Implementación de un controlador de VirtualBox para OpenNebula

Héctor Sanjuan Redondo Pablo Donaire Calleja

David Rodríguez González

Facultad de Informática
Sistemas informáticos

Director de proyecto
Rubén Manuel Santiago Montero
Universidad Complutense de Madrid
Curso 2010-2011

Índice general

Declaración de conformidad

Los alumnos Héctor Sanjuan Redondo, Pablo Donaire Calleja y David Rodríguez González, aquí firmantes, autorizan a la Universidad Complutense de Madrid a la difusión de la memoria, implementación, y código del proyecto realizado con fines exclusivamente académicos y mencionando expresamente a los autores del mismo.

HECTOR SANJUAN REDONDO

PABLO DONAIRE CALLEJA

DAVID RODRIGUEZ GONZALEZ

Agradecimientos

David:

Me gustaría agradecer a mis compañeros, Héctor y Pablo, su ayuda, apoyo y trabajo realizado en esta bonita etapa de mi vida, que refleja la culminación de una fase importante de formación de muchos años. Sin ellos, y sin todos mis compañeros, habría sido muy complicado llegar a este punto.

Por otro lado, querría dar las gracias a Rubén, nuestro director de proyecto, su entrega y dedicación a la hora de dirigirnos a alcanzar este objetivo, del cual me siento muy orgulloso. Gracias a él y a todos los profesores que han ayudado a que adquiera los conocimientos necesarios para llevar a cabo todo aquello que me hace mayor ilusión.

Por último, no quisiera pasar por alto a mi familia, sin ella, habría sido imposible formarme profesionalmente y alcanzar las metas que me he propuesto hasta ahora.

Pablo:

A todas esas personas que han compartido asiento conmigo en su época de Universidad, haciendo de los malos momentos algo pasajero y de los buenos algo memorable. En especial, destacar a mis compañeros de proyecto, por haber conseguido que esto no sea nada más que un proyecto fin de carrera.

A aquellos que se pusieron delante de mí en la pizarra, desde mi niñez hasta el momento en el que escribo estas líneas, por haber conseguido que cumpliera mis objetivos.

Y por último, y no por ello menos importante, a mi familia. Por soportar mis malos humos en los momentos de desesperación, por tener esas palabras de ánimo en los momentos de flaqueza, por mantenerme siempre con los pies en el suelo, por millones de cosas más.

A todos, sin excepción, GRACIAS.

Héctor:

Cita ineludible resulta, por más que retrasarla intento, la hora de escribir las líneas siguientes de agradecimiento.

Puesto al tajo, agradezco sin dilación a mis compañeros de trabajo. Han sabido hacer aquello que les correspondía, no sólo en este proyecto, sino en el resto de sus asignaturas. Todo un ejemplo ante quien ha afrontado el curso con el proyecto como única andadura.

Agradezco al mismo tiempo al equipo de OpenNebula al completo. El interesante trabajo que me han dado ha ido bien acompañado de su soporte constante.

No me permito olvidarme del profesorado, que desde las demostraciones por inducción, a las gramáticas de traducción me han formado hasta ser capaz de entender aquello que a partir de ahora me toque aprender.

Y agradezco finalmente a mi padre y a mi madre, mecenas de todo lo hecho, y que no desisten en su empeño de hacer de su retoño una persona mejor.

0.1. Resumen

El «Cloud Computing» («Computación en la Nube») se ha convertido en el nuevo paradigma tecnológico de la década. Las ventajas son palpables -optimización de los recursos, diversificación de los servicios, independencia real entre hardware y software, adaptación en tiempo real a las demandas de capacidad y la flexibilidad...- y han hecho que esa cosa conocida como «la nube» se esté convirtiendo en el lugar en el que muchos proveedores almacenan sus datos y desde el que ofrecen sus servicios.

Este proyecto se sumerge de lleno en «la nube» y explora tanto el ámbito de aplicación -publico/privado-, como los diferentes servicios para los que se utiliza: software como servicio «SaaS», plataforma como servicio «PaaS», infraestructura como servicio «IaaS». Centrándose en este último, se estudian algunas de las aplicaciones software que hacen posible la IaaS, es decir, que posibilitan la utilización de diversos recursos físicos (procesadores, memorias, discos duros) por múltiples clientes (sistemas operativos, programas...): OpenStack, Eucalyptus, VMware y, más en profundidad, OpenNebula.

De la mano de esta última, revisamos las soluciones de virtualización (hipervisores) más populares y que permiten la utilización del hardware por múltiples sistemas operativos al mismo tiempo. Centrarán nuestra atención KVM, XEN, VMware y, sobre todo, VirtualBox.

El grueso de nuestro trabajo estará en el desarrollo de un «driver» de VirtualBox para OpenNebula. Mediante este driver, OpenNebula será capaz de crear, monitorizar, administrar y migrar máquinas virtuales que funcionan sobre VirtualBox.

El driver de VirtualBox, escrito en Ruby e integrado a modo de «plugin», será totalmente independiente de OpenNebula y fácilmente instalable. Tomando como base las descripciones en XML de máquinas virtuales generadas por OpenNebula, interactuará directamente con VirtualBox a través de la interfaz de línea de comandos que ofrece. Evitaremos así la utilización de capas intermedias (como libvirt), y procuramos un mayor control sobre las acciones que se realizan.

De esta manera, conseguimos ampliar las posibilidades de utilización de OpenNebula con una característica demandada por su comunidad de usuarios, aumentando su versatilidad y habilitando a VirtualBox como una opción más de virtualización dentro del campo de la «IaaS».

Palabras clave: VirtualBox, virtualización, OpenNebula, IaaS, infraestructura, servicio, cloud, computing, driver.

0.2. Abstract

Cloud Computing has become the new technological paradigm of the decade. The advantages are real (resource optimization, diversification of services, real independence between hardware and software, real-time response to the capacity and flexibility demands) and have made that many providers keep their data and offer their services from that thing known as "The Cloud".

This project dives into The Cloud, and explores both the public/private use case and the different services supported: Software As A Service (SaaS), Platform As A Service (PaaS), Infrastructure As a Service (IaaS). Paying attention to the latter, some of the applications making IaaS possbile are studied. These applications allow the use of several physical resources (processors, memories, storage) by multiple clients (operating systems, software...). OpenStack, Eucalyptus, VMware and OpenNebula, in which we will go in depth, are some of them.

Hand in hand with OpenNebula, we will explore the most popular virtualization solutions (hypervisors) which allow the use of a single piece of hardware by several operating systems at the same time, together with the management and assignment of its resources. We will mention KVM, XEN, VMware and above all, VirtualBox.

Our work will be focused on the development of a VirtualBox driver for OpenNebula. Using this driver, OpenNebula will be able to create, monitor, manage and migrate virtual machines running on VirtualBox.

The VirtualBox driver, written in Ruby and pluggable to the OpenNebula installation, will be totally independent and will come with an easy setup. Taking the XML virtual machine's descriptions generated by OpenNebula as the starting point, the driver will communicate directly with VirtualBox through its command line interface. This way, we will avoid using extra layers (such as libvirt) and gain greater control over the actions that are runned.

By making VirtualBox one more option for IaaS virtualization, we will manage to increase OpenNebula's versatility and use cases and give answer to a feature which was requested by its user community.

Keywords: VirtualBox, virtualization, OpenNebula, IaaS, infrastructure, service, cloud, computing, driver.

Capítulo 1

Introducción y objetivos

1.1. Introducción

Este proyecto de Sistemas Informáticos consiste en la realización de un «driver» para OpenNebula que permite a la aplicación la gestión de máquinas virtualizadas utilizando el hipervisor VirtualBox.

Se enmarca, por tanto, en el ámbito del «Cloud Computing» («Computación en la nube»), concepto relativamente nuevo, de moda en la actualidad y que posiblemente se convierta en una tecnología clave del avance de la computación durante al menos toda una década, como lo fueron anteriormente los conceptos de «mainframe» o los microprocesadores en los pasados años.

La carrera por ofrecer más y mejores capacidades está dejando paso a un concepto que aboga, ante todo, por el aprovechamiento máximo de los recursos existentes, pero también por la flexibilidad, la simplicidad y la coexistencia de diferentes sistemas operativos sobre un mismo hardware. Esto se consigue gracias a una capa de software que virtualiza los recursos físicos, y que es capaz de gestionarlos y ofrecerlos simultáneamente a varios sistemas huésped.

Gracias a la virtualización, es posible exprimir las capacidades del hardware, pero también de ser completamente independientes de él. Así, un servicio puede ahora ejecutarse en múltiples máquinas, de muy diferentes características, en convivencia con otros servicios y al mismo tiempo ser totalmente estanco y ajeno a este hecho. Aún mejor, es posible, trasladar este servicio desde una máquina física a otra, incluso en ejecución («live migrate», «VMotion»...), sin que ello suponga una reconfiguración, recompilación o reinicio del servicio.

La “nube”, y de ahí su nombre, permite olvidarnos de dónde se ejecutan los servicios. Y por servicios, se puede entender desde sistemas operativos, a aplicaciones varias (cortafuegos, servidores) o simple almacenamiento de datos. Lo importante para el usuario exterior es que están ahí, en la ”nube”. Por su lado, lo importante para el administrador es que aprovecha sus re-

cursos al máximo y puede adaptarse fácilmente a los cambios de demanda que le sean necesarios.

Desde esta perspectiva, la lógica que asignaba un servidor a un servicio ya no es válida. Los más avezados en este campo, como Amazon, ofrecen todo lo necesario para que todo tipo de clientes aprovechen una infraestructura física compartida entre todos («nube pública») y en la que, previo pago, pueden adquirir la potencia o capacidad necesaria en cada momento. Las «nubes híbridas» o «nubes privadas» ofrecen recursos que no están per se disponibles para un público general, pero siguen el mismo principio.

El éxito del «Cloud Computing» se demuestra con la proliferación tanto de empresas dedicadas a facilitar su uso a sus clientes en sus múltiples formas, como en su implantación para ofrecer los propios servicios, como en el aumento de herramientas de software que hacen posible que la nube sea una realidad, en sus diferentes niveles.

En este proyecto trabajaremos con dos herramientas clave. La primera, OpenNebula, una aplicación de código abierto desarrollada en la Universidad Complutense y que hace realidad un tipo de nube en la que el servicio ofrecido es la la propia infraestructura («IaaS»). OpenNebula es una navaja suiza del «Cloud Computing». Permite administrar máquinas físicas, redes, e imágenes (almacenamiento). Con todo ello, es capaz de ejecutar y gestionar máquinas virtuales, debidamente configuradas, monitorizando sus recursos y, a través de las políticas de emplazamiento adecuadas, decidiendo sobre qué hardware real van a funcionar.

OpenNebula destaca por su flexibilidad para adaptarse a múltiples escenarios. Parte de esta flexibilidad se consigue permitiendo la utilización de varios sistemas de virtualización o hipervisores como KVM, XEN o VMware.

La estructura modular y orientada a plugins de OpenNebula nos permite extender sus características y dar soporte a un hipervisor más, VirtualBox, antiguo proyecto de Sun Microsystems y propiedad de Oracle en la actualidad.

Con este trabajo respondemos a una de las demandas de la activa comunidad de usuarios de OpenNebula y ofrecemos una solución que, más allá de estar en papel, esperamos sea utilizada, probada y mejorada por los usuarios.

1.2. Objetivos

Este proyecto tiene por objetivo el desarrollo de un «driver» que permita la utilización del hipervisor VirtualBox con OpenNebula con las siguientes características:

- El «driver» deberá ser independiente de la distribución original de OpenNebula e integrarse a modo de «plugin».
- Siguiendo la estructura de los «drivers» existentes, estará compuesto por un gestor de información (im_mad) que sea capaz de monitorizar las características de las máquinas físicas (memoria, cpu), y de un gestor de máquinas virtuales (vmm_mad) capaz de realizar las operaciones comunes sobre máquinas virtuales (despliegue, salvaguarda, borrado, migración, pausa).
- Se utilizará una descripción XML de las máquinas generadas por OpenNebula y se traducirán los parámetros especificados para configurar las máquinas en VirtualBox.
- Se deberá documentar apropiadamente el «driver» para facilitar su uso y mejora por parte de la comunidad de usuarios de OpenNebula.

Aprovechamos, en la memoria de este proyecto, para colocar el «driver» en un contexto apropiado repasando el concepto de «Cloud Computing», las aplicaciones «IaaS» más populares, con especial atención a OpenNebula y los hipervisores que hacen posible la virtualización.

1.3. Organización de la memoria

La memoria se organiza en los siguientes apartados:

- «Cloud Computing»: En este apartado se realiza un repaso al concepto de Computación en la Nube. Se comentan los diferentes tipos de nubes (SaaS, PaaS, IaaS) y se describen algunas aplicaciones de administración de nubes IaaS similares a OpenNebula.
- Virtualización: En este apartado se revisa el concepto de virtualización y se comentan algunos de los hipervisores más utilizados, con especial mención a VirtualBox.
- Arquitectura OpenNebula: Esta sección repasa el funcionamiento interno de OpenNebula y su organización.
- Desarrollo: Este capítulo explica con detenimiento el desarrollo del «driver» de VirtualBox para OpenNebula, su organización y la forma en la que se desarrollan las acciones y cómo utilizarlo.

- Conclusiones: Cierran esta memoria algunas conclusiones sobre el trabajo realizado y el futuro del «driver».
- Anexos: Se incluye una guía de instalación del «driver» y un ejemplo de uso.

Capítulo 2

«Cloud Computing»

2.1. Introducción

Para entender qué es el «Cloud Computing» es necesario estudiar las distintas definiciones que han ido surgiendo desde que apareció en escena, dado que, al ser un concepto novedoso, ni siquiera los expertos del sector encuentran un punto en común:

"El «Cloud computing» sería la tendencia a basar las aplicaciones en servicios alojados de forma externa, en la propia web."

"Podemos entender «Cloud Computing» como un nuevo concepto tecnológico que se basa en que las aplicaciones software y los equipos hardware están ubicados en un «Datacenter» que permite a los usuarios acceder a los aplicaciones y servicios disponibles a través de Internet de una forma sencilla y cómoda."

"«Cloud computing» es un paradigma que permite ofrecer servicios de computación a través de Internet"

A la vista de estas definiciones, se puede concluir que es un nuevo modelo de prestación de servicios y tecnología, ofrecidos a través de Internet y alojados de manera externa a la computadora del propio usuario.

¿Es el «cloud computing» algo realmente novedoso actualmente? En los años 90, podemos encontrar los primeros indicios de uso de este tipo de tecnología, por ejemplo los cajeros automáticos, donde los usuarios podían realizar operaciones y obtener servicios desde cualquier terminal, utilizando un software y un hardware externos a ellos. Más recientemente, encontramos más ejemplos como el correo personal, páginas de alojamiento de imágenes y ficheros... y un largo etcétera.

Entonces, ¿Por qué se produce este «boom» ahora? Para entenderlo, vamos a analizar las ventajas e inconvenientes de su uso, aplicado a un contexto económico actual donde priman las dificultades de las empresas, que ven en este modelo de negocio una oportunidad.

2.2. Ventajas e inconvenientes

2.2.1. Ventajas

- Facilidad de integración. Al no existir diferentes versiones del software y estar todo más centralizado permite que a la hora de integrar distintos componentes se tenga una mayor facilidad para llevarlo a cabo.
- Prestación de servicios a nivel mundial, con una recuperación de desastres completa. Al prestarse estos servicios a través de Internet, el acceso a los mismos está universalizado.
- Contribuye al uso eficiente de la energía. En una estructura «Cloud Computing» cada usuario paga o utiliza los recursos que necesita y no más, esto permite un aprovechamiento de los recursos, que en definitiva se traduce en un ahorro de energía.
- Implementación rápida y con menos riesgos, con actualizaciones automáticas. Al estar todo más centralizado, todas las operaciones se pueden realizar desde un mismo punto, y programar actualizaciones de una manera más eficiente.
- Simplicidad, traducida a un menor coste, ya que una estructura 100 % «Cloud Computing» (tanto la infraestructura como las aplicaciones se encuentras alojadas en servidores externos en su totalidad y se accede a todo ello a través de Internet) no requiere instalar ningún tipo de hardware.
- Escalabilidad. Permite incorporar a la estructura «Cloud» nuevos elementos sin que el funcionamiento general se vea por ello perjudicado.

2.2.2. Inconvenientes

- Dependencia de los proveedores de servicios, debido a la centralización de los datos. Si estos proveedores fallan, toda la estructura del cliente alojada en sus servidores se verá afectada en gran medida.
- Dependencia de acceso a Internet para acceder a dichos servicios. Actualmente se trabaja en mejorar en este aspecto, de tal manera que el usuario pueda trabajar sobre una copia de seguridad de sus datos realizada con anterioridad.
- Vulnerabilidad a la sustracción de los datos debido a que la información no se aloja en los servidores propios del usuario o de la empresa. En el caso de nubes públicas, podrían alojarse en una misma máquina datos de distintas empresas o clientes y de esta manera afectar a la seguridad. De hecho, es uno de los grandes problemas actualmente, ya que los clientes necesitan saber dónde se encuentran sus datos para sentirse más seguros.
- Seguridad. La información recorre muchos nodos donde cada uno de ellos puede representar un riesgo. Si se usan protocolos de seguridad como HTTPS disminuye la velocidad por la sobrecarga producida.

Es de esperar que al ser una tecnología reciente, existan desventajas importantes, pero cabe destacar que se está avanzando notablemente en estos aspectos consiguiendo reducir su impacto.

2.3. Tipos de nubes

Encontramos tres tipos de nubes: pública, privada, e híbrida, las cuales se diferencian por el tipo de cliente que las utiliza o dónde se encuentra alojada la información.

2.3.1. Nube pública

En este tipo de nube, la información se encuentra totalmente alojada externamente al usuario, el cuál puede acceder a los servicios de forma gratuita o de pago.

Su principal ventaja reside en el hecho de que el usuario no se tiene que preocupar de su mantenimiento, coste, actualizaciones...etc y el coste de su adquisición es bastante bajo, pero tiene un inconveniente importante: el acceso y salvaguarda de la información por parte de entidades ajenas y dependencia total del acceso a Internet.

Además, en el apartado económico, se gana capacidad de procesamiento y almacenamiento sin tener que instalar nada, por lo que toda la carga de las operaciones realizadas recae sobre el hardware y software del proveedor. De esta forma, la inversión inicial es bastante menor y puede ser interesante para muchos clientes que no necesiten almacenar datos sensibles.

Muchas empresas en la actualidad, a pesar de las ventajas económicas que ofrecen, son reacias a utilizar este tipo de nubes, al no ser posible tener un control sobre dónde se almacenan sus datos.

2.3.2. Nube privada

A diferencias de las públicas, el despliegue de este tipo de tecnología se encuentra propiamente dentro de las instalaciones del usuario, por lo que en raras ocasiones se depende de terceros. Por tanto, existe un menor riesgo en la seguridad de la información, cuyo precio se paga en forma de mantenimiento, actualizaciones e instalación de la estructura.

Es lógico que una empresa haya hecho una inversión inicial moderada empezando con nubes híbridas, para luego poco a poco, según vaya creciendo la demanda y las posibilidades económicas aumenten, pasarse a la nube privada.

Las nubes privadas dan la posibilidad de conocer dónde se almacenan físicamente los datos. Esto, a pesar de no tener porque ser necesario para la operabilidad, permite ajustarse a aquellas legislaciones que obligan un control estricto sobre datos sensibles con el requerimiento, por ejemplo, de que los datos estén dentro del territorio nacional.

2.3.3. Nube híbrida

En este tipo de nubes se intenta aprovechar las ventajas económicas de una nube pública como las ventajas en cuestión de seguridad de las nubes privadas. Por ejemplo, muchas empresas han visto que es más económico usar un «IaaS» (Infrastructure As A Service), como por ejemplo «Amazon Simple Storage Service (S3)», para almacenar imágenes, vídeos y documentos al tiempo que mantienen datos sensibles en su propia infraestructura.

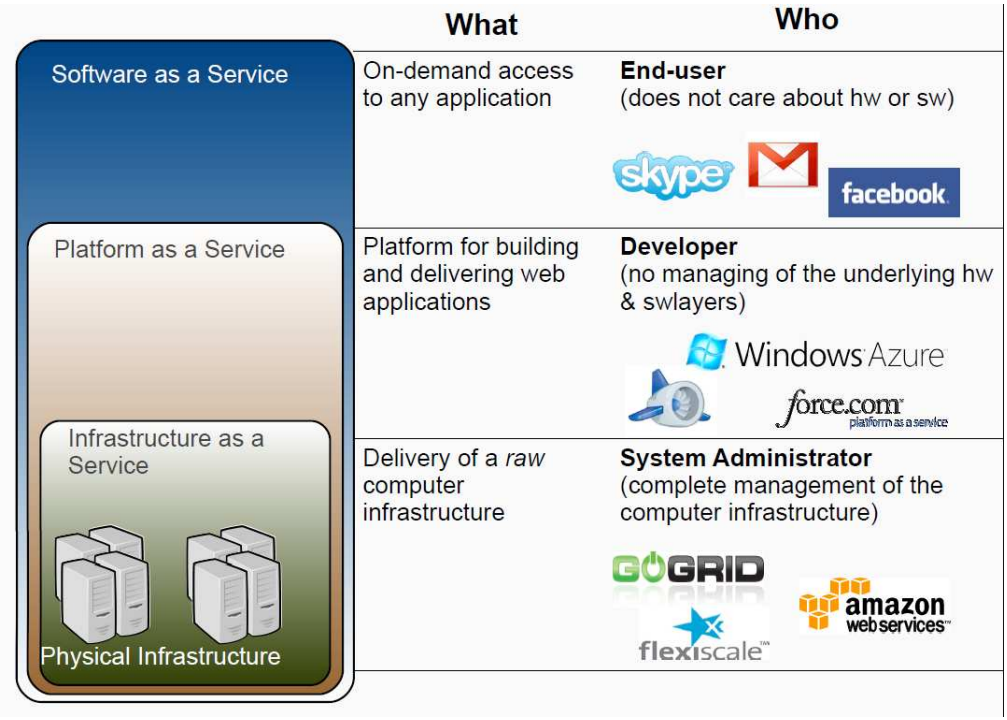
El modelo híbrido también se presta a un enfoque incremental, ya que se parte de una inversión moderada y se pueden trasladar aplicaciones desde servidores externos a servidores propios del usuario según convenga.

Tiene la ventaja principal de una inversión inicial moderada y a la vez contar con SaaS, PaaS y IaaS según el cliente vaya necesitando. Si a la hora de querer ampliar la capacidad de la empresa, se encuentra con necesidades que permanecerán a lo largo del tiempo, quizás puede ser interesante pasar estas nuevas capacidades a la nube propia, mientras que si es algo puntual, puede ser trasladado a la nube pública.

2.4. Capas o modelos de servicio

Una nube puede ofrecer distintos modelos de servicio, representados en distintas capas, analizando de esta manera la infraestructura del «Cloud».

Nos encontramos principalmente con tres tipos de capas o modelos de servicio, «SaaS»(Software As A Service), «PaaS»(Platform As A Service), «IaaS»(Infraestructure As A Service).



2.4.1. SaaS

«SaaS» es un modelo de distribución del software que proporciona a los clientes el acceso al mismo a través de la red (generalmente Internet), de

manera que les "aparta" del mantenimiento de las aplicaciones, de sus actualizaciones y de su soporte.

Las aplicaciones distribuidas en la modalidad «SaaS» pueden llegar a cualquier lugar. Se trata de un modelo que une el producto (software) al servicio, de esta manera el usuario puede optimizar sus recursos y su coste obteniendo una solución completa.

En este modelo, los usuarios pasan a ser "suscriptores" de este servicio, pagando si procede por su uso. Estos servicios se ofrecen de una manera pública de manera que puedan acceder a ellos multitud de clientes. De esta forma, se convierte en un modelo descentralizado de uso del software permitiendo una escalabilidad ilimitada.

Pero, ¿qué gana el usuario con «SaaS»?

- El usuario puede despreocuparse de los aspectos técnicos de los servicios que utiliza.
- Esto supone a su vez un menor coste e inversión inicial, al no tener que mantener infraestructura alguna.
- Alta escalabilidad asegurada
- La respuesta ante los cambios es muy rápida.

Los servicios «SaaS» se podrían agrupar en cuatro categorías:

- Soluciones de «Back Office» para agilizar tareas cotidianas o establecer mecanismos de control de actividades
- Soluciones de mensajería, de gestión de correos electrónicos, tratamiento de «SPAM», protección frente a virus, etc.
- Aplicaciones «CRM», para gestionar la relación con los clientes, poder obtener información para productos de mercado, etc.
- Soluciones de integración para gestión de datos.

Algunos ejemplos de «SaaS» podrían ser «SalesForce» o «BaseCamp», «Gmail»...

2.4.2. PaaS

Sigue la misma idea que el «SaaS» pero aplicado al hardware, es decir, intenta abstraer el hardware físico al cliente pero permitiendo tener acceso a los servicios que ofrece.

El modelo «PaaS» intenta abarcar el ciclo completo para el desarrollo de aplicaciones y su implantación en Internet. Para ello, aporta facilidades al cliente como las de prototipar, analizar, desarrollar, testear, documentar y poner en funcionamiento aplicaciones en un sólo proceso.

Para conseguirlo, «PaaS» da servicio de integración a bases de datos, seguridad, escalabilidad, almacenaje,copias de seguridad, control de versiones,... y todo ello a través de la red.

Dicho modelo debe tener dos características principales, como son la arquitectura multiusuario y el soporte para desarrollo colaborativo.

Con la arquitectura multi-usuario se pretende asegurar la escalabilidad del sistema al desarrollador mientras que con el soporte para el desarrollo colaborativo se pretende implementar y compartir código fuente entre varios desarrolladores que se puedan encontrar en diferentes puntos geográficos.

¿ Por qué contratar un servicio «PaaS»?

- De cara al propio desarrollador:

  Para implementar software se necesitan: BBDD, servidores, redes, y herramientas de desarrollo. Además, se necesita personal para mantener todo esto. Con «PaaS» permite olvidar esta parte y centrarse en innovar y desarrollar.

- De cara a los clientes de las aplicaciones del desarrollador:

  Los clientes se benefician también de este servicio ya que no se ven obligados a adquirir nuevo hardware o software, ni preocuparse de las licencias. Además, pueden tener acceso desde cualquier dispositivo en cualquier momento siempre que tengan conexión a Internet.

2.4.3. IaaS

Infraestructura como servicio se basa en proporcionar una infraestructura de computación a través de la cual puede obtenerse la posibilidad de adquirir la plataforma necesaria para montar servidores, capacidades para desplegar aplicaciones, manejo de recursos, satisfacer las necesidades de almacenamiento o incluso proporcionar equipamiento de red, todo ello virtualizado.

Por tanto IaaS nos permite abstraer algunas de las restricciones que puede llegar a imponer el hardware físico.

El usuario de «IaaS» puede aumentar y reducir el número de máquinas que utiliza a voluntad, es decir, es posible proporcionar los recursos en función de las necesidades particulares de cada momento.

Normalmente esta modalidad es de pago. Es decir, el usuario paga por la cantidad de recursos de almacenamiento y sólo éstos. De esta manera, el resto de recursos los podrá dedicar a lo que realmente le importa al cliente en su empresa.

Principales ventajas de «IaaS»:

- Consolidación mediante virtualización.
- Ventajas económicas. Sólo se utilizan y pagan los recursos que demande el cliente.
- Dedicación de los recursos "sobrantes" a otros aspectos del negocio.
- Olvidarse del hardware físico y su costoso mantenimiento.
- Todo el aprovisionamiento de estos servicios se hace a través de la red.

Principales desventajas de «IaaS»:

- Al realizarse todo aprovisionamiento del servicio a través de redes, se depende de la conexión para poder acceder a los recursos.
- Si la gestión de la infraestuctura está en manos de terceros, se crean dependencias fuertes con los proveedores del servicio. Fallos en la gestión pueden suponer problemas graves como la no operabilidad, la pérdida de datos, etc.

Llegados a este punto, vamos a pasar a analizar algunos ejemplos de tecnologías «IaaS» que están en funcionamiento en la actualidad, como son «OpenStack», «Eucalyptus», «VMware» y «OpenNebula».

OpenStack

OpenStack es una tecnología «IaaS» de código abierto. De esta forma, se consigue estandarizar y abrir la nube, evitando la privatización de las soluciones que ofrece el «Cloud Computing», que ayuda a fragmentar el mercado y ofrecen poca transparencia.

Se basa en las implementaciones «cloud» de «RackSpace» y el código Nebula de la NASA, las cuales comparten la filosofía del software libre. OpenStack surge debido a su necesidad de manejar grandes cantidades de datos.

OpenStack está dirigido a aquellas compañías que quieran usar los servicios de almacenamientos, gestión de recursos, gestión de redes... olvidándose del hardware físico, y también a aquellas compañías cuyas características les previene de usar la nube pública.

Hasta la fecha, más de 53 compañías han contribuido a su desarrollo en mayor o menor medida. Podemos destacar, ademas de las ya comentadas «RackSpace» y la NASA, otras como Dell, Intel o Cisco.

Eucalyptus

Al igual que OpenStack, Eucalyptus es una tecnología «IaaS» de código abierto para implementar nubes privadas.

Su nombre proviene de «Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems». Una de sus principales características es que su actual interfaz es compatible con la interfaz de la nube de «Amazon Web Services».

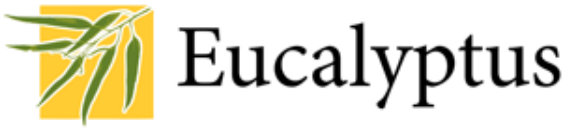
Eucalyptus nació a partir de un proyecto universitario en los laboratorios «MAYHEM» en Santa Bárbara. Dichos laboratorios tienen una amplia experiencia en computación «Grid», HPC y sistemas de alta escalabilidad, siendo un entorno ideal para el crecimiento de este software.

A partir de 2009 surgió como compañía. Actualmente, existen dos versiones, Eucalyptus (de código abierto) y Eucalyptus EE («Enterprise Edition»), que es la versión comercial del mismo.

Una pequeña muestra del éxito de esta tecnología es su integración en Ubuntu como herramienta «cloud computing» por defecto. Este éxito es debido en gran parte a su alta escalabilidad, su fácil instalación y un mantenimiento poco costoso.

Algunas otras características son:

- Diseño modular.
- Alta escalabilidad.
- Flexible.
- Soporta hipervisores «KVM» y «Xen». Además, en su edición «Enterprise» también se soporta «VMware».



VMware

VMware ofrece soluciones a nivel de «IaaS», como su software «vCloud Express».

Los servicios de centros de datos de VMware «vCloud» aprovechan modelos de infraestructura, administración y seguridad coherentes de manera global, lo que permite a los clientes empresariales trasladar rápidamente y sin dificultades cargas de trabajo entre la infraestructura virtualizada interna y una nube externa, en ambos sentidos.

Dicha iniciativa fue lanzada en septiembre de 2008, y desde entonces ha consguido ser usada por más de 1000 proveedores de servicios, como Terremark, BlueLock, Hosting.com, Logica y Melbourne IT.

OpenNebula

OpenNebula es un software de código abierto desarrollado por la Universidad Complutense de Madrid desde finales de julio de 2008 que permite controlar y gestionar distintas máquinas virtuales en la nube.

La flexibilidad que ofrece para poder operar sobre distintas redes, unidades de almacenamiento o hipervisores, permite crear una infraestructura «cloud» rápidamente, ya sea a nivel público, privado o híbrido.

En cuanto a sus capacidades, mencionar su gran adaptabilidad para «datacenters», su escalabilidad, una destacable interoperabilidad por hacer uso de los estándares y de las API's cloud más populares además de dar soporte a los hipervisores más conocidos.

La participación en grandes proyectos y su desarrollo «open-source», ha propiciado que distintas empresas y proyectos de renombre hagan a dia de hoy uso de los servicios que ofrece. Como ejemplos, podemos mencionar empresas del sector de la comunicación como Telefonica I+D o China Mobile,

servicios de consultoría financiera como «KPMG», centros de supercomputación como CESGA o SARA o centros de investigación como FermiLab o el CERN.

Capítulo 3

Virtualización

3.1. Introducción

La virtualización es una tecnología cada vez más presente en nuestro ambiente de trabajo, ya que permite separar el hardware del software, lo cual posibilita ejecutar en una misma máquina simultáneamente varios sistemas operativos, aplicaciones o plataformas de cómputo, aunque lo más común en este campo es el uso de una aplicación software para permitir que nuestro propio sistema operativo sea capaz de autorizar el manejo de varias imágenes de sistemas operativos diferentes.

Por tanto, se puede decir que la virtualización es una técnica que trabaja sobre las características físicas de la máquina para ocultarlas al usuario, lo que implica que un mismo recurso físico pueda ser a la vez varios recursos lógicos, o incluso que varios recursos físicos pueden aparecer a su vez como un único recurso lógico.

Se puede clasificar la virtualización en dos grupos:

3.1.1. Virtualización de plataforma

El objetivo principal de la virtualización de plataforma es la creación de máquinas virtuales para poder ejecutar sobre una única máquina física diversos sistemas operativos.

A su vez, hay varios tipos de virtualización de plataforma:

- Emulación del hardware:

  Se basa en virtualizar todo el hardware presente en una máquina real e instalar sobre éste el sistema operativo elegido. Es la tecnología menos eficiente y más costosa ya que es necesario crear todos los componentes de un ordenador y traducir todas las peticiones de uso del hardware que haga el sistema invitado al hardware real.

- Virtualización con hipervisor:

  Este tipo de virtualización de plataforma es el que más éxito ha alcanzado, ya que permite que los sistemas operativos invitados se ejecuten

en algunos casos igual que si estuvieran instalados de forma nativa. Se basa en la implementación de una capa intermedia entre el hardware y el sistema operativo anfitrión denominada hipervisor.

- Virtualización a Sistema Operativo:

  Se genera un entorno dentro de un sistema operativo en el cual se virtualizan aplicaciones.

- Virtualización a nivel de librerías:

  Se crea un entorno virtual en el que las aplicaciones pueden ser ejecutadas. «Wine» es el referente en este tipo de virtualización y actúa como una «API» de Windows para GNU/Linux.

3.1.2. Virtualización de recursos

Esta permite agrupar varios dispositivos, generalmente medios de almacenamiento para que sean vistos como uno solo, o dividir un recurso en múltiples recursos independientes.

Una forma muy conocida de este tipo de virtualización son las redes privadas virtuales o VPN, una tecnología de red que permite una extensión de la red local sobre una red pública o no controlada, como por ejemplo Internet.

Básicamente existen tres arquitecturas de conexión VPN: de acceso remoto, punto a punto y overLAN.

La arquitectura VPN de acceso remoto consiste en el hecho de que varios usuarios se conectan a un mismo punto desde diferentes lugares a través de Internet, autentificándose en el mismo.

La arquitectura punto a punto permite a varias oficinas conectarse con una sede central, la cual acepta sus conexiones entrantes estableciendo así un túnel VPN.

Por último, la arquitectura «overLAN» es quizás la menos utilizada, pero muy potente dentro de una empresa, ya que permite conectarse entre distintos puntos de una misma empresa a través de su red local en lugar de hacer uso de Internet.

3.2. Ventajas y desventajas

3.2.1. Ventajas

- Ahorro de costes: nos permite adquirir un sólo servidor , aunque más potente, y a partir de ahí generar varias máquinas virtuales.
- Crecimiento flexible: la instalación de un nuevo servidor es mucho más sencillo y rápido que tratándose de un servidor físico.
- Administración simplificada: toda gestión de recursos se puede realizar desde el gestor de máquinas virtuales, centralizando todas estas operaciones.
- Aprovechamiento de las aplicaciones antiguas: se puede modernizar el software del usuario sin miedo a dejar de poder ejecutar aplicaciones antiguas, al poder correr estas sobre máquinas virtuales de sistemas operativos anteriores.
- Centralización de tareas de mantenimiento: se pueden realizar copias de seguridad de todas las máquinas virtuales desde un mismo punto.
- Disminución de tiempos de parada: las tareas comentadas anteriormente se pueden realizar en un breve espacio de tiempo.
- Mejor gestión de los recursos: se puede aumentar la memoria o almacenamiento de la máquina huésped para aumentar los recursos de todas las máquinas virtuales a la vez.
- Balanceo de recursos: se puede hacer uso una aplicación que haga un balanceo de los mismos, otorgando más memoria, recursos de la CPU,

almacenamiento o ancho de banda de la red a la máquina virtual que lo necesite.

3.2.2. Desventajas

- Rendimiento inferior: un sistema operativo virtualizado nunca alcanzará las mismas cotas de rendimiento que si estuviera instalado de forma nativa.
- No es posible utilizar hardware que no esté gestionado o soportado por el hipervisor.
- La avería del servidor anfitrión de virtualización afecta a todas las máquinas virtuales alojadas en él: a la hora de centralizar todas las tareas, se crea una mayor dependencia con el servidor anfitrión.

Ahora pasaremos a desarrollar las características de varios hipervisores conocidos, como Xen, KVM y VirtualBox.

3.3. Xen

El hipervisor Xen es una capa software que se ejecuta sobre el hardware del ordenador, permitiendo ejecutar de esta manera varios sistemas operativos sobre uno ya instalado concurrentemente.

Es soportado por varios tipos de procesadores, entre los que se encuentran x86, x86-64, Itanium, Power PC y ARM y soportado a su vez por varios sistemas operativos como Linux, NetBSD,FreeBSD, Solaris y Windows. Dicho hipervisor tiene además licencia GNU.

Una máquina ejecutando el hipervisor de XEN, tiene tres componentes:

- Hipervisor XEN
- «Domain» 0, una máquina virtual con privilegios de acceso directo al hardware. De esta manera se evita la «traducción» de las operaciones hardware haciéndola correr más rápidamente.
- Otros «Domains», los cuales no tienen los privilegios anteriormente comentados.

XEN nació como un proyecto de investigación en la Universidad de Cambridge liderado por Ian Pratt (fundador de XenSource). La primera versión pública de este software nació en el año 2003. En el año 2007, XenSource fue adquirida por Citrix Systems, una multinacional fundada en 1989 destinada a promover la virtualización de escritorio, «SaaS» y tecnologías «Cloud Computing».

3.4. Kvm

KVM («Kernel-based Virtual Machine») es un hipervisor de código abierto con el cual se puede hacer correr múltiples máquinas virtuales sobre un mismo sistema operativo. Cada máquina virtual tiene su propio hardware virtualizado como puede ser una tarjeta de red, un disco, adaptadores gráficos, etc.

Está basado en hardware x86 para Linux. Consiste en un módulo del kernel que proporciona el núcleo de la infraestructura virtual y un módulo específico para el procesador, Intel o AMD. Todos estos componentes están incluidos en Linux desde la versión 2.6.20.

3.5. VirtualBox

VirtualBox está diseñado para arquitecturas x86. Fue creado inicialmente por la empresa Innotek GmbH y más tarde desarrollado por Sun Microsystems, aunque actualmente forma parte de Oracle.

La primera versión de VirtualBox surgió el 15 de Enero de 2007. Fue una de las primeras soluciones de virtualización de código abierto. A finales de 2008 sacaron la versión 2.0 con cambios significativos, entre los cuales destacaron la creación de máquinas de 64 bits y una interfaz de Max OS X

nativa. Durante el primer semestre de 2009 no se realizaron cambios significativos, pero fue en el mes de Junio cuando salió la versión 3.0, en la cual se introdujeron cambios importantes en los que se refiere a «SMP» (Symmetrycal Multiprocessing). Bastante más tarde, en Mayo de 2010, salió la versión 3.2, con abundantes pequeñas mejoras.

Este proyecto está realizado para la versión 4.0 de VirtualBox, que fue publicada en el mes de Diciembre del 2010, con importantes cambios en lo que se refiere a tratamientos de los discos, capacidad de soportar más RAM en host de 32-bits, nuevo hardware virtual y otras pequeñas funcionalidades.

Una de las características que diferencian a VirtualBox de otros hipervisores es que ofrece un gestor de imágenes, es decir, discos que pueden ser usados en varias máquinas virtuales y que existen de manera independiente a éstas.

En este proyecto, hemos tenido que aprender a manejarnos con las funcionalidades que nos brinda VirtualBox para poder así generar las máquinas virtuales. Trabajando de esta manera sobre creación de máquinas virtuales, modificación de sus parámetros referentes a discos, redes, gráficos, y manejo de las imágenes de los sistemas operativos invitados, ya sea para ejecutarlas, pararlas, migrarlas, cancelarlas...

Capítulo 4

Arquitectura y Drivers de OpenNebula

Este capítulo está dedicado a repasar cómo está formado OpenNebula, qué estructura sigue y cuáles son las funciones de sus componentes.

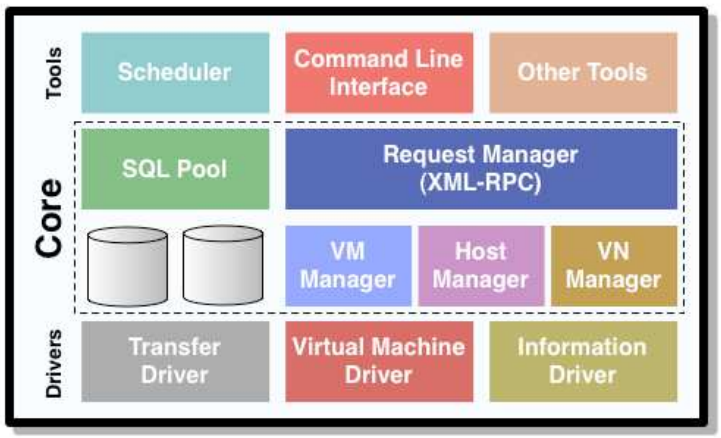
4.1. Arquitectura interna de OpenNebula

Podemos dividir OpenNebula en tres grandes segmentos:

«Tools»: Contiene las herramientas de gestión que ofrecen el núcleo.

«Core»: Componente principal de OpenNebula. Este módulo se encarga de gestionar las máquinas y redes virtuales y los nodos de nuestra infraestructura.

«Drivers»: Contiene las tecnologías necesarias para la virtualización, almacenamiento y monitorización de los servicios «cloud».



4.1.1. Capa «Tools»

Esta capa se divide a su vez en el CLI («Command Line Interface»), «scheduler» (planificador) y una serie de herramientas como la implementación de APIs populares.

CLI

Dirigido a los administradores de la nube. Ofrece manipular manualmente la infraestructura virtual, agregando nuevos nodos físicos, creando nuevas máquinas virtuales, etc.

El uso de CLI como sistema de acceso al hipervisor otorga grandes ventajas a los administradores de sistemas, ya que posibilita la realización de «scripts» para efectuar tareas de gestión, pudiendo configurarse cada uno de los comandos de manera detallada y según las exigencias de cada uno.

«**Scheduler**»

Una de las ventajas de OpenNebula antes mencionada es su gran flexibilidad a distintas condiciones. Esto se debe en gran parte a la implementación modular y el planificador hace uso de ello. Es una entidad completamente independiente a la arquitectura OpenNebula, pudiendo adpatarse o cambiarse sin afectar al sistema. Como opciones de configuración, podemos encontrar:

- Puerto de conexión a oned (por defecto, 2633).
- Tiempo de separación entre dos acciones de planificación (por defecto, 30).
- Número limite de máquinas gestionadas en cada acción de planificación (por defecto, 300).
- Número máximo de máquinas virtuales con las que comunicamos en cada acción de planificación (por defecto, 30).
- Número máximo de máquinas virtuales atendidas para un host indicado en cada acción de planificacion (por defecto, 1).

4.1.2. Capa «Core»

El núcleo lo constituyen todos los componentes encargados de monitorear maquinas y redes virtuales, el almacenamiento y los hosts. Podemos clasificar sus componentes en seis módulos funcionales:

«**Request Manager**»

Es el encargado de administrar las solicitudes de los clientes, ofreciendo una interfaz XML-RPC para llamar internamente a un componente. El XML-RPC disgrega la mayoría de las funcionalidades del núcleo de Opennebula a partir de componentes externos como el planificador.

«**Virtual Machine Manager**»

Encargado de la gestión y monitorización de las máquinas virtuales, dando una vista uniforme del conjunto de recursos. Los tres pilares que integran el administrador son la virtualización, la creación de redes y la gestión de imágenes.

«**Transfer Manager**»

Encargado de todas las transferencias de archivos necesarias para que las implementaciones de las máquinas virtuales y las migraciones entre distintos nodos en frío sean correctas.

«**Virtual Network Manager**»

Encargado de gestionar las direcciones IP y MAC para permitirnos crear redes virtuales entre los distintos nodos. La estructura de datos será un conjunto de redes, donde cada una podrá ser pública o privada y dentro de cada una, un grupo de direcciones IP y MAC.

«**Host Manager**»

Administra y supervisa los nodos físicos del sistema. Este gestor, al igual que el resto de infraestructuras posee gran flexibilidad y nos permite incluir cualquier atributo del host.

«**Database**»

Base de datos persistente que almacena los distintos datos de OpenNebula. Este componente otorga a OpenNebula la escalabilidad y fiabilidad que requiere un sistema de gestión de máquinas virtuales. Soporta tanto MySQL como SQLite3.

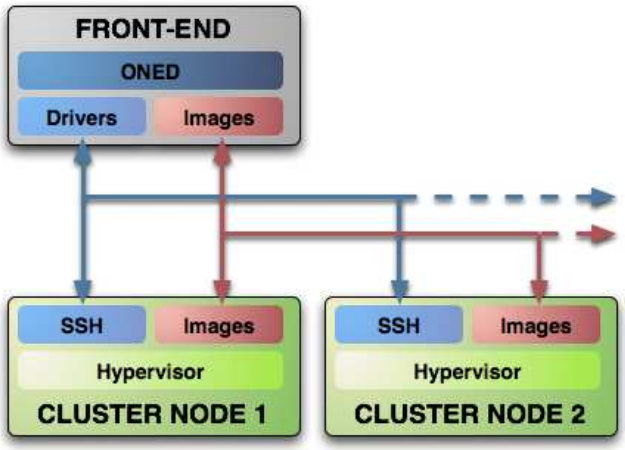
4.1.3. Capa «Drivers»

OpenNebula ofrece distintos módulos que permiten interactuar con middleware específico, como hipervisores de virtualización (por ejemplo VirtualBox) o servicios en la nube (como Amazon EC2), entre otros.

Cabe destacar que esta capa se encuentra en continua expansión gracias al ecosistema de OpenNebula, con una muy activa comunidad de usuarios que ofrece un amplio catálogo de extensiones, herramientas y «plugins» para el hipervisor, todo ello en código abierto.

4.2. Infraestructura de OpenNebula

OpenNebula está formado por un «front-end» que ejecutará tanto el sistema como los servicios del «cluster», y un conjunto de nodos en los que ejecutaremos las distintas máquinas virtuales. Estas dos partes se interconectarán por, al menos, una red que configuraremos previamente. Dentro del «fron-end» podemos distinguir tres componentes:



Repositorio de imágenes: Directorio que contendrá las imágenes virtuales base.

Demonio OpenNebula (ONED): Responsable de manejar todas las solicitudes entrantes, tanto del CLI como de la API, y comunicarse con otros procesos cuando sea necesario.

Drivers: Usados por el core como interfaz para un hipervisor o sistema de almacenamiento. Según su funcionalidad, distinguimos:

- «VMM driver» («Virtual Machine Manager»): Gestión de los hipervisores.
- «IM drivers» («Image Manager»): Monitorización.

- «TM drivers» («Transfer Manager»): Transferencia de máquinas virtuales.

Capítulo 5

Desarrollo

En este capítulo repasamos el diseño del driver, la organización seguida y la forma en la que se desarrollan las operaciones para monitorizar y controlar las máquinas virtuales, además del «plugin» para Sunstone, la interfaz web de OpenNebula.

5.1. El «driver» de VirtualBox

5.1.1. Funcionamiento general

El «driver» de VirtualBox para OpenNebula consiste en una serie de «scripts» escritos en lenguajes Ruby y, en menor medida, Shell. Como el resto de drivers de OpenNebula, se divide en dos partes. La primera se ocupa de recabar información sobre las máquinas en las que se van a ejecutar las máquinas virtuales. Se trata del «im_mad», o administrador de información. La segunda parte se ocupa de realizar las operaciones necesarias con las máquinas virtuales, y se denomina el «vmm_mad».

Estas dos partes forman los «scripts remotos». Se denominan así porque son copiados a cada máquina gestionada a través de OpenNebula, en el momento en el que son añadidas. Mediante su ejecución remota permiten a OpenNebula extraer información de las máquinas y realizar las operaciones necesarias sobre ellas en cuanto a la gestión de máquinas virtuales.

5.1.2. Monitorización de máquinas

El «im_mad» es un conjunto de «scripts» bastante simples para obtener información de las máquinas. Para ello utilizamos herramientas del sistema ('uname', 'top') y parte de la información que ofrece el hipervisor de VirtualBox sobre el sistema en el que se ejecuta.

La información se devuelve en forma de pares CLAVE=VALOR por la salida estándar. Estos pares son entonces procesados por OpenNebula y la información que contienen incluida en la descripción detallada de las máquinas que controla. Algunos valores, como la memoria, son de especial importancia, ya que son tenidos en cuenta a la hora de decidir en qué máquina realizar el despliegue de una nueva máquina virtual.

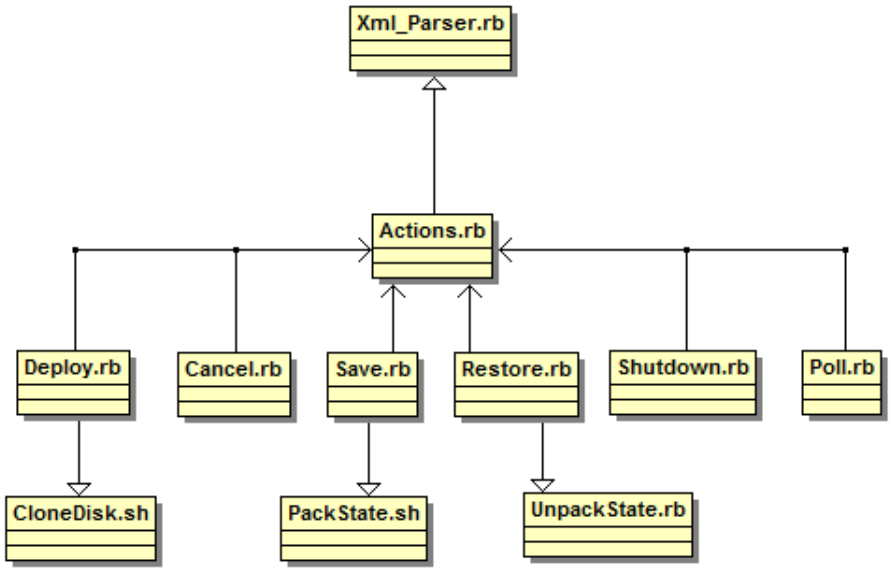
5.1.3. Administración de máquinas virtuales

La parte correspondiente a la gestión de máquinas virtuales («vmm_mad») es la que ocupa el grueso del código realizado. VirtualBox tiene algunas características que lo diferencian de otros hipervisores populares como KVM o XEN, por lo que la estrategia a seguir ha sido completamente diferente. Allí donde la ejecución de una operación sobre la máquina virtual, como crear una nueva, significa un comando simple (ya sea a través de libvirt o de algún comando del hipervisor), en VirtualBox supone una serie de pasos bien diferenciados. Todos ellos se realizan a través de la interfaz de línea de comandos ofrecida por el hipervisor.

Este proceder está condicionado por la característica de VirtualBox de tener definidos un conjunto de discos que pueden ser utilizados por cualquier máquina virtual, mientras que en KVM o XEN los discos van asociados desde un principio a una máquina.

Además, OpenNebula ofrece soporte nativo para KVM, XEN y VMware, lo que significa que se generan ficheros de configuración comprensibles para estos hipervisores. No es el caso para VirtualBox. OpenNebula va a generar para nosotros un fichero XML con la descripción de la máquina virtual y será nuestro driver el que lo traduzca e interprete para realizar las operaciones necesarias.

En el siguiente gráfico podemos observar un diagrama que muestra cómo se estructura el «vmm_mad»:



Veamos en detalle cómo se realizan estos procesos.

Procesando la configuración

La configuración de una máquina virtual creada a través de una plantilla de OpenNebula llega a nuestro «driver» en formato XML en el momento de la creación. Contamos con un «script» (`xml-parser.rb`) que es capaz de parsear este XML y de ahí sacar los argumentos necesarios para los comandos que realizan operaciones en el hipervisor.

Es importante señalar que para nosotros es necesario guardar este fichero de configuración y conocer su localización no sólo en el momento de la creación de la máquina virtual (como en otros drivers), sino también cuando se realizan operaciones como la salvaguarda o el apagado. Esto se debe a que a veces necesitamos conocer toda la información sobre los medios de almacenamiento (discos...), ya que para borrar o migrar una máquina virtual hay que desenchufarlos y desregistrarlos primero.

Esto nos obliga a almacenar en un fichero auxiliar una relación entre los nombres de máquinas virtuales y la localización de sus ficheros de configuración. Esta localización puede ser cambiada en la configuración de Open-

Nebula y sólo se conoce en la creación, por tanto no podemos fiarnos de que vaya a estar siempre en el lugar por defecto.

Ejecutando los comandos

Los comandos sobre el hipervisor se ejecutan con los argumentos extraídos de la configuración en XML de las máquinas. Esta ejecución está controlada por otro script (`actions.rb`). Varios comandos encadenados de este tipo componen las operaciones que puede realizar OpenNebula. Algunos, como la migración fría de máquinas virtuales, requieren operaciones más complicadas que veremos en detalle.

La ejecución de estos comandos es observada por si se produce algún error. En estos casos se vuelcan los mensajes sobre la salida de error, de manera que OpenNebula pueda informar al usuario de cuál ha sido el problema. También se dispone de un modo «debug» en el que se vuelcan todas la salidas a un fichero para su posterior revisión.

Operaciones de OpenNebula

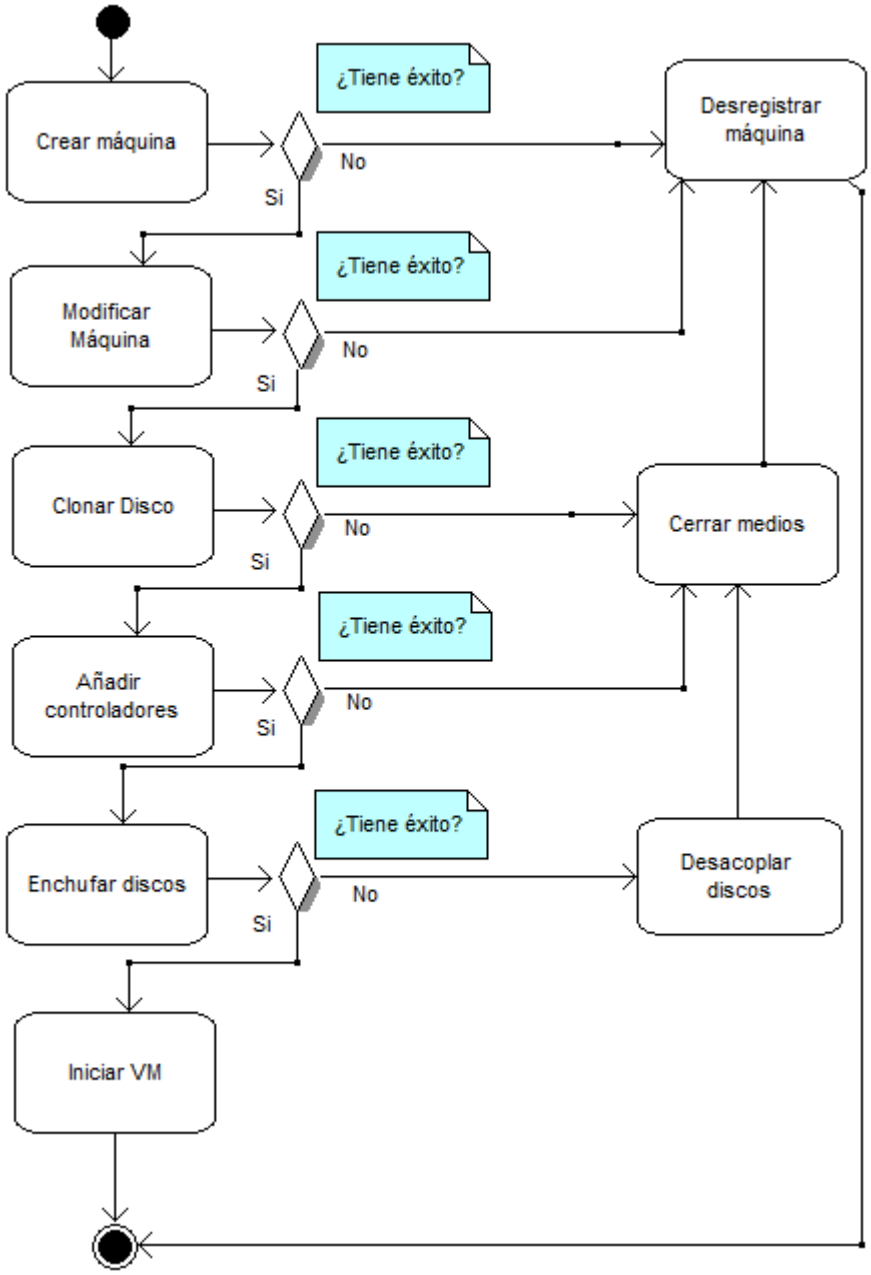
Pasamos a detallar cómo se realizan las operaciones de OpenNebula sobre VirtualBox.

- Despliegue (deploy)

  El despliege de una máquina virtual (VM) consiste en su creación y puesta en ejecución a través del hipervisor correspondiente. En VirtualBox es un proceso que consiste en varios pasos:

  1. Registrar la VM: Se le asigna un nombre a la máquina virtual y se crea en el hipervisor.
  2. Modificar la VM: De acuerdo a la plantilla que define las características de la VM (memoria, interfaces de red...), se ejecuta el comando de VirtualBox con los argumentos correspondientes para configurar la máquina creada en el paso 1.
  3. Clonación de discos: VirtualBox trata los discos de manera independiente a las VM que tiene registradas. Cada disco, tiene un UUID único. Esto significa que no es posible añadir el mismo disco dos veces para ser, por ejemplo, utilizado por dos VM diferentes. Por ello, antes de utilizar un disco es necesario clonarlo de manera que obtenga un nuevo UUID único. Este paso hace uso de un script bash auxiliar para realizar la operación de clonado y que sustituye el disco original por su clon.

4. Añadir controladores: El siguiente paso es añadir los controladores de discos necesarios. Por ejemplo, si tenemos discos IDE, hay que añadir un controlador IDE. Lo mismo para SATA y otros tipos de buses disponibles.
5. Enchufar discos: Una vez completados los pasos anteriores, los discos (también llamados imágenes), pueden ser enchufados a nuestra VM. Al añadirlos se estipula el bus al que se conectan y el orden.
6. Lanzar la máquina: La VM está configurada, con su almacenamiento conectado y lista para funcionar. En este paso se decide si se desea mostrar la salida gráfica a través de una ventana SDL, o activar la salida VRDP para poder conectarse remotamente a ella (a modo de VNC).



- Cancelación (cancel)

  La cancelación de una máquina virtual (VM) consiste en su desenchufado y borrado completo del sistema. Sólo se utiliza cuando por alguna razón no es posible apagar una máquina de manera ordenada (a través de una señal ACPI). En VirtualBox el proceso es prácticamente inverso al de creación y consiste en:

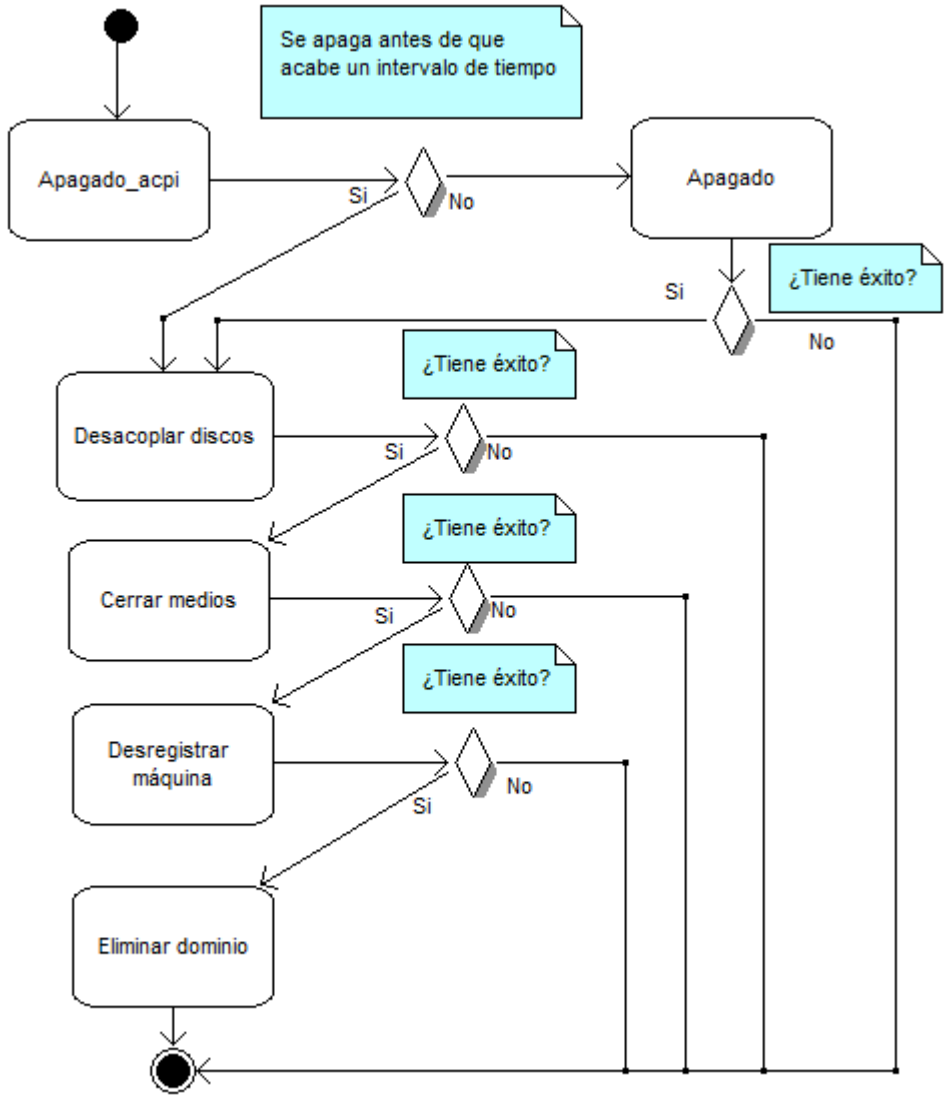
1. Desenchufar la VM: La máquina es apagada de manera abrupta.
2. Desconectar los discos: Los medios de almacenamiento conectados a la VM se desacoplan.
3. Cerrar los discos: Una vez desacoplados, se cierran los medios de almacenamiento asociados a la VM (y se borran físicamente).
4. Borrar la VM: La máquina virtual se borra del hipervisor sin que quede rastro de ella.

- Apagado (shutdown)

  El apagado es la manera ordenada de cerrar y borrar una máquina virtual (VM). La diferencia con respecto a la cancelación de una VM es que el apagado se realiza mediante el envío de una señal ACPI. Típicamente, la VM estará preparada para recibir este tipo de señales y comenzará un proceso de apagado en el que el sistema operativo termine los procesos y desmonte los discos de manera controlada.

  Sin embargo, hay casos en los que el apagado ACPI no funciona. El «script» de apagado espera durante un intervalo de tiempo y comprueba de manera periódica si la máquina ha alcanzado el estado «power off» de manera satisfactoria. Si se alcanza un límite de tiempo (configurable al principio del «script»), se procede a desenchufar la máquina de manera abrupta.

  El resto del proceso es el mismo que en la cancelación de una VM, por lo que no lo detallamos.



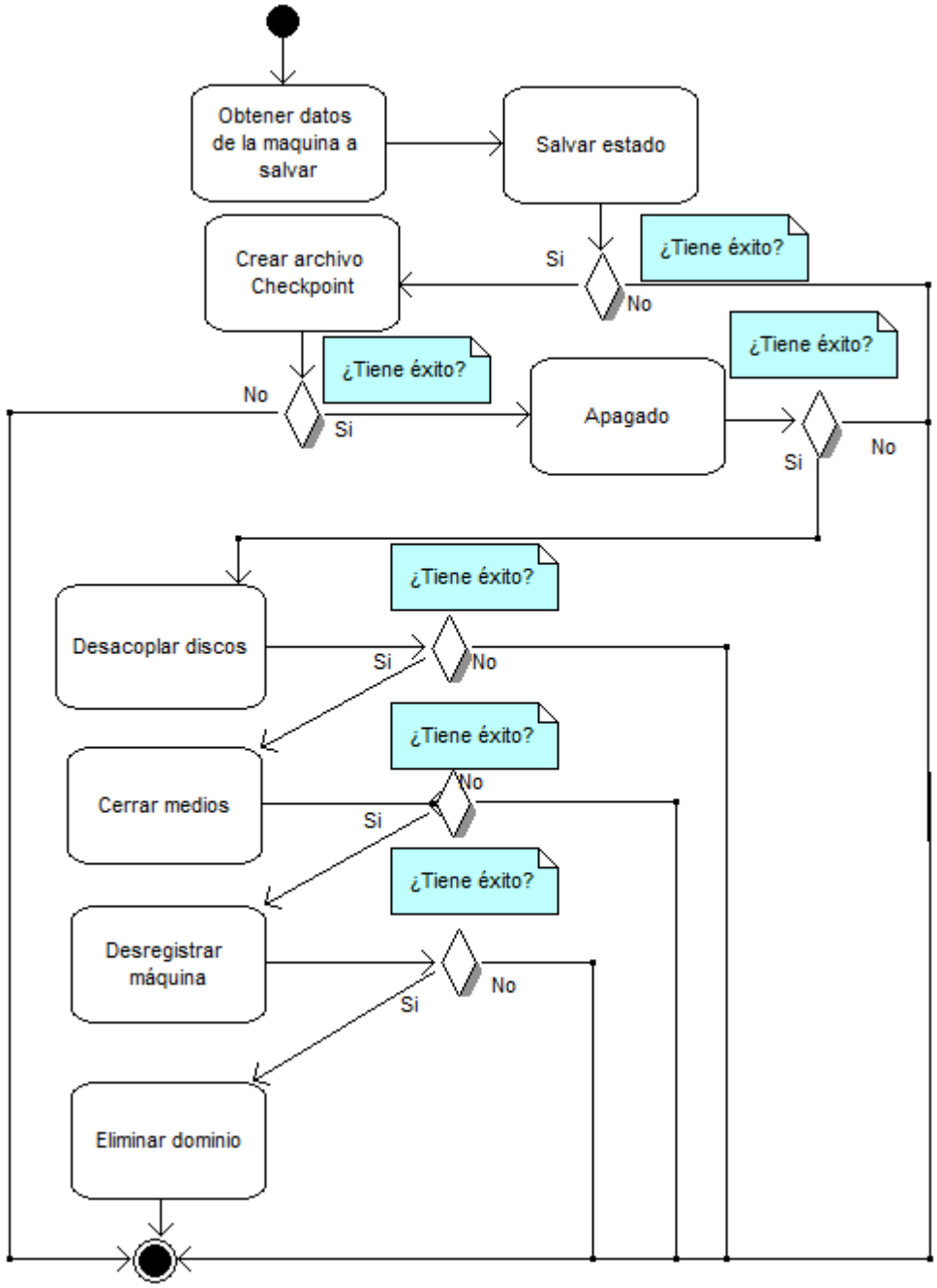
- Salvaguarda («save»)

  La salvaguarda de una máquina virtual (VM) significa su congelación. El estado en el momento de la orden es conservado en un fichero llamado «checkpoint» y debe poder permitir la restauración de esa máquina en el momento exacto y en las condiciones correspondientes al momento que fue guardada.

La salvaguarda, junto con la restauración, son dos procesos muy importantes ya que permiten la migración en frío de una VM: la máquina se guarda en un lugar, se copia a otro, y se restaura. Esto sirve para balancear la carga entre los diferentes nodos físicos existentes.

El proceso de salvaguarda en VirtualBox no es inmediato como en otros hipervisores, ni está soportado de manera estándar. Por ello, hemos seguido una serie de pasos de manera que sea posible ejecutarlo:

1. Salvar estado: VirtualBox permite cerrar una máquina virtual salvando su estado, y ese es el primer paso.
2. Empaquetado del estado: VirtualBox crea un fichero .sav que contiene la información del estado de la máquina, para restaurarla en cualquier momento. Sin embargo, no está diseñado pensando que esa máquina puede ser restaurada en otro lugar diferente. Por ello, nuestro «driver» copia el fichero de configuración original de la VM y el fichero de estado y los empaqueta en un fichero TAR. Este paso se hace mediante un «script» bash auxiliar (packState.sh).
3. Cancelación: Una vez empaquetado todo lo necesario para la restauración de la VM, se desenchufa completamente y se cancela (desconectar discos, etc.), de manera que lo único que queda atrás es un fichero «checkpoint» cuya localización se devuelve a OpenNebula.



- Restauración (restore)

La restauración de una máquina virtual (VM) es el proceso inverso a la salvaguarda: a partir de un fichero «checkpoint» se relanza la VM en el punto en el que se había guardado. Aquí cobra sentido el haber

empaquetado el fichero de configuración original en el fichero, ya que VirtualBox no salva la configuración de la máquina ni la información de los discos asociados.

El proceso seguido para restaurar una VM consiste en desempaquetar el fichero de «checkpoint» y realizar el despliegue de la máquina como si fuese nueva a partir del fichero de configuración. La diferencia con el despligue original es que el lanzamiento de la máquina no se realiza. En su lugar, se copia el fichero .sav creado por VirtualBox a su lugar original (dentro de las carpetas de configuración del hipervisor) y entonces se adopta el estado. La VM continúa tal y donde estaba antes de guardarla, pero sin enterarse de que ha sido completamente borrada y recreada, que tal vez ha cambiado de nodo físico y de que sus nuevos discos son sólo copias de los originales.

- Consulta (poll)

Esta es la única orden que se procesa de forma sencilla y en un único paso, aprovechando que VirtualBox, como otros hipervisores, ofrece información detallada sobre una máquina virtual registrada.

5.1.4. Ficheros

Ofrecemos a continuación una relación breve de los directorios y ficheros que componen el driver, con una pequeña descripción de las funciones de cada uno. Los directorios principales son:

- etc: Contiene ficheros necesarios para establecer los valores de configuración por defecto de las máquinas virtuales soportadas por VirtualBox.
- remotes: Contiene las mayor parte de las fuentes específicas del driver: los ficheros que se copian en los nodos para monitorizar y controlar las máquinas virtuales que se despliegan en ellos:
  - Carpeta im («Information Manager»): Contiene algunos scripts para obtener información básica de un host: memoria total, memoria disponible, velocidad y tipo de CPU...
  - Carpeta vmm («Virtual Machine Manager»): Contiene los scripts para controlar las máquinas virtuales.
    - `xml-parser.rb` «Script» de Ruby encargado de generar los argumentos para la línea de comandos de VirtualBox a partir de la descripción XML que genera OpenNebula para configurar una máquina virtual.

- `actions.rb` «Script» de Ruby encargado de realizar "las acciones", es decir, las llamadas a la línea de comando de VirtualBox. Obtiene los argumentos necesarios para esas llamadas del "xml-parser.rb". que será almacenado en el directorio /tmp/vbox.log.
- `deploy, cancel, shutdown, poll, restore, save` «Scripts» de Ruby encargados de la ejecución de las operaciones de OpenNebula sobre máquinas virtuales. Hacen uso de una o varias acciones del "actions.rb".
- `cloneDisk.sh` «Script» de Shell encargado de clonar los discos de manera que se les asigne un UUID único, y así poder utilizar varias veces el mismo disco en la misma máquina con VirtualBox.
- `packState.sh/unpackState.sh` «Scripts» de Shell empaquetan y desempaquetan los checkpoints que permiten la salvaguarda, migración y restauración de una máquina virtual.
- `vboxrc.rb` Fichero que incluye constantes configurables utilizadas en diferentes «scripts».

- `share/examples/vbox`: Contiene ficheros de ejemplo: una imágen de ttylinux, un ejemplo de plantilla, un esquema de plantilla, y un oned.conf con los valores configurados para usar el «driver» de VirtualBox.

- `sunstone/vbox-plugin.js`: Plugin para Sunstone de OpenNebula. Permite utilizar la interfaz gráfica Sunstone con el plugin de VirtualBox.

- `install.sh`: «Script» de instalación. Copia los ficheros del driver en la instalación de OpenNebula para poder usarlos.

5.2. El «plugin» para Sunstone

OpenNebula lanzó su «Sunstone Cloud Operations Center» con su versión 2.2 a finales de Marzo de 2011. Sunstone es una interfaz web que permite el manejo de OpenNebula de manera gráfica y sencilla.

A pesar de no estar contemplado en las caraterísticas iniciales del proyecto, y al observar que la versión planeada de Sunstone 3.0 tiene una arquitectura orientada a plugins, decidimos aprovecharla y desarrollar un plugin que permita la utilización de nuestro driver desde la interfaz gráfica.

El plugin de VirtualBox para OpenNebula Sunstone realiza modificaciones relevantes en los plugins por defecto, de manera que:

- Sea posible añadir un «host» desde Sunstone eligiendo VirtualBox como IM y VMM, de la misma manera que se pueden elegir los drivers de KVM o XEN.
- Sea posible crear una máquina virtual con las características soportadas por VirtualBox. Esto se ha conseguido añadiendo un formulario de creación de máquinas virtuales específico para VirtualBox. El formulario facilita mucho la creación de máquinas virtuales, comprobando que los parámetros obligatorios están incluidos, pudiendo añadir redes e imágenes predefinidas, etc.

Este plugin será pionero pues, al estar la versión 3.0 de Sunstone todavía en desarrollo, será el primer plugin externo a la distribución oficial y servirá para mostrar que las características de Sunstone son fácilmente extendibles.

Capítulo 6

Configuración y uso

Describimos brevemente la configuración y uso del «driver». Este apartado se amplía de manera práctica en los anexos, dónde se incluye una guía de configuración orientada a usuarios y un ejemplo de uso práctico.

6.1. Configuración de OpenNebula

El primer paso es la instalación del «driver». Esta se realiza fácilmente gracias al «script» "install.sh". El «script» copia los ficheros dentro de la instalación de OpenNebula, de manera que sean accesibles y usables.

A continuación, es necesario especificar que se desea utilizar el «driver» de VirtualBox en el archivo de configuración `/etc/oned.conf` de OpenNebula. Se distribuye junto al «driver» un archivo de configuración listo para utilizar. Los cambios consisten en configurar las variables de `IM_MAD` y `VMM_MAD` de manera que se utilicen los «scripts» proporcionados por nosotros. Además, hay que especificar que el tipo de formato deseado para el `VM_MAD` (para la configuración de las máquinas virtuales) es XML.

Nuestro driver no requiere configuración específica adicional. Su funcionamiento dependerá ahora de que la instalación general de OpenNebula sea satisfactoria y esté bien configurada, según se indica en la documentación del producto.

6.1.1. Configuración de Sunstone

No es necesaria configuración adicional del «plugin» de Sunstone. El «script» de instalación lo coloca en la carpeta adecuada y Sunstone lo carga de manera automática (al menos está previsto que lo haga en la 3.0).

6.2. Uso de OpenNebula

- A través de la interfaz de línea de comandos: El uso de OpenNebula no difiere con nuestro driver. Las operaciones se realizan de la misma manera que con cualquier otro, con la excepción de que al añadir un «host» hay que especificar "im_vbox" y "vmm_vbox". Cabe tener en

cuenta que VirtualBox no soporta «live migration», por lo que si se intenta realizar esta operación, se producirá un error.

- A través del centro de operaciones Sunstone: El «plugin» de VirtualBox para Sunstone adapta las facilidades de Sunstone a VirtualBox, sin que haya que mencionar ninguna particularidad en cuanto a usabilidad. En la creación de «hosts», VirtualBox está seleccionado automáticamente. Y en la creación de máquinas virtuales, el «wizard» de VirtualBox es el único disponible, por lo que se minimiza cualquier riesgo de confusión.

Se incluye en los anexos un ejemplo de uso que amplia la información aquí contenida de manera práctica.

Capítulo 7

Conclusiones

El «driver» de VirtualBox para OpenNebula supone una nueva funcionalidad para un software de administración de «clouds» IaaS que puede presumir de una gran versatilidad para adaptarse a múltiples escenarios. Al estar escrito bajo una licencia de código abierto (Apache), el código queda disponible para que la comunidad siga trabajando sobre él.

Precisamente, el hecho de que nuestro trabajo pueda ser [re]utilizado en el mundo real ha supuesto una gran motivación a la hora de llevarlo adelante. Hemos ampliado por ello los horizontes del plan inicial incluyendo un «plugin» para la interfaz web de OpenNebula (Sunstone) que facilite el uso y complete su capacidad de integración. Se han redactado además, instrucciones detalladas de instalación y uso (ver anexos) para alcanzar este objetivo.

Otro de los aspectos a los que se ha prestado especial atención es a la compatibilidad de nuestro driver con la versiones existentes del software con el que trabaja. En un año hemos visto pasar a OpenNebula de la versión 1.4, a la 2.0, la 2.2 y la 3.0 (Julio 2011). Intimamente ligado a esta evolución está OpenNebula Sunstone, que ni siquiera existía. De igual manera, VirtualBox ha dado el salto de la versión 3 a la 4, y el uso de Ruby 1.9.2 (Agosto 2010) se ha generalizado frente a la versión 1.8.7. Esta rápida evolución de las herramientas con las que trabajamos nos ha obligado a adaptarnos constantemente. Por ello podemos afirmar que el «driver» funciona con las últimas versiones de estos programas.

El trabajo realizado abre a su vez muchos caminos al desarrollo y mejora del propio «driver». Sería muy interesante explorar, por ejemplo, si OpenNebula sería capaz de proporcionar las condiciones requeridas para una migración «en caliente» de máquinas virtualizadas con VirtualBox (va más allá de las capacidades de un «plugin» como tal). También sería una posibilidad dar soporte a muchas más opciones de VirtualBox, ahora que OpenNebula 3.0 permite la declaración de atributos personalizados en las plantillas de máquinas virtuales. Igualmente, se podría buscar una solución para integrar un cliente RDP en OpenNebula Sunstone, de la misma manera que se ha integrado uno VNC, para poder visualizar las máquinas lanzadas.

Esperamos que este proyecto, que nos ha permitido conocer y sumergirnos en el área del «Cloud Computing», sea de utilidad y ayude a OpenNebula a seguir creciendo, progresando y dando a la Facultad de Informática de la UCM el renombre internacional que se merece.

Capítulo 8

Glosario

Cloud Computing: modelo que ofrece servicios de computación a través de Internet.

Driver: Controlador de un dispositivo, que permite hacer una abstracción del hardware proporcionando una interfaz para hacer uso del recurso.

IaaS: Infraestructure As A Service. Servicio de insfraestructura cloud para manejo de software, servidores, redes y tareas relacionadas con los mismos.

Infraestructura: Conjunto de elementos o servicios que se consideran necesarios para la creación y funcionamiento de una organización.

OpenNebula: Herramienta opensource para manejar infraestructuras en la nube. Capaz de construir nubes públicas, privadas e híbridas.

Servicio: Conjunto de herramientas destinadas a satisfacer las necesidades del público o de una entidad pública o privada.

VirtualBox: Software de virtualización. Con él es posible instalar sistemas operativos calificados de invitados dentro sistemas operativos conocidos como anfitriones, cada uno con su ambiente virtual.

Virtualización: Abstracción de lo recursos de una máquina. Crea una capa entre el software y el hardware de la máquina para poder adaptar varios recursos al mismo hardware.

Checkpoint: Checkpoint es un punto de control, en nuestro contexto es un punto en que guardamos el estado de la máquina para luego posteriormente poder ser restaurada a partir de ahí.

KVM: Software para implementar virtualización completa con Linux sobre hardware x86. Permite ejecutar máquinas virtuales utilizando imágenes de disco que contienen sistemas operativos sin modificar.

XEN: Monitor de máquina virtual de código abierto que permite controlar los recursos, garantizar calidad de servicio y migrar máquinas virtuales en caliente.

Domain: Nombre de identificación que permite distinguir entre distintos host o, en nuestro caso, máquinas virtuales.

SaaS: Modelo de servicio de cloud que permite al cliente hacer uso de software a través de Internet, sin necesidad de preocuparse del soporte, de la instalación ni del mantenimiento.

PaaS: Modelo de servicio de cloud que proporciona a los clientes la capacidad de trabajar en todos los aspectos de desarrollo de un software, todo ello a través de Internet.

PaaS: Modelo de servicio de cloud que proporciona a los clientes la capacidad de trabajar en todos los aspectos de desarrollo de un software, todo ello a través de Internet.

Hipervisor: Monitor de máquina virtual (VMM) que permite aplicar técnicas de control para virtualizar varios sistemas operativos al mismo tiempo.

SunStone: GUI que pretende facilitar tantos a usuarios como administradores de OpenNebula, la posibilidad de realizar operaciones sobre infraestructuras cloud privadas e híbridas sin necesidad de hacer uso de CLI.

Máquina Virtual: Software que emula a una máquina sobre los cuales se pueden realizar las mismas operaciones que si fuese sobre una máquina real. Los procesos que ejecutan están limitados por los recursos proporcionados por ellas.

Host: Máquina conectada a una red, que proveen y utilizan servicios de la misma, como la transferencia de ficheros, conexión remota, servidores...

Cluster: Conjunto de máquinas cuyas componentes hardware son comunes a todas ellas y que en su globalidad actúan como si fuesen una única computadora.

Ruby: Lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por Yukihiro Matz. Su implementación oficial es distribuida bajo licencia de software libre.

Bibliografía

[1] Guijarro Olivares, Jordi. *Infraestructuras orientadas al servicio en la Nube*. Boletín de RedIRIS, nº 88-89, abril de 2010. Centre de Supercomputación de Catalunya. Universitat Oberta de Catalunya.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Febrero 2009. University of California, Berkeley.

[3] Antonopoulos, Nick. *Cloud Computing: Principles, Systems and Applications (Computer Communications and Networks)*. Ed. Springer, 2010

[4] http://www.opennebula.org

[5] http://slideshare.net/rsmontero/building-clouds-one14

[6] CESGA. *Instalación y configuracion de OpenNebula*. Mayo 2011. Centro de supercomputación CESGA.

[7] http://open.eucalyptus.com/learn/what-is-eucalyptus

[8] http://openstack.org

[9] http://www.xen.org

[10] http://www.linux-kvm.org/page/Main_Page

[11] http://www.cloudsigma.com/es/computacion-en-nube/que-es-la-nube

[12] http://www.cloudcomputingla.com/2010/08/saas-paas-e-iaas.html

[13] http://geeksroom.com/2010/04/16293/16293

[14] http://www.tecnologiapyme.com/software/que-es-la-virtualizacion

Apéndice A

Instalación y configuración

Este anexo incluye una guía que explica de manera detallada cómo se configura e instala el «driver» de VirtualBox. Se trata de la traducción de la guía escrita como elemento de documentación que acompaña al «driver» de cara a la comunidad de OpenNebula.

A.1. Características

Los «drivers» de VirtualBox para OpenNebula (one-vbox) soportan la creación de «hosts» y lanzamiento de Máquinas Virtuales usando su hipervisor.

La monitorización y tratamiento de las Máquinas Virtuales se realiza a través de la interfaz de la línea de comandos de VirtualBox e implementa gran parte de las operaciones relacionadas con VM presentes en OpenNebula.

Dicho «driver» soporta migración en frío. Esto quiere decir que la VM es guardada, y seguidamente copiada en otro «host» y finalmente restaurada a partir de esa imagen guardada. No soporta migración en caliente hoy en día.

A.2. Requisitos

- Los nodos «cluster» deberán tener instalado VirtualBox (actualmente es soportada la versión 4.0). VirtualBox debe estar disponible para el uso del usuario de OpenNebula (usualmente oneadmin), lo que significa que este usuario debe pertenece al grupo de VirtualBox y los permisos de los ejecutables de VirtualBox deben de ser configurados correctamente.

- Una versión de «Ruby» (1.8.7 o 1.9.2) debe de estar instalada en el nodo remoto.

- Se necesita «GNU tar» en el nodo remoto para salvar, restaurar y migrar máquinas virtuales.

A.3. Archivos del «driver»

El paquete del «driver» one-vbox contiene los siguientes ficheros. Ten en cuenta de que son referenciados usando la variable de entorno $ONE_LOCATION como base del directorio, lo que quiere decir que OpenNebula estaría instalado en modo «self-contained».

- `$ONE_LOCATION/etc/vmm_ssh/vmm_ssh_vbox.conf`

  Fichero de configuración que define los valores por defectos de las definiciones de los «domains» de VirtualBox.

- `$ONE_LOCATION/lib/remotes/vmm/vbox` y `$ONE_LOCATION/lib/remotes/var/vbox`

  Scripts utilizados para llevar a cabo las operaciones de las máquinas virtuales. Los ficheros de `remotes/lib` son una copia de los ficheros en `remotes/var`. Estos ficheros en var, son copiados al «host» remoto.

- `$ONE_LOCATION/lib/remotes/im/vbox.d`

  «Scripts» utilizados para extraer información de los «host» remotos. Es ahí donde se copian estos «scripts».

- `$ONE_LOCATION/share/examples/vbox`

  Algunos ficheros útiles de VirtualBox: una imagen ttylinux de VirtualBox, una plantilla estándar, un ejemplo de oned.conf con el «driver» de one-vbox activado, y una plantilla esquema indicando los atributos soportados para las plantillas de VirtualBox.

A.4. Instalación

Para instalar el «plugin» de VirtualBox, descargárselo, descomprimirlo y ejecutar el «script» install.sh como onedamin (o como propietario de la instalación de OpenNebula). Ten en cuenta que es necesario tener la variable de entorno `ONE_LOCATION` definida.

El «script» de instalación copiará los ficheros del driver y los «examples» al directorio de instalación de OpenNebula.

Si estás usando una instalación en modo «system-wide» de OpenNebula, tendrás que copiar manualmente los ficheros en el directorio de instalación del mismo.

A.5. Configuración

Para habilitar el «driver» de one-vbox, es necesario modificar adecuadamente el fichero oned.conf. Esto se consigue estableciendo las opciones de `IM_MAD` y `VM_MAD` para usar el «driver» one-vbox, como sigue:

```
IM_MAD = [
        name       = "im_vbox",
        executable = "one_im_ssh",
        arguments  = "vbox" ]

VM_MAD = [
        name       = "vmm_vbox",
        executable = "one_vmm_ssh",
        arguments  = "vbox",
        default    = "vmm_ssh/vmm_ssh_vbox.conf",
        type       = "xml" ]
```

El nombre del «driver» necesita ser proporcionado a la hora de añadir un nuevo «host» a OpenNebula, por ejemplo:

```
onehost create hostname im_vbox vmm_vbox tm_ssh
```

«Executable» apunta al «path» del fichero ejecutable del «driver».

«Default» apunta al fichero de configuración del «driver».

«Type» identifica el hecho de que este «driver» usa el formato XML en el fichero del dominio pasado desde OpenNebula al «driver» de VirtualBox.

A.6. Plantillas

Un esquema de una plantilla one-vbox para una máquina virtual:

```
# ------------------------------------
# ATRIBUTOS DE LA VM PARA ONE-VBOX
# ------------------------------------

#-------------------------------------
# Nombre de la VM
#-------------------------------------

NAME = "vm-ejemplo" # Opcional
# por defecto: one-$VMID

#-------------------------------------
#               Capacidad
#-------------------------------------

CPU    = "Cantidad_de_CPU_requerida" # No soportado
MEMORY = "Cantidad_de_MEM_requerida" # Opcional
VCPU   = "Numero de CPUs virtuales"  # Opcional

#-------------------------------------
#         SO y opciones de inicializacion
#-------------------------------------

#Nada soportado
OS = [
  kernel     = "Ruta_al_SO_kernel",            #
  initrd     = "Ruta_a_imagen_initrd",         #
  kernel_cmd = "Linea_de_comandos_kernel",      #
  root       = "Disp. que se montara como root" #
  boot       = "Disp. desde el que iniciamos" ] #

#-------------------------------------
#      Caracteristicas del hipervisor
#-------------------------------------

FEATURES = [
  pae  = "yes|no",   # Opcional
  acpi = "yes|no" ]  # Opcional
```

```
#---------------------------------------
#                Discos VM
#---------------------------------------

DISK = [
  type     = "disk|dvd|floppy",#Obligatorio
  source   = "Ruta a la imagen de disco", #Oblig
  size     = "tamano_en_GB", #No soportado
  target   = "device_to_map_disk", #Obligatorio
  bus      = "ide|scsi|sata|floppy|sas", #Oblig
  readonly = "yes|no", #Opcional
  clone    = "yes|no", #No soportado
  save     = "Ruta al archivo de imagen de disco" ]
   #No soportado

#---------------------------------------
#           Interfaces de red
#---------------------------------------
#Several ones can be specified

NIC = [
  network = "Nomkbre_de_la_red_virtual",
  #No soportado
  ip      = "Direccion_IP", #No soportado
  bridge  = "name_of_bridge_to_bind_if", #Oblig
  target  = "Nombre disp. para mapear",
  #No soportado
  mac     = "Direccion HW", #Opcional
  script  = "path_to_script_to_bring_up_if"]
   #No soportado

#---------------------------------------
#  Interfaces E/S
#---------------------------------------

INPUT = [ #No soportado
  type = "mouse|tablet",
  bus  = "usb|ps2|xen" ]

GRAPHICS = [ #Opcional
  type   = "vrdp", #Obligatorio
  listen = "IP-a-escuchar", #Obligatorio
  port   = "puerto", #Opcional
```

```
  passwd = "password" ] #No soportado

#---------------------------------------
#  Atributos del hipervisor RAW
#---------------------------------------

RAW = [ #Opcional
  type = "vbox",
  data = "Configuracion_dominio_RAW"]

#---------------------------------------
#  Contexto de la maquina virtual
#    Los valores pueden usar:
#    $<Variable_plantilla>
#    $<Variable_plantilla>[<atributo>]
#    $<Variable_plantilla>[<atributo>,
     <atributo2>=<valor2>]
#    $NETWORK[<atributo_vnet>,
 NAME=<nombre_vnet>]
#---------------------------------------

CONTEXT = [ #Opcional
  var_1 = "valor_1",
  var_n = "valor_n",
  files = "space-separated list of paths to include
  in context device",
  target= "device to attach the context device" ]

#---------------------------------------
#  Planificador
#    Los requisitos pueden usar expresiones como:
#    $<variable_plantilla>
#    $<variable_plantilla>[<atributo>]
#    $<variable_plantilla>[<atributo>,
<atributo2>=<valor2>]
#---------------------------------------

REQUIREMENTS = "Expr_booleana_para_requisitos"
#Opcional
RANK         = "Expr_aritm_para posicionar_hosts"
#Opcional
```

Apéndice B

Ejemplo de uso

En este anexo recogemos un par de ejemplos de uso simplificados. Para ello utilizaremos tanto la interfaz de línea de comandos de OpenNebula, como el centro de operaciones Sunstone. La topología de nuestra nube está formada por un «front-end» y un nodo (de nombre "www") sobre el que desplegaremos las máquinas virtuales. Éstas lanzarán una imagen de «ttylinux».

B.1. Interfaz de línea de comandos

La interfaz de línea de comandos es la forma más básica, pero más completa, de manejar OpenNebula. Veamos paso a paso cómo lanzar y cancelar una máquina virtual:

1. **Iniciar OpenNebula:** Para iniciar OpenNebula ejecutamos:

   ```
   oneadmin@hector:-> one start
   ```

   Este comando prepara y lanza el demonio de OpenNebula junto con el «scheduler». Los drivers de VirtualBox, configurados en el "oned.conf" se cargan en este momento.

2. **Registrar un host:** Para añadir nuestro nodo a OpenNebula, ejecutamos el comando:

   ```
   oneadmin@hector:-> onehost create www im_vbox
   vmm_vbox tm_nfs
   ```

   Tras unos momentos es posible comprobar que el nodo se ha añadido y se monitoriza correctamente:

   ```
   oneadmin@hector:-> onehost list

     ID NAME  CLUSTER  RVM   TCPU   FCPU   ACPU
      0 www   default    0      0      0   100

     TMEM    FMEM    STAT
     869M    425.3M   on
   ```

```
oneadmin@hector:-> onehost show 0

HOST 0 INFORMATION
ID                    : 0
NAME                  : www
CLUSTER               : default
STATE                 : MONITORED
IM_MAD                : im_vbox
VM_MAD                : vmm_vbox
TM_MAD                : tm_ssh

HOST SHARES
MAX MEM               : 889892
USED MEM (REAL)       : 454396
USED MEM (ALLOCATED)  : 0
MAX CPU               : 0
USED CPU (REAL)       : 0
USED CPU (ALLOCATED)  : 0
RUNNING VMS           : 0

MONITORING INFORMATION
ARCH=i686
CPUSPEED=0
FREECPU=0.0
FREEMEMORY=435496
HOSTNAME=www
HOST_TIME=2011-06-11t11
HYPERVISOR=vbox
MEMORY_AVAILABLE=401 mbyte
MEMORY_SIZE=869 mbyte
MODELNAME=Mobile Intel(R) Pentium(R) 4 CPU
3.20GHz
NETRX=0
NETTX=0
OPERATING_SYSTEM=Linux
OPERATING_SYSTEM_VERSION=2.6.34.8-0.2-desktop
PROCESSOR_0_DESCRIPTION=Mobile intel(r)
 pentium(r) 4 cpu 3.20ghz
PROCESSOR_0_SPEED=3200 mhz
PROCESSOR_1_DESCRIPTION=Mobile intel(r)
pentium(r) 4 cpu 3.20ghz
PROCESSOR_1_SPEED=3200 mhz
PROCESSOR_COUNT=2
```

```
PROCESSOR_ONLINE_COUNT=2
TOTALCPU=0
TOTALMEMORY=889892
USEDCPU=0.0
USEDMEMORY=454396
\end{lstlisting}
\item{\bf Registrar una imagen:} Para registrar necesitamos una plantilla \newl
\begin{lstlisting}
oneadmin@hector:-> cat vbox_image.one

NAME = ttylinux_vbox
PATH = "/srv/cloud/one/share/examples/vbox/
/ttylinux.vdi"
TYPE = OS
```

Podemos registrar esta plantilla fácilmente con:

```
oneadmin@hector:-> oneimage register vbox_image.one
```

Y comprobamos que se ha registrado correctamente (estado «ready»):

```
oneadmin@hector:-> oneimage list

  ID   USER         NAME       TYPE
   0 oneadmin   ttylinux_vbox   OS

     REGTIME            PUB   PER  STAT  #VMS
Jun 11, 2011 11:49      No    No   rdy     0
```

3. **Instanciar una máquina virtual:** En este momento estamos listos para instanciar una máquina virtual utilizando VirtualBox. Para ello, hemos escrito una plantilla "vbox_vm.one" para describirla. En esta plantilla indicamos a OpenNebula que debe utilizar la imagen previamente registrada y mostrar la salida de video de la máquina en una ventana SDL:

```
oneadmin@hector:-> cat vbox_vm.one

NAME = "vbox-test"
MEMORY = "300"

FEATURES = [
  pae  = "no",
  acpi = "yes" ]
```

```
DISK = [
  image_id = 1,
  target = "hda",
  bus = "sata",
  type = "disk"
 ]

GRAPHICS = [
  type = "sdl" ]
```

Para crear la máquina ejecutamos:

```
oneadmin@hector:-> onevm create vbox_vm.one
```

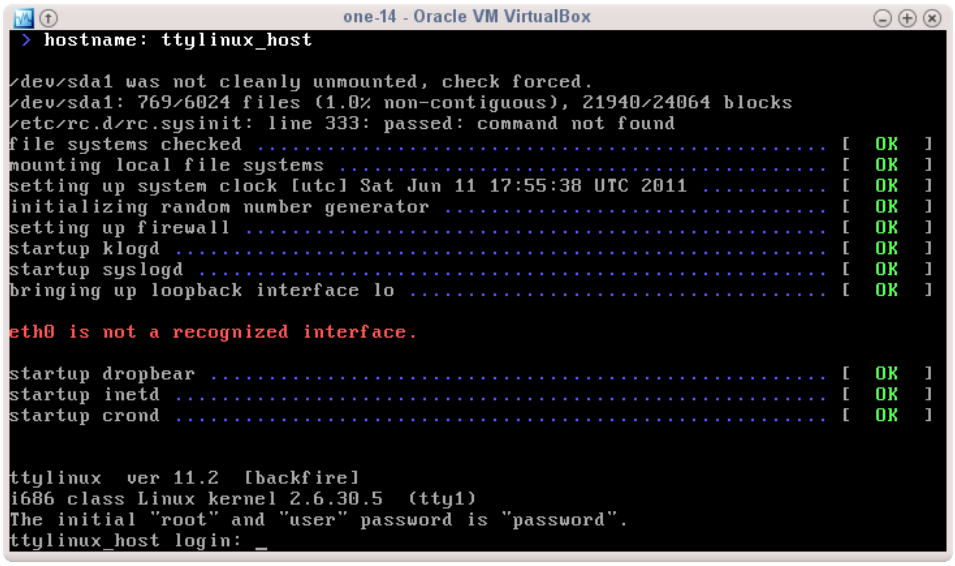
El «scheduler» de OpenNebula se encargará de buscar un nodo dónde instanciar esta máquina virtual. Se realizarán las operaciones necesarias (copiar la imagen al nodo por ejemplo) y, si todo va bien, el estado de la máquina virtual pasará a «RUNNING»:

```
oneadmin@hector:-> onevm list

   ID     USER     NAME    STAT  CPU     MEM
   11   oneadmin vbox-tes  runn   0      0K

   HOSTNAME        TIME
     www       00 00:01:25
```

Observamos que en el nodo remoto aparece una ventana SDL con la salida de video de la máquina virtual (tener en cuenta que hay que tener una instancia del servidor X lanzada, proporcionar acceso al usuario oneadmin a esta instancia y tener definida la variable display para que funcione).



4. **Cancelar una máquina virtual:** Finalmente, podemos cancelar la máquina virtual con:

   ```
   oneadmin@hector:-> onevm cancel 11
   ```

   Con esta orden se apagará la máquina y se eliminará de OpenNebula.

B.2. Interfaz gráfica Sunstone

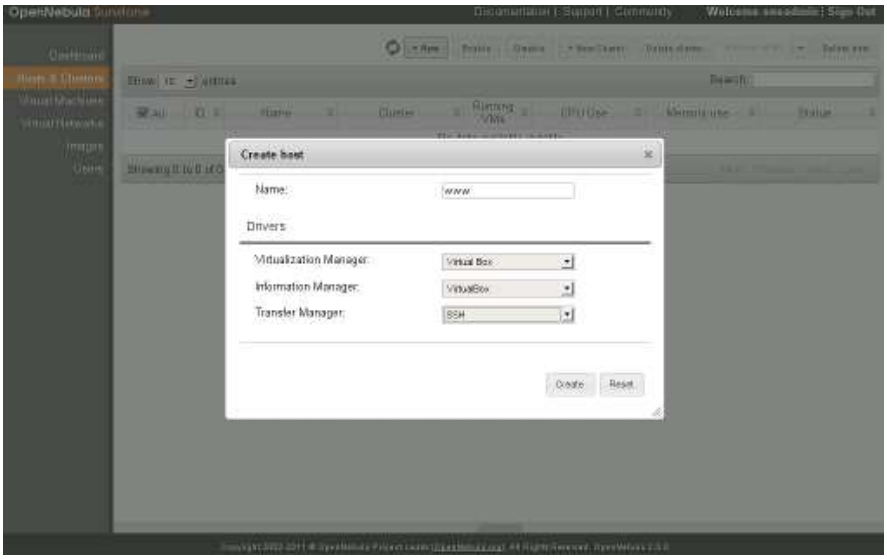
El centro de operaciones Sunstone ofrece una interfaz web para la utilización de OpenNebula de manera fácil y sencilla. Con ayuda del plugin para VirtualBox desarrollado, vamos a observer cómo lanzar una máquina virtual, salvaguardarla y restaurarla. Finalmente la apagaremos:

1. **Iniciar Sunstone:** Para iniciar Sunstone es necesario, una vez OpenNebula está funcionando, ejecutar el comando:

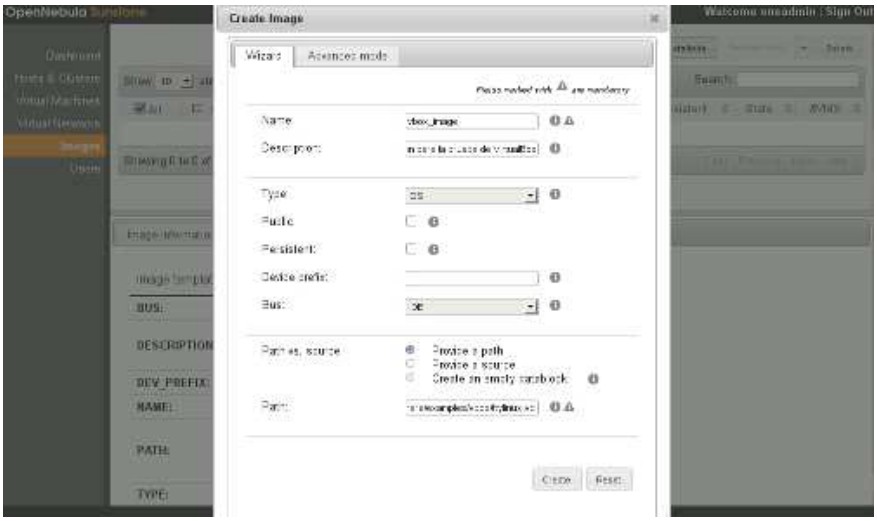
   ```
   oneadmin@hector:-> sunstone-server -p 9292 start
   sunstone-server started
   ```

   Seguidamente es posible acceder al interfaz a través de nuestro explorador en el puerto especificado:

2. **Registrar un host:** Para registrar un nuevo host hay que ir a la pestaña de «Host & Clusters», hacer click en "+ New" y completar el formulario. El host aparecerá en la lista tras ser creado:



3. **Registrar una imagen:** En la pestaña de «Images» podemos registrar la imagen que usaremos para nuestra máquina virtual:



4. **Instanciar una máquina virtual:** Para crear una máquina virtual sólo tenemos que rellenar el formulario para VirtualBox en la pestaña de «Virtual Machines» y pulsar el botón crear. Hemos seleccionado la imagen que hemos definido antes. A los pocos segundos podemos observan que nuestra máquina virtual ha pasado ha estado «RUNNING». Además, en el host ha aparecido la ventana SDL de ttylinux, tal y como había pasado en el ejemplo de la línea de comandos:

5. **Salvaguardar una máquina virtual:** Salvaguardar una máquina virtual es tan fácil como seleccionarla y hacer click en la acción «stop» el desplegable de acciones. La máquina pasa a estado «stopped» cuando ha finalizado la transferencia de los discos y el estado al «frontend»:

6. **Restaurar una máquina virtual:** Seleccionarla de nuevo y hacer click en la acción de «resume» la devolverá a estado «PENDING», lista para ser instanciada de nuevo en el host que esté disponible y ejecutándose en el mismo punto en el que fue salvaguardada.

7. **Apagar una máquina virtual:** Por último, la acción shutdown nos permite apagar la máquina virtual, que recibirá una señal acpi de apagado y desaparecerá de la lista cuando se haya completado: