DEPLOYING FREENEST PORTABLE PROJECT PLATFORM TO AN OPENSTACK BASED CLOUD PLATFORM

A pragmatic study into an emerging technology

Ilkka Turunen

Bachelor's Thesis
December 2011

Degree programme in Software Engineering
Technology, ICT

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
TURUNEN, Ilkka	Bachelor´s Thesis	12.12.2011
	Pages 64	Language English
	Confidential ( ) Until	Permission for web publication (X)

Title
DEPLOYING FREENEST PORTABLE PROJECT PLATFORM TO AN OPENSTACK BASED CLOUD PLATFORM

Degree Programme
Software Engineering

Tutor(s)
RANTONEN, Mika

Assigned by
JAMK University of Applied Sciences, SkyNEST-project

Abstract

The focus of this thesis was to explore and document cloud computing and what it can do to an organization with existing virtualization solutions. It uses the FreeNEST Project Platform as a reference point and as an example of an information system based on existing virtualization technology. The work was done with the SkyNEST project at JAMK University of Applied sciences, as a part of Tivit Oy's Finnish Strategic Centres for Science, Technology and Innovation research program called the Cloud Software Program.

The thesis explains the terms and technologies behind cloud computing and OpenStack, an open source project that aims to produce a system to help organizations build private clouds. The second half of the thesis documents the building of a demonstration environment utilizing OpenStack, the work being structured with the design science research method. The thesis also proposes a way to structure any thesis work on the engineering degree programmes around DRSM. Additionally, it answers two research questions about the viability of OpenStack as a production ready technology and asserts a new method to convert already existing virtual machines to cloud compatible instances. Finally the thesis explores the future possibilities of cloud computing and presents two derivative infrastructure testing environments based on the contents of the appendices. It also proposes a possibility of exploring new territories with regular office computers via an office cloud.

The appendices contain technical tutorials on how to install OpenStack Bexar release on a single node installation and also on how to convert VMware-based virtual machines to Ubuntu Enterprise Cloud-instances.

Keywords
SkyNEST, FreeNEST, Cloud Computing, OpenStack, Cloud Software, Virtualization, Instances, Virtual Machines

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN

KUVAILULEHTI

Tekijä(t) TURUNEN, Ilkka	Julkaisun laji Opinnäytetyö	Päivämäärä 12.12.2011
	Sivumäärä 64	Julkaisun kieli Englanti
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)
Työn nimi DEPLOYING FREENEST PORTABLE PROJECT PLATFORM TO AN OPENSTACK BASED CLOUD PLATFORM		
Koulutusohjelma Ohjelmistotekniikka		
Työn ohjaaja(t) RANTONEN, Mika		
Toimeksiantaja(t) Jyväskylän Ammattikorkeakoulu, SkyNEST-projekti		

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutkia ja dokumentoida pilvipalveluita ja mitä aiemmin virtualisoinnilla ratkaisuja toteuttanut organisaatio voi niiden avulla toteuttaa. Työssä käytettiin FreeNEST Project Platformia referenssinä ja esimerkkinä virtualisoinnin avulla toteutetusta järjestelmästä. Opinnäytetyö toteutettiin SkyNEST-projektissa Jyväskylän Ammattikorkeakoulun teknologia-yksikössä osana Cloud Software Programia, joka on Tivit O.y:n Strategisen Huippuosaamisen Keskittymä (SHOK)-tutkimushanke.

Opinnäytetyö selittää termejä ja teknologioita jotka ovat pilvipalveluiden ja OpenStack-nimisen projektin taustalla, jonka tarkoituksena on tuottaa avoimen lähdekoodin järjestelmä pilvipalveluiden toteuttamiseen. Opinnäytteen toinen osio dokumentoi OpenStackin avulla rakennetun demonstraatioympäristön rakentamista. Työ jäsenneltiin käyttäen design science-tutkimusmetodia.

Projektin aikana saatiin vastaus kahteen tutkimuskysymykseen OpenStack-projektin tuotantokypsyydestä ja aiemmin luotujen virtuaalikoneiden käytöstä pilvipalveluiden avulla. Projektin johdosta kehitettiin uusi metodi olemassa olevien virtuaalikoneiden muuntamiseksi pilviyhteensopiviksi. Lopuksi, projektin jälkeen on tuotettu myös kaksi johdannaisympäristöä, jotka on tehty työssä tuloksena saatujen aineistojen avulla.

Avainsanat (asiasanat)

Virtualisointi, Pilvipalvelut, SkyNEST, FreeNEST, Cloud Software Program, OpenStack, Virtuaaliko-

Muut tiedot

TABLE OF CONTENTS

ABBREVIATIONS

API	Application Programming Interface.
AWS	Amazon Web Services, a set of cloud computing services provided by Amazon Inc.
CPU	Central Processing Unit, often the abbreviation often used when talking about computer processors.
DDOS	Distributed Denial of Service. A cyber-attack technique using which a large amount of clients make simultaneous requests to a web-site server, effectively bringing the server down.
DHCP	Dynamic host control protocol, a protocol that provides clients with a range of necessary information in a network, such as their IP addresses.
DSRM	Design Science Research Method.
EC2	Elastic Compute Cloud, an Infrastructure as a Service – type product provided by Amazon Inc.
IaaS	Infrastructure as a Service, used to describe any service that offers heterogeneous and undefined hardware to run virtual machines.
ICT	Information and Communications Technology.
IT	Information Technology, an abbreviation that is more commonly used to mean computers and information systems.
OS	Operating System.
PaaS	Platform as a Service, used to describe a “middle-layer” services that run with software that are hosted as a cloud service.

RAM	Random Access Memory, a form of memory used inside computers to read and write data fast.
S3	Simple Safe Storage, a cloud-based file-hosting service provided by Amazon Inc.
SaaS	Software as a Service, used to describe any piece of software that is provided from the “cloud”.
VLAN	Virtual local area network.
VM	Virtual machine, a virtual computer that can be run using a physical computer.
XML	eXtensible Markup Language, a markup language for transferring structured sets of data across multiple platforms.

1 BACKGROUND

Cloud computing is the talk of the town; the most marketed and whispered word in information and communications technology industry in the last few years in various publications. Companies, like Dell, all around the world are increasingly concentrating their efforts in gaining an advantage in the field (VentureBeat 2011). Another example of such efforts is the Finnish Cloud Software Program, a research and development program running under administration of the Strategic Center for Science, Technology and Innovation: Tivit Oy. Cloud Software Program is a consortium-formed research initiative that consists of 30 different companies and research institutes (Tivit Oy 2011) that range from small to middle sized enterprises like RM5 Software to large multinational partners like F-Secure and Aalto University. The goal of the Cloud Software Program is to promote Finnish Cloud Expertise and establish world-class capabilities in the field inside Finland. Ultimately the program wishes to create a lively ecosystem inside Finland called Finn-Cloud that would differentiate Finland as a safe and sustainable source of cloud computing hosting.

Contrary to popular belief, cloud computing technologies and methodologies are nothing new or ground breaking in the ever-changing landscape of ICT. Larry Ellison, CEO of Oracle, a database systems development and provider corporation, stated in an interview in 2008 that cloud computing in his opinion is nothing short of a fashion term for technologies that already exist and are being implemented in real corporate cases (CNET 2008).

Be things as they may, it is however evident that cloud computing is currently one of the primary focuses of future investments and research projects. Amazon was one of the first operators to popularize and commoditize distributed computing services with their elastic compute cloud. Other major service providers and corporations have quickly followed suite, such as Microsoft introducing their own cloud platform Windows Azure in early 2010 (Microsoft 2009). Hosted solutions have been on the market for a few years, but recent times have seen the emerging of a range of open

source solutions that try to replicate the functionality of commercially available services. These open source projects are however not small efforts either. The most profilic of them is OpenStack, originally a co-operative project started by the National Aeronautics and Space Administration of The United States of America (NASA) and an American hosting provider called Rackspace that has already amassed over 53 different companies contributing back to its cause. The OpenStack project is moving at unparalleled speed, with it already having gone through two major releases in its short lifetime, currently the effort being only just under a year old (Rackspace 2010).

JAMK University of Applied Sciences was granted admittance to the Cloud Software Program in early 2011 with the help of its in-house Nest Concept, an integrated virtual machine image that acts as a reference for an environment that in turn can be used to oversee any software research and development project. JAMK University of Applied Sciences is a Jyväskylä-based higher educational facility, with a staff of 683 and approximately 8000 students studying under various degree programmes (JAMK University of Applied Sciences 2010). The employees at JAMK can be categorized in three different and separate categories: Teaching staff with 387 members, Project staff with 152 members, and other staff that measures 222 people strong. The organization is highly oriented towards teaching and also deriving extra value in teaching; many of the members of the project staff are also part-time lecturers or contribute otherwise to teaching.

At JAMK, the intention was to participate in the program by taking the Nest Concept and deploying it to a reference Private Cloud, complete with a reference architecture and model deployment. This thesis documents the approach taken and the technologies used, and can be thought as a contemporary look into the latest buzz word in the ICT field. It explains the most popular terms used when reading or talking about cloud technologies and finally provides technical documentation of a testing infrastructure implemented during the second quarter of 2011.

The project in which the deployment was performed started on January 2nd 2011, and was called SkyNEST (JAMK University of Applied Sciences 2011). The SkyNEST

project was JAMK's sub-project under the Cloud Software Program, and its main focus was on using as many students as possible and coordinating their efforts to creating valuable contributions back to the Consortium. All in all, at the time of writing, the project employed 3 members of faculty staff (the writer of this document included), and 15 students. The writer's main responsibilities were to act as a Scrum Master and a mentor for the student employees, and to research the usage of OpenStack and Cloud technologies.

2 TECHNOLOGIES AND DEFINITIONS

This section is dedicated to delving more in depth into the terms and technologies and their histories, which are relevant to understanding the technologies behind the illusive cloud.

2.1 Cloud Computing

Cloud as a term arrived to the mouths of the people working in the ICT industry as early as in the 1980s, originally used to illustrate the different access points between multiple different telecom systems. The industry gradually assimilated the usage of the symbol and term to describe large networks of computers (Heino 2010). This allows simplifying charts in visualizations to a more general overhead when larger detail is not necessary.

Cloud computing itself has roots in distributed computing that has been more or less in use since the 1960s. Originally, the idea was to distribute the computing time and capacity of large mainframe computers between different programs and sets of tests. This practice is called time-sharing and its popularity peaked in the 1970s when network speeds were an issue in rapidly commercializing the practice. Time-sharing remained primarily a way to distribute time between university mainframes for students and test datasets.

The service that first could be pinned under the moniker of Cloud Computing was Amazon's Web Services. It came to be when Amazon came to realize that they had

optimized their hosting hardware to efficiently handle the rare peak hours of a day, but on non-peak hours most of the capacity was being unused. Amazon decided to sell this computing time as a service, and the service became known as AWS (Heino 2010). Since then, Amazon has gone from simply providing hosting to being one of the frontrunners in creating a new standard in distributed computing. Amazon's cloud computing infrastructure and service offering is one of the most widely used, and from a technical point of view, other cloud computing services are often benchmarked against those of Amazon. Even to such large extent that two of its best known and most popular services have almost been elevated to a "de-facto" standard, based on which other services are often designed and re-iterated. No small feat for a former book-trading company.

Defining a clear and unified description that would summarize in one sentence what cloud computing essentially is a very demanding task. Many descriptions are often affixed to the term. Examples of such terms are elasticity, flexibility, reliability, and scalability to mention a few. However, no one of these terms are present in every service or offering that deems itself to be cloud computing. Thus, the only reliable descriptive sentence said of the cloud is that the cloud is not the end user's problem, it is someone else's (Ruottu;Lagerspetz ja Tarkoma 2011).

2.1.1 Virtualization

One of the cornerstone technologies behind cloud computing is virtualization. The term virtualization is very straightforward in its meaning; it is a technology to create virtual computers inside actual physical computers. These virtual computers or **virtual machines** operate as if they were real physical computers, and in most cases they can run standard unmodified operating systems.

Virtualization carries with itself multiple benefits. It allows optimizing load on a physical machine in to smaller pieces and quantities, so that a single one server is not confined in serving only one purpose or application. The computational capacity of the underlying machine can be used to as close to maximum efficiency as is possible. Virtualization also awards software developers an easy way of standardizing their

development environments. If we assume company X has a product Y, that is a server, and there are software developers working for that company, all of these developers need to do their work on a unified version of the server. One approach to solving the problem would be to create a multi-user dedicated server; this however has the risk of running low on capacity on peak hours, and on the other hand overusing electricity and power on non-peak hours. Virtualization offers a solution to this problem: each of the developers can create their own copy of the server by running it in a copy of a virtual machine that has the server software preinstalled. Another key benefit with virtualization is the ability to run multiple operating systems in one physical machine, without having to do a full-on power-off reboot in between changing the operating system necessary. Instead, it is possible to create multiple virtual machines, each with their own operating systems, but similar hardware. This comes especially handy when testing web-site compatibility between multiple browsers and operating systems.

The origins of virtualization lie way back in the 1960s. An article in the Association of Computing Machinery describes the history of virtualization as follows: "*The term virtual machine initially described a 1960s operating system concept: a software abstraction with the looks of a computer system's hardware (real machine).*" (Association of Computing Machinery 2004). Virtualization as an idea has indeed been around over 50 years at the time of the writing of this thesis. The concept of virtualization was first described in the 1966 article "*A Virtual Machine System for the 360/40*" by IBM researchers R. J. Adair, R. U. Bayles, L. W. Comeau, and R. J. Creasy, as conceptual software architecture for IBM's CP/CMS mainframe machine operating system. The virtualization techniques at that time consisted of software level abstraction and emulation of physical resources. Over time, virtualization has evolved significantly from a concept to a viable and cheap technology. Contemporary virtualization is common in several usage scenarios. Examples are web servers, distributed computing servers for large data sets, and most recently cloud computing. Virtualization has rapidly evolved in the 2000's in to a cost-effective and viable option for setting up computational infrastructures. Many companies now sell "virtual servers" affordably for individuals and organizations, which are actually virtual ma-

chines running on physical servers. Virtualization has evolved to the point where large processor manufacturers now include hardware hooks for enabling smooth brokering of low-level resources to the hypervisor.

Virtualization as a technology consists of multiple different components that will now be explored. The first of these is a physical computer that is called a ***host machine***. The host machine can be any sort of regular computer that can execute normal machine code necessary to run software. On the host machine, a specialized piece of software called ***the hypervisor*** is installed. The hypervisor handles the creation and execution of the virtual machine, provisioning different sections of the host machine's hardware to the virtual machines it creates. Examples of hypervisor software are: Oracle VirtualBox, VMware products, Xen, and the Kernel Virtual Machine or KVM. The machines that the hypervisor creates are often called ***guest machines***. As previously stated, these guest machines act as if they were actual computers running on real hardware. Depending on the virtualization technology used, the performance of these virtual machines can reach almost native machine levels, provided that hardware-assisted processor extensions can be used. These extensions include Intel VT-X and AMD-V. A generalized visualization of virtualization is presented in **figure 1**.

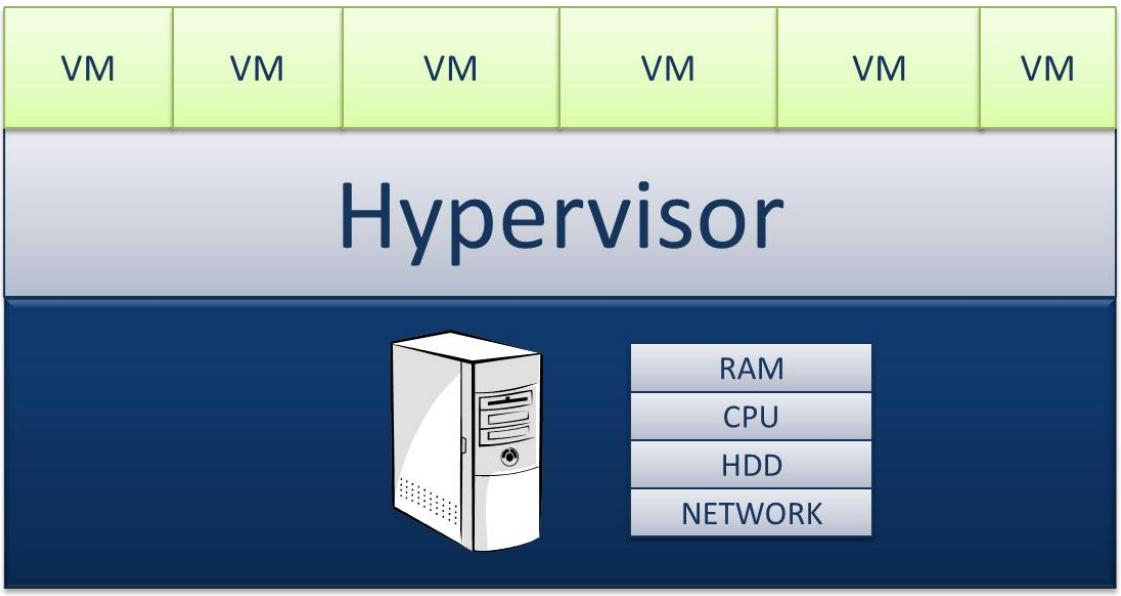

FIGURE 1 - A simplified representation of how virtualization works on a single host machine. The abbreviation VM stands for Virtual Machine

Modern virtualization can be asserted to mean the virtualization of x86-based computer systems, on which most modern computers run to this day. The different types of contemporary virtualization of this x86 architecture are paravirtualization, full virtualization, partial virtualization and hardware-assisted virtualization. The difference between each type is in the amount of isolation they provide from the host operating system and hardware, and the level of modification required to the guest operating system. For instance, paravirtualization requires the usage of a modified kernel in the guest operating system, to enable translations of some kernel procedures that cannot easily be virtualized (VMware 2007). On the other opposite end of the spectrum lies hardware assisted virtualization, allowing for direct execution of user requests on applications on the guest machine with the host's hardware, also eliminating some hypervisor overhead.

2.1.2 Types of Computational Clouds

Cloud services can generally be classified in three specific categories: public clouds, private clouds or hybrid clouds. These types represent different models of hosting altogether and the last one a combination of the two first.

A public cloud is essentially the most offered solution today. It is a type of combined computational and (/or) storage cluster that is hosted at a solution provider's premises, and from which the end customer only purchases capacity in computational time or storage space, and which does not partake in the upkeep procedures necessary to maintain the hardware in question at all. It is also a frequently used example of the benefits of cloud computing: with the help of public clouds, a client can only pay for the capacity they need, and not for excess computational power that would go to waste on low-traffic times. And in times of higher demand, the capacity can be temporarily scaled up to suit the needs of the client. This has long been a classic problem with traditional ICT-infrastructure branches, for example an IT-department inside an organization. Optimizing the fine balance between the consumption of concrete resources (electricity, cooling, building maintenance), work hours necessary to install and maintain the computational services up to date and running as intended,

to finally maintaining the actual servers and proper distribution of their resources amongst clients.

One prominent and recent example of the possibilities of this capacity management is the case of Wikileaks, an activist organization that aims to “leak” out confidential information originating from various governments, corporations and other organizations. Wikileaks recently published a large amount of classified US Embassy Cables on their website. In the aftermath of the publications, their site was subjected to A Distributed Denial of Service attack or DDOS for short, and as a result they moved their front page to be hosted by Amazon’s elastic compute cloud (abbreviated EC2). This solution allowed the front page of the site to be subjected to the huge traffic loads evident to that kind of attack, without any significant downtime. (CloudTweaks 2010)

Private cloud as a term simply means offering the same kind of capability as a service, but to a restricted group of people. This restriction is usually inside an organization, the private cloud solutions being offered as an internal service only accessible inside organization borders in an extranet.

2.1.3 Service layer models

Cloud computing is usually divided into three major service level categories that describe the different functionalities attributed to the different structures. These three layers are Infrastructure as a Service (**IaaS**), Platform as a Service (**PaaS**), and Software as a Service (**SaaS**), also sometimes called the SPI-model (illustrated in **figure 2**). This model is referred to by some researchers as the Naïve service layer model for Cloud Computing (Ruottu;Lagerspetz ja Tarkoma 2011), used most often to illustrate in an easy manner the different functionalities that are found in a given implementation of some cloud architecture.

Out of these three, Software as a service is most often mentioned or referenced when cloud computing is discussed. This hierarchy is abstracted in a way so, that a given layer does not need to be aware of the specifics of the layer(s) below it. For instance the Software layer, e.g. a hypothetical website located at

http://example.com, need not be aware of the complex infrastructure lying under it. The website looks out to be a normal hosted server. It has an IP address, a name service, and it acts as a regular server. The administrator might be granted terminal access, and all they would see upon connecting would be a regular Linux machine running web server services, with designated folders to put their code in. This is identical to how a physical machine acts and feels in the hands of a developer. In reality, however, the server is a virtual machine partially hosted in Silicon Valley, San Francisco U.S.A and partially in a server farm in Brussels, Belgium. The server acts as if it were a real physical machine running a regular distribution of an operating system (e.g. Ubuntu Linux Server edition), when in fact its functions are split amongst multiple different machines spread in different locations. The developer or the end users do not even need to know where the different parts of the whole running service are geographically situated. The experience is no different from using a regular virtual machine.

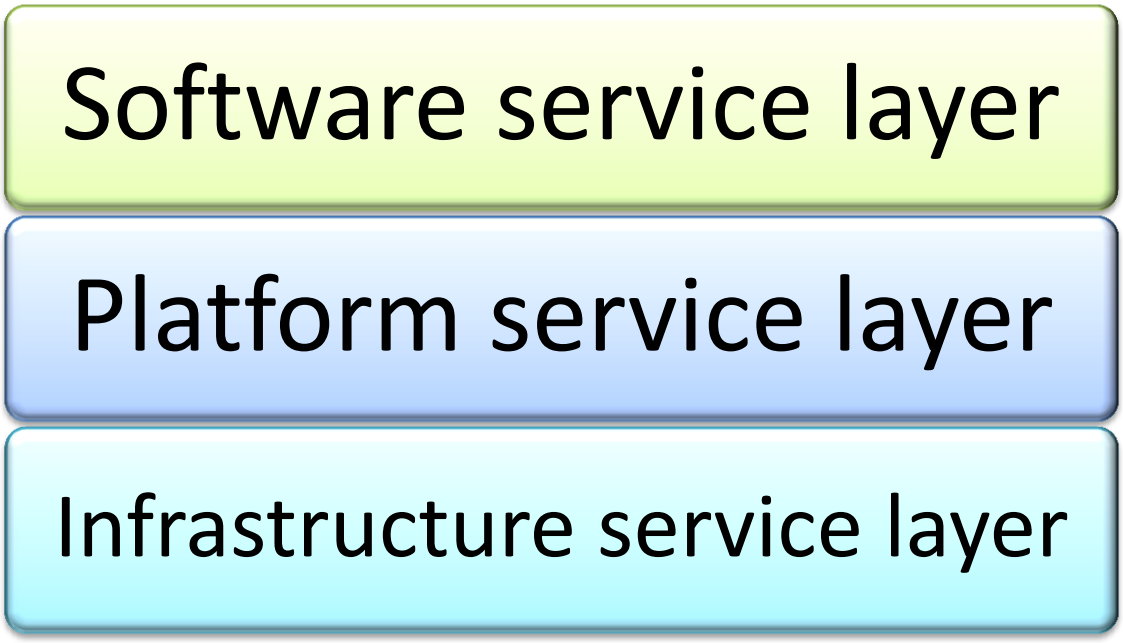

FIGURE 2: The three layers of Cloud Computing as a service (Naïve Service layer model)

Another part of the three abovementioned, Infrastructure as a service, is as the name provides, a way to provide computing infrastructure as a service, instead of an in-house support organization. IaaS therefore is essentially an abstraction term for all the hardware that is in use as the underlying backbone for the cloud, and could incorporate everything from networking to computers and servers. Because the infra-

structure is abstracted, individual numbers of worker machines do no matter anymore. The infrastructure can be made up of two or two thousand machines. Networking between these machines is also incorporated in the abstraction, and the upper levels do not need to be aware of the specifics.

Finally, the third part of the model is Platform as a Service. This layer does not have a clear definition; it is however most often recognized to include applications and services that are present on the provisioned cloud computational instances, but which do not directly present themselves to the end users. By that definition, they provide a platform on top of which a developer can build their applications. Examples of such platform services could include a script language interpreter, a web server, a database server, operating system services that enable the overlaying applications to interact with other present services, and allow developers to create applications on the cloud infrastructure.

The combining characteristic however is that there is no clear definition of each layer. The naïve separation into three segments is commonly agreed upon as the chosen method of representing cloud services, but there is no clear borderline when talking about specific components. For instance, a database server cannot always be instantly placed on either the Platform or the Infrastructure layers. It shares characteristics with both, but does not completely fulfill the desired and described functionalities of either layer. Also, none of the layers have clearly defined methods of communication neither between nor amongst components inside one another. Instead, it has been proposed the naïve model should be extended to encompass different solution sets for these layers. (Ruottu;Lagerspetz ja Tarkoma 2011).

This model states that one can implement a layer of the cloud computing architecture by combining enough solutions together. One such example is Google's cloud based services, the Google apps and their underlying infrastructure, illustrated in **figure 3**.

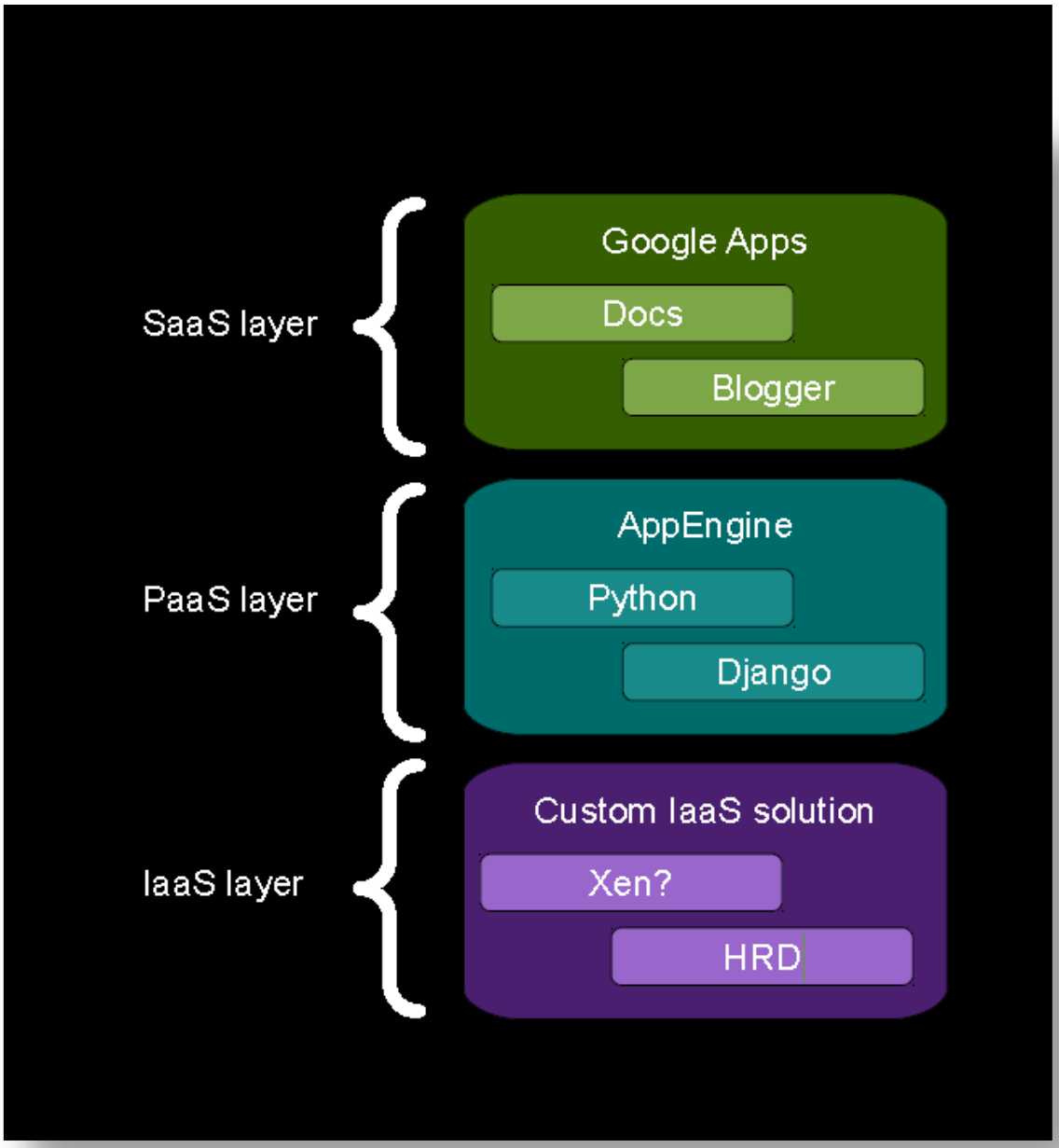

FIGURE 3- Extended SPI model with solution sets, in the context of Google Apps (Ruottu;Lagerspetz ja Tarkoma 2011)

2.1.4 Elastic Compute Cloud

Elastic Compute Cloud (often abbreviated EC2 after a service provided by Amazon LLC) is the common name for most computational cloud infrastructure services. These EC2 compliant clouds, or computation infrastructure, run copies of virtual machines called ***instances*** that are spawned from a ***virtual machine bucket.*** These instances are provisioned throughout the underlying infrastructure of hardware by a

machine called the ***API-machine***, which in a sense acts as the brains of the infrastructure network. The infrastructure consists of machines working in several different roles such as computation, scheduling, storage, networking, among others. Physical (or virtual) machines that have a certain role in such an infrastructure network are called ***nodes***. The nodes then interact through a scheduler program that uses specialized messaging software such as rabbitmq, a simple messaging bus software used for example in OpenStack, to coordinate and synchronize the work done in different nodes. The whole process is presented as a generalized visualization in Figure 4.

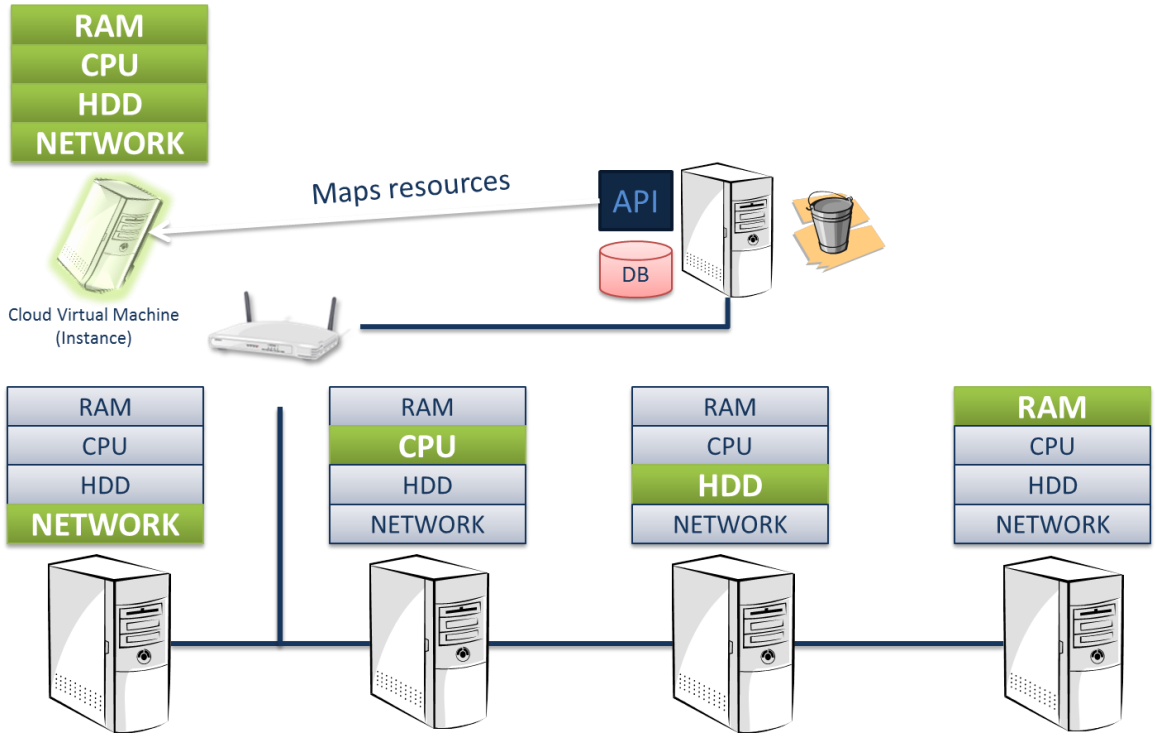

FIGURE 4 - How an EC2 compliant infrastructure works. The API machines provisions different hardware capacities from the networked infrastructure machines, a journal of which is kept in a database. This provision runs a copy of a virtual machine called an instance, the template of which has been stored in a virtual machine storage called a virtual machine bucket.

These instances reside inside their own reserved network space, which in the most common case is their own virtual local area network (abbreviated ***VLAN***). Other types of networks also exist, however, VLANs provide complete isolation of network addresses and other features meant to sandbox instances that do not exist in other alternatives, such as network masking with dynamic host control protocol-based network management solutions.

2.1.5 Simple Safe Storage

Simple Safe Storage is often abbreviated to S3, which stands for a set of services that are meant to provide scalable file storage from a storage cloud that allow files to be taken and retrieved fast from the cloud. The method is named after a similar service provided by Amazon, which was also the first implementation and commercially available version of such a system.

The S3 system itself stores files on storage nodes that split the file accordingly and then retrieve said files back when the user requests so. Communication to and from a S3 cloud is done through the S3 Application Programming Interface (abbreviated API), a XML standard that is modeled after what Amazon uses in their own systems. Individual user files are stored in individual ***buckets*** that can be accessed using a special xml-based key. These buckets can then be used to for multiple usages, such as blob storage for huge files, as an image storage bucket for the EC2 cloud, and for multiple other usages that require the handling of large data sets.

2.1.6 OpenStack

OpenStack is an open source project that aims to produce a full-blown cloud operating system for the infrastructure layer that is massively scalable (OpenStack 2011). It consists of three separate main projects: Nova, Swift, and Glance. In addition, there are also two subprojects called Authenticate and Dashboard. OpenStack is a large project moving at a very fast pace. Its primary proponents and initial founders are two companies known for large computing and facilities: RackSpace and NASA. The two institutions are also joined by some of the largest companies on the ICT industry today, such as HP, Dell, Intel, AMD, NTT, Canonical, Cisco, NEC, SUSE, to name a few. In total, at the time of writing this thesis, OpenStack has 128 companies listed as official participants, with over 1600 people working on the project in a “global community of technologists, developers, researchers, corporations and cloud computing experts.” (OpenStack 2011). The OpenStack project initially started out as collaboration between NASA and Rackspace that aimed to unify two pieces of software from both the respective companies. Originally both organizations had decided to open source some of their technologies, NASA their Nebula platform and Rackspace their file

hosting server software. The two technologies were then unified under a single project name, OpenStack (B. Piatt 2010).

Nova, also known as OpenStack Compute, is a project dedicated to creating a piece of software that aids in the creation of a virtualization cluster infrastructure out of any machine capable of running the Linux Operating system. It is based on the EC2-cloud computing model that can be found with e.g. Amazon's cloud products, described earlier in the previous chapter. It has all the same roles with nodes as an EC2 computational network. Nova consists of five different executable python binaries integrated along with 3rd party software meant to act as the messaging queue between components and nodes, and also for database needs. The original pieces of software that are associated with OpenStack nova are called nova-api, nova-compute, nova-objectstore, nova-network and nova-scheduler. Nova-api is the executable responsible for filling the role of the API-machine described in the EC2 chapter. The API-binary also interacts directly with a meta-information database to keep track of all the other different machines, their addresses and their respective node roles in the nova-network. Nova-schedule is the executable that schedules the action between nodes. The final three remaining nova-executables are responsible for node actions. The nova executables can be installed together in one machine, creating a ***single-node*** installation. Single-node installations are primarily used for testing purposes, but sometimes can also be useful in production networks. The executables can also be installed individually on separate machines. These individual machines then fill just the role of one node, and announce themselves to the API-machine with rabbitmq for further guidance in the network. The API-machine alongside 1-n nodes then creates the computational network. This installation type is also called ***multi-node***. (OpenStack 2011)

Another one of the major three OpenStack projects is OpenStack Swift, otherwise called OpenStack Object Storage. Swift, like nova, can be installed as either multi- or single-node installation with a single or multiple machines. Swift is modeled after the S3 architecture of Amazon's products, acting many ways in the same way. The Swift project also has quite deep roots in Rackspace's cloud hosting offerings, in the sense

that the first version of Swift was more or less an open source version of the production code used on Rackspace's file server infrastructure at the time. (OpenStack 2011)

Finally there is the third major project; OpenStack Glance, a virtual machine image sharing engine that helps when operating Nova across a network of heterogeneous nodes. Glance effectively replaces the nova-objectstore executable and allows sharing of binary form virtual machine images found in the network's virtual machine bucket with other nova-compute nodes.

OpenStack takes many pointers from Amazon's EC2 and S3 application programming interface and architectural models, but still takes on a distinct path in technologies and functionalities. Technically all the OpenStack projects are developed using the Python programming language and other open source technologies. The project is deliberately split into multiple projects to help focus on efforts working in individual functionalities. The integration of all different projects is illustrated in Figure 5.

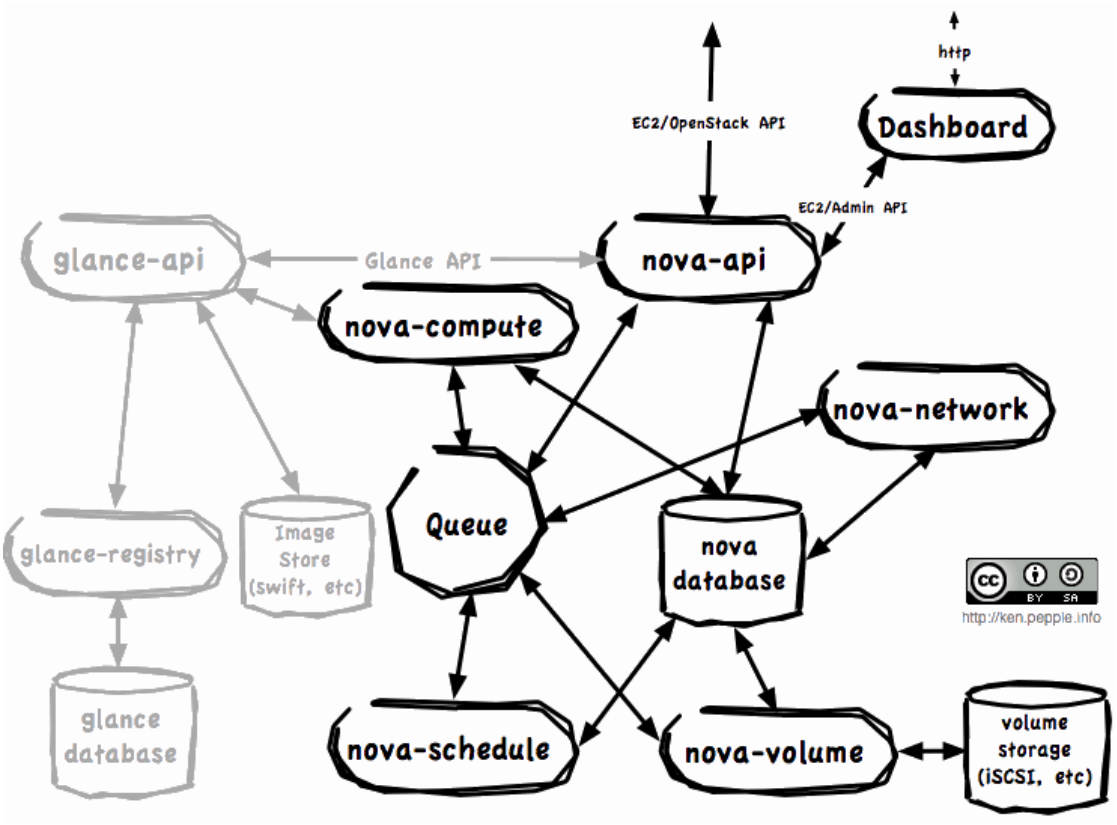

FIGURE 5 - The general architecture of OpenStack (Pepple 2011)

2.2 FreeNEST Portable Project Platform

FreeNEST Portable Project Platform is an integrated R&D platform based entirely around Open Source tools aimed at software development (Rintamäki, 2008; Perälä, 2008). FreeNEST is distributed as a virtual machine image that can be run on any hardware that supports it (in practice meaning that the processor has either Intel Vt-x or AMD-V capabilities), ranging from regular desktop hardware to full-fledged virtualization clusters. Virtualization enables the FreeNEST server to act as if it were a full-fledged individual computer running on it's own isolated hardware, with the additional option of customising such important attributes as RAM or networking easily.

FreeNEST ships as only a few files, but however inside it there is a very potent package. At the most bottom layer of the virtual machine system, FreeNEST has Ubuntu Linux 10.04 installed as its operating system. Ubuntu Linux was chosen as the OS for its easy administration tools, frequent updates and high reliability standards. On top of Ubuntu, FreeNEST has multiple different server software that see to regular functions of the platform. Examples of such servers are the Apache2 web server for handling the serving of the final web pages to clients, MySQL Database server that houses 12 different databases used for different functions of the platform, and OpenLDAP Lightweight Directory Access Protocol server that handles all authentication. Present also is a server called ircd-hybrid, an IRC or Internet Relay Chat server that enables rapid and extremely potent communication methods between clients. Additionally present are different version control system servers, such as Git and Subversion, two of the most widely used products in the software industry. Each of these components have been chosen for their high usage rates in the enterprise world and for their widely documented functions.

The third and final layer of the FreeNEST Platform consists of different applications that have their own specific focus on an Application Lifecycle Management or LEAN project models. Examples of such applications include Foswiki, an enterprise-level structured wiki (meant for knowledge and document sharing), Trac, an Open Source ticketing system that is widely in use in software houses all around the world,

Bugzilla, a bug tracking system that helps software writers and clients communicate different deficiencies and change requests to a particular product of the project. The applications are all integrated using different methods, ranging from creative usage of authentication modules to custom built plugins and user interface additions. These integrations allow the applications to communicate data and metrics relevant to each other and the end users with relative ease, and also allow at the most visible layer for the user to see a constantly present “Top Bar” that significantly helps navigation inside such an extensive system. FreeNEST also has several unique pieces of software that significantly make administering users in the server, broadcasting news, and configuring the available programs significantly easier. The relationships and the contexts of the FreeNEST Project Platform are explained with **figure 6**, and a screenshot of the system’s front page is illustrated in **figure 7**.

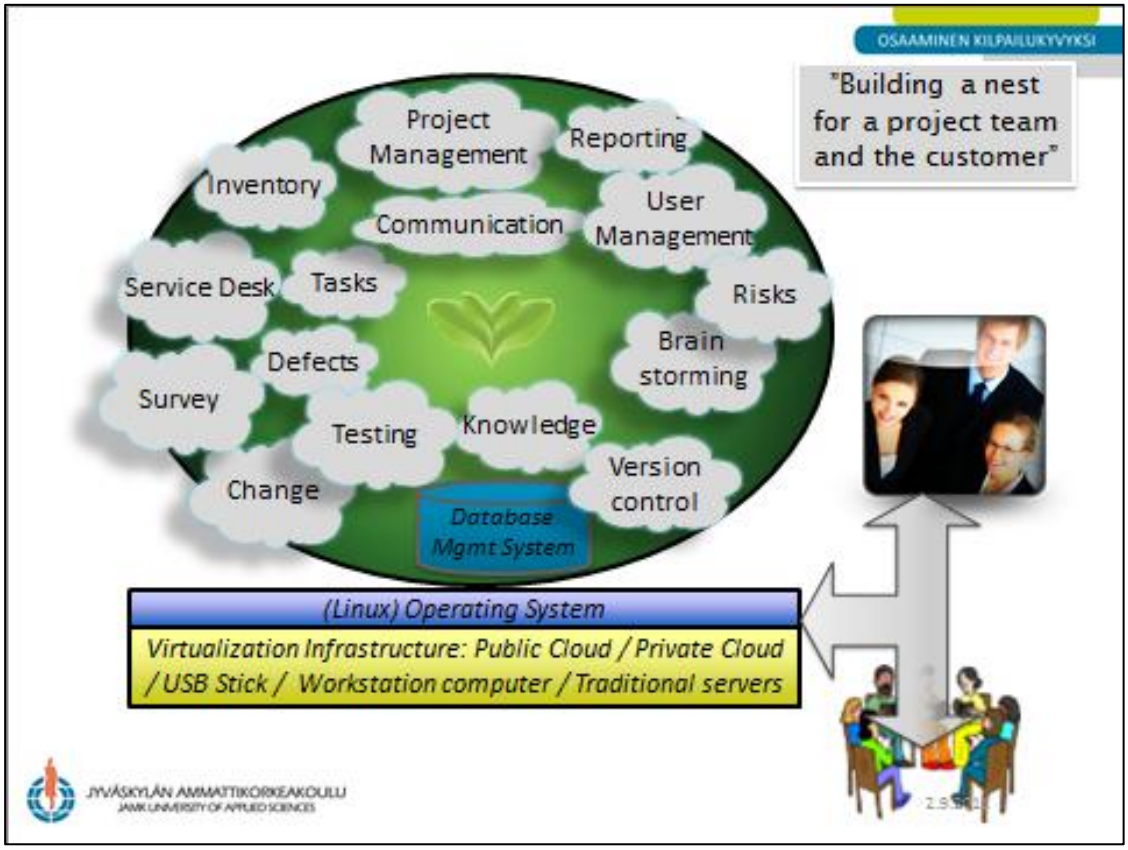

FIGURE 6 - A high-level visualization of the different domains and areas of focus in the FreeNEST platform

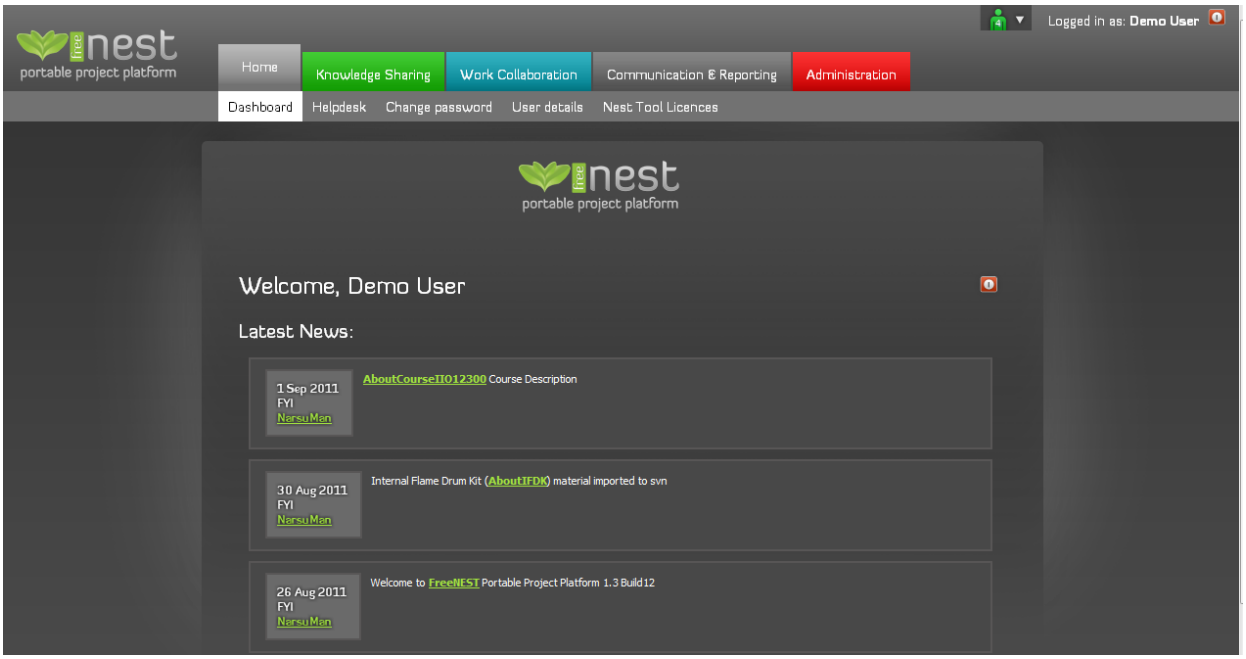

FIGURE 7 - Screenshot of the dashboard and main page of FreeNEST 1.3 build 9. This particular instance houses materials for a software testing course

FreeNEST was originally named Nest Portable Project Platform, but the product was renamed alongside its initial release as a open source project for legal reasons.

The concept behind FreeNEST was originally created by Marko Rintamäki in the beginning of 2000's, while working at Finnish ICT solutions provider corporation. Rintamäki wanted to create a lightweight product development and project management platform that would be based on Open Source products, instead of the more popular closed-source and heavy-weight Enteprise solutions provided by large players in the field, such as IBM Rational software. (Rintamäki, Conversations 2009-2011) This platform originally came to be know as Nestix, after Ixonos, the ICT solutions and subcontracting company in Finland. This proto-version of the platform incorporated all the elements of the modern system, namely integrations between different web-based open-source applications all housed in one VMware based virtual machine. Compared to the modern version, multiple components were different and the underlying operating system was RedHat Linux instead of Ubuntu. The Nestix platform was used internally, but it was deemed not to be the core competency nor focus of Ixonos, and therefore the development of the product was halted. Rintamäki, however, continued development of the product on his free time.

In 2006 Rintamäki started part-time lecturing at JAMK and created a similar platform to the Nestix to act as a platform for the course students to document their learning and also to store course materials. This new platform was renamed to Nest Project Platform to avoid conflicts and to change the image and brand of the system in to a more open direction. Rintamäki was handed over full IPR rights to the product in 2010 by Ixonos. (Rintamäki, Conversations 2009-2011)

3 DESIGN SCIENCE RESEARCH METHOD

To formulate a solution to the unique problem "what is the most efficient way to deploy an existing virtual machine as a cloud instance?" the need for a scientifically viable research approach came apparent. Design Science Research Method (Henceforth abbreviated as DSRM) was chosen to be the method of choice, because of its established roots in research in engineering. (Peffers, ym. 2008)

The Design Science Research Method is an iterative research method that has been approved by the scientific community to be used as a pragmatic research method for information system's development. (Peffers, ym. 2008)

Peffers et Al. describe the research method to being an iterative method with six distinct artefacts. These artefacts were examined and a model was derived using them to help structure the work done in the research.

3.1 Activity 1: Problem identification and motivation

> *Define the specific research problem and justify the value of a solution. Because the problem definition will be used to develop an artifact that can effectively provide a solution, it may be useful to atomize the problem conceptually so that the solution can capture its complexity. Resources required for this activity include knowledge of the state of the problem and the importance of its solution.* (Peffers, ym. 2008)

The initial activity merits a thorough accumulation of theory to help formulate the research problems and also the viable steps that can be taken to solve it. In the the-

sis context, it is proposed to write proper ground work for the theory behind the problem.

3.2 Activity 2: Define the objectives for a solution

> *Infer the objectives of a solution from the problem definition and knowledge of what is possible and feasible. The objectives can be quantitative, such as terms in which a desirable solution would be better than current ones, or qualitative, such as a description of how a new artifact is expected to support solutions to problems not hitherto addressed. The objectives should be inferred rationally from the problem specification. Resources required for this include knowledge of the state of problems and current solutions, if any, and their efficacy.* (Peffers, ym. 2008)

In an engineering problem domain context, a qualitative definition of objectives is very much preferable, since at the time of writing of this thesis existing solutions to similar problems are scarce at best.

3.3 Activity 3: Design and development

> *Create the artifact. Such artifacts are potentially constructs, models, methods, or instantiations (each defined broadly) or "new properties of technical, social, and/or informational resources". Conceptually, a design research artifact can be any designed object in which a research contribution is embedded in the design. This activity includes determining the artifact's desired functionality and its architecture and then creating the actual artifact.* (Peffers, ym. 2008)

The design and development phase and documentation can be thought of as to being the implementation part of a thesis report. After thorough planning the scheme is executed and documented in a standard report, which is later to be evaluated by a group of tutors and faculty staff.

3.4 Activity 4: Demonstration

> *Demonstrate the use of the artifact to solve one or more instances of the problem. This could involve its use in experimentation, simulation, case study, proof, or other appropriate activity. Resources required for the demonstration include effective knowledge of how to use the artifact to solve the problem.* (Peffers, ym. 2008)

The demonstration in a thesis is done via the thesis report itself, a standardized report that is evaluated by a group of tutors, peers and faculty staff of the assigned university.

3.5 Activity 5: Evaluation

> *Observe and measure how well the artifact supports a solution to the problem. This activity involves comparing the objectives of a solution to actual observed results from use of the artifact in the demonstration. (…) At the end of this activity the researchers can decide whether to iterate back to activity 3 to try to improve the effectiveness of the artifact or to continue on to communication and leave further improvement to subsequent projects.* (Peffers, ym. 2008)

This iterative reflection will be taken into account during the process of the thesis work. These steps will ensure that the quality of work is up to par, and to ensure that unfertile lines of research are cut out of the thesis work early on.

3.6 Activity 6: Communication

"Communicate the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences such as practicing professionals, when appropriate." (Peffers, ym. 2008)

This communication could be thought to include not only the actual thesis work, but also communicating through all of the steps through the problem. A thesis is a way of communicating the importance of the matter at hand, and therefore a vital part of the process. In the general process, the Thesis Presentation and Maturity test can be thought to be included in this activity.

3.7 Derived model visualized

Through these structured activities, a 6-step iterative process for research in information systems can be formed that is also very suitable for tacking problems in the domain of software and software infrastructure engineering. The process is illustrated in **Figure 8**. This model was then utilized in structuring the actual work done from planning out the research questions and later on the executed portion of the work.

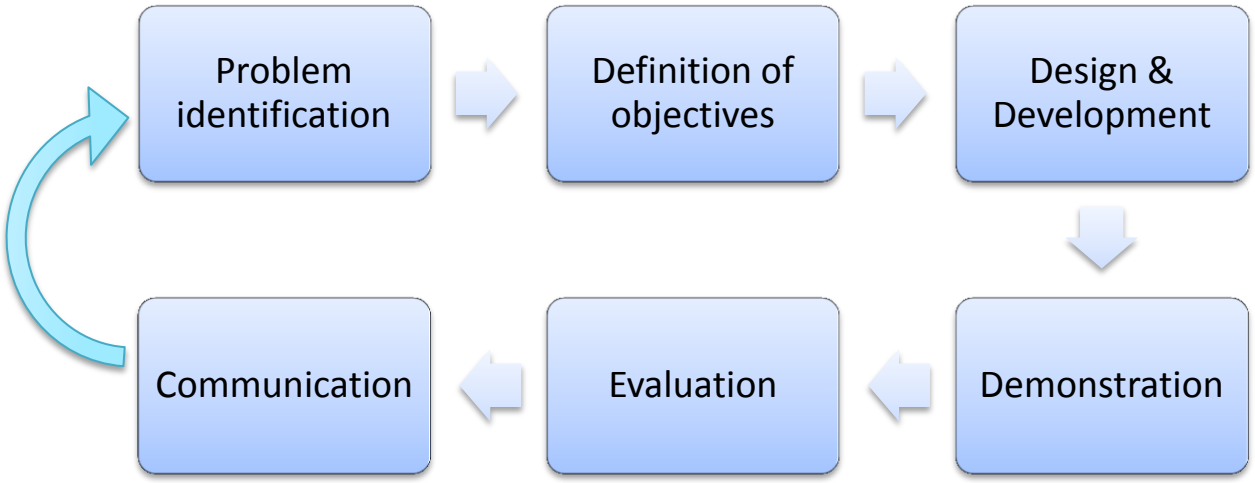

FIGURE 8 - DSRM iterative process for systems development

In practice, this model is implemented by structuring the work around writing as much tutorials of the technical work as possible. Not only does it reduce the strain of writing a tutorial in Thesis form, it also allows for communication of parts of the results already during the work of the thesis, as opposed to waiting for the publication of the document. The other phases are implemented as chapters in the thesis work. By utilizing this model of research, the complex problem domain of the subject of this thesis can be methodically researched and implemented.

4 IMPLEMENTATION WITH OPENSTACK

4.1 Formulation of the research problem

In the formulation phase, it was asserted from numerous sources also presented earlier in this thesis that cloud technology presents itself as a new paradigm shift in the history of industrial computation, as told by industry leaders. This shift merits research in how it can most efficiently be used in a given target organization. According to the initial premise, a model for an organization was chosen to be a large software developing enterprise with over 500 employees and with a focus in research and development of new products. These constraints were then used as a framework for all the decisions made in the progress of the thesis.

Cloud computing and related technologies can then be said to be a truly rising trend. However, the question still remains; if cloud computing is a valuable enough trend for a company to invest in, should the company then outsource their IT department altogether or should they instead focus on creating a modern ICT-infrastructure? Private clouds are a debatable technology. Being based on virtualization, it could be said that any ICT infrastructure that is remotely based on virtualization can be called a private cloud.
For the purpose of this thesis, it was chosen to assert that in the context of the word, traditional ICT infrastructure would not cut it. Instead, the focus of the work would be in researching OpenStack as a private cloud solution. OpenStack was chosen due to the popularity of the project, and also due to its open source nature

Also, if we look at a company that already has invested in the development of its infrastructure, previous services and solutions based on virtualization are likely to exist. Virtualization has been around as a major technology since the beginning of the 21st century, thus previous solutions are very likely to exist, since they have previously been shown to benefit organizations. In the case of this thesis, the specific target of this generalized question will be the FreeNEST Project Platform, which fulfills the characteristics of the generalized question.

From these starting points, these research questions were then formulated:

1. *How can you build a private cloud solution with the aid of OpenStack and is it a viable option for production use?*
2. *Can existing virtual machines be run on an OpenStack-based infrastructure as instances?*

4.2 Definition of actual objectives

After establishing the research questions, more concrete milestones were defined to help measure the progress of work. Based on the established research questions, the following milestones were then established:

1. *Design and implement a test server hardware setup for running OpenStack properly.*
2. *Installation of a single-node installation of OpenStack and completion of related documentation.*
3. *Design and deploy a virtual machine on the single node installation.*
4. *Scale up the experiment to a multi-node installation by adding additional hardware to the setup.*
5. *Deployment of an existing virtual machine as an instance, with written documentation.*
6. *Determining if running existing virtual machines on OpenStack powered infrastructure is a viable option by utilizing FreeNEST Project Platform as a reference.*

4.3 Development phase

The development process to answer the research questions started out by purchasing the adequate hardware for creating the previously specified testing infrastructure. The test setup was determined to be adequate at two machines and a network between them. The hardware was deliberately chosen to be of a desktop specification as opposed to a more common rack-mounted server, due to not only limitations in the budget available, but also due to the rising trend with big companies of creating computational clusters out of exhaustible hardware. The chosen specification of the computer was still high-end on a workstation scale, but only cost approximately 1000€ per unit, as opposed to approximately 10 000€ or more for a rack-mounted server. The complete specification of the server computers are listed on **table 1.**

TABLE 1 – Hardware specifications of the two node computers

Component	Description
Make and model of computer	HP 8100 Pro Midi
Processor	Intel Core i7 2600
RAM	16GB DDR3
Hard disks	2x 1TB with Raid 0+1
Networking	1Gbit Ethernet
GPU	Integrated Intel circuit on motherboard

The operating system chosen for the individual computers was Ubuntu Linux 10.04 Long Term Support version. Ubuntu Linux was chosen to act as the operating system first and foremost because it is the primary operating system OpenStack supports with its distribution system. Other alternatives such as Redhat or Debian were considered, but Ubuntu was chosen also due to the fact that the FreeNEST platform is also based on it. Previous experiences in configuration and system folder locations could then most smoothly be transferred to the new setup.

4.3.1 Installation of a single node network

Setting up the computers and a network between them was the first task. The computers were isolated in their own sub-network (henceforth abbreviated subnet). The subnet had access to WAN via a router which had Network Address Translation active, allowing updates to be fetched from open repositories from the internet, such as the OpenStack nova repository. Installation to this point was done relatively fast, without any outstanding problems. The network adapter interfaces (eth0 in both cases) on the physical computers were configured to have static addresses to simplify the network topology and because the OpenStack Nova configuration file also required permanent addresses to create the nova-node network, when utilizing the chosen network manager (a decision that will be explored later on in this chapter).

Installation of OpenStack took a longer time than originally anticipated. Issues were mostly faced due to the stable release called ‘Bexar’ being not so stable, not only in

the code but also on the documentation side. Installation attempts of the Bexar release took more time than anticipated. Initially, the greatest problem was due to the documentation of the nova project being lacking. One instance of such lacking was in the installation tutorials found on the official project site, which erroneously pointed towards the unstable developer repository, instead of the intended stable release branch. The primary outcome of trying to install the application according to the instructions was the application failing to start after installation. Other problems were also encountered. Mostly they arose either during the installation phase due the application crashing due to internal configuration errors, or from the network configurations. After a few weeks of trying to debug the nova project error log, which unfortunately would become a familiar sight for months to come, the error in the documentation was found. When finally pointed to the right repository for the project data, the installation process itself was fairly straightforward. Still, the project being a very active one, one could encounter changing errors on a day-to-day basis, encountering one error one day and another one the next. In between the developers of the project would release fixes for bugs found at that point, but unfortunately also create new ones in the process. This problem became less prevalent as time went by.

After a few weeks of trying, the OpenStack nova installation on a single node was complete. After consulting the documentation, deployments of initial virtual machine instances were done all according to the plan. Again, some errors were found during the deployment process. Sometimes the nova binaries would fail to launch an instance, only logging an error message known as “Unknown error”, which of course did not have any solutions present at any know documentation or support sources. Part of the problem also arose from the fact that OpenStack uses the Eucalyptus project’s tool application “euca2ools” to do all actions related to instance administration and compute network administration. This became a problem when searching for solutions regarding to instance spawning failures for OpenStack, and the only solutions available were for Eucalyptus’ corresponding. This interoperability of tools in this case is achieved due to compatibility between both the Eucalyptus API and OpenStack API. After a month of test deployments, multiple debugging of logs and

sometimes just plainly waiting for updates to roll in, the first successful deployments of Instances were complete.

4.3.2 Scaling up to a multi-node installation

At this point, the experiment was scaled up. Adding a new computer to the infrastructure node network was a relatively easy task, due to the fact that any new nodes in the computational network can be installed in very much the same way as on the first installation, with the exception that instead of accepting the default options during the process, the user must specify the IP address of the first machine installed. This effectively makes this first machine the API node of the network. The OpenStack nova application can then automatically report itself as a slave node to the API, which automatically then logs in information about the specifications (such as type of CPU, amount of RAM, etc.) of the newly joined slave. This exercise can then be repeated as many times as is wanted to join the network, and can even be automated to occur during networked installation of the operating system.

OpenStack offers 3 different network managers to create the node network. Of these, initially the FlatManager was chosen. The FlatNetworkManager allows for a computational node network to occupy a normal local area network, without the creation of any new sub-networks nor virtual networks. OpenStack does not recommend its use outside of test environments, instead of opting to recommend the usage of VLANManager for maximum safety and isolation. However, the router used for the test setup purposes did not support VLAN headers on packets, and therefore was uncapable of creating and supporting virtual networks. It was then decided, that the network manager of choice would be FlatManager. This decision however turned out to be the wrong one. The FlatManager is enabled by simply typing it in the OpenStack nova configuration file, nova.conf. A full print of this file can be found **in Appendix 1**. The Flat manager does not require any special interfaces to be created in the Linux network adapter configuration files, therefore it was deemed the simplest to start with. The only requirement would be to specify the correct IP address of the API node in the nova configuration file or installer. However, this approach did not

work. Even though correct addresses were typed in, the second node could not report itself to the API node. After a few weeks of debugging logs and looking for answers at the official OpenStack Launchpad site, this approach was deemed a dead end.

After this approach, the only option left was to try to create the network using the DHCP Manager. It required the creation of new bridged network adapters to the host operating system. This creation is done by simply editing the interface's configuration file in the Linux system files folder. Initial concerns were that the bridge specifications would collide with the static network addresses of the original network. These worries were, however, quickly proven wrong. After re-installing the whole network, first on the API machine and then on the slave machine, using DHCP Manager from the get-go, the creation of the network worked on the first go. The creation process of the bridge is documented again in **Appendix 1**. After the successful creation of the node network, template instances could be launched using the tools in the API machine. The machine would then chart an available spot on either the API machine or the computational node and launch the instance there. The only limitation, however, was that template images for the instances needed to be registered using the target machine or node, meaning that one had to physically register the image using API tools on the machine they were supposed to run. Automatic instance migration was at this point of time a feature still in the making.

4.3.3 Converting an existing virtual machine to an instance

Finally, when the initial network was completed, research into converting old virtual machines to OpenStack instances could begin. At this point of time, not many pieces of documentation or written experiences existed in converting old virtual machines to instance machines. The primary difference between a regular virtual machine and a cloud instance is that instances are split up into multiple components and they communicate with the computational network via the same OpenStack API as the underlying physical nodes themselves. It was also found out that instances are usually also split up into multiple parts. Normal virtual machines contain all aspects of a

computer in one package. Instances however can have separate hard disk images, ramdisk images, and kernel images. This approach has advantages when using highly standardized and bare-bones (containing only a clean operating system installation) images as templates. The end user can then choose from a wide variety of template combinations in creating the perfect platform for their application(s). This is not a necessary feature with existing virtual machines, however, since most of them have already been installed and deployed into use with their respective machine specifications, including versions and configurations. Therefore, if the system forces separation of these elements, deployment can become a cumbersome and lengthy process. Initially, the only instructions available anywhere (mostly the internet) were for creating new virtual machines and then customizing them. In the absence of instructions in converting existing machines, this method was chosen to be the focus of testing first. If there were no ways of actually converting existing machines, perhaps one could customize a new one using the list of installed packages from the old machines.

To customize a newly deployed instance, most tutorials and instructions guided the user to create a new machine and then transferring a pre-configuration script to install required packages on the first boot. The other option provided was to handle the installation of customized packages via a **chroot**-command trick. This method requires the creation of a new virtual machine using existing Linux tools, mounting the newly created and installed machine as a part of the real machine's file system, and then changing the root directory of the host operating system to the root directory of the mounted image. Technically, this would allow installation of packages e.g. using a package manager like apt-get that resides on the host machines, on the mounted (and unbooted) guest machine. In real life usage scenarios, however, this rarely works. The host package manager might use the host machine's package database as a starting point in determining whether dependencies need to be installed, and also very usually install packages on the host machine, instead of the guest. This was practically tested out. A list of installed packages could easily be obtained from the existing FreeNEST virtual machine via a simple package manager command. The obtained list was then transferred to the host machine and from there to a newly created test machine, made according to tutorials on the OpenStack site. This ap-

proach did not bear fruit, since as mentioned above, some of the packages or their dependencies did not install properly. This method would have also been a challenge in terms of configuration management, each upgrade of the original virtual machine requiring the creation of new templates and package lists. Even if one managed to install some of the packages and get the machine to boot as an instance on top of the infrastructure network, problems with misconfigurations of the packages occurred more often than not.

Towards the end of the work in this thesis, it was concluded that a conversion of existing machines was not simply possible to achieve. What was lacking was a piece of software that allowed the instance machine to communicate with the host network, and no package was located that could help with this issue. Writing new software for this purpose did not come into question due to limitations in time and resources.

Then, just when this conclusion had been reached and tested out, and other alternative ways of doing the conversion had been explored and exhausted, a small gust of renewed hope blew over the project. First, a new tutorial describing a slightly different way of managing instances was found at the end of April. (CSS Corp Open Source Services 2011) What was exceptional about the method described was that it instructed in creating firstly an ordinary virtual machine, and then guided in installing two packages on it that had not previously been described anywhere. One of these packages, called ***cloud-init*** turned out to be the breakthrough needed to complete the conversion process. It is described in the official Launchpad page in the following way: *"Package provides configuration and customization of cloud instance"* and a quick overview of the history showed that the package was originally named ***ec2-init***. (Canonical ltd ei pvm). Then, by applying some of these methods to the existing virtual machine, instead of creating a new machine, the desired outcome was reached. The FreeNEST instance was converted from a regular VMware virtual machine to be a cloud-compatible instance. This method also retained every configuration file necessary for the operation of the FreeNEST platform. It essentially allows for a "quick and dirty" way of making the same machine available on both traditional virtualization methods, and the cloud. As an additional bonus, the FreeNEST instance can be run

on any EC2-compatible infrastructure thanks to the cloud-init package, since communication with the underlying infrastructure abides mostly by the same principles and calls.

In the end, as a final coup de grâce, an explanation of the arbitrary appearance of the package was unfolded; halfway through May 2011, Canonical ltd. announced that the Ubuntu distribution would favor OpenStack in future releases, as the Ubuntu Enterprise Cloud infrastructure of choice (BusinessWire 2011). The package had been renamed in the process preparing for the transition, making it more widely usable for all compatible infrastructure services.

5 RESULTS

The first presentable result of the work documented in this thesis is the conclusion that OpenStack is still very much a work in progress. The state of the documentation of the various projects, not only the observed nova project, is very lacking. Especially, if the use cases deviate in any way from the very basic ones that are discussed in the documentation, the user is left to fend for themselves in looking for answers. This process for looking up solutions can then be a very time consuming task. OpenStack is a very new project, only a bit more than a year old, and it is progressing very rapidly. Therefore the state of documentation can easily be left behind, and the end user needs to keep track of a wide variety of different developer forums, official and unofficial, to decrypt often very trivial and uninformative error messages. The situation however is improving. Even though rapid updates also bring with themselves new bugs, they usually fix more issues than break new things. The state of the documentation is something OpenStack is improving almost constantly. At the beginning of the work, many foxholes could only be avoided by accidentally tripping in them and watching out for them in the future. The currently live release, Diablo, already has improved the documentation site by leaps and bounds. The OpenStack community is also heavily investing in documentation, sometimes holding special events to fix errors in documentation (Gentle 2011). However, as OpenStack is now, it cannot be

deemed viable for production deployments, at least if the development has to take place on a tight schedule. OpenStack, however, is great for a testing environment, even if the final production environment is a public Amazon cloud, providing many of the same functions and a similar environment. After one can manage the installation process of OpenStack, it is relatively rapid to deploy, given the underlying hardware specifications of the machines allow it. It must also be noted that this situation could change in less than a year. OpenStack has such massive numbers of developers and such big partners backing it up, that these limiting factors will undoubtedly be fixed in the coming releases. If one were to start planning a product ready for launch at the second half of 2012, OpenStack would already be a viable technology to consider using.

One concrete result of this work is a tutorial written based on the knowledge obtained in the installation process. It incorporates many known elements from existing installation instructions, but also includes readouts of all necessary configuration files as examples, so that the readers can see what actually working configurations look like. The tutorial has afterwards been used as a baseline in training students during courses and experts alike on the installation of OpenStack on multiple different occasions. The tutorial has also been shared to the general usage in the Cloud Software Program, helping other experts in avoiding the errors and blights that the writer of this thesis had to overcome. The full tutorial has been enclosed in this work as **Appendix 1**. In addition, a second tutorial, also shared to students and experts at the Cloud Software Program, was written on the instance conversion process. It again incorporates many elements of existing tutorials, but gives context by actually describing a viable method of converting existing machines. This tutorial has been enclosed to this work as **Appendix 2.**

The most essential result of the work is the actual demonstration environment. It proved that one can build a fully viable cloud infrastructure using only open source tools and normal desktop hardware. An illustration of this final implementation is presented in **figure 9**.

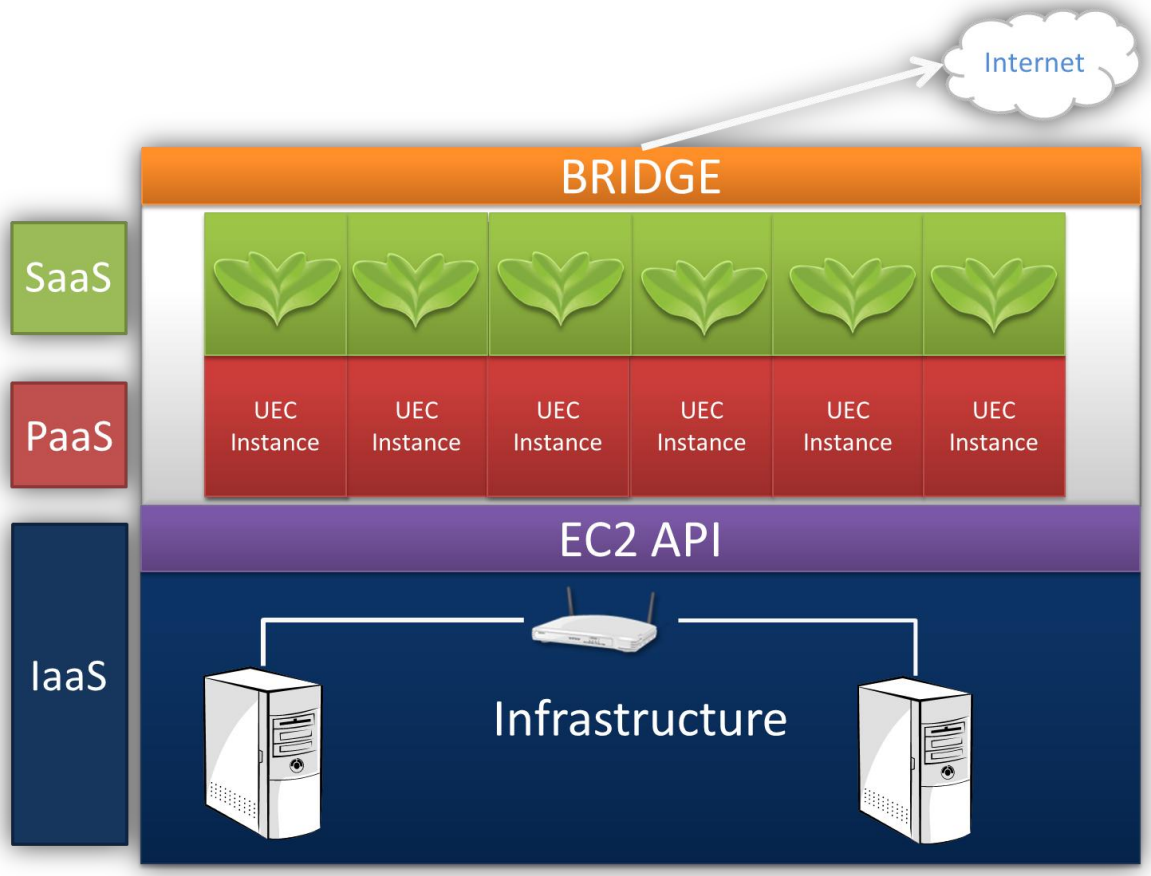

Figure 9 - Visualization of the final implementation of the test infrastructure and overlaying layers of software

The final result is the answer to the second research question: although it was deemed possible to convert a regular virtual machine to a cloud-supported instance, it might not be a very worthwhile thing to do in the long run. The conversion, even at its simplest form can take multiple steps and has plenty of room for error. Even if conversion is successful, the requirements for running the instance differ greatly from regular instances. In most cases, the instance must be deployed with an extra-large specification meaning a great deal of RAM allocation per instance. Light performance testing of the FreeNEST instance revealed, that although the instance actually did boot up, it did not run very fast when subjected to CPU intensive tasks, such as rendering python web pages through a web server. Therefore, it must be concluded that even though converting existing machines is possible, it is advisable to retain from doing so.

6 DISCUSSION AND RAMIFICATIONS

It was deduced that while one can use relatively cheap and old regular workstation computers as compute workers, there are still certain requirements for the infrastructure cloud to function properly. This issue is can be most prominent with older machines with only dual core processors, which rapidly run out of resources and threads to run virtual machines. OpenStack can be installed on older machines, as long as they support virtualization extensions like the Intel VT-X, but ratio of machines available in the network to the capacity to run instances increases very much. More modern machines are usually closer to the specification of the demo machines. OpenStack specifies in a community presentation that a proper size production environment should at least consist of 40+ machines (B. Piatt 2010). Therefore, in the future the experiment should be scaled up more considerably. By the time of writing, inside the JAMK University of Applied Sciences, it already has. The work presented in this thesis has already been used as a component in creating two later iterations of the demo setup. The first one was called the classroom cloud, and it was created by a group of students during the summer of 2011. Both the appendices presented with this thesis were given to the students to work with, and they allowed the students to install OpenStack on a classroom of computers, on about 15 machines in one third of the time it took the writer of this thesis to initially do so on just one machine. This illustrates the power of documentation and the necessity of digging into uncharted territory. The second cloud, currently still under construction is a direct descendant of the classroom cloud, called the “Junk Cloud”. Again, actual installation of this hardware setup was done by a group of students utilizing techniques learnt from the appendices. The hardware setup is aptly named, because it currently consists of computers that would have otherwise been removed from usage and scrapped altogether. It represents a worst case scenario of a cloud infrastructure, something that a small enterprise might consider building. A generalized architecture and the future vision of the “Junk Cloud” in presented in **figure 10**.

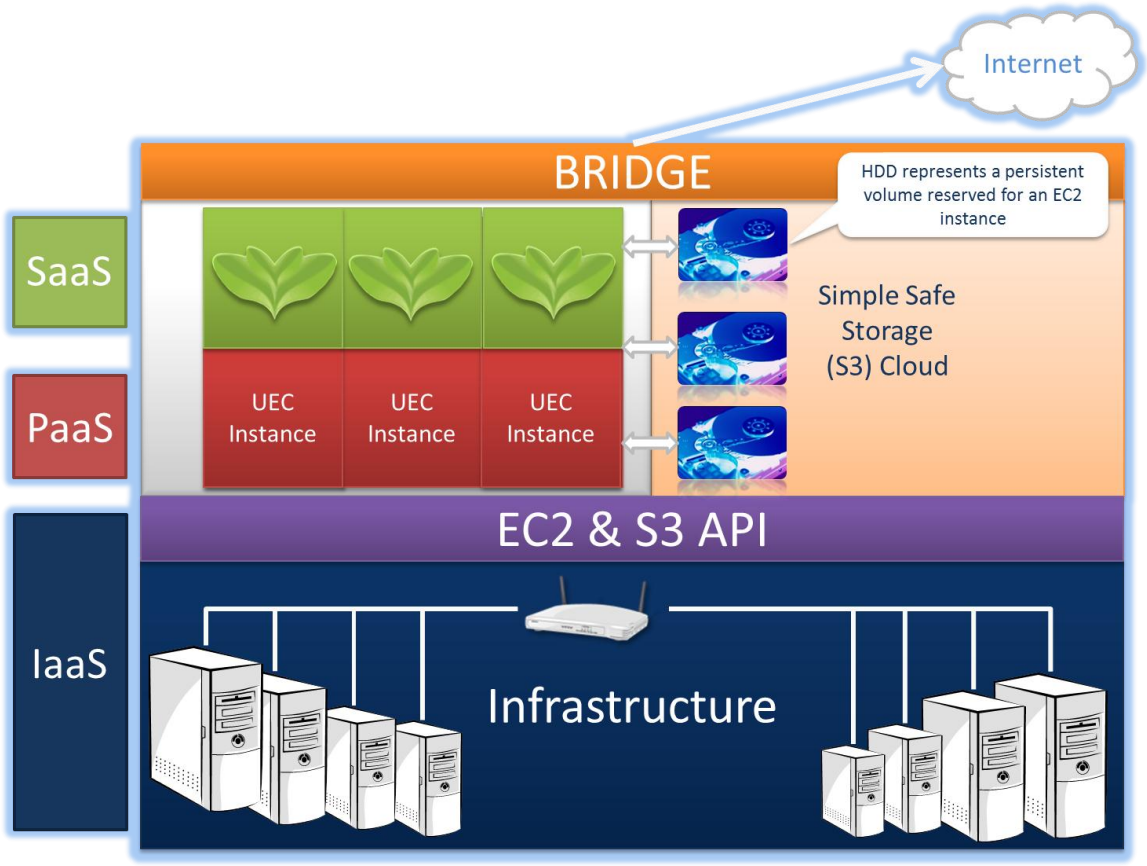

Figure 10 - The future vision of "junk Cloud", an infrastructure capable of offering both S3 and EC2-capabilities

This exercise of thought opens a window to a wide variety of new usage scenarios that might be worthwhile to explore in the future. One such scenario would be called the Office cloud. Basically this means that one could build a virtualization cloud that would run on workstation machines in an office environment. These computers could be used as regular workstations during the day and as an Infrastructure layer during the night, running complex tasks utilizing virtualization, including but not limited to building code, automated testing, batch processing using Hadoop or similar technologies, or even philanthropic software such as Folding@home. It could help to optimize computer capacity and usage of workstations that most likely lie dormant during most of the night, maybe even help the companies to vend excess computational capacity to the highest bidder. On the other hand, the increased electricity bill might be enough of a reason to forget this train of thought, not to mention the rapid swift of focus in modern working environments to irregular working hours.

One of the key ramifications is proposing a shift of focus in software architecture, when it comes to designing software that runs on top of the cloud. The computa-

tional cloud paradigm brings with itself a shift of thought from fitting every function of a piece of software in one virtual server. An EC2-derivative cloud excels when it comes to parallel processing with multiple instances. It is suggested that when designing software for an EC2-derivative cloud service, the developers focus on creating a multi-instanced business logic component for their application, meaning that they design modular software that is spread out across multiple virtual machines. This in turn might even help creating more reliable software, since splitting functionality between instances not allows for limiting damages in case of failure, but also quick and easy recovery from these error situations. By creating template machines in their infrastructure's VM Bucket, a replacement server in case of a crash is but one instance booting away. It also allows for rolling updates on part. In addition, with cloud computing also a need to separate static code from changing binary data emerges. Since EC2 instances do not have persistent memories, developers need to allocate separate block storage units from a file cloud, such as S3, to be attached to the instance. This essentially forces the developer to abide by a strict Model-View-Controller divide of the code, where functionality and data are strictly separated.

Since the completion of the work documented inside this thesis by June 2011, the work has progressed significantly. Not only has the test cloud infrastructure already experienced two new iterations, and the tutorials shared with the general researcher crowd of the Cloud Software Program, the knowledge obtained alongside this thesis has also been used in writing the article **Designing IDE as a Service** (Aho, ym. 2011).

REFERENCES

Aho, Timo, et al. "Designing IDE as a Service." *Communications of Cloud Software*, 2011: www.cloudsw.org.

Association of Computing Machinery. *The reincarnation of virtual machines.* July 1, 2004. http://queue.acm.org/detail.cfm?id=1017000 (accessed 10 23, 2011).

BusinessWire. *Ubuntu project to transition Ubuntu Cloud to OpenStack.* 5 10, 2011. http://www.businesswire.com/news/home/20110510006281/en/Ubuntu-project-transition-Ubuntu-Cloud-OpenStack (accessed 11 11, 2011).

Canonical ltd. *"cloud-init" package in Ubuntu.* n.d. https://launchpad.net/ubuntu/+source/cloud-init (accessed 11 11, 2011).

CloudTweaks. *Wikileaks evades hackers with shift to Amazon EC2 | CloudTweaks.com - The Cloud Computing Community.* November 29, 2010. http://www.cloudtweaks.com/2010/11/wikileaks-evades-hackers-with-shift-to-amazon-ec2/ (accessed April 08, 2011).

CNET. *Oracle's Ellison nails cloud computing | Outside the Lines - CNET News.* September 26, 2008. http://news.cnet.com/8301-13953_3-10052188-80.html (accessed April 07, 2011).

CSS Corp Open Source Services. *[OpenStack Beginner's Guide for Ubuntu 11.04] – Image Management.* 04 27, 2011. http://cssoss.wordpress.com/2011/04/27/openstack-beginners-guide-for-ubuntu-11-04-image-management/ (accessed 11 11, 2011).

Gentle, Anne. *The OpenStack Blog.* 09 12, 2011. http://www.openstack.org/blog/2011/09/openstack-documentation-blitz/ (accessed 11 11, 2011).

Heino, Petteri. ”Pilvipalvelut.” Teoksessa *Pilvipalvelut*, tekijä: Petteri Heino, 32-33. Talentum, 2010.

JAMK University of Applied Sciences. "Jyväskylän Ammattikorkeakoulu vuosikertomus 2010." 2010.

—. *Projektit - Jyväskylän Ammattikorkeakoulu.* 01. January 2011. http://www.jamk.fi/projektit/1233 (haettu 09. April 2011).

Microsoft. *Windows Azure Platform Launch Update - Windows Azure - Site Home - MSDN Blogs.* October 29, 2009. http://blogs.msdn.com/b/windowsazure/archive/2009/10/29/windows-azure-platform-launch-update.aspx (accessed April 08, 2011).

OpenStack. *Home >> OpenStack Open Source Cloud Computing Software.* August 15, 2011. http://www.openstack.org (accessed August 15, 2011).

—. *openstack/swift - GitHub.* 2011. https://github.com/openstack/swift (accessed 11 20, 2011).

—. *System Architecture.* 2011. http://docs.openstack.org/diablo/openstack-compute/admin/content/system-architecture.html (accessed 10 30, 2011).

Peffers, Ken, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. "A Design Science Research Methodology." *Journal of Management Information Systems*, 2008: 45 - 77.

Pepple, Ken. *Ken Pepple.* 04 22, 2011. http://ken.pepple.info/ (accessed 10 30, 2011).

Perälä, Kai. *NEST 1.1 – Avoimen lähdekoodin projektinhallintaratkaisu.* Jyväskylä University, 2008.

Piatt, Bret. "OpenStack Tutorial." *Slideshare.* 12 2010. http://www.slideshare.net/bpiatt/openstack-tutorial (accessed 11 11, 2011).

Piatt, Brett. "Object Management Group." *OpenStack Overview.* 2010. http://www.omg.org/news/meetings/tc/ca-10/special-events/pdf/5-3_Piatt.pdf (accessed 10 24, 2011).

Rackspace. "Openstack - 6 Month Anniversary." *Youtube.* January 17, 2010. http://www.youtube.com/watch?v=qfh-bKRww1U (accessed April 08, 2011).

Rintamäki, Marko, interview by Ilkka Turunen. *Conversations* (2009-2011).

Rintamäki, Marko. *NEST-Verkkotyöympäristön kehittämiskohteiden kartoittaminen opetuskäytön näkökulmasta.* Jyväskylä University, 2008.

Ruottu, Toni, Eemil Lagerspetz, and Sasu Tarkoma. "What is "Cloud"?" *Communications of Cloud Software*, 2011: TBP.

Tivit Oy. *Partners - Cloud Software Program.* April 01, 2011. http://www.cloudsoftwareprogram.org/partners (accessed April 08, 2011).

VentureBeat. "Dell to invest $1B in Data Centers offering cloud services | VentureBeat." *VentureBeat.* April 07, 2011.

http://venturebeat.com/2011/04/07/dell-to-invest-1b-in-data-centers-offering-cloud-services/ (accessed April 08, 2011).

VMware. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist.* Palo Alto: VMware, 2007.

APPENDIX 1. TUTORIAL ON HOW TO INSTALL OPEN-STACK SINGLE NODE

About this How To

This topic is a tutorial on how to install OpenStack Nova, a distributed cloud computing system on any ubuntu 10.x machine. This private cloud can then be used to run virtual machines that can be launched and stopped with a single command. The tutorial uses OpenStack Bexar release. Most of this tutorial is based on OpenStack's documentation found at http://docs.openstack.org, but has additions where ever applicable.

The tutorial also assumes that you are using a blank image of ubuntu 10.10 Desktop, with no MySQL installed. Also, please note that this needs to be done on a machine, that has Linux natively (i.e. not in a virtual machine). This is because you need to have bios level support for virtualization, since KVM is the virtualization system of choice in Nova. Openstack documentation specifies that it should also run on Virtualbox, so if you don't have a physical machine to spare, try that.

By default, the Cloud VM's will not have internet access, but I've added our own configuration files to show how we managed to get them to see the outside network.

Description

OpenStack is a system that allows you to create cloud platforms on regular computers and then later add many "nodes" to your private cloud. There are multiple ways of installing it, but this document only describes how to install the whole setup in one machine and make it run Ubuntu Enterprise Cloud based virtual machines.

Terms to know

IAAS - *Infrastructure as a Service. OpenStack is a solution of implementing this layer.*

UEC - *Ubuntu Enterprise Cloud - A derivative of Ubuntu server, can be run on a IAAS engine*

EC2 - *Amazon EC2 - Elastic Compute Cloud - A "de facto"-standard that is based on mimicing the APIs and functionality of Amazon's cloud offerings*

Image - A physical virtual machine image. Openstack and other EC2-compliant VMs are usually split into three different parts: The machine image (hdd), a kernel image, and a ramdisk image. They are usually packaged in a tar.gz but can also be deployed separately.

1. Install pre-requisite for OpenStack:

```
sudo apt-get install python-software-properties
```

2. Add OpenStack Bexar release repository to your aptitude:

```
sudo add-apt-repository ppa:nova-core/release
```

After this, update your apt-get repositories:

```
sudo apt-get update
```

The machine will attempt to get and verify a public key from the repository. If your network policies do not allow for the key to be downloaded, don't worry. It just means that you'll have to manually press yes to confirm the installation of the nova packages, since the OS won't trust the source by default.

3. Install RabbitMq-server:

```
sudo apt-get install -y rabbitmq-server
```

RabbitMQ acts as the messaging queue for the cloud. It allows the API-server to coordinate the different computational nodes' tasks correctly.

4. Install other python-prerequisites:

```
sudo apt-get install -y python-greenlet python-mysqldb
```

Nova won't install without these, so install them first

5. Install Nova-packages

After we are done installing the pre-requisites, we can now move on to installing the actual nova-packages which act as the backbone and brains of the cloud:

```
sudo apt-get install nova-common nova-doc python-nova nova-api nova-network
nova-objectstore nova-scheduler nova-compute
```

This step will announce that the packages are not trusted, if the machine could not verify repository keys. It asks you twice to confirm that you want to install them, this is normal. ***Answer yes***.

6. Install euca2ools and unzip

OpenStack needs to be administered somehow. Euca2ools are actually Eucalyptuses tools, but openstack also uses them to launch, administer, etc. the virtual machines.

```
sudo apt-get install -y euca2ools unzip
```

7. Switch to root user (if not already)

The next steps are easiest to execute if you do them as root. To switch to root run:

```
sudo -i
```

If you are already logged in as root, you can skip this step.

8. Install MySQL

Next, we need to install a database for nova to use, and create proper users for said databases. Use the following syntax (as root) to ***seed the installation*** with proper values, so you don't need to set them by hand during the installation with dialogs:

```
bash

MYSQL_PASS=nova

cat <<MYSQL_PRESEED | debconf-set-selections

mysql-server-5.1 mysql-server/root_password password $MYSQL_PASS

mysql-server-5.1 mysql-server/root_password_again password $MYSQL_PASS

mysql-server-5.1 mysql-server/start_on_boot boolean true

MYSQL_PRESEED
```

Then just apt-get mysql:

```
sudo apt-get install -y mysql-server
```

Finally, when the installation is done, edit the mysql configuration file to allow database connections from any ip:

```
sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf

sudo service mysql restart
```

9. Configure MySQL

Create databases used by nova with the following:

```
sudo mysql -uroot -p$MYSQL_PASS -e 'CREATE DATABASE nova;'



sudo mysql -uroot -p$MYSQL_PASS -e "GRANT ALL PRIVILEGES ON *.* TO

                                'root'@'%' WITH GRANT OPTION;"



sudo mysql -uroot -p$MYSQL_PASS -e "SET PASSWORD FOR 'root'@'%' =

                                PASSWORD('$MYSQL_PASS');"
```

This creates the databases and grants the root user access rights and adjacent privileges.

10. Create a network bridge

OpenStack defaults to a bridge called br100, so you need to create one. This is done by editing a file in /etc/network/interfaces

```
< begin /etc/network/interfaces >

# The loopback network interface

auto lo

iface lo inet loopback



# Networking for OpenStack Compute
```

```
auto br100



iface br100 inet dhcp

    bridge_ports         eth0

    bridge_stp            off

    bridge_maxwait   0

    bridge_fd             0

< end /etc/network/interfaces >
```

This is my interfaces configuration file for our network which has static ip's:

```
# This file describes the network interfaces available on your system# and how to activate them. For more information, see interfaces(5).

# The loopback network interfaceauto loiface lo inet loopback

# The primary network interface

auto br100

iface br100 inet static

address 192.168.8.20

network 192.168.8.0

netmask 255.255.255.0

broadcast 192.168.8.255

gateway 192.168.8.254

dns 192.168.8.254

bridge_ports eth0

bridge_fd 0

bridge_hello 2

bridge_maxage 12
```

```
bridge_stp offd
```

Restart your networking for the change to come into effect:

```
sudo /etc/init.d/networking restart
```

11. Restart all relevant services

```
libvirtd restart; restart nova-network; restart nova-compute;

restart nova-api; restart nova-objectstore; restart nova-scheduler
```

Restart all the services just in case.

12. Edit nova.conf

Nova ships with a barebone conf file that needs to be augmented with the correct information. Below is an example how to get it running in one machine:

First, open the conf file:

```
sudo nano /etc/nova/nova.conf
```

Then edit the file to look like this, replace the ip with your computer's ip address:

```
--dhcpbridge_flagfile=/etc/nova/nova.conf

--dhcpbridge=/usr/bin/nova-dhcpbridge

--logdir=/var/log/nova

--state_path=/var/lib/nova

--lock_path=/var/lock/nova

--verbose

--s3_host=184.106.239.134

--rabbit_host=184.106.239.134

--cc_host=184.106.239.134
```

```
--ec2_url=http://184.106.239.134:8773/services/Cloud

--fixed_range=10.0.0.0/12

--network_size=8

--FAKE_subdomain=ec2

--routing_source_ip=184.106.239.134

--sql_connection=mysql://root:nova@184.106.239.134/nova
```

Or if you want to have the computers in the same network as the host computer, here's my conf that helped me achieve this:

```
--dhcpbridge_flagfile=/etc/nova/nova.conf

--dhcpbridge=/usr/bin/nova-dhcpbridge

--logdir=/var/log/nova

--state_path=/var/lib/nova

--verbose

--s3_host=192.168.8.20

--rabbit_host=192.168.8.20

--cc_host=192.168.8.20

--ec2_url=http://192.168.8.20:8773/services/Cloud

--fixed_range=10.0.0.0/12

--network_size=8

--FAKE_subdomain=ec2

--routing_source_ip=192.168.8.20

--sql_connection=mysql://root:adminuser@192.168.8.20/nova

--network_manager=nova.network.manager.FlatDHCPManager

--flat_network_dhcp_start=10.0.0.2

--flat_injected=False
```

```
--public_interface=eth0
```

Remember to replace the ip to your host computer's ip. Openstack basically comes with three different Network managers, a Flat (no management just a static subnet), a Flat DHCP manager (DHCP-based subnet, no management), and finally a VLAN-manager (default option, creates a VLAN for node communication and virtual machines. Needs a physical switch that understands VLAN-flags)

13. Create new user groups for nova

You need grant ownership of the nova configuration file to a usergroup named nova, so we create it:

```
sudo addgroup nova


chown -R root:nova /etc/nova


chmod 644 /etc/nova/nova.conf
```

14. Sync nova Db (as root)

Now if all of the configurations are correct, try to sync your database with the following command:

```
nova-manage db sync
```

if this creates errors, it means that the program could not connect to your mysql database. Check your configuration. This step fills the database you seeded earlier with necessary data for nova usage.

15. Create new user and project

It is time to create a new user and project, where to associate your cloud virtual machines.

```
nova-manage user admin dub

nova-manage project create dubproject dub
```

you can substitute dub or dubproject wth your own user names if you want. Dub is just an example.

16. Create new network

You must create a virtual network space where the virtual machines run. This is done by:

```
nova-manage network create 192.168.0.0/24 1 255
```

The nova-manage service assumes that the first IP address is your network (like 192.168.0.0), that the 2nd IP is your gateway (192.168.0.1), and that the broadcast is the very last IP in the range you defined (192.168.0.255). If this is not the case you will need to manually edit the sql db ‘networks’ table.o.

17. Create certifications

To access your virtual machines, you need to create a set of certifications for your "dub"-user. This is done by creating a zip-file with nova's internal tools (still as root):

```
mkdir -p /root/creds
```

This creates a directory to put your credentials in. Then run:

```
nova-manage project zipfile dubproject dub /root/creds/novacreds.zip
```

Finally, unzip the credentials:

```
unzip /root/creds/novacreds.zip -d /root/creds/
```

18. Reboot

Now it is time to reboot your computer:

```
sudo reboot
```

After the reboot is done, log in again as admin:

```
sudo -i
```

19. Source global variables

The cloud framework detects your access rights with public keys, that have to be put in your global variables. This is simply achieved by running:

```
cd /root/creds

. novarc
```

Every time you want to use and administer your cloud, you need to source the novarc-file first, so remember this step.

20. Get a UEC-cloud virtual image

Finally we get to the good stuff! wget a cloud UEC images for the testing purposes:

```
wget http://uec-images.ubuntu.com/releases/10.04/release/ubuntu-10.04-server-uec-amd64.tar.gz
```

or

```
wget http://uec-images.ubuntu.com/releases/10.10/release/ubuntu-10.10-server-uec-amd64.tar.gz
```

This image contains a machine image that will be deployed to the cloud. Inside the tar.gz there are a hdd-image, kernel-image and a ramdisk image. This is the complete machine.

21. Publish tarball to the cloud api

The next step is to deploy the tarball to the cloud API, so you can start up your instances of said image. This is done by running:

```
uec-publish-tarball ubuntu-10.04-server-uec-amd64.tar.gz mybucket
```

mybucket is just the "bucket" where you store your cloud machine images. With this, the computer will start to untar the file, and it'll take some time. At this point my suggestion is to grab a cup of coffee. In the end it will print out three sets of characters, ami, eki and a third one. They are the id numbers of the computer image. The one you need to remember is the ami.

"Checking bucket: XXXX failed to upload kernel" <<-- error is related to novarc configuration. fix it to correspond nova.conf"

euca-"command" "UnknownError: An unknown error has occurred. Please try your request again." <- check your nova-api.log and if there is "[errno 111] and error from glance.py, then install glance and start it with "glance-control glance-api start".. (will become issue in future releases and now in trunk-release)

22.Create new keypairs

You also need to create a set of rsa keys to be used in the authentication when you start up your virtual machine. This is achieved by running the following command (still being root):

```
euca-add-keypair mykey > mykey.priv

chmod 600 mykey.priv
```

This create a private-public keypair, which will be used when starting up an instance.

23. Start an instance (the virtual machine)

Ok, now we are ready to boot up a VM! You do it simply by running:

```
euca-run-instances ami-g06qbntt -k mykey -t m1.tiny
```

where the ami-hash is the ami-number you got earlier when publishing your tarball. Mykey is the key file used in the authentication of your machine. Finally the -t flag specifies the machine spec.

When you run the command, you will get a response that looks like this:

```
RESERVATION     r-0at28z12      IRT

INSTANCE        i-1b0bh8n       ami-g06qbntt    10.0.0.3        10.0.0.3
scheduling      mykey (IRT, None)       m1.tiny 2010-10-18 19:02:10.443599
```

This tells you that everything went okay. This tells you the instance id-hasd (the i-1b0bhn8n in this example), the ip the machine has been assigned, it's status, and finally the associated key and bucket it has been deployed from.

After you have told the cloud to start up an instance, you will need to wait until the machine has started up properly before you can connect to it. It can take a while, and to check this status, please run:

```
euca-describe-instances
```

You will see a line like above when you published the machine. Instead of scheduling, it will say either pending, starting or running. If the machine gets stuck in pending for extended periods (over 15m) of time, it means there's something wrong somewhere. But if it's starting, it should eventually change to running. Keep running the euca-describe-instances command until it does, to know when to move on. When it changes to running, you can move on to the next step.

24. Authorize SSH connections

By default, all ports to the virtual machines are blocked. You need to enable this traffic in their virtual network to be able to interact with the machines. To do this, you must run:

```
euca-authorize -P tcp -p 22 default

euca-authorize -P icmp -t -1:-1 default
```

This opens up port 22, which is used in SSH-communication and port for ICMP-protocol to test network connectivity.

25. SSH to your VM and enjoy!

Finally we are ready to actually make use of the Cloud Machine. Connect to it with ssh by using

```
ssh -i mykey.priv root@10.0.0.3
```

mykey.priv is the file you created earlier with euca-create-keypairs. This is your public key that allows you to connect to the vm. The ip after the @-mark is of course the address to your vm. This address information can later be obtained by running euca-describe-instances.

26. Destroy an instance

After you are done playing around with your instance, you will need to remove it. This is done by running

```
euca-terminate-instances i-1b0bh8n
```

where the i-hash is yet again obtained by checking the id of the wanted vm with euca-describe-instances. The machine has now been destroyed and taken off the cloud.

There! You have now completed the single-instance tutorial of OpenStack and managed to run a server image of Ubuntu 10.10 on your own virtual private cloud. Congratulations! The next step is to create your own custom UEC-images to be deployed in your cloud.

APPENDIX 2. TUTORIAL ON HOW TO CONVERT EXISTING VMWARE VIRTUAL MACHINES TO UEC INSTANCES

About this How To

This topic tells you how to convert an existing virtual machine (VmWare .vmdk in this example, since we VmWare as a template. You can apply this to virtually any kind of vm via conversion) to a UEC-compatible instance that will boot up. This only applies to Linux images so far.

This topic assumes that you have OpenStack Installed somewhere and capable of running instances. It is also assumed you are doing the conversion on ubuntu 1x.x (tested on 10.10 and 11.04)

Most of this topic has been applied via the instructions written in these articles: http://cssoss.wordpress.com/2011/04/27/openstack-beginners-guide-for-ubuntu-11-04-image-management/ http://www.linux-kvm.org/page/How_To_Migrate_From_Vmware_To_KVM

1. Convert VMDK to Raw KVM

In order to successfully deploy anything to the cloud, the easiest way will be to convert your existing virtual machines to a KVM image. KVM allows easy modification and mounting of the image via command line.

To convert your image, enter:

```
kvm-img convert -O raw <yourimage>.vmdk <newname>.img
```

2. Boot image and install necessary cloud packages

You must also install some software on your new instance, so it can easily communicate with the EC2 API. To boot up, write:

```
sudo kvm -m 512 <newname>.img
```

The first parameter defines memory for your vm. Please adjust it to match the needs of your machines. Most will probably make do just fine with approximately 512.

After booting, install these two packages to allow communication to the API:

```
sudo apt-get install openssh-server cloud-init
```

Cloud-init is a package you can obtain immediately from the ubuntu repositories.

You will also need to remove persistent network rules because this will intefere with the cloud controller mappings otherwise. Removal is done by running

```
sudo rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

After installation you can shut the virtual machine with

```
sudo shutdown now
```

3. Mount image to loop

First, mount the new image to your physical machine's filesystem, as a loop system.

```
sudo losetup  -f  server.img

sudo losetup -a
```

If this was done correctly, you will get output that looks something like this:

```
/dev/loop0: [fc01]:11927880 (/path/to/your/image.img*)
```

Take note of the path your image was mounted (in this case /dev/loo0).

4. Find out the start section of the primary partition

We need to extract the primary partition from the vm. To find out where it starts, run fdisk:

```
sudo fdisk -cul /dev/loop0
```

you should get output that resembles this:

```
Disk /dev/loop0: 17.2 GB, 17179869184 bytes255 heads, 63 sectors/track, 2088
cylinders, total 33554432 sectors
```

```
Units = sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x00002bc6



      Device Boot      Start         End      Blocks   Id  System

/dev/loop0p1   *        2048    32055295    16026624   83  Linux

/dev/loop0p2         32057342    33552383      747521    5  Extended

/dev/loop0p5         32057344    33552383      747520   82  Linux swap / So-
laris
```

We need to look at the first partition /dev/loop0p1 or the partition that has the Id of 83. Make note of the Start section bytes. In this case, it was 2048. This number needs to be multiplied by 512 to find out the length of the starting sector.

In this case 2048*512 = 1048576.

Make note of the number.

Finally, you can unmount the image from the loop:

```
sudo losetup -d /dev/loop0
```

5.Create UEC-compatible image without start sectors

You will need to mount back your image to loop, but without the starting sectors. This is where the number from the last step comes in. Run:

```
sudo losetup -f -o 1048576 server.img

sudo losetup -a
```

where the number in the first losetup is the marked up lenght of the starting sectors of the first vm's linux partition.

again, you will get output like this:

```
/dev/loop0: [0801]:16908388 ($filepath) offset 1048576
```

Again, remember where the image was mounted (loop0 in this case).

Now you need to copy the mounted image to a ext4 machine image. This is quite simply done by:

```
sudo dd if=/dev/loop0 of=/path/to/serverfinal.img
```

After copying is done (may take a while with larger images), unmount:

```
sudo losetup -d /dev/loop0
```

6. Final Tweaks to image

Now we will need to tweak the new obtained image a bit. First, modify the fstab file in the image. We will first need to mount the new image as a part of the new filesystem, by running

```
sudo mount -o loop serverfinal.img /mnt
```

Now we will need to edit the fstab file of the mounted image. Run:

```
sudo nano /mnt/etc/fstab
```

and modify the line:

```
UID=e7f5af8d-5d96-45cc-a0fc-d0d1bde8f31c  /               ext4    er-
rors=remount-ro  0       1
```

to

```
LABEL=uec-rootfs              /          ext4           defaults     0    0
```

7. Extract kernel and ramdisk from mounted image

To grant a bit of extra flexibility to our images, we usually want to run them with different versions of the kernel or ramdisk. So we also need to extract them from the original image. This is done with:

```
sudo cp /mnt/boot/vmlinuz-2.6.38-7-server /home/localadmin/location/to/vm


sudo cp /mnt/boot/initrd.img-2.6.38-7-server /home/localadmin/location/to/vm
```

now that we have copied all we need, you can unmount the image:

```
sudo umount  /mnt
```

8. Upload image to OpenStack

You will now need to upload the image to your openstack controller machine (or publish it). This done by using all of the three obtained files (the machine disk image, the kernel, and the ramdisk). Run:

```
uec-publish-image -t image --kernel-file vmlinuz-2.6.38-7-server --ramdisk-
file initrd.img-2.6.38-7-server amd64 serverfinal.img mybucket
```

this will bundle and upload the image to your OpenStack image bucket mybucket. This command will however take some time, and even when it releases the terminal (usually immediately), the uploading will continue on the background. You can create instances when

```
euca-describe-images
```

lists the newly published image.

9. Run image

Run image in the usual

```
euca-run-image ami-xxxxxxx -k mykey -t m1.tiny
```

fashion. Please note that you necessarily do not need to the exchanged keys if your machine does not contain the root user. In that case you can just ssh to the newly started instances in the usual manner without including the generated rsa-key.

Starting up the instance can take a while especially with larger vms, so please do not be alarmed if your deployment takes a while to start up. In my example case the old vm had an image that was 11GB, so publishing and booting it took up to 5-15mins each.

That's it, you are done! You have successfully deployed your existing virtual machine to an EC2 compatible cloud.