by offloading to hardware certain frequently performed operations. There can be many virtual machines running on the same host machine. The VMs on the same host may want to communicate between each other and there will be a need to switch between the VMs.

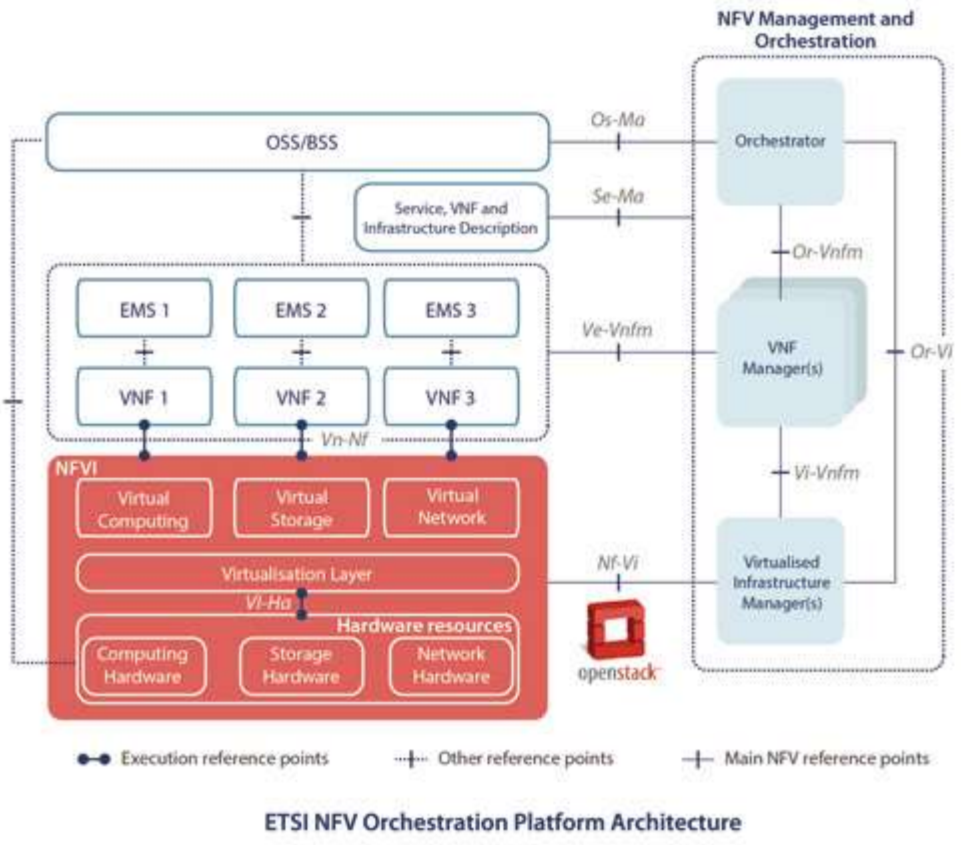


Figure 7:OpenStack in NFV Architecture

The NFV Infrastructure (NFVI) architecture is primarily concerned with describing the Compute, Hypervisor and Infrastructure domains, and their associated interfaces. The present document is primarily focused on describing the hypervisor domain, which comprise the hypervisor which:

• provides sufficient abstract of the hardware to provide portability of software appliances;

• allocates the compute domain resources to the software appliance virtual machines;

• provides a management interface to the orchestration and management system which allows for the loading andmonitoring of virtual machines.

3.3 Hypervisor to VIM (Nf-Vi-H) Interface (OpenStack)

The Nf-Vi-H is the interface between the hypervisor and the Virtualisation Infrastructure Manager (VIM). This interface serves multiple purposes below is a description of each purpose:

1) The hypervisor sends monitoring information to the VIM, of the underlying infrastructure. This is currently done through various vendor specific packages. It is a requirement of the VIM to utilize the current vendor specific packages. There may be a gap in this interface with respect to a common standard API requirement in order for the VIM to be able to access various different hypervisor schemes and extend the requirements of the interface.A common standard hypervisor monitoring API has yet to be defined and represents a gap. There are software packages available to implement across different hypervisors. However research is needed and input from across NFV working groups on the gaps with regard to a standard API,what information is transferred (are all the metrics covered) and how the information is transferred (CIM, SNMP, etc.).

2) The VIM is the sole hypervisor controller. All necessary commands, configurations, alerts, policies, responses and updates go through this interface. 5.2.1 Nature of the Interface The nature of this interface is an informational model. There are informational models supporting the data communication between the virtualisation layer and the virtualisation infrastructure manager (VIM) in deployment today. Vmware, Citrix, Redhat, Wind River System., Debian, CentOS all have a VIM. Openstack potentially could be used to be the framework of a VIM that would utilize any VIM thru a standard method. Currently there are software products on the market today that interoperate with the various VIMs in the market place: As an example, one such product is Hotlink: http://www.virtualizationpractice.com/hotlink-supervisor-vcenter-forhyper-v-kvm-and-xenserver-15369/. It is a recommendation that there is a standard, or standard opensource that can be a VIM to interwork with multiple commercial VIMs in deployment today in order to not re-write, re- create current VIMs. Below is a starting point for discussion with regards to virtualisation, BIOS, hypervisors, firmware, networking and hardware. Ultimately there are levels of Metrics, from high level KQIs: (how long does it take to start up the system, how long does it take to delete a system, etc.); all the way down to the hardware or compute domain. Or alternatively from the compute domain, there are hardware capabilities exposed into the software domain via registers, bios, OS, IPMI, drivers, up thru the virtualisation domain, which runs algorithms sent into the VIM for further calculations and from the VIM to the NFV Orchestrator for additional calculations to get to the evaluation of KQIs.Information models are used along the way to gather and communicate these metrics, results and performance. The next section gives examples of data contained in an information model, followed up by a section that gets into what the hardware provides to software in order for the information model to get the data into feature designed to calculate SLA performance criteria and requirements. Neither of these two following sections are anywhere close to be exhaustive. Thus, the need for an NFV WI to research what is required above and beyond what is available today. 5.2.1.1 Example of MIB information

There are over 1 000 parameters/variables/metrics that are used in Virtualisation/ Cloud Service Assurance MIBs. Hypervisors and cloud programs use IDs, Bios, IPMI, PCI, I/O Adapters/Drivers, memory subsystems, etc. to get to all of the items of current concerns, including information for failover, live migration, placement of the VMs/applications. It is not clear that there is a gap at this point. The hypervisor team has indicated the exposure of the hardware capabilities thru what is in the documentation today (id info, impi, drivers, bios, pci, memory subsystems, etc.) has not exposed any gaps. The hypervisor team is looking forward to working with the Metrics WI, SWA, SEC and MANO for any gaps or requirements beyond what is available today. The NFVINF hypervisor domain VIM is expected to leverage available managers such as CloudStack, vCenter, Openstack, others as packages. There are software packages available today that implement this scheme, e.g. HotLink SuperVisor: editor's note to scrub for trademarks. Below is an example of some of the components currently in a MIB, informational model, that show some of the functions. MIBs are generally broken up into 'functional' areas. Below are some examples. MIB Functions/objects:

• Resources (CPU, Memory, Storage, Adapters, Resource pools, Clusters): - Ex: Adapters: PCI ID, I/O, memory, bus, model, status, capabilities.

• Systems (Logs, NUMA, I/O, etc.).

• Events.

• VM management.

• Obsolete/Legacy (compatibility with older versions).

• Products (supported products (hw and sw)).

• Analytics: - Checks the incoming metrics for abnormalities in real time, updates health scores, and generates alerts when necessary. - Collects metrics and computes derived metrics. - Stores the collected metrics statistics. (filesystem). - Stores all other data collected, including objects, relationships, events,dynamic thresholds and alerts.

• Multicore Processors: - Hyperthreading. - CPU Affinity's. - Power management. Table 2 contains some details with regard to the VIM information that is gathered. These tables are not complete. Research is required to determine the performance characteristics and the gaps of what is provided today and what is needed.

OpenStack Compute supports many hypervisors, which might make it difficult for you to choose one. Most installations use only one hypervisor. However, you can use the section called “ComputeFilter” and the section called “ImagePropertiesFilter” to schedule different hypervisors within the same installation. The following links help you choose a hypervisor.

The following hypervisors are supported:

KVM - Kernel-based Virtual Machine. The virtual disk formats that it supports is inherited from QEMU since it uses a modified QEMU program to launch the virtual machine. The supported formats include raw images, the qcow2, and VMware formats.

LXC - Linux Containers (through libvirt), use to run Linux-based virtual machines.

QEMU - Quick EMUlator, generally only used for development purposes.

UML - User Mode Linux, generally only used for development purposes.

VMware vSphere 4.1 update 1 and newer, runs VMware-based Linux and Windows images through a connection with a vCenter server or directly with an ESXi host.

Xen - XenServer, Xen Cloud Platform (XCP), use to run Linux or Windows virtual machines. You must install the nova-compute service in a para-virtualized VM.

Hyper-V - Server virtualization with Microsoft's Hyper-V, use to run Windows, Linux, and FreeBSD virtual machines. Runs nova-computenatively on the Windows virtualization platform.

Bare Metal - Not a hypervisor in the traditional sense, this driver provisions physical hardware through pluggable sub-drivers (for example, PXE for image deployment, and IPMI for power management).

4. Architecture of OpenStack

4.1 Overview

The OpenStack project [3] is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack provides an Infrastructure-as-a-Service (IaaS) solution through a variety of complemental services. Each service offers an application programming interface (API) that facilitates this integration. The following table provides a list of OpenStack services:

Table 1:OpenStack services

Table 1.1. OpenStack services		
Service	**Project name**	**Description**
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.
Networking	Neutron	Enables Network-Connectivity-as-a-Service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture

Table 1.1. OpenStack services

Service	Project name	Description
		facilitates the creation and management of block storage devices.
Shared services		
Identity service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.
Higher-level services		
Orchestration	Heat	Orchestrates multiple composite cloud applications by using either the native HOTtemplate format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Database Service	Trove	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

This guide describes how to deploy these services in a functional test environment and, by example, teaches you how to build a production environment. Realistically, you would use automation tools such as Ansible, Chef, and Puppet to deploy and manage a production environment.

4.1.2 Conceptual architecture

Launching a virtual machine or instance involves many interactions among several services. The following diagram provides the conceptual architecture of a typical OpenStack environment.

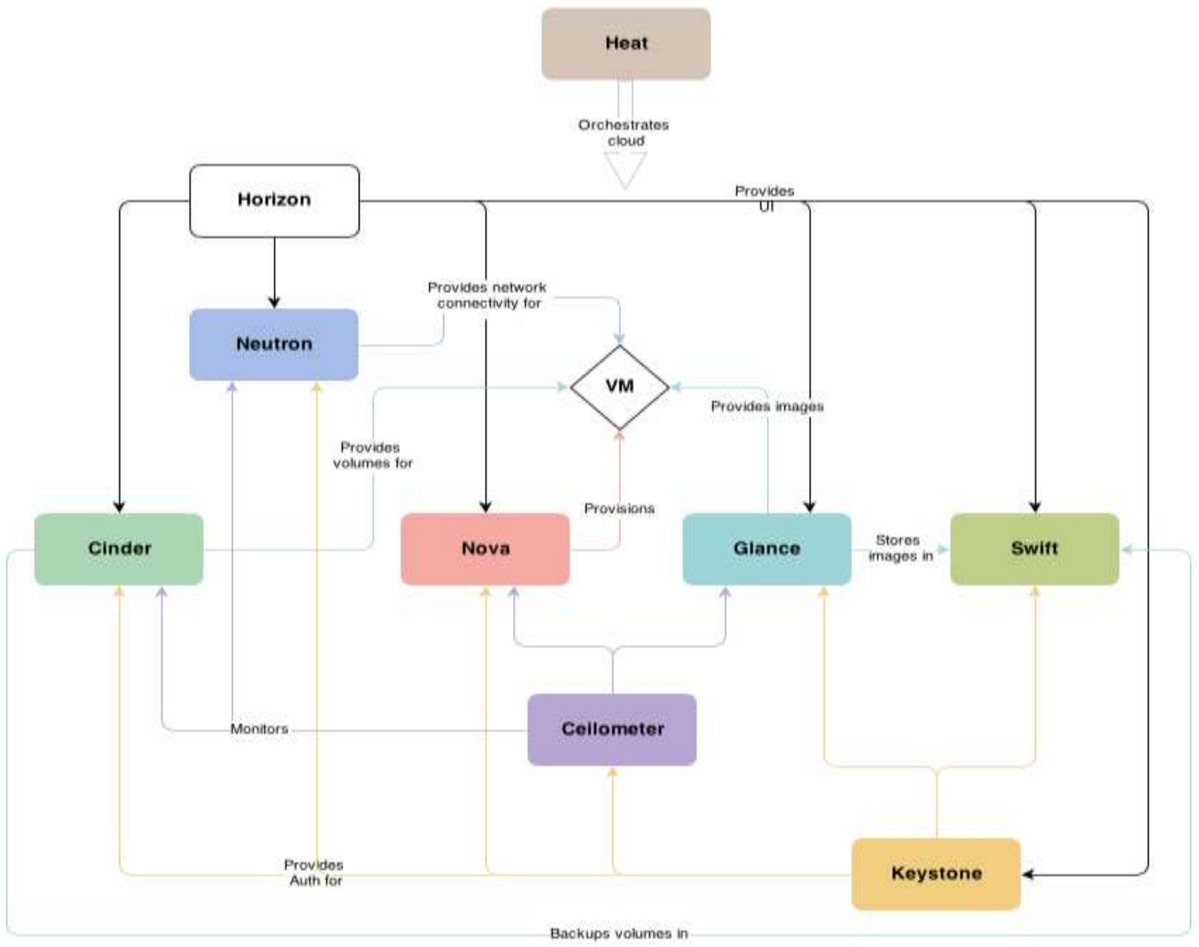


Figure 8: OpenStack Arcitecture

4.2 Induction to OpenStack components

The present chapter represents with more analytics the components of OpenStack. The role of each component, the interaction with each other and the functions they implement.

4.2.1 Horizon

Horizon is the canonical implementation of OpenStack's Dashboard, which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc.

Horizon holds several key values at the core of its design and architecture:

Core Support: Out-of-the-box support for all core OpenStack projects.

Extensible: Anyone can add a new component as a “first-class citizen”.

Manageable: The core codebase should be simple and easy-to-navigate.

Consistent: Visual and interaction paradigms are maintained throughout.

Stable: A reliable API with an emphasis on backwards-compatibility.

Usable: Providing an *awesome* interface that people *want* to use.

Core Support

Horizon ships with three central dashboards, a “User Dashboard”, a “System Dashboard”, and a “Settings” dashboard. Between these three they cover the core OpenStack applications and deliver on Core Support.

The Horizon application also ships with a set of API abstractions for the core OpenStack projects in order to provide a consistent, stable set of reusable methods for developers. Using these abstractions, developers working on Horizon don’t need to be intimately familiar with the APIs of each OpenStack project.

Extensible

A Horizon dashboard application is based around the **Dashboard** class that provides a consistent API and set of capabilities for both core OpenStack dashboard apps shipped with Horizon and equally for third-party apps. The **Dashboard** class is treated as a top-level navigation item.

Should a developer wish to provide functionality within an existing dashboard (e.g. adding a monitoring panel to the user dashboard) the simple registration pattern makes it possible to write an app which hooks into other dashboards just as easily as creating a new dashboard. All you have to do is import the dashboard you wish to modify.

Manageable

Within the application, there is a simple method for registering a **Panel** (sub-navigation items). Each panel contains the necessary logic (views, forms, tests, etc.) for that interface. This granular breakdown prevents files (such as api.py) from becoming thousands of lines long and makes code easy to find by correlating it directly to the navigation.

Consistent

By providing the necessary core classes to build from, as well as a solid set of reusable templates and additional tools (base form classes, base widget classes, template tags, and perhaps even class-based views) we can maintain consistency across applications.

Stable

By architecting around these core classes and reusable components we create an implicit contract that changes to these components will be made in the most backwards-compatible ways whenever possible.

Usable

Ultimately that’s up to each and every developer that touches the code, but if we get all the other goals out of the way then we are free to focus on the best possible experience.

4.2.2 Neutron

Management Network

A management network (a separate network for use by your cloud operators) typically consists of a separate switch and separate NICs (network interface cards), and is a recommended option. This segregation prevents system administration and the monitoring of system access from being disrupted by traffic generated by guests. Consider creating other private networks for communication between internal components of OpenStack, such as the message queue and OpenStack Compute. Using a virtual local area network (VLAN) works well for these scenarios because it provides a method for creating multiple virtual networks on a physical network.

Public Addressing Options

There are two main types of IP addresses for guest virtual machines: fixed IPs and floating IPs. Fixed IPs are assigned to instances on boot, whereas floating IP addresses can change their association between instances by action of the user. Both types of IP addresses can be either public or private, depending on your use case. Fixed IP addresses are required, whereas it is possible to run OpenStack without floating IPs. One of the most common use cases for floating IPs is to provide public IP addresses to a private cloud, where there are a limited number of IP addresses available. Another is for a public cloud user to have a "static" IP address that can be reassigned when an instance is upgraded or moved. Fixed IP addresses can be private for private clouds, or public for public clouds. When an instance terminates, its fixed IP is lost. It is worth noting that newer users of cloud computing may find their ephemeral nature frustrating

IP Address Planning

An OpenStack installation can potentially have many subnets (ranges of IP addresses) and different types of services in each. An IP address plan can assist with a shared understanding of network partition purposes and scalability. Control services can have public and private IP addresses, and as noted above, there are a couple of options for an instance's public addresses. An IP address plan might be broken down into the following sections:

Subnet router

Packets leaving the subnet go via this address, which could be a dedicated router or a nova-network service.

Control services public interfaces

Public access to swift-proxy, nova-api, glance-api, and horizon come to these addresses, which could be on one side of a load balancer or pointing at individual machines.

Object Storage cluster internal communications

Traffic among object/account/container servers and between these and the proxy server's internal interface uses this private network.

Compute and storage communications

If ephemeral or block storage is external to the compute node, this network is used.

Out-of-band remote management

If a dedicated remote access controller chip is included in servers, often these are on a separate network. In-band remote management Often, an extra (such as 1 GB) interface on compute or storage nodes is used for system administrators or monitoring tools to access the host instead of going through the public interface.

Spare space for future growth

Adding more public-facing control services or guest instance IPs should always be part of your plan.

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

172.22.42.0/24:

172.22.42.1 - 172.22.42.3 - subnet routers

172.22.42.4 - 172.22.42.20 - spare for networks

172.22.42.21 - 172.22.42.104 - Compute node remote access controllers (inc spare) 172.22.42.105 - 172.22.42.188 - Compute node management interfaces (inc spare) 172.22.42.189 - 172.22.42.208 - Swift proxy remote access controllers (inc spare) 172.22.42.209 - 172.22.42.228 - Swift proxy management interfaces (inc spare) 172.22.42.229 - 172.22.42.252 - Swift storage servers remote access controllers (inc spare) 172.22.42.253 - 172.22.42.254 – spare

172.22.87.0/26:

172.22.87.1 - 172.22.87.3 - subnet routers

172.22.87.4 - 172.22.87.24 - Swift proxy server internal interfaces (inc spare)

172.22.87.25 - 172.22.87.63 - Swift object server internal interfaces (inc spare)

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

OpenStack Compute with nova-network provides predefined network deployment models, each with its own strengths and weaknesses. The selection of a network manager changes your network topology, so the choice should be made carefully. You also have a choice between the triedand-true legacy nova-network settings or the neutron project for OpenStack Networking. Both offer networking for launched instances with different implementations and requirements. For OpenStack Networking with the neutron project, typical configurations are documented with the idea that any setup you can configure with real hardware you can re-create with a software-defined equivalent. Each tenant can contain typical network elements such as routers, and services such as DHCP.

VLAN Configuration Within OpenStack VMs VLAN configuration can be as simple or as complicated as desired. The use of VLANs has the benefit of allowing each project its own subnet and broadcast segregation from other projects. To allow OpenStack to efficiently use VLANs, you must allocate a VLAN range (one for each project) and turn each compute node switch port into a trunk port. For example, if you estimate that your cloud must support a maximum of 100 projects, pick a free VLAN range that your network infrastructure is currently not using (such as VLAN 200–299). You must configure OpenStack with this range and also configure your switch ports to allow VLAN traffic from that range.

Multi-NIC Provisioning OpenStack Networking with neutron and OpenStack Compute with nova-network have the ability to assign multiple NICs to instances. For nova-network this can be done on a per-request basis, with each additional NIC using up an entire subnet or VLAN, reducing the total number of supported projects. Multi-Host and Single-Host Networking The nova-network service has the ability to operate in a multi-host or single-host mode. Multi-host is when each compute node runs a copy of nova-network and the instances on that compute node use the compute node as a gateway to the Internet. The compute nodes also host the floating IPs and security groups for instances on that node. Single-host is when a central server—for example, the cloud controller—runs the nova-network service. All compute nodes forward traffic from the instances to the cloud controller. The cloud controller then forwards traffic to the Internet. The cloud controller hosts the floating IPs and security groups for all instances on all compute nodes in the cloud. There are benefits to both modes. Single-node has the downside of a single point of failure. If the cloud controller is not available, instances cannot communicate on the network. This is not true with multi-host, but multi-host requires that each compute node has a public IP address to communicate on the Internet. If you are not able to obtain a significant block of public IP addresses, multi-host might not be an option. OpenStack, like any network application, has a number of standard considerations to apply, such as NTP and DNS.

NTP

Time synchronization is a critical element to ensure continued operation of OpenStack components. Correct time is necessary to avoid errors in instance scheduling, replication of objects in the object store, and even matching log timestamps for debugging. All servers running OpenStack components should be able to access an appropriate NTP server. You may decide to set up one locally or use the public pools available from the Network Time Protocol project.

DNS

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

Instance Storage Solutions

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

• Off compute node storage—shared file system

• On compute node storage—shared file system

• On compute node storage—nonshared file system In general, the questions you should ask when selecting storage are as follows:

• What is the platter count you can achieve? •

Do more spindles result in better I/O despite network access?

• Which one results in the best cost-performance scenario you're aiming for?

• How do you manage the storage operationally? Many operators use separate compute and storage hosts. Compute services and storage services have different requirements, and compute hosts typically require more CPU and RAM than storage hosts. Therefore, for a fixed budget, it makes sense to have different configurations for your compute nodes and your storage nodes. Compute nodes will be invested in CPU and RAM, and storage nodes will be invested in block storage.

However, if you are more restricted in the number of physical hosts you have available for creating your cloud and you want to be able to dedicate as many of your hosts as possible to running instances, it makes sense to run compute and storage on the same machines. We'll discuss the three main approaches to instance storage in the next few sections.

4.2.3 Off Compute Node Storage—Shared File System

In this option, the disks storing the running instances are hosted in servers outside of the compute nodes. If you use separate compute and storage hosts, you can treat your compute hosts as "stateless." As long as you don't have any instances currently running on a compute host, you can take it offline or wipe it completely without having any effect on the rest of your cloud. This simplifies maintenance for the compute hosts. There are several advantages to this approach:

• If a compute node fails, instances are usually easily recoverable.

• Running a dedicated storage system can be operationally simpler.

• You can scale to any number of spindles.

• It may be possible to share the external storage for other purposes. The main downsides to this approach are:

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

On Compute Node Storage—Shared File System

In this option, each compute node is specified with a significant amount of disk space, but a distributed file system ties the disks from each compute node into a single mount. The main advantage of this option is that it scales to external storage when you require additional storage. However, this option has several downsides:

• Running a distributed file system can make you lose your data locality compared with nonshared storage.

• Recovery of instances is complicated by depending on multiple hosts.

• The chassis size of the compute node can limit the number of spindles able to be used in a compute node.

• Use of the network can decrease performance.

On Compute Node Storage—Non shared File System

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

• Heavy I/O usage on one compute node does not affect instances on other compute nodes. • Direct I/O access can increase performance. This has several downsides:

• If a compute node fails, the instances running on that node are lost.

• The chassis size of the compute node can limit the number of spindles able to be used in a compute node.

• Migrations of instances from one node to another are more complicated and rely on features that may not continue to be developed.

• If additional storage is required, this option does not scale. Running a shared file system on a storage system apart from the computes nodes is ideal for clouds where reliability and scalability are the most important factors. Running a shared file system on the compute nodes themselves may be best in a scenario where you have to deploy to preexisting servers for which you have little to no control over their specifications. Running a non shared file system on the compute nodes themselves is a good option for clouds with high I/O requirements and low concern for reliability. Issues with Live Migration We consider live migration an integral part of the operations of the cloud. This feature provides the ability to seamlessly move instances from one physical host to another, a necessity for performing upgrades that require reboots of the compute hosts, but only works well with shared storage. Live migration can also be done with non shared storage, using a feature known as KVM live block migration. While an earlier implementation of block-based migration in KVM and QEMU was considered unreliable, there is a newer, more reliable implementation of block-based live migration as of QEMU 1.4 and libvirt 1.0.2 that is also compatible with OpenStack. However, none of the authors of this guide have first-hand experience using live block migration.

Choice of File System If you want to support shared-storage live migration, you need to configure a distributed file system. Possible options include:

• NFS (default for Linux)

• GlusterFS

• MooseFS

• Lustre

We've seen deployments with all, and recommend that you choose the one you are most familiar with operating. If you are not familiar with any of these, choose NFS, as it is the easiest to set up and there is extensive community knowledge about it.

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

• CPU allocation ratio: 16:1

• RAM allocation ratio: 1.5:1

The default CPU allocation ratio of 16:1 means that the scheduler allocates up to 16 virtual cores per physical core. For example, if a physical node has 12 cores, the scheduler sees 192 available virtual cores. With typical flavor definitions of 4 virtual cores per instance, this ratio would provide 48 instances on a physical node. The formula for the number of virtual instances on a compute node is (OR*PC)/VC, where:

OR CPU overcommit ratio (virtual cores per physical core)

PC Number of physical cores

VC Number of virtual cores per instance

Similarly, the default RAM allocation ratio of 1.5:1 means that the scheduler allocates instances to a physical node as long as the total amount of RAM associated with the instances is less than 1.5 times the amount of RAM available on the physical node. For example, if a physical node has 48 GB of RAM, the scheduler allocates instances to that node until the sum of the RAM associated with the instances reaches 72 GB (such as nine instances, in the case where each instance has 8 GB of RAM).

Scaling

Determining the scalability of your cloud and how to improve it is an exercise with many variables to balance. No one solution meets everyone's scalability goals. However, it is helpful to track a number of metrics. Since you can define virtual hardware templates, called "flavors" in OpenStack, you can start to make scaling decisions based on the flavors you'll provide. These templates define sizes for memory in RAM, root disk size, amount of ephemeral data disk space available, and number of cores for starters.

The starting point for most is the core count of your cloud. By applying some ratios, you can gather information about:

• The number of virtual machines (VMs) you expect to run, ((overcommit fraction × cores) / virtual cores per instance)

• How much storage is required (flavor disk size × number of instances) You can use these ratios to determine how much additional infrastructure you need to support your cloud. Here is an example using the ratios for gathering scalability information for the number of VMs expected as well as the storage needed. The following numbers support (200 / 2) × 16 = 1600 VM instances and require 80 TB of storage for /var/lib/nova/instances:

• 200 physical cores.

• Most instances are size m1.medium (two virtual cores, 50 GB of storage)

. • Default CPU overcommit ratio (cpu_allocation_ratio in nova.conf) of 16:1. However, you need more than the core count alone to estimate the load that the API services, database servers, and queue servers are likely to encounter. You must also consider the usage patterns of your cloud. As a specific example, compare a cloud that supports a managed webhosting platform with one running integration tests for a development project that creates one VM per code commit. In the former, the heavy work of creating a VM happens only every few months, whereas the latter puts constant heavy load on the cloud controller. You must consider your average VM lifetime, as a larger number generally means less load on the cloud controller.

Aside from the creation and termination of VMs, you must consider the impact of users accessing the service—particularly on nova-api and its associated database. Listing instances garners a great deal of information and, given the frequency with which users run this operation, a cloud with a large number of users can increase the load significantly. This can occur even without their knowledge—leaving the OpenStack dashboard instances tab open in the browser refreshes the list of VMs every 30 seconds. After you consider these factors, you can determine how many cloud controller cores you require. A typical eight core, 8 GB of RAM server is sufficient for up to a rack of compute nodes — given the above caveats. You must also consider key hardware specifications for the performance of user VMs, as well as budget and performance needs, including storage performance (spindles/core), memory availability (RAM/core), network bandwidth (Gbps/core), and overall CPU performance (CPU/core).

4.2.4 Cloud Controller Nodes

You can facilitate the horizontal expansion of your cloud by adding nodes. Adding compute nodes is straightforward—they are easily picked up by the existing installation. However, you must consider some important points when you design your cluster to be highly available. Recall that a cloud controller node runs several different services. You can install services that communicate only using the message queue internally—nova-scheduler and nova-console—on a new server for expansion. However, other integral parts require more care. You should load balance user-facing services such as dashboard, nova-api, or the Object Storage proxy. Use any standard HTTP load-balancing method (DNS round robin, hardware load balancer, or software such as Pound or HAProxy). One caveat with dashboard is the VNC proxy, which uses the WebSocket protocol—something that an L7 load balancer might struggle with. See also Horizon session storage. You can configure some services, such as nova-api and glance-api, to use multiple processes by changing a flag in their configuration file—allowing them to share work between multiple cores on the one machine.

Segregating Your Cloud When you want to offer users different regions to provide legal considerations for data storage, redundancy across earthquake fault lines, or for low-latency API calls, you segregate your cloud. Use one of the following OpenStack methods to segregate your cloud: cells, regions, availability zones, or host aggregates. Each method provides different functionality and can be best divided into two groups:

• Cells and regions, which segregate an entire cloud and result in running separate Compute deployments.

• Availability zones and host aggregates, which merely divide a single Compute deployment.

Cells and Regions

OpenStack Compute cells are designed to allow running the cloud in a distributed fashion without having to use more complicated technologies, or be invasive to existing nova installations. Hosts in a cloud are partitioned into groups called cells. Cells are configured in a tree. The top-level cell ("API cell") has a host that runs the nova-api service, but no nova-compute services. Each child cell runs all of the other typical nova-* services found in a regular installation, except for the nova-api service. Each cell has its own message queue and database service and also runs nova-cells, which manages the communication between the API cell and child cells. This allows for a single API server being used to control access to multiple cloud installations. Introducing a second level of scheduling (the cell selection), in addition to the regular nova-scheduler selection of hosts, provides greater flexibility to control where virtual machines are run. Unlike having a single API endpoint, regions have a separate API endpoint per installation, allowing for a more discrete separation. Users wanting to run instances across sites have to explicitly select a region. However, the additional complexity of a running a new service is not required. The OpenStack dashboard (horizon) can be configured to use multiple regions. This can be configured through the AVAILABLE_REGIONS parameter.

Availability Zones and Host Aggregates

You can use availability zones, host aggregates, or both to partition a nova deployment. Availability zones are implemented through and configured in a similar way to host aggregates.

Availability zone

This enables you to arrange OpenStack compute hosts into logical groups and provides a form of physical isolation and redundancy from other availability zones, such as by using a separate power supply or network equipment. You define the availability zone in which a specified compute host resides locally on each server. An availability zone is commonly used to identify a set of servers that have a common attribute. For instance, if some of the racks in your data center are on a separate power source, you can put servers in those racks in their own availability zone. Availability zones can also help separate different classes of hardware. When users provision resources, they can specify from which availability zone they want their instance to be built. This allows cloud consumers to ensure that their application resources are spread across disparate machines to achieve high availability in the event of hardware failure.

Host aggregates zone

This enables you to partition OpenStack Compute deployments into logical groups for load balancing and instance distribution. You can use host aggregates to further partition an availability zone. For example, you might use host aggregates to partition an availability zone into groups of hosts that either share common resources, such as storage and network, or have a special property, such as trusted computing hardware. A common use of host aggregates is to provide information for use with the nova-scheduler. For example, you might use a host aggregate to group a set of hosts that share specific flavors or images. The general case for this is setting key-value pairs in the aggregate metadata and matching key-value pairs in flavor's extra_specs metadata. The AggregateInstanceExtraSpecsFilter in the filter scheduler will enforce that instances be scheduled only on hosts in aggregates that define the same key to the same value. An advanced use of this general concept allows different flavor types to run with different CPU and RAM allocation ratios so that high-intensity computing loads and low-intensity development and testing systems can share the same cloud without either starving the high-use systems or wasting resources on low-utilization systems. This works by setting metadata in your host aggregates and matching extra_specs in your flavor types. The first step is setting the aggregate metadata keys cpu_allocation_ratio and ram_allocation_ratio to a floating-point value. The filter schedulers AggregateCoreFilter and AggregateRamFilter will use those values rather than the global defaults in nova.conf when scheduling to hosts in the aggregate. It is important to be cautious when using this feature, since each host can be in multiple aggregates but should have only one allocation ratio for each resources. It is up to you to avoid putting a host in multiple aggregates that define different values for the same resource. This is the first half of the equation. To get flavor types that are guaranteed a particular ratio, you must set the extra_specs in the flavor type to the key-value pair you want to match in the aggregate. For example, if you define extra_specs cpu_allocation_ratio to

"1.0", then instances of that type will run in aggregates only where the metadata key cpu_allocation_ratio is also defined as "1.0." In practice, it is better to define an additional key-value pair in the aggregate metadata to match on rather than match directly on cpu_allocation_ratio or core_allocation_ratio. This allows better abstraction.

For example, by defining a key overcommit and setting a value of "high," "medium," or "low," you could then tune the numeric allocation ratios in the aggregates without also needing to change all flavor types relating to them.

Scalable Hardware

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

Hardware Procurement

“The Cloud” has been described as a volatile environment where servers can be created and terminated at will. While this may be true, it does not mean that your servers must be volatile. Ensuring that your cloud’s hardware is stable and configured correctly means that your cloud environment remains up and running. Basically, put effort into creating a stable hardware environment so that you can host a cloud that users may treat as unstable and volatile. OpenStack can be deployed on any hardware supported by an OpenStack-compatible Linux distribution. Hardware does not have to be consistent, but it should at least have the same type of CPU to support instance migration. The typical hardware recommended for use with OpenStack is the standard value-for-money offerings that most hardware vendors stock. It should be straightforward to divide your procurement into building blocks such as "compute," "object storage," and "cloud controller," and request as many of these as you need. Alternatively, should you be unable to spend more, if you have existing servers—provided they meet your performance requirements and virtualization technology—they are quite likely to be able to support OpenStack.

Capacity Planning
OpenStack is designed to increase in size in a straightforward manner. Taking into account the considerations that we've mentioned in this chapter—particularly on the sizing of the cloud controller—it should be possible to procure additional compute or object storage nodes as needed. New nodes do not need to be the same specification, or even vendor, as existing nodes. For compute nodes, nova-scheduler will take care of differences in sizing having to do with core count and RAM amounts; however, you should consider that the user experience changes with differing CPU speeds. When adding object storage nodes, a weight should be specified that reflects the capability of the node.

Ephemeral Storage If you deploy only the OpenStack Compute Service (nova), your users do not have access to any form of persistent storage by default. The disks associated with VMs are "ephemeral," meaning that (from the user's point of view) they effectively disappear when a virtual machine is terminated. Persistent Storage Persistent storage means that the storage resource outlives any other resource and is always available, regardless of the state of a running instance. Today, OpenStack clouds explicitly support two types of persistent storage: object storage and block storage. Object Storage With object storage, users access binary objects through a REST API. You may be familiar with Amazon S3, which is a well-known example of an object storage system. Object storage is implemented in OpenStack by the OpenStack Object Storage (swift) project. If your intended users need to archive or manage large datasets, you want to provide them with object storage. In addition, OpenStack can store your virtual machine (VM) images inside of an object storage system, as an alternative to storing the images on a file system. OpenStack Ops Guide July 27, 2015 62 OpenStack Object Storage provides a highly scalable, highly available storage solution by relaxing some of the constraints of traditional file systems. In designing and procuring for such a cluster, it is important to understand some key concepts about its operation. Essentially, this type of storage is built on the idea that all storage hardware fails, at every level, at some point. Infrequently encountered failures that would hamstring other storage systems, such as issues taking down RAID cards or entire servers, are handled gracefully with OpenStack Object Storage. A good document describing the Object Storage architecture is found within the developer documentation—read this first. Once you understand the architecture, you should know what a proxy server does and how zones work. However, some important points are often missed at first glance. When designing your cluster, you must consider durability and availability. Understand that the predominant source of these is the spread and placement of your data, rather than the reliability of the hardware. Consider the default value of the number of replicas, which is three. This means that before an object is marked as having been written, at least two copies exist—in case a single server fails to write, the third copy may or may not yet exist when the write operation initially returns. Altering this number increases the robustness of your data, but reduces the amount of storage you have available. Next, look at the placement of your servers. Consider spreading them widely throughout your data center's network and power-failure zones. Is a zone a rack, a server, or a disk? Object Storage's network patterns might seem unfamiliar at first. Consider these main traffic flows: • Among object, container, and account servers • Between those servers and the proxies • Between the proxies and your users Object Storage is very "chatty" among servers hosting data—even a small cluster does megabytes/second of traffic, which is predominantly, “Do you have the object?”/“Yes I have the object!” Of course, if the answer to the aforementioned question is negative or the request times out, replication of the object begins. Consider the scenario where an entire server fails and 24 TB of data needs to be transferred "immediately" to remain at three copies—this can put significant load on the network. OpenStack Ops Guide July 27, 2015 63 Another fact that's often forgotten is that when a new file is being uploaded, the proxy server must write out as many streams as there are replicas— giving a multiple of network traffic. For a three-replica cluster, 10 Gbps in means 30 Gbps out. Combining this with the previous high bandwidth demands of replication is what results in the recommendation that your private network be of significantly higher bandwidth than your public need be. Oh, and OpenStack Object Storage communicates internally with unencrypted, unauthenticated rsync for performance—you do want the private network to be private. The remaining point on bandwidth is the public-facing portion. The swift-proxy service is stateless, which means that you can easily add more and use HTTP load-balancing methods to share bandwidth and availability between them. More proxies means more bandwidth, if your storage can keep up. Block Storage Block storage (sometimes referred to as volume storage) provides users with access to block-storage devices. Users interact with block storage by attaching volumes to their running VM instances. These volumes are persistent: they can be detached from one instance and re-attached to another, and the data remains intact. Block storage is implemented in OpenStack by the OpenStack Block Storage (cinder) project, which supports multiple back ends in the form of drivers. Your choice of a storage back end must be supported by a Block Storage driver. Most block storage drivers allow the instance to have direct access to the underlying storage hardware's block device. This helps increase the overall read/write IO. However, support for utilizing files as volumes is also well established, with full support for NFS, GlusterFS and others. These drivers work a little differently than a traditional "block" storage driver. On an NFS or GlusterFS file system, a single file is created and then mapped as a "virtual" volume into the instance. This mapping/translation is similar to how OpenStack utilizes QEMU's file-based virtual machines stored in /var/lib/nova/instances.

OpenStack Storage Concepts
Choosing Storage Back Ends Users will indicate different needs for their cloud use cases. Some may need fast access to many objects that do not change often, or want to set a time-to-live (TTL) value on a file. Others may access only storage that is mounted with the file system itself, but want it to be replicated instantly when starting a new instance. For other systems, ephemeral storage— storage that is released when a VM attached to it is shut down— is the pre- OpenStack Ops Guide July 27, 2015 65 ferred way. When you select storage back ends, ask the following questions on behalf of your users: • Do my users need block storage?

• Do my users need object storage?

• Do I need to support live migration?

• Should my persistent storage drives be contained in my compute nodes, or should I use external storage?

• What is the platter count I can achieve? Do more spindles result in better I/O despite network access?

• Which one results in the best cost-performance scenario I'm aiming for?

• How do I manage the storage operationally?

• How redundant and distributed is the storage? What happens if a storage node fails? To what extent can it mitigate my data-loss disaster scenarios?

4.2.5 OpenStack Object Storage (swift)

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

Ceph

A scalable storage solution that replicates data across commodity storage nodes. Ceph was originally developed by one of the founders of DreamHost and is currently used in production there. Ceph was designed to expose different types of storage interfaces to the end user: it supports object storage, block storage, and file-system interfaces, although the file-system interface is not yet considered production-ready. Ceph supports the same API as swift for object storage and can be used as a back end for cinder block storage as well as back-end storage for glance images.

Ceph supports "thin provisioning," implemented using copy-on-write. This can be useful when booting from volume because a new volume can be provisioned very quickly. Ceph also supports keystone-based authentication (as of version 0.56), so it can be a seamless swap in for the default OpenStack swift implementation. Ceph's advantages are that it gives the administrator more fine-grained control over data distribution and replication strategies, enables you to consolidate your object and block storage, enables very fast provisioning of bootfrom-volume instances using thin pro-visioning, and supports a distributed file-system interface, though this interface is not yet recommended for use in production deployment by the Ceph project. If you want to manage your object and block storage within a single system, or if you want to support fast boot-fromvolume, you should consider Ceph.

Gluster

A distributed, shared file system. As of Gluster version 3.3, you can use Gluster to consolidate your object storage and file storage into one unified file and object storage solution, which is called Gluster For OpenStack (GFO). GFO uses a customized version of swift that enables Gluster to be used as the backend storage. The main reason to use GFO rather than regular swift is if you also want to support a distributed file system, either to support shared storage live migration or to provide it as a separate service to your end users. If you want to manage your object and file storage within a single system, you should consider GFO.

LVM

The Logical Volume Manager is a Linux-based system that provides an abstraction layer on top of physical disks to expose logical volumes to the operating system. The LVM back-end implements block storage as LVM logical partitions. On each host that will house block storage, an administrator must initially create a volume group dedicated to Block Storage volumes. Blocks are created from LVM logical volumes.

ZFS

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

Sheepdog

Sheepdog is a userspace distributed storage system. Sheepdog scales to several hundred nodes, and has powerful virtual disk management features like snapshot, cloning, rollback, thin provisioning. It is essentially an object storage system that manages disks and aggregates the space and performance of disks linearly in hyper scale on commodity hardware in a smart way. On top of its object store, Sheepdog provides elastic volume service and http service. Sheepdog does not assume anything about kernel version and can work nicely with xattr-supported file systems.

5. Description of Costs [7]

5.1 Direct or Directly Attributable Costs

Are costs which are being generated by the production of a specific service or a specific product. These costs will not been generated if the company or organization stops the production of this specific product/service.

Joint Cost:
A joint cost is an expenditure that benefits more than one product, and for which it is not possible to separate the contribution to each product. The accountant needs to determine a consistent method for allocating joint costs to products.Joint costs are likely to occur to some extent at different points in any manufacturing process.

5.1.1 Common costs

Costs that are common to several products, processes, activities, departments, territories, etc. Often common costs are subsequently allocated to each of the joint products, joint processes, etc. in order to determine the cost of each

Such benefits can increase customer satisfaction and reduce churn, which may prove to be the tipping point when service providers evaluate whether to move certain applications to a cloud platform.

5.2 NFV and innovation in operations

Cloud computing and network functions virtualization (NFV) can help manage rapid demand growth while reducing capital and operational expenditures (CAPEX and OPEX). No wonder service providers are paying attention.

These savings significantly lower service providers' total cost of ownership (TCO) and increase agility — critical to thriving in today's challenging telecom environment. Virtualizing applications also simplifies complex processes, such as healing, scaling, and software upgrades, providing further agility and flexibility.

Much has been said about how virtualization and the cloud may be used in the telecom industry to improve the infrastructure and operations TCO. Initial efforts centered on virtualization's ability to optimize hardware. Lately, the focus has shifted to operations.

While virtualizing some network functions will undoubtedly bring CAPEX savings, NFV's greatest contribution will be that it enables a new way of approaching telecommunications. This means much more than just optimizing inefficiencies inherent in current processes.

Service providers can — and should — take advantage of NFV technology to redefine their current operations. This will require 3 major steps:

Map out every current process in detail

Analyze what can be automated (that is, handled by an NFV platform) to reduce complexity

Redesign operations to be much simpler and more agile

5.2.1 Cost drivers for NFV

Many parameters may be considered when developing a business case to analyze the impact of migrating an application to NFV (Figure 1).

There are 3 categories of cost drivers:

CAPEX: one-time investments in fixed assets with a useful life extending beyond the taxable year

Infrastructure OPEX: ongoing costs directly related to the infrastructure (e.g., maintenance)

Process OPEX: ongoing staffing costs directly related to the daily management of activities or processes required to provide services or applications

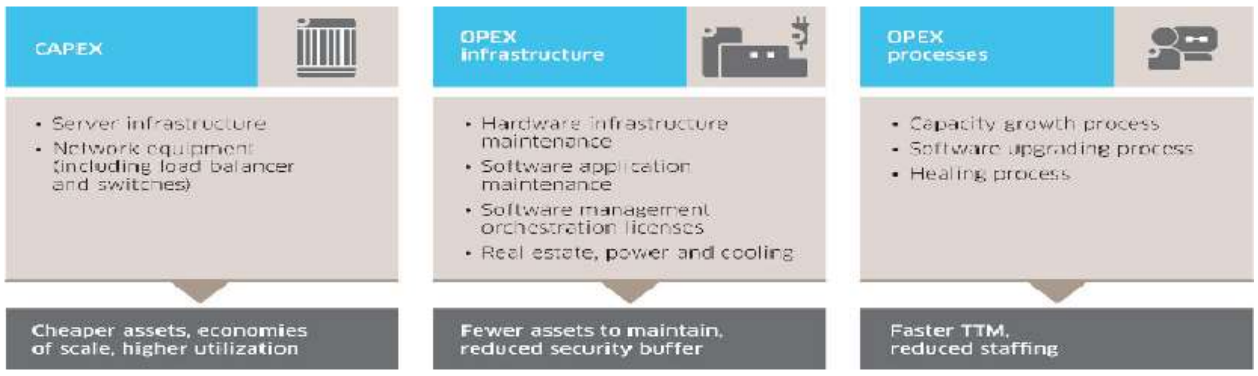

CAPEX
• Server infrastructure
• Network equipment (including load balancer and switches)
Cheaper assets, economies of scale, higher utilization
OPEX infrastructure
• Hardware infrastructure maintenance
• Software application maintenance
• Software management orchestration licenses
• Real estate, power and cooling
Fewer assets to maintain, reduced security buffer
OPEX processes
• Capacity growth process
• Software upgrading process
• Healing process
Faster TTM, reduced staffing

Figure 9: Main NFV cost drivers

5.3 Six areas of cost saving enabled by NFV [6]

Capacity growth
Traditional approaches to adding capacity follow a 4-step process (Figure 10) to deploy a new server infrastructure.

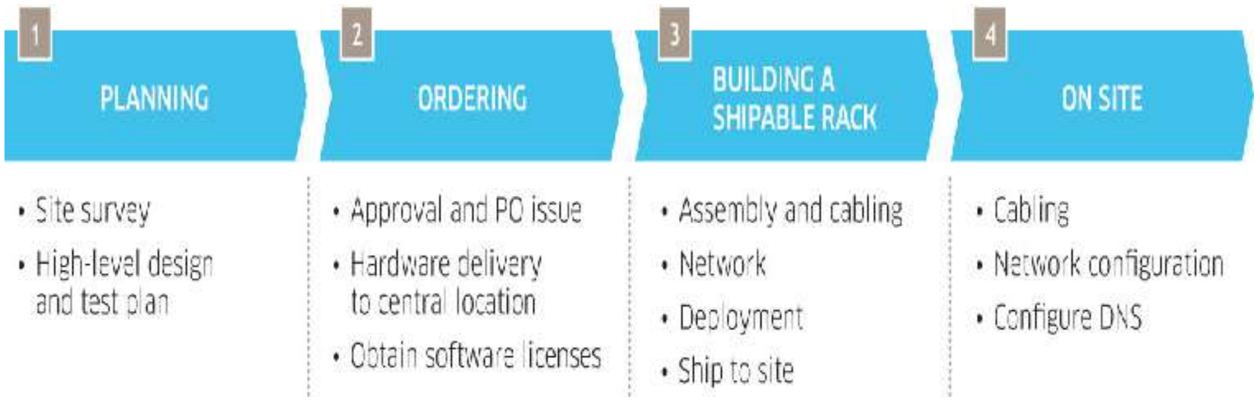


Figure 10: Traditional deployment process

The NFV deployment process (Figure 11) differs from the traditional process in a number of ways.

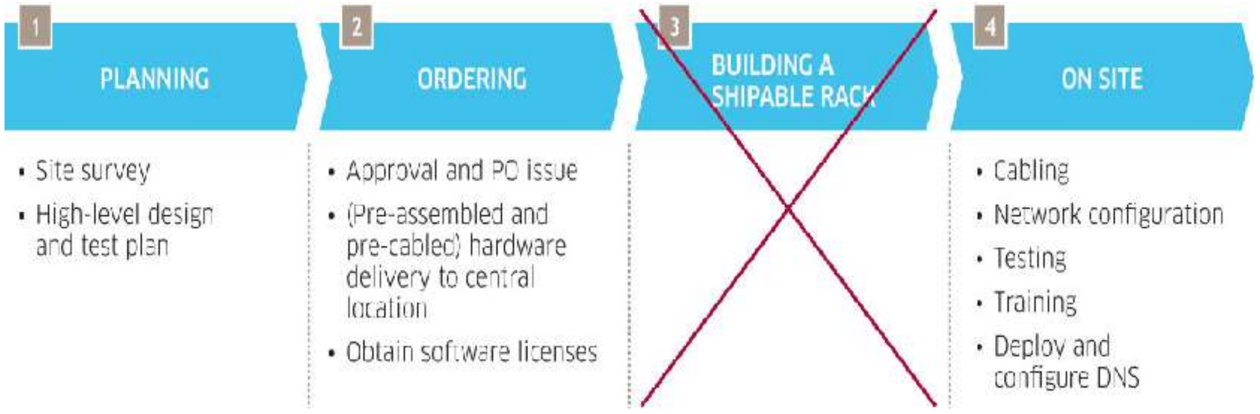

1
PLANNING
• Site survey
• High-level design and test plan
2
ORDERING
• Approval and PO issue
• (Pre-assembled and pre-cabled) hardware delivery to central location
• Obtain software licenses
3
BUILDING A SHIPABLE RACK
4
ON SITE
• Cabling
• Network configuration
• Testing
• Training
• Deploy and configure DNS

Figure 11: NFV deployment process

Costs for deploying NFV are slightly higher initially due to the professional services needed during deployment. However, this is a one-time cost. As the service provider becomes more familiar with the infrastructure, it will likely use its own operators and perform these tasks in house. With NFV, applications can share infrastructure, so the service provider’s operations team will only need to be familiar with a very limited number of infrastructure elements.

In succeeding years, total server replacement and growth process costs are greatly reduced. NFV’s virtual scaling and automated application deployment capabilities reduce capacity growth process costs significantly.

Software upgrades
Today, upgrading with both new programmed software releases and ad-hoc patches follows 4 phases:

Plan

Obtain the new software

Test the new software

Install and configure

The last phase generally consumes the most time and resources.

Introducing an NFV platform doesn't typically change the way the provider plans and obtains software. However, NFV offers a reduced timeline and lower costs to stage tests and create environments. Service providers can use "sandbox" testing environments without dedicated equipment. This lets them create simplified test cases which can be executed in parallel, and reduces testing time by about one-third.

NFV simplifies installation and configuration.
Traditionally, service providers open maintenance windows at night to install and configure a predefined number of servers individually. With NFV, the service provider can upgrade 4 servers per night in a 5-hour maintenance window. The lead time maintenance window grows over time as the service provider increases the number of physical servers to keep up with growing traffic needs (Figure 12).

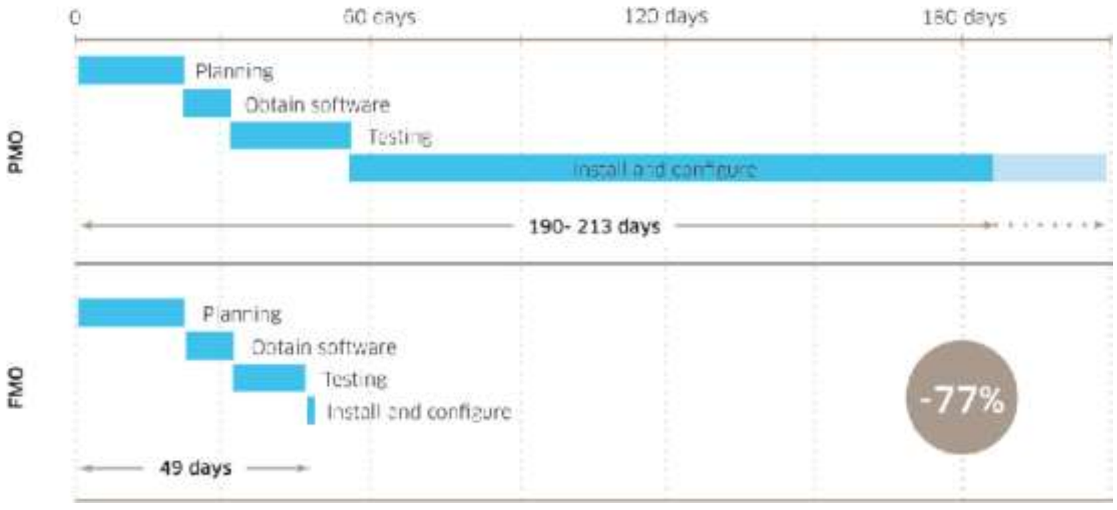


Figure 12: Example of lead times to upgrade software

NFV changes the whole process. The total number of servers is no longer relevant for installation and configuration. Application recipes are used to push upgrades automatically, in a matter of minutes, to all servers in parallel. This automation provides dramatic gains in agility.

Healing process
Device failures can result in loss of service for many users and increase churn. To reduce this risk, service providers traditionally deploy fully redundant architectures. This costly security buffer requires double the amount of physical infrastructure, with much of it standing idle.

A device failure is not the only issue that can require a healing process. Service providers also need to be able to address OS failures, application failures and distributed denial of service (DDOS) attacks.

Traditional healing process
Today's healing process consists of 3 stages:

Issue identification

Trigger and execute solution process

Perform ‘Post-mortem’ root cause analysis (RCA)

Lead times to identify and solve a problem vary depending on the issue at hand. It tends to be simpler and faster at the hardware and operating system layers, whereas actually solving an issue tends to be faster at the application layer. DDOS attacks are the fastest both to identify and solve, but tend to consume more of the operations team’s time because they are so common.

RCA is performed by operators once service continuity has been assured. Identifying the root cause of a problem allows service providers to make the changes necessary to avoid reoccurrences.

NFV healing process
With NFV, devices run as virtualized functions and are protected by the self-healing properties of the hypervisor and orchestration layer. The healing process is fully redefined as the business continuity process is decoupled from the problem itself. To provide end-to-end application resiliency and reliability, NFV platforms incorporate mechanisms for automated healing, based on the monitored infrastructure and application-level KPIs. When failures occur, the system automatically creates a new instance with the same specifications to ensure application availability at all times.

By simplifying the healing process and developing a simple solution using automated virtual scaling capabilities, NFV can significantly reduce healing costs.

Floor space, power and cooling
Real estate, power and cooling are OPEX infrastructure costs. They are directly related to the number and characteristics of physical infrastructure items managed for a specific deployment. Provided all constants remain equal, reducing physical hardware will lower the total costs of real estate, power, and cooling by the same proportion.

The main drivers for these costs are:

Real estate: number and size of infrastructure items and square foot cost

Power: rate of energy consumption and cost per kilowatt hour

Cooling: a factor of 1:1 of power consumption

With NFV, real estate costs are reduced because the technology requires fewer physical infrastructure items. With traditional approaches, load balancers and other networking equipment such as switches are placed separately from servers.

Power costs are reduced because NFV makes it possible for service providers to replace older servers sooner. Older servers consume about twice as much energy as new ones.

Lastly, cooling costs are generally calculated as a 1:1 ratio to power costs, hence, cooling costs decrease in the same proportion as encountered with power.

Maintenance and software licenses
Maintenance is also an OPEX infrastructure cost. It’s directly related to the number and characteristics of physical infrastructure items managed by the

operations teams. Many traditional infrastructure elements require a yearly maintenance fee, including servers and the network equipment, such as load balancers, switches, and routing ports.

While any chosen NFV system will have associated licenses and maintenance fees, there will be considerably fewer licenses than when using a traditional approach. This is because far fewer infrastructure elements are required, and NFV platforms can be shared between applications or services as capacity needs change.

Hardware infrastructure
Virtualizing physical assets improves resource utilization by creating virtual machines, each with its own operating system on a single physical hardware asset. An NFV platform goes a step further. It enables dynamic placement of the virtual machines, which further improves hardware optimization.

Traditional deployments operate in a “siloed” architecture. Servers are dedicated to one application, resulting in an inefficiently high number of servers.

NFV enables a new model, where all underlying hardware forms a pool of resources shared by all the applications running on the same platform. Furthermore, the ability to share the infrastructure permits a new cost model. The cost of idle capacity should not be allocated to a specific application, but is available for other applications on demand.

Service providers can expect significant reductions in server costs with NFV, since it uses far fewer servers than traditional approaches. More importantly, physical appliances, such as load balancers, can be eliminated.

5.4 Comparison of NFV and Hardware Approach

NFV approach
Capacity growth→ Joint Cost as it common for a family of services that a company produce.

Software Upgrades→Joint Cost as it is common for a family of services that a company produce

Healing Process→Joint Cost as it is common for a family of services that a company produce

Real estate, power and cooling→Joint Cost

Maintenance Cost→Directly Costs as it is referred to cost for a specific service

Licenses Cost→Joint Cost as licenses are related to product as a whole and not for a specific service

Hardware infrastructure→Joint Cost as hardware in NFV approach is related to many elements and services which are being produced by it.

Traditional approach

Capacity growth→Direct-Variable Cost

Software Upgrades→Direct-Variable Cost

Healing Process→ Direct Fixed Cost

Real estate, power and cooling→Joint Cost

Maintenance Costs→Directly Cost

Licenses Cost→Directly Cost

Hardware Cost→Joint Cost

NFV:

Direct Cost: Maintenance Cost

Joint Cost: Capacity growth-Software Upgrades-RPC-Healing-Licences-Hardware

Common Cost

Traditional:

Direct Cost: Maintenance Cost- Licences- Healing- Capacity growth-Software Upgrades

Joint Cost: RPC- Hardware

Common Cost

If the Common Cost for the two approaches are the same we see that in NFV approach most Cost become Joint from Direct where they were in Traditional Approach.

NFV:

Direct Cost: Maintenance Cost

Joint Cost: Capacity growth-Software Upgrades-RPC-Healing-Licences-Hardware

Traditional:

Direct Cost: Maintenance Cost- Licences- Healing- Capacity growth-Software Upgrades

Joint Cost: RPC- **Hardware**

As we can figure out from the above analysis of the comparison of sum of these costs are as shown below and this is because the discrete costs of NFV are lower than those of Traditional Approach. So we can write:

NFV Costs Approach < Traditional Costs Approach

NFV Costs (Maintenance Cost + Capacity growth +Software Upgrades+RPC-Healing+Licences+Hardware) < Traditional Costs (Maintenance Cost + Capacity growth +Software Upgrades+RPC-Healing+Licences+Hardware)

In this Chapter we will try to use different types of costing which are being used in Telecom industry in order to support the fact that NFV gives less Operational Costs to Vendors than the Hardware approach which is being used today. For our analysis we will use three different models of costing.

5.5 COSTS

5.5.1 Fully Distributed Cost (FDC) [9]

FDC assumes that there exist some accounts that can be specifically allocated to a single service, while other accounts are classified as common or overhead cost to two or more services. To allocate common costs, input coefficients are usually developed as parameters to be estimated when dividing common costs among groups of shared inputs. For example, assume there are two services, A and B. Total costs for the production of the two services is 100, of which 20 is attributable to service A and 30 is attributable to service B. The remaining cost of 50 is considered common costs. If the input coefficient is 0.5 for each service, then the common costs are shared equally, that is 25 for each service. The cost of each service is therefore 45 for service A and 55 for service B.

With the above costing method is clear that NFV approach has lower operation costs and this is because most costs of NFV are Joint costs. This gives the advantage of adding more services to the chain of costs. For example in traditional approach for having a Capacity Growth, the cost of this issue is being added as a whole something that is not happened in NFV architecture where the cost is shared in joint Costs.

5.5.2 Embedded Direct Cost (EDC)

This method calculates only the Direct or Directly attributable Cost. Joint Costs and Common Costs are not calculated at all. This method helps in understanding the characteristics costs of each service explicitly. These Costs are either constant or directly for a service.

The use of this method is shown to us that NFV approach is better than the Traditional. This is because in NFV approach we have only a direct Cost, and this is the maintenance Cost, in addition to traditional approach where we have extra services like Licenses, Healling process, Software Upgrade and Capacity Growth.

NFV:

Direct Cost: Maintenance Cost

Traditional:

Direct Cost: Maintenance Cost- Licences- Healing- Capacity growth-Software Upgrades

So the traditional approach add Licences- Healing- Capacity growth-Software Upgrades extra costs to vendors.

.

5.5.3 Stand Alone Cost (SAC)

Last but not least the approach of stand alone cost calculate its specific cost like it would be the only one in the chain of Cost.

So a cost for a service is the sum of the Direct Cost of that service, the whole common Cost and the whole Joint Cost. If we think that Common Cost is the same for both approaches then the whole Cost for all services in NFV is as below:

Direct Cost(Maintenance Cost) + Joint Cost (Licenses-Healing-Capacity Growth-Software Upgrade-RPC-Hardware)

In traditional approach the whole costs is calculated as below:

Direct Cost(Maintenance Cost) + Joint Cost (RPC-Hardware) + Direct Cost(Licenses Cost) + Joint Cost (RPC-Hardware)+ Direct Cost(Healling Cost) + Joint Cost (RPC-Hardware) + Direct Cost(Growth Capacity Cost) + Joint Cost (RPC-Hardware) + Direct Cost(Software Upgrade Cost) + Joint Cost (RPC-Hardware)

Or

Direct Cost(Maintenance Cost) + Direct Cost(Licenses Cost) + Direct Cost(Healling Cost) + Direct Cost(Growth Capacity Cost) + Direct Cost(Software Upgrade Cost) + 5 [Joint Cost (RPC-Hardware)]

As a result the difference between two approaches if we consider that all specific costs, one by one, are the same is:

SAC(Traditional) - SAC(NFV) = 4 [Joint Cost (RPC-Hardware)]

If we think that NFV(RPC) is lower than Traditional(RPC) and NFV(Hardware) is lower than Traditional(Hardware), then the difference is more than 4 [Joint Cost (RPC-Hardware)].

6. RESULTS

From this document there are several results we can find. First of all the use of OpenStack software in new Telecommunication architecture is something that can be done. OpenStack can be used in order to manage and orchestrate better the available hardware sources for the deployment of Telecommunication. OpenSatck is also compatible with the general approach of NFV architecture which ETSI has described. This is very important because an Opensource software can be used for the maintenance of Telecommunication networks. But the most important of all is that NFV saves money from vendors and especially from OPEX which have to invest for their networks.

ABBREVIATIONS-ACRONYMS

NFV	Network Function Virtualization
VNF	Virtual Network Functions
TCP/IP	Transmission Control Protocol/ Internet Protocol
FDC	Fully Distributed Cost
EDC	Embedded Distributed Cost
SAC	Stand Alone Cost
VIM	Virtual Infrastructure Management
W3C	World Wide Web Consortium
ETSI	European Telecommunication Standard Industry
MANO	Maintenance and Orchestration
CAPEX	Capital Expenditure
OPEX	Operational Expenditure
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

REFERENCES

[1] http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/004/01.01.01_60/gs_NFV-INF004v010101p.pdf

[2] http://www.ietf.org/proceedings/88/slides/slides-88-opsawg-6.pdf

[3] http://docs.openstack.org/openstack-ops/openstack-ops-manual.pdf

[4]http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[5] https://en.wikipedia.org/wiki/Cloud_computing

[6] https://techzine.alcatel-lucent.com/6-concrete-ways-nfv-can-save-you-money

[7] http://www.investopedia.com/ask/answers/041415/what-are-different-types-costs-cost-accounting.asp

[8] https://en.wikipedia.org/wiki/Cloud_computing_architecture

[9] http://regulationbodyofknowledge.org/faq/telecommunication-regulation-interconnection/what-are-common-cost-models-used-for-determining-interconnection-tariffs-and-how-do-they-deal-with-common-costs/