jamk.fi

INTRODUCTION TO PRINCIPLES OF BILLING IN CLOUD INFRASTRUCTURE

Tero Oinonen

Bachelor's Thesis
May 2014

Degree Programme in Information Technology
Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)
Oinonen, Tero

Type of publication
Bachelor´s Thesis

Date
26052014

Pages
51 + 20

Language
English

Permission for web publication
(X)

Title
INTRODUCTION TO PRINCIPLES OF BILLING IN CLOUD INFRASTRUCTURE

Degree Programme
Information Technology

Tutor(s)
Rantonen, Mika
Häkkinen, Antti

Assigned by
Rintamäki, Marko

Abstract

The original idea for writing the thesis was to establish an open source billing tool system for SkyNEST project with available resources in accordance with Cloud Software's goals. However, because a practical solution such as this was under development in the early part of the thesis research, it was decided that the practical execution was to be left with less priority and the focus was to be on comparing and researching already existing billing systems and the possibilities they provide.

Since the billing systems in cloud services are in an early phase themselves, there are fundamental problems in choosing the billing principals and reliable meters needed. Before the larger service providers in the industry are able to find a common and accepted service model, it is difficult to point out a functioning model that could scale to serve all sizes of services and that billing could be scaled to fit all needs.

During the writing of the thesis two kinds of basic models set themselves apart from the others, namely open source systems and systems based on proprietary code. The differences in these two approaches were easily visible in both the presentation and installation phases. At the moment it is difficult to find one solution that suits all needs, however, it is possible to recommend alternative combinations based on a certain profile of using the billing service.

Keywords
Cloud services, billing, OpenStack, FreeNest

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) Oinonen, Tero	Julkaisun laji Opinnäytetyö	Päivämäärä 26.05.2014
	Sivumäärä 51 + 20	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
INTRODUCTION TO PRINCIPLES OF BILLING IN CLOUD INFRASTRUCTURE

Koulutusohjelma
Tietotekniikan koulutusohjelma

Työn ohjaaja(t)
Rantonen, Mika
Häkkinen, Antti

Toimeksiantaja(t)
Rintamäki, Marko

Tiivistelmä

Opinnäytetyön tarkoituksena oli alun perin toteuttaa itse mahdollisuuksien mukaan OpenStackin ympärille avoimen lähdekoodin työkaluilla laskutusjärjestelmää. Työ tehtiin SkyNEST-projektille Cloud Softwaren tavoitteiden mukaisesti. Koska ajatellun kaltaisen käytännön toteutuksen havaittiin jo opinnäytetyön alkuvaiheen tutkimuksessa olevan kehitteillä, päätettiin käytännön toteutus jättää opinnäytetyössä vähemmälle painoarvolle ja keskittyä sen sijaan vertailemaan ja erittelemään jo olemassa olevien laskutusjärjestelmien eroavaisuuksia ja mahdollisuuksia.

Koska pilvipalveluiden laskutusjärjestelmät ovat itsessään varsin varhaisessa vaiheessa, on myös laskutusperiaatteiden ja vaadittavien mittareiden valinnassa omat ongelmansa. Ennen kuin isompien palveluntarjoajien valitsemana ja hyväksymänä löytyy jokin yhdessä noudatettava palvelumalli, on vaikeaa sanoa yhtä kokonaisuudessaan toimivaa palvelumallia, joka kattaisi kaiken kokoiset palvelut, ja laskutus saataisiin skaalautumaan riittävän hyvin kaikkiin tarpeisiin.

Työn toteutuksen aikana erilaisista laskutusjärjestelmistä erottui selkeästi kaksi erilaista mallia: avoimen lähdekoodin järjestelmät ja suljetut järjestelmät. Näiden kahden lähestymistavan erot olivat selkeästi nähtävissä sekä palveluiden esittelytavassa että eroissa niiden käyttöönotossa. Yksiselitteistä ratkaisua koko laskutusjärjestelmän toteuttamiseen on ainakin tällä hetkellä vaikea löytää, mutta useita tiettyyn käyttömalliin sopivia vaihtoehtoja on mahdollista suositella

Avainsanat (asiasanat)
Pilvipalvelut, laskutus, OpenStack, FreeNest

Muut tiedot

TABLE OF CONTENTS

FIGURES

TERMINOLOGY AND ABBREVIATIONS

API – Application Programming Interface, a specification that defines how software components communicate with each other.

ASP.NET – A free web framework that has been intended for building websites, apps and services using HTML, CSS and JavaScript as programming languages.

Client – In networking client usually is the endpoint of communications which runs the service and queries the server via service provider for certain data.

Cloud service – A service that is executed physically somewhere else than on user's local workstation and is accessed via the internet. In practice this means that user is able to use service like this from any location supplied with a working internet connection.

CPU – Central Processing Unit, also referred to as a processor. Core hardware component in computers that performs basic input/output operations and calculations.

Devstack – A simplified installation of OpenStack that is based on scripts and can also be installed on a single computer, all services in one node.

FreeNest – A part of SkyNEST project, team working environment that consists of several open source development tools.

FTP – File Transfer Protocol, a standard network protocol that can be used to transfer files from host to another using TCP-based network.

Git – A revision control and source code management system that has set its emphasis on speed.

GitHub – A hosting service for Git projects. Offers free accounts and paid private repositories.

HTTP – Hypertext Transfer Protocol, an application protocol that is the foundation of data communication for World Wide Web. It is structured text that uses logical links, hyperlinks, to provide access between nodes containing text.

ICMP – Internet Control Message Protocol, one of the main protocols in Internet Protocol suite. Used by network devices to send error messages indicating exceptions to normal functioning. Ping is a common example of diagnostic tools using ICMP.

ICT – Information and communications technology, general branch that consists of electronics industry, computer-related sciences and telecommunications.

Instance – Within this subject instance usually means a single virtual server that operates in a cloud environment. For example, user may establish an installation of several physical servers with OpenStack to create a cloud environment that is based on a certain operating system. A single virtual machine working in this environment is called instance.

IPMI – Intelligent Platform Management Interface, a standardized interface used by system administrators for management and monitoring of computer systems.

IP-address – Address that is used for data transfer between devices in the network. In IPv4 standard addresses consist of four sections (for example 192.168.0.1) and in IPv6 of eight sections (e.g. 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

JavaScript – A computer programming language originally developed by Netscape Communications. Most commonly used as a part of web browsers, handling communication with user and various types of other interaction.

JMX – Java Management Extensions, a technology that supplies tools for management and monitoring of applications, devices, system objects and networks.

Latency – Used to describe delay between the input and output, usually used with network connections.

Linux – A definition group of operating systems that are using Linux-kernel, developed using Unix-based operating system. Distributed with open source licences and has been tailored for many different types of systems.

MySQL – MySQL is a widely used open source relational database management system. Several applications use MySQL databases and it is distributed under GPL licence.

NIC – Network interface card, a computer peripheral that connects and maintains network connections.

Node – In cloud environment the servers maintaining the cloud are called nodes. Various types of nodes are for example controller, network and compute.

Node.js – A software platform for networking and server-side applications. Applications are written in JavaScript and can be easily scaled.

OpenStack – Open source software that allows the users to create a cloud service environment using various types of hardware configurations. This environment can be used for various tasks such as running virtual machines or providing storage space.

Open source – Type of code that is being distributed under a license that allows it to be free and modifiable, usually under the condition that if code is modified, the changes and the original source codes must be accessible in some way so that everybody has at least theoretically the means to achieve the same results. More accurate details depend on the details used in the license agreement.

PHP – PHP: Hypertext Preprocessor, a server-side scripting language that has been designed for web development. It is also used as a general-purpose programming language.

PostgreSQL – An object-relational database management system, designed to comply with already existing standards and to provide extensibility. It is open source software and has been released under PostgreSQL License.

Quota – Describes the amount of capacity or calculation power allocated to a certain service. In networking it usually also means the amount of data transfer that user is provided within a certain amount of time. Generally in cloud services the quota admitted to a customer is defined by the amount of payment for usage, e.g. by paying more monthly the customer may raise the amount of data he is able to access.

RHEL – Red Hat Enterprise Linux, a commercial Linux distribution mostly used in server computers. Developed by Red Hat.

Server – A specialized and centralized computing unit with the purpose of providing services and controlling connections to Internet. Simplified, the Internet itself is a huge network of servers linked to each other via service providers.

SkyNEST – Project that is a part of Cloud Software Finland and uses mostly students of JAMK University of Applied Sciences as employees.

SLES – SUSE Linux Enterprise Server, a Linux distribution designed for servers, mainframes and workstations, developed by SUSE.

SMTP – Simple Mail Transfer Protocol, a standard for e-mail transmission. Extensively supported by email clients and services and also commonly used for mail transfer between proprietary systems.

SNMP – Simple Network Management Protocol, Internet-standard protocol for managing devices on IP networks. Devices that support SNMP include for example routers, switches and printers.

SQL – Structured Query Language, special-purpose programming language that has been designed for use in databases and management systems.

SSD – Solid-State Drive, a type of data storage that does not use mechanical parts or disks unlike traditional hard-disk drives. It uses integrated circuit assemblies to store data and has much higher performance and arguably better reliability compared to disk-based alternatives.

SSH – Secure Shell is a network protocol for secure data communication and also allows remote features, such as login and command execution. It uses secure channel to connect a server and a client running respective programs.

TCP – Transmission Control Protocol (also TCP/IP), part of Internet Protocol suite and a protocol that web browsers use when they connect to other servers.

Telnet – A network protocol used on the Internet and local area networks for communication.

UI – User Interface. A term used to describe how user interacts with the service and applies commands and changes to it. Most usually used in conjunction with graphical layouts and usually has its own design teams to improve usability.

Unix – An operating system initially released in 1973. Unix is the basis for many current operating systems, such as Linux and Mac OS variants.

Uptime – A term most commonly used when talking about server computers. Defines the amount of time that a computer has been constantly powered up without being interrupted. Being shut down for some reason resets the uptime counter.

Virtual machine – A computer that basically exists only in digital form. In other words, it is a machine with a hardware definition; however, it has been defined by another computer. It needs actual physical computer to exist but one physical computer can be used for running several virtual machines at the same time. Running several virtual machines at once is usually very processor-intensive task.

XMPP – Extensible Messaging and Presence Protocol. A communications protocol based on XML and designed to be extensible. Has been used for numerous purposes, such as file transfer and social networking services.

1 INTRODUCTION AND THESIS OBJECTIVES

1.1 Introduction

SkyNEST was a project within JAMK University of Applied Sciences that was mostly carried out with student workforce and is a part of Cloud Software Finland. The benefits of using students as employees in a project are, for example using the various talents among different people to form teams that get along together well and also create one larger team for project purposes. Practical example of this way of working would be suggestions from other teams to feature a team that finally executes the changes to the environment after thinking and planning for the most efficient way. One particular benefit in the project to help with understanding the development environment is actually to use the tools under development for reporting and working at the same time. This provides much better understanding of the actual usage of the environment and prevents coders from ignoring problems that actually hinder the overall usability.

The users could also be seen as testers at the same time in this case. As a project where teams change frequently it is important to have some kind of product development cycle that allows new teams to efficiently carry on where the previous ones left off. One particularly important method to allow this cycle to work is documentation. It also helps team members working at the same time to have a better insight on other team members' work. Other important tools for controlling the efficiency while leading the project are Lean and Agile; these will, however, be left out since they are not an essential part of this thesis work.

The idea of writing the bachelor's thesis on billing systems within OpenStack came from Marko Rintamäki, the long-time leader of the SkyNEST project and Cloud Software program that SkyNEST was a part of. The author joined the project in September 2012 as an intern, being basically the only worker specialized mostly in networking, as other workers were more software development oriented students. This sometimes proved to be an asset; however, much better coding skills would have been an asset in this project. Luckily the work carried out was not very dependent on the ability to code; therefore, the internship time was productive, including configurations and modifications for cabling and classroom switches, several OpenStack server envi-

ronments, most of which were established by the author from the beginning, even though it must be admitted that when the knowledge how to deploy some version of OpenStack is there, its next update is usually not that difficult to establish.

While being in the project the interest towards cloud computing has increased, and at the same time the interest in networking has somewhat diminished. When thinking about the future, this could most likely be a benefit when trying to find work and also provides an asset compared to graduates with networking specialization.

1.2 Objectives of thesis

The main objective during the thesis was to carry out some research of already existing billing systems on various cloud service platforms and to see how difficult it would be to establish a billing system based on OpenStack using tools already existing for the purpose. There were some functionalities lacking in OpenStack's own tools for the job, and it was not be very easy to make a working entity from start to finish, especially taking into account the author's limited ability to code and script. Thus, the study mostly concentrated in finding out what possibilities OpenStack's API's posed for getting information on instances used, and after that finding out if there was a simple way to parse the data extracted into file where it would be efficient and easy to deliver data for billing purposes. In the time given for thesis it was relatively clear there was no time for corporate-grade service suite for billing; however, considering that some of the software groundwork was already done with Ceilometer, the task to make a simple system based on scripts would not prove to be impossible.

Due to extended time taken while working on thesis there were some changes that affected writing the thesis, such as version changes, inability to remember some details about the environment after longer pauses and structural changes in the project itself. To increase efficiency and eventually finish working on the thesis, author decided to concentrate on some things as they were during the beginning of writing to save time. For example, several versions and updates have been implemented to OpenStack and its components even to the time this thesis is published, therefore

some details have been introduced in a more general method to avoid unnecessary blending among different software versions.

2 BASICS OF CLOUD SERVICES

2.1 Brief introduction to cloud computing

The term “cloud computing” is sometimes difficult to determine, as there are so many different yet partially similar solutions and technologies that are related to the internet. One official view to this matter is from NIST Handbook which determines cloud computing using five essential characteristics. These are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. This basically sets cloud services apart from other network services by leaving all the service of cloud environment to service provider, requiring that the services are available regardless of networking device used and on technical level. The user’s actions while using the services have to be measurable, scalable and controllable at some level. (Mell, Grance, 2011)

The term “cloud” itself usually has been explained as a system where several components and services reside, but which has several forms that are not necessarily easily located physically or may be pinpointed exactly, as if a cloud would be surrounding the larger system to prevent majority of users from knowing matters that are not directly linked to simple working service itself.

As of today, most cloud services used are provided by large companies with an existing palette of services that play well together and that can effortlessly be used online. The ideal benefit of using cloud services is that work and other computer-related matters can be continued even when transitioning between several devices, as if one still had the same device in use all the time. The most important benefit of this service model would probably be when using desktop computers divided across several physical locations, being able to synchronize the work already done in the first location, carrying on with it in a new location. It could be stated that having a small lap-top or a somewhat similar device makes using cloud services a little less revolutionary; on the other hand, other modern devices with some limitations, for example small physical

storage space can definitely benefit from data being accessible over the internet, without the need for local copies of files.

There is usually something in every branch of software development benefitting from open sourcing. In case of cloud computing, it can be mostly seen as several environments built upon a basic idea from some closed source service and bringing it to a community for more and larger possibilities of continuous development. The downsides related to this open source approach are usually related to support towards entry-level users and as developers are usually loosely tied to a company they are working for, when compared with closed source variants, there is a larger possibility of bending deadlines and changing plans in the middle of certain product cycle.

A downside for more experienced users in cloud services resides in the way cloud stores and processes data. When there is some problem with certain data, in most cases it is impossible for the end users to know where their data is located physically in a cloud, as data is spread across several nodes. Usually even system administrators have a hard time trying to find out where certain user's data is stored in a large cloud, so in rare cases where data is stored in a cloud and corruption or some other fault occurs, data may be irreversibly lost. That is why not even a cloud can always be relied on as the only place data should be kept safe. When used in addition to copies in another physical location, for example backup purposes, cloud computing offers great possibilities for keeping files safe.

2.2 Basic classes of cloud services

SaaS, Software as a Service, is usually described as the highest level of cloud service infrastructure. It is also the lowest level of end user customization, most of the service controlled by service provider, easy to initialize - SaaS is intended to typical entry-level users. When concerning security, responsibility for security is definitely on the service provider. The best example of a SaaS type cloud infrastructure service is Google Docs.

PaaS, Platform as a Service, mid-level of service architecture; hardware and operating system is provided by service provider, the rest is left for user to decide – PaaS is ideally designed for system analysts, designers and such. Responsibility for security is distributed between the customer and service provider, after certain level service provider may only be able to provide tools for improving security. Good examples of this type of platform are Windows Azure and Google Apps Engine.

IaaS, Infrastructure as a Service, presents the lowest level; it leaves plenty of room for users to customize the cloud service how they want, it is more difficult to use and initialize and it usually contains only network connections, disk space, servers and their maintenance. IaaS is mostly meant for IT professionals and other users that value customizability and features over simplified usage. From security point of view, in IaaS the security among most of other functions usually falls into the hands of the user. Amazon EC2 is a good example of IaaS level service. (O'Neill, 2011)

2.3 Basic structure of a billing system

In order to understand and compare various billing platforms it is logical to start by taking a look at what the most usual methods and principles centered around billing system have in common. By forming an idea of a typical billing system it is easier to spot differences in services that deviate from the common pattern and aim to set themselves apart among competition. Even the basic structure of a billing system can be divided into several categories, showing the typical questions that billing system designers have to face during the design and initiation of their service.

Two most basic types of billing engines are hosting billing and telco billing. Hosting billing is basically subscription billing with a fixed amount each month on a single bill, compared to telco billing, where the billing is based on numerous meters and records. The name comes from resemblance to practices that telephone companies have. This thesis mostly focuses on telco billing, due to fixed price billing being much simpler subject because of less variables used. (Bligh, 2012.)

2.4 Payment model

One of the basic problems in designing the billing system is how the customer is charged based on their usage of services resources (see Figure 1 for common payment methods). There is also an exceptional method of ad-based revenue model in which the revenue comes from advertisers rather than the users themselves; however, this is usually quite a rare model within cloud billing platforms due to expensive maintenance costs. There would have to be loads of advertisements to cover the upkeep costs of the service in case there is loads of hard disk space and memory and other hardware components concerned. However, for example a trial version of a cloud platform could probably be well profitable using this model, provided that usage limits have been set strict or the trial period is time limited with no easy workaround to reset the timer.

These same problems mostly exist with freemium service model, as cloud service in its most typical form is usually quite a simple and invisible service that consists of few easily measurable basic services. This leads to a problem where somehow limiting the service causes a severe deterioration in service usability and quality. For example a basic cloud service limited to a quota of for example 256 megabytes of RAM and less than a gigabyte of hard disk space is mostly usable only for brief testing purposes. But as even that amount of service provided has its maintenance costs, it is usually more profitable to keep testing services both time-limited and in low capacity.

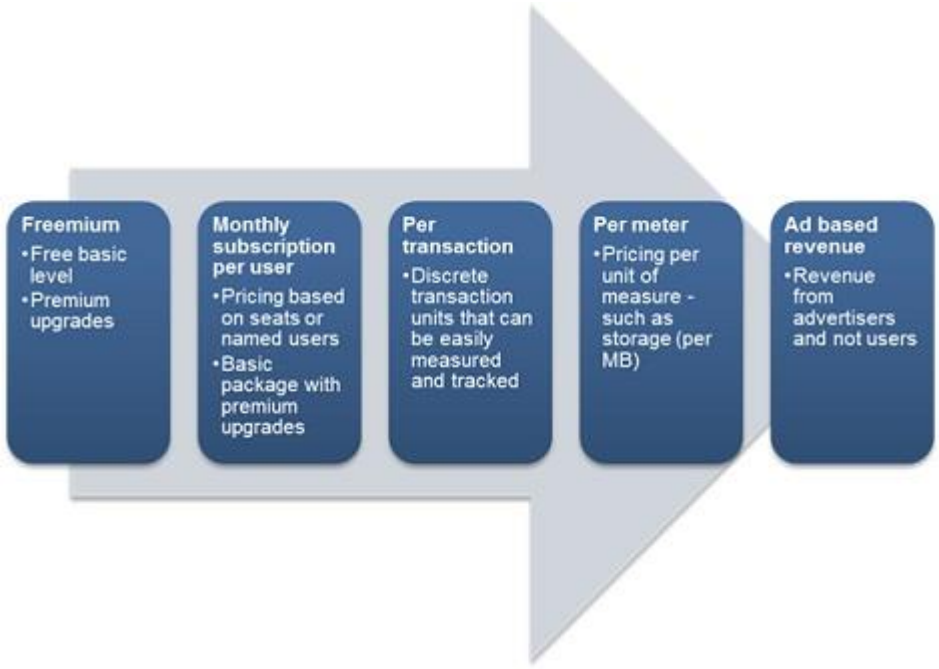

Figure 1: Various payment methods
(Netmagic site, 2014)

2.5 What data is needed for reliable billing?

Even though cloud service providers usually advertise their solution as a simple and carefree option, the system behind the users' cost calculation may be in some cases very complicated (see Figure 2 for an example). In its most simplified form, the basic data needed for reliable billing consists of hardware that the customer is using as a backbone of his service platform, the amount of data that has been either transferred in and out of the platform or the capacity that user is allowed to access with his current plan. As of now, the billing in cloud systems is still searching for common standard that could be generally accepted as the basis for all systems.

Currently there is still a lot of differentiation between the large service providers in their billing systems, but it is possible that in the near future as research information of systems gathers, the providers start moving towards more unified billing methods based on information gathered. When taking customer's viewpoint, it is somewhat complicated to define what kind of service model is the most efficient without more thorough research, as the cost basically varies due to certain variables used in billing systems. Simplified, two different users may get a whole lot difference between the

costs of various cloud services due to differences in their ways of using the service. (Butler, 2013)

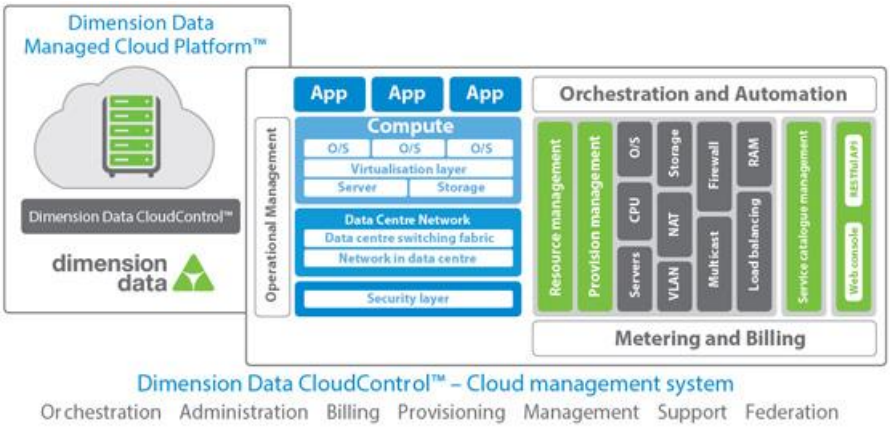

Figure 2: Example of measurable components in cloud service
(Dimension Data site, 2014)

2.6 What software combination serves the purpose best?

There are several things that need to be taken into account when choosing the best possible software for billing in company, and one of them is to evaluate how long it will take to implement the chosen billing system into use. Billing system has to be in somehow functioning order for company to successfully do business, and all the extra time that goes into prolonged implementation and solving issues in billing systems will cause loss of revenue in time. For starters, if subscription management is not core competency for the company designing the billing system itself, it would be the best choice to concentrate on systems that have some sort of customer support in case of problems.

Another important aspect is the flexibility of the chosen billing system. As the company evolves, there will most likely be changes in how the system needs to adapt to current needs. If there is no flexibility in chosen service package to start with, the problems will most likely occur very soon, so it is primarily important to take services scalability and modifiability into account when making comparisons between available options. In some cases there are also company regulations that need to be complied with, and finding out about them and how they limit the options available is crucial before making decisions between services.

In case the company already has some information systems that are needed to be implemented with the billing system, it helps a whole lot if the system chosen is intended to work well with other already existing systems. Also if the service has allowed an access to API it is possible for more experienced workers in company to further modify compatibility between their existing software and the billing system of choice. One last thing to look into is how viable the vendor of software actually is. When looking at cloud billing services, this may be somewhat difficult issue to evaluate as there are basically no well-known companies that are providing the services, and most of these new companies that have focused their commercial efforts around these billing systems have not yet made a name for themselves.

This issue will mostly resolve itself in time, as service providers that have issues with their customers and are not satisfactory among their competitors will not remain profitable, and those that have done well in industry remain. Until then, the customer will have to choose the platform solely to fit their needs in billing and see how the service provider can cope with keeping the service maintained.

As stated earlier, choosing a combination that can be scaled for all kinds of usage purposes in this case is very difficult. If the user is experienced in using scripts and certainly knows what he is doing and does not need support other than developer documented manuals, the most modifiable and basic solution for billing would be using OpenStack's Ceilometer API combined with simple third party project like BillingStack (see chapter 5.1). However, as BillingStack is itself somewhat in progress, it should be definitely noted that a self-established billing system using only these API-based options is only recommended for those that really understand what the code does and how to fill the blanks in certain points to make the system fit the purpose in question.

The most simplified solution apart from services that themselves contain everything from the cloud services to billing systems and other services would be using OpenStack with a service that has been developed to allow easier usage, such as OpenBook by Talligent.
(Primault, 2011)

3 EXAMPLES OF MAJOR BILLING IMPLEMENTATIONS

3.1 Amazon

The work on the thesis was started by surveying already existing methods of billing among large providers in cloud services. A logical starting point was Amazon that had gained a strong foothold in cloud services with their EC2 platform. Their site was visited to find a separate FAQ section in which they provided insight into their principles of billing. There is no minimum value for purchase, and the user is charged directly per usage of virtual machines. The modifiers affecting how much user is charged depend on the area where the virtual instance is being used and on the type of these instances (Figure 3 for price calculator example). Virtual instance has several different options from which users can choose what kind of system is best for their purposes. Among the options are for example light, medium and heavy usage, and as for operating systems Linux, RHEL (Red Hat Enterprise Linux), SLES (SUSE Linux Enterprise Server) and Windows are available. In case customers choose Windows, they can also select service templates to be installed, e.g. SQL (Structured Query Language) if needed. Finally, the amount of hard disk space is chosen. Additional services include elastic IP addresses and CloudWatch that is separately billed due to having several variations of its own. The site even contains a calculator that can be used to estimate user's monthly billing costs based on current usage. (AWS Billing FAQs, 2013)

For testing purposes Amazon currently provides AWS Free Tier option that includes 750 hours of Linux and Windows Micro Instances each month for one year. As this option is quite limited it is obviously best used for getting used to Amazon's cloud environment and comparison between competing platforms, but free version for testing is of course always a welcome option. However as usually in this kind of testing periods, users' billing address and credit card information is required when registering for free AWS account, so in case user happens to exceed limits sets to free AWS account, he will be billed according to amount of exceeding usage.

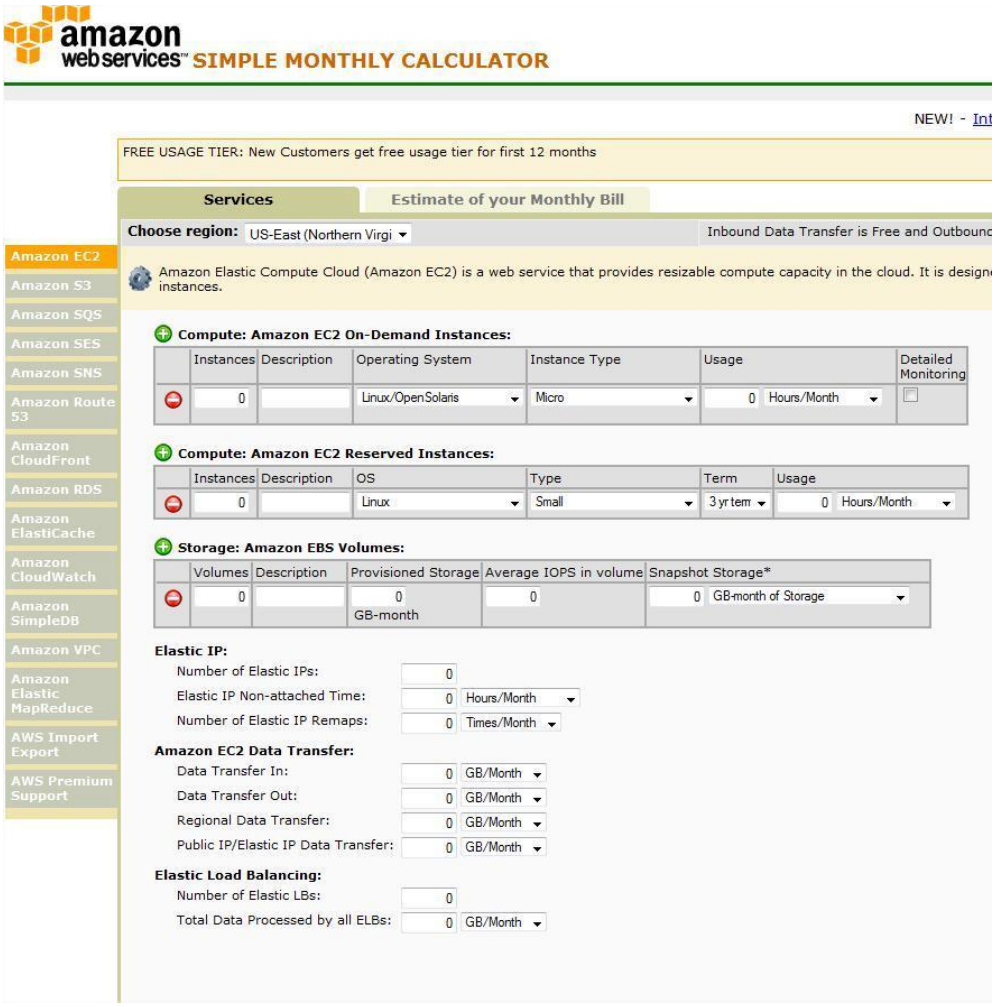

Figure 3: AWS price calculator

(ScienceLogic Blog, 2011)

3.2 Rackspace

Rackspace also charges their users based on the usage of their servers. The billing cycle of 30 days begins on the day when the service has been activated. The exception to this usage billing rule are services called Cloud Services and Cloud Load Balancers. These two types of services are billed based on uptime, independent of the time being actually used. Therefore, if user has activated these services, they are constantly being billed from the time of being activated onward. Various templates of service provided by Rackspace are Cloud Servers, Cloud Load Balancers, Cloud Sites and Cloud Files, each with their own billing principles and calculators for overall cost. (Rackspace Support, 2013)

The three main service variations Rackspace provides are Public Cloud, Managed Hosting and Private Cloud. Public and Private Clouds are basically limited to service-only level and are built on OpenStack, whereas Managed Hosting also includes hardware customized to customers' needs. Rackspace is also trying to set itself apart from competing service providers by combining these three service variation in a single larger entity that is called Hybrid cloud. This combination basically allows the user to take all the benefits in different implementations of cloud environment and further customize the system for their own needs, to maximize performance, reliability and security. As the author had some experience working with OpenStack environment, it was quite interesting to see what one of OpenStacks creators, Rackspace had implemented around OpenStack for its own commercial purposes.

One key difference in getting a personalized cloud environment built on OpenStack from Rackspace and building one yourself is of course customer support. OpenStack being open source code definitely helps when in need of information; however, having paid for support provides a much easier method for troubleshooting cloud environment.

3.3 DigitalOcean

DigitalOcean provides a cloud service aimed towards developers and their goal is to make starting the use of service as effortless as possible. Their site states that SSD cloud server can be deployed in less than 55 seconds. The basic plans that DigitalOcean offers are called Pay-As-You-Grow and their pricing ranges between 5 and 80 dollars per month. The smallest plan contains 512 megabytes of memory, 1 processor core, 20 gigabytes of SSD disk and a one-terabyte quota of data transfer. The largest plan consists of 8 gigabytes of memory, 4 processor cores, 80 gigabytes of SSD drive space and five terabytes of transfer quota (basic plans in Figure 4). There are also many plans exceeding these five basic plans they are not as visible on site as the basic alternatives. Only outbound transfer counts in bandwidth and once monthly transfer limit has been reached, every gigabyte transferred after this limit adds $0.02 to monthly costs. DigitalOcean calls each of its virtual machines a Droplet, apparently based on the idea that large ocean is a sum of huge amount of drops. Droplets are

charged even when in powered-off state and they are billed once per hour. Digital-Ocean has servers in US and Europe.

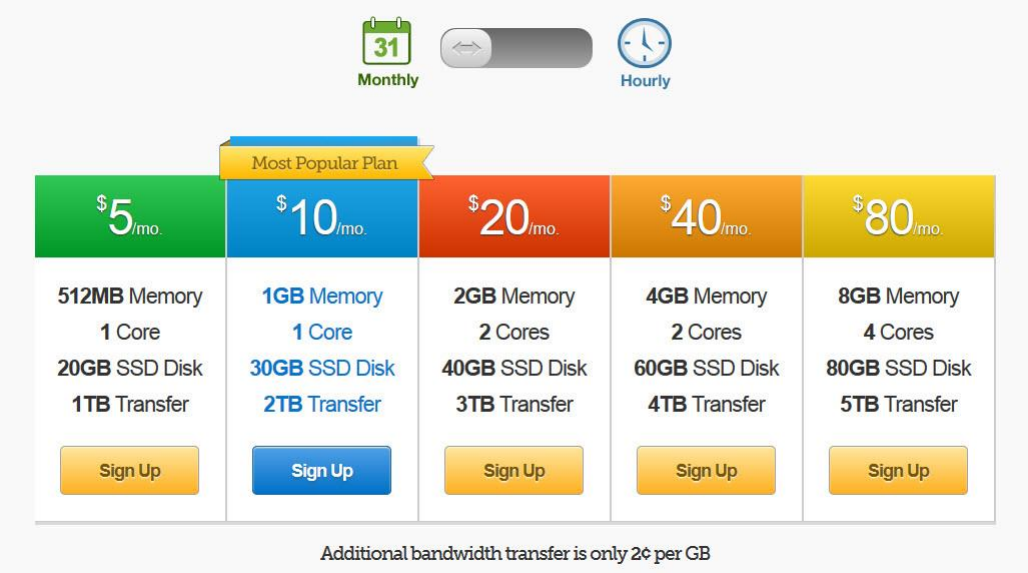

Figure 4: List of DigitalOcean billing plans
(DigitalOcean site, 2014)

3.4 Azure

Azure is Microsoft's cloud platform that markets itself as fast and innovative service that supports plenty of different technologies, most notably many of Microsoft's own standards and thus provides thorough end-to-end support for enterprises. Integration with Microsoft's own already existing systems is a major benefit to those already connected to compatible operating systems and services. According to Microsoft, 57% of Fortune 500 listed companies use Azure platform, and some notable customers include BMW, NBC and Toyota.

A free one month trial was also provided at the time of writing this thesis, which is an obvious benefit to those willing to test the service themselves before purchase decision. Unfortunately, the research work for Azure was left for a late part on thesis, so there was not enough time to actually test the platform in practice. The advertised benefits of the service include support for ASP.NET, PHP, Node.js and Python as programming languages FTP, Git, Visual Studio and GitHub as tools for deploying services.

4 STANDALONE BILLING SERVICES

4.1 Aria Systems

Unlike previous services mentioned in this chapter, Aria does not provide the cloud platform itself, but only the tools for billing in cloud environment. As Aria's service is less transparent and relies more on commercial methods of making the service known, it was more difficult to get detailed information on how their billing works than in the other cases. Still, there were some very useful principles mentioned and therefore it was important to mention how their method of billing works.

When comparing Aria's methods to other somewhat similar services, it becomes clear that Aria fully concentrates on promoting the benefits of keeping the billing system as a separate component from cloud system itself. This may have some benefits, such as the billing itself may be more freely customized according to clients' needs, however, on the other hand ensuring the compatibility between this third party billing system and the cloud that is being monitored may be problematic.

Like most other service providers, Aria's method of demonstrating their service on their website is centered around slogans and taglines to highlight the issues they have paid special attention to. Somewhat comparable to Zuora's approach, they also have a wide variety of materials on their site to support their approach. From the developer's or administrator's point of view, the most significant issue with Aria is that they have taken a slightly more commercial approach towards their billing system than most of their competitors. What this practically means is that even getting a hands on – demonstration on their product requires providing Aria with some information and scheduling for a demo. At least for many of developers who have been working mostly with open source components, this kind of closed approach may be somewhat concerning.

Due to Aria's commercial style approach the author mostly used their site for researching material related to the topic and to pay attention to common partnering companies that like to work with projects on cloud systems. Resources on their site contained plenty of useful information for designing the billing process, such as in-

fographics that systematically explains the questions needed to take into account when moving into a subscription-based model. As the cloud billing is itself quite new as a topic, most materials were focused around customer relationships and controlling revenue. (Aria site, 2014)

4.2 OpenBook

The OpenBook billing platform by Talligent is a billing system that supports VMware, Virtual Bridges and Red Hat in addition to OpenStack. Their site is full of simplified slogans and it quickly becomes apparent that their system has been designed with ease of use in mind. The simplicity is not even limited to their website, as their UI (User Interface) has been designed to contain the information in a simplified form and terms that are correct but not too complicated to understand.

Even though Talligent markets their service as open solution, it seems that at least taking a look at their site more carefully reveals that they have taken many features into their platform and service model from more commercial solutions. To name a few examples, contacting the company to learn more about the service requires, at least that the time of writing the thesis, leaving contact information and scheduling a time for a demo on the product. Their site is simple and does not contain much information on the product itself, however, they have a separate blog where more information can be found. The blog is mostly focused around events that are being held on cloud services. (Talligent site, 2014)

4.3 Zuora

Zuora provides their billing system for all sizes of companies across any industry. The method of billing can be chosen from several plans available (Figure 5 for invoice example). The payment periods available include up-front, monthly, quarterly and annual payments, and models one-time, recurring and usage based pricing. It is also possible to setup free trials, discounts and promotions. Zuora also supports a wide variety of payment providers in their ecosystem. They also provide different types of service aimed towards either SaaS or IaaS/PaaS service model companies. Zuora's webpage has one certain advantage over most competitors of the same field; they pro-

vide lots of material in their website to prove that their approach towards establishing a billing system is the right one. They have plenty of slogans and taglines too, but also many references to factual releases to back up their claims. As an example in the Resources section they had at the time of writing a list of videos, whitepapers and webinars on related topics.

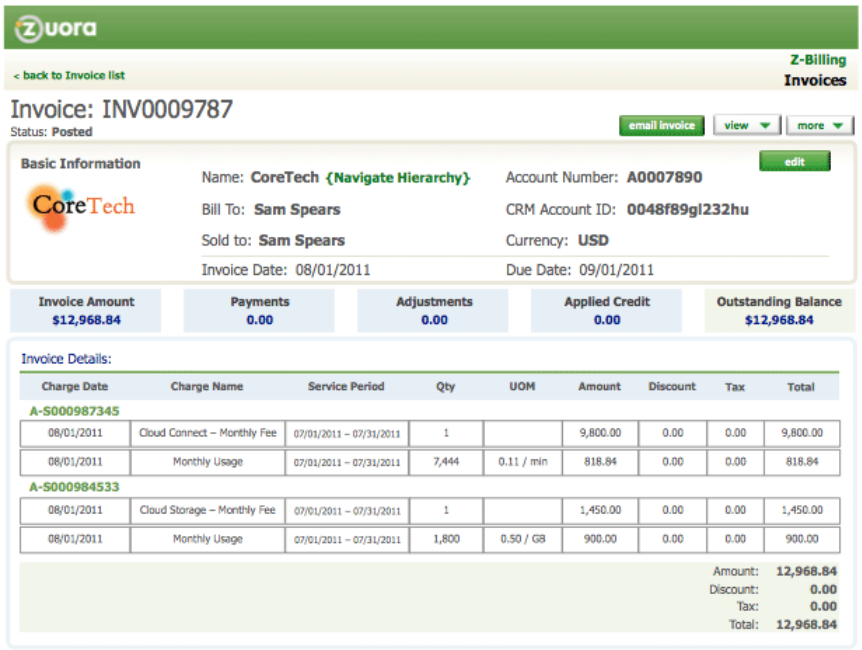

Figure 5: Example of Zuora invoice
(Zuora website, 2014)

4.4 MetraTech

Metanga and MetraNet are cloud-based billing applications provided by MetraTech and they support Salesforce.com and other leading financial applications. MetraNet has pre-configured product catalog templates for AWS, Azure, Office 365, SmartCloud, Cisco vBlock and Hosted Collaboration Services. Their MetraNet works with IaaS, PaaS and SaaS applications and it provides the means for more agile administration in cloud. MetraNet itself is built on metadata-driven architecture, allowing new services to be easily added and modified without compromising the usability. It also has the ability to implement the service by automatically creating data models, user interfaces and such using parameters defined.

The list of MetraTech's partners includes many well-known companies, such as Fujitsu, Oracle, PayPal and even the Finnish IT service company Tieto. Their site is well equipped with various information, including case studies, datasheets, infographics and other useful material of cloud billing. Of MetraTech's services, Metanga is more aimed for simple cloud-based billing, whereas MetraNet provides a larger set of billing solutions directed mostly towards enterprises.
(Metratech site, 2014)

4.5 Other service providers

To emphasize the large number of companies and services already working in the industry, the author has listed here in Table 1 other services that offer billing solutions for cloud services, but were discarded from having a chapter of their own due to lack of information or other reason.

Table 1: List of other cloud billing providers

Service provider	Reason for discarding
Spreedly	Briefly mentions support for cloud-based platforms but no detailed information was available on brief period of research.
CheddarGetter	Has cloud storage provider Put.io as a customer but site does not specifically emphasize support for billing cloud services.
Fusebill	In the features section, cloud-based business is mentioned but no other specific cloud billing support is introduced.
Chargify	Cloud service –based companies called Panda and Vend are mentioned among customers, but cloud billing is not specifically mentioned in service specifications.

(Information gathered from services' respective home pages mentioned in the references section)

4.6 Comparison

Due to many services offering similar features to each other, the amount of products included to this comparison (Table 2) is limited by their popularity and approach towards code.

Table 2: Comparison table on billing platforms

Service	Aria Systems	BillingStack	Openbook	Zuora
Proprietary/ open source	Proprietary	Open source	Proprietary	Proprietary
Costs	Customized for customer	Free	Customized for customer	Customized for customer
Pros	- Several partners within industry	- Customizable	- Supports several platforms	- Service models for commerce, billing and finance - Frequent live demos - Integration with other services and customer support
Cons	- Demo option requires scheduling	- Installation and configuration a bit tricky	- Not much detailed information available without inquiry	- API may be too complex for loads of small payments

5 TOOLS FOR BILLING

5.1 BillingStack

The author originally found out about BillingStack (Appendix 3 for installation instructions) because it was mentioned in a convenient location, in Ceilometer documentation as the lone project using Ceilometer API at the time of writing this thesis. It quickly occurred though that BillingStack had one very specific function that set it apart from other billing systems mentioned and had somewhat mixed impact to writing of this thesis.

The author had originally ambitiously wanted to first take a look at how the APIs work in cloud billing services and Ceilometer in particular, and then, after learning how Ceilometer gathers data and processes user commands, to make a collection or a list of command scripts that makes commanding Ceilometer directly much easier. Then, if possible the author would even try to create an component of its own that is solely dedicated to the purpose of working as a messenger between the administrative level user that has need for data that can be used for billing, and the Ceilometer API itself. This BillingStack component basically is exactly the component that does all this. BillingStack functions by using simple Json commands to communicate with Ceilometer API and its documentation basically covers all the parties that a simple billing system contains. It could be said that this discovery saved the author a lot of practical work, however, it also caused a new problem in its wake; the practical study and actual, concrete benefit of thesis was still missing.

BillingStack is an open source project that provides the user with various commands and scripts that can be used to fetch data to be parsed into form for billing purposes. In addition to OpenStack, it also functions with CloudStack or basically any other IaaS software stack system. Due to the open nature and simple methods how BillingStack works the creators have seemingly aimed for a solution that can easily be modified for just about any purposes that revolve around billing, mostly because Json queries are well documented and there are no proprietary components used in its code. Documentation for BillingStack is available in many forms and contact details, mailing lists and other usual contact methods to developers are available.

BillingStack's architecture consists of two parts, pgp (PaymentGatewayProvider)/ pgm (PaymentGatewayMethod) and API. Pgp is a provider for processing the transaction and pgm an alternative component for connecting to a payment method such as Visa or similar. API is the standard interface for accessing various functions within the BillingStack system. BillingStack's documentation was at the time of writing somewhat unfinished, so it is to be seen if the project continues more actively in the near future or if the developers have decided to leave room for more commercial solutions and just allow their code and scripts to be used by those that are interested in continuing their work. At least the lack of recent commits in their GitHub seemed slightly troubling, however their domain at stacksherpa.com seemed to be much more informative than most of the other information on their product.
(BillingStack documentation, 2014)

5.2 Zabbix

The author had previous experience on using and configuring Zabbix prior to writing the thesis, therefore it was a logical decision to include a short description when it was mentioned as a capable tool for gathering billing information. However, the purpose in which author used Zabbix previously was not strictly related to billing, but actually it presents a more general approach towards monitoring network. This included gathering information from various devices in network and monitoring amount of data, uptimes and other useful information needed in optimizing the local network (user interface in Figure 6). The author believes that having various components and configuration tools for different metering purposes, Zabbix could be an efficient tool for gathering data; however, according to the author's personal experiences on the software, one major issue for inexperienced Zabbix user is its steep learning curve in the beginning.

The user interface of Zabbix consists of multiple layers of tabs, and some settings and configurations have been placed irrationally. This may be just a problem with inexperience in using the software, however, a more simple user interface would save much time in finding the correct data when the purpose is as specific as only gathering certain data for billing the customer. Due to the general-purpose nature of the software

Zabbix probably has too many components that do not really benefit establishing a billing system, and those components that would be needed, could be easier to use.

The methods for storing data used by Zabbix are MySQL, PostgreSQL, SQLite, Oracle, and IBM DB2, and its backend is written in C language while web frontend has been written in PHP. Zabbix provides several methods for monitoring networks and applications. There are direct checks for availability and responsiveness queries using standard services such as SMTP or HTTP. This method does not require components or software installed on the monitored host.

The second usual method is monitoring the host by using a software agent that is installed on the host. This allows the user to reach statistics such as CPU load, network utilization and available disk space. As an alternative to installing agent software, Zabbix supports monitoring capabilities using certain protocols that have been generally designed for monitoring purposes. Examples of such protocols are SNMP (Simple Network Management Protocol), TCP (Transmission Control Protocol) and ICMP (Internet Control Message Protocol) checks, IPMI, JMX, SSH and Telnet. Even real-time notification mechanisms such as XMPP are supported. In establishing a billing system most of these methods could be used, however, the best options would probably be monitoring using agent software or specifically designed protocols. Direct checks using simple standard services would not likely provide the billing admin with enough information. Installing a user agent for monitoring would be possible provided that platform used fully supports necessary parameters that are needed for establishing a billing database. Using specifically designed monitoring protocols depends a great deal on the protocol in question and how well it is suited for its purpose on the compatibility side.
(Zabbix site, 2014)

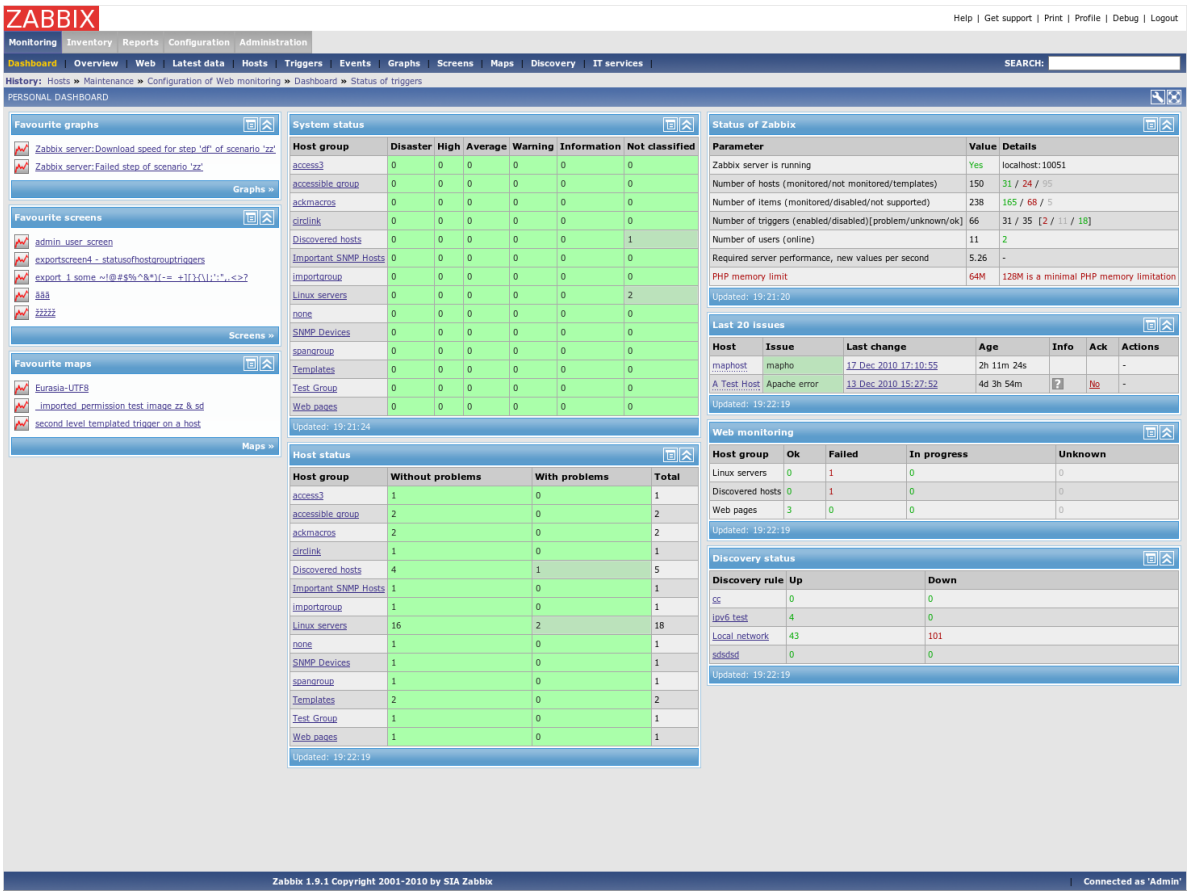

Figure 6: A screenshot of Zabbix user interface
(Zabbix blog, 2012)

5.3 Synaps

Synaps is currently the only AWS CloudWatch compatible monitoring system. There have been no modifications to roadmap that can be found under OpenStack wiki and backlog tells that it is to be integrated with component called Heat that was added while in OpenStack Havana development cycle. Therefore it is safe to assume that this certain method of gathering information is still usable but due to its integration to other components, it would be more efficient to use the most recent versions of components that Synaps has become part of. The latest information on Synaps is that it has been open sourced to the public and as there is no clear indication on Heat documentation that Synaps's functions are fully initialized, the earlier versions of Synaps may be still usable if AWS CloudWatch support is needed.
(Several writers, Synaps – OpenStack, 2014)

5.4 Healthmon and Stacktach

As Healthmon (or Healthnmon) is a cloud resource monitor provided by HP, there are some issues in compatibility between OpenStack API and Healthmon, such as duplicate data collected in both of these components at the same time. StackTach is a debugging and monitoring utility designed for OpenStack developed by Rackspace Hosting. It can be scaled for use with multiple datacenters which may include multi-cell deployments. The way how StackTach functions is based on OpenStack's ability to publish notifications to a RabbitMQ exchange as they occur. This allows requests and logs to be centered into a single location.
(Several writers, Ceilometer/CeilometerAndHealthnmon, 2013/ Several writers, Welcome to StackTach's documentation!, 2014)

6 INTRODUCTION TO OPENSTACK

6.1 OpenStack

OpenStack (Appendix 2 for several installation variations) is a cloud-computing project based on open source code and is managed by the OpenStack Foundation. It has been released under the terms of Apache License and at the time of writing latest stable version is Havana, released on October 17, 2013. OpenStack has been planned to work with various device configurations to establish a cloud service environment. In this project the aim was to demonstrate specifically older hardware's capacity to establish a cloud server environment with low budget machinery. There has been some variation in results; however, in general computers of several years of age have yielded surprisingly good results, provided that certain basic technologies can be found. The most important services needed for the computers are for example virtualization (Intel VT-x, for example) and hardware support for 64-bit operating systems. If hardware only supports 32-bit operating systems, it also limits the bits of virtual machine run in the base hardware.

Managing the OpenStack is currently best carried out with its graphical Horizon tool (user interface from Essex version in Figure 7). Horizon gathers OpenStack management tools under a quite simple browser user interface so that all basic tasks can be

performed in the graphical interface. The UI itself works relatively well, the only cause for more significant problems is usually in the order of configurations and changes. For example, creating key pairs can be quite tricky in the middle of instance launch process. These certain prerequisites should be met before starting to create an instance, and it would help the users a great deal if there was something to help them to remember the order of commands. However, this is not a major problem, concerning the users that usually have to do something with OpenStack. Also, when matters have been familiarized with everything starts to work quite effortlessly. Grizzly version of OpenStack is currently used in this project, and it has been greatly improved from previous versions, at least in Horizon tools department. In the earlier state of the project when using the Essex version the developers were forced to create and use a self-made tool for creation of instances, because if an instance was terminated in that version of Horizon UI, the IP address that was forgotten to detach was permanently lost.

Various software components currently used in OpenStack are Nova, Neutron (previously Quantum), Swift, Cinder, Horizon, Keystone, Glance, Heat and Ceilometer. Nova is the cloud controller and main part of an IaaS system and was also used for simple networking in earlier versions of OpenStack. Swift and Cinder are used for controlling storage, in a way that Swift contains objects and files spread across servers and Cinder provides volume blocks for use in compute instances. Neutron is more complicated and much more capable and customizable system for managing networks and IP addresses than its predecessor Nova. Horizon is the component for Dashboard tool that provides administrators and users with graphical user interface for adjusting settings, automating services, starting instances and so on. Keystone serves as a central directory for user account information, Glance contains images that can be used for starting a virtual instance. Heat is a service introduced with OpenStack Havana and is used to orchestrate cloud applications from templates. The most important component for this thesis is Ceilometer, the telemetry service first introduced as a simple barebone version in OpenStack Grizzly.
(OpenStack.org and documentation, 2013)

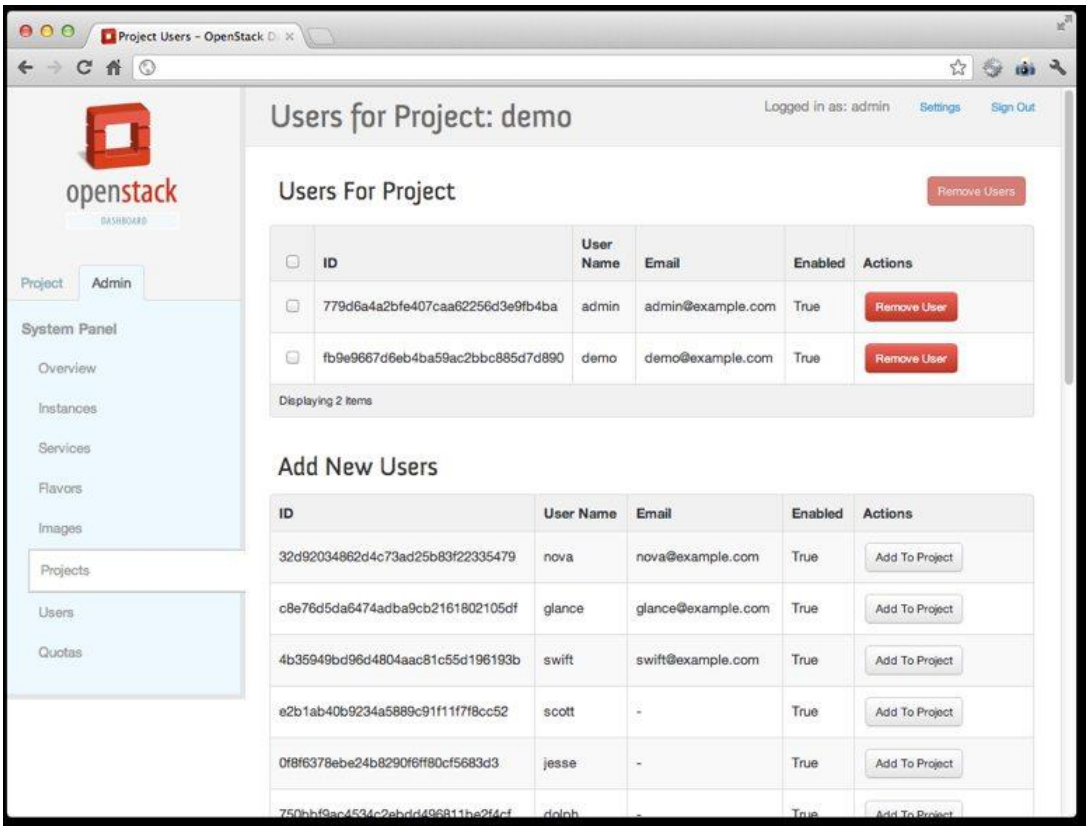

Figure 7: OpenStack user interface, Essex version
(OpenStack site, 2013)

6.2 Ceilometer

Ceilometer project (see Appendix 4 for installation instructions, and Appendix 1 for a full list of measurement components) is a part of OpenStack, which has one simple purpose in larger system, to receive and control data to allow billing interface for virtual instances running in the environment. The first project blueprint was frozen on 15 December 2012, and from there on Ceilometer has been improved among the major updates of OpenStack. Occasionally there have also been bug fixing events to get rid of the largest bugs before remarkable releases. Currently Ceilometer is in a state where it is encouraged to be tested with DevStack tool and probably in the near future it will be properly implemented into OpenStack installation.

The official release should have been during OpenStack Grizzly release; however, it still seemed somewhat experimental at that point. Most of the basic functions prom-

ised were already there, however, only a small part of functions had somewhat been integrated with graphical UI and most of the configurations and getting data output had to be done by scripts. The documentation could also have been much better. However, if all the tools had been finished and fully working when the thesis was started, there would not have been much to research, and therefore the usefulness of this thesis for the project would have been less significant. List of all available meters in Ceilometer is included in the appendices section and API architecture in Figure 8.

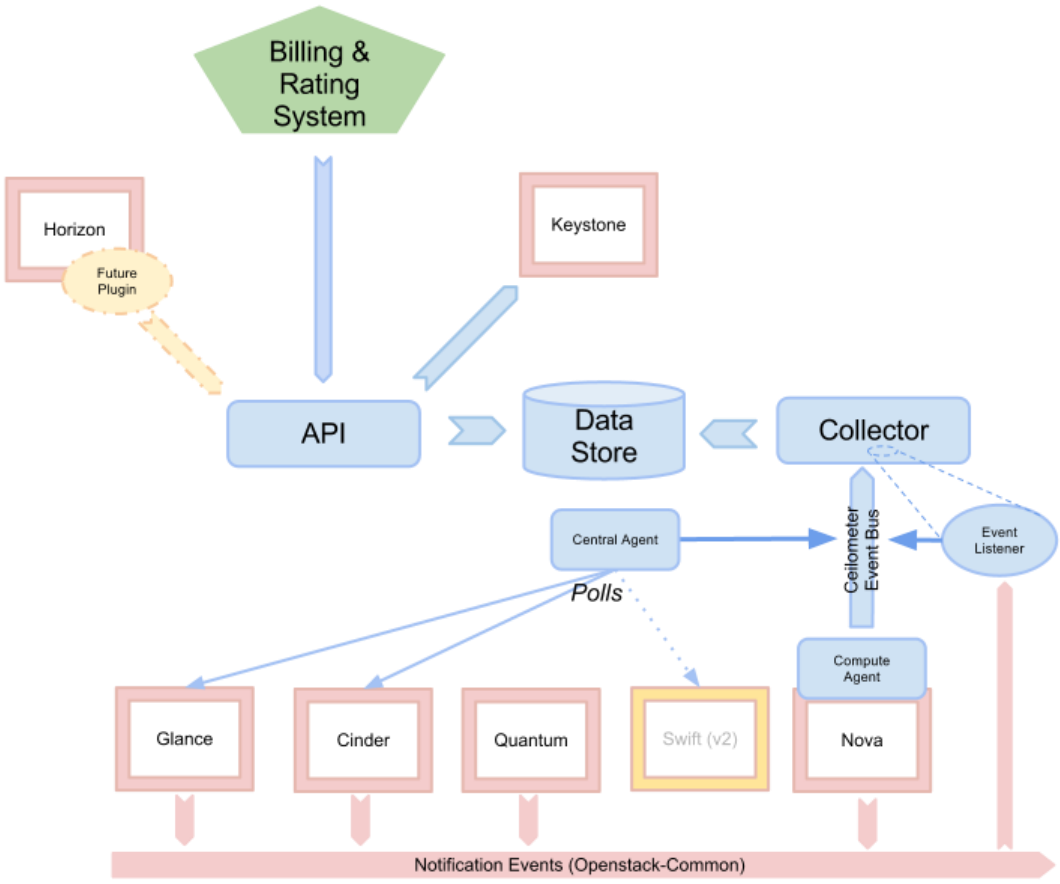

Figure 8: Grizzly API architecture
(Ceilometer Developer Documentation, 2013)

7 SKYNEST PROJECT

7.1 Earlier work within project

While thinking for a proper subject for a thesis the author was presented with several different theses that had been previously completed for the project. These studies included "OpenStack infrastructure monitoring" by Joonas Suuronen and "Configuration management of FreeNEST cloud services" by Ossi Rantapuska. Marko Kaunismäki's thesis also benefitted this research paper, he being a person who had also been working in SkyNEST for his thesis work. His thesis mostly provided with useful definitions on understanding how cloud services basically work. During The author's time in the project there were also many other useful theses to be used as material for studying the environment and implementation of software, however, these particular ones were the closest to this project work in the cloud environment.

As it happened, Cloud Software project was interested in researching the possibilities of building a billing system within OpenStack. This was, after all, very close to the author's own working field as a networking specialist. Before I given a subject for this thesis work certain requirements were defined that should be completed while doing the thesis. Most importantly, the previous studies conducted about this topic had laid a basic groundwork to implement something practically and turn the previous theoretical research work into something actually working. Being a networking student with somewhat lacking skills of coding and not much experience in scripting, the idea of implementing a system for billing sounded somewhat difficult, and the fact that OpenStack's Ceilometer component was far from complete did not help much. Nevertheless, as this was an interesting subject beneficial for the project, the author decided to accept this subject as a thesis project.

7.2 History of SkyNEST testing environment

During the author's time in the project there were several different hardware configurations available with a goal to provide a proof of concept type environment for testing and troubleshooting. Several students worked on the environment during the beginning of the internship in the project, and most of the working time went into im-

proving the tools of maintaining the environment, and mastering the commands to maintain the cloud. At that time the OpenStack version in use was called Essex (see Table 3). Probably the largest problem with Essex had to do with the use of floating IP addresses. The team was not entirely sure if the functionality Essex provided was a bug or if the team ran out of time to implement better connections to instances when using the graphical interface. However, when using graphical user interface, called Dashboard in OpenStack, while terminating an instance with a given floating IP address, Dashboard did not release the IP mapped to instance being terminated. This led into a problem where the mapped IP address was mystically lost. It could not be mapped to another instance and also, it could not be released after the instance attached to it was already terminated.

Table 3: OpenStack version history as of April 2014
(OpenStack site, 2014)

Series	Status	Releases	Date
Icehouse	Under development	Due	Apr 17, 2014
Havana	Current stable release, security-supported	2013.2	Oct 17, 2013
		2013.2.1	Dec 16, 2013
		2013.2.2	Feb 13, 2014
		2013.2.3	Apr 03, 2014
Grizzly	Security-supported	2013.1	Apr 4, 2013
		2013.1.1	May 9, 2013
		2013.1.2	Jun 6, 2013
		2013.1.3	Aug 8, 2013
		2013.1.4	Oct 17, 2013
		2013.1.5	Mar 20, 2014
Folsom	EOL	2012.2	Sep 27, 2012
		2012.2.1	Nov 29, 2012
		2012.2.2	Dec 13, 2012
		2012.2.3	Jan 31, 2013
		2012.2.4	Apr 11, 2013
Essex	EOL	2012.1	Apr 5, 2012
		2012.1.1	Jun 22, 2012
		2012.1.2	Aug 10, 2012
		2012.1.3	Oct 12, 2012
Diablo	EOL	2011.3	Sep 22, 2011
		2011.3.1	Jan 19, 2012
Cactus	Deprecated	2011.2	Apr 15, 2011
Bexar	Deprecated	2011.1	Feb 3, 2011
Austin	Deprecated	2010.1	Oct 21, 2010

When the testing environment was finally upgraded to Folsom, it was found out that the aforementioned issue had been fixed. This is the part when tools developed by the team to release the IP address through command line interface commands were basically abandoned when no longer needed. The tools developed used Fabric, Puppet and Chef functions to work and basically ran scripts that allowed the user to start several instances and while terminating them, it also released the floating IPs attached to instances. This was somewhat tricky but necessary at that time to prevent constant loss of IP addresses while terminating instances. However, the decision to later abandon the knowledge about these useful scripts as the termination functionality in OpenStack had improved in newer versions proved quite problematic later on.

Because Folsom was moved to quite late because of being somewhat afraid of breaking already achieved level of stability with Essex cloud, the upgrade to the next version, Grizzly, came quite quickly. After upgrading to Grizzly the small cloud team quickly noticed that functionality in Grizzly version was even further improved especially in graphical UI. A stable version of OpenStack Havana was released during the writing of this thesis, but the research needed for Ceilometer components was done on Grizzly due to extra time it would have taken to transform one of the cloud environments into new version and get to know the new environment in the middle of research and testing work.

7.3 Testing environment used for testing Ceilometer

The current testing environment consists of three different cloud stacks, each of them established in a different way. The environment is known as the JunkCloud due to old hardware used in it, and it is pictured in Figure 10. As of writing this thesis, one of the environments is not in use due to an accident that happened during the summer; however, as these environments are mostly based on the same versions of OpenStack, there is no real necessity to have several environments running for the thesis purpose. As these environments have been established mostly as proof of concept, after SkyNEST Summer Factory the author was given mostly free hands in working with the environment the best way he saw fit to make research for his thesis work. Two of the environments were running on a 64-bit Ubuntu 13.04 Server, and one on a 12.04 64-bit. The older version was used in one installation because the installation instruc-

tions for this certain script had not been updated in a while and because it instructed the author to install it to 12.04 he decided to play safe and just obey. Also, because he already had two base systems with 13.04 on them there was no specific need for a third one.

However, installing the basic configuration of OpenStack was just the beginning to get something done for the thesis. Luckily it was quickly covered because there was experience on it available from earlier work as an intern during summer. When entering Ceilometer matters became much more complicated. The first part on getting the Ceilometer working was of course the installation. Basically all of the environments established so far were compatible with Ceilometer; however, in order to save time it was reasonable to find the easiest way to get it working just to test first how it starts to work. As usually in open source environment, the documentations were somewhat spread across few different sites and some sources were more complete than others.

However, the official and newest guides directly from OpenStack community were the best bet here, as one would expect. By following the guide some reasonable options to activate Ceilometer components for the OpenStack installations were detected. The manual installation consisted of several parts, including installation of database system of choice, activation of several services and retrieving objects from already existing databases and configurations. For quick testing, there was luckily a much easier method to be found.

Alongside OpenStack itself, a variation called Devstack exists. It is basically a simplified and stripped version of OpenStack that can be modified for various testing purposes. One particular example of what Devstack can easily do and OpenStack cannot is establishing a cloud environment on a single computer. This can of course also be done on OpenStack; however, it is highly unlikely that making it actually work would not be easy. The Devstack works mostly on scripts and it just installs all the essential components on a single node, using the most simplified IP addressing schemes possible and the best of all is that all that user has to do is input a few lines of script to initiate the establishment of environment.

Of course there are a few downsides, because an environment like this has not been designed from ground up to work on a single node. Devstack was very useful during the internship in the project, for the author during his exploration into the OpenStack itself, and during the mentorship to new workers when others were helped to learn what the author already knew. During other times Devstack was also used for testing issues and components that would not most definitely work straight away and needed testing in an environment where system failures would not lead to catastrophic consequences.

A much easier option to installing Ceilometer manually into already working cloud was to activate it using Devstack. In Devstack the activation of Ceilometer basically required only unstacking (temporary shutdown of cloud services), a few changes to configuration files that determine which services are started during establishment, and then running the stack script to get the cloud back up and running. After that, the Ceilometer tool and user defined for it can be seen in graphical user interface and it can be called using scripts. Due to Ceilometer being in relatively early stages of development, there was not yet much implementation to graphical user interface, and even the installation process was somewhat troublesome at best. However it was good to know that installation using Devstack was possible, as it did not require much rollback in terms of using the cloud environment itself and was very easy to reverse in case the user ran into some trouble after establishing the environment.

Figure 9: Testing environment used, OpenStack Grizzly and Ceilometer installed

8 RESEARCH, RESULTS AND CONCLUSION

8.1 Research and testing

After making plans on how to start working on the billing system it was necessary to start estimating the limitations that OpenStack API had and what data could be obtained from the system. The first and easiest part was to take a look at developer documentations to find out the syntax and how to get certain parts of recorded data out in a certain format. As the data itself was in quite a challenging format by default, an online Json parser was used to get data output transformed into a more easily understandable format. During the research on OpenStack Ceilometer documentation the list of other projects using Ceilometer was studied. At the time of writing there was only one project on the list; however, this particular project proved to be very important. While making plans for the billing system research in the beginning, there were small hopes of being able to use OpenStack API so that there would be an easy way of bringing data accessible into graphical Dashboard user interface. The team was

somewhat disappointed that Ceilometer was still in such an early stage that there were only basic APIs accessible and all the data available for access had to be manually transferred into a parser to bring it into an understandable form (Output from online Json parser in Figure 11).

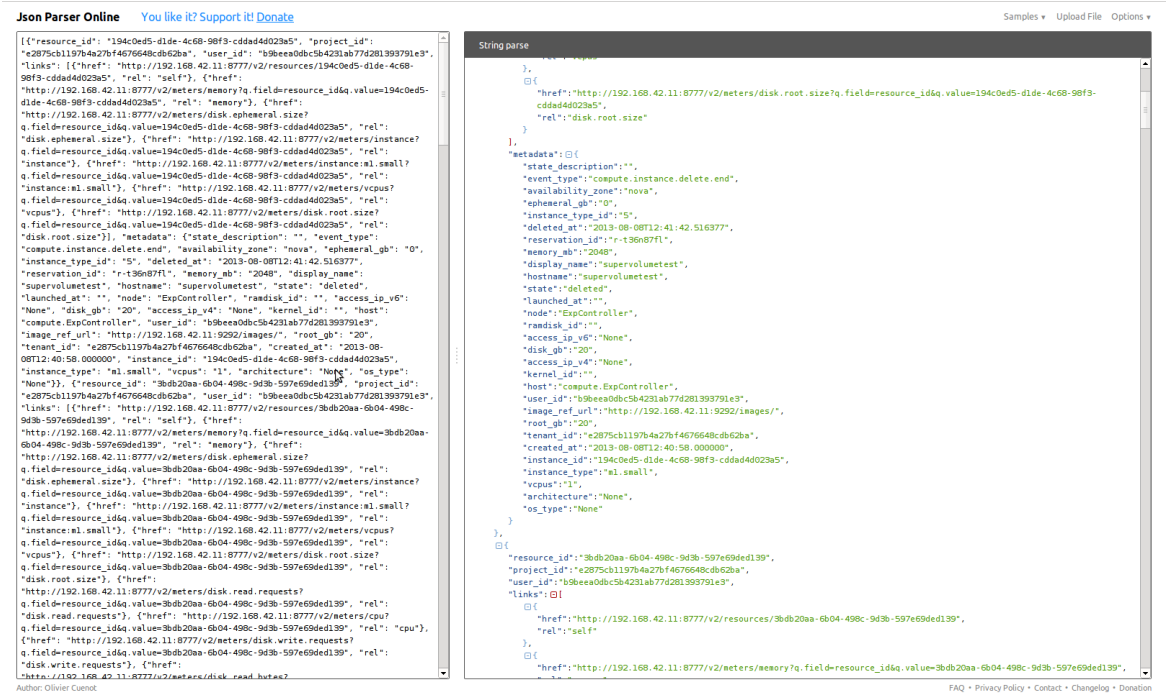

Figure 10: Output from Json parser

When the project using Ceilometer was studied more in detail it was realized that this certain project was already aiming to do the things that the thesis was trying to do. If there had been a simple way of implementing this data directly into graphical interface that would definitely have been the way this project could have done it. This project called BillingStack had already done what was next in the research and testing part of the thesis. At this point the team was somewhat disappointed on the amount of actually beneficial work for the project as was originally intended.

Using the data that BillingStack project had already gathered it was possible to make even more complete documentation for creating a complete billing system in the future for OpenStack; however, at the moment several things indicate that OpenStack Ceilometer API is not yet mature enough for complete and fully functional billing system on itself, and for most administrators planning to establish a billing system, having an API as a basis for their own billing solutions is most likely just enough.

As Ceilometer clearly was a component originally intended purely for metering purposes and not providing the framework for billing itself, it took a bit of extra time during the research on thesis to understand what Ceilometer itself was capable of. When taking a look at Ceilometer schedules and version history, it quickly became clear that developers have also ran into problems that have set them back several times during the development process. If they had been able to follow the original timetable, perhaps Ceilometer would already have been in a state where there would have been more finalized projects that could have been used as a complete billing solution revolving around OpenStack. As of the time of writing, however, there were some solutions that could work in combination, however, not any kind of obvious solution that would combine all the best properties of having an open source platform combined with simplicity of use and loads of customization and properties for billing.

8.2 Results

The original idea for this thesis was to build a simple system for collecting billing data from Ceilometer API that had been recently implemented into OpenStack. However, the quick development cycle among open source projects quickly revealed one of its typical aspects to the author as he was preparing to start the practical work on billing infrastructure. The author found out about a project that had been using Ceilometer as its API, BillingStack, and quickly noticed that the work that they had been doing in their project was just about identical with what the author had been planning to do as his practical part of the thesis.

So, to replace that part the most logical matter to attend to was to move into a more theoretical direction also in this part. What this meant in practice was to delve even deeper into general methods that various billing-based services used in their data collection. The author needed to find out what data was exactly used for billing, where it was gathered and how. In this way the comparisons between differences among competitive billing platforms could be made more easily and even though third party components were used as part of a system, it would be easier to know about potential compatibility issues if the components themselves were well known.

8.3 Conclusion

After the critical change in direction during the beginning of thesis, which involved finding out about how OpenStack API works and how BillingStack communicates with it, the general direction and goal of this thesis became more accurate. After all the most important issue to solve was to find out differences in various billing systems and the ways how they gather data from APIs. With this information it would be easier to choose the best billing system or combination of several software components to establish upon. The research work was mostly centered upon the most popular releases available, due to the fact that a larger user base usually leads to a more actively maintained software and bugs are reported more frequently. Also, if there is a large difference between various methods, it is likely that more popular software has at some point made decisions between various frameworks and has a justifiable reason to end up using a certain solution.

In the end, several capable services were found that could be used for solving the mystery that is optimizing the billing in cloud environment, but no simple, one size fits all –kind of solution did exist as of yet. As a result of thesis cloud billing systems were believed to be divided in two larger classes that set themselves apart from each other greatly; the open source and proprietary approach. Whereas the open source solutions were mostly built around OpenStack compatibility or other open source variant, the proprietary products were trying to stand out on their own as an independent product suite. Proprietary billing solution products had their sites filled with simple slogans and the details were typically hidden away from view, and finding out more about the actual product in action was limited to customers that were ready to schedule a demo for the product. Some exceptions were made, such as live demos on certain days, but mostly the difference between these two different approaches was clear.

When this thesis work was close to being finished, SkyNEST had already changed into another type of project, however, it is still clear that cloud services are changing shape and reforming, and all the information that allows the cloud services to evolve and move forward helps finding unified and common models within the field. At the beginning of writing this thesis, billing in cloud service environment was very complicated, and this thesis work will not change that, however, the author of this thesis hopes that simplifying this subject by disassembling it to smaller components helps to

understand better how billing systems work and what the options are for building a billing system at the time of writing.

REFERENCES

Rantapuska, O. 2012. Configuration management of FreeNEST cloud services. Bachelor's thesis. JAMK University of Applied Sciences, Degree Programme in Software Engineering Technology, ICT. Accessed on 19.10.2013.

Kaunismäki, M. 2012. FreeNest-pilvipalvelun tarjoaminen turvallisesti VPN-tekniikan avulla [Providing FreeNest cloud service securely via VPN technology]. Bachelor's thesis. JAMK University of Applied Sciences, Degree Programme in Software Engineering Technology, ICT. Accessed on 4.11.2013.

O'Neill, M. 2011. SaaS, PaaS, and IaaS: A security checklist for cloud models. Accessed on 4.11.2013. Retrieved from http://www.csoonline.com/article/2126885/cloud-security/saas--paas--and-iaas--a-security-checklist-for-cloud-models.html

Several writers, 2013. Home >> OpenStack Open Source Cloud Computing Software. Accessed on 25.6.2013. Retrieved from http://www.openstack.org/

Several writers, 2013. Ceilometer/Consumers – OpenStack. Accessed on 5.11.2013. Retrieved from https://wiki.openstack.org/wiki/Ceilometer/Consumers

2013. AWS Billing FAQs. Accessed on 17.6.2013. Retrieved from https://aws.amazon.com/billing/

Rackspace Support, 2013. Rackspace Cloud Essentials 1 – Billing Services Overview. Accessed on 17.6.2013. Retrieved from http://www.rackspace.com/knowledge_center/article/rackspace-cloud-essentials-1-billing-services-overview

Several writers, 2013. Ceilometer – OpenStack. Accessed on 18.6.2013. Retrieved from https://wiki.openstack.org/wiki/Ceilometer

Several writers, 2013. Welcome to the Ceilometer developer documentation! Accessed on 25.7.2013. Retrieved from http://docs.openstack.org/developer/ceilometer/

Mell, P., Grance, T. 2011. The NIST Definition of Cloud Computing. Accessed on 22.10.2013. Retrieved from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Several writers, 2013. SSD Cloud Server, VPS Server, Simple Cloud Hosting by DigitalOcean. Accessed on 3.12.2013. Retrieved from https://www.digitalocean.com/

Kiyanchuk, R. 2013. OpenStack Metering Using Ceilometer. Accessed on 8.1.2014. Retrieved from http://www.mirantis.com/blog/openstack-metering-using-ceilometer/

Several writers, 2014. Homepage of Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution. Accessed on 14.1.2014. Retrieved from http://www.zabbix.com/

Several writers, 2014. Synaps – OpenStack. Accessed on 18.1.2014. Retrieved from https://wiki.openstack.org/wiki/Synaps

Netmagic Solutions, 2012. Cloud pricing models. Accessed on 1.3.2014. Retrieved from http://www.asiacloudforum.com/content/cloud-pricing-models

Brandon Butler, 2013. Cloud pricing: Complex and complicated. Accessed on 6.4.2014. Retrieved from http://cw.com.hk/feature/cloud-pricing-complex-and-complicated

Several writers, 2014. Enterprise Subscription Billing and Recurring Revenue Solutions. Accessed on 6.4.2014. Retrieved from http://www.ariasystems.com/

Several writers, 2014. Subscription Billing, Recurring Revenue, Recurring Payments and Recurring Billing Solutions | Zuora. Accessed on 15.4.2014. Retrieved from http://www.zuora.com/

Several writers, 2014. Welcome to BillingStack's documentation! Accessed on 27.4.2014. Retrieved from http://billingstack.readthedocs.org/en/latest/

Several writers, 2013. Ceilometer/CeilometerAndHealthnmon. Accessed on 27.4.2014. Retrieved from https://wiki.openstack.org/wiki/Ceilometer/CeilometerAndHealthnmon

Several writers, 2014. Welcome to StackTach's documentation! Accessed on 27.4.2014. Retrieved from http://stacktach.readthedocs.org/en/latest/

Several writers, 2014. Talligent | OpenBook Metering & Billing for the Cloud. Accessed on 27.4.2014. Retrieved from http://talligent.com/

Several writers, 2014. IT Managed Service Providers with Datacenters in India – Netmagic. Accessed on 27.4.2014. Retrieved from http://www.netmagicsolutions.com/

Christophe Primault, 2011. Which Billing and Subscriptions App to Choose for your Business? Accessed on 1.5.2014. Retrieved from http://www.getapp.com/blog/billing-subscriptions-saas/

Alex Bligh, 2012. A few Moments on Cloud Billing. Accessed on 2.5.2014. Retrieved from http://www.flexiant.com/2012/03/13/a-few-moments-on-cloud-billing/

Several writers, 2014. Dimension Data - A Cloud Solution Provider | Dimension Data Europe. Accessed on 3.5.2014. Retrieved from http://cloud.dimensiondata.com/eu/en/

Several writers, 2014. Subscription and Enterprise Billing | MetraTech. Accessed on 4.5.2014. Retrieved from http://www.metratech.com/

Several writers, 2011. ScienceLogic Blog on Cloud Monitoring and IT Infrastructure Management. Accessed on 4.5.2014. Retrieved from http://blog.sciencelogic.com/

Several writers, 2014. Spreedly. Accessed on 4.5.2014. Retrieved from https://spreedly.com/

Several writers, 2014. Online Recurring Billing Manager | CheddarGetter. Accessed on 4.5.2014. Retrieved from https://cheddargetter.com/

Several writers, 2014. Recurring Billing, Subscription Billing, Web 2.0 and SaaS Billing – Chargify. Accessed on 4.5.2014. Retrieved from http://chargify.com/

Several writers, 2014. Fusebill | Recurring Billing and Payments Solution. Accessed on 4.5.2014. Retrieved from http://www.fusebill.com/

Several writers, 2014. Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services. Accessed on 8.5.2014. Retrieved from http://azure.microsoft.com/

APPENDICES

Appendix 1: List of OpenStack Ceilometer measurement components

(http://docs.openstack.org/developer/ceilometer/measurements.html, accessed 16.4.2014)

Measurements

Three type of meters are defined in ceilometer:

Type	Definition
Cumulative	Increasing over time (instance hours)
Gauge	Discrete items (floating IPs, image uploads) and fluctuating values (disk I/O)
Delta	Changing over time (bandwidth)

Units

1. Whenever a volume is to be measured, SI approved units and their approved symbols or abbreviations should be used. Information units should be expressed in bits (‘b’) or bytes (‘B’).
2. For a given meter, the units should NEVER, EVER be changed.
3. When the measurement does not represent a volume, the unit description should always described WHAT is measured (i.e.: apples, disk, routers, floating IPs, etc.).
4. When creating a new meter, if another meter exists measuring something similar, the same units and precision should be used.
5. Meters and samples should always document their units in Ceilometer (API and Documentation) and new sampling code should not be merged without the appropriate documentation.

Dimension	Unit	Abbreviations	Note
None	N/A		Dimension-less variable
Volume	byte	B	
Time	seconds	s	

Here are the meter types by components that are currently implemented:

Compute (Nova)

All meters are related to the guest machine, not the host.

Name	**Type***	**Unit**	**Resource**	**Origin ****	**Support *****	**Note**
instance	g	instance	inst ID	both	1, 2, 3	Existence of instance
instance:<type>	g	instance	inst ID	both	1, 2, 3	Existence of instance <type> (open-stack types)
memory	g	MB	inst ID	n	1, 2	Volume of RAM allocat-ed in MB
memory.usage	g	MB	inst ID	p	3	Volume of RAM used in MB
cpu	c	ns	inst ID	p	1, 2	CPU time used
cpu_util	g	%	inst ID	p	1, 2, 3	Average CPU utilisation
vcpus	g	vcpu	inst ID	n	1, 2	Number of VCPUs
disk.read.requests	c	request	inst ID	p	1, 2	Number of read requests
disk.read.requests.rate	g	request/s	inst ID	p	1, 2, 3	Average rate of read re-quests per second
disk.write.requests	c	request	inst ID	p	1, 2	Number of write requests
disk.write.requests.rate	g	request/s	inst ID	p	1, 2, 3	Average rate of write re-quests per second
disk.read.bytes	c	B	inst ID	p	1, 2	Volume of reads in B
disk.read.bytes.rate	g	B/s	inst ID	p	1, 2, 3	Average rate of reads in B per second
disk.write.bytes	c	B	inst ID	p	1, 2	Volume of writes in B
disk.write.bytes.rate	g	B/s	inst ID	p	1, 2, 3	Average vol-ume of writes in B per sec-ond
disk.root.size	g	GB	inst ID	n	1, 2	Size of root disk in GB
disk.ephemeral.size	g	GB	inst ID	n	1, 2	Size of ephemeral

Name	Type*	Unit	Resource	Origin**	Support***	Note
						disk in GB
network.incoming.bytes	c	B	iface ID	p	1, 2	Number of incoming bytes on a VM network interface
network.incoming.bytes.rate	g	B/s	iface ID	p	1, 2, 3	Average rate per sec of incoming bytes on a VM network interface
network.outgoing.bytes	c	B	iface ID	p	1, 2	Number of outgoing bytes on a VM network interface
network.outgoing.bytes.rate	g	B/s	iface ID	p	1, 2, 3	Average rate per sec of outgoing bytes on a VM network interface
network.incoming.packets	c	packet	iface ID	p	1, 2	Number of incoming packets on a VM network interface
network.incoming.packets.rate	g	packet/s	iface ID	p	1, 2, 3	Average rate per sec of incoming packets on a VM network interface
network.outgoing.packets	c	packet	iface ID	p	1, 2	Number of outgoing packets on a VM network interface
network.outgoing.packets.rate	g	packet/s	iface ID	p	1, 2, 3	Average rate per sec of outgoing packets on a VM network interface

Legend:

*

[g]: gauge

[c]: cumulative

**

[p]: pollster

[n]: notification

***

[1]: Libvirt support

[2]: HyperV support

[3]: Vsphere support

Contributors are welcome to extend other virtualization backends' meters or complete the existing ones.

The meters below are related to the host machine.

By default, Nova will not collect the following meters related to the host compute node machine. Nova option 'compute_monitors = ComputeDriverCPUMonitor' should be set in nova.conf to enable meters.

Name	**Type**	**Unit**	**Resource**	**Origin**	**Note**
compute.node.cpu.frequency	Gauge	MHz	host ID	notification	CPU frequency
compute.node.cpu.kernel.time	Cumulative	ns	host ID	notification	CPU kernel time
compute.node.cpu.idle.time	Cumulative	ns	host ID	notification	CPU idle time
compute.node.cpu.user.time	Cumulative	ns	host ID	notification	CPU user mode time
compute.node.cpu.iowait.time	Cumulative	ns	host ID	notification	CPU I/O wait time

Name	Type	Unit	Resource	Origin	Note
compute.node.cpu.kernel.percent	Gauge	%	host ID	notification	CPU kernel percentage
compute.node.cpu.idle.percent	Gauge	%	host ID	notification	CPU idle percentage
compute.node.cpu.user.percent	Gauge	%	host ID	notification	CPU user mode percentage
compute.node.cpu.iowait.percent	Gauge	%	host ID	notification	CPU I/O wait percentage
compute.node.cpu.percent	Gauge	%	host ID	notification	CPU utilization

Network (Neutron)

Name	Type	Unit	Resource	Origin	Note
network	Gauge	network	netw ID	notification	Existence of network
network.create	Delta	network	netw ID	notification	Creation requests for this network
network.update	Delta	network	netw ID	notification	Update requests for this network
subnet	Gauge	subnet	subnt ID	notification	Existence of subnet
subnet.create	Delta	subnet	subnt ID	notification	Creation requests for this subnet
subnet.update	Delta	subnet	subnt ID	notification	Update requests for this subnet
port	Gauge	port	port ID	notification	Existence of port
port.create	Delta	port	port ID	notification	Creation requests for this port
port.update	Delta	port	port ID	notification	Update requests for this port
router	Gauge	router	rtr ID	notification	Existence of router
router.create	Delta	router	rtr ID	notification	Creation requests for this router
router.update	Delta	router	rtr ID	notification	Update requests for this router
ip.floating	Gauge	ip	ip ID	both	Existence of floating ip
ip.floating.create	Delta	ip	ip ID	notification	Creation requests for this floating ip
ip.floating.update	Delta	ip	ip ID	notification	Update requests for this floating ip

Image (Glance)

Name	Type	Unit	Resource	Origin	Note
image	Gauge	image	image ID	both	Image polling -> it (still) exists

Name	Type	Unit	Resource	Origin	Note
image.size	Gauge	B	image ID	both	Uploaded image size
image.update	Delta	image	image ID	notification	Number of update on the image
image.upload	Delta	image	image ID	notification	Number of upload of the image
image.delete	Delta	image	image ID	notification	Number of delete on the image
image.download	Delta	B	image ID	notification	Image is downloaded
image.serve	Delta	B	image ID	notification	Image is served out

Volume (Cinder)

Name	Type	Unit	Resource	Origin	Note
volume	Gauge	volume	vol ID	notification	Existence of volume
volume.size	Gauge	GB	vol ID	notification	Size of volume

Make sure Cinder is properly configured first: see *Installing Manually*.

Object Storage (Swift)

Name	Type	Unit	Resource	Origin	Note
storage.objects	Gauge	object	store ID	pollster	Number of objects
storage.objects.size	Gauge	B	store ID	pollster	Total size of stored objects
storage.objects.containers	Gauge	container	store ID	pollster	Number of containers
storage.objects.incoming.bytes	Delta	B	store ID	notification	Number of incoming bytes
storage.objects.outgoing.bytes	Delta	B	store ID	notification	Number of outgoing bytes
storage.api.request	Delta	request	store ID	notification	Number of API requests against swift
storage.containers.objects	Gauge	object	str ID/cont	pollster	Number of objects in container
storage.containers.objects.size	Gauge	B	str ID/cont	pollster	Total size of stored objects in container

In order to use storage.objects.incoming.bytes and storage.outgoing.bytes, one must configure Swift as described in *Installing Manually*. Note that they may not be updated right after an upload/download, since Swift takes some time to update the container properties.

Orchestration (Heat)

Name	Type	Unit	Resource	Origin	Note
stack.create	Delta	stack	stack ID	notification	Creation requests for a stack successful
stack.update	Delta	stack	stack ID	notification	Updating requests for a stack successful
stack.delete	Delta	stack	stack ID	notification	Deletion requests for a stack successful
stack.resume	Delta	stack	stack ID	notification	Resuming requests for a stack successful
stack.suspend	Delta	stack	stack ID	notification	Suspending requests for a stack successful

To enable Heat notifications configure Heat as described in *Installing Manually*.

Energy (Kwapi)

Name	Type	Unit	Resource	Origin	Note
energy	Cumulative	kWh	probe ID	pollster	Amount of energy
power	Gauge	W	probe ID	pollster	Power consumption

Network (From SDN Controller)

These meters based on OpenFlow Switch metrics. In order to enable these meters, each driver needs to be configured.

Meter	Type	Unit	Resource	Origin	Note
switch	Gauge	switch	switch ID	pollster	Existence of switch
switch.port	Gauge	port	switch ID	pollster	Existence of port
switch.port.receive.packets	Cumulative	packet	switch ID	pollster	Received Packets
switch.port.transmit.packets	Cumulative	packet	switch ID	pollster	Transmitted Packets
switch.port.receive.bytes	Cumulative	B	switch ID	pollster	Received Bytes
switch.port.transmit.bytes	Cumulative	B	switch ID	pollster	Transmitted Bytes
switch.port.receive.drops	Cumulative	packet	switch ID	pollster	Receive Drops
switch.port.transmit.drops	Cumulative	packet	switch ID	pollster	Transmit Drops
switch.port.receive.errors	Cumulative	packet	switch ID	pollster	Receive Errors

Meter	Type	Unit	Resource	Origin	Note
switch.port.transmit.errors	Cumulative	packet	switch ID	pollster	Transmit Errors
switch.port.receive.frame_error	Cumulative	packet	switch ID	pollster	Receive Frame Alignment Errors
switch.port.receive.overrun_error	Cumulative	packet	switch ID	pollster	Receive Overrun Errors
switch.port.receive.crc_error	Cumulative	packet	switch ID	pollster	Receive CRC Errors
switch.port.collision.count	Cumulative	count	switch ID	pollster	Collisions
switch.table	Gauge	table	switch ID	pollster	Duration of Table
switch.table.active.entries	Gauge	entry	switch ID	pollster	Active Entries
switch.table.lookup.packets	Gauge	packet	switch ID	pollster	Packet Lookups
switch.table.matched.packets	Gauge	packet	switch ID	pollster	Packet Matches
switch.flow	Gauge	flow	switch ID	pollster	Duration of Flow
switch.flow.duration.seconds	Gauge	s	switch ID	pollster	Duration(seconds)
switch.flow.duration.nanoseconds	Gauge	ns	switch ID	pollster	Duration(nanoseconds)
switch.flow.packets	Cumulative	packet	switch ID	pollster	Received Packets
switch.flow.bytes	Cumulative	B	switch ID	pollster	Received Bytes

Dynamically retrieving the Meters via ceilometer client

To retrieve the available meters that can be queried given the actual resource instances available, use the `meter-list` command:

```
$ ceilometer meter-list -s openstack

+-----------+-------+----------------------------------+---------+----------------------------------+

| Name      | Type  | Resource ID                      | User ID | Project ID                       |

+-----------+-------+----------------------------------+---------+----------------------------------+
```

```
| image       | gauge | 09e84d97-8712-4dd2-bcce-45970b2430f7 |          |
57cf6d93688e4d39bf2fe3d
```

Appendix 2: Methods for installing OpenStack

Method 1: Official OpenStack documentation, manual

http://docs.openstack.org/grizzly/openstack-compute/install/apt/content/

Method 2: mseknibilel's method (multi node/ single node), manual

https://github.com/mseknibilel/OpenStack-Grizzly-Install-Guide

Method 3: jedipunkz's method, scripted

https://github.com/jedipunkz/openstack_grizzly_install

Appendix 3: Installing BillingStack manually

(http://billingstack.readthedocs.org/en/latest/install/manual.html, accessed 3.5.2014)

Common Steps

Note

The below operations should take place underneath your <project>/etc folder.

1. Install system package dependencies (Ubuntu):
2. $ apt-get install python-pip python-virtualenv
3. $ apt-get install rabbitmq-server mysql-server

$ apt-get build-dep python-lxml

4. Clone the BillingStack repo off of Github:
5. $ git clone https://github.com/billingstack/billingstack.git

$ cd billingstack

6. Setup virtualenv:

Note

This is to not interfere with system packages etc.

$ virtualenv –no-site-packages .venv $ . .venv/bin/activate

4. Install BillingStack and it's dependencies:
5. $ pip install -rtools/setup-requires -rtools/pip-requires -rtools/pip-options

$ python setup.py develop

Copy sample configs to usable ones, inside the *etc* folder do:

$ ls *.sample | while read f; do cp $f $(echo $f | sed "s/.sample$//g"); done

Installing Central

Note

This is needed because it is the service that the API and others uses to communicate with to do stuff in the Database.

1. See Common Steps before proceeding.
2. Configure the *central* service:

Change the wanted configuration settings to match your environment, the file is in the *etc* folder:

$ vi etc/billingstack.conf

Refer to the configuration file for details on configuring the service.

3. Create the DB for *central*:

$ python tools/resync_billingstack.py

4. Now you might want to load sample data for the time being:

$ python tools/dev_samples.py

5. Start the central service:

$ billingstack-central

Installing the API

Note

The API Server needs to able to talk via MQ to other services.

1. See Common Steps before proceeding.
2. Configure the *api* service:

Change the wanted configuration settings to match your environment, the file is in the *etc* folder:

$ vi billingstack.conf

Refer to the configuration file for details on configuring the service.

3. Start the API service:

$ billingstack-api

Appendix 4: Installing Ceilometer

(http://docs.openstack.org/developer/ceilometer/install/manual.html, accessed 4.5.2014, included only until installing the collector because the necessity of following parts depend on the build used)

Installing Manually

Storage Backend Installation

This step is a prerequisite for the collector, notification agent and API services. You may use one of the listed database backends below to store Ceilometer data.

Note

Please notice, MongoDB (and some other backends like DB2 and HBase) require pymongo to be installed on the system. The required minimum version of pymongo is 2.4.

MongoDB

The recommended Ceilometer storage backend is *MongoDB*. Follow the instructions to install the MongoDB package for your operating system, then start the service. The required minimum version of MongoDB is 2.4.

To use MongoDB as the storage backend, change the 'database' section in ceilometer.conf as follows:

```
[database]

connection = mongodb://username:password@host:27017/ceilometer
```

SQLalchemy-supported DBs

You may alternatively use *MySQL* (or any other SQLAlchemy-supported DB like*PostgreSQL*).

In case of SQL-based database backends, you need to create a *ceilometer* database first and then initialise it by running:

```
ceilometer-dbsync
```

To use MySQL as the storage backend, change the ‘database’ section in ceilometer.conf as follows:

```
[database]

connection = mysql://username:password@host/ceilometer?charset=utf8
```

HBase

HBase backend is implemented to use HBase Thrift interface, therefore it is mandatory to have the HBase Thrift server installed and running. To start the Thrift server, please run the following command:

```
${HBASE_HOME}/bin/hbase thrift start
```

The implementation uses HappyBase, which is a wrapper library used to interact with HBase via Thrift protocol. You can verify the thrift connection by running a quick test from a client:

```
import happybase



conn = happybase.Connection(host=$hbase-thrift-server, port=9090, table_prefix=None)

print conn.tables() # this returns a list of HBase tables in your HBase server
```

Note

HappyBase version 0.5 or greater is required. Additionally, version 0.7 is not currently supported.

In case of HBase, the needed database tables (*project*, *user*, *resource*, *meter*, *alarm*,*alarm_h*) should be created manually with *f* column family for each one.

To use HBase as the storage backend, change the ‘database’ section in ceilometer.conf as follows:

```
[database]

connection = hbase://hbase-thrift-host:9090
```

DB2

DB2 installation should follow fresh IBM DB2 NoSQL installation docs.

To use DB2 as the storage backend, change the ‘database’ section in ceilometer.conf as follows:

```
[database]

connection = db2://username:password@host:27017/ceilometer
```

Installing the notification agent

1. If you want to be able to retrieve image samples, you need to instruct Glance to send notifications to the bus by changing notifier_strategy to rabbit or qpid in glance-api.conf and restarting the service.
2. If you want to be able to retrieve volume samples, you need to instruct Cinder to send notifications to the bus by changing notification_driver tocinder.openstack.common.notifier.rpc_notifier and control_exchange to cinder, before restarting the service.
3. In order to retrieve object store statistics, ceilometer needs access to swift with ResellerAdminrole. You should give this role to your os_username user for tenant os_tenant_name:

```
4.  $ keystone role-create --name=ResellerAdmin
5.  +----------+----------------------------------+
6.  | Property |              Value               |
7.  +----------+----------------------------------+
8.  |    id    | 462fa46c13fd4798a95a3bfbe27b5e54 |
9.  |   name   |          ResellerAdmin           |
```

```
+----------+----------------------------------+

$ keystone user-role-add --tenant_id $SERVICE_TENANT \
                         --user_id $CEILOMETER_USER \
                         --role_id 462fa46c13fd4798a95a3bfbe27b5e54
```

You'll also need to add the Ceilometer middleware to Swift to account for incoming and outgoing traffic, by adding these lines to `/etc/swift/proxy-server.conf`:

```
[filter:ceilometer]
use = egg:ceilometer#swift
```

And adding `ceilometer` in the `pipeline` of that same file, right before `proxy-server`.

Additionally, if you want to store extra metadata from headers, you need to set `metadata_headers`so it would look like:

```
[filter:ceilometer]
use = egg:ceilometer#swift
metadata_headers = X-FOO, X-BAR
```

Note

Please make sure that ceilometer's logging directory (if it's configured) is read and write accessible for the user swift is started by.

14. Clone the ceilometer git repository to the management server:

```
$ cd /opt/stack
```

```
$ git clone https://git.openstack.org/openstack/ceilometer.git
```

16. As a user with `root` permissions or `sudo` privileges, run the ceilometer installer:

```
$ cd ceilometer
```

```
$ sudo python setup.py install
```

18. Copy the sample configuration files from the source tree to their final location.

```
$ mkdir -p /etc/ceilometer
$ cp etc/ceilometer/*.json /etc/ceilometer
$ cp etc/ceilometer/*.yaml /etc/ceilometer
```

```
$ cp etc/ceilometer/ceilometer.conf.sample /etc/ceilometer/ceilometer.conf
```

22. Edit `/etc/ceilometer/ceilometer.conf`
    1. Configure RPC

       Set the RPC-related options correctly so ceilometer's daemons can communicate with each other and receive notifications from the other projects.

       In particular, look for the `*_control_exchange` options and make sure the names are correct. If you did not change the `control_exchange` settings for the other components, the defaults should be correct.

       Note

       Ceilometer makes extensive use of the messaging bus, but has not yet been tested with ZeroMQ. We recommend using Rabbit or qpid for now.

2. Set the metering_secret value.

   Set the metering_secret value to a large, random, value. Use the same value in all ceilometer configuration files, on all nodes, so that messages passing between the nodes can be validated.

23. Refer to *Configuration Options* for details about any other options you might want to modify before starting the service.
24. Start the notification daemon.

```
$ ceilometer-agent-notification
```

Note

The default development configuration of the collector logs to stderr, so you may want to run this step using a screen session or other tool for maintaining a long-running program in the background.

Installing the collector

1. Clone the ceilometer git repository to the management server:

2. $ cd /opt/stack

```
$ git clone https://git.openstack.org/openstack/ceilometer.git
```

3. As a user with root permissions or sudo privileges, run the ceilometer installer:

4. $ cd ceilometer

```
$ sudo python setup.py install
```

5. Copy the sample configuration files from the source tree to their final location.

```
6.  $ mkdir -p /etc/ceilometer
7.  $ cp etc/ceilometer/*.json /etc/ceilometer
8.  $ cp etc/ceilometer/*.yaml /etc/ceilometer
```

```
$ cp etc/ceilometer/ceilometer.conf.sample /etc/ceilometer/ceilometer.conf
```

9. Edit `/etc/ceilometer/ceilometer.conf`
    1. Configure RPC

       Set the RPC-related options correctly so ceilometer's daemons can communicate with each other and receive notifications from the other projects.

       In particular, look for the `*_control_exchange` options and make sure the names are correct. If you did not change the `control_exchange` settings for the other components, the defaults should be correct.

       Note

       Ceilometer makes extensive use of the messaging bus, but has not yet been tested with ZeroMQ. We recommend using Rabbit or qpid for now.

    2. Set the `metering_secret` value.

       Set the `metering_secret` value to a large, random, value. Use the same value in all ceilometer configuration files, on all nodes, so that messages passing between the nodes can be validated.

10. Refer to *Configuration Options* for details about any other options you might want to modify before starting the service.
11. Start the collector.

```
$ ceilometer-collector
```

Note

The default development configuration of the collector logs to stderr, so you may want to run this step using a screen session or other tool for maintaining a long-running program in the background.