An Implementation of the Sun Cloud API for the OpenNebula Toolkit

Javier García Núñez Jorge Hernández Sánchez Daniel Molina Aranda

Proyecto de Sistemas Informáticos
Facultad de Informática

Director
Rubén Manuel Santiago Montero
Universidad Complutense de Madrid
Curso 2009 - 2010

Índice general

Declaración de conformidad

Los alumnos:

Javier García Núñez , Jorge Hernández Sánchez , Daniel Molina Aranda aquí firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, 2 de Julio de 2010

Javier García Núñez	**Jorge Hernández Sánchez**	**Daniel Molina Aranda**

Agradecimientos

Javier:

Este Proyecto pone punto y final a una de las mejores etapas de mi vida. Quiero dar las gracias a todas las personas que habéis estado a mi lado, por todos los momentos que hemos compartido y que nunca olvidaré.

A mis compañeros de Proyecto y otras muchas asignaturas les agradezco su amistad y todo el trabajo que hemos sido capaces de llevar a cabo juntos.

La realización de este trabajo habría sido más costosa de no ser por la ayuda de nuestro director Rubén Manuel Santiago Montero y de "los sabios" . Gracias por ayudarnos a salir de los atascos.

Alguien dijo: " Para tener éxito hay que confiar en la amistad ". Yo lo transformo en : Para tener éxito hay que saber vincularse a personas que te quieran, te aprecien y se preocupen por ti. Gracias a ellos he conseguido mejorar día tras día.

Gracias a todos.

Jorge:

Quiero agradecer todo el trabajo realizado por mis compañeros de proyecto, ya que este año hemos compartido muchas asignaturas y aunque hayamos tenido alguna que otra discusión, siempre recordaré todos los momentos buenos que tuvimos.

También me gustaría agradecer a todas aquellas personas que en mayor o menor medida me han acompañado en la carrera, y aunque con muchas de ellas ya haya perdido la relación siempre recordaré el tiempo que pasamos juntos.

Por último, también quisiera agradecer el trabajo realizado por nuestro director de proyecto Rubén y todos aquellos profesores con los que coincidí en la carrera, puesto que en mayor o menor medida tienen mucho que ver con que hoy yo esté aquí.

Por todos ellos, soy quien soy. Así que, Gracias

Daniel:

> "Las personas mayores me aconsejaron abandonar el dibujo de serpientes boas, ya fueran abiertas o cerradas, y poner más interés en la geografía, la historia, el cálculo y la gramática. De esta manera a la edad de seis años abandoné una magnífica carrera de pintor. Había quedado desilusionado por el fracaso de mis dibujos número 1 y número 2. Las personas mayores nunca pueden comprender algo por sí solas y es muy aburrido para los niños tener que darles una y otra vez explicaciones."
>
> *E*l Principito (1943), Novela de Antoine de Saint-Exupéry. Capitulo I

Gracias a todos por explicar ó escuchar; por estar ahí cuando hacía falta y cuando no, también.

Gracias en especial a ti Ana, parte fundamental en el camino que me ha llevado hasta aquí.

Gracias porque, como alguien dijo una vez, un pilar sin algo que sujetar no sirve de nada. Inevitablemente una cosa depende de la otra, al igual que yo dependo de todos vosotros.

Gracias.

Resumen

OpenNebula Sun Cloud Server es un servicio web que le permite lanzar y gestionar máquinas virtuales en su instalación de OpenNebula a través de la API Sun. El servicio de Sun Cloud se implementa sobre la nueva API de OpenNebula (OCA) capa que expone todas las capacidades de un cloud privado, y Sinatra, un framework Web ampliamente utilizado.

OpenNebula es un conjunto de herramientas de código abierto para crear fácilmente cualquier tipo de cloud: privados, públicos y mixtos. OpenNebula ha sido diseñado para ser integrado con cualquier solución de almacenamiento en red y para encajar en cualquier centro de datos existente.

Las llamadas implementadas en este proyecto son:

1. Peticiones a los recursos de un Cluster Server:
   a) Obtener cluster
   b) Crear VM
   c) Crear Vnet
2. Peticiones al VDC:
   a) Obtener VDC
   b) Crear Dirección Pública
   c) Crear Volumen
   d) Obtener catálogo
3. Peticiones sobre una máquina virtual:
   a) Obtener VM
   b) VM Eliminar
   c) VM Control
4. Peticiones sobre una red virtual:
   a) Obtener Vnet
   b) Eliminar Vnet
5. Peticiones a volúmenes:
   a) Obtener volumen

Además se ha implementado un cliente de línea de comando con el fin de comunicarse con el servidor de una manera amistosa.

Asbtract

The OpenNebula Sun Cloud Server is a web service that enables you to launch and manage virtual machines in your OpenNebula installation through the Sun API. The Sun Cloud web service is implemented upon the new OpenNebula Cloud API (OCA) layer that exposes the full capabilities of an OpenNebula private cloud; and Sinatra, a widely used light web framework.

OpenNebula is an open-source toolkit to easily build any type of cloud: private, public and hybrid. OpenNebula has been designed to be integrated with any networking and storage solution and so to fit into any existing data center.

The calls implemented in this project are:

1. Requests to Cluster Resources.
   a) Get Cluster
   b) Create VM
   c) Create Vnet
2. Requests to VDC Resources.
   a) Get VDC
   b) Create Public Address
   c) Create Volume
   d) Get Catalog
3. Requests to VM Resources.
   a) Get VM
   b) Delete VM
   c) Control VM
4. Requests to VNet Resources.
   a) Ger Vnet
   b) Delete Vnet
5. Requests to Volume Resources.
   a) Get volumen

In addition to a command line client was implemented in order to communicate with the server in a human friendly way.

Capítulo 1

Introducción y objetivos

Este proyecto gira en torno a la implementación de una nueva API que cumpliera las especificaciones propuestas por el Sun-Cloud Api para la creación y gestión de recursos de cloud incluyendo funciones de computo y almacenamiento. Se trata de un servicio web que permite lanzar y gestionar máquinas Virtuales, redes virtuales y volúmenes.

El Sun Cloud está implementado sobre un sistema de gestión de nubes privadas, públicas y mixtas, OpenNebula y Sinatra, un web framework.

Una de sus muchas características que se irán explicando a lo largo de la documentación es que usa el paradigma REST caracterizado entre otras cosas por un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: POST, GET, PUT y DELETE y una sintaxis universal para identificar los recursos. Además cada recurso es direccionable únicamente a través de su URI.

Todos los servicios ofrecidos por el Sun Cloud son a través de la Web. Otro detalle característico de esta API es el uso de JSON para el intercambio de datos a través de Internet.

El API de Sun se puede encuadrar dentro de lo que se conoce como Infraestrucutre as a Service (IaaS) cuya traducción al castellano es Infraestructura como Servicio. La idea es que los clientes en vez de adquirir servidores, espacio en un centro de datos o equipamiento de redes, compran todos estos recursos a un proveedor de servicios externo. Estos servicios se ofrecen de manera íntegra a través de la Web.
Algunos ejemplos existentes de proveedores de servicio son GoGrid, Rimuhosting, Terremark y Dreamhost.

A lo largo de esta documentación se va presentando con más detalle algunos puntos ya mencionados.
En un primer momento se busca introducir al lector en los conceptos básicos explicando qué es exactamente la tecnología cloud y los modelos que se diferencian actualmente (IaaS, PaaS y SaaS) . Cuál es la estructura básica de una aplicación WEB, el paradigma REST para interfaces cloud. EC2 y OCCI.

Una vez abordados estos temas introductorios, se presenta con más detalle la aplicación desarrollada exponiendo diagramas de casos de uso, arquitectura utilizada y diseño del sistema.

Capítulo 2

Cloud Computing

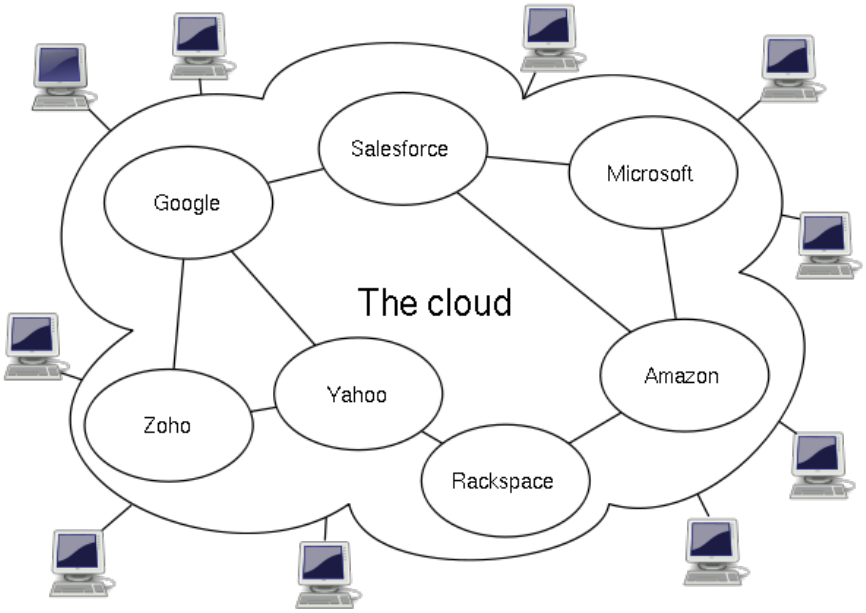
2.0.1. ¿Qué es Cloud Computing?

En este tipo de computación todo se ofrece como servicio, de modo que los usuarios puedan acceder a los servicios disponibles en la nube de Internet sin necesidad de entrar en la gestión de los recursos que usan. Es un paradigma en el que la información se almacena de manera permanente en servidores en Internet y se envía a cachés temporales de cliente.

Cloud Computing se refiere tanto a las aplicaciones entregadas como a servicios a través de Internet, como al hardware y los sistemas software en centros de datos que proporcionan estos servicios.

Alguno de los aspectos novedosos del Cloud Computing es por ejemplo el generarle al usuario la ilusión de que dispone de infinitos recursos de computación y que estos estarán disponibles según los vaya pidiendo. Otra característica importante es que los usuarios finales pueden acceder al servicio en cualquier momento y en cualquier lugar, compartir datos y colaborar con más facilidad.

Algunos ejemplos de Cloud Computing son: AmazonWeb Services, Google AppEngine y Microsoft Azure.



2.0.2. Comienzos

El inicio del Cloud Computing no ha sido sencillo, durante mucho tiempo la gente no tenia una idea clara de lo que era, por ejemplo el Vicepresidente de Hewlett-Packard Presidente de Ventas de Software Europeo dijo:

> "A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of "the cloud. "

Andy Isherwood, quoted in ZDnet News, December 11, 2008

que vendría a ser:

> "Muchas personas se están subiendo al carro de nubes [], pero no he escuchado a dos personas que digan lo mismo. Existen múltiples definiciones por ahi "del cloud "

El concepto de cloud (nube) empezó con proveedores de servicio de Internet de gran escala tales como Google y Amazon.

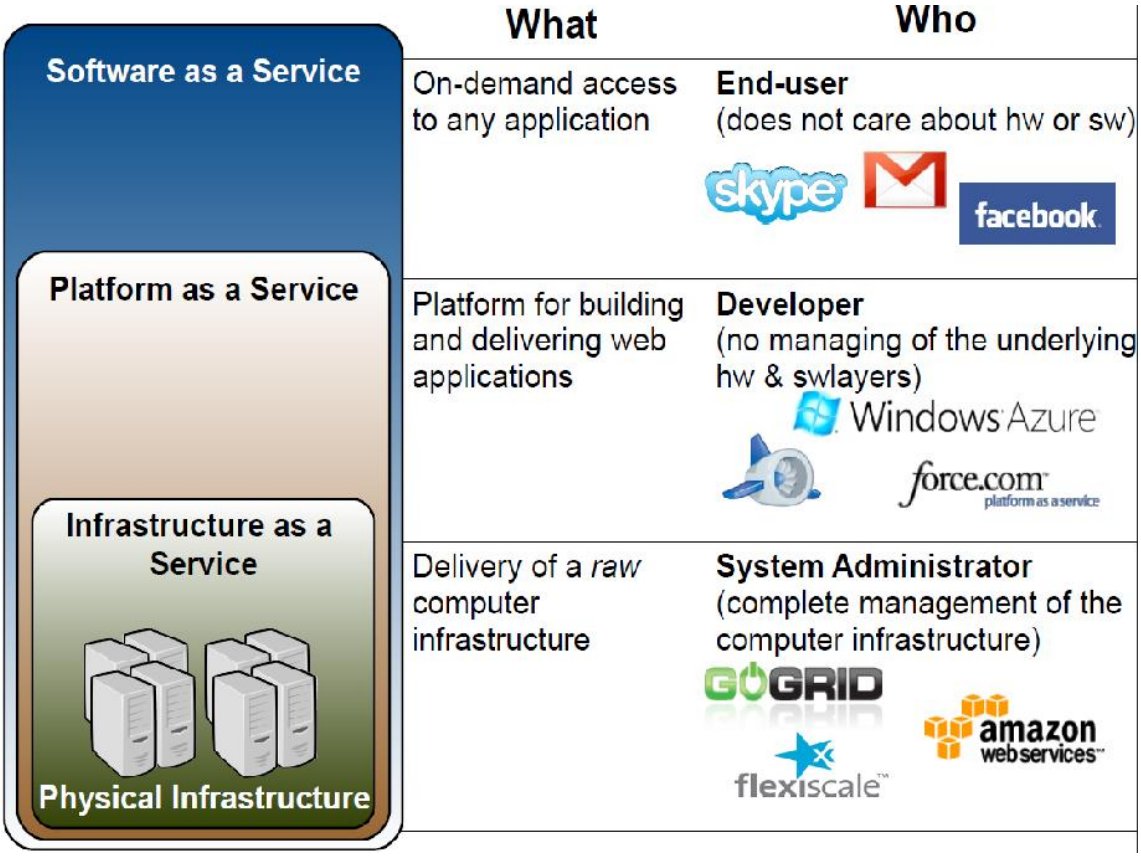
Una arquitectura emergió: un sistema de recursos horizontalmente distribuidos, introducidos como servicios virtuales masivamente escalados y manejados como recursos continuamente configurados y mancomunados.

Este modelo arquitectónico fue inmortalizado por George Gilder en su artículo de octubre 2006 en la revista Wired titulado "Las Fábricas de Información". Las granjas de servidores acerca de las cuales Gilder escribió, eran similares en su arquitectura al cómputo grid (red, parrilla), pero mientras que los grids son utilizados para aplicaciones de cómputo técnico, este nuevo modelo de nube se estaba aplicando a los servicios de Internet.

Tanto las nubes como los grids están construidos para resistir fallos de los elementos o nodos individuales . Ambos son cargados "por-uso". Pero mientras que los grids típicamente procesan los trabajos en batch (lote), con un punto definido de inicio y final, los servicios nube pueden ser continuos.

Lo que es más, las nubes expanden los tipos de recursos disponibles - almacenamiento de archivos, bases de datos, y servicios Web - y extienden la aplicabilidad a la Web y a las aplicaciones de la empresa.

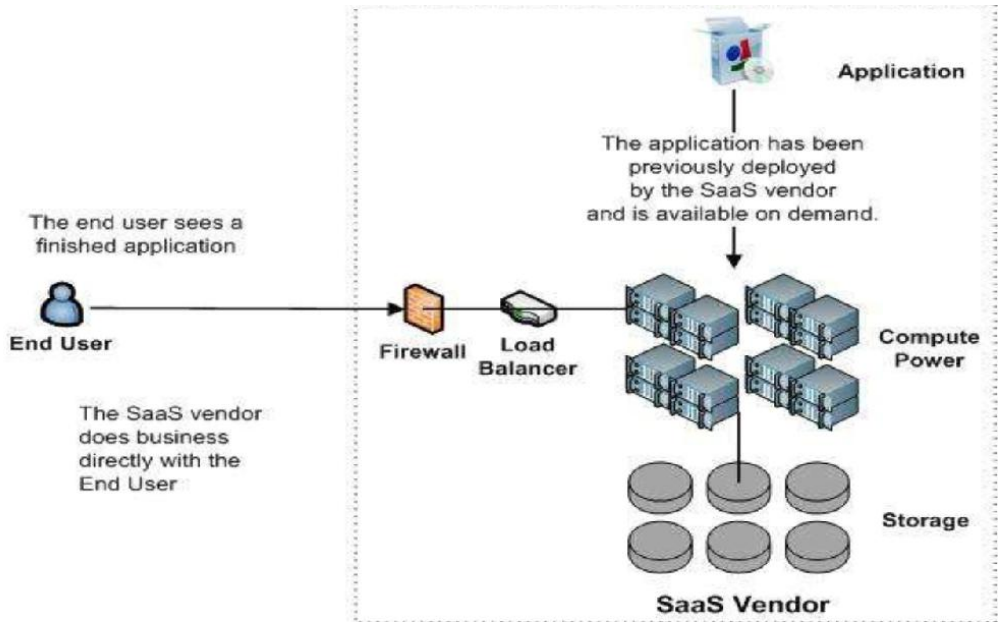
2.0.3. SaaS, IaaS y PaaS: Las tres clases de Cloud Computing



Software as a Service (SaaS): Software como Servicio. Modelo de distribución de software donde una empresa sirve el mantenimiento, soporte y operación que usará el cliente durante el tiempo que haya contratado el servicio.

El cliente utilizará el sistema alojado por esa empresa, la cual mantendrá la información del cliente en sus sistemas y proveerá los recursos necesarios para explotar esa información.

Ejemplos: Google Apps

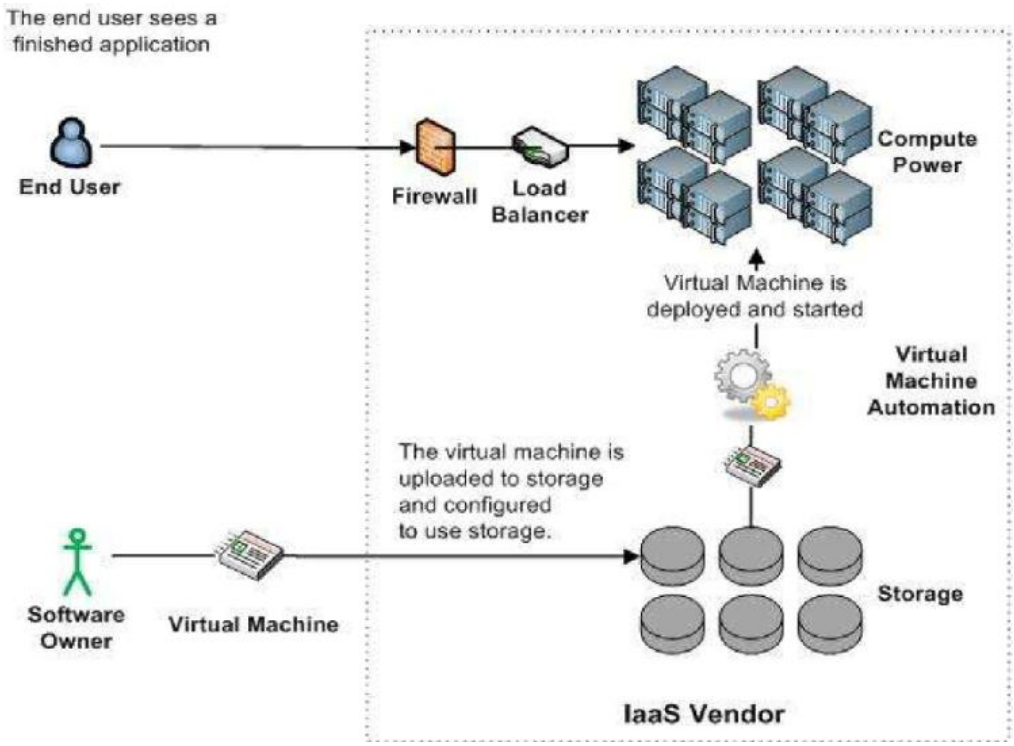


Infrastructure as a Service (Iaas): Infraestructura como Servicio. Modelo de distribución de infraestructura de computación como un servicio, normalmente mediante una plataforma de virtualización.

En vez de adquirir servidores, espacio en un centro de datos o equipamiento de redes, los clientes compran todos estos recursos a un proveedor de servicios externo.

El aprovisionamiento de estos servicios se hace de manera integral a través de la web.

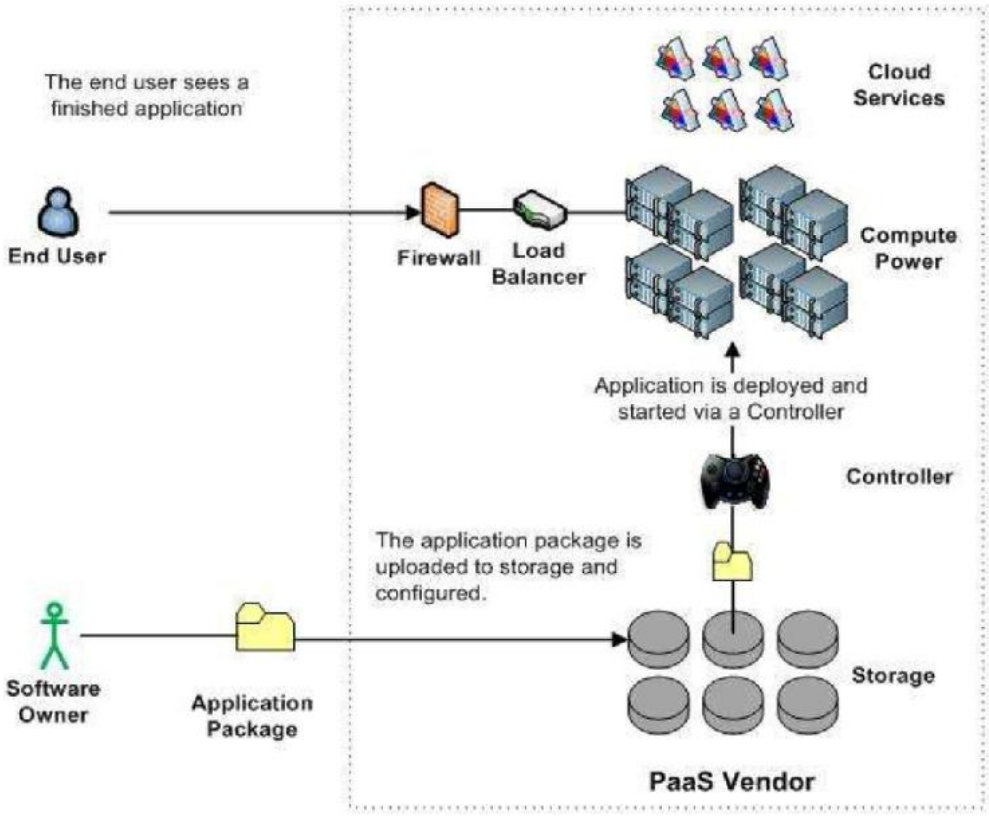
Ejemplos: Amazon Web Services EC2 y GoGrid.



Platform as a Service (PaaS): Plataforma como Servicio. Aunque suele identificarse como una evolución de SaaS, es más bien un modelo en el que se ofrece todo lo necesario para soportar el ciclo de vida completo de construcción y puesta en marcha de aplicaciones y servicios web totalmente disponibles en Internet.

Otra característica importante es que no hay descarga de software que instalar en los equipos de los desarrolladores. PaaS ofrece múltiples servicios, pero todos provisionados como una solución integral en la web.

Ejemplos: Microsoft Azure.



2.0.4. Virtualización

Historia

La Virtualización se diseñó por primera vez en los años 60 para optimizar el uso del hardware mainframe. Fue IBM la que se encargó de transformar un mainframe en distintas máquinas virtuales. Esto permitió alcanzar la multitarea, es decir ejecutar múltiples aplicaciones y procesos al mismo tiempo.

La Virtualización se abandonó durante los años 1980 y 1990 cuando las aplicaciones cliente-servidor, desktops y servidores x86 de bajo costo llevaron a la computación distribuida.

La amplia adopción de sistemas Windows y la aparición de Linux como servidores que operan en la década de los noventa, estableció como servidores estándar para la industria a los x86.

El rápido crecimiento de los servidores x86 y las aplicaciones de escritorio plantearon unos nuevos retos como : Sólo se usa entre el 10 y 15 % de la capacidad total disponible. La mayoría de la infraestructura informática debe permanecer operativa en todo momento, lo que revierte en el consumo de energía, refrigeración y los costos de las instalaciones.

El aumento de los costes de gestión, como los entornos informáticos se vuelven más complejos. El nivel de especialización y la experiencia necesarias para el personal de gestión de la infraestructura tiene que ser mayor

y los costes asociados de ese personal se ven incrementados; por lo que los nuevos x86 se estaban enfrentando a los mismos problemas que tenían los antiguos mainframes

En 1999, VMware introdujo la Virtualización para sistemas x86 para solucionar muchos de los retos arriba mencionados y transformar los sistemas x86 en sistemas de propósito general.

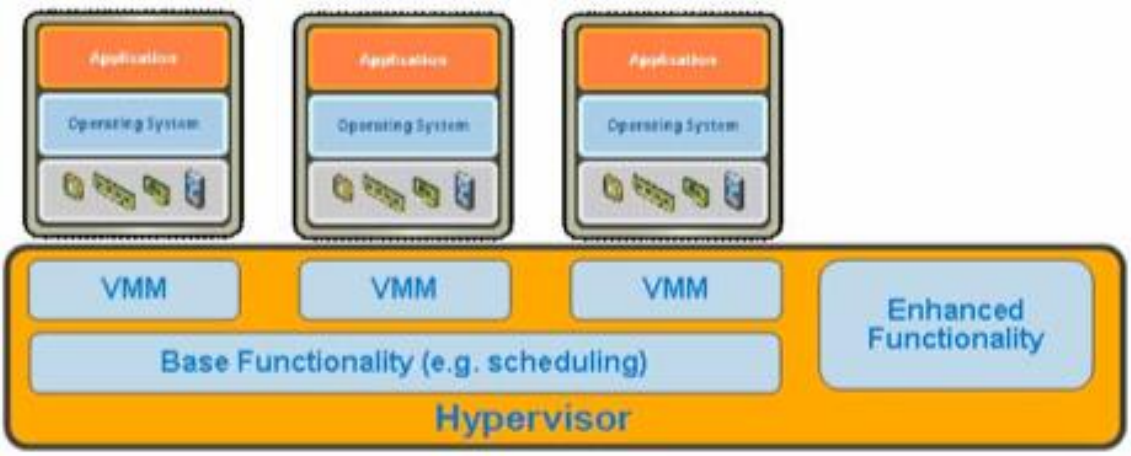
¿Qué es Virtualización?

Hoy en día se fabrican computadores muy potentes pensados para un único sistema operativo y una sola aplicación. Esto deja a la mayoría de las máquinas muy infrautilizadas , la Virtualización permite ejecutar múltiples máquinas virtuales en una única máquina física, compartiendo los recursos de esa sola computadora a través de múltiples entornos. Diferentes máquinas virtuales pueden ejecutar diferentes sistemas operativos y múltiples aplicaciones en el mismo equipo físico.

¿Cómo funciona la Virtualización?

La idea es crear una máquina virtual totalmente funcional pudiendo virtualizar CPU, memoria y disco duro. De tal manera que podrá ejecutar su sistema operativo independiente con sus aplicaciones como si fuese una máquina real.

Generalmente los sistemas de Virtualización poseen una máquina virtual que se conoce como "hipervisor" que asigna los recursos de hardware de forma dinámica y transparente. Multiples sistemas operativos ejecutan de manera simultánea en una única computadora, compartiendo el hardware y los recursos existentes.



¿Por qué usar Virtualización?

Algunas de las razones por las que elegir la Virtualización son :

1. Saca más partido a los recursos existentes: recursos de infraestructura comunes rompiendo la idea de "una aplicación, un servidor".

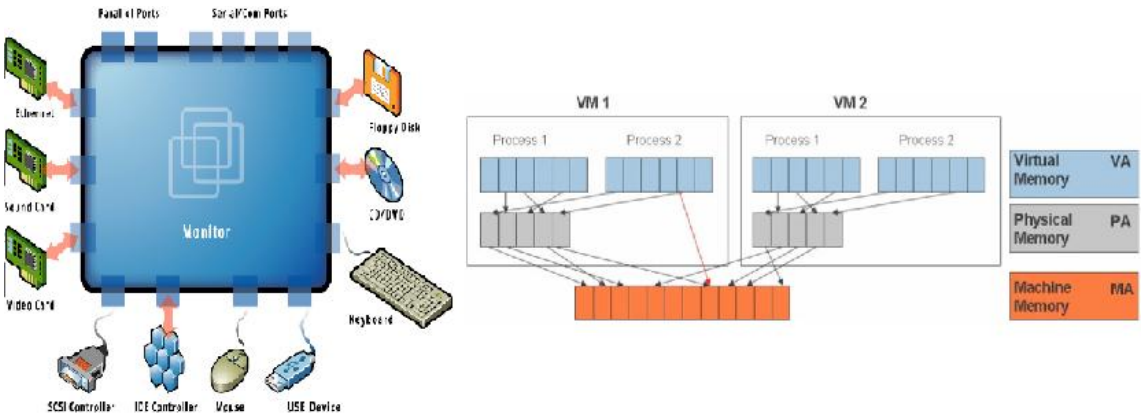
2. Reduce los costes del centro de datos mediante la reducción de su infraestructura física: menos servidores y hardware implica una reducción de bienes, un ahorro de energía y refrigeración.

3. Ganancia de una flexibilidad operativa.

La Virtualización permite reducir costos aumentar la eficiencia, la utilización y la flexibilidad.

¿Qué es una máquina virtual?

Una máquina virtual es un contenedor de software que puede ejecutar sus propios sistemas operativos y aplicaciones como si fuese una máquina real.
Una máquina virtual se comporta exactamente como un equipo físico y contiene su propia CPU, disco duro ,memoria RAM, tarjeta de interfaz de red (NIC) todas ellas virtuales es decir, basadas en software.



Un sistema operativo no puede diferenciar entre una máquina virtual y una máquina física, ni las aplicaciones u otras computadoras en una red pueden. Incluso la máquina virtual piensa que es un "verdadero" equipo. Sin embargo, una máquina virtual está compuesta enteramente de software y no contiene componentes de hardware.

A VMware virtual machine

Beneficios de las máquinas virtuales:

1. Compatibilidad: Son compatibles con todos los estándares de ordenadores x86.
2. Aislamiento: Están aisladas unas de otras, como si físicamente separados.
3. Encapsulación: Encapsulan un entorno informático completo
4. Independencia de hardware: Se ejecutan de manera independiente del hardware subyacente.

Tipos de virtualización de la CPU

1. Full Virtualization with Binary Translation
2. OS Assisted Virtualization /Paravirtualization
3. Hardware Assisted Virtualization

Full Virtualization with Binary Translation :

Este tipo de Virtualización crea un sistema virtual completo en el que el sistema operativo y las aplicaciones pueden ejecutarse.

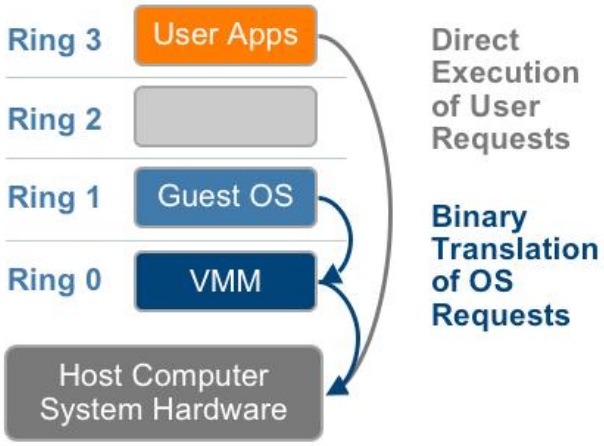
El resultado es un sistema virtualizado que consta de las mismas capacidades de ejecución que un sistema funcionando directamente sobre hardware. De este modo se capacita al software de independencia con respecto al hardware.

Full Virtualization es la única opción que no requiere asistencia de hardware o sistema operativo para virtualizar.

El hipervisor traduce todas las instrucciones del sistema operativo en el momento e introduce los resultados en las caches para un uso futuro mientras que las instrucciones a nivel de usuario se ejecutan sin modificar en los tiempos originales.

La Virtualización completa ofrece el mejor aislamiento y seguridad para máquinas virtuales.

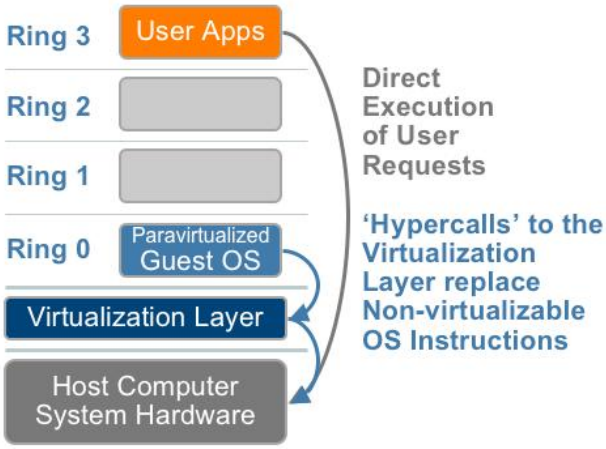
Productos de Virtualización de VMware y Microsoft Virtual Server son ejemplos de virtualización completa



OS Assisted Virtualization /Paravirtualization :

Paravirtualización se refiere a la comunicación entre el sistema operativo huésped y el hipervisor para mejorar el rendimiento y la eficiencia.

Paravirtualización implica modificar el sistema operativo del kernel para reemplazar las instrucciones no virtualizables con hypercalls que se comunican directametne con la capa de virtualizacion hipervisor.

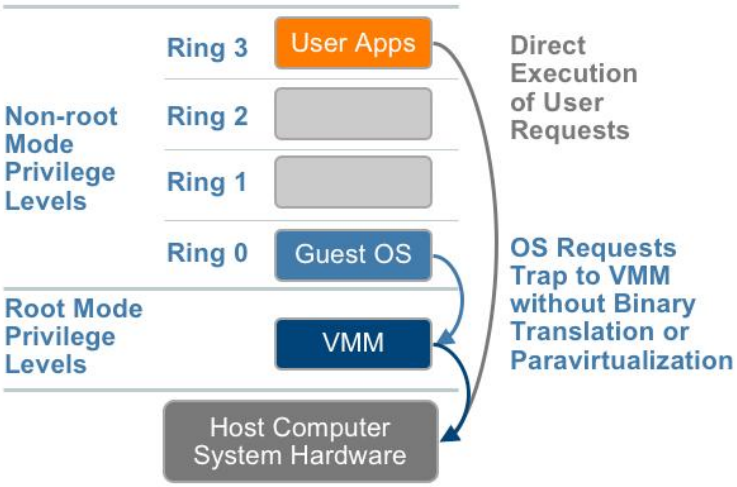


El proyecto de código abierto Xen es un ejemplo de paravirtualización que virtualiza el procesador y la

memoria utilizando una versión modificada del kernel de Linux y virtualiza la E / S con controladores de dispositivos de evaluación personalizada del sistema operativo.

Hardware Assisted Virtualization:

La principal diferencia con Full o Paravirtualización es que las llamadas privilegiadas se establecen de forma automática para el hipervisor, eliminando la necesidad de cualquier traducción binaria o paravirtualización. El estado de resultados se almacena en la Virtual Machine Control Structures (VT-x) o Virtual Machine Control Blocks (AMD-V).



2.0.5. OpenNebula

¿Qué es OpenNebula?

OpenNebula es una herramienta open-source que permite la creación de cualquier tipo de cloud: privado, publico e hibrido, de una forma sencilla. OpenNebula ha sido diseñado para encajar en cualquier centro de datos existente.

Mediante OpenNebula se puede transformar cualquier "data center" en una infraestructura virtual ágil y flexible, la cual se adapta dinámicamente a los cambios en función de la carga de trabajo.

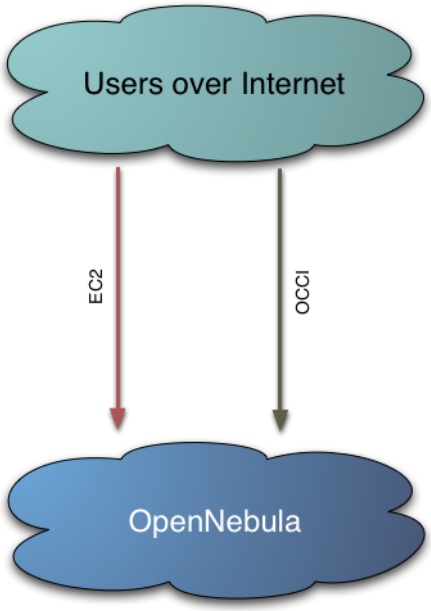
¿Cómo funciona?

OpenNebula junta las tecnologías de almacenamiento, red y Virtualización para permitir el despliegue dinámico de servcios sobre arquitecturas distribuidas, combinando tanto los recursos del "data center", como los recursos remotos del cloud.

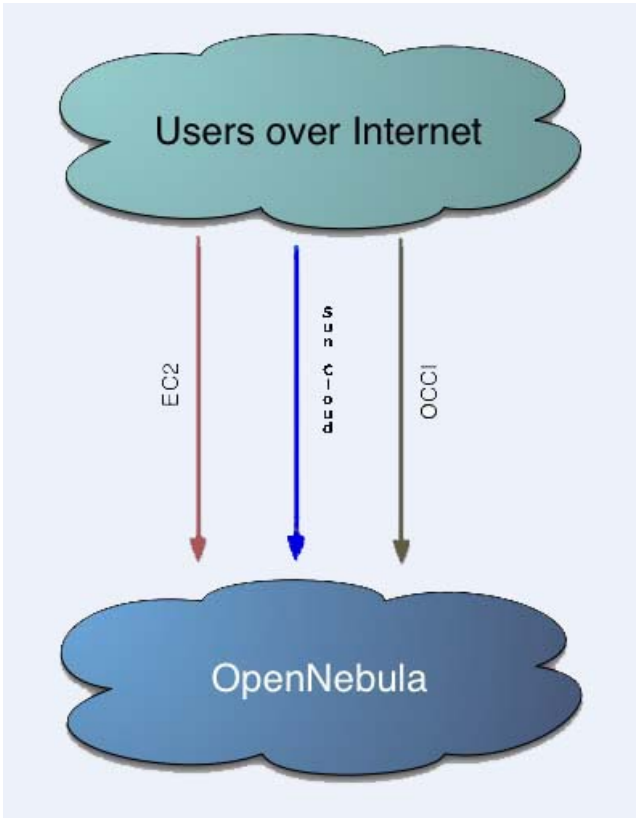
Más adelate se explica su estructura y componenetes con más precisión.

OpenNebula y SunCloudAPI

OpenNebula v1.4 Cloud ofrece un entorno de computación virtual accesible desde dos interfaces remotas diferentes OCCI y EC2. Estas permiten lanzar máquinas virtuales basíndose en una variedad de imágenes disponibles con diferentes sistemas operativos y configuraciones.



SunCloudAPI incorpora una nueva interfaz a las ya existentes sobre OpenNebula, dando al igual que las restantes la posibilidad de crear y gestionar máquinas virtuales utilizando los servicios ofrecidos por OpenNebula.


Users over Internet
EC2
Sun Cloud
OCCI
OpenNebula

Capítulo 3

Interfaces Cloud

3.0.6. Servicios Web

El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Los Servicios Web son ejecutados en el sistema que los aloja y pueden ser accedidos mediante un API dentro de una red, ya sea privada o pública.

A la hora de hacer uso de los Servicios Web existen diferentes estilos de arquitectura Software a los que se pueden adaptar. A continuación se detallan tres de los principales.

1. Remote Procedure Calls (RPC, LLamadas a Procedimientos Remotos): Los Servicios Web basados en RPC presentan una interfaz de llamada a procedimientos y funciones distribuidas, tomando como unidad básica la operación WSDL, XML como lenguaje para definir el IDL y el HTTP como protocolo de red.

   Se trata de uno de los primeros estilos empleados en la elaboración de Servicios Web, llegando a ser denominados como la primera generación. Aunque RPC es uno de los estilos más extendidos, posee algunas desventajas que provocan que muchos desarrolladores piensen que acabará desapareciendo tarde o temprano. Entre estas podemos encontrar: el balanceo de carga y la tolerancia a fallos no están disponibles, por ello la gestión de picos de carga y la priorización se convierte en una tarea difícil.

2. Service-Eriented Architecture (SOA, Arquitectura Orientada a Servicios): en un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. Este estilo de arquitectura toma como unidad básica el mensaje y no la operación como en el caso anterior.

   Los Servicios Web basados en SOA son soportados por la mayor parte de desarrolladores de software y analistas. Una de las peculiaridades de este estilo es que se centra en el "contrato" proporcionado por el documento WSDL, más que en los detalles de implementación subyacentes. La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo.

3. REST (REpresentation State Transfer): Los Servicios Web basados en REST intentan emular al protocolo HTTP o protocolos similares mediante la restricción de establecer la interfaz a un conjunto conocido de operaciones estándar (DELETE, GET, PUT, POST). Por tanto, este estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones.

   En los últimos años se ha popularizado el empleo de REST en los Servicios Web. A continuación se profundizará de una manera más extensa en este estilo de arquitectura de Software, llegando a conocer las causas que le han llevado a esta posición.

3.0.7. REST

REST acrónimo de REpresentational State Transfer, se trata de una técnica de arquitectura de software para sistemas hipermedia distribuidos. Como bien define la ingeniería del software:

> "a software architectural style is a set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem"

Se trata de una serie de reglas a las que se atiene el programador para adaptar su aplicación al estilo REST. No existe ningún tipo de estandar que especifique REST, ni paquete de desarrollo. Aunque REST no se trate de una estándar, internamente hace uso de una serie de estos como bien puede ser HTTP, URL, para la representación de recursos puede emplear XML, HTML, GIF o como tipos MIME text/xml, text/html.
El origen de este término procede de la tesis doctoral elaborada por Roy Fielding en el año 2000.

Fielding es uno de los principales autores de la especificación del protocolo HTTP, y ha conseguido que el estilo REST sea aceptado por la mayor parte de la comunidad en la actualidad. Fue dasarrollado en paralelo al HTTP1.1 teniendo como base la especificación de HTTP1.0.

Uno de los principales objetivos de REST es intentar capturar las características que han hecho que la Web sea tan exitosa. Dentro de estos objetivos, podemos encontrar la escalabilidad de la interacción con los componentes, generalidad de interfaces, puesta en funcionamiento independiente y la compatibilidad con componentes intermedios.

Para conseguir estos objetivos REST plantea las siguientes restricciones que deben tenerse en cuenta:

1. **Un protocolo cliente/servidor:** se separan la interfaz de usuario del almacenamiento de datos. De esta forma se puede mejorar la portabilidad de la interfaz entre distintas plataformas y en la parte del servidor se puede mejorar la escalabilidad mediante la simplificación de componentes del mismo.

2. **Sin estado:** cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

   En ocasiones esta restricción provoca un aumento del tráfico de información al tener que mantener en cada petición el mismo contenido. Por ello se añaden etiquetas que permiten determinar si la respuesta

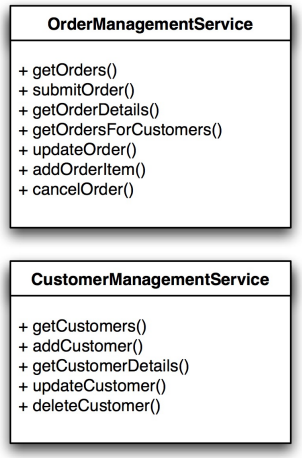
obtenida es cacheable o no-cacheable para así poder usar esa información en peticiones equivalentes.

3. **Interfaz uniforme:** existe un conjunto de operaciones bien definidas que se aplican a todos los recursos de información. Estas operaciones definidas en HTTP se identifican mediante verbos que representan la acción que se lleva a cabo.

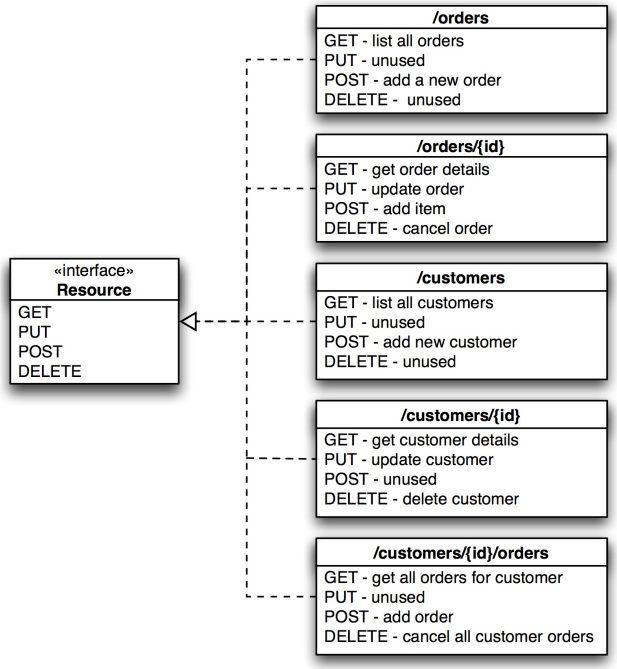
Cuadro 3.1: Relación entre SQL y verbos HTTP

Action	**SQL**	**HTTP**
Create	Insert	Put
Read	Select	Get
Update	Update	Post
Delete	Delete	Delete

Considerando esta situación:



El equivalente en REST para la situación anterior sería:



a)Sintaxis universal para identificar los recursos: Los recursos son los objetos lógicos a los que se le envían mensajes. No pueden ser directamente accedidos o modificados, para referirnos a ellos empleamos representaciones. Esto se consigue mediante el uso de URIs. A cada recurso le corresponde un identificador único representado en forma de URI.

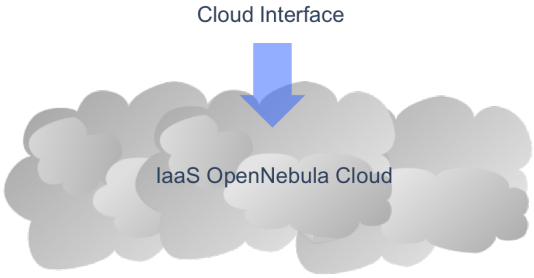
```
http://example.com/customers/1234
http://example.com/orders/2007/10/776654
http://example.com/products/4554
http://example.com/processes/salary-increase-234
```

b)Uso de hipermedios: Los recursos están interconectados mediante URIs permitiendo al cliente progresar de un estado a otro. Para representar esta situación se emplea HTML o XML permitiendo ir de un recurso REST a muchos otros sin necesidad de usar registros u otra infraestructura adicional.

```
<order self='http://example.com/customers/1234' >
<amount>23</amount>
<product ref='http://example.com/products/4554' />
<customer ref='http://example.com/customers/1234' />
</order>
```

c)Componentes por capas: Permite la inclusión de intermediarios como pueden ser servidores proxies, servidores cache, gateways entre los clientes y los recursos. Obteniendo con ello un mejor rendimiento y ciertas condiciones de seguridad.

3.0.8. Interfaces Comerciales



En el ámbito del Cloud Computing se ha extendido el uso del estilo REST para la unión cliente-servidor proporcionando este último un API pública para permitir el acceso a la infraestructura que ofrece. Alguna de las APIs más conocidas que permiten la administración de recursos cloud de forma simple y remota a un alto nivel de abstracción pueden encontrarse:

1. Amazon EC2
2. OCCI
3. vCloud

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) se trata de un servicio web que permite al usuario lanzar y controlar tanto instancias con servidores Linux/UNIX como la versión Windows de los mismos, en sus "data centers". Para ello Amazon EC2 proporciona al propietario de estos recursos un control total sobre ellos. Una de las ventajas del servicio es que solamente se paga por la capacidad de computo que se está usando en cada momento.

A la hora de interactuar con este servicio Amazon EC2 proporciona dos APIs: SOAP y REST. En ambas interfaces el "XML body" devuelto es el mismo. Para obtener información más detallada dirigirse a: [1]

OCCI

El grupo de trabajo Open Grid Forum Open Cloud Computing Interface (OCCI) en el momento del desarrollo de esta memoria está llevando a cabo la especificación de un API para la gestión remota de la infraestructura de Cloud Computing, permitiendo el desarrollo de herramientas para las tareas más habituales, la creación, escala automática y seguimiento de recursos cloud.

OpenNebula posee la primera implementación del borrador de esta API, permitiendo interactuar mediante esta API con diversas herramientas cliente. El borrador de esta especificación se puede encontrar en: [2]

VCLOUD

Es la misma filosofía que en los dos casos anteriores, pero siendo esta vez la conocida empresa VMware la que proporciona la especificación de este APIb [3]

Son muchos los proveedores de servicios que se están acogiendo a esta API, entre ellos se pueden encontrar BlueLock, Hosting, Melbourne IT y Terremark, englobandose dentro del concepto VMware vCloud Express.[4]

Como se aprecia el número de APIs que se pueden encontrar para servicios cloud es muy amplio y va en aumento en los últimos años. Debido a ello algunas compañías están desarrollando aplicaciones en el intento de unificar toda esta disparidad de interfaces. Entre las más destacadas de estas aplicaciones se pueden encontrar:

1. Libcloud
2. Deltacloud

LIBCLOUD

Libcloud es una librería cliente escrita en Python para interactuar con muchos de los proveedores más importantes de la ”nube”, entre los que se encuentran Amazon EC2, OpenNebula, GoGrid, Rackspace y Terremark(vCloud Express) .

Se creó para facilitar a los desarrolladores la tarea de implementar productos que funcionen entre cualquiera de los proveedores de servicios que soporta.

DELTACLOUD

Deltacloud se construye como un servicio basado en REST. No es una conexión directa entre una librería y el programa que lo utiliza, sino que se trata de un cliente que a través del API Deltacloud habla con el servidor que implementa la interfaz REST sobre HTTP.

Para ello, proporciona una serie de "drivers" que permiten la comunicación del servidor con cada una de las APIs de los proveedores que soporta entre los que se pueden encontrar Amazon EC2, OpenNebula, Rackspace, RHEV-M, Rimuhosting.

Capítulo 4

Arquitectura y Diseño del Sistema

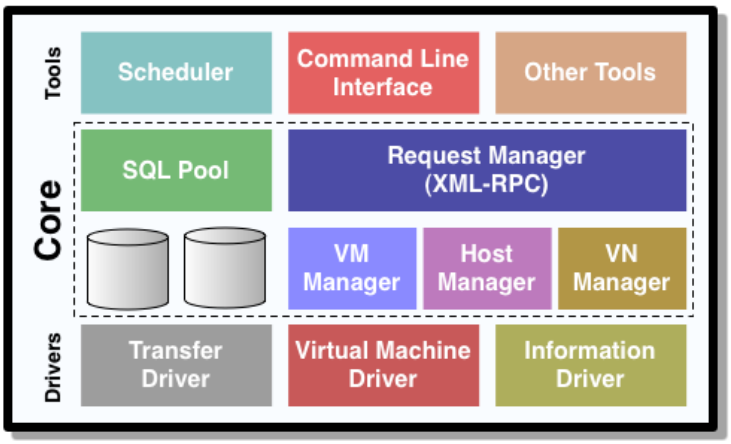
La arquitectura interna de OpenNebula puede dividirse en tres capas

1. Herramientas, desarrolladas usando la interfaz proporcionada por el núcleo de OpenNebula.

   Esta capa contiene herramientas distribuidas con OpenNebula, como puede ser la "CLI", el "scheduler", la implementación del API de libvirt o las interfaces RESTful Cloud y también herramientas de terceros que pueden ser creadas fácilmente usando la interfaz XML-RPC o la nueva API de OpenNebula (OCA).

2. Núcleo, consiste en un conjuntos de componentes para el manejo de máquinas virtuales, almacenamientos, redes virtuales y hosts.

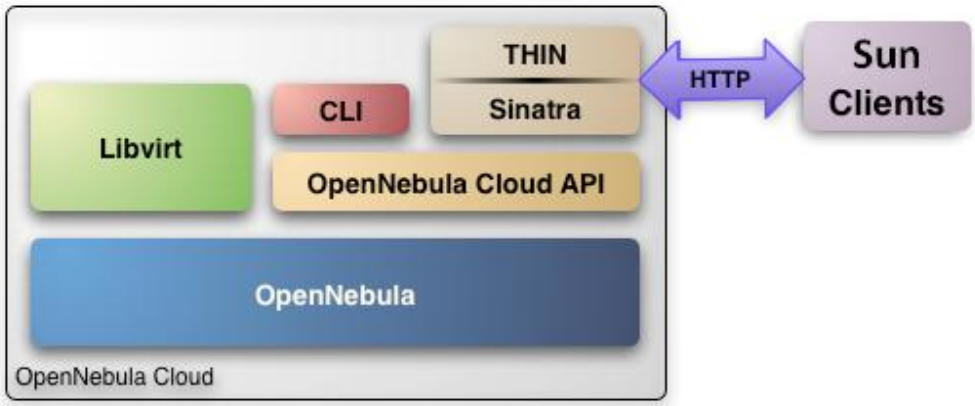
3. Drivers, para incorporar diferentes tecnologías de Virtualización, almacenamiento y monitorización.



El servicio de Sun Cloud implementado en este proyecto está situado por encima de la arquitectura explicada anteriormente, usando la nueva capa OpenNebula Cloud API (OCA) que ofrece todas las funcionalidades de un Cloud privado.

Para conseguir convertir un cloud privado en un cloud público accesible desde el exterior es necesario seguir una filosofía de cliente servidor. A la hora de crear la parte servidora se usó Thin como servidor y Sinatra, un framework web, que facilita la interacción con las llamadas de tipo REST.

Por otro lado, se implementó un cliente por linea de comandos para permitir interactuar con el servidor de una forma sencilla.



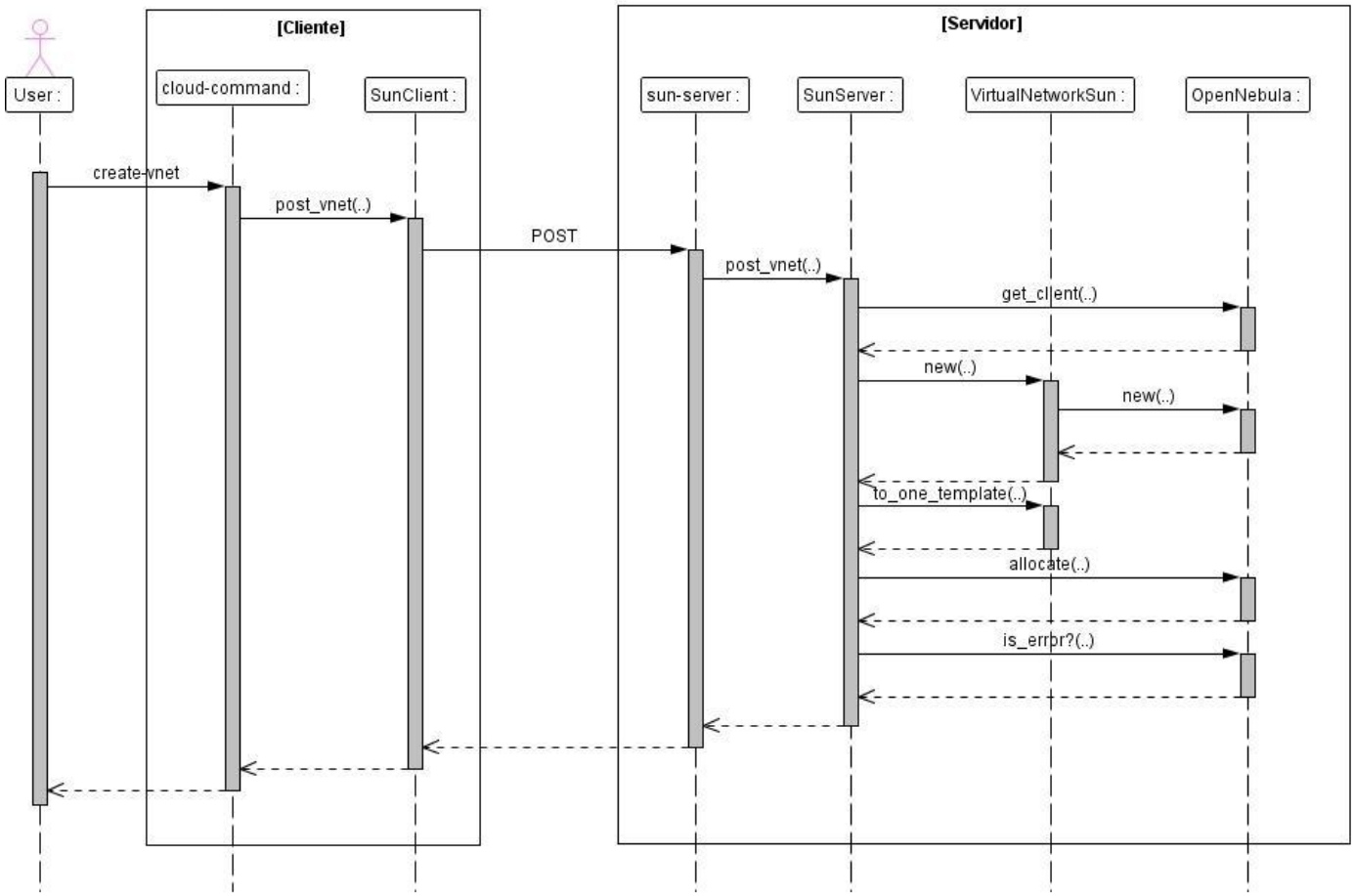
Para entender cómo funciona el sistema, usaremos uno de los lenguajes de modelado más extendido y usado en el mundo llamado UML. Más en concreto, utilizaremos diagramas de secuencias para explicar las funciones más significativas de nuestro sistema y así poder ver como se realizan e interactúan con las distintas partes de nuestra arquitectura.

Primero se explicaran las funciones relativas a la manipulación de Redes Virtuales, en referencia a como se crean, consultan y destruyen, pasando más tarde a hablar del uso de Máquinas Virtuales.

4.0.9. Redes Virtuales

Creación

Para la creación de Redes Virtuales se hace uso de la función create-vnet que llevará como parámetro el nombre de la nueva red. El funcionamiento de dicha acción queda reflejado en el siguiente diagrama:



En este diagrama se refleja los distintos módulos de nuestro sistema y cómo se van realizando distintas acciones para lograr el fin esperado. Las distintas funciones usadas son:

1. Create-vnet : es la acción principal que activa toda la secuencia y cuya implementación se encuentra en el archivo "cloud-command.rb". Dicha función debe ser llamada a través de un terminal con la siguiente sintaxis

```
$ cloud command –create-vnet –name <name>
```

donde <name>es el nombre que le quieres asociar a la nueva red.

La dirección ip de la red y el bridge al cual está conectada deben ser especificados en el archivo de configuración, $ONE_LOCATION/etc/sun-server.conf.

2. Post_vnet(name): Este método recibe como parámetro el nombre de la red que se desea crear. Su función principal es construir el paquete que se enviará por Internet al servidor y rellenarlo adecuadamente tal y como especifica Sun.

3. POST: En este caso nos referimos a cómo viaja el paquete por Internet, identificando que es una petición de tipo POST tal y como se especifica en los principios de REST. El paquete llegará más tarde a nuestro servidor y será desmenuzado para seguir así con el proceso.

4. Post_vnet(request): La función coge como parámetro el paquete y lo usará para sacar de él la información que necesita para la creación de la red.

5. Get_client(requestEnvironment): En este paso se usará parte de la información que transporta el datagrama para a través de ella obtener un cliente con sus credenciales. Tanto este paso como los siguientes los realiza el módulo de OpenNebula.

6. New(xml,client): Un método que crea una nueva red virtual mediante un fragmento de xml por defecto y el cliente obtenido anteriormente.

7. To_one_template(name, bridge, network_address): Utiliza los parámetros que recibe, para a través de ellos rellenar el template tipo OpenNebula que identificará la nueva red.

8. Allocate(vntemplate): Coge como atributo vntemplate, que es el template que representa la nueva red y se lo manda a OpenNebula para que lo aloje en la red creada anteriormente por defecto, obteniendo así nuestra propia configuración.

9. Is_error?(rc): Lo que recibe esta función es el valor que devuelve allocate y sirve para identificar si la operación ha tenido éxito o no. En caso de tener éxito se le devuelve al cliente un json indicando la creación de la nueva red, en el otro caso se devuelve el error surgido.

Consulta de una Red

La consulta de una red virtual anteriormente creada se realiza mediante la orden

donde vnet_id debe ser el identificador de la red que se desea consultar. Dicha red debió ser creada con anterioridad, en caso contrario provocará un error que se mostrará por la linea de comandos.

La estructura de funcionamiento es muy parecida a la anterior con estas diferencias principales:

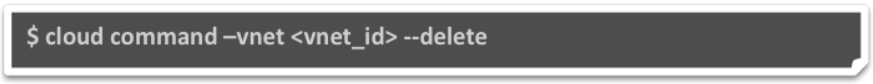
1. En este caso el paquete que se manda vía Internet lleva una petición de tipo GET dado que la acción que se va a realizar es una consulta.

2. Respecto a la interacción que se realiza con OpenNebula, ahora se dispone del identificador de la red que va a ser consultada, por lo que se le pedirá a OpenNebula que la devuelva en un atributo network, para a través de él, acceder a sus campos y así devolverle al cliente un json con la información de dicha red o con un error en caso de que la operación falle.

Borrado de una red

Al igual que ocurre con la consulta de una red, esta operación es muy parecida a la primera y es por ello por lo que su diagrama de secuencia no se muestra, únicamentese enumeran las diferencias con las anteriores.

Para el borrado de una Red Virtual se usará el siguiente comando



Donde vnet_id es el identificador de la red que se desea borrar.

Las diferencias principales con la consulta son:

1. Para seguir con la sintaxis de REST, en este caso el paquete mandado por Internet contendrá una petición de tipo DELETE.

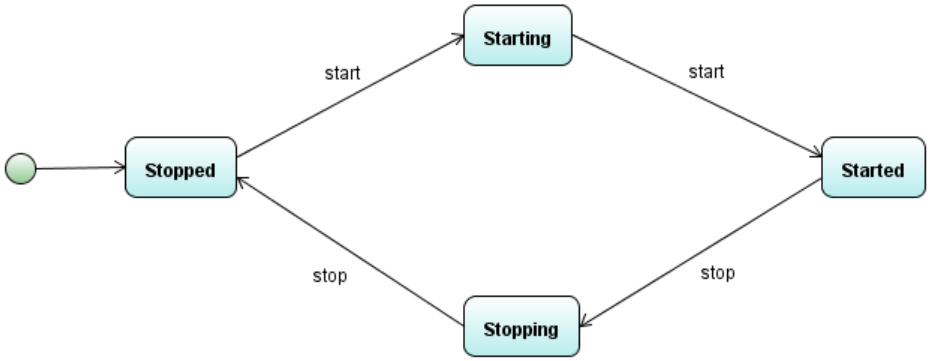
2. Se sigue el mismo procedimiento que una consulta, sólo que una vez obtenida la network buscada se hace uso de una función delete que se encarga de borrar dicha red del sistema. Entonces devolverá un json que indicará el éxito o error de dicha operación.

4.0.10. Máquinas Virtuales

Para detallar el funcionamiento referente a las máquinas virtuales, se presentan dos tipos de diagramas. El primero de ellos se referirá a cómo se modifica el estado de la máquina según se van ejecutando las funciones de control; mientras que el segundo reflejará como interactúan los métodos con los distintos componentes de la arquitectura de nuestro sistema.

Estados de las Máquinas Virtuales

Toda máquina virtual desde su creación pasa por distintos estados, los cuales quedan reflejados en el siguiente diagrama.



Dichos estados son:

1. Stopped: Dode se encuentra la máquina virtual nada más crearse y que también se puede alcanzar realizando un stop. Simboliza una situación en la cual la máquina se encuentra en pausa.
2. Starting: Es un estado intermedio entre Stopped y Started, al cual se llega una vez que el usuario ha enviado una petición start al sistema. Dicho estado refleja la situación en la que se encuentra la máquina momentos antes de que arranque.
3. Started: Situación final que se alcanza después de haber realizado un start. En caso la máquina se encuentra activa.
4. Stopping: Es una posición entre Started y Stopped, a la que se llegará una vez que el usuario ha enviado una petición stop al sistema. Este estado refleja la situación en la que se encuentra la máquina momentos antes de su parada.

La sintaxis para el cambio de estado de una máquina virtual es la siguiente:

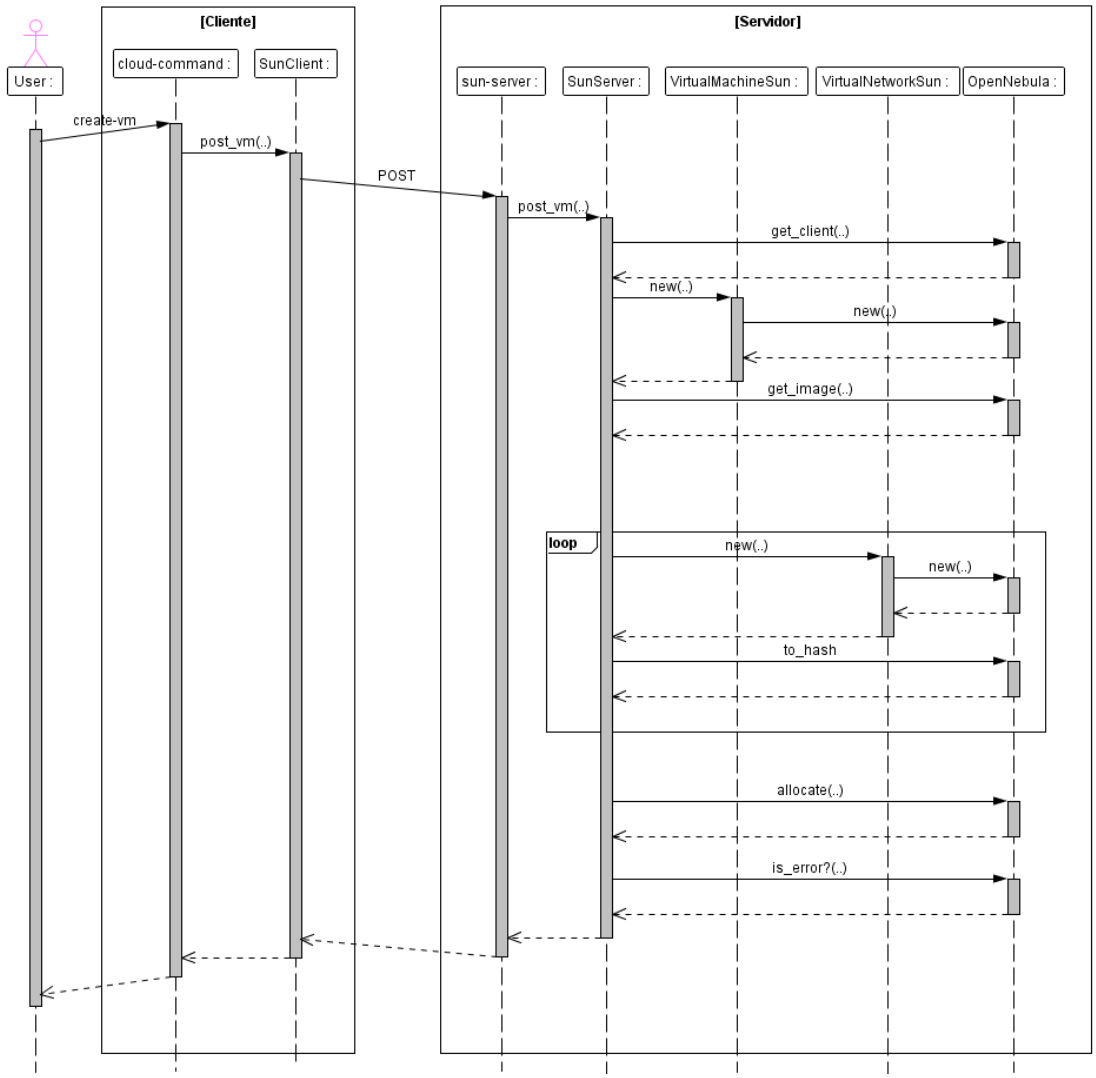
```
$ cloud command –vm <vm_id> [ACTIONS]

                    [ACTIONS]=
                      --start
                      --stop
```

Donde " vm_id " se refiere al identificador de la máquina cuyo estado se quiere modificar.

Creación

La creación de un máquina virtual se lleva a cabo mediante la función create-vm, que actuará de una forma u otra según los parametro recibidoscual según los parámetros recibidos. Dicha actuación será explicada posteriormente cuando se entre más en detalle en cada método.

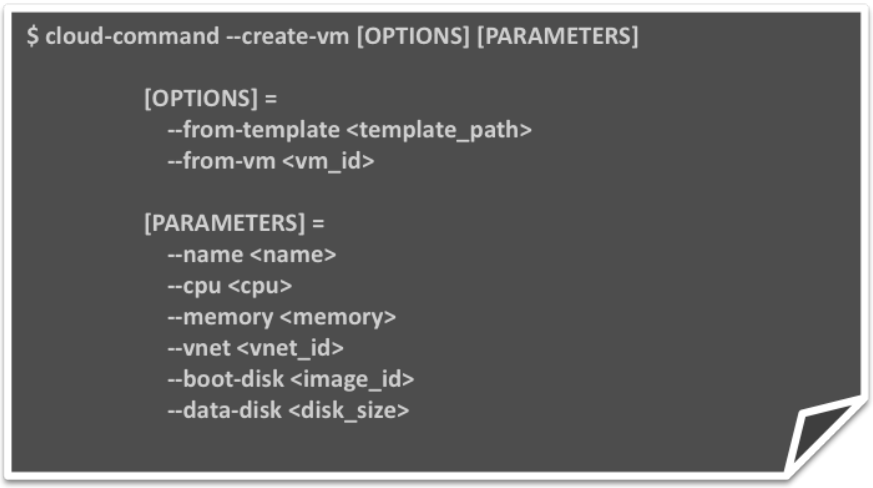
El efecto que produce dicha llamada en la estructura de nuestro sistema se puede ver reflejado claramente en el siguiente diagrama.



Observándole, podemos ver una estructura clara del modelo cliente-servidor, quedando reflejado qué archivos forman parte del cliente y cuales del servidor y cómo se enlazan en su interior.

Para entender cómo funciona el sistema, ahora se procederá a una explicación detallada de los métodos anteriormente nombrados:

1. Create-vm: es la acción principal que ejecuta el usuario por línea de comandos, y conlleva la siguiente sintaxis:



A la hora de crear una nueva máquina virtual se pueden seleccionar dos opciones como base:

El primer caso sería ”from-template”, que a partir de un template JSON elaborará el template para OpenNebula.

El segundo caso el template se basaría en una máquina ya creada anteriormente.

Para ambos casos, al igual que si no seleccionamos ninguna opción, los parámetros que se introduzcan por la linea de comandos serán tomados como preferentes frente a los parámetros de base.

Los parámetros que se pueden introducir por la linea de comandos son los siguientes:

a) name, nombre de la nueva máquina virtual.

b) cpu, número de cpus destinados a la VM.

c) memory, memoria de la VM

d) vnet, identificador de la red a la cual conectar dicha VM

e) boot-disk, imagen de disco

f) data-disk, capacidad para un disco de datos.

2. (Cliente) Post_vm(opts): recibe como parámetro un array opts, que contiene la configuración que se le quiere dar a la máquina. Dicha información la almacena en formato json en el cuerpo de un paquete y se la manda al servidor con el resto de campos del datagrama rellenos tal y como indica Sun.

3. POST: hace referencia al movimiento de dicho paquete por la red desde que sale del cliente hasta que llega al servidor. Es una petición de este tipo porque se va a crear un recurso (máquina virtual) en la red y es en este caso cuando se debe usar según indica REST.

4. (Servidor) Post_vm(request,params): esta función recibe el propio paquete mandado por la red y además unos parámetros que saca el servidor de la URL que le llega. Internamente con ambos datos crea la máquina virtual al gusto del usuario.

5. Get_client(requestEnvironment): como ya explicamos en el apartado de redes virtuales, este método es usado para conseguir un cliente con credenciales.

6. (MáquinaVirtual) New(xml,client): sirve para crear una máquina virtual por defecto la cual cogerá la configuración deseada después de realizar la orden allocate.

7. Get_image(image_id): sirve para obtener información de la imagen que queremos asociar, la cual se usará posteriormente para la configuración de la máquina.

8. (RedVirtual) New(xml,client) y to_hash: ambos métodos están representados en el diagrama dentro de un bucle. Esto es debido a que se van a tener que ejecutar tantas veces como redes virtuales queramos asociar a la máquina. El proceso que sigue en cada vuelta es el siguiente:
   Se usa el new para sacar la red que queremos asociar y una vez obtenida, usamos la función to_hash para crear un array que contenga toda la información de dicha red, la cual usaremos para la configuración de la máquina.

9. Allocate(vmtemplate) : esta función coge toda la información contenida en el archivo xml vmtemplate y la usa para alojar en la máquina virtual creada anteriormente por defecto, la máquina que realmente deseaba el usuario.

10. Is_error?(rc) : comprobación que realiza OpenNebula para saber si el alojamiento de la máquina se ha producido satisfactoriamente o con errores.

Consulta de una máquina virtual

Esta funcionalidad sigue un esquema parecido al de creación de la máquina, sólo que la interacción final es bastante más simple como explicaremos más adelante.

El comando que realiza esta función es:

```
$ cloud command –vm <vm_id> --get
```

donde vm_id es el identificador de la máquina que queremos consultar.

Las principales diferencias con la creación son:

1. El paquete que se enviará por Internet contendrá una petición GET dado que lo que nosotros queremos hacer es consultar un recurso de Internet, y no como antes que se hacía un POST porque queríamos crearlo.

2. Una vez el servidor reciba el paquete, consultará la máquina virtual que buscamos y sacará su información para después almacenarla en un json que enviará al cliente, el cual se encargará de mostrársela al usuario.

Borrado de una máquina virtual

El comando que realiza esta acción es:

```
$ cloud command –vm <vm_id> --delete
```

siendo vm_id el identificador de la máquina que queremos borrar.

Las diferencias con el anterior son:

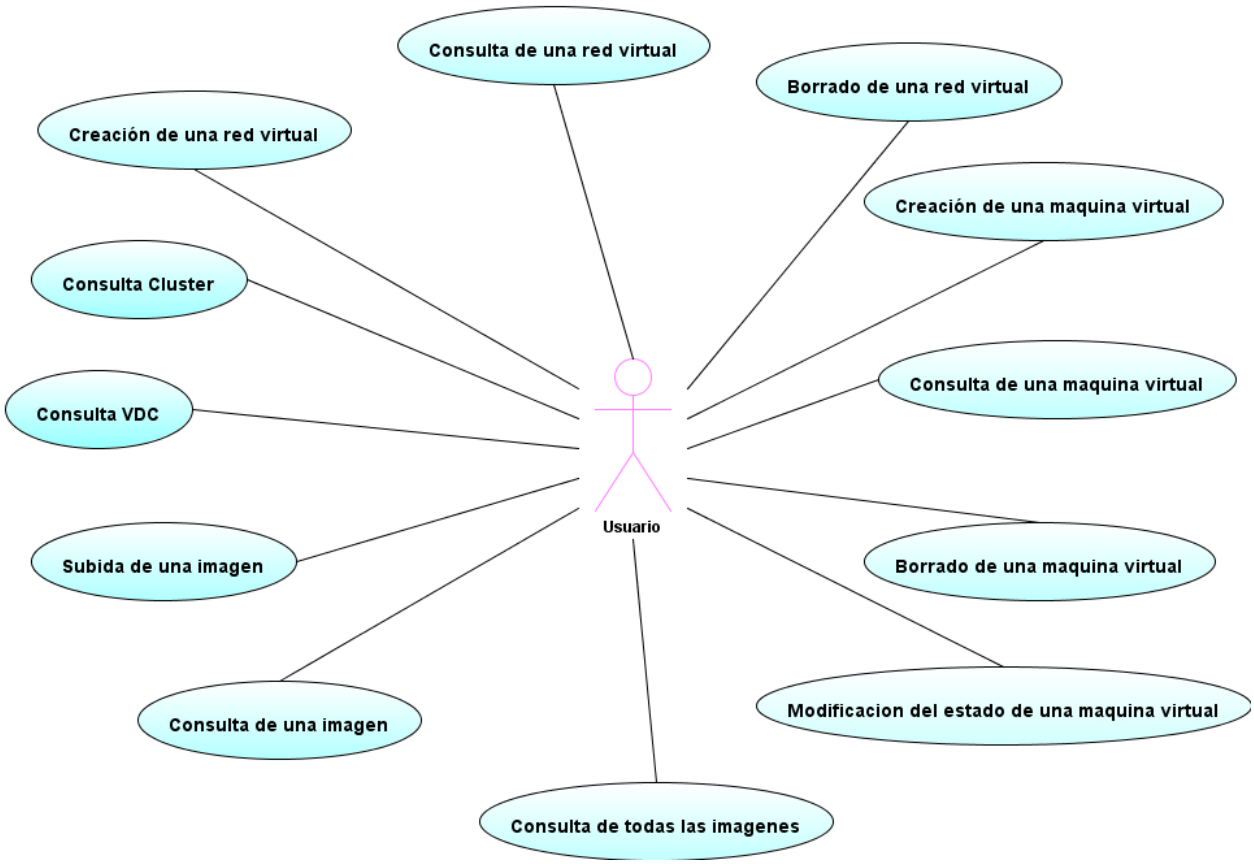
1. El paquete que se enviará esta vez contendrá una petición DELETE tal y como indica REST que se debe hacer en estos casos.
2. En el servidor, se sacará la máquina virtual que se quiera borrar y se hará uso de la función delete que cumple dicho objetivo.

Capítulo 5

Casos de Uso del Sistema

En este apartado se estudia cuáles son las distintas funciones que soporta nuestro sistema y que nos van a permitir la creación y manejo de las llamadas "nubes".

Pueden verse reflejadas en el siguiente diagrama



Y se pueden clasificar en cuatro grandes grupos:

1. **Funciones que hacen referencia al manejo de redes virtuales:**

   a) Creación de una red virtual.

```
$ cloud command –create-vnet –name <name>
```

   b) Consulta de una red virtual.

```
$ cloud command –vnet <vnet_id> --get
```

   c) Borrado de una red virtual.

```
$ cloud command –vnet <vnet_id> --get
```

2. **Funciones para el control de máquinas virtuales:**

   a) Creación de una máquina virtual.

```
$ cloud-command --create-vm [OPTIONS] [PARAMETERS]

        [OPTIONS] =
          --from-template <template_path>
          --from-vm <vm_id>

        [PARAMETERS] =
          --name <name>
          --cpu <cpu>
          --memory <memory>
          --vnet <vnet_id>
          --boot-disk <image_id>
          --data-disk <disk_size>
```

   b) Consulta de una máquina virtual.

```
$ cloud command –vm <vm_id> --get
```

   c) Borrado de una máquina virtual.

```
$ cloud command –vm <vm_id> --delete
```

d) Modificación del estado de una máquina virtual.

```
$ cloud command –vm <vm_id> [ACTIONS]

                    [ACTIONS]=
                       --start
                       --stop
```

3. **Funciones que permiten el manejo de imágenes:**

a) Subida de una imagen.

```
$ cloud command --create-volume [PARAMETERS]

                  [PARAMETERS]=
                       --name <name>
                       --path <image_path>
                       --multipart
```

b) Consulta de una imagen.

```
$ cloud-command --volume <volume_id> --get
```

4. **Funciones para consultas globales:**

a) Consulta Cluster.

```
$ cloud-command --cluster <cluster_id> --get
```

b) Consulta VDC.

```
$ cloud.command --vdc <vdc_id> --get
```

c) Consulta de imágenes.

```
$ cloud command –templates
```

Para poder entender el funcionamiento de alguna de estas órdenes. A continuacion se van a desarrollar dos ejemplos ilustrados, de cómo se usan y cuál es el resultado que se debería obtener.

5.0.11. Ejemplos de uso

Creación de una máquina virtual

Lo que sigue es el desarollo de un ejemplo completo para la creación de una máquina virtual, incluyendo la creación de una red y la imagen que se utilizará en la máquina virtual.

Para la creación de la red, primero se deberá arrancar tanto OpenNebula como el servidor de Sun ejecutando las ordenes siguientes:

```
$ one start
$ sun-server start
```

Una vez arrancado el servidor deberemos ejecutar el siguiente comando para poder crear una red virtual:

```
$ cloud-command --create-vnet --name ejemplo1
```

Para hacer que el ejemplo sea más completo también se va a asociar una imagen a la máquina virtual, por lo que es necesario cargar una imagen para su posterior uso.

Mediante el siguiente comando subimos la imagen al servidor para que de este modo podedamos asociarla a la máquina que vamos a crear.

```
$ cloud-command --create-volume
                --name test_image
                --path /srv/cloud/images/ttylinux/ttylinux.img
```

Una vez hecho esto, debemos comenzar con la creación de la máquina virtual. Se plantean tres opciones distintas a la hora de crear una nueva máquina:

1. Crearla nueva.

2. Crear una copia de una ya existente.

3. Uso de un template.

En este ejemplo se usará la última opción por lo que, antes de ponernos con la creación, es necesario rellenar el archivo large.json dentro de la carpeta de templates con el siguiente contenido:

```
{
"cpu" : "2",
"memory" : "1024",
"params" : {
                    "vnets" : "5"
},
"boot_disk" : "https://cloud.server/volumes/2"
}
```

En este punto se puede crear una nueva máquina virtual a través del siguiente comando :

```
$ cloud-command --create-vm --from-template large --name test_vm
```

Para verificar si la máquina virtual se ha creado de manera satisfactoria ejecutamos la siguiente instrucción.

```
$ cloud command --vm <id> --get
```

Borrado de la máquina, red e imagen creadas anteriormente

Los siguientes comandos realizan el borrado del elemento correspondiente mostrando la información pertinente dependiendo del comando ejecutado.

1. Borrado de una máquina virtual

```
$ cloud command --vm <id> --delete
```

2. Borrado de una red

```
$ cloud command --vnet <id> --delete
```

Una vez realizadas las pruebas, cierre del servidor de One y Sun.

```
$ sun-server stop
$ one stop
```

Capítulo 6

Conclusiones y Trabajo Futuro

En esta implementación del API de Sun se han dejado sin implementar las llamadas relacionadas con la gestión de "Public Addresses", "Backupz "Snapshots"por encontrarse fuera de los objetivos de este proyecto y no tener una correlación directa con las funcionalidades de OpenNebula.

Entre las mejoras aplicables a esta implementación se encuentran:

1. La gestión de redes y la información obtenida de las mismas puede ser ampliable. En estos momentos para permitir un manejo más dinámico y amplio sobre las redes sería conveniente el uso de la CLI del propio OpenNebula.
2. La gestión de volúmenes se ha hecho mediante llamadas http en lugar de hacer uso del sistema WebDAV debido a que el manejo de esta última forma implicaba una complicación excesiva en comparación al sistema implementado.
3. Las acciones sobre las máquinas implementadas han sido solamente start y stop, dejando sin implementar resume, hibernate y reboot. Estas acciones no han sido implementadas ya que no dependen sólo de OpenNebula para poderse llevar a cabo, sino que dependen del hipervisor que se hace uso para el control de estas máquinas.

Al tratarse de un borrador de la especificación del API de Sun Cloud, no quedan cerrados todos los matices referentes a la relación entre los diferentes recursos, por ello ha sido necesario realizar ciertas suposiciones a la espera de un próxima release"de la especificación.

Entre estas suposiciones se pueden encontrar:

1. La forma de definir las redes asociadas a una máquina virtual no queda reflejada en la implementación.
2. El boot_disk dentro de una máquina virtual hace referencia al tamaño del disco, pero sería conveniente que hiciese referencia a la imagen asociada con ese disco en lugar de a su tamaño.

Palabras clave

1. Cloud Computing.
2. Sun Cloud
3. Virtual Machine
4. Virtual Network
5. Volume
6. Rest
7. Json
8. Cloud
9. Cluster
10. Virtual Data Center

Glosario

1. **JSON** : Acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos
2. **Cloud Computing** : Es un paradigma que permite ofrecer servicios de computación a través de Internet. De modo que los usuarios puedan acceder a los servicios disponibles en el cloud
3. **Sun Cloud** : Plataforma Cloud computing de Sun Microsystems
4. **Virtual Machine** : Es un software que emula a una computadora y puede ejecutar programas como si fuese una real.
5. **Cloud** : En el ámbito del Cloud Computing se refiere a Internet y a todos los recursos que se pueden almacenar en él.
6. **Cluster** : Se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes, que se comportan como si fuesen una única computadora.
7. **Virtual Data Center** : Es un concepto bastante abstracto de definir y suele hacer referencia a un grupo lógico de aplicaciones virtuales. En el caso de nuestro proyecto almacenará tanto las máquinas y redes virtuales como las imágenes que se suban.
8. **API** : Una interfaz de programación de aplicaciones o API (del inglés application programming interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
9. **Google Apps** : Es un servicio ofrecido por Google para uso de sus diversos productos. En él se encuentran diversas aplicaciones para Internet, con un funcionamiento similar a los tradicionales programas para escritorio. Algunos ejemplos serían Gmail, Google Agenda, Talk, Docs y Sites.
10. **GoGrid** : Una empresa privada que ofrece un servicio de infraestructura de nube, alojando máquinas virtuales de Linux y Windows gestionadas por un panel de control multi-servidor.
11. **CPU** : La unidad central de procesamiento,es el componente que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.
12. **Ruby** : Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Su creador, Yukihiro "matz " Matsumoto, mezcló partes de sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada, y Lisp) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la programación imperativa.
13. **Sinatra** : Sinatra es un DSL escrito en Ruby para el desarrollo de aplicaciones web con un mínimo esfuerzo.

14. **DSL** : En el desarrollo de software a domain-specific language (DSL) es un lenguaje de programación o especificación de lenguaje dedicada al dominio de problema en particular.
    Una técnica de representación del problema en particular y / o una técnica de solución particular. El concepto no es nuevo, lenguajes de programación de proposito general y todo tipo de modelado y especificación de lenguajes han existido siempre, pero el término ha pasado a ser de los más populares debido a la subida de los modelos de dominio específico.

15. **Thin** : Es un servidor web Ruby que junta sus tres mejores librerías(Mongrel parser,Event Machine,Rack)

16. **x86** : Es la denominación genérica dada a ciertos microprocesadores de la familia Intel, sus compatibles y la arquitectura básica a la que estos procesadores pertenecen. Su origen está en la terminación de sus nombres numéricos: 8086, 80286, 80386, 80486, etc.

17. **W3C** : El World Wide Web Consortium (W3C) es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo.

18. **WSDL** : Web Services Description Language. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.
    Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

19. **CLI** : Interfaz de Línea de Comandos, por su acrónimo en inglés de Command Line Interface (CLI), es un método que permite a las personas dar instrucciones a algún programa informático por medio de una línea de texto simple.
    Debe notarse que los conceptos de CLI, Shell y Emulador de Terminal no son lo mismo, aunque suelen utilizarse como sinónimos.

20. **Scheduler** : El planificador (o scheduler en inglés) es un componente funcional muy importante de los sistemas operativos multitarea y multiproceso, y es esencial en los sistemas operativos de tiempo real.
    Su función consiste en repartir el tiempo disponible de un microprocesador entre todos los procesos que están disponibles para su ejecución.

Anexo: Guía de Instalación y Configuración

Guía de Instalación

Guía de configuración 1.4 del servicio Sun Cloud

OpenNebula Sun es un servicio web que permite el manejo y control de máquinas virtuales sobre una instalación de OpenNebula usando el API definido por Sun.

Este servicio está implementado sobre la capa OpenNebula Cloud API (OCA) que pone a disposición todas las funcionalidades de un "cloud" OpenNebula privado.

Requisitos e instalación

Como requisito inicial es necesario tener OpenNebula correctamente instalado y configurado para poder instalar el servicio Sun Cloud. Para llevar a cabo esta tarea, creando un cloud privado, se puede seguir la guía ubicada en la página web de OpenNebula [5]

Sun Cloud, a diferencia de otras implementaciones de servicios web integrados en OpenNebula como pueden ser OCCI o EC2 Query, se trata de un paquete autónomo. Por ello hay que llevar a cabo los siguientes pasos previos a la configuración del servicio.

1. **Descargar el source del servicio Sun Cloud:**

```
$ svn co http://svn.dsa-research.org/sun-cloud/trunk
```

2. **Ejecutar:**

```
$ ./install.sh <install_options>
```

donde <install_options>puede ser uno o más de:

a) -u : Usuario que ejecutará Opennebula.

b) -g : Grupo de usuario que ejecutará opennebula.

c) -k : Mantener archivos de configuración. Será útil a la hora de actualizar.

d) -d : Directorio de instalación. Si se define especificará el directorio para la instalación "self-contained". La instalación será "system wide"

e) -h : Muestra la ayuda de instalación

Una vez instalado el servicio Sun Cloud junto a OpenNebula se deben instalar los siguiente paquetes para cubrir las dependencias de ejecución.

```
$ sudo gem install thin
$ sudo gem install sinatra
$ sudo gem install json
```

Para el "Storage Repository" y las herramientas cliente también son necesarias las siguientes librerías. El uso de "curb" permite una subida de archivos de un forma más rápida:

```
$ sudo gem install sequel
$ sudo gem install curb
$ sudo gem install multipart-post
$ apt-get install ruby-sqlite3
$ apt-get install libopenssl-ruby
```

Configuración

El servicio de Sun es configurado a través del archivo $ONE_LOCATION/etc/sun-server.conf, donde se pueden ajustar los siguientes parámetros:

1. Administration Account, el servidor web necesita llevar a cabo ciertas operaciones usando la cuenta oneadmin, por ejemplo chequear la identidad de los usuarios del cloud. Hay que especificar el USER y el PASSWORD de oneadmin

2. Connection Parameters, el servicio xml-rpc del demonio oned, el servidor y el puerto en el que el servidor Sun estará escuchando.

3. Volume Repository, este servicio permite la gestión de imágenes de forma fácil, para ello es necesario especificar la DATABASE y el IMAGE_DIR del servicio.

4. Virtual Networks, el nombre del bridge al cual tiene que conectar la máquina virtual para así conseguir la conexión a la red y la ip de la red.

5. Filesystem format, formato por defecto al que los sistemas de ficheros vacíos serán formatados. En caso de omisión el formato por defecto será ext3.

6. JSON templates, templates disponibles para la creación de máquinas virtuales.

```
# OpenNebula administrator user
USER=user
PASSWORD=pass

# OpenNebula server contact information
ONE_XMLRPC=http://localhost:2633/RPC2

# Host and port where the sun-server will run
SERVER=127.0.0.1
PORT=4567

# SSL proxy that serves the API (set if is being used)
SSL_SERVER=https://localhost:443

# Configuration for the image repository
DATABASE=/srv/cloud/one/var/sun.db
IMAGE_DIR=/srv/cloud/sun/images

# Configuration for OpenNebula's Virtual Networks
BRIDGE=vbr1
NETWORK_ADDRESS=192.168.0.0

# Default format for FS
FS_FORMAT=ext3

# VM types allowed and its template file (inside templates directory)
VM_TYPE=[NAME=template, TEMPLATE=template.erb]

JSON_TEMPLATE=[NAME=small, TEMPLATE=small.json]
JSON_TEMPLATE=[NAME=medium, TEMPLATE=medium.json]
JSON_TEMPLATE=[NAME=large, TEMPLATE=large.json]
```

Configurando un proxy SSL

La especificación de Sun obliga al uso de la seguridad SSL en todas sus conexiones. Para llevar a cabo esta tarea se puede montar un servidor proxy que se encargue de la recepción de las peticiones al servicio Sun y su posterior envio de la respuesta correspondiente.

Para esta configuración es necesario:

1. Un certificado para las conexiones SSL
2. Un proxy HTTP que entienda SSL
3. El servicio Sun configurado para aceptar peticiones del proxy

Para más información de cómo establecer esta configuración se puede recurrir a las guías ubicadas en la página web de OpenNebula, dentro de las configuraciones de EC2 u OCCI indistintamente. [6]

Arranque y parada del servidor

Para lanzar el servidor Sun simplemente basta con ejecutar el siguiente comando:

```
$ sun-server start
```

El log del servidor de sun se encuentra en el archivo $ONE_LOCATION/var/occi-server.log en caso de una instalación "standalone", o en /var/log/one/occi-server.log en una instalación "system-wide".

Por último para detener el servidor:

```
$ sun-server stop
```

Bibliografía

[1] http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/

[2] http://forge.ogf.org/sf/docman/do/listDocuments/projects.occi-wg/docman.root.drafts.occi_specification

[3] http://communities.vmware.com/community/developer/documentation#

[4] http://www.vmware.com/appliances/services/vcloud-express.html

[5] http://opennebula.org/documentation:documentation

[6] http://opennebula.org/documentation:rel1.4:occicg

http://kenai.com/projects/suncloudapis/pages/Home

http://es.wikipedia.org/wiki/Wikipedia:Portada

http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

http://dsa-research.org/doku.php?id=people:ruben

http://www.vmware.com

http://www.opennebula.org/

http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

http://www.xfront.com/REST-Web-Services.html

http://java.sun.com/developer/technicalArticles/WebServices/restful/

http://www.infoq.com/articles/rest-introduction

http://www.occi-wg.org/doku.php

http://aws.amazon.com/ec2/

http://www.vmware.com/go/vcloudapi

http://incubator.apache.org/libcloud/

http://deltacloud.org/