Herramienta de Provisión de Entornos Virtuales en Clouds Privados

José Gabriel Puado Puado *David Baena Menoyo* *Fernando Martínez-Conde Mayor*

Proyecto de Sistemas Informáticos, Facultad de Informática
Universidad Complutense de Madrid

Director: Rubén Santiago Montero
Departamento de Arquitectura de Computadores y Automática
Curso 2011/2012

Declaración de conformidad

Los alumnos:

José Gabriel Puado Puado, David Baena Menoyo, Fernando Martínez-Conde Mayor, 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 2012

José Gabriel Puado Puado	David Baena Menoyo	Fernando Martínez-Conde Mayor

Agradecimientos

José Gabriel:

El proyecto de Sistemas Informáticos supone el principio el fin de una etapa, por ello no puedo olvidar mencionar a todos los compañeros y amigos que me han apoyado y ayudado a lo largo de toda la carrera. Por supuesto agradecer a mi familia, que han demostrado que siempre estarán ahí, he pasado buenos y malos momentos y ellos lo saben mejor que nadie.

También agradecer a las comunidades de software abierto y software libre, y en especial de OpenNebula y Chef. A toda la gente que desinteresadamente realiza sus aportaciones, no solo en forma de código, también ayudando desde listas de correo, chats irc o foros, a los que estábamos "un poco perdidos" por allí.

David:

Y de repente todo vuelve a cambiar, el final de una etapa y el comienzo de una nueva. Mientras escribo estas líneas cada vez estoy más cerca de llegar a la tan ansiada meta, en la que siempre me he sentido respaldado por mis amigos, tanto los que ya han llegado a ella, como los que no han llegado pero llegarán al igual que yo. Y sobre todo por mi familia, los que sabían desde el primer día que podría.

Gracias por ser mi motivo de superación.

"…Tu tiempo es limitado, así que no lo malgastes viviendo la vida de otro. No te dejes atrapar por el dogma que implica vivir según los resultados del pensamiento de otros. No dejes que el ruido de las opiniones de los demás ahogue tu propia voz interior. Y lo que es más importante, ten el coraje de seguir a tu corazón y tu intuición. De algún modo él ya sabe lo que realmente quieres llegar a ser. Todo lo demás es secundario.
Sigue hambriento. Sigue alocado."

Steve Jobs

Fernando:

"Nunca dejes que nadie te diga que no puedes hacer algo. Si tienes un sueño, tienes que protegerlo. Las personas que no son capaces de hacer algo te dirán que tú tampoco puedes. Si quieres algo, ve a por ello, y punto."

En busca de la felicidad

Hay decisiones, buenas o malas que te cambian la vida. Durante un tiempo estuve convencido de que la peor decisión en la mía fue estudiar informática. Ahora, a toro pasado, creo todo lo contrario, no pude haber elegido mejor, ya que de haberlo dejado no habría aprendido tanto, conocido tan buena gente, ni crecido como lo he hecho.

Gracias de corazón a mi familia y a mis amigos por apoyarme en todo momento en esta etapa, y lograr que me sienta como si no tuviera límites para hacer cualquier cosa que me proponga.

"Fuerza y Honor"

Índice de Contenidos

1. Resumen

El Cloud Computing consiste en un paradigma en el que el usuario accede a los distintos recursos a través de la red, sin preocuparse de más, tan solo espera que estos “le lluevan” desde la nube. Sus características lo hacen un paradigma muy potente, gracias en gran parte a su escalabilidad y la popularidad cada vez mayor de Internet.

Podemos clasificar las nubes, en primer lugar, atendiendo a su modelo de implementación: Nubes Privadas, Nubes Públicas, Nubes Comunitarias y Nubes Híbridas. Por otro lado, atendiendo a los modelos de servicio, es decir, que capa de servicio nos proporciona: SaaS, IaaS y PaaS.

La virtualización es una técnica que permite que varios recursos físicos aparezcan como sólo un recurso lógico, o por el contrario, que un recurso físico aparezca como si fueran varios recursos lógicos.

La herramienta desarrollada aportará a OpenNebula la capacidad de provisión de entornos virtualizados, apoyándonos en la potencia de Chef.

OpenNebula es un software que permite la creación y administración de máquinas virtuales en la nube, para ello emplea diferentes hipervisores, como Xen o KVM.

Chef se define como “un framework de integración de sistemas orientado a la nube”, y tiene como lema “infraestructura como código”. Chef se distribuye en diferentes “sabores”, para nuestro desarrollo escogemos Chef Solo, su versión *standalone*.

La herramienta de provisión de entornos ha sido desarrollada en respuesta a la necesidad de realizar de forma rápida y flexible la provisión de estos entornos por diversos motivos, como la posible caída de uno de los hosts o la réplica automática de los entornos. En la memoria se detallan los distintos aspectos en cuanto a su desarrollo y funcionamiento.

Por último, se analizan las posibles ampliaciones futuras, fruto del trabajo realizado y de la experiencia adquirida.

1.1. Palabras Clave:

Herramienta de Provisión, Entornos, Computación en la Nube, Virtualización

2. Abstract

Cloud Computing is a paradigm where users can access to various resources through a network, without worrying too much, just expects that these resources "will rain down" from the cloud. Its features make it a very powerful paradigm, thanks to its scalability and the growing popularity of Internet.

Firstly, we can classify the clouds, according to their implementation model: Private Clouds, Public Clouds, Community Clouds and Hybrid Clouds. On the other hand, taking into account the service models, which one provides service layer: SaaS, IaaS and PaaS.

Virtualization is a technique that allows multiple physical resources appearing as a logical resource, or conversely, a physical resource appearing as if they were more logical resources.

The developed tool will provide OpenNebula the ability to provision virtualized environments, relying on the power of Chef.

OpenNebula is toolkit that allows creating and managing virtual machines in the cloud, to do it, uses different hypervisors such as Xen or KVM.

Chef is defined as "a systems integration framework built specifically for automating the cloud" and has as slogan "Infrastructure as Code". Chef is distributed in different "flavors", we have chosen Chef's standalone version: Chef Solo.

The provisioning environments tool has been developed in response to the need to provide these environments quickly and flexibly for various reasons: the possible fail of one of the hosts or automatic environments replication. In this document is detailed the different aspects in their development and function.

Finally, we analyze the possible future expansions, as a result of the work done and the experience acquired.

2.1. Keywords:

Provisioning Tools, Environments, Cloud Computing, Virtualization

3. Introducción

Este proyecto otorga una nueva funcionalidad al software de OpenNebula, configurar de forma muy sencilla una máquina virtual mediante contextualización.

Esta nueva funcionalidad permite abaratar costes en la gestión y mantenimiento de máquinas virtuales pudiendo configurar una determinada instalación con las características deseadas por el usuario de forma muy simple y flexible. De hecho, el coste de configurar la instalación de un cierto software en una máquina y hacerlo en más de una es el mismo.

El proyecto entra dentro del ámbito del Cloud Computing, tecnología que día a día va ganando cada vez más terreno y que se podría decir que es presente y futuro de la informática.

En la ejecución de este proyecto se ha trabajado principalmente con dos herramientas, que son OpenNebula y Chef, conceptos que en capítulos posteriores se tratarán más a fondo.

OpenNebula es, a grandes rasgos, una herramienta open-source desarrollada en la Universidad Complutense de Madrid que permite crear fácilmente cloud privados, públicos e híbridos.

Con la aplicación de Chef a la hora de gestionar los entornos en OpenNebula, nos adentramos en el mundo del los DevOps. Toda una filosofía emergente, sin aún una definición clara, que pretende eliminar barreras entre administradores y desarrolladores.

Uno de los caballos de batalla de los DevOps es Chef. Esta herramienta proporciona, como veremos en detalle, diversos recursos para describir configuraciones mediante código. Para el que emplea un DSL basado en Ruby, lenguaje interpretado y orientado a objetos, enfocado a la simplicidad y de creciente popularidad.

Estas configuraciones son independientes de la máquina sobre la que se aplican, es decir, podrán ser desplegadas con los mismos resultados sobre diferentes distribuciones.

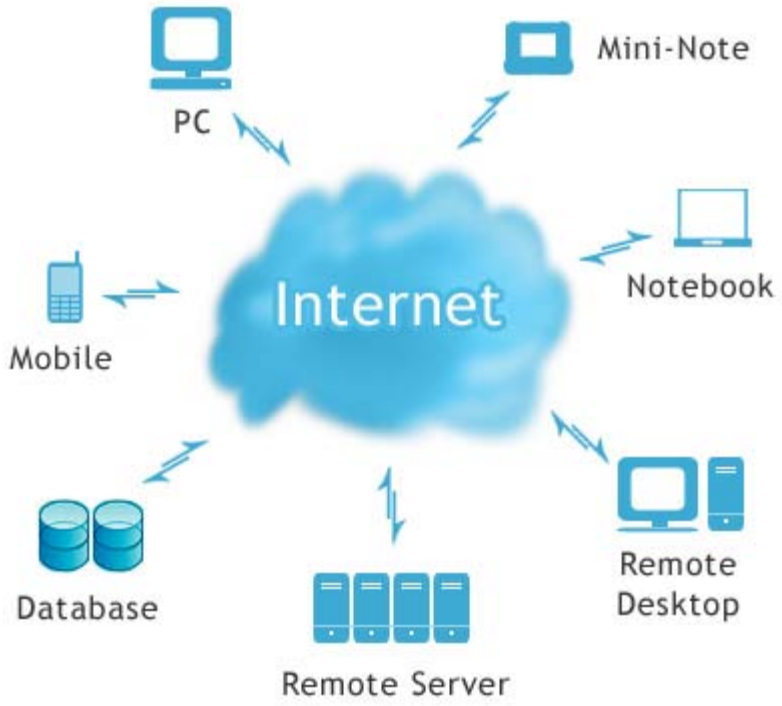
A todo esto podemos sumar la numerosa comunidad que tiene detrás, que nos pueden aportar no solo ayuda, sino numerosos recursos y configuraciones adaptables a nuestros objetivos.
Todo ello en conjunto, hace de Chef una herramienta muy potente, que aportará a OpenNebula posibilidades casi infinitas a la hora de crear y proveer entornos.

4. Cloud Computing

4.1. ¿Qué es el Cloud Computing?

El término Cloud significa nube, y es una metáfora de internet basado en el dibujo que se utiliza para representar el diagrama de red de ordenadores. El usuario no tiene que preocuparse de cómo está construida esa nube, sino que “espera que le lluevan los recursos”.

Por tanto la computación en la nube se refiere a las aplicaciones y servicios que se ejecutan en una red distribuida utilizando recursos virtualizados, a los que se accede a través de Internet.



Representa un cambio en el paradigma en el modo que se usan los sistemas, ya que no es necesario tener todos los recursos alojados en su propio dominio físico. La escalada masiva de los sistemas de computación en la nube es posible gracias a la popularización de Internet.

En definitiva, la nube hace posible la computación de utilidades con un sistema de prepago, escalable supuestamente hasta el infinito.

4.2. Características

Podemos enumerar una serie de características relacionadas con este paradigma de computación.

Autoservicio a demanda: Un consumidor puede provisionar capacidades computacionales así como tiempo de uso del servidor, gestión de almacenamiento dependiendo de sus necesidades, de forma automática, sin intermediación humana con cada proveedor de servicio.

Amplio acceso a través de la red: Las capacidades están disponibles a través de la red, y se puede acceder desde plataformas diferentes (teléfonos móviles, PDAs, laptops)

Disponibilidad de recursos: Un proveedor de servicios de nube crea recursos que se reúnen en un sistema compatible con el uso en multitenencia. Los sistemas físicos y virtuales se asignan dinámicamente según sea necesario.

Elasticidad rápida: Los recursos se pueden suministrar de manera rápida y elástica. El sistema puede añadir recursos escalando sistemas. Esta es una de las principales características, la sensación para el usuario de tener recursos ilimitados.

Servicio medido: Se suelen usar métricas para poder facturar los recursos utilizados por los clientes, de manera que se podrá contabilizar el almacenamiento, transacciones, ancho de banda o cantidad de energía utilizada. De esta forma el cliente podrá saber qué elementos realmente utiliza, y el proveedor sabrá qué beneficio obtendría al proporcionárselos.

4.3. Modelos

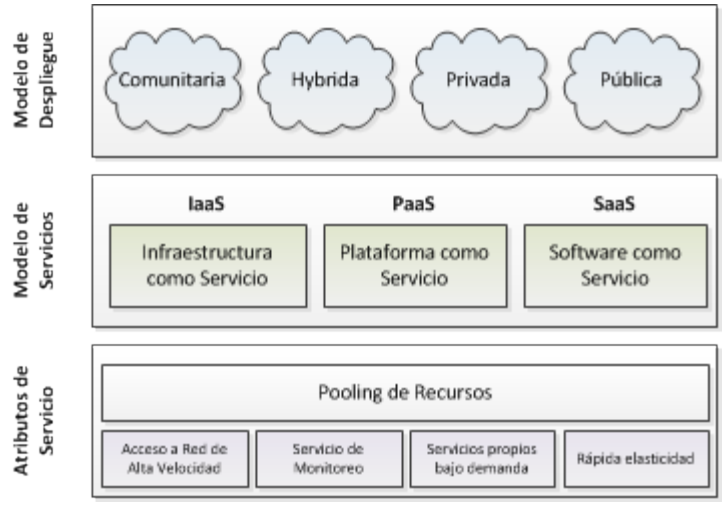
Para poder hablar de Cloud Computing es necesario definir ciertos términos de computación. Mucha gente separa la nube en dos conjuntos de modelos distintos: **modelos de implementación y modelos de servicio**.

Los modelos de implementación se refieren a la ubicación y administración de la infraestructura de la nube. Son modelos que definen donde están alojados los recursos.

Los modelos de servicio son los que definen los servicios a los que se puede acceder en una plataforma de computación en la nube.

4.4. El modelo NIST

El gobierno de EEUU es un gran consumidor de servicios informáticos, y uno de los principales usuarios de redes de computación en la nube. El NIST[1] tiene un conjunto de definiciones que separan la computación en la nube en modelos de servicio y modelos de implementación.



En sus orígenes, NIST no requería una nube para la virtualización para reunir los recursos, ni requería que una nube fuera compatible con la multitenencia.
La versión más reciente de la definición NST requiere que las redes de computación en la nube utilicen virtualización y soporten la multitenencia.

Debido a que la computación en la nube se mueve hacia un conjunto de componentes modulares que interactúan entre ellos en base a ciertos estándares (como la arquitectura orientada a servicios), podemos esperar que las futuras versiones del modelo NIST añada también estas características.

Este modelo no trata varios servicios de intermediarios como agentes de servicios, integración y abastecimiento que forman parte de muchos debates sobre la computación en la nube.

4.5. Modelos de Implementación

El modelo de implementación se refiere a al propósito de la nube y donde se ubica esta. Existen cuatro modelos de implementación según el NIST:

[1] *National Institute of Strandards and Technology*, Instituto nacional de normas y tecnología

4.5.1. Nube Pública

La infraestructura de la nube pública está disponible al uso público, o para un gran grupo industrial y es propiedad de una organización que vende servicios de nube.

4.5.2. Nube Privada

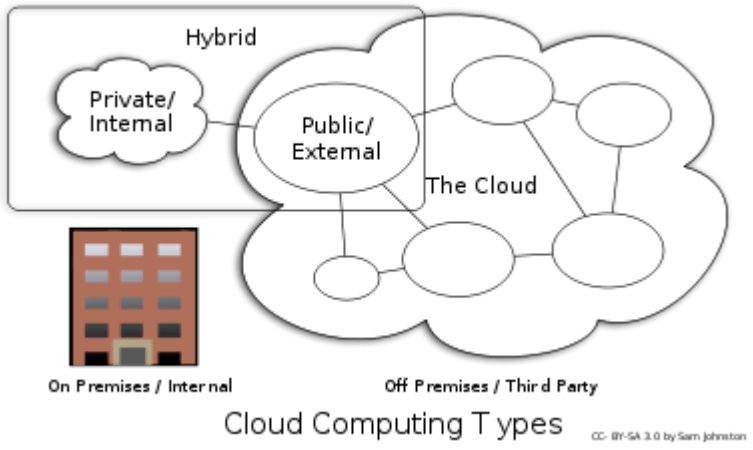
La infraestructura de una nube privada funciona para el uso exclusivo de una organización: Puede ser administrada desde la misma o desde terceros, las nubes privadas pueden estar alojadas dentro o fuera de las instalaciones.

4.5.3. Nube Comunitaria

Es la que se ha organizado para servir a un propósito o función común. Comparten preocupaciones comunes como su misión, políticas, seguridad. Podrá administrarse a través de la organización u organizaciones que la constituyen, o de terceros.

4.5.4. Nube Híbrida

Una nube híbrida combina varias nubes (privada, comunitaria o pública), donde se mantienen sus identidades pero se vinculan como una unidad.



4.6. Modelos de servicio: Saas, Iaas, Paas

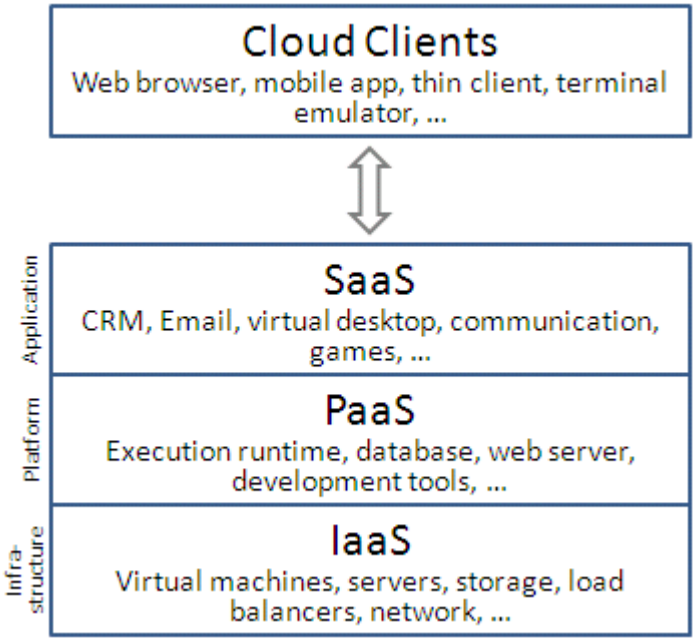
En el modelo de implementación, podemos imaginar la nube como el límite entre donde acaban la administración, las responsabilidades, y la red de un cliente y donde empieza el proveedor de servicios.

Dichos servicios podrán variar según lo que ofrezca el proveedor, por lo que entramos en un terreno donde nos interesaría saber que clases de servicios se pueden ofrecer. A este conjunto de definiciones de servicios lo llamamos modelos de servicio.

Hay varios modelos de servicio, los cuales toman todos toman una forma predeterminada.
“Xaas, o <X> como servicio”.

Dependiendo del nivel de abstracción que tengamos podemos tener un tipo de servicio u otro. Hay tres clases diferentes, con ello el usuario final puede variar dependiendo cuales sean sus necesidades.

La pila de servicios sería la siguiente:



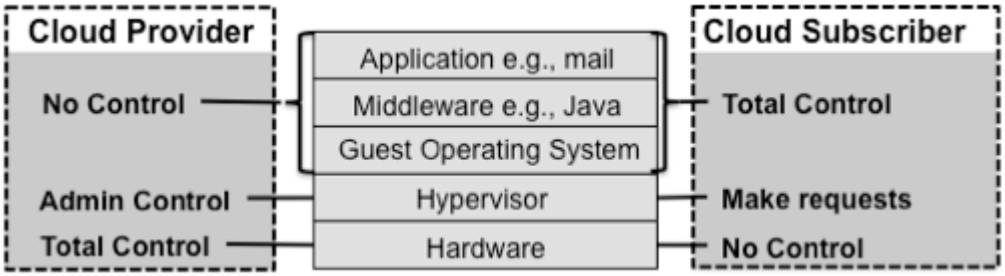
Finalmente, llegaríamos a los usuarios que utilizan los recursos software de la nube por medio de aplicaciones como visores web, terminales, etc. De manera que estos son la mayoría de consumidores de la nube.

4.6.1. IaaS (*Infrastructure as a Service*)

El proveedor facilita el acceso a recursos hardware virtualizados, así como máquinas virtuales, sistemas de almacenamiento, y redes.

Gracias a ello el cliente renuncia a tener sus propios recursos físicos, por lo que usa unos los que le son provistos. De esta forma el cliente sería el encargado de incorporar a esta infraestructura las aplicaciones que necesite.

Por otro lado no es responsable de configuración y mantenimiento de la infraestructura subyacente (servidores, router, switches, etc)



En cuanto al proveedor no tiene control sobre los elementos incorporados al recurso virtualizado.

El principal suscriptor a este tipo de servicios es de tipo empresarial para poder tener virtualizado todos sus sistemas para dar servicio a sus propios usuarios.

Una serie de proveedores comerciales de este tipo de infraestructuras serían: Amazon Web Services (AWS EC2, AWS S3), Rackspace Cloud, GoGrid.

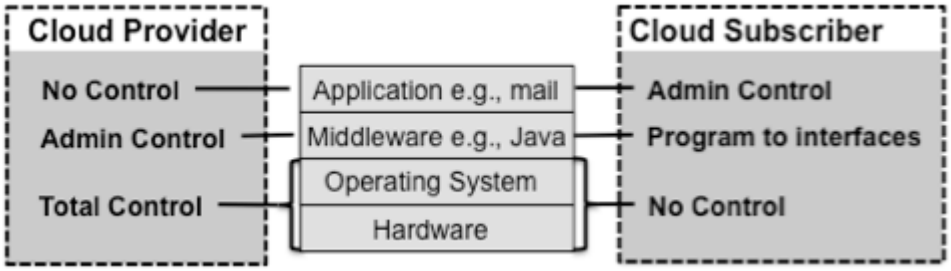
Un cliente de este tipo de servicios es Facebook, el cual necesita más de 12000 servidores en Amazon para poder dar servicio a uno de sus juegos más usados en el sistema

4.6.2. PaaS (*Platform as a Service*)

Provee un acceso a un entorno de programación, de desarrollo y ejecución, el cual se sustenta sobre una infraestructura escalable en la que existen componentes hardware y middleware pertenecientes al proveedor.

El suscriptor realiza todas las actividades necesarias para la vida en el ciclo del software, así como su desarrollo, ejecución, y testeo.

El desarrollador no gestiona ni controla la infraestructura, por tanto los elementos hardware, servidores, y todo aquello hasta llegar al sistema operativo es para él transparente. Por otro lado tiene que administrar las aplicaciones desplegadas y configuración de los entornos para la ejecución de estas.



El proveedor de servicios tiene acceso a la infraestructura subyacente, y no puede controlar las aplicaciones y entornos configurados por el usuario.

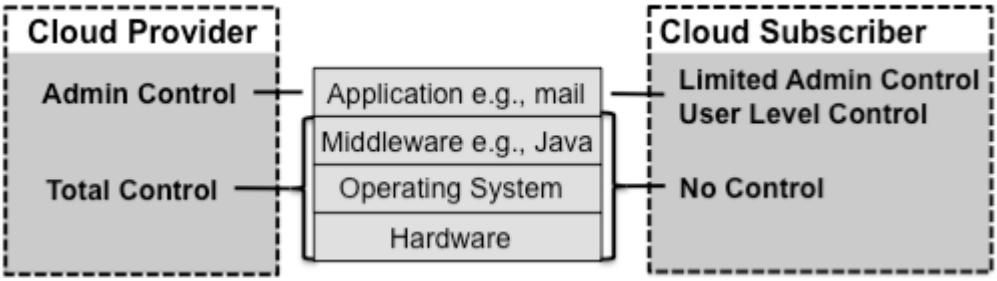
Un ejemplo de proveedores comerciales de este tipo de servicios son: Google App Engine, Microsoft Azzure Services.

Los posibles clientes de este tipo de servicios van desde desarrolladores individuales, hasta empresas que se dedican al desarrollo y venta de software como ISV (*Independent software vendor*)

4.6.3. SaaS (*Software as a Service*)

Es un entorno operativo completo con aplicaciones, administración e interfaz de usuario, para que pueda utilizar directamente ese software.

En este modelo, la manera de proporcionarle al usuario el servicio es por medio de un cliente ligero, normalmente es un navegador. Todas estas aplicaciones son ejecutadas en la infraestructura del proveedor y controladas por él, es decir todo lo que va desde la aplicación hasta la infraestructura es responsabilidad del proveedor.



Por ejemplo, Gmail entraría en este nivel de abstracción. Es Saas, donde Google es el proveedor, y el consumidor sería el usuario que utiliza ese servicio a través de un navegador.

El usuario no tiene conocimiento de donde realmente se están almacenando sus correos, o el protocolo usado para enviarlos, sino que únicamente se limita a enviar y recibir correos.

No hace falta que el usuario sea un experto para utilizar dicho servicio, pero también podría configurar ciertos parámetros de esa aplicación a su gusto, por ejemplo poner filtros de correo spam, crear bandejas específicas para cada temática de correo, etc. Es decir, si tiene capacidad de configurar la aplicación pero sólo hasta donde el proveedor de servicios le permita.

Normalmente los suscriptores van desde empresas y organizaciones hasta usuarios, con la característica común que no necesitan tener conocimientos avanzados sobre el servicio que quieren utilizar, es decir, no necesitan tener conocimientos de cómo montar un servidor de correo electrónico.

Ejemplos de Saas son: Google Docs, Google Maps, GMail también ofrecido por Google.

Los tres modelos de servicio en conjunto se conocen como modelo SPI de computación en la nube. Aunque se ha hablado de otros modelos como StaaS[2], como por ejemplo el servicio de almacenamiento de DropBox, IdaaS[3] donde se prestan servicios de identificación de usuarios para entornos donde la seguridad es un punto clave.

4.7. Virtualización:

La virtualización es una técnica utilizada sobre las características físicas de ciertos recursos computacionales para ocultarlas de otros usuarios, sistemas y aplicaciones con los que interactúan. Por ello hay que hacer que varios recursos físicos (servidores, dispositivos de almacenamiento) aparezcan como sólo un recurso lógico, o que un recurso físico (SSOO, servidor, etc...) aparezca como si fueran varios recursos lógicos.

Por otro lado también se puede virtualizar un sistema operativo (SO) mediante aplicaciones que permiten que un mismo SO maneje varias imágenes de SSOO a la vez.

4.7.1. Tipos

Existen múltiples formas de ver la virtualización pero las principales son la Emulación, Virtualización completa o full-virtualization y la Paravirtualización:

- **Emulación:**
  Se basa en crear máquinas virtuales que emulan el hardware de una o más plataformas hardware distintas. Obliga a simular completamente el comportamiento de la plataforma hardware a emular.

[2] Storage as a Service

[3] Identity as a Service

Cada instrucción que se ejecuta en estas plataformas ha de ser traducida al hardware real.
Debido a esto es la menos eficiente y la más costosa.

- **Virtualización completa:**
  Permite ejecutar Sistemas operativos huésped (Guest) sobre un sistema anfitrión (Host) sin tener que modificarlo. Para ello se utiliza un hipervisor como capa intermedia que permite compartir el hardware real. El hipervisor es el encargado de monitorizar los Guest para poder capturar instrucciones protegidas de acceso al hardware (que de forma nativa no pueden realizarse debido a que no se tiene acceso directo a él).
  Los sistemas operativos se pueden ejecutar sin necesidad de modificar la plataforma, aunque el sistema operativo debe estar soportado en la arquitectura virtualizada.
  El rendimiento es mucho mayor que en la emulación, aunque aún dista del rendimiento frente a una plataforma nativa, por el uso de la monitorización y el hipervisor.

- **Paravirtualización:**
  Se basa en que los sistemas virtualizados deben estar basados en sistemas operativos debidamente modificados para ejecutarse sobre un hipervisor. Esto es porque, de ese modo, no es ya necesario que el hipervisor monitorice todas y cada una de las instrucciones, si no que es ayudado por los sistemas operativos huésped y anfitrión.

4.7.2. Ventajas en el uso de virtualización

En el ámbito empresarial tenemos múltiples ventajas. Por ejemplo, si una empresa ha de cambiar aplicaciones o realizar otro tipo de cambios, podría (mediante virtualización) conservar los mismos equipos y realizar todos los cambios antes mencionados a través de un entorno virtualizado en la red, donde la fuente sería el servidor.

Otra opción sería tener dos módulos de trabajo, uno de ellos sería utilizable de forma totalmente libre, donde el usuario podría instalar software, agregar dispositivos a su antojo, etc...Mientras que en el otro sólo estaría lo realmente vital para la supervivencia de la empresa, así, si el primero sufriera algún accidente como colapsos o caídas el segundo podría seguir funcionando de forma totalmente normal evitando considerables pérdidas económicas a la empresa.

4.7.3. KVM

Kvm o Kernel-based Virtual Machine es una herramienta de virtualización basada en GNU/Linux. Realiza una virtualización completa, lo que da más flexibilidad y usabilidad que el modelo de virtualización por emulación. Al ser un módulo del propio kernel no hace falta arrancar kernels especiales ni aplicar parches, así como modificar el kernel del sistema operativo que ejecutaremos dentro de la máquina virtual.

Debido a que soporta tecnología NUMA tiene gran escalabilidad. Otra de sus ventajas es su fácil instalación, sólo son necesarios tres paquetes, quemu, kvm-kmp y kvm. Como partes negativas están el que sólo sirve para Linux y que es necesario poseer un procesador con soporte para virtualización por hardware para poder hacer uso de esta tecnología.

Por último, destacar su interfaz de escritorio para administrar las máquinas virtuales, Virtual Machine Manager, que permite visionar el funcionamiento y la utilización de los recursos en tiempo real. Trae incorporado también un cliente de VNC.

5. Herramienta de gestión de Clouds

5.1. OpenNebula

5.1.1. ¿Qué es?

OpenNebula es un software de código abierto creado para la virtualización de infraestructuras de Nube y centros de datos.

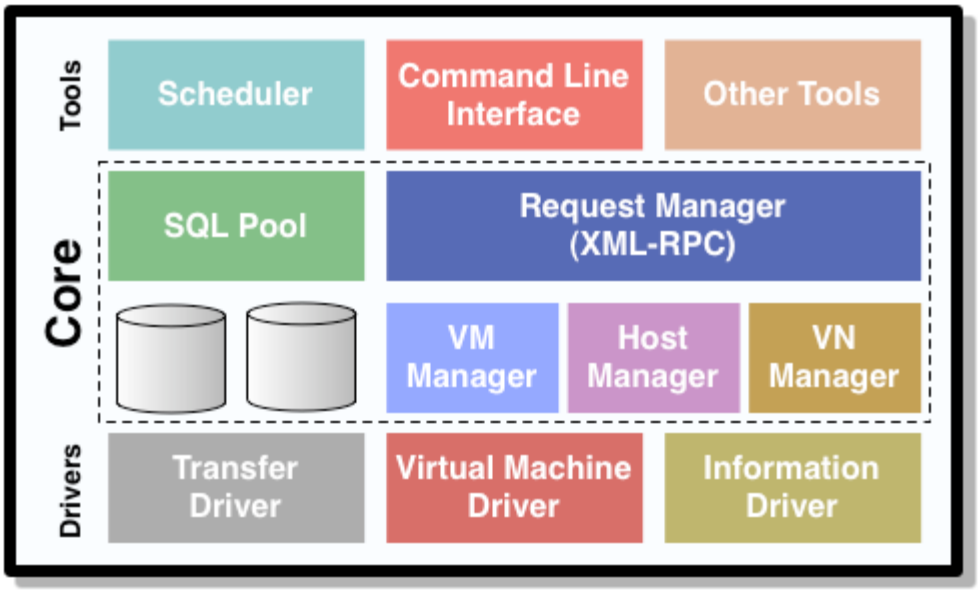
Permite crear y gestionar las máquinas virtuales existentes en la nube y proporciona una gran adaptabilidad para los datacenters, así como una alta escalabilidad. Destaca ciertamente la posibilidad de hacer uso de hipervisores como XEN ó KVM.

Por su simplicidad y potencia, así como su compromiso con el open-source, es una herramienta por la que, cada vez más, grandes y pequeñas empresas así como otros usuarios se interesan.

5.1.2. Arquitectura

La arquitectura de OpenNebula está formada por 3 capas:

- Herramientas(Tools)
- Núcleo (Core)
- Controladores (Drivers)



- **Capa de herramientas (Tools)**
  Estas herramientas se desarrollan utilizando las interfaces proporcionadas por la capa de núcleo de OpenNebula. Tenemos dos elementos principales en esta capa, que son el CLI y el Scheduler.
  - Scheduler: Usa la interfaz XML-RPC para invocar acciones en máquinas virtuales.
  - CLI: Para la interacción entre el usuario y OpenNebula.
  - Otras Herramientas: Herramientas que pueden crearse fácilmente usando la interfaz XML-RPC o la nueva API de OpenNebula, OCA.

- **Núcleo (Core):**
  En esta capa se encuentran componentes diseñados para el manejo de almacenamiento, redes virtuales, host y las máquinas virtuales.
  - Bases de datos: Un modelo de almacenamiento persistente basado en SQLITE3 para las estructuras de datos ONE.
  - Request Manager: Utilizado para manejar las peticiones de usuario.
  - Virtual Machine Manager: Su uso es el de gestionar y monitorizar las máquinas virtuales.
  - Transfer Manager: Para la gestión de imágenes de máquinas virtuales.
  - Virtual Network Manager: Utilizado para gestionar las diversas redes virtuales.
  - Host Manager: Gestiona y monitoriza los recursos físicos.

- **Controladores(Drivers) :**
  Para conectar con el núcleo diferentes tipos de virtualización, almacenamiento y tecnologías de monitorización, así como servicios propios de la nube.
  Esta capa interactúa con el hipervisor, servicios de la nube, así como mecanismos de transferencia de archivos.

5.1.3. Modelo de gestión

Para lanzar una máquina virtual requerimos de cuatro elementos indispensables, host, template, virtual network e imagen. Todos ellos se pueden gestionar bien desde la interfaz de línea de comandos, o bien desde la GUI de Sunstone, mucho más sencilla.

Antes de nada, deberemos iniciar sesión como ‘oneadmin’ y después iniciar sesión en OpenNebula mediante el comando:

```
oneadmin@ejemplo:~$ one start
```

Host

Un host es un servidor que tiene la capacidad de arrancar máquinas virtuales y que está conectado al servidor Frontend de OpenNebula.
OpenNebula se puede adaptar a las distintas configuraciones que les queramos dar a los Host, es decir, pueden conectarse Hosts con distintos hipervisores, así como diversas distribuciones de Linux.

Para el manejo de host tenemos el comando 'onehost', que tiene diversas funciones, como son, por ejemplo:

- Añadir Host a OpenNebula.
- Listar todos los Hosts.
- Borrar uno de ellos.
- Habilitar o Deshabilitar Host.
- Mostrar información sobre uno de ellos.

Imagen

En OpenNebula un usuario puede añadir imágenes de sistemas operativos o de datos para usarlos posteriormente en las máquinas virtuales de un modo muy simple.
Una imagen puede usarse en distintas máquinas virtuales al mismo tiempo, así como ser compartida entre distintos usuarios.

En este caso, el manejo de imágenes será mediante el comando 'oneimage', teniendo como funciones principales para este comando las de:

- Añadir Imagen.
- Borrar una imagen determinada.
- Hacer pública o privada una Imagen (esto es, para que otros usuarios puedan hacer uso de esta si es pública, y no pudiendo hacerlo si es privada).
- Hacer una imagen persistente: (para que se guarden los cambios una vez apagada la máquina virtual).
- Mostrar información de una de las imágenes en particular.

Red Virtual

Un host está conectado a una o más redes virtuales que están disponibles para las máquinas virtuales a través de los puentes. En OpenNebula podemos definir redes privadas, pudiendo asociarse las máquinas virtuales a varias de estas redes.
Por otro lado, también podemos conectar un conjunto específico de máquinas virtuales. Incluso se les puede permitir el acceso a internet conectándose a una red virtual específica.

El manejo de redes virtuales se puede ejercer por medio del comando 'onevnet'. Este comando tiene funciones entre otras como:

- Crear Red virtual, desde un fichero template.
- Borrar una Red virtual.
- Hacer pública o privada, de cara a que la puedan utilizar otros usuarios o no.
- Añadir 'leases' a la red virtual.

- Mostrar información sobre una red virtual específica.

Template

Una template de OpenNebula es una definición de una máquina virtual. Estas definiciones se guardan en un repositorio de templates en los que las templates se pueden ver como ficheros de texto (con un formato determinado) que almacenan una información importante de cara a lo que será luego nuestra máquina virtual. Las templates tienen dueño y contienen información sobre los privilegios en el manejo de estas.

Hay muchos parámetros de configuración en el fichero de la template, entre otras cosas se puede configurar:

- Capacidad de la memoria y de la CPU.
- ID de la red virtual que vamos a usar.
- ID de la imagen que vamos a usar.

En cuanto al comando para manejar templates, se llama ‘onetemplate’, y sus principales funciones son:

- Crear template (a partir del fichero del que antes se ha hablado)
- Listar templates.
- Mostrar información de una template determinada.
- Borrar una template.
- Actualizar template.
- Hacer pública o privada una template.
- Instanciar template (Esto crea una instancia de máquina virtual a partir de la template seleccionada).

Maquina Virtual

Una vez hemos instanciado la template obtenemos una máquina virtual. Sobre ella podremos hacer uso de ciertas operaciones por medio del comando ‘onevm’. Tenemos las siguientes posibles operaciones:

- Create (Añade una nueva máquina virtual desde el fichero template al Pool de las máquinas virtuales)
- Deploy (Arranca una máquina virtual sobre un Host)
- Shutdown(Apagar una máquina virtual ya existente).
- Migrate: para la máquina virtual y continúa en un Host determinado.
- LiveMigrate (Igual que Migrate pero sin parar la máquina virtual. Esto requiere de un sistema de almacenamiento compartido).
- Hold: retiene la VM pasando esta a un estado Hold (ver imagen posterior adjunta).
- Release. Se libera la VM del estado Hold y pasa al estado Pending (pendiente).
- Stop: Se transfiere el estado de la máquina virtual de vuelta al front-end.
- Cancel: La VM es destruida. Es distinto a shutdown en cuanto a que si estamos utilizando imágenes persistentes, se guardan los cambios.

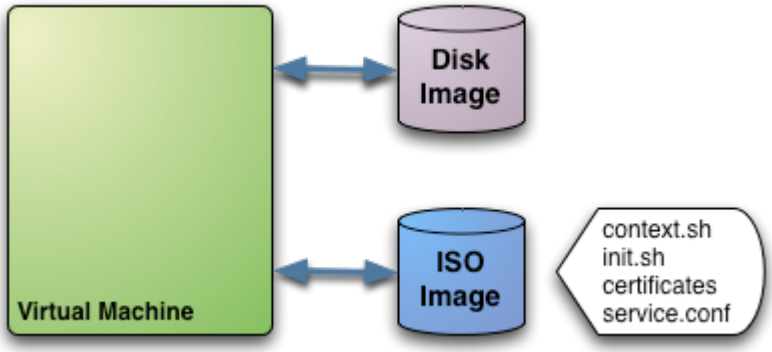
- Suspend: Hace lo mismo que Stop pero el estado de la VM permanece en el Host, de forma que se podrá continuar si se desea
- Resume: Se prosigue con la ejecución de una VM.
- Delete: Destrucción total de la VM.
- Reboot: Se vuelve a arrancar la VM.
- Resubmit: La VM vuelve al estado Pending.
- Otras: Saveas, List, Show y top.

La siguiente imagen recoge todos los posibles estados de VM posibles así como sus transiciones dependiendo de las operaciones que apliquemos sobre ella.

5.1.4. Contextualización Genérica

El método que proporciona OpenNebula para pasar los parámetros de configuración a una nueva máquina virtual es utilizar una imagen ISO, método que también puede ser usado para configurar las interfaces de red.

Para ello tenemos que especificar los contenidos del ISO en el fichero de descripción de la máquina virtual.



En el ejemplo de la imagen superior tenemos, por una lado una máquina virtual y por otro dos imágenes asociadas a ella, una imagen de Disco y una imagen ISO.

- Imagen de Disco: Tiene el sistema de archivos desde el que el sistema operativo se ejecutará.

- Imagen ISO: Tiene la contextualización para la máquina virtual, con los siguientes objetos:

    - context.sh: Fichero que contiene las variables de configuración, obtenidas por OpenNebula a partir de los parámetros especificados por el usuario en el fichero de descripción de la máquina virtual. Este fichero es el único que siempre se pasa por defecto.

    - init.sh: Script que será llamado por la máquina virtual al arranque del sistema. Configurará servicios específicos para esta máquina virtual.

    - certificates: Directorio que contiene ciertos certificados que pueden ser necesarios para ejecutar diversos servicios.

    - service.conf: Para la configuración de servicios.

6. Herramientas de Provisión

Por herramientas de provisión, entendemos el software empleado para la gestión y provisión de configuraciones. Este software, es la base de trabajo de los DevOps y nos permite gestionar, crear y configurar entornos de un modo sencillo y flexible. Las dos herramientas de facto son Chef y Puppet. Ambas con una extensa comunidad detrás, punto muy importante a la hora de resolver problemas, y aprovechar los recursos que otros miembros brindan.

6.1. Chef

Se autodefinen como un framework de integración de sistemas especialmente orientado a la nube. Chef es desarrollado por el equipo de Opscode, quienes lo distribuyen bajo licencia open-source y se encuentra escrito en Ruby. Su lema es "infraestructura como código", y permite mediante "porciones" de código denominadas recetas, configurar entornos de forma dinámica y muy flexible, junto con otros elementos como veremos más adelante. El lenguaje que nos permitirá realizar las configuraciones es Chef-DSL, basado en Ruby, el cual destaca por su sencillez. No obstante, no será necesario ser un conocedor amplio de Ruby, desde la wiki de Chef se pueden consultar los conceptos necesarios para comenzar a escribir nuestras propias recetas[4].

6.1.1. Recursos

Uno de los principios de Chef reside en que hay diferentes formas de realizar lo mismo (TMTOWTDI[5]). Ello se permite mediante una serie de elementos que mostramos a continuación:

Nodos (Nodes)

Un nodo está asociado a la máquina que corre Chef y que es objeto de configuración. Se trata de un archivo JSON compuesto por en un listado de atributos y el runlist. Este último consiste en una lista de roles y recetas que serán expandidos y ejecutados por Chef. En resumen, es el punto de partida desde el que Chef generará el árbol de recursos necesarios para configurar una máquina concreta.

[4] http://wiki.opscode.com/display/chef/Just+Enough+Ruby+for+Chef

[5] "There's More Than One Way To Do It"

Atributos (Attributes)

Elementos fundamentales en Chef, se trata de datos o parámetros asociados a los nodos, que pueden ser asignados dinámicamente desde diferentes sitios como veremos más adelante. Por su parte, Chef a través de Ohai, dará valor a algunos de estos atributos al inicio en función del nodo sobre el que se ejecuta, aunque estos valores podrán ser sobrescritos al definirlos de nuevo desde *cookbooks*, *environments*, *roles* y *nodes*.

Chef permite definirlos según una jerarquía de precedencia. Se distinguen tres niveles de mayor a menor precedencia: *override*, *set* o *normal* y *default*. A su vez, como hemos visto los atributos pueden ser definidos desde diferentes lugares, por lo que tendremos además un orden de precedencia en función de donde lo hagamos. Por ejemplo, un atributo al que damos valor desde un rol y más tarde de una forma más específica desde una receta, tomará el valor asignado en la receta si tienen ambos el mismo valor de prioridad por tener la esta última prioridad sobre el rol; Si de otro modo en el rol estuviera declarado como *override* y en la receta como *normal*, estaríamos forzando a tomar el valor declarado por el rol.

Atendiendo a ambos factores tenemos la siguiente lista de mayor a menor precedencia:

1. atributos *override* aplicados directamente desde la receta.
2. atributos *override* aplicados desde un entorno.
3. atributos *override* aplicados desde un rol.
4. atributos *override* aplicados desde un archivo de atributos (cookbook).
5. atributos *normal* aplicados directamente desde la receta.
6. atributos *normal* aplicados desde un archivo de atributos (cookbook).
7. atributos *default* aplicados directamente desde la receta.
8. atributos *default* aplicados desde un role.
9. atributos *default* aplicados desde un entorno.
10. atributos *default* aplicados desde un archivo de atributos (cookbook).

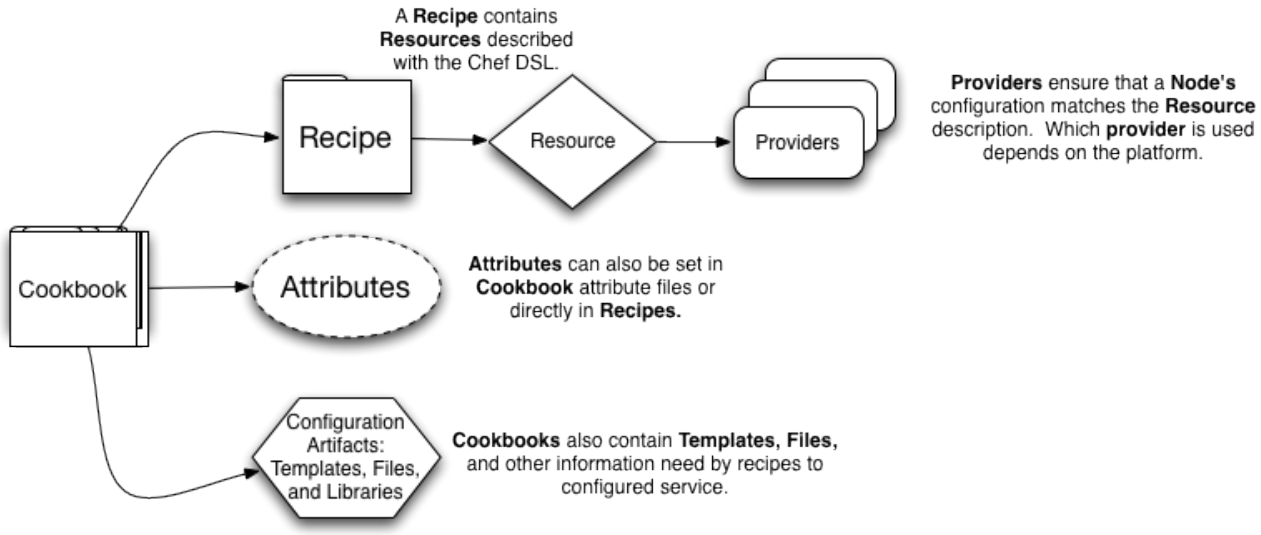
Cookbooks

Son las "unidades fundamentales de distribución". Se trata de paquetes que encapsulan una o varias recetas como elementos principales, sin embargo estas se apoyan en diferentes elementos de configuración:

- *Metadata*: Todo cookbook debe contener un archivo *metadata.json,* introducido a mano o generado por knife a partir de *metadata.rb*. Este contiene información dirigida a Chef-Server, tal como nombre, versión del cookbook, descripción, dependencias...
- *Templates*: Se tratan de archivos plantilla, escritos en formato eRuby. Este formato nos permite introducir expresiones Ruby ha ser ejecutadas en un texto plano. Esto nos permite generar dinámicamente archivos en función de las

variables o atributos introducidos. Comúnmente, las templates son empleadas para crear archivos de configuración para el cookbook que las contiene, podríamos por ejemplo, generar dinámicamente un archivo .vimrc para Vim personalizado, en función de algunos atributos.

- Recetas (*Recipes)*
- *Ligthweight Resources* and *Providers*: Permiten programar recursos y proveedores más específicos para ser ejecutados por las recetas.
- Librerías (*Libraries)*: Como su propio nombre indica, se trata de librerías que extienden el cookbook aportando nuevas clases y/o funcionalidades.
- *Definitions*: Definen recursos a partir de otros más básicos, permitiendo la reutilización del código desde la misma o desde distintas recetas dentro del cookbook.



Recetas (Recipes)

Se trata del código que determinará la configuración deseada, las “piezas” ejecutables que serán lanzadas en un determinado orden para obtenerla. Pueden contener dependencias de otras recetas del mismo o distinto cookbook, conformando así una estructura modular y jerárquica. A su vez están compuestas por recursos.

Recursos (Resources)

Son las unidades básicas de trabajo en Chef. Indican las acciones que serán ejecutadas desde las recetas. Estas acciones son independientes de la plataforma sobre la que se aplican, aquí es donde entran en juego los proveedores. Los proveedores son quienes deciden cómo llevar a cabo estas acciones en función de la plataforma en cuestión, por ejemplo, para el recurso *package* se aplicará la instalación mediante *apt-get* si la plataforma es Debian, o por el contrario empleará *yum* si la plataforma es Fedora o RedHat. Los recursos son también los responsables de

asegurar la idempotencia, es decir, el resultado de ejecutarlo varias veces tendrá siempre el mismo resultado. Es importante conocer, que se pueden comunicar entre sí mediante subscripción y notificación, y que se ejecutan en distinto tiempo que el código Ruby que pueda acompañar en la receta.

Roles

Muy similares a los archivos nodo en cuanto a funcionalidad, se componen por atributos y un runlist al igual que ellos. Su características son más bien semánticas y jerárquicas, agrupando roles y recetas con propiedades u objetivos comunes. También añaden una nueva capa de prioridad en la asignación de atributos. Chef acepta roles escritos en DSL con extensión ".rb" o en formato JSON (".json").

Data Bags

Como su propio nombre indica, se tratan de "bolsas de datos". Contienen información que puede ser accedida desde las recetas para trabajar con ellas. Por ejemplo, para crear los distintos usuarios del sistema. Se estructuran como archivos json (*data*) contenidos en un directorio determinado (*bag*).

Entornos (Environments)

Llegaron como novedad con la llegada de la versión 0.10 de Chef. Los entornos, añaden una capa de abstracción más a Chef, al permitir dar valor a atributos, seleccionar rangos de versiones para los cookbooks y poder asociar Data Bags; también permiten un nuevo tipo de runlists en los roles, definiendo el runlist que se expandirá dependiendo del entorno en que se ejecute. El formato es similar al de los roles, y al igual que en estos se pueden encontrar tanto en formato ruby como JSON, en el primer caso, Chef lo transformará a formato JSON para poder manejarlo con mayor facilidad.

6.1.2. Arquitectura

El equipo de Opscode proporciona dos "sabores" de Chef:

- Chef Client + Chef Server: Es la opción típica y más completa para aprovechar toda la potencia de Chef. En ella un servidor mantendrá los datos persistentes a los que el cliente accederá (para descargar cookbooks, realizar búsquedas...), mediante REST.
- Chef Solo: Se trata de una versión *standalone* del cliente de Chef, que no necesita del servidor para su funcionamiento. El resultado es una versión más ligera y portable, a cambio de algunas restricciones. Careceremos de este modo de los environments y de soporte de versionado de los cookbooks, y de las operaciones de búsqueda de *data bags* (aunque existen otros modos para poder emplearlas[6]).

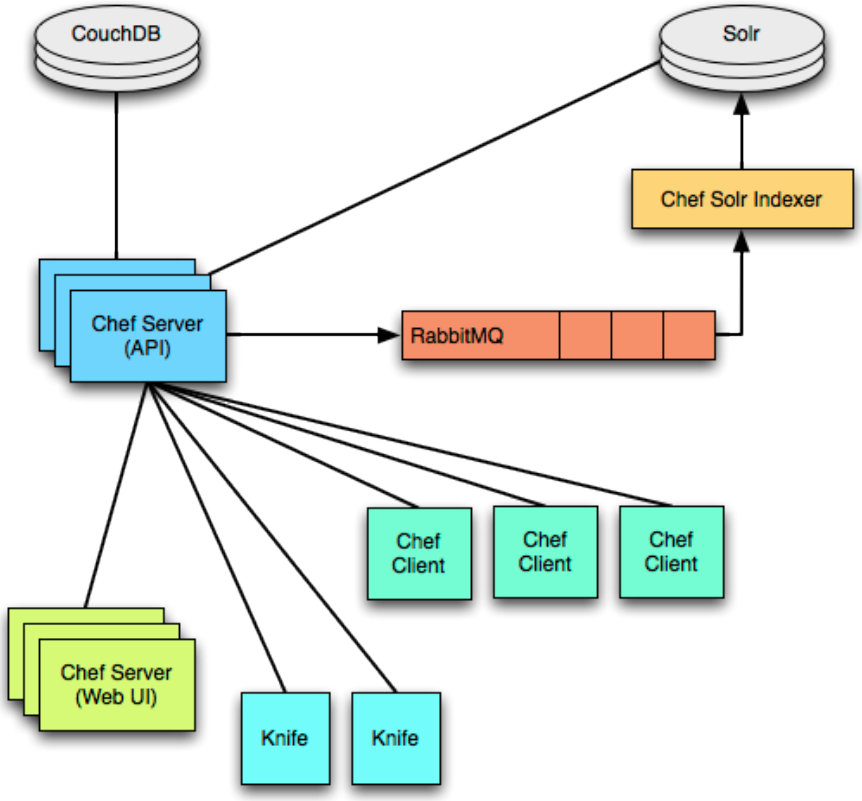
[6] Chef Solo sólo permite acceder a las data bags directamente mediante los métodos *data_bag()* y *data_bag_item()*. Sin embargo, desde la wiki de Chef se puede acceder a una cookbook desarrollada por un miembro de la comunidad, que permite realizar búsquedas: https://github.com/edelight/chef-solo-search

A pesar de que para el desarrollo de nuestra herramienta hemos empleado Chef Solo, es importante conocer la arquitectura de la versión completa para comprender en mayor profundidad el modo de trabajo y el funcionamiento de ambas.

En primer lugar, conviene aclarar que existen tres modos de servidor: Hosted, Private y Open Source. Los dos primeros son gestionados por Chef, por lo que nos centraremos en el último.

Chef Server proporciona el punto central de distribución de cookbooks, lleva a cabo la autentificación de los nodos y permite las operaciones de búsqueda. Para llevar a cabo todo esto se apoya en las siguientes herramientas:

- CouchDB: Base de datos donde almacenan los datos referentes a nodos, roles, data bags... Siempre en formato JSON.
- RabbitMQ: Se trata de un software de negociación de mensajes que implementa el estándar AMQP. En nuestro contexto, es una herramienta que permite al servidor las peticiones de indexado para Chef Solr, actuando como buffer ante posibles picos de peticiones a los que el indexador Solr no pueda hacer frente, o almacenándolos en disco en caso de fallo.
- Solr Indexer: Solr es un buscador de código abierto de la fundación Apache, que Chef emplea en este caso para indexar todos los elementos en la base de datos.



Cada vez que se realiza una modificación a través de la API (vía la interfaz web de usuario, knife o Chef Client), esta se encola en RabbitMQ. Toda interacción con la API llevará un proceso de autentificación y autorización. En este punto, Chef-expander tomará esta información y tras procesarla se la enviará a Solr para su indexado. Chef-expander está compuesto por uno o varios *workers*, cada *worker* es *single thread*, por lo que empleará como máximo el 100% de una CPU.

Este método nos permitirá realizar búsquedas rápidamente, para encontrar los cookbooks necesarios para su descarga en el cliente, o la búsqueda de los data bags desde una receta por ejemplo.

En cualquier caso, el verdadero protagonista es Chef Client. Distinguimos varias fases:

Convergencia

1. Creación del nodo: El primer paso de la ejecución de Chef, en el que Ohai toma todos los datos útiles del sistema operativo y la máquina sobre la que se está ejecutando, y cualquier dato previo proveniente del servidor. Con parte de esta información se crean los atributos generados por Ohai como *automatic*.
2. Registro con Chef Server: Se procede al registro del cliente en el servidor. Para ello el cliente tomará temporalmente la identidad de un cliente especial denominado *chef-validator*, con ella será capaz de interactuar con el servidor para crear una nueva clave privada específica para ese cliente (*client.pem*).
3. Sincronización de Cookbooks: Una vez autentificado en el servidor, el cliente hará una petición a este, a la que responderá con una lista de las cookbooks almacenadas actualmente y sus componentes.

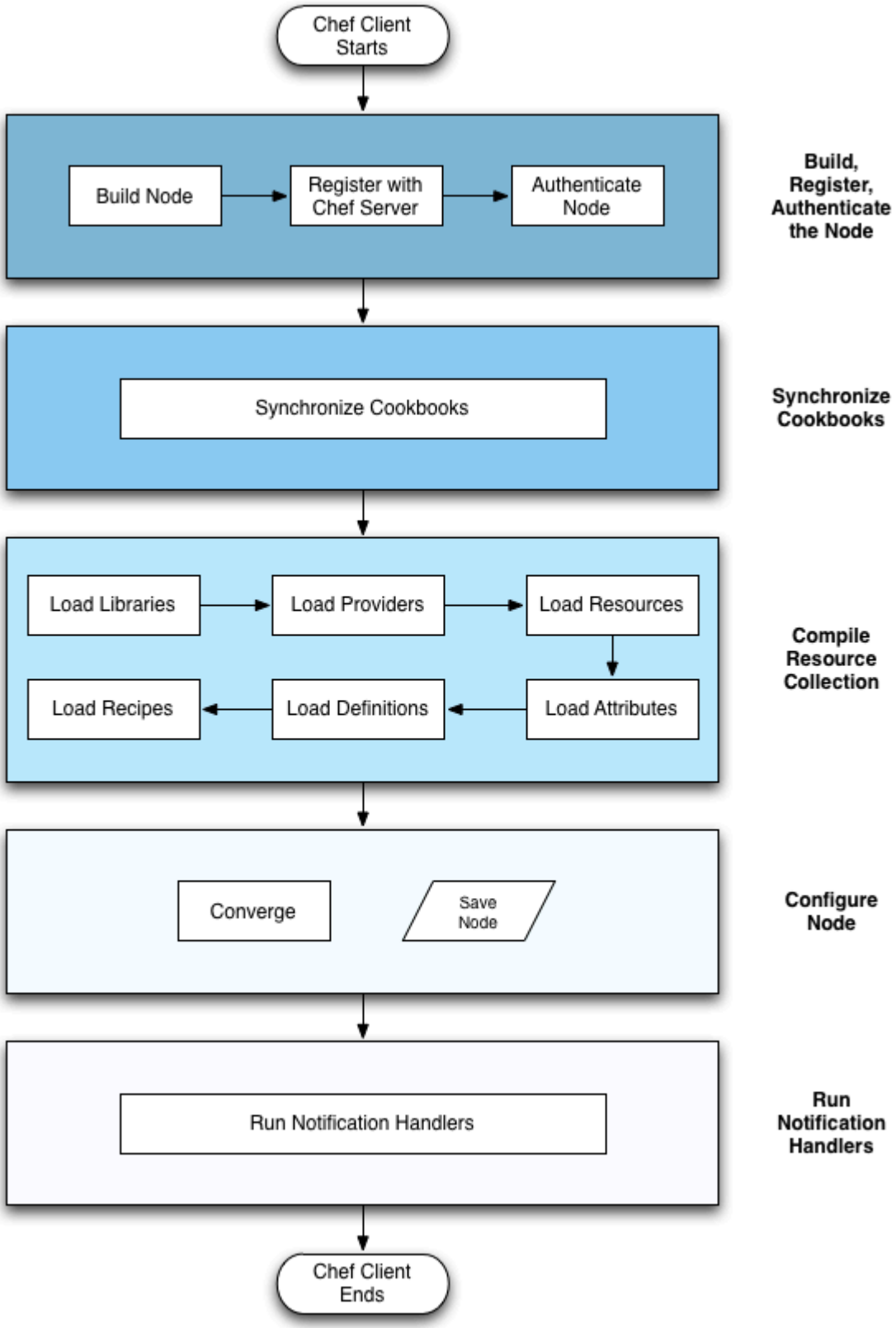
Compilación

Se procede a la carga de los distintos elementos de cada cookbook necesario para ese nodo en orden: librerías, atributos, definiciones y recetas. Por último se procede a la evaluación de las recetas, esto tiene como resultado un array de cada uno de los recursos evaluados, junto con la evaluación del resto de código ruby fuera de estos. Es lo que se conoce como *Resource Collection*. En cualquier caso, aún no se ha llevado a cabo ninguna acción de las recetas.

Ejecución

En este punto Chef ya dispone de los elementos necesarios para comenzar la ejecución:

1. Convergencia: Se toma cada recurso en la *Resource Collection*, y se le asigna un proveedor encargado de tomar la acción.
2. Guardar el nodo: Chef guarda el nuevo estado del nodo en el servidor, para tenerlo disponible en la búsqueda.
3. Ejecución de notificaciones.


Chef Client Starts
Build Node
Register with Chef Server
Authenticate Node
Build, Register, Authenticate the Node
Synchronize Cookbooks
Synchronize Cookbooks
Load Libraries
Load Providers
Load Resources
Load Recipes
Load Definitions
Load Attributes
Compile Resource Collection
Converge
Save Node
Configure Node
Run Notification Handlers
Run Notification Handlers
Chef Client Ends

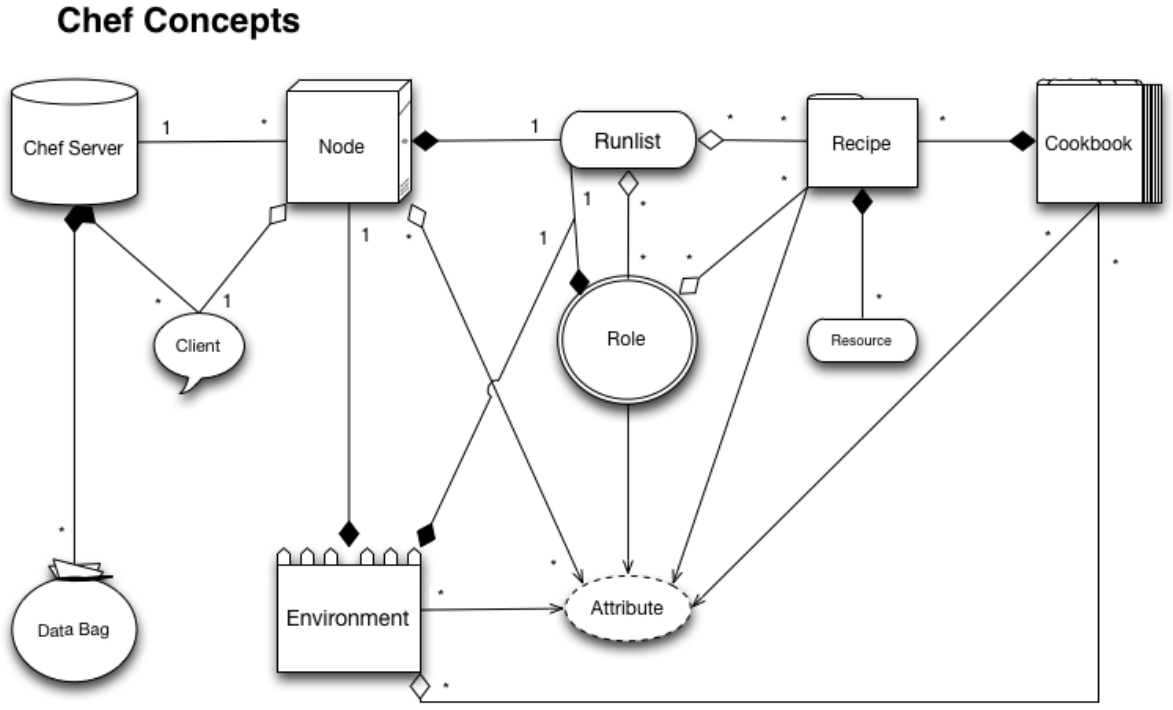
En resumen, existe un único Chef Server que da servicio a varios clientes, comúnmente asociados a un único nodo. Cada nodo puede pertenecer a uno o varios entornos, y se caracteriza por su runlist y atributos. El runlist es una lista de roles y recetas a ejecutar en el nodo.

Cada rol, a su vez, contiene su propio runlist que puede contener más nodos y/o roles. Chef expandirá el árbol partiendo del runlist del nodo, hasta alcanzar todas las recetas a ejecutar en este.

Las recetas se agrupan en cookbooks, y pueden existir dependencias entre ellas dentro o fuera del cookbook al que pertenecen. Todas las recetas están compuestas por recursos que plantean las acciones que deberán ejecutar los proveedores para alcanzar la configuración deseada.

Los atributos pueden aparecer en nodos, entornos, roles, recetas y cookbooks. Como vimos anteriormente, siguen un orden de prioridades según cómo y en qué lugar sean declarados.

Los data bags, permanecen en el servidor, donde son accedidos cuando el cliente desea realizar alguna consulta para la ejecución de las recetas.



7. Sistema de provisión de entornos

7.1. ¿Qué es un entorno virtual?

Podríamos definir entorno como el conjunto de condiciones extrínsecas que necesita un sistema informático para funcionar correctamente. Dependiendo del contexto de dicho entorno podrían variar esas condiciones mínimas a cumplir.

Gracias a ese contexto podemos catalogar los entornos según las funcionalidades que nos ofrezcan. Un ejemplo sería un entorno de desarrollo software, donde tendríamos un IDE como Eclipse y JDK 1.6 para poder crear aplicaciones sobre el lenguaje JAVA.

Las condiciones de un correcto funcionamiento sería tener instalado el IDE y el JDK 1.6 para el correcto funcionamiento de ese entorno de desarrollo.

Existen muchos tipos de entornos, los cuales no hacen falta que tengan que interactuar directamente con una persona, como por ejemplo un servidor WEB APACHE, donde podemos alojar una página web.

7.2. Gestión de entornos

Desde el punto de vista del usuario, un entorno tendría que tener ciertas características para explotar al máximo sus funcionalidades. Que fuese lo más fácilmente configurable posible sería una de ellas, también es vital que el tiempo empleado para generar el entorno no fuese demasiado largo. Todo esto hace que se empiece a pensar que se necesita alguna herramienta que ayude a gestionar dichos entornos.

Un entorno se tiene que apoyar sobre algún sistema informático, ya sea real o virtualizado, dependiendo para que sea usado se apoyará sobre un sistema u otro. En nuestro caso, nos interesa que sea de manera virtualizada, ya que se va a utilizar como un componente en un Cloud privado.

En definitiva, la herramienta que gestiona los entornos también tendrá que adoptar el paradigma del Cloud computing, junto a la contextualización de entornos. Por lo que se ha desarrollado en base a la herramienta de gestión de Clouds OpenNebula y la herramienta de provisión Chef.

7.2.1. Componentes necesarios

Necesitamos una serie de componentes y requisitos para nuestros entornos alojados en dichos clouds.

Template

Para poder crear un entorno necesitamos una especificación del sistema que va a ser virtualizado, así como la memoria, el número de procesadores, el sistema de almacenamiento, etc. Es decir todos los recursos hardware que queremos virtualizar. A partir de ahora llamaremos a esta especificación *template*.

Un ejemplo de configuración sencilla sería:

- Memory: 512 MB
- CPU:2 procesadores
- DISK:
  - Tamaño: 1GB
  - Tipo: swap
- DISK:
  - Tamaño: 200GB
  - Tipo: hd

Imagen Base

Una vez que tengamos esa especificación, necesitamos un sistema operativo que pueda manejar todos esos recursos. La manera más cómoda y práctica de tener este sistema operativo es a partir de una imagen base que tiene la instalación mínima necesaria para que funcione el sistema. Dicha imagen debe de tener instalado Chef Solo y, por tanto, Ruby para su correcta contextualización.

También será necesaria, la modificación del archivo `/etc/rc.local`, para montar la imagen con el contexto e iniciar el script `init.sh`. Por ejemplo:

```
#!/bin/sh -e

mount -t iso9660 /dev/vdb /mnt/

if [ -f /mnt/init.sh ]; then
    . /mnt/init.sh
fi
umount /mnt

exit 0
```

Contexto

Por último necesitamos un contexto para dicho entorno, con el que se configure como un componente útil para dicho Cloud. Un ejemplo de contexto, sería una especificación

de instalación para el servidor WEB APACHE. Dicha especificación se realiza sobre un Cookbook de Chef.

Si nos fijamos en los componentes que hemos descrito, son todos independientes, por lo que pueden hacerse combinaciones de ellos para crear un entorno diferente cada vez.

Máquinas virtuales

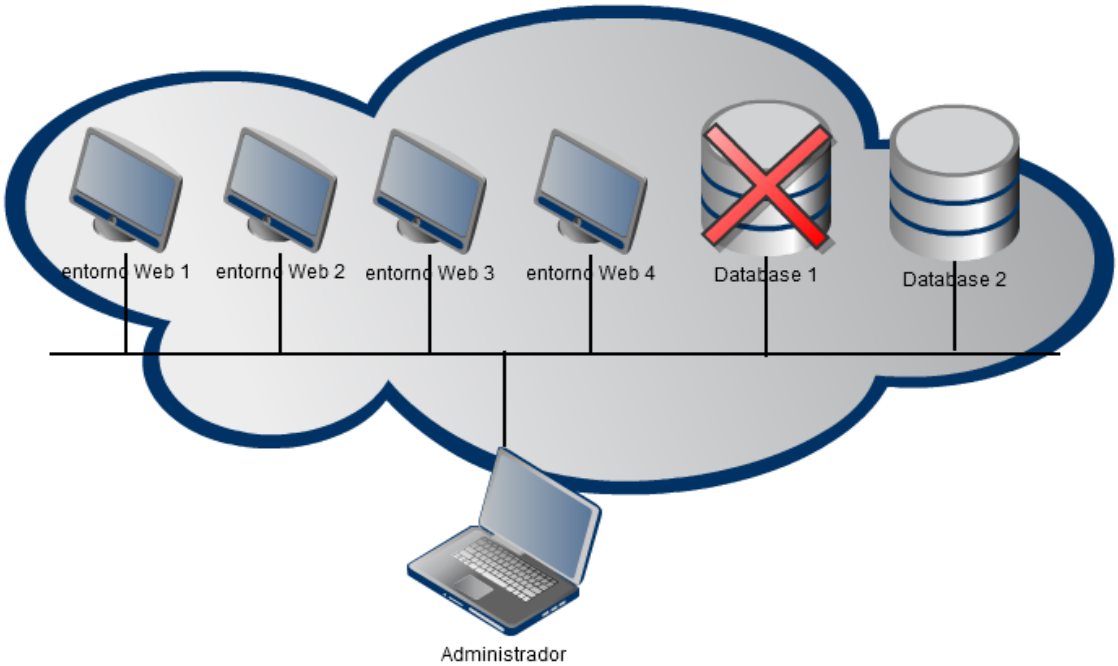
Los componentes descritos anteriormente, son las piezas fundamentales para crear un entorno. Por tanto una vez que las tenemos, nos queda integrarlas en un componente final que es donde tendremos ese entorno. Dicho componente será una máquina virtual que se creará al instanciar una template concreta, junto a una imagen base y un contexto.

Dado que una máquina virtual es usada como uno de los componentes fundamentales de los cloud, podremos tener dicho entorno en nuestra nube.

7.2.2. ¿Por qué aprovisionamiento de entornos en un Cloud?

Si usted es el responsable de un Cloud privado, y tiene gran cantidad de sistemas funcionando a la vez, y por ejemplo uno deja de funcionar correctamente, le gustaría tenerlo en el menor tiempo posible en funcionamiento, y que funcione como la primera vez que lo dejó correctamente configurado.

Un ejemplo sería un Cloud que da servicio de alojamiento de una página WEB. Este se compone de varias clases de entornos: un entorno de base de datos, otro es un servidor web, y un balanceador de carga de trabajo.



Bien una vez que tenemos estas clases de entornos, los tenemos replicados en varias máquinas. Tenemos cuatro entornos WEB, dos entornos de Base de Datos, y un

balanceador de carga. Cada máquina funciona correctamente y todo el metaentorno funciona como debe.

Por causas que se desconocen, una de las máquinas que da servicio de base de datos, deja de funcionar, y por tanto este metaentorno podría fallar. El administrador de este Cloud privado, tendría que volver a instalar desde cero todo ese entorno. Perdería tiempo, en hacer esta acción, y lo peor de todo que tendría que hacerlo siempre que fallase alguna máquina.

Sería más sencillo tener un control de aprovisionamiento que cree entornos en el Cloud, y que cumpla las expectativas de tiempo y sencillez.

7.2.3. Creación de entorno en un Cloud

La herramienta necesaria para gestionar los entornos en Clouds tiene la capacidad de crear un entorno. La manera de hacerlo en la nuestra sería creando una máquina virtual contextualizada por medio de Chef, en un Cloud privado gestionado por OpenNebula.

OpenNebula, dispone de métodos de contextualización de máquinas virtuales. Por medio de una template podemos indicar el contexto para dicha máquina. El contexto será expresado por medio de una o varias recetas de la herramienta Chef.

Por tanto a la hora de crear un entorno, tendremos que indicar que imagen, plantilla y recetas se quieren utilizar.

De esta forma se cumpliría la necesidad del usuario que sea fácil crear un entorno, ya que necesitaría sólo esos tres componentes, al igual que el tiempo necesario para crearlo, ya que no necesitaría instalar todo un sistema operativo y contextualizarlo cada vez que quiera instanciar un entorno nuevo.

7.2.4. Colección de contextos

Llegados a este punto, el usuario necesitará tener alojado en algún sitio todas los contextos, en nuestro caso particular, Cookbooks, para tener un control sobre ellos.
Nuestro sistema tiene un repositorio local que es indicado por el usuario, y es donde están todos situados.

7.2.5. Proceso de contextualización

El método de contextualización, como hemos visto anteriormente, se hace por medio de la template que utiliza OpenNebula para instanciar máquinas virtuales. Usando la contextualización genérica, en la que los datos necesarios son introducidos por medio de una imagen ISO, según la recomendación del estándar OVF[7].

[7] Open Virtualization Format

Los datos almacenados en dicha imagen son los necesarios para dicha contextualización, es decir, la colección de Cookbooks a usar, los ficheros de instalación de dichos Cookbooks, y el Runlist de ejecución de dichos Cookbooks.

Tipos de contextualización

Según con la expresividad que queramos contextualizar una máquina, con nuestro sistema, podemos tener una contextualización sencilla, o una contextualización avanzada.

El criterio de desempate para saber si es de un tipo u otro, son los elementos que se usen dentro del Runlist para poder saber que queremos instalar. Dependiendo si usamos recetas, o roles podremos saber si estamos en un caso u otro.

Los roles podemos verlos como clases de funcionalidades en el sistema, por lo que un rol, podría ser “editor de textos”, y otro “plataforma web”. Si queremos uno mixto sería juntar ambos roles.

Contextualización sencilla

Es aquella en la que el entorno no tiene roles, y con ello únicamente tiene una única clase de funcionalidad. Llamaremos a partir de ahora a este tipo de contextualización, *“runlist plano”*

node.json

```
{"run_list": ["recipe[git::server]", "recipe[vim]"]}
```

Contextualización avanzada

Para una configuración más avanzada se pueden hacer uso de Roles, para encapsular en esta clase de ficheros las funcionalidades especificas que queremos usar.
Por ejemplo, Un Rol WEB. Este necesitará un Cookbook Apache para poder dar servicio web, y posiblemente otro Cookbook MySQL para dar servicio de base de datos. Además si queremos tener un editor Vim, podemos introducir otro Cookbook Vim fuera de dicho Rol web.

Por tanto una configuración sencilla de dicho json sería:

node.json

```
{"run_list": ["role[web]", "recipe[vim]"]}
```

web.json

```
{
    "name": "web",
    "json_class": "Chef::Role",
    "run_list":["recipe[apache2]", "recipe[mysql]"]
    "chef_type": "role"
```

```
}
```

Si nos fijamos, tendríamos que saber que contiene el Rol “web”, por lo que tendríamos que “expandir” dicho Rol para saberlo.

Aún se podría complicar más la situación, ya que dentro de un Rol, puede haber más Roles, y se generaría un grafo de dependencias.

web.json

```
{
    "name": "web",
    "json_class": "Chef::Role",
    "run_list":["recipe[apache2]", "recipe[mysql]"], “role[blog]”
    "chef_type": "role"
}
```

blog.json

```
{
    "name": "blog",
    "json_class": "Chef::Role",
    "run_list":["recipe[wordpress]", "recipe[php]"],
    "chef_type": "role"
}
```

La expansión quedaría finalmente de esta forma:

```
 "run_list":["recipe[vim]"],"recipe[apache2]","recipe[mysql]",
     "recipe[wordpress]" "recipe[php]"]
```

Nuestro sistema es capaz de diferenciar estos dos tipos de contextualización, y expandirá o no dependiendo del tipo que sea.

7.3. Requisitos y Restricciones

Para su correcto funcionamiento, la herramienta necesita que se cumplan ciertos requisitos, que detallamos a continuación:

- Instalación previa de OpenNebula: La instalación de la herramienta se realizará sobre esta, integrándose con ella.
- Uso de templates base: El despliegue del entorno se realizará sobre una imagen base, que debe incluir la instalación mínima para la ejecución de las recetas, es decir Chef Solo y por tanto, Ruby. Esta imagen será incluida en lo que denominamos template base. Consultar Gestión de entornos/Componentes necesarios.

Por otro lado, aunque se ha intentado exprimir las capacidades de las herramientas todo lo posible, nos encontramos con algunas restricciones:

- OpenNebula: En principio, no debería causar ningún problema el uso de otras versiones mientras incluyan la característica de contextualización. Sin embargo, se recomienda usar la herramienta junto a versiones de OpenNebula de la 3.0 en adelante, al haberse realizado y probado sobre estas versiones.
- Chef Solo: Con la versión 0.10 de Chef se incluyeron numerosas mejoras. La herramienta solo funcionará correctamente con versiones 0.10 o superior, y se recomienda versiones superiores a la 0.10.4, a partir de la cual se incluyó el soporte de data bags.
- Ejecución como demonio: La herramienta no permitirá lanzar Chef Solo como demonio en el entorno de forma (Consultar Posibles Ampliaciones). El usuario podrá ejecutar el demonio desde el propio entorno una vez lanzado.
- Restricciones propias de Chef Solo: Existen diferentes restricciones en Chef Solo respecto de Chef Client, las cuales debemos tener en cuenta a la hora de escribir nuestras recetas y preparar nuestras configuraciones. Principalmente, no podremos hacer uso de los entornos Chef ni realizar búsquedas mediante el método *search*.
- Dependencias: En lo que refiere a nuestro desarrollo, la lectura de las dependencias entre recetas por parte de la herramienta, se realiza desde las propias recetas. Para el correcto funcionamiento de esta característica, deben aparecer en formato: `include_recipe "nombre_cookbook::nombre_receta"`, evitando el uso de variables y líneas múltiples.

7.4. Diseño

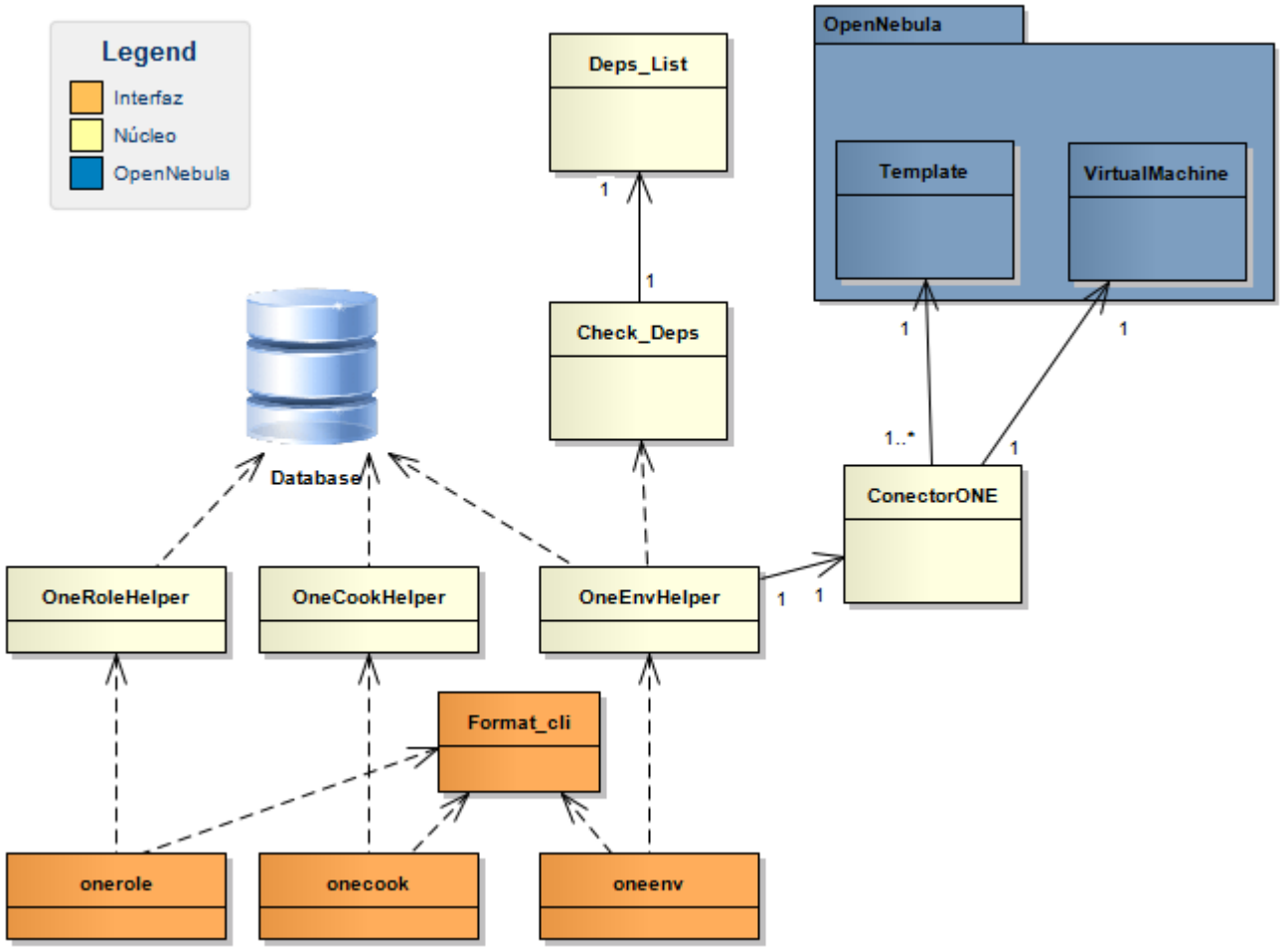
Para el desarrollo de la herramienta hemos escogido el lenguaje Ruby, no sólo por su sencillez y curva de aprendizaje, también por ser empleado tanto en Chef como en OpenNebula, y de esta forma facilitarnos su integración. Su distribución se realiza mediante un paquete comprimido, y tendrá como prerrequisito una instalación previa de OpenNebula.

Tras su instalación, la herramienta quedará integrada con OpenNebula, quedando accesible por línea de comandos gracias a los comandos o*neenv* y *onecook*.

A continuación se detallan los distintos aspectos en torno a la construcción de la herramienta y su funcionamiento:

7.4.1. Arquitectura

Podemos diferenciar tres partes en la arquitectura del sistema, atendiendo a su funcionalidad:



Interfaz

Actualmente la interacción del usuario con la herramienta se realiza siempre a través de la línea de comandos. Para su implementación nos apoyamos en la librería OptionParser, que nos permite la lectura de los comandos y opciones introducidos de una forma fácil y clara. Los scripts oneenv y onecook implementarán los objetos OptionParser, encargados de capturar y comprobar la validez de estos comandos.

En cuanto a la salida, la clase Format_cli, implementa algunos métodos para una correcta impresión por pantalla. Estos métodos nos permiten, entre otros, una salida correctamente formateada, o la impresión de cabeceras.

Núcleo

Se refiere a la aplicación en sí. Contiene a OneEnvHelper y OneCookHelper como componentes principales. Estas clases estáticas son los encargadas de la implementación de las instrucciones en la línea de comandos (ver Interfaz de Usuario), y de interactuar con la base de datos.

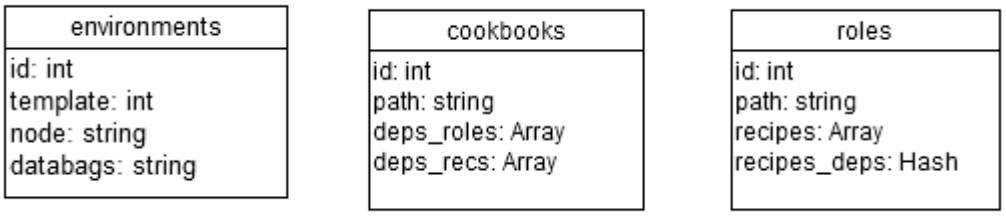
Para llevar a cabo estas acciones, se apoyan en diferentes clases detalladas brevemente a continuación:

- Check_Deps: Colección de métodos, empleados para la búsqueda y recorrido del árbol de dependencias.
- Deps_List: Estructura de datos para almacenar las dependencias actuales
- ConectorOne: Encargado de interactuar con la API XML-RPC de OpenNebula, y de generar la *template* correspondiente.

Base de Datos

La aplicación desarrollada se apoya y gestiona una base de datos sencilla implementada en SQLite. Para su manejo desde Ruby se ha empleado la implementación nativa de Active Record.

Gracias a la base de datos nos aseguramos la persistencia de estos y la inclusión de ciertas restricciones que ayudan a evitar incoherencias. Su estructura es la siguiente:



Constará por tanto de tres tablas independientes con información de los roles y los Cookbooks incluidos en el sistema, y los entornos instanciables. A continuación repasamos el significado de sus argumentos:

- environments
  - name: Nombre dado por el usuario al entorno. Este valor debe ser único.
  - template: Plantilla de OpenNebula que servirá de base para la instanciación del entorno.
  - node: Ruta hacia el archivo JSON con la información del nodo Chef.
  - databags: Ruta hacia los posibles data bags del entorno
- roles
  - name: Nombre del rol. Este valor debe ser único.
  - path: Ruta hacia el archivo rol.
  - deps_roles: Dependencias del rol con otros roles. Almacenado como un array con los nombres de estos.
  - deps_recs: Dependencias del rol con recetas. Almacenadas igualmente como un array, en el formato de Chef: “nombre_cookbook::nombre_receta”.
- cookbooks
  - name: Nombre del cookbook. Debe ser único.
  - path: Ruta hacia el directorio cookbook.
  - recipes: Lista de las recetas que contiene. Implementado como un array.
  - recipes_deps: Lista con las dependencias de cada receta del cookbook. Se almacena como un Hash, donde la clave es el nombre de la receta

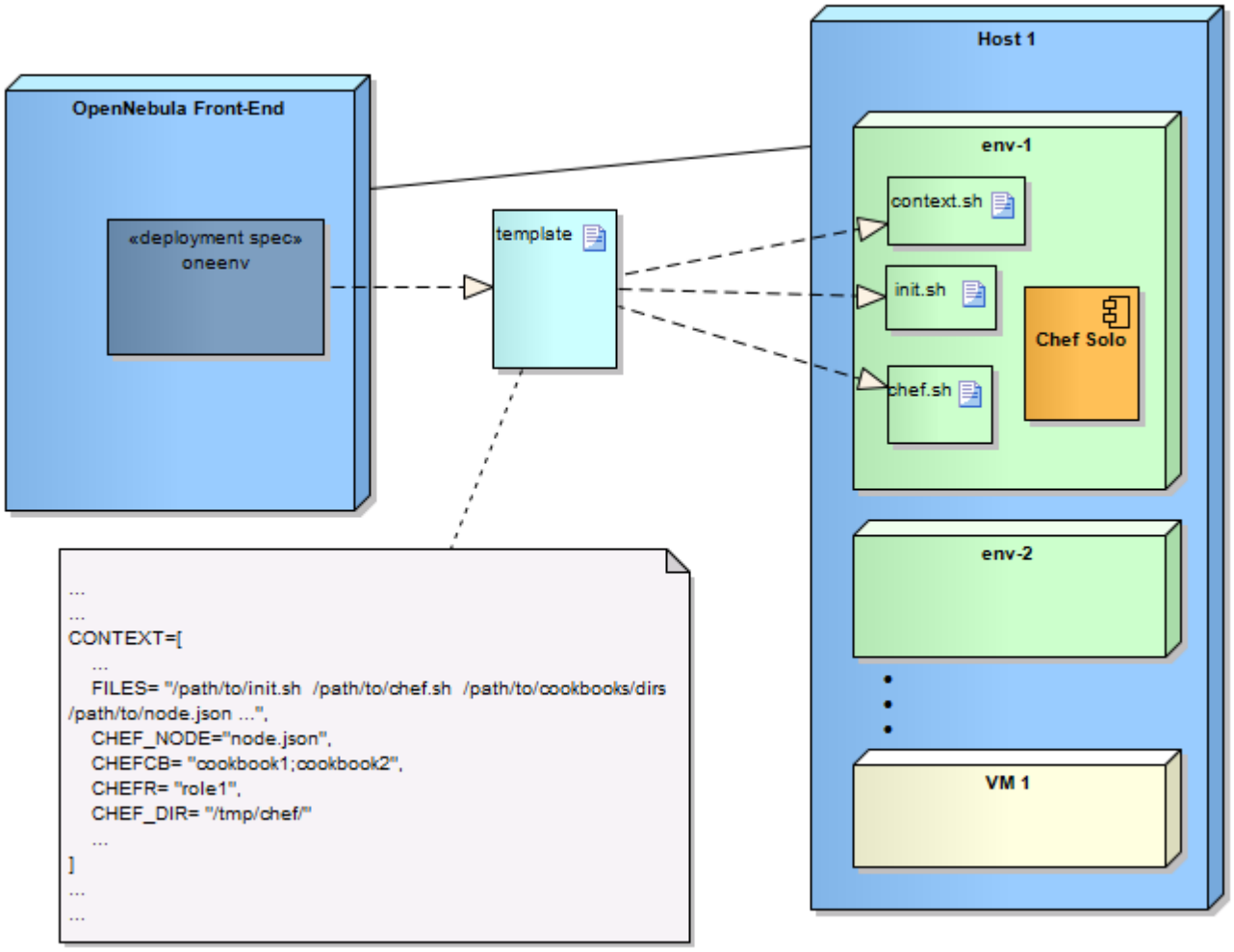
contenida en el cookbook, y el valor un array con todas sus dependencias en el formato de Chef: “nombre_cookbook::nombre_receta”.

7.4.2. Despliegue

Como hemos visto en capítulos anteriores, la herramienta se apoya en las herramientas de contextualización que proporciona OpenNebula. La imagen base que emplearemos quedará representada junto con los recursos de la máquina, por una *template* incluida en OpenNebula, en adelante hablaremos de *template base*.

En el momento de lanzar nuestro entorno, la herramienta *oneenv* construirá a partir de la *template base* asociada, una nueva *template* que instanciará en una nueva máquina virtual. Esta máquina quedará preparada para que sea el usuario quien, a través de OpenNebula, decida en qué *host* y en qué momento levantar el entorno.

Para su construcción, sólo se modificará el contexto, preservando todos los atributos que el usuario pudiera haber añadido en la *template base*. Por parte del entorno, se incluirán todos aquellos elementos necesarios para la ejecución de Chef, tales como cookbooks, roles... También estarán incluidos los scripts shell encargados de configurar e iniciar Chef al arrancar la máquina virtual: *init.sh* y *chef.sh*.



Para el correcto funcionamiento de estos scripts, *oneenv* incluirá una serie de variables en el contexto. El usuario no deberá hacer uso de estas variables en la template base, en caso de hacerlo serán sobrescritas. Estas variables son:

- CHEF_NODE: Nombre del archivo nodo.
- CHEFCB: Listado de las cookbooks incluidas en el lanzamiento del nuevo entorno.
- CHEFR: Listado de los roles incluidos en el lanzamiento del nuevo entorno.
- CHEF_DATABAGS: Nombre del directorio de databags incluido.
- CHEF_DIR: Ruta donde serán incluidos los distintos elementos y archivos de configuración de Chef Solo.

7.5. Interfaz de usuario

En la interfaz de usuario se trabaja con dos tipos distintos de funciones, las funciones sobre cookbooks y las funciones sobre Entornos, en esa segunda categoría quedan incluidas también las funciones sobre Roles, que para mayor claridad se mostrará como otro tipo distinto de función.

Es importante aclarar que tenemos tres tablas distintas, una para cookbooks, otra para Entornos y otra para Roles.

En cuanto a los argumentos, los que van entre corchetes son argumentos no obligatorios salvo en el caso en el que aparece el símbolo "|", caso en el que se debe usar un argumento u otro, es decir, uno y sólo uno. Por ejemplo:

[-n <COOKBOOK_NAME>]|[-i <COOKBOOK_ID>]

En este caso tendríamos que escribir **-n 'NOMBRE'**, o bien **-i 'ID'**.

7.5.1. Funciones sobre Cookbooks

Agregar Cookbook

- Agrega una nueva cookbook a la tabla y la añade al repositorio.
  - **Name:** Nombre de la nueva cookbook, debe de coincidir con el nombre de la carpeta donde se aloja.
  - *Path:* Path de donde se aloja el cookbook

```
$ onecook add <NAME>   [ -p <PATH> ]
```

Listar Cookbooks

- Muestra por pantalla todo cookbook presente en la tabla así como el número de recetas que contiene cada una de ellas.

```
$ onecook list
```

Importar Cookbook

- Importa todas las cookbooks de una ruta dada al repositorio cookbook y las añade a la tabla.

- **Path:** Dirección de la carpeta que contiene los cookbooks que queremos importar. En el caso que sea la misma que el repositorio, solo importa aquellos que no esté en la tabla de cookbooks.

```
$ onecook import-repo <PATH>
```

Borrar Cookbook

- Elimina una cookbook ya existente en la tabla.

```
$ onerole delete  [-i <ID_Role>]|[-n <NAME_Role>]
```

Mostrar información de una Cookbook

- Muestra nombre, ruta, recetas y dependencias (con otras cookbooks) de una cookbook ya existente en la tabla.

```
$ onecook show [-n <COOKBOOK_NAME>]|[-i <COOKBOOK_ID>]
```

Actualizar Cookbook

- Actualiza una cookbook ya existente en la tabla. Vuelve a introducir en la base de datos, las dependencias y recetas que contiene dicho cookbook.

```
$ onecook update-cb [-n <COOKBOOK_NAME>]|[-i COOKBOOK_ID>]
```

Comprobar las dependencias de una Cookbook

- Comprueba las dependencias de una cookbook ya perteneciente a la tabla.

```
$ onecook check [-n <COOKBOOK_NAME>]|[-i COOKBOOK_ID>]
```

7.5.2. Funciones sobre Entornos

Crear entorno

- Crea un nuevo entorno
  - **Name:** Nombre de la nueva cookbook, debe de coincidir con el nombre de la carpeta donde se aloja

- **ID_TEMPLATE:** Identificador de la template de OpenNebula
- **Node_Path:** Path del node que contiene la información necesaria para contextualizar.(Runlist, atributos..)
- *DataBag_path:* Path del databag que se usa en el node

```
$ oneenv create <NAME> <ID_TEMPLATE> <NODE_PATH> [-p <DATABAG_PATH>]
```

Listar entornos

- Lista todos los Entornos de la tabla, mostrando que nodes son usados y si contiene databags.

```
$ oneenv list
```

Clonar entorno

- Crea una copia idéntica de un Entorno que ya existía previamente en la tabla.

```
$ oneenv clone [-i <ID_Env>]|[-n <NAME_Env>]
```

Mostrar información de Entorno

- Muestra nombre, ruta del nodo, template y de un Entorno ya existente en la tabla.

```
$ oneenv show [-i <ID_Env>]|[-n <NAME_Env>]
```

Borrar entorno

- Elimina un elemento determinado de la tabla.

```
$ oneenv delete [-i <ID_Env>]|[-n <NAME_Env>]
```

Actualizar nodo de un entorno

- Cambia la ruta de un nodo que ya estaba previamente asociado a un entorno.

```
$ oneenv update-node [-i <ID_Env>]|[-n <NAME_Env>] [-p <NODE_PATH>]
```

Asociar databag a un entorno

- Dada una ruta, asocia el databag existente en ese emplazamiento a un entorno.

```
$ oneenv set-databags [-i <ID_Env>]|[-n <NAME_Env>] <DB_PATH>
```

Lanzar Entorno

- Levanta una maquina virtual con el entorno, que será alojada por el usuario en un host.
    - ChefPath: Indica el path donde se van a alojar los ficheros de contextualización en el entorno
    - Check: Flag que indica si queremos comprobar dependencias sobre las cookbooks a la hora de lanzarlo. En el caso que sea indicado, añadirá los cookbooks que esté en la tabla para solucionar las dependencias.

```
$ oneenv up [-i <ID_Env>]|[-n <NAME_Env>] [-p <CHEF_PATH>] [-c]
```

7.5.3. Funciones sobre Roles

Añadir rol

- Añade todos los roles que hay en la carpeta path al repositorio

```
$ onerole add <PATH>
```

Importa repositorio de roles

- Añade todos los roles que hay en la carpeta path al repositorio

```
$ onerole import-repo <PATH>
```

Listar Roles

- Muestra todos los Roles existentes en la tabla.

```
$ onerole list
```

Actualizar Rol

- Actualiza el contenido de un Rol en la base de datos si este es modificado, refleja la información del runlist en la base de datos, es decir actualiza los roles y recipes que contiene.

```
$ onerole update  [-i ]|[-n ]
```

Borrar Rol

- Borra un rol ya perteneciente a la tabla.

```
$ onerole delete  [-i <ID_Role>]|[-n <NAME_Role>]
```

Mostrar Rol

- Muestra un rol detalladamente

```
$ onerole delete  [-i <ID_Role>]|[-n <NAME_Role>]
```

8. Posibles Ampliaciones

Como hemos comprobado a lo largo de los capítulos, la herramienta *oneenv* es completamente funcional, y abre un enorme abanico de posibilidades en la provisión de entornos, gracias a la potencia de Chef. Sin embargo, el desarrollo y las distintas pruebas realizadas junto con la experiencia adquirida, nos invitan a sugerir algunas posibles ampliaciones de la herramienta, que quedan fuera del alcance del proyecto.

A continuación indicamos algunas de ellas:

8.1. Entornos Múltiples

En ocasiones podríamos querer definir ciertas restricciones entre entornos. Esta ampliación aportaría a la herramienta la capacidad de definir dependencias entre varios entornos ya definidos.

8.2. Chef como demonio

Tanto Chef Solo como Chef Client, permiten su ejecución como demonios. En este modo, Chef queda en segundo plano, ejecutándose cada cierto periodo de tiempo.

La ampliación propuesta integraría esta opción, el usuario podría escoger si desea que alguna o algunas de sus recetas se ejecuten cada cierto intervalo de tiempo seleccionado.

8.3. Soporte de lectura metadata.rb

En la interacción entre Chef Client y Chef Server, este emplea el archivo metadata.rb para administrar los cookbooks. Este archivo contiene información diversa acerca del cookbook al que pertenece.

Para conseguirlo, Chef transforma este archivo a formato JSON facilitando su lectura desde el código. Podría resultar interesante portar esta funcionalidad a la aplicación, lo que permitiría la lectura de ciertos campos con valor informativo para el usuario como versión o descripción del cookbook, o más importante, mejorar la comprobación de dependencias (ver apartado Restricciones).

8.4. *Plugin* para Sunstone

Sunstone es la interfaz web de OpenNebula, que simplifica las operaciones a realizar a la hora de gestionar OpenNebula. En las últimas versiones, Sunstone permite incluir plugins para extender su funcionalidad. Podríamos por tanto incluir una nueva pestaña “environments”, desde la que poder gestionar nuestra herramienta de provisión.

9. Glosario

- Active Record: Patrón de arquitectura, que permite manejar y realizar consultas frente a una base de datos relacional. En esta cada tabla y sus registros, son proyectados como objetos para ser manejados fácilmente por un lenguaje de programación. Estos objetos implementan métodos que permiten su creación y modificación.
- AMQP (*Advance Message Queuing Protocol*): Se trata de un protocolo de estándar abierto en la capa de aplicación. Se caracteriza por ser orientado a mensajes, encolamiento, enrutamiento, exactitud y seguridad. Permite que dos máquinas cualesquiera capaces de entender este formato, puedan interactuar entre sí, independientemente del lenguaje o sistema operativo.
- eRuby: Sistema de plantillas que permite empotrar código Ruby en un documento con texto plano, generalmente con extensión '.erb'. Es muy usado para generar código HTML, de forma similar a como lo hace PHP. Existen varias implementaciones, la empleada por Chef es erubis.
- DSL (*Domain Specific Language*): Se denomina a todo lenguaje de programación creado para ser empleado en un dominio específico, orientados a resolver un problema determinado. En contraposición se encontrarían los lenguajes de propósito general. Se suelen emplear cuando el lenguaje permite expresar el problema más fácilmente o con mayor claridad, y cuando la situación se repite lo suficiente. Chef y Puppet emplean los suyos propios, basados en Ruby.
- Imagen Base: Denominamos de esta forma, a toda imagen de OpenNebula con la instalación mínima necesaria para el despliegue del entorno en ella.
- Knife: Herramienta en línea de comandos de Chef, que permite, entre otros, interactuar con la API del servidor, acceder al repositorio, o realizar *bootstraping* de Chef.
- Ohai: Herramienta desarrollada por el equipo de Opscode, con el principal objetivo de servir a Chef. Su función es la de captar datos de interés de la máquina sobre la que está corriendo tales como memoria, CPU, plataforma... Estos datos son devueltos en formato JSON.
- REST (*Representational State Transfer*): Modelo de arquitectura software para sistemas, que tiene su más claro ejemplo en la World Wide Web. En nuestro contexto se refiere, al uso de una interfaz que usa las operaciones características de REST: Create, Read, Update y Delete.

- Virtualización: Es la creación a través de software de un recurso tecnológico, ya sea una plataforma Hardware, un sistema operativo, un dispositivo de almacenamiento u otro componente informático.
- Multitenencia: Es el uso compartido de recursos entre dos o más clientes.
- Máquina Virtual: Es un software que emula a una computadora y puede ejecutar programas como si fuese una computadora real.
- Template: Especificación de los recursos hardware que son usados por una máquina virtual.
- Template Base: Denominamos así a la template que incluye una imagen base, necesaria para el correcto funcionamiento de la herramienta.
- Metaentorno: Conjunto de entornos que funcionan coordinados, para prestar un servicio.
- OVF(Open Virtualization Format): Estándar abierto para empaquetar y distribuir servicios virtualizados o de forma más general software a ejecutar en máquinas virtuales.
- CMS(Content Management System): Programa que permite crear una estructura de soporte para la creación y administración de contenidos.

10. Anexo I: Ejemplos Prácticos

10.1. Servidor Web sencillo

En este primer ejemplo contextualizamos una máquina para que pueda dar servicio web. Usaremos contextualización sencilla, por lo que solo tendremos un cookbook Nginx que para dicho servicio.

Los componentes necesarios son:

Template OpenNebula:

```
CPU=1
DISK=[
  IMAGE_ID=9 ]
FEATURES=[
  ACPI=yes ]
GRAPHICS=[
  LISTEN=0.0.0.0,
  TYPE=vnc ]
MEMORY=512
NAME=my_ubuntu
NIC=[
  NETWORK_ID=17 ]
SO=[
  ARCH=x86_64 ]
TEMPLATE_ID=2
```

Node: Node.json

```
{"run_list": [ "recipe[nginx]"] }
```

Con nuestro gestor de entornos hacemos el resto:

Primero añadimos dicha cookbook al repositorio[8]

```
$ onecook add nginx
```

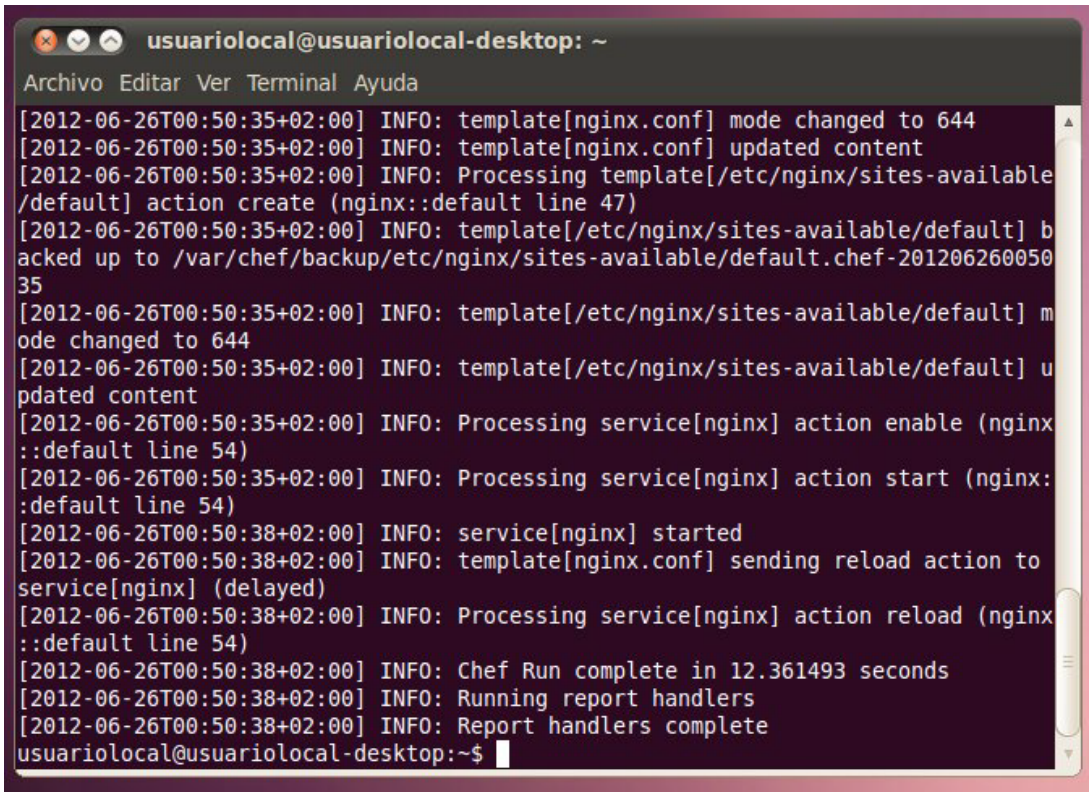
A continuación creamos un entorno indicando el node.json y la template con id=2
oneenv create web_sencillo 2 ~/nodes/node.json

[8] En este caso la cookbook está en la misma carpeta del repositorio, por lo que no hace falta indicar el PATH

Dado que dicha cookbook está autocompletada, no hacemos uso del Check dependencias y lanzamos con el comando UP:

```
$ oneenv up -n web_sencillo
```

El log creado a la hora de la instalación, refleja que se ha instalado correctamente el servidor web Nginx.



Si nos conectamos a la dirección IP de nuestra maquina virtual (en este caso 192.168.122.243), podremos ver el mensaje de bienvenida del servidor web nginx.

10.2. Cliente Samba y Vim

Para continuar demostrando ejemplos prácticos, ahora vamos a utilizar un role sencillo que nos hemos creado para poder instalar un cliente de ficheros Samba y un editor sencillo de texto.

Para ello tenemos el Node que contendrá el runlist de ejecución, con el role que queremos ejecutar:

Node: files.json

```
{"run_list": "role[file]"}
```

Por otro lado tenemos el Role que contendrá las cookbooks que usaremos, Samba y Vim.

```
{
    "name": "file",
    "json_class": "Chef::Role",
    "run_list":["recipe[samba]", "recipe[vim]"],
    "chef_type": "role"
}
```

Una vez tenemos estos componentes, creamos un entorno client_fich

```
$ oneenv create client_fich 2 ~/nodes/files.json
```

Añadir los cookbooks correspondientes, desde una carpeta que no está en nuestro repositorio, por lo que serán copiados al repositorio.

```
$  onecook add vim -p  ~/receta1/vim/
```

```
$ onecook add samba -p ~/receta2/cookbooks/samba/
```

Finalmente levantamos el entorno con el comando UP

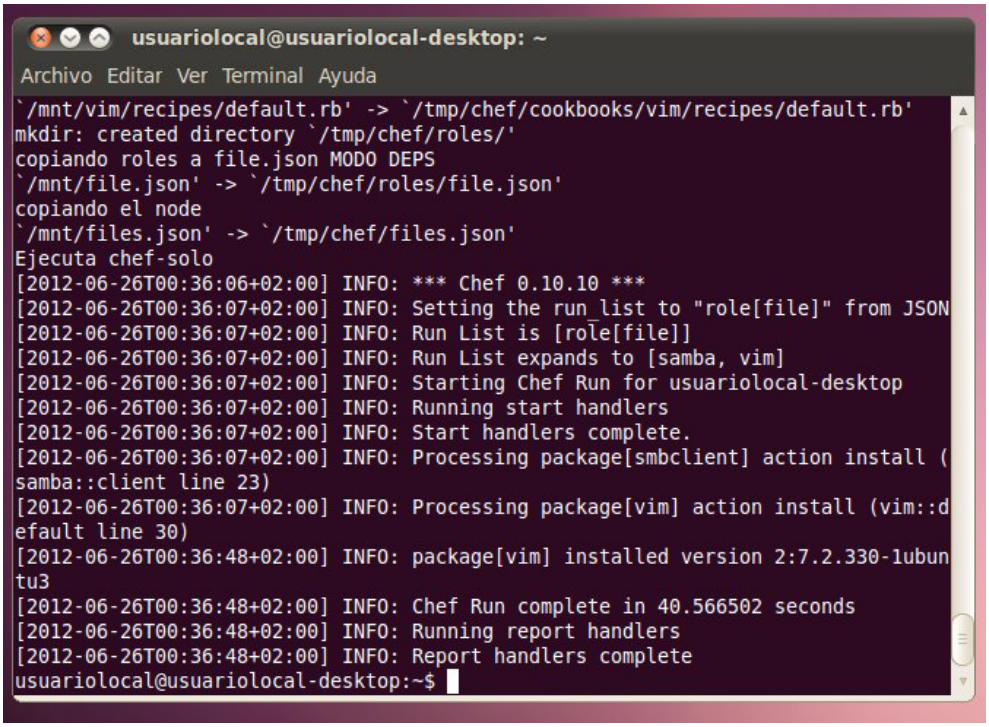
```
$ oneenv up -i 2
```

El proceso de instalación será el siguiente:
El node contiene un role “File”, el cual será expandido por nuestro sistema, para saber que cookbooks transferir a la máquina virtual, en este caso son Samba y Vim.

Estas serán copiadas a la máquina virtual, por medio de la contextualización genérica, que explicamos anteriormente, y se lanzará Chef-solo ejecutando el node files.json.

Una vez ejecutado el node files.json, Chef-solo expandirá el node, y buscará las cookbooks que hemos transferido nosotros por el Context.

Finalmente, este instalará cada cookbook, y tendremos contextualizado nuestro entorno.



10.3. Servidor Web con Wordpress

En el ejemplo anterior pudimos ver que se puede levantar fácilmente un servidor web con un cookbook. Este no tenía ningún tipo de dependencia con respecto a los demás, pero hay algunos cookbooks que si lo tienen, vamos a ver un ejemplo para el que existen ciertas dependencias de otras recipes de cookbooks y veremos cómo se resuelven.

Dado que un servidor web aloja algún tipo de página Web, nosotros optaremos por instalar un CMS de blogs bastante popular, WordPress. Así podremos tener nuestro propio blog en un entorno.

Los componentes serán, una vez más la template de OpenNebula, que será la anterior que hemos usado, y el node que contendrá el runlist.

Node: Web.json

```
{"run_list": ["recipe[wordpress]"]}
```

Creamos un nuevo entorno:

```
$ oneenv create blog 2 ~/nodes/web.json
```

Añadimos el cookbook Wordpress a nuestro sistema:

```
$ onecook add wordpress -p ~/cms/wordpress
```

En este caso, no tenemos nuestro cookbook en el repositorio, por lo que a modo de demostración vamos a añadirlo desde una carpeta externa. De modo que será copiado al repositorio, y añadido a la base de datos.

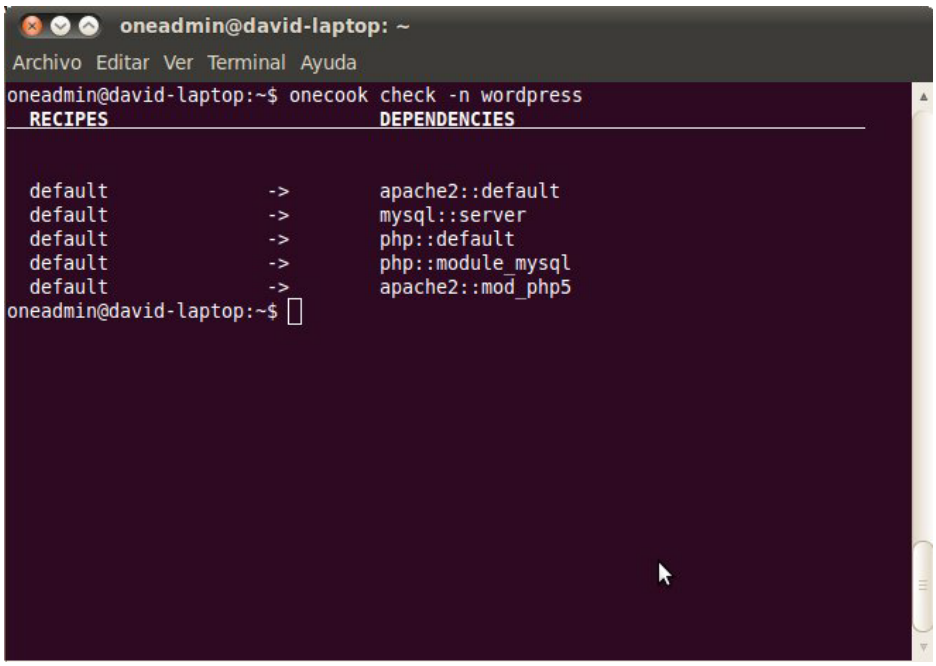
Como dijimos antes, tenía dependencias de otras recipes de cookbooks por lo que vamos a ver cuáles son los cookbooks de los que depende. Para ello hacemos uso del siguiente comando:

```
$ onecook check -n wordpress
```

Y obtendremos que necesita las recipes siguientes:

- apache2::default
- mysql::server
- php::default
- php::module_mysql
- apache2::mod_php5

Para poder completar las dependencias necesitamos los cookbooks Apache2, MySQL y PHP.

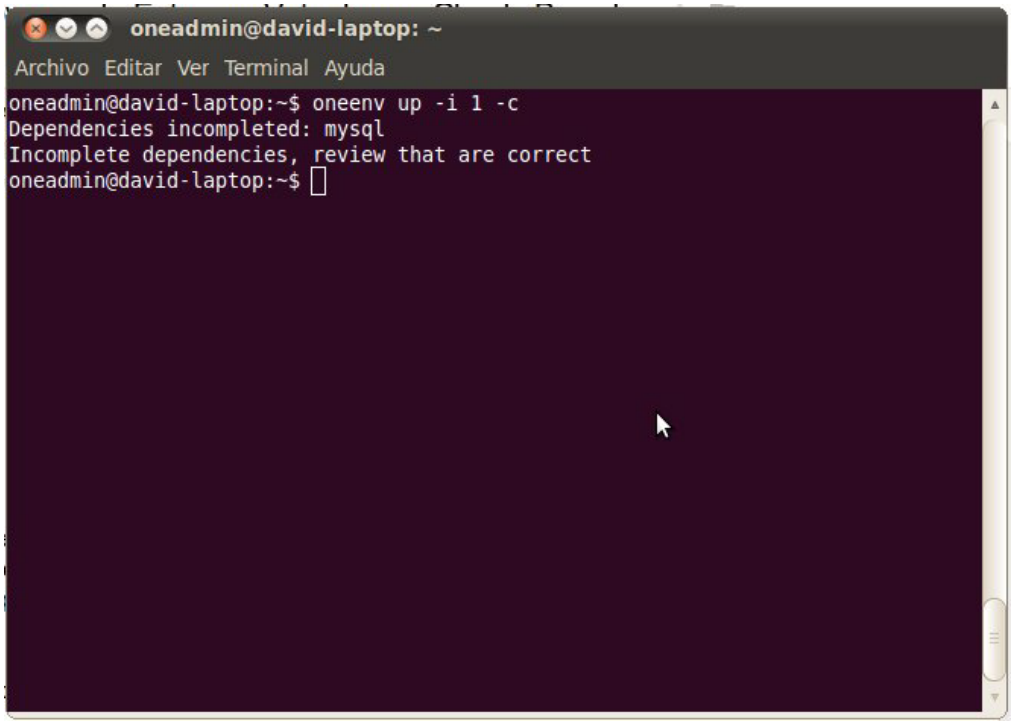


Finalmente con nuestro comando UP, podemos levantar dicho entorno, comprobando las dependencias, y en el caso que las necesite, buscará en la base de datos si tenemos cubiertas dichas dependencias.

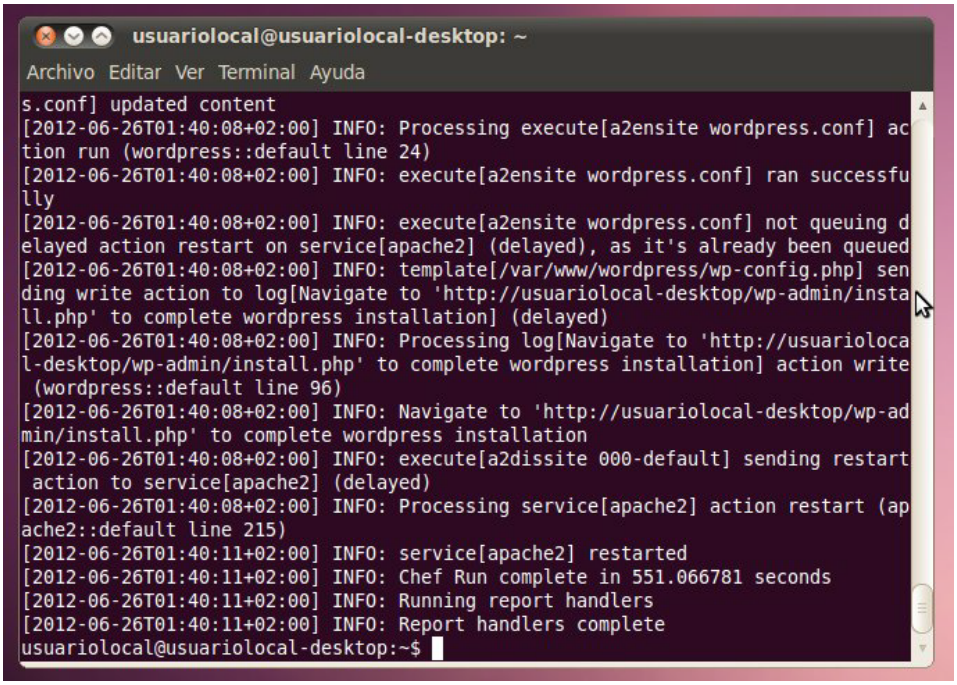
```
$ oneenv up -i 1 -c
```

En el caso que estén levantará dicha máquina virtual, con todos los cookbooks necesarios. Por el contrario si no tenemos dichas dependencias, nos informará que no las tenemos cubiertas, indicando cuáles faltan.

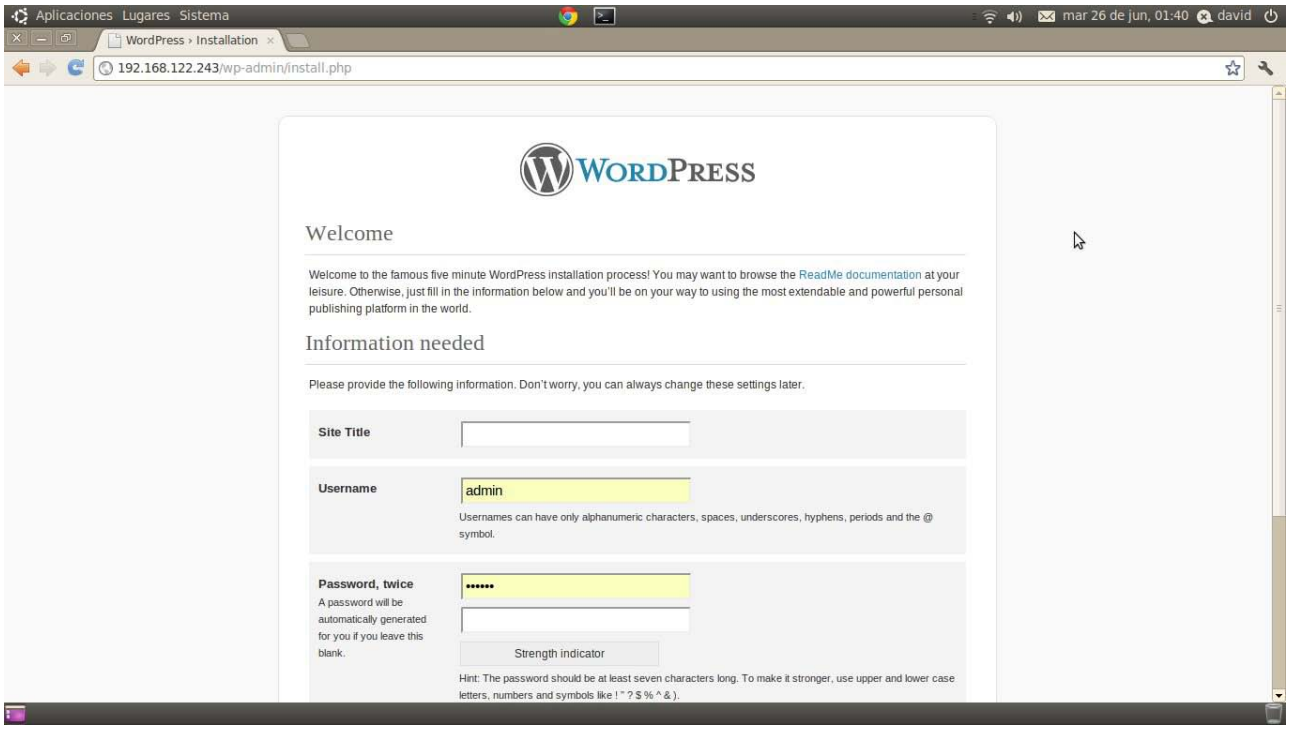
Veamos los posibles casos, para nuestro ejemplo.
En el primer caso, no tenemos la dependencia del cookbook MySQL, el sistema nos lo indicará.



El segundo caso, es que tengamos todos los cookbooks necesarios en nuestro repositorio, y con esto, los transfiera a la máquina virtual, como explicamos en el ejemplo anterior.



Para terminar, accedemos a nuestra máquina virtual que da dicho servicio web a través de la dirección IP con la que ha sido configurada. Al acceder vemos la página principal de Wordpress, con el que podremos administrar nuestro blog.



11. Bibliografía

[1] http://www.opennebula.org/
http://www.makeinstall.es/2011/04/virtualizar-con-la-maquina-virtual-del.html
[2] http://wiki.opscode.com/display/chef/Home
[3] http://www.ruby-lang.org/es/
http://www.developerfusion.com/article/84435/activerecord-for-ruby-and-rails/
http://tardigra.de/mcblog/2008/03/standalone-activerecord-and-sq.html
http://rubylearning.com/satishtalim/ruby_activerecord_and_mysql.html
http://snippets.dzone.com/posts/show/3097
http://www.ruby-doc.org/core-1.9.3
[4] http://madrid-devops.jottit.com/
http://allanfeid.com/content/cooking-chef
[5] http://agilesysadmin.net/chef-dependencies
https://github.com/opscode/chef
http://ruby.about.com/od/advancedruby/a/optionparser2.htm
http://ruby.about.com/od/advancedruby/a/optionparser.htm