Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de un sistema de configuración dinámica de OpenStack

Autor/es

Diego López Pérez

Director/es

Unai Arronategui Arribalzaga

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza

2014

RESUMEN: Diseño de un sistema de configuración dinámica OpenStack

El concepto de *cloud* estuvo de moda muchos años atrás, hablándose de él como el futuro, pero que ese futuro nunca llegaba. Hoy en día, se puede afirmar que es una realidad. El *cloud* es usado día a día por todos nosotros, incluso sin que nos demos cuenta.

Antiguamente, si una empresa quería montarse un servidor, debía comprárselo y mantenerlo físicamente, es decir, pagar una cantidad de dinero considerable en un servidor y además pagar el mantenimiento. Si se necesitaban más recursos, se debían comprar. En el momento que no se necesitaran, se malempleaban, dando lugar a servidores con mucho más rendimiento del que en verdad se usaba. Con el *cloud*, surge una nueva alternativa mucho más económica y atractiva: el usuario paga únicamente por los recursos que necesite, pudiendo de forma elástica demandar más recursos o menos, gracias a la virtualización.

El objetivo principal del TFG consiste en crear un entorno *cloud* privado tipo *IaaS* a través del proyecto OpenStack, utilizando máquinas virtuales como servidores y herramientas de despliegue y configuración automática para que el administrador despliegue de forma rápida y sencilla el entorno OpenStack que desee.

Los pasos que se realizaron durante todo el proyecto fueron los siguientes:

En un primer momento, se realizó un análisis de todo el entorno que se compone alrededor de OpenStack, desde la investigación del funcionamiento de sus componentes y sus posibles variaciones de implementación hasta las herramientas que ofertan diferentes empresas sobre su propia implementación y configuración OpenStack.

Tras comprender la funcionalidad y uso de cada servicio, se pasó a desplegar un entorno funcional OpenStack, teniendo en cuenta recursos mínimos necesarios de las máquinas virtuales donde se lanzarán los componentes OpenStack (RAM, disco duro, etc), para poder analizar de forma más exhaustiva una configuración en un entorno controlado. Se realizaron diferentes tipos de despliegues, desde poner todos los servicios en una misma máquina, hasta desplegar los diferentes servicios en un entorno distribuido.

Una vez se esté familiarizado con OpenStack y cómo se opera tanto a nivel de usuario como a nivel de administración, se pasó a analizar diferentes herramientas para el despliegue de los componentes y configuración automática de OpenStack. El cometido de dicha herramienta es el de asegurar tanto la instalación automática de los módulos en las máquinas virtuales correspondientes como el de realizar una configuración automática personalizada.

Tras ello, se realizó un diseño de la aplicación final, teniendo en cuenta decisiones de diseño tras los análisis realizados.

Más adelante se implementó a través de un lenguaje de alto nivel la interacción entre la herramienta que realiza el despliegue automático entre las diferentes máquinas virtuales, un fichero de configuración donde el administrador podrá editar de forma sencilla la configuración que desea desplegar, y las máquinas virtuales involucradas, con el objetivo de poder configurar un entorno mínimo funcional OpenStack.

Para finalizar, se desplegaron diferentes casos de uso, validando su correcto funcionamiento desde el punto de vista de un usuario final al que se le ofrezca nuestro servicio *cloud*.

Como conclusión, resaltar la complejidad que tuvo el realizar el despliegue, dado que se involucran muchos servicios distintos y además considerando que estamos en un entorno distribuido. La finalidad del proyecto trata entre otras cuestiones de eliminar esta complejidad a un administrador OpenStack, siendo consciente de lo que está desplegando (es decir, que siga teniendo el control de lo que hace).

Contenido

Introducción

El *cloud* hasta hace unos años era un concepto bastante ambiguo, el cual con el paso de los años se ha ido definiendo. En la actualidad, prácticamente de forma diaria usamos *cloud* y, muchas veces, sin darnos cuenta de ello. Todo esto oculta la problemática de los servicios asociados a dar soporte a esta infraestructura, que un usuario común no ve.

El problema principal es de alta complejidad debido a la gran cantidad de servicios interdependientes que hay que configurar, poner en marcha y comprobar su correcto funcionamiento, así como cambiar la configuración en marcha desactivando servicios y configurando nuevos. Se deberán tener en cuenta la gestión de aspectos distribuidos como autenticación, máquinas virtuales, almacenamiento básico y monitorización. Además, considerando aspectos de alta disponibilidad.

El objetivo que aborda el siguiente TFG es el de diseñar e implementar un sistema que permita realizar cambios rápidos y flexibles de configuraciones del sistema *cloud* OpenStack y de sus diferentes componentes de los que consta.

Toda la información tanto teórica como práctica tiene intención de servir de utilidad en el futuro en la preparación y docencia de la asignatura *Redes y Sistemas Distribuidos* del master y de la asignatura *Administración y Sistemas 2* del grado en ingeniería informática de la universidad de Zaragoza. Los alumnos harán uso de OpenStack tanto a nivel de usuario como a nivel de administración en las prácticas y en el trabajo personal de cada alumno, lo que tanto la documentación como la herramienta a realizar servirán de apoyo para lograr los objetivos docentes.

El resultado final es el de abstraer al administrador que quiera desplegar un entorno OpenStack de su complejidad interna, teniendo control en todo momento de los servicios que se están desplegando en su entorno.

Para el despliegue hemos tenido en cuenta la limitación de los recursos disponibles, ya que nos movemos en un entorno universitario donde los recursos son mucho más reducidos que en grandes empresas, luego siempre hemos tenido en consideración tanto memoria mínima necesaria para que funcione un entorno OpenStack, procesadores, almacenamiento y movernos en entornos distribuidos, con más de un computador. Para solventar este tipo de problemas y movernos en un entorno distribuido, se ha hecho uso de un entorno de máquinas virtuales, en el cual podemos controlar de forma virtualizada los recursos asociados a cada máquina virtual.

En el primer apartado del proyecto, hablaremos sobre los conceptos y tecnologías de forma general de los que trata este proyecto, donde aclararemos el concepto de *cloud*, sus tipos y hablaremos de OpenStack y sus componentes, así como detallar otros conceptos de interés.

Pasaremos en el siguiente apartado a hablar sobre el análisis, diseño e implementación que se ha originado en torno al proyecto, para concluir con verificaciones y pruebas, así como el resultado final.

Finalmente, obtendremos unas conclusiones en torno al proyecto y unas conclusiones generales y personales. Acabaremos hablando sobre el trabajo futuro sobre este proyecto, es decir, qué podríamos ampliar del proyecto.

En el "*Anexo 1: gestión del proyecto"* se detalla la gestión que se realizó a lo largo de este proyecto, así como las horas que se dedicaron a dicho TFG.

Conceptos y tecnologías

• Introducción al *Cloud*

El concepto de *Cloud*, *Cloud Computing* o la Nube se debe entender como un modelo rápido, escalable y de alta disponibilidad de proporcionar recursos tales como computación, almacenamiento, programas, etc.; es decir, recursos IT. Económicamente la principal ventaja es que el usuario paga únicamente por lo que realmente necesita y durante el tiempo que necesite el servicio que demanda. Por ello, se suele denominar también computación bajo demanda.

Se suelen diferenciar tres sectores básicos en el *cloud*: *SaaS, PaaS* e *IaaS*.

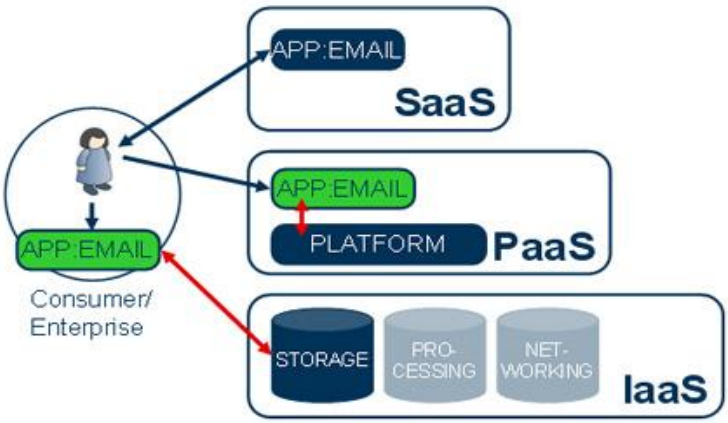

Figura 1: Sectores básicos del Cloud

SaaS: *Software as a service*, consiste en ofrecer aplicaciones al usuario accesibles a través de un navegador web, escritorio o cliente remoto. En la *figura 1* se aprecia cómo el usuario puede acceder directamente a la aplicación de correo electrónico, sin necesidad de pagar una licencia software o preocuparse del mantenimiento.

PaaS: *Platform as a service*, usado generalmente por usuarios desarrolladores, proporciona los elementos necesarios para el desarrollo y puesta en marcha de aplicaciones y servicios web. El proveedor ofrece la plataforma, con la principal ventaja de que el usuario no tiene que modificar el hardware cuando crezca, ya que eso se realiza de forma automática y transparente. En la *figura 1* se ve cómo ofrece la plataforma para que el usuario desarrolle sobre ella la aplicación de correo electrónico.

IaaS: *Infraestructure as a service*, ofrecer la infraestructura de computación como un servicio, haciendo uso normalmente de la virtualización. En vez de comprar servidores y todo lo que ello conlleva (equipamiento de redes, unidades de almacenamiento, etc.), el usuario compra (alquila) a proveedores externos dichos recursos. En la *figura 1* vemos cómo al usuario se le provee de recursos básicos tales como computación, almacenamiento y red, para poder crear por encima un servidor o conjuntos de servidores que ofrezcan el servicio de correo electrónico.

• *IaaS* y servicios asociados

En este proyecto nos centraremos en ofrecer un servicio *IaaS* al usuario final, así que vamos a pasar a entrar en detalle en los elementos que compone dicho *cloud*.

IaaS ofrece como servicios principales el de computación, almacenamiento (de imágenes de sistemas operativos y de bloques) y red. Otros pilares sobre los que se sustentan estos servicios son el de autenticación y normalmente una interfaz para operar de manera sencilla

sobre el *cloud*. A partir de aquí, se pueden añadir otro tipo de servicios que hacen más rico el servicio *IaaS* que se ofrece, como monitorización, gestión de bases de datos, almacenamiento de objetos, orquestación, etc. Todos los elementos deben estar fuertemente coordinados entre ellos, ya que la interrelación entre ellos es un hecho (como se puede apreciar en la *figura 2*).

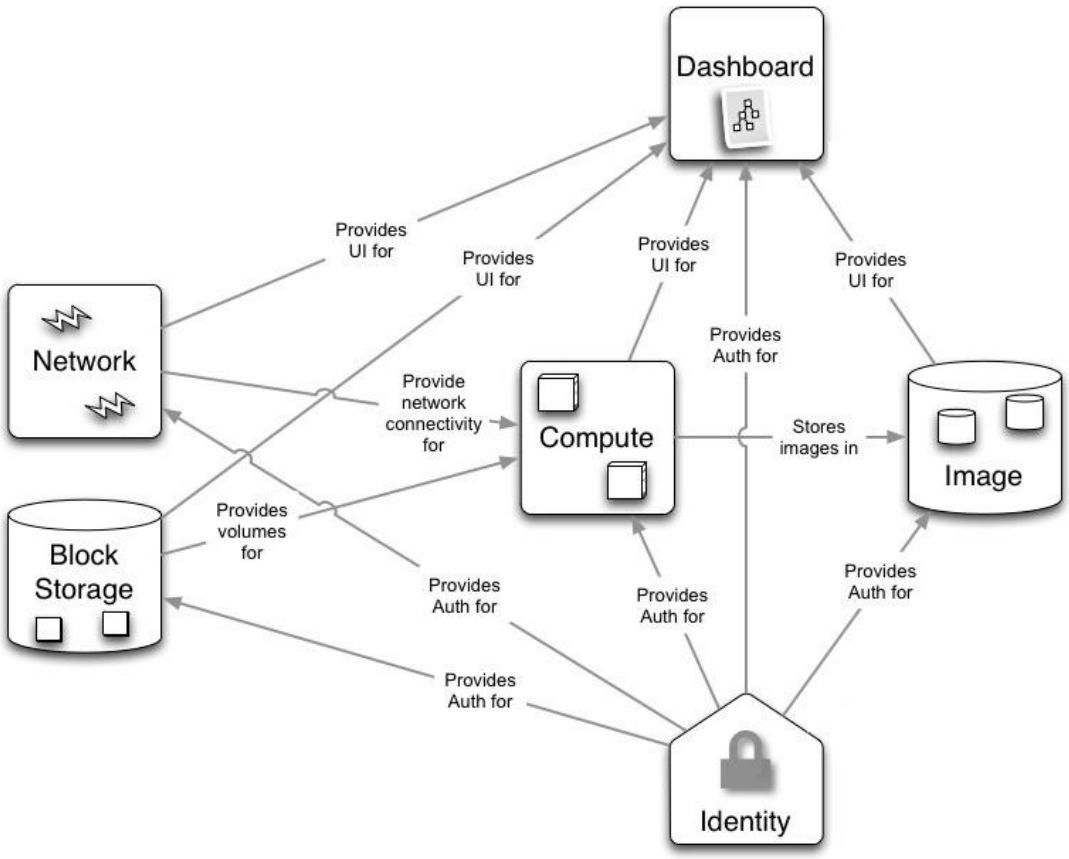

Figura 2: Componentes básicos en un entorno tipo IaaS

El servicio de computación permite ofrecer recursos de computación bajo demanda, mediante el aprovisionamiento y la gestión de grandes redes de máquinas virtuales. Estos recursos suelen ser accesibles a través de una API para desarrolladores y a través de interfaces web para administradores y usuarios.

El servicio de almacenamiento suele dividirse en dos tipos (*figura 3*): almacenamiento de bloques y de objetos. La principal diferencia entre ambas arquitecturas de almacenamiento es que el almacenamiento de objetos maneja los datos como objetos, es decir, cada objeto viene identificado por un ID único global, con una cantidad variable de metadatos y los datos en sí, mientras que el almacenamiento de bloques gestiona los datos como bloques dentro de los sectores y pistas de las unidades de almacenamiento, sobre el cual se definirá un tipo concreto de sistema de ficheros (ext3, ext4, fat32, etc.).

El almacenamiento de bloques se puede ver como un contenedor donde se definen los objetos a partir de un conjunto de bloques que tienen un tamaño fijo, sin separar datos y metadatos, mientras que el almacenamiento de objetos puede albergar objetos de diferentes tamaños, separando datos y metadatos. Como vemos en la imagen, el protocolo de interactuación es distinto si hablamos de almacenamiento de bloques o de objetos.

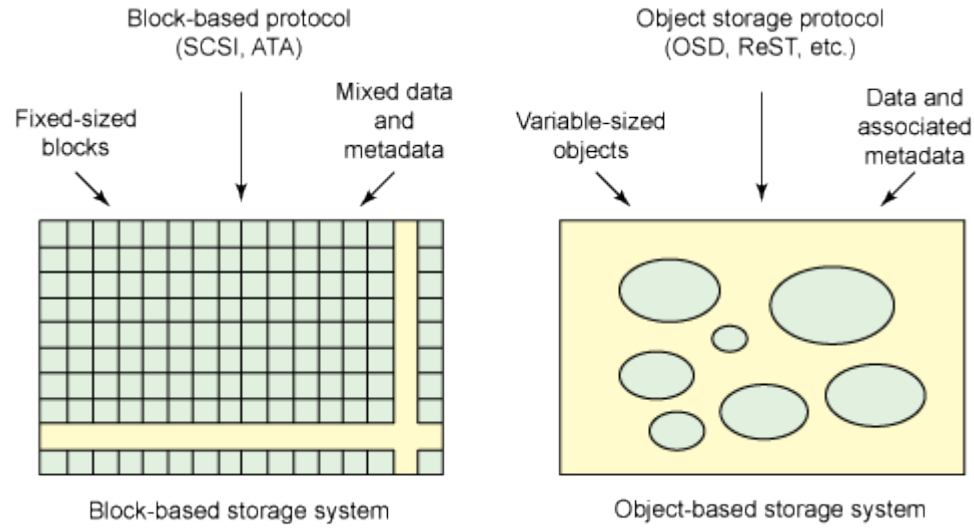

Figura 3: almacenamiento de bloques y de objetos

El servicio de red ofrece la posibilidad de manejar redes y direcciones IP de un conjunto de máquinas virtuales. Las técnicas tradicionales de gestión de redes no llegan a proporcionar un enfoque verdaderamente escalable y automatizado para la gestión de estas redes. Al mismo tiempo, se demanda más control y flexibilidad con aprovisionamiento más rápido. Hay que ser conscientes de la gran cantidad de máquinas virtuales que puede haber en un centro de datos, y la necesidad de disponer de este servicio que controle las redes, movilidad de las máquinas a otras subredes, y otro tipo de consideraciones como VLANs para separar el tráfico, la gestión de IPs de los DHCPs, creación de routers virtuales a petición del usuario para poder crear sus propias subredes dentro de su dominio específico, cortafuegos virtuales, VPNs, etc.

El resto de servicios que se ofrecen como básicos:

- El servicio de autenticación, tanto para usuarios y sus privilegios asociados, como para los servicios *cloud* que interactúan entre ellos.
- El servicio de dashboard, que ofrece una interfaz normalmente a través de un servidor web para acceder de forma rápida tanto a usuarios como administradores al *cloud*.
- El servicio de provisión de imágenes virtuales, en los cuales se almacenan snapshots de los diferentes sistemas operativos que se quieran dejar accesibles al usuario. Es decir, este servicio interactuará con el servicio de almacenamiento de objetos para almacenar las imágenes virtuales como objetos.

Existen diferentes soluciones para implementar un entorno *IaaS* con Open Source. Los más conocidos son OpenStack, Eucalyptus, OpenNebula y CloudStack. Todas las soluciones están pensadas para crear un *cloud* híbrido. Esto significa que en caso de que la infraestructura interna donde opere el *cloud* privado se vea limitado en recursos, rápidamente hará uso de los recursos necesarios en un *cloud* público. Por ejemplo, supongamos una empresa que tenga desplegado internamente un *cloud* privado con OpenStack, durante un periodo concreto de tiempo necesita más potencia computacional para hacer un determinado cálculo. Entonces OpenStack hará uso de la computación de otro *cloud* público (normalmente el más utilizado es Amazon). *Cloud* público en la actualidad es ofrecido por la mayoría de empresas más conocidas a nivel mundial en el sector de la informática: Amazon, Gloogle, Microsoft, etc. Siendo Amazon el líder indiscutible actualmente.

• OpenStack y servicios asociados

OpenStack es un proyecto de computación en la nube gratuito y de código libre dirigido por OpenStack Foundation para ofrecer *IaaS*. Consta de una arquitectura modular, con unos componentes interrelacionados que controlan procesamiento, almacenamiento y recursos de

red, administrables vía web o línea de comandos. Desde que apareció como un proyecto conjunto entre la NASA y RackSpace en 2010, se han desarrollado ya 9 *releases*. Los que se van a usan en este TFG son los *release* de Havana y Icehouse.

OpenStack Havana Release: versión de OpenStack lanzada el 17 de octubre de 2013, mejorando la versión anterior (Grizzly) y que consta de los siguientes componentes oficiales: Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer.

OpenStack Icehouse Release: versión de OpenStack lanzada el 17 de octubre de 2013, con la depuración de problemas de la anterior versión (Havana) y añadiendo el componente Trove.

En la siguiente tabla se muestran todos los servicios, su nombre y breve descripción:

Servicio	Nombre	Descripción
Dashboard	Horizon	Proporciona un portal web para interactuar con los servicios OpenStack, como el lanzamiento de una instancia, la asignación de direcciones IP y la configuración de los controles de acceso
Computación	Nova	Controla el ciclo de vida, puesta en marcha, decidir dónde se aloja y terminar una instancia.
Red	Neutron	Provee conectividad de red como servicio a otros servicios OpenStack, como por ejemplo a Nova. Ofrece una API para permitir a los usuarios definir redes y asociar instancias a dichas subredes.
Almacenamiento		
Almacenamiento de objetos	Swift	Almacena objetos de datos sin estructurar vía RESTful, API basada en HTTP. Tolerante a fallos gracias a la replicación de los datos. Su implementación no se basa en un servidor de archivos con montado de directorios.
Almacenamiento de bloques	Cinder	Provee almacenamiento persistente de bloques a las instancias que están corriendo. Facilita la creación y manejo de dispositivos de almacenamiento de bloques.
Servicios compartidos		
Servicio de identidad	Keystone	Provee el servicio autenticación y autorización para el resto de servicios OpenStack. También provee los *endpoints* de todos los servicios que operan.
Servicio de imágenes	Glance	Almacena y recupera imágenes de discos de máquinas virtuales. Nova hace uso de este servicio durante el aprovisionamiento para lanzar una instancia.
Telemetría	Ceilometer	.Monitoriza OpenStack por motivos de manufacturación, comparativas, escalabilidad y fines estadísticos.
Servicios de alto nivel		
Orquestación	Heat	Organiza el despliegue de aplicaciones, detallando por ejemplo el número de instancias sobre las que se va a lanzar, a través de plantillas de diferente tipo (formato HOT, formato de AWS, o formato propio de OpenStack).
Servicio de bases de datos	Trove	Provee bases de datos como servicio tanto para motores de bases de datos relacionales como no relacionales.

La interacción entre los diferentes componentes lo podemos ver en la *figura 4*:

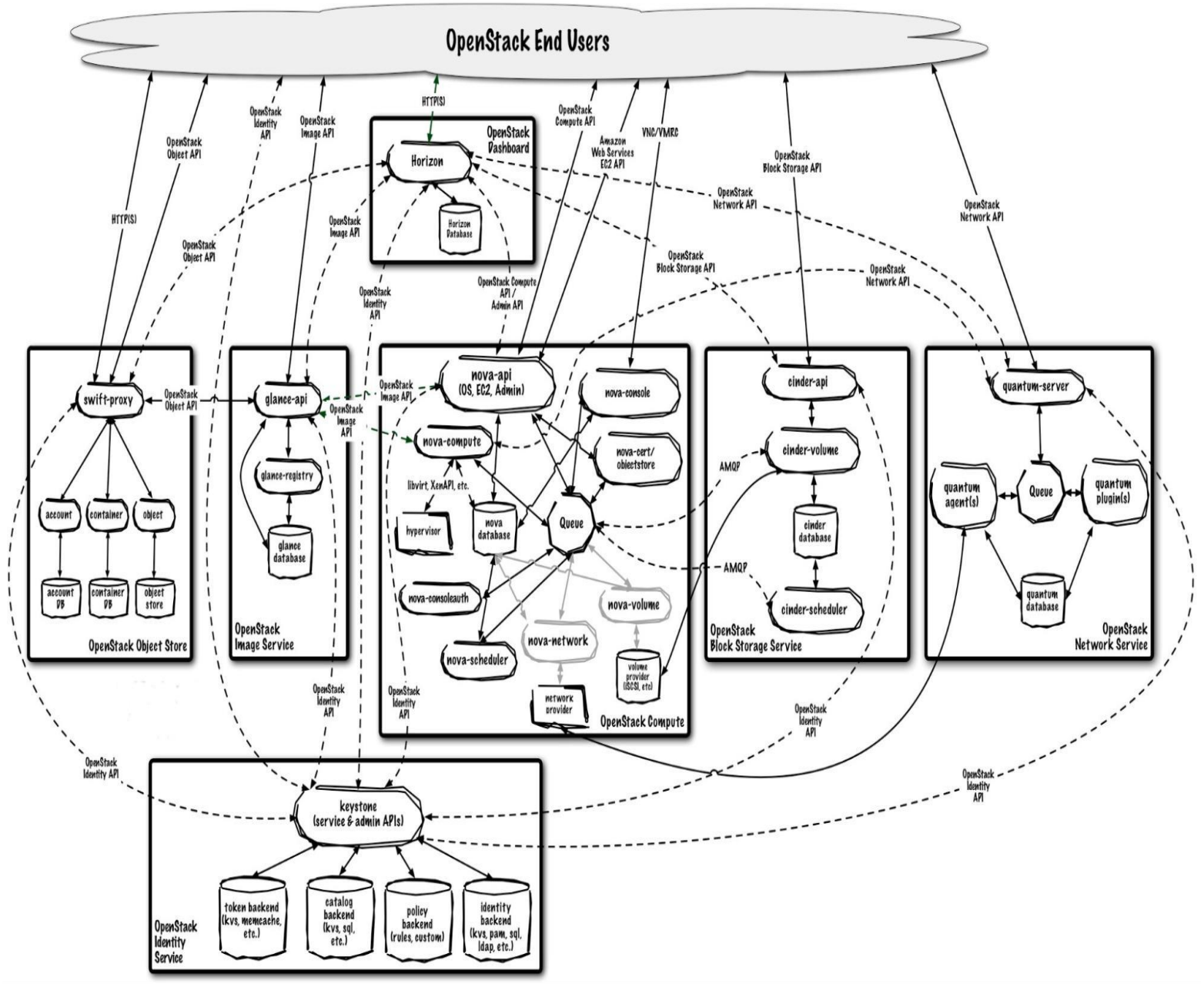

Figura 4: interacción y sub-servicios de los servicios OpenStack

En dicha imagen (*figura 4*) se aprecia la complejidad en la interacción de los servicios y la importancia que tiene tanto la comunicación como la sincronización de cada elemento. Como observación apuntar que no están en la imagen todos los servicios, ya que si estuviesen no sería si quiera entendible el esquema dada su alta complejidad.

Cada servicio OpenStack tiene asociado un *endpoint* (una URI) con el cual se puede comunicar cualquier otro tipo de servicio (ya sea un servicio OpenStack o un desarrollador desde el exterior, siempre que se autentique con sus credenciales de acceso). De hecho, cada servicio OpenStack tiene asociado un usuario y contraseña, como cualquier otro usuario (salvo que tiene privilegios de administración), que le permite interactuar con los servicios a través de la

API que ofrecen. Así pues, vemos la importancia que tiene el servicio Keystone, pilar básico en OpenStack y en cualquier entorno distribuido.

Se observa en la imagen como cada servicio OpenStack está compuesto por un conjunto de sub-servicios. Por ejemplo, Glance dispone de los sub-servicios glance-api (que acepta peticiones para descubrir/recuperar/almacenar imágenes), glance-registry (almacena/procesa/recupera meta información sobre imágenes de sistemas operativos, tales como tamaño, tipo, etc.) y la base de datos donde se almacena dicha información.

Dado que el despliegue de un entorno OpenStack ad-hoc no es trivial, muchas empresas ya se dedican a facilitar este despliegue y ofrecer monitorización de sus servicios. Entre las más conocidas están Mirantis, con su propia implementación denominada Fuel, RedHat, con RDO, DevStack con su Shell Script pensado para desarrolladores, Ubuntu Cloud, RackSpace, etc.

- ## Herramientas automáticas de configuración

Supongamos que queremos desplegar un entorno OpenStack. Con todo lo descrito anteriormente se prevé que sería un proceso arduo y complicado. Un administrador, aun teniendo conocimientos avanzados en informática, debería ir uno a uno a cada fichero de configuración de cada servicio, modificando los valores que crea necesarios. Esto se agrava cuando en lugar de un nodo, tenemos cientos de ellos. Para evitar tener que ir uno a uno modificando su configuración, contamos con herramientas de configuración automáticas.

Dichas herramientas sirven tanto para el despliegue de los componentes necesarios para su puesta en marcha como la modificación de su configuración de manera automática. El modo habitual de funcionamiento de este tipo de herramientas es que a través de uno o varios ficheros, definimos lo que queremos desplegar y cómo lo queremos desplegar (con qué tipo de configuración). En este tipo de herramientas contamos los más usados, tales como Puppet, Chef o Ansible.

Con esto se consigue abstraer al administrador de la modificación de los ficheros de configuración a mano, utilizando una herramienta por encima que ofrece un lenguaje de programación fácilmente leíble y entendible. Además, podemos gestionar y controlar la configuración de cada nodo desde un único computador, lo que facilita el mantenimiento de las configuraciones de cada máquina.

Existen scripts automáticos que realizan el despliegue como DevStack que tratan de hacer despliegues automáticos, como DevStack.

DevStack se define como un Shell Script para construir entornos de desarrollo OpenStack, mantenido y desarrollado por la comunidad.

Está pensando para desarrolladores, con lo cual siempre dispone de la última versión con la que la comunidad está desarrollando las futuras versiones OpenStack. Esto nos da la ventaja de poder conocer la nueva versión Icehouse antes de que salga el *release* oficial, pero también nos da la desventaja de que la versión está sin depurar y con la mínima documentación. No obstante, también se puede elegir la versión que queramos, es decir, *realease* estables. Dado que en nuestro caso la nueva versión de OpenStack estaba cerca, se decidió operar con la última versión que operaba DevStack.

No obstante, cabe recordar que Shell Script no es una herramienta pensada para automatizar configuraciones, aunque se pueda usar como tal. Además su nomenclatura es mucho más compleja, en comparación con la que nos ofrecen herramientas como Puppet o Ansible.

Análisis

El análisis de OpenStack es la parte más importante de este proyecto, debido a la cantidad de servicios que existen y de las implementaciones que subyacen en cada servicio. Conocerlos y entrar al detalle de cada componente es un proceso costoso, más aún cuando es la primera vez que se entra a analizar un sistema de estas dimensiones.

Se realizará un análisis *top-down*, es decir empezando desde arriba (puesta en marcha rápida a través de DevStack) y bajando los niveles hasta tocar a mano el más mínimo detalle (puesta en marcha manual).

Para ello, primero configuraremos el entorno sobre el cual se realizarán las pruebas del análisis y desplegaremos DevStack. Extraeremos resultados de la configuración que realiza y sobretodo tendremos un primer contacto con OpenStack. Posteriormente, seguiremos la guía de despliegue y configuración que ofrece OpenStack en su documentación oficial, a parte de otros documentos de ayuda, para desplegar manualmente OpenStack y ver qué alternativas ofrece.

• DevStack

Los pasos que se decidieron hacer fueron: primero crear un nodo *All-in-one*, esto es que un nodo tuviese todos los servicios OpenStack, sin depender de ningún nodo externo. Analizar a nivel de usuario las posibilidades que ofrecía y a nivel de administrador, para observar cómo se implementaba por debajo. Posteriormente, se añadirá un nuevo nodo, el de computación, y veríamos cómo se comportaba OpenStack en un entorno distribuido. También utilizaremos este nodo como un segundo nodo de almacenamiento Cinder.

DevStack consta de 3 scripts fundamentales más un fichero de configuración. El script *stack.sh* es el encargado de lanzar el despliegue OpenStack. Los parámetros necesarios de configuración los podemos definir en el fichero de configuración *local.conf*.. El script *rejoin-stack.sh* relanza los servicios OpenStack, en el caso de que se haya ejecutado anteriormente el script *stack.sh,* y por último está el script *clean.sh* que elimina por completo la anterior ejecución del script *stack.sh* (es decir, elimina todos los servicios OpenStack).

Tras el lanzamiento del script *stack.sh*, nos informará de la IP a la que conectarnos a través del navegador a Horizon y de los credenciales a usar. También nos facilitará el script *openrc.sh*, que consta de los credenciales para entrar a través de línea de comandos. Es decir, ejecutando el siguiente comando: *source openrc.sh*, definiremos en nuestra máquina las variables de entorno (tanto el *endpoint* de OpenStack, como el usuario, contraseña y *tenant*), así que podremos ejecutar desde el terminal, por ejemplo, *nova list* para listar las instancias que haya en ejecución.

En el "*Anexo 2: DevStack: Pruebas y ejemplos de errores y soluciones*" podemos ver algunas pruebas interesantes realizadas sobre este entorno, así como errores y soluciones surgidos.

○ Entorno para el análisis

Para crear nuestro entorno de pruebas, utilizamos dos máquinas virtuales, con el sistema operativo GNU/Linux Ubuntu 12.04 de 64bits (uno de los recomendados por Openstack). Como hipervisor, utilizaremos KVM (Kernel-based Virtual Machine, es una solución para implementar virtualización completa con Linux.Cada máquina virtual tiene su propio hardware virtualizado: una tarjeta de red, discos duros, tarjeta gráfica, etc.). Una de estas máquinas actuó como controlador (tiene todos los módulos de la versión Havana necesarios para operar individualmente), mientras que la otra máquina virtual se utilizó como nodo computacional (para dar apoyo a la computación al nodo controlador). Visto de forma gráfica, lo que tuvimos fue lo

siguiente (*figura 5*):

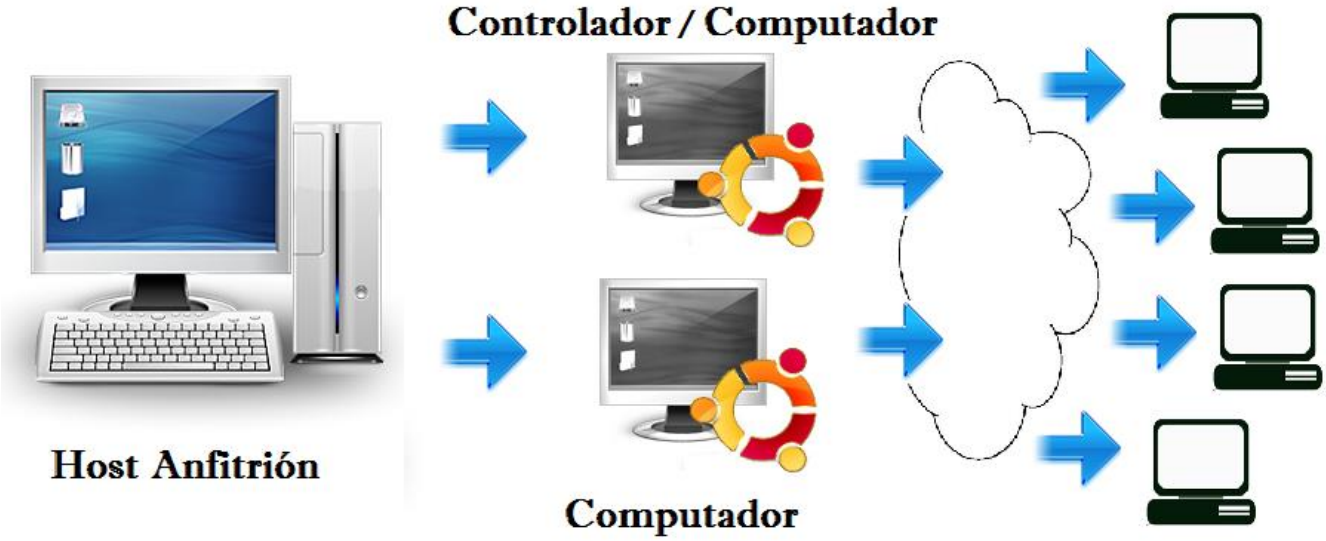

Figura 5: Esquema del entorno

○ Nodo All-in-One

El fichero de configuración que hay que editar se encuentra en el *"Anexo 2: DevStack: Fichero de configuración".* En él, podemos ver cómo se definen los servicios, los que queremos que estén activos y los que no, así como otros aspectos de configuración (contraseñas, IPs, etc.).

Hay que tener cuidado con la disposición de los elementos en DevStack, ya que si se lanza un servicio que depende de otro y este otro no está operativo, el script fallará.

Dado que es un fichero con una cantidad considerable de líneas de código Shell Script, un fallo en el fichero de configuración o en cualquier otro detalle (ya sea mala configuración de red, algo que no encaje con su forma de ejecutar, etc.) suele acabar fallando en su ejecución, luego no es trivial desplegar un entorno con todos los servicios OpenStack funcionando al 100%. De hecho, si no se configura nada en el fichero de configuración, hará un despliegue mínimo (componentes básicos). En mi caso, desplegué todos los componentes y comprobé que funcionaban.

○ Nodo Computacional

En la anterior instalación, sólo teníamos un nodo computacional, que fue el propio controlador.

Si quisiésemos añadir más máquinas físicas al *cloud*, DevStack nos proporciona una solución relativamente sencilla.

Primero deberemos clonar en la máquina que queramos añadir al *cloud* el proyecto de DevStack, tal y como hicimos en la máquina controladora. Tras esto, crearemos en el interior del proyecto (carpeta devstack) el fichero localrc (en teoría ahora el fichero se llamará local.conf para nuevas versiones, pero con localrc funciona correctamente) con los parámetros necesarios para decirle al nodo controlador que esta nueva máquina va a actuar como nodo computacional. Más información en "*Anexo 2: DevStack: Añadiendo Nodo Computacional*".

La ventaja que nos ofrece la virtualización es que en este punto, podemos copiar esta máquina virtual que actúa como nodo computacional y replicarla, teniendo tantos nodos computacionales como queramos (habrá que cambiar algún detalle, por ejemplo el nombre de la máquina para que no coincidan o las IPs para que sean distintas).

- ## Análisis en profundidad mediante el despliegue manual

En la segunda fase del análisis, pasamos a instalar todos los componentes de la versión que había en aquel momento (Havana) a mano, siguiendo el manual de instalación y configuración de la web oficial. Durante este proceso, aparecieron tanto la nueva versión de Ubuntu como la nueva versión de OpenStack, luego tuvimos que tomar la decisión de ver qué camino escoger. Dado que las versión nueva de Ubuntu (14.04) prometía estar más integrada con el *cloud* y dado que Ubuntu apostaba fuerte por OpenStack, decidimos que los despliegues pasarían a realizarse sobre máquinas Ubuntu 14.04 (anteriormente desplegamos DevStack en Ubuntu 12.04). En cuanto a la versión OpenStack Icehouse, aunque prometían ser una versión estable yendo de la mano de Ubuntu, en un primer momento consideramos demasiado pronto entrar a desplegar esta nueva versión (preferimos ver cómo se estabilizaba en el mercado y los comentarios de la comunidad). Se investigó sobre los nuevos cambios de la versión, obteniendo los siguientes resultados:

Solución de una gran cantidad de bugs de la versión anterior.
Cambio en la interfaz Horizon: más moderna y con más funcionalidad.
Poder lanzar volumen en una determinada zona (antes sólo dejaba hosts).
Extender volúmenes desde la interfaz.
Crear y manejar bases de datos (nuevo componente, Trove).
Agregaciones de host en zonas, posible desde la interfaz de Horizon.

Luego con esta información vemos que en principio no sería difícil migrar de la versión Havana a Icehouse (en una configuración mínima, no necesitaríamos Trove como servicio). Así que este despliegue manual se haría sobre Ubuntu 14.04 y OpenStack Havana. Más información de los cambios de la versión en el "*Anexo 3: Aspectos adicionales OpenStack: Cambios versión OpenStack Icehouse*".

En el "*Anexo 2: Instalación manual OpenStack Havana en Ubuntu 12.04*" detallamos los pasos que hay que hacer para la instalación de cada componente. Como se aprecia, no es nada trivial y requiere mucho tiempo y conocimiento de lo que se está haciendo.

- ## Análisis de productos similares en el mercado

Vamos a estudiar dos de los productos más relevantes en el mercado que ofrecen despliegues y configuración automática de entornos OpenStack: Mirantis y RDO. También se vio RackSpace, que ofrece *cloud* privado como servicio, para contemplar alta disponibilidad (detallado en el "*Anexo 2: RackSpace: alta disponibilidad*").

Mirantis

Mirantis nos ofrece una forma sencilla y rápida de desplegar de manera automática un entorno *cloud* basado en OpenStack, sin preocuparnos por los componentes de OpenStack necesarios que hay que instalar y configurar. Su *release* se denomina Fuel 4.0.

La configuración se despliega a partir del script *launch.sh.* Vemos cómo lanza 4 máquinas virtuales sobre VirtualBox (4 CentOS): Uno contendrá toda la configuración de Fuel 4.0, y las otras 3 serán nodos que usaremos posteriormente en el *cloud*.

Desde la interfaz web que nos ofrece Fuel, podemos crear de manera rápida y sencilla todo el entorno de OpenStack. Fuel nos identifica las máquinas virtuales que están corriendo en nuestra computadora, y podemos asociarles, entre otras cosas, si queremos configurar las máquinas como nodos controladores, nodos computacionales, nodos de almacenamiento, etc.

Podemos elegir la configuración de OpenStack que queremos:

- Ejecutar Openstack sobre máquinas CentOS 6.4, RHEL 6.4 o Ubuntu 12.04.
- Queremos HA (*High Abailability*) o no (implicaría más máquinas virtuales, 3 nodos controladores al menos).
- Elección del hipervisor (KVM o QEMU).
- Elección del controlador de red (nova-network o Neutrón).
- El almacenamiento de bloques Cinder sobre LVM e iSCSI (por defecto) o sobre Ceph.
- El almacenamiento de imágenes Glance sobre Swift o Ceph.
- Instalación de servicios adicionales: Savanna, Murano, Celiometer.

Con un par de "clicks", pulsando en "*Apply Changes*" y esperando, el entorno estará listo. Podremos acceder a la interfaz web de OpenStack (distinta de la interfaz web Fuel) vía la IP que nos proporcionará al finalizar la configuración.

Posteriormente, podremos hacer los cambios de configuración que consideremos oportunos, o bien mediante la interfaz de Fuel/OpenStack o bien sobre las propias máquinas virtuales (para ello se requieren más conocimientos sobre OpenStack que los que podría tener un administrador. Por ello cobran el mantenimiento).

Otros datos de interés:

El nodo controlador tiene 2GB de RAM y los usados como computacionales 1GB, luego podemos considerar que son requisitos mínimos para que OpenStack funcione relativamente sin problemas.

Por defecto instala una imagen CirrOS para lanzar instancias (tamaño = 14.1MB).

○ RDO

RDO está desarrollado por Red Hat, y también realiza despliegues de entornos OpenStack (en teoría en 15 minutos). Nosotros nos centraremos en los paquetes que utiliza para realizar dicho despliegue: PackStack + Python.

Básicamente PackStack es un conjunto de módulos Puppet para instalar a través de SSH en otras máquinas los componentes OpenStack que se necesiten.

Vemos cómo RDO define desde Python las variables necesarias de los manifiestos de Puppet. Las variables las recibe a partir de un fichero de configuración. Se las pasará como parámetros a las plantillas de Puppet.

Por ejemplo, vemos cómo se crearía un manifiesto Puppet para la definición de las credenciales de acceso (autenticación) de Nova en Keystone:

plugins/nova_300.py:

```
def createkeystonemanifest(config):
    manifestfile = "%s_keystone.pp"%controller.CONF['CONFIG_KEYSTONE_HOST']
    manifestdata = getManifestTemplate("keystone_nova.pp")
    appendManifestFile(manifestfile, manifestdata)
```

keystone_nova.pp:

```
class {"nova::keystone::auth":
    password    => "%(CONFIG_NOVA_KS_PW)s",
    public_address => "%(CONFIG_NOVA_API_HOST)s",
    admin_address => "%(CONFIG_NOVA_API_HOST)s",
    internal_address => "%(CONFIG_NOVA_API_HOST)s",
    cinder => true,
}
```

Utiliza una arquitectura Puppet del tipo cliente/servidor. Más información de este modelo en "*Anexo 2: Puppet: Arquitectura*".

- Resultado análisis

Gracias a los dos primeros análisis, hemos conseguido analizar la interacción de los servicios, directorios y ficheros a modificar, logs, los *backends* que se suelen utilizar y otros detalles dependiendo del servicio. Con el último análisis podemos obtener ideas de hacer despliegues automáticos y qué debemos tener en cuenta (cómo debe estar configurada la red, fallos que pueden surgir, etc.). Comentaremos únicamente los servicios que vamos a implementar en nuestro despliegue OpenStack.

-**Keystone**: lo primero que hay que distinguir es: usuario, rol y *tenant*. El *tenant* es el proyecto al que está asociado un usuario. El rol definirá los privilegios que tiene un usuario frente a un *tenant* asociado. Esto es, un usuario puede ser administrador de un *tenant* y puede ser usuario sin privilegios en otro *tenant*. Se pueden definir más roles, pero no se suele hacer (sólo se distingue por lo general al administrador del usuario común). Dicho usuario al validarse contra Keystone recibirá un *token*, que usará para posteriores autenticaciones.

El *backend* común es utilizar una base de datos donde se almacenará esta información. Se puede usar cualquier relacional (MySQL, PostgreSQL, etc.). Por defecto, en la propia instalación se instala una base de datos MSQLite, pero lo normal es que se use MySQL. También se puede implementar sobre otros backends más potentes y seguros, como por ejemplo LDAP (almacena usuarios y *tenants* en sub-árboles, y los roles como entradas de los usuarios) o PAM (usa el servicio PAM del sistema).

Los ficheros de configuración se suelen encontrar en /etc/keystone/, siendo keystone.conf el fichero principal de configuración.

-**Glance**: Utilizará, como todos los servicios, una base de datos relacional propia para almacenar la información necesaria.

Los *backends* sobre los que se suele sustentar Glance son: el propio sistema de ficheros de la máquina donde se instala Glance, que es el más utilizado en un entorno básico ya que no requiere software adicional. Utilizar como *backend* las infraestructuras que proveen tanto Swift como Cinder, incluso servicios de *cloud* público, como puede ser el S3 de Amazon (almacenamiento que ofrece Amazon. Si se usara este servicio ya pasaría a ser *cloud* híbrido). Otros: Ceph rbd, sheepdog**,** vsphere.

Los ficheros de configuración se encuentran en /etc/glance, siendo glance-api.conf el fichero de configuración principal.

-**Nova**: Es el núcleo central de OpenStack. Desde el CLI (*command line interface*) de Nova podemos controlar el resto de servicios. Es decir, la API de Nova tiene llamadas al resto de APIs, pero sólo a las necesidades básicas que provee cada servicio (no temas específicos). Por ejemplo, podremos llamar a "network-list" y este a su vez llamará a Neutron, pero no podemos crear un router virtual desde Nova.

La interacción entre Nova y el resto de servicios se ve de forma clara en la *figura 6*:

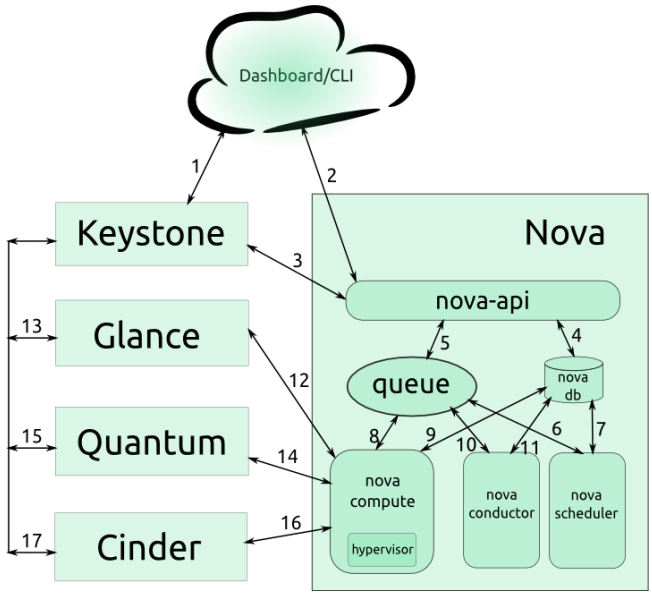

Figura 6: Interacción Nova - Servicios - Subservicios

Primero el usuario se valida desde el Dashboard o el CLI en Keystone, para posteriormente interactuar con la api de Nova. A su vez, Nova también se identifica frente a Keystone para operar con el resto de servicios. Los subservicios se comunican entre ellos a través de un gestor de mensajes (*queue*) que utiliza el protocolo AMQP (Advanced Message Queuing Protocol) y que puede estar implementada con distintos soportes: el comúnmente utilizado es RabbitMQ, pero también se da opción a implementar Qpid o Zero MQ. Más información en el "*Anexo 3: Aspectos adicionales OpenStack: AMQP*".

Nova-scheduler recoge de la cola de mensajes los mensajes de petición de instanciación de una nueva máquina virtual y determina en qué nodo deberá ser lanzado (teniendo en cuenta los recursos de la máquina host).

Nova-compute es el *core* de Nova. Utiliza la API del hipervisor correspondiente para crear/eliminar instancias (máquinas virtuales). Existe una amplia gama de hipervisores compatibles con OpenStack, pero los más utilizados son KVM y XenServer/XCP. Las APIs para el manejo de estos hipervisores son: XenAPI para XenServer/XCP, libvirt para KVM o QEMU, VMwareAPI para VMware, etc. En el "*Anexo 3: Aspectos adicionales OpenStack: Lista Hipervisores*" se listan otros hipervisores que se pueden usar.

Nova conductor simplemente es utilizado para mediar entre el acceso de la base de datos y nova-compute, es decir, evita el acceso directo de los nodos nova computacionles a la base de datos, gestionándolo adecuadamente.

Utiliza una base de datos relacional (normalmente MySQL) para almacenar la información relevante a Nova.

El directorio se aloja en /etc/nova, siendo nova.conf el fichero central de configuración, donde se detallará el acceso a Keystone y con qué credenciales se valida, end-point de acceso de Glance y puerto, acceso al servicio AMQP, acceso a Neutron, etc.

-**Neutron**: el servicio de red es el más problemático de poner en marcha, dada su elevada complejidad. Hasta la versión de Havana, existían dos formas de configurar la red: a través de Nova y eliminando Neutron: nova-network, o únicamente a través de Neutron. En la versión de

Icehouse, nova-network se considera obsoleto, luego nuestra implementación se basa en Neutron.

En OpenStack se suelen diferenciar varias redes:

- Red pública: comunica el cluster con el exterior.
- Red flotante: comunica las máquinas virtuales (instancias de un usuario) al exterior.
- Red de almacenamiento: separa el tráfico de almacenamiento de otras comunicaciones internas.
- Red de gestión: para comunicaciones internas de BD, AMQP,…
- Redes privadas: para comunicación entre *tenants*.

Aunque para nosotros, con diferenciar red pública (pública + flotante) de la red interna (almacenamiento + gestión + privadas) es suficiente.

Existen a su vez diferentes formas de gestionar la red privada:

Nova-network: 3 formas distintas: FlatManager, FlatDHCPManagery VLANManager. Dado que es obsoleto, pondremos la información en el "*Anexo 3: Aspectos adicionales OpenStack: Nova-Network"*.

Neutron: utiliza Open VSwitch (OVS) o cables de ethernet virtuales de Linux (linuxbridge) como *backend* para implementar la arquitectura de red que desee el usuario. Se usa OVS dado que presenta más funcionalidad que linuxbridge. OVS consta de bridges y puertos. Los puertos representan conexiones con otras cosas, como intefaces físicas o cables. Los paquetes de cualquier puerto de un bridge son compartidos a los demás puertos de ese bridge. Más información en el "*Anexo 3: Aspectos adicionales OpenStack: Open VSwitch*".

En la nueva versión Icehouse considera utilizar ML2 (*Modular Layer 2*) como *backend*. Se habla como ML2 como el futuro, dejando obsoleto el simple uso de OVS o linuxbridge. Es decir, lo recomendado en Icehouse es utilizar Open vSwitch mediante el nuevo plugin ML2.

Se puede escoger entre dos tecnologías principales: GRE o VLAN.

- GRE (Generic Routing Encapsulation): encapsula el paquete IP para cambiar la información de encaminamiento. Neutron crea para ello túneles GRE. Estos túneles son puertos de un bridge, y permite ver a los bridges de sistemas diferentes como un único bridge. Es lo que se usa en muchos VPNs.
- VLANs (Virtual LANs): Se añaden 4 bytes de etiquetado VLAN. Los paquetes etiquetados para una VLANs son sólamente compartidos con otros dispositivos configurados para estar en esa VLAN.

Los ficheros de configuración los podemos encontrar en la ruta /etc/neutron, siendo neutron.conf el fichero central de configuración.

-**Horizon**: vemos que se aloja en un servidor Apache, usando Django como servidor de aplicaciones web. Django es un framework web de código abierto escrito en Python que permite construir aplicaciones web.

Como dato, se debe tener instalado Keystone, Glance y Nova para poder instalar Horizon correctamente. El resto de proyectos los enganchará en la interfaz tanto antes de instalar Horizon como después.

-**Cinder**: El backend básico es utilizar LVM (*Logical Volume Manager*) de Linux para proveer almacenamiento de bloque persistente a las instancias sobre el protocolo iSCSI. Existen soluciones más completas y complejas que están apareciendo con relativa rapidez y están estandarizándose como *backend* de Cinder. Entre ellas cabe destacar Ceph y GlusterFS, que

intrínsecamente aportan alta disponibilidad entre otras prestaciones. DevStack usa LVM, mientras que Mirantis Fuel permite implementar Ceph. Más detalle en el "*Anexo 3: Aspectos adicionales OpenStack: iSCSI, LVM, Ceph*"

Como dato interesante, Cinder sólo deja asociar un volumen a una instancia (no permite compartir un volumen entre varias instancias).

El directorio principal de Cinder se hubica en /etc/cinder, siendo cinder.conf el fichero de configuración principal.

Diseño

La aplicación que vamos a diseñar constará de una estructura organizada que contendrá un elemento suministrado al administrador para configurar su despliegue, otro elemento encargado de iniciar el despliegue y directorios para almacenar el código y compilados necesarios. A partir del lanzamiento del despliegue y teniendo en cuenta la configuración que desea el administrador, se realizarán una serie de operaciones que den como resultado final la configuración de cada nodo OpenStack. Vamos a detallar estas cuestiones en los siguientes apartados.

• Flujo de despliegue

A continuación detallaremos el flujo de despliegue que seguirá nuestra aplicación una vez ejecutada. Podemos verlo gráficamente en la *figura 7*:

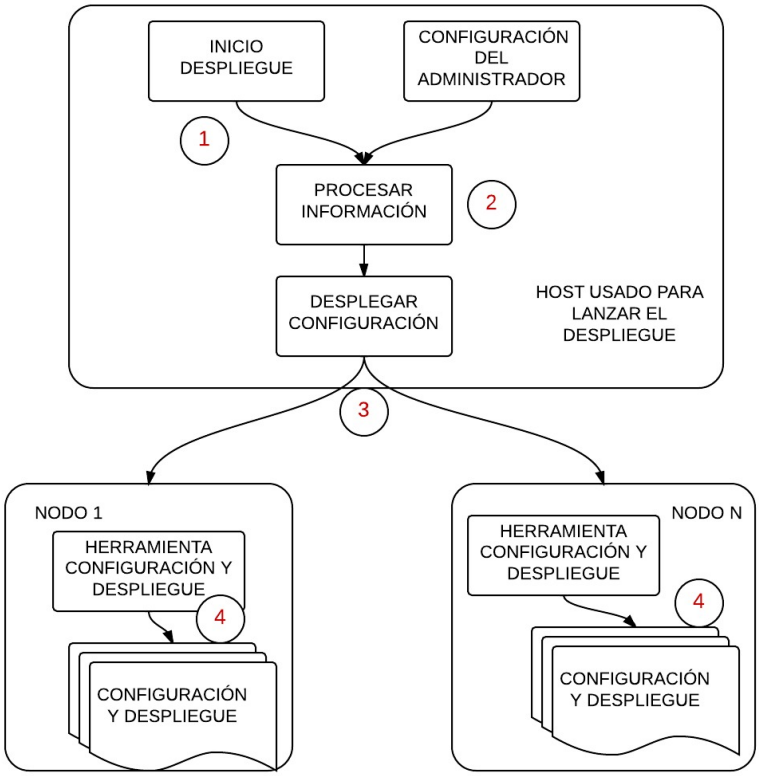

Figura 7: Mapa despliegue alto nivel

En primer lugar, el administrador deberá definir la configuración que desea desplegar en sus máquinas. A partir de aquí, iniciará el despliegue (1). La información de la configuración será procesada en el siguiente estado de la aplicación (2), pasando a desplegar los servicios OpenStack que haya definido el administrador en su configuración en la máquina que corresponda (3). La herramienta de configuración y despliegue se encargará en cada nodo de desplegar el servicio concreto y de asegurar que cumple con la configuración deseada (4).

La configuración del administrador deberá tener detallado aspectos tales como los nodos donde se debe aplicar el despliegue, qué componentes se deberían desplegar, acceso a la máquina, etc.

Cada máquina deberá tener un motor interno que se encargue de desplegar y configurar en su nodo el servicio (o los servicios) que le suministre el host usado para lanzar el despliegue.

Supondremos un ejemplo real para profundizar en el contexto de nuestro diseño. Vamos a suponer que el administrador quiere instalar el servicio Keystone, Glance, Horizon, Nova Controller, Neutron y Cinder Controller en un nodo controlador, con IP = 192.168.1.1. Además, querrá instalar en un segundo nodo con IP = 192.168.1.2 un servicio de computación, es decir, Nova Computer, y en un tercer nodo con IP = 192.168.1.3 querrá instalar Cinder Computer, dado que esta tercera máquina sólo la quiere usar para almacenaje.

El administrador insertaría los datos necesarios en la configuración, es decir, algo tal que así:

Keystone = 192.168.1.1
Glance = 192.168.1.1
Horizon = 192.168.1.1
Nova (Controller) = 192.168.1.1
Neutron = 192.168.1.1
Cinder (Controller) = 192.168.1.1
Nova (Computer) = 192.168.1.2
Cinder (Computer) = 192.168.1.3

Como vemos, el administrador sólo suministra la información básica y necesaria, teniendo control absoluto sobre qué se despliega. El administrador no se preocupa de tocar configuraciones por debajo, ya que de eso se encargará la aplicación, añadiendo un nivel de abstracción.

En este punto, la misión del administrador ha terminado. A partir de ahora, se encargará de automatizar todas las tareas la aplicación, apoyándose de la herramienta de despliegue y configuración.

En un primer momento, la aplicación parseará la información que le ha suministrado el administrador, y deberá crear la configuración final. Es decir, a través de que el usuario quiera una configuración “Keystone”, ya sabrá otros parámetros de configuración necesarios, por ejemplo el *backend* que utilizará (en este caso sería una base de datos relacional). Siguiendo el ejemplo, podríamos tener algo tal que así:

Enviar al Nodo 1 (192.168.1.1)	Nodo 2 (192.168.1.2)	Nodo 3 (192.168.1.3)
Config Keystone	*Config Nova Comp.*	*Config Cinder Comp.*
Config Glance		
Config Horizon		
…		

Esta configuración será pasada a la máquina correspondiente. A partir de esto, el motor de la herramienta de despliegue se encargará de procesar esta información y actuar en consecuencia.

Al finalizar, se deberá mostrar al administrador el correcto éxito del despliegue, así como la posterior supervisión, en caso de que lo necesite, de dicho despliegue (mediante un sistema que guarde el historial de lo que se ha ejecutado).

Implementación

Una vez definido el diseño, pasaremos a implementarlo. Para ello, en primer lugar definiremos las tecnologías que vamos a utilizar para la implementación, así como la explicación posterior del flujo de despliegue y cómo se interrelacionan los elementos del proyecto con el máximo nivel de detalle. Por último definiremos el esquema de directorios final del proyecto. El código implementado se puede encontrar en el "*Anexo 5: Código implementado*".

• Selección de tecnologías

Lo que en diseño hemos comentado como el elemento que inicia el despliegue, lo vamos a implementar mediante Shell Script y llamadas al CLI de OpenStack. En la página oficial OpenStack se habla sobre los SDKs que existen y los más utilizados, pero dado que ya tendremos accesible el CLI no merece la pena meter más capas a la implementación.

La configuración que debe insertar el administrador se realizará sobre un fichero con formato YAML. Esta decisión es tomada, primero, por la necesidad de tener una estructura de datos organizada que nos ofrezca el propio formato (luego un formato plano tipo TXT lo descartamos). Descartamos a su vez XML debido a que el pesado etiquetado que ofrece XML es innecesario, y nuestro fichero debe ser lo más simple de leer para el administrador, ya que debe modificarlo de forma rápida. En cuanto a JSON también lo descartamos, ya que, como XML, está muy orientado al simple intercambio de información entre máquinas. Además, YAML permite crear comentarios, es decir, está más orientado a la lectura de una persona.

Para el lenguaje de alto nivel, se barajaron dos opciones: utilizar Python o utilizar Ruby. Dado que el CLI de OpenStack utiliza librerías Python para las llamadas a los servicios y Horizon utiliza también Python mediante Django, es más lógico que sigamos utilizando Python como el lenguaje de alto nivel. Además, Python cuenta con la librería Fabric.

Fabric permite ejecutar de manera sencilla comandos en diferentes máquinas. Simplemente se le debe definir las credenciales de acceso a las máquinas remotas, y con sencillas operaciones ("put", "run", "sudo", etc.) se puede ejecutar de forma remota lo que necesitemos. A su vez, gracias a Fabric podemos definir roles. Por ejemplo podemos definir un rol "nova-computer", que tendrá asociado un rango de IPs, y a ese rango de IPs se le ejecutarán los comandos necesarios para la configuración e instalación de un nodo computacional.

En cuanto a las herramientas de despliegue y configuración: Dado que ambas soluciones analizadas anteriormente (Mirantis y RDO) utilizan Puppet, nos centraremos en analizar esta herramienta, así como en apoyarnos en los módulos ya implementados de los componentes OpenStack que podemos encontrar en Puppet-Labs. Otro punto a favor de Puppet es que es la herramienta de este tipo más asentada y con más años de desarrollo, mientras que los competidores (Ansible, etc.) son relativamente modernos.

• Flujo despliegue

En la *figura 9* vemos el flujo de despliegue. Todo comienza cuando el administrador ejecuta el script run.sh. Llamará a Fabric mediante la herramienta de línea de comandos (fab) y ejecutará la tarea "startOpenstack". La tarea se encuentra definida en el fichero fabfile.py, que podemos encontrar en la carpeta "python" del proyecto. La tarea importará el módulo openstack y ejecutará la función "startDeployment" asociada a dicho módulo. Con esto, se pasará a ejecutar el código que encontramos en el fichero openstack.py.

Primero se crearán las variables necesarias requeridas por la librería de Fabric, esto es: usuario y contraseña de las máquinas a las que nos conectemos e IPs de las máquinas. Estos datos los obtendrá del fichero configOpenstackUser.yaml, así como el rol que desempeña cada

nodo. Acto seguido, se crearán mediante la función “startConfigFiles” los ficheros de texto que se pasarán a cada nodo, dependiendo del rol que desempeñen. Estos ficheros de texto contienen variables Facter (Facter contiene variables del sistema en formato Puppet y se le pueden definir variables externas), y a su vez se harán uso de ellas en las plantillas de Puppet. Posteriormente, se pasará a lanzar los servicios en los nodos que corresponda, es decir, al ejecutar “startKeystone”, Fabric ejecutará esta función únicamente en los nodos que estén en el rol “keystone”. Si un nodo no está en este rol, no ejecutará la función sobre dicho nodo. Con esto nos aseguramos que cada nodo recibe exactamente el despliegue de servicios que queramos.

“startKeystone” llamará al módulo keystone.py. Este módulo moverá, gracias a Fabric, los ficheros y directorios necesarios de una máquina (la máquina anfitrión) a las máquinas virtuales (sólo los ficheros necesarios para configurar e instalar el servicio Keystone).

Tras ubicar cada fichero donde corresponda (por ejemplo, el fichero keystone.txt en la ruta donde a Facter se le pueden añadir nuevas variables), keystone.py pasará a ejecutar a través de Puppet las plantillas correspondientes para instalar y configurar el servicio Keystone. En este caso, instalará y configurará la base de datos MySQL, instalará Keystone y además insertará datos necesarios en el servicio de Keystone (por ejemplo, el usuario administrador de OpenStack).

Las plantillas Puppet buscarán las variables que necesiten en las variables que tenga definidas Facter, haciendo *matching*. Por ejemplo, si la plantilla mysqlDB.pp necesita una variable llamada “root_password”, Puppet buscará dicha variable en Facter. Si Facter tiene definida una asignación a dicha variable (por ejemplo root_password=pass1234), entonces en la plantilla Puppet se cambiará “root_password” por “pass1234”). Esto nos permite tener plantillas únicas para diferentes implementaciones.

Tras terminar con Keystone, pasará a ejecutar “startGlance”. Los pasos que siguen son similares, con la salvedad de que habrá que mover ficheros de Glance e instalar y configurar el servicio Glance.

Para finalizar, se moverán los scripts de configuración a posteriori para que el administrador, cuando desee, los ejecute.

Podemos encontrar el manual del administrador para el despliegue en el “*Anexo 4: Manual del administrador*”.

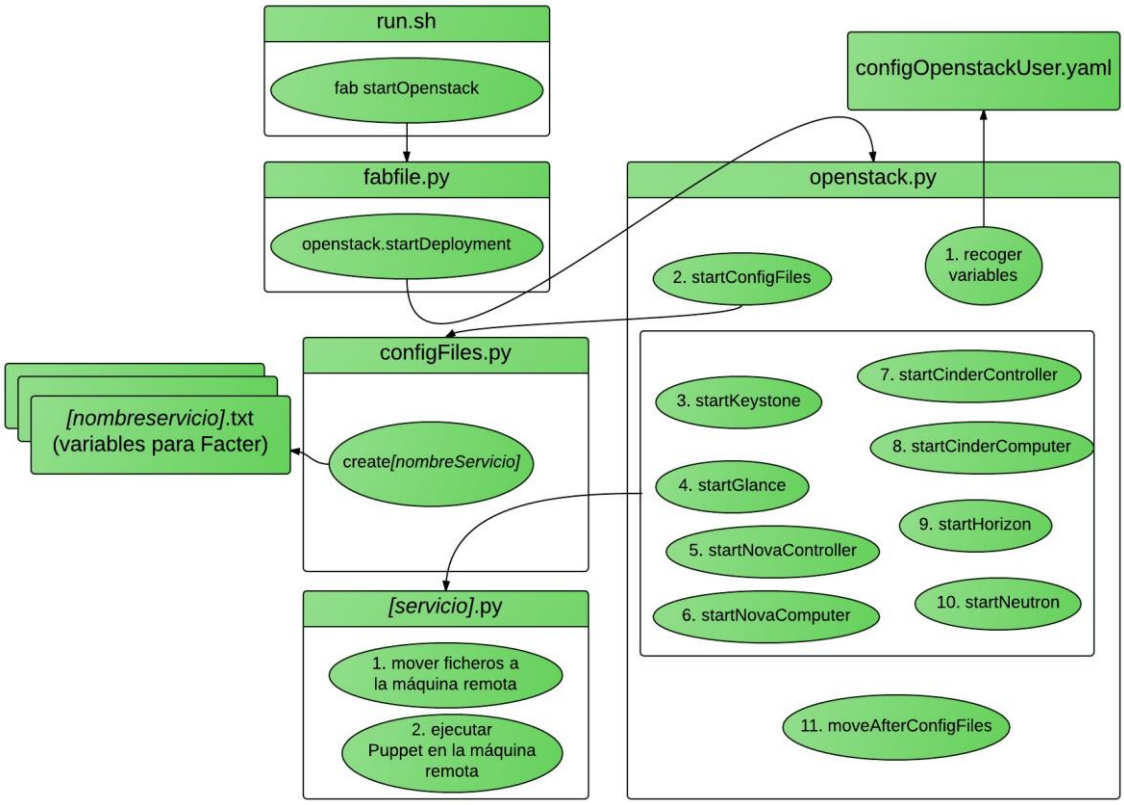

Figura 9: Flujo despliegue detallado

- ## Implementación de la lógica de la aplicación

El primer Shell Script que se creará será el que ejecute toda la lógica de la aplicación: *run.sh*. Consta de una llamada al módulo predefinido en el fichero *fabfile.py* a través de la librería de Fabric. Al lanzar el script se mostrará toda la ejecución por el terminal, y además se alojará toda la información en un fichero de logs, llamado *openstack.log*, para su posterior revisión, en caso de que sea necesario.

Fabric aparte de definirnos los roles para diferenciar cada máquina del servicio que se le asocia (como se vio en las decisiones de diseño), cuenta con la noción de paralelismo. Con ello, podemos lograr lanzar servicios que no dependan entre sí (en nuestro caso, los nodos computacionales tanto de Nova como de Cinder) paralelamente, pudiendo lanzar la configuración a *N* máquinas, ocupando prácticamente el mismo tiempo que si se lanzase en una única máquina.

En cuanto a las operaciones que se realizan mediante Python, se pueden sintetizar en las siguientes:

- Instalar Puppet en la máquina remota.
- Mover los manifiestos de Puppet de la maquina local a la remota:
  - Creamos la ruta $HOME/[tenant] si no está creada (donde se depositan los elementos necesarios para el despliegue).
  - Movemos al $HOME/[tenant] los módulos Puppet y una vez estén en la máquina remota, los copiamos a la ruta de Puppet (/etc/puppet/modules).
- Mover el fichero de configuracion a la ruta de Facter:
  - Crea el directorio Facter en el caso de que no exista.
  - Copia el fichero de configuración que contiene las variables a la ubicación de Facter.
- Mover los plantillas Puppet del servicio a desplegar a la ruta especificada.
- Ejecutar las plantillas Puppet (*puppet apply nombrePlantilla.pp*).

La estructura del fichero de configuración del administrador la encontramos en el "*Anexo 4: fichero de configuración del usuario*". Vemos que tiene una etiqueta *admin* con 3 campos: *user*, *password* y *tenant*. La potencia que nos da una estructura de datos es que podemos definir diferentes campos *user* para diferentes etiquetas, es decir, podemos tener una etiqueta *keystone* con un campo *user*, otra etiqueta *glance* con un campo *user* distinto al anterior, etc. En el fichero se definirá la ubicación de la base de datos en la etiqueta *mysql* y la contraseña de root asociada a la base de datos, para que desde Puppet podamos modificar dicha base de datos. En la misma etiqueta vemos la definición de un usuario, contraseña y nombre de la base de datos que se creará para cada servicio (por ejemplo, para el servicio keystone se le asociará la base de datos definida en la etiqueta keystone-db_name, usuario de la base de datos será keystone-user y la contraseña keystone-password).

En el siguiente punto del fichero, el administrador podrá definir tanto el usuario y contraseña de rabbitMQ, así como su ubicación (su IP). Esto es debido a que por defecto rabbitMQ crea un usuario "guest" con misma contraseña que el nombre de usuario. Al considerarlo inseguro, se decide que se pueda cambiar a elección del administrador.

Para cada servicio, se define una etiqueta y la IP donde se desplegará el servicio en cuestión. Cabe destacar que Cinder y Nova tienen dos subniveles más de etiquetación: *controller* y *computer*. En el *controller* definimos el controlador del servicio, y en el *computer* definimos un rango (por ejemplo, si ponemos 192.168.122.200-192.168.122.202, los nodos serán 192.168.122.200, 192.168.122.201 y 192.168.122.202) que actuarán como cómputos del servicio (para Nova más computación, para Cinder más sitio para almacenar volúmenes).

La implementación de las clases Puppet la sustentaremos sobre los módulos predefinidos que nos ofrece Puppet Forge. Existen módulos para todos los componentes OpenStack, pero nosotros nos centraremos únicamente en los necesarios para poder ofrecer un servicio mínimo funcional de OpenStack (computación, red, almacenamiento e interfaz de acceso y control).

Los módulos necesarios serán previamente descargados para evitar problemas de versiones, así tenemos controlado localmente los ficheros de Puppet Forge sin depender de la red para descargar los módulos, ni de cualquier otro tipo de problema (los tenemos nosotros y los gestionamos nosotros, y en el caso de que queramos utilizar una versión superior, modificaremos dichos ficheros locales).

Para cada servicio de OpenStack, se desarrollará un manifiesto Puppet con una estructura general, a modo plantilla, para que podamos definirle la configuración que queramos.

A modo de ejemplo, se muestra la siguiente línea de un manifiesto:

sql_connection => "mysql://${user_mysql}:${password_mysql}@${ip_mysql}/${db_mysql}"

Siendo que ${user_mysql} será sustituido por el contenido de la variable "user_mysql". Así se mapeará cada variable en su correspondiente lugar. Un posible mapeo sería:

sql_connection => "mysql://root:rootpass@127.0.0.1/nombreBaseDatos"

Entonces Puppet verificará que la conexión MySQL sea con el usuario "root", contraseña "rootpass", en la dirección IP localhost (127.0.0.1) y en la base de datos con nombre "nombreBaseDatos".

El valor de las variables las obtendrá de Facter. A Facter se le puede definir variables externas si se depositan los ficheros donde estarán dichas variables en un formato que pueda comprender (JSON, YAML o TXT). La ruta de Facter donde se depositan los ficheros hay que crearla: /etc/facter/facts.d.

Las variables en nuestro caso serán pasadas a través de un fichero de texto, ya que no requerimos la necesidad de utilizar un lenguaje de definición de datos estructurados. Para cada módulo OpenStack generaremos su propio fichero, para asegurar la independencia entre servicios, y que cada módulo tenga las variables necesarias. Esto nos da el beneficio concreto

de que cada servicio sólo sabrá por ejemplo su contraseña de acceso a la base de datos, o la IPs concretas que necesite, sin necesidad de que tenga otro tipo de información que pueda ser sensible.

Un ejemplo de fichero de texto que se le pasará lo podemos encontrar en "*Anexo 4: ejemplo fichero de configuración Facter*".

Para finalizar, en la carpeta llamada *afterDeployment* encontramos una serie de scripts para lanzar una configuración mínima funcional OpenStack.

- *trasDespliegue.sh:* ejecuta todos los scripts que vemos a continuación.
- *Create_user.sh:* crea un usuario no administrador y un *tenant* asociado.
- *create_image.sh:* crea en Glance una imagen de un sistema operativo CirrOS.
- *create_network.sh:* crea la red externa e interna de OpenStack.
- *create_volume.sh:* crea un grupo volumen asociado a Cinder.
- *modify-external-network.sh:* convierte el nodo controlador en un router, para permitir salida al exterior.
- *restartServices.sh:* reinicia todos los servicios OpenStack.

• Estructura de directorios

Vamos a ver en detalle cada elemento que existe en nuestro directorio principal de la aplicación.

Vemos que en la raíz tenemos el ejecutable (run.sh), el fichero que modificará el administrador (configOpenstackUser.yaml), el fichero de logs (openstack.log) y 4 directorios.

Python, donde tendremos los ficheros implementados en dicho lenguaje para cada servicio, así como otros elementos necesarios en Python (como __init__.py, necesario para que Python interprete los ficheros como módulos y poder ser llamados desde otros módulos).

Podemos observar que Puppet se desglosa en 2 sub-directorios: puppet-labs, que contiene los directorios y ficheros descargados de la web de Puppet-Labs, usando el último *release* asociado a la versión 3 (no entramos al detalle dado que existen demasiados directorios y ficheros asociados) y el directorio puppet-plantillas, que a su vez este último contendrá 6 directorios. Estos 6 directorios guardan las plantillas Puppet asociadas a cada servicio OpenStack (tampoco entraremos al detalle de sus nombres, ya que la mayoría se basan en un fichero de configuración de la base de datos MySQL y otro fichero para el despliegue y configuración del servicio concreto).

El directorio afterDeployment almacena los ejecutables Shell Script que se usarán para la configuración posterior al despliegue OpenStack (creación de red, usuario no administrador, etc.).

El directorio configFiles se usará para almacenar los ficheros de texto (txt) que serán usados por Facter. El formato que tendrá cada fichero de texto de dicho directorio será [nombreServicio].txt. Es decir, tendremos keystone.txt (con las variables Facter necesarias para desplegar Keystone), glance.txt, novaComputer.txt, etc.

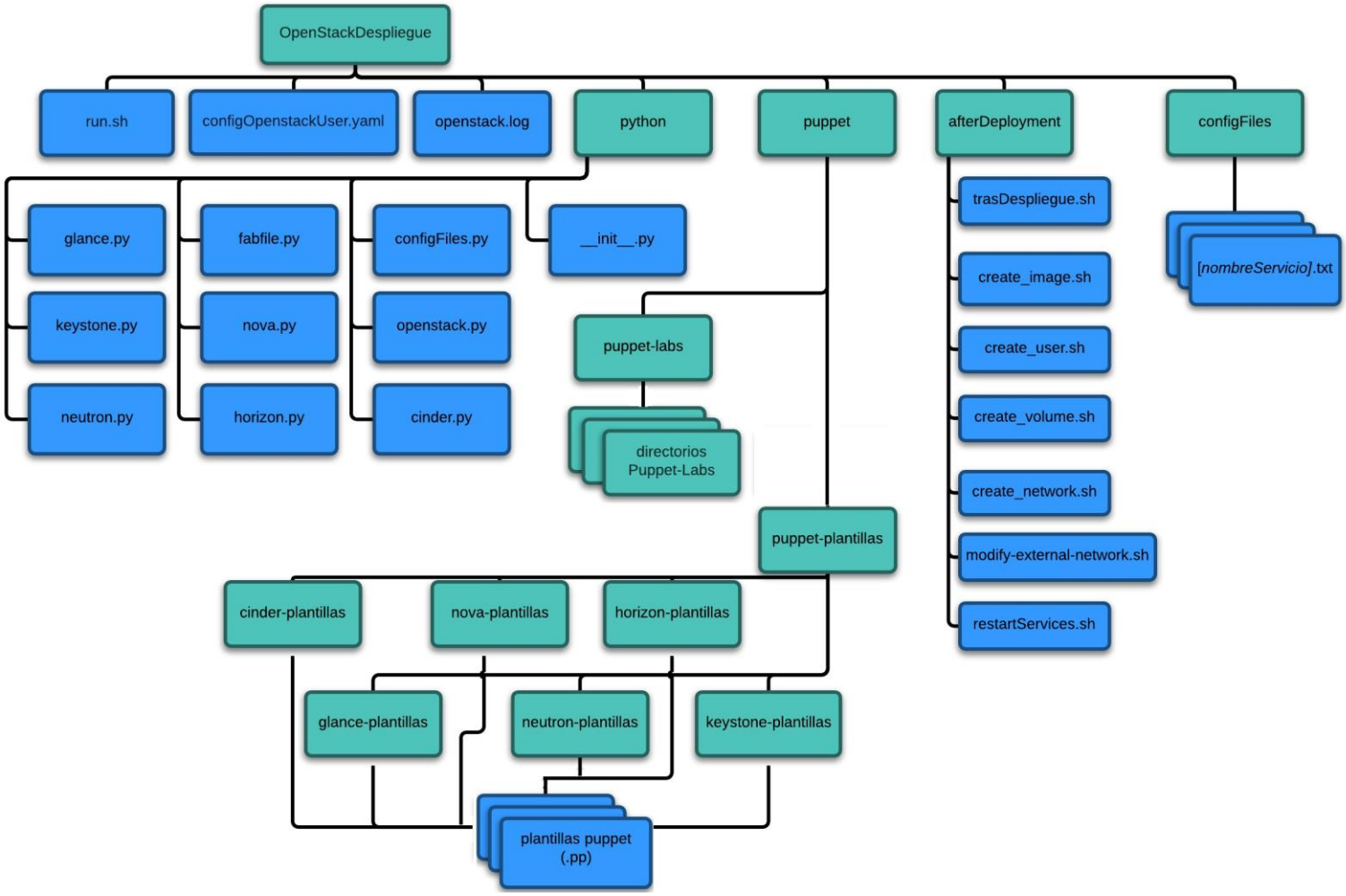

Figura 10: Estructura de directorios detallada

Evaluación

Con el fin de testear nuestro despliegue OpenStack y validar su correcto funcionamiento, vamos a suponer dos escenarios de uso en cuanto al despliegue se refiere, y además haremos una prueba sobre estos casos de uso, suponiendo que un usuario quiere desplegar por encima a través de una instancia un sistema de gestión de contenidos (CMS), proveyendo OpenStack la infraestructura necesaria para lograr su objetivo.

• Escenarios de uso

En el primer escenario, se desplegará en una única máquina virtual un nodo *All-in-one*, es decir, en dicho nodo se albergará toda la configuración mínima para que funcione OpenStack.

En el segundo escenario de uso, se añadirá a dicho nodo más nodos computacionales tanto de procesamiento (Nova) como de almacenaje (Cinder). Con un nuevo nodo que tenga ambos servicios nos será suficiente, ya que en caso de querer añadir nuevos nodos sería replicar este proceso.

○ All-in-One

Es el que más se utiliza para realizar pruebas sobre OpenStack. Cambiamos del fichero de configuración del administrador todas las IPs por la del nodo donde se va a desplegar OpenStack, así como su usuario y contraseña de acceso. Lanzamos el script “run.sh” y veremos en la terminal el despliegue (que se almacenará en el fichero de logs para su posterior revisión).

○ Añadiendo nodos

Despliegue típico en un entorno de producción que use OpenStack. Este tipo de despliegue viene detallado en el “*Anexo 4: Manual del administrador*”. La idea es tener un nodo controlador y además computacional, y tener los nodos computacionales que queramos, todos ellos bajo la misma subred. Además, se creará la red exterior de OpenStack que conectará las instancias (alojadas en redes privadas) con el exterior. Se observa en la *figura 11* que la red externa OpenStack se conecta directamente el “router exterior” a través del “router exterior *tenants*”. Bajo el “router exterior *tenants*” definimos una subred, en nuestro caso: 192.168.1.0/24.

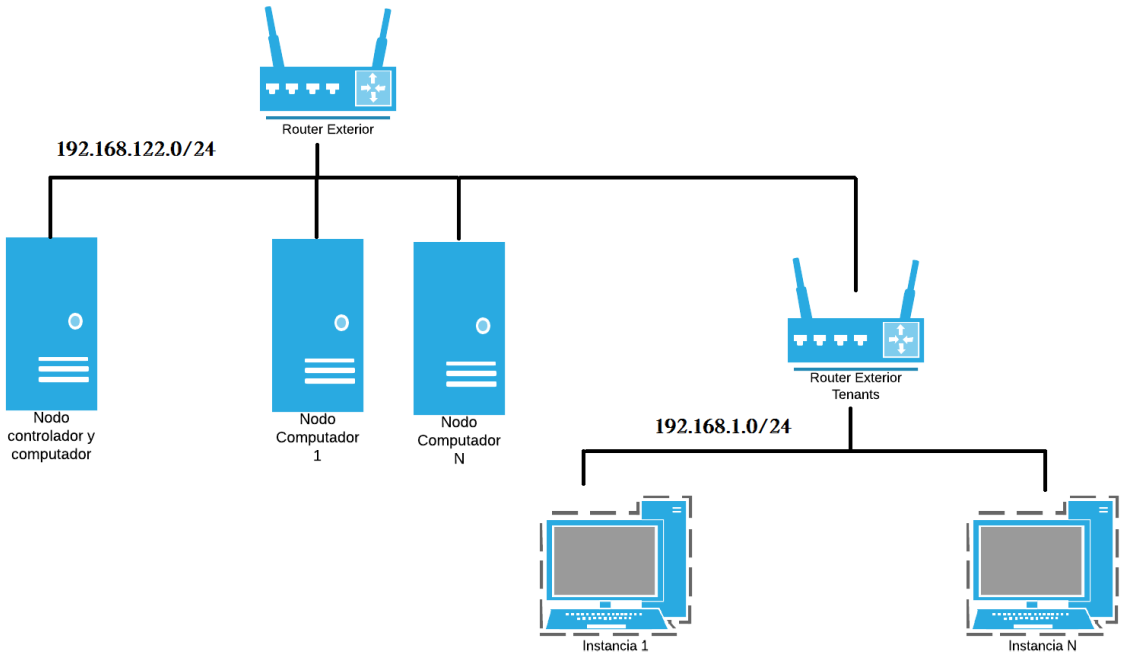

Figura 11: Esquema OpenStack con nodos

- Validación

Tras finalizar la instalación que corresponda, comprobaremos que todos los servicios se han desplegado. Para ello, ejecutamos en el nodo *All-in-one* el fichero openrc.sh que se ha creado en la ruta $HOME/[tenant]. En este fichero se definen variables de entorno para acceder a OpenStack (credenciales y *endpoint*). Ahora podemos ejecutar comandos típicos del CLI para comprobar que los servicios nos responden: *keystone list, nova list, cinder list, glance image-list, neutron route-list,* etc.

Podemos comprobar que los nodos computacionales se han detectado correctamente si ejecutamos, para el caso de Nova:

sudo nova-manage service list

Podemos ver si está operativo si encontramos en el estado una cara sonriente ":-)". Si no lo estuviese, encontraríamos "XXX".

También podemos hacer la comprobación en otros servicios:

sudo cinder-manage service list
neutron agent-list

Los servicios están disponibles, luego vamos a testearlos. Para comprobar Keystone es tan fácil como por ejemplo probar a hacer *login* con un usuario en la interfaz Horizon. Si nos valida correctamente, el servicio funciona. Al acceder a través de Horizon, por consiguiente, este servicio será validado.

Para Glance, subimos la imagen CirrOS con el script *create_image.sh*. Si no hay problemas, deberíamos ver a través de Horizon en la sección de imágenes la imagen CirrOS (o desde CLI*: glance image-list*).

Para comprobar la red (Neutron), lo mejor es que hagamos uso de ella. Con el script create_network.sh, se despliega la red. Desde Horizon podemos ver si se ha creado o no tanto la red externa como interna, si entramos en la sección *network* (hay que estar en el proyecto del administrador, ya que la red interna se crea para ese proyecto). Deberíamos ver gráficamente dos redes conectadas mediante un router.

Para finalizar, lanzaremos una instancia en la red interna. Si se instancia correctamente, tenemos Nova validado. Además, si entramos en la máquina (desde la interfaz Horizon) y hacemos *login* en la instancia lanzada, a través del comando *ifconfig* podemos ver si nos ha asignado una IP de la subred correctamente. Podemos hacer "ping" para comprobar que las conexiones son correctas (y con esto terminaríamos de validar Neutron).

Vamos a comprobar el correcto funcionamiento del almacenamiento de bloques. Para ello, comprobamos mediante el comando "vgs" que existe al menos un grupo de volúmenes sobre el cual se van a crear los volúmenes lógicos de cada *tenant* OpenStack. Si no existe ningún grupo, verificamos que existe al menos un volumen físico en la máquina virtual. Si no existen, pasaremos a crearlos. Por ejemplo, añadiremos a la máquina virtual un segundo disco duro, que lo llamaremos /dev/hdb. Debemos crear un volumen físico sobre este dispositivo (sudo pvcreate /dev/hdb) y finalmente crearemos el grupo. El grupo lo podemos llamar por ejemplo "cinder-volumes": sudo vgcreate cinder-volumes /dev/vdb.

Una vez tenemos el grupo de volúmenes creado, podemos lanzar desde la interfaz OpenStack la creación de un volumen. Una vez veamos que está activo, podemos comprobar que se ha creado correctamente si obtenemos los volúmenes lógicos que están disponibles. Podemos comprobarlo mediante el comando "lvs".

Nivel de usuario

Como validación a nivel de usuario, se plantea desplegar una red en la que existe una máquina virtual con acceso al exterior. Dicha máquina virtual dispondrá de un CMS (un gestor de contenido, por ejemplo WordPress). Apache y MySQL serán los elementos centrales que apoyen el CMS. Con esto comprobaremos que podemos conectarnos a través de un navegador web al CMS y autenticarnos para editar el contenido del CMS.

Para ello, debemos tener preparada una máquina que pueda albergar un servidor Apache, MySQL y PHP (las imágenes CirrOS que instalo por defecto no nos sirven), así que se preparará una imagen Ubuntu 14.04 servidor (1.8GB de tamaño, en comparación a los 12.5MB de una CirrOS).

Suponiendo que el usuario ha creado una subred interna conectada a través de un router a la subred de salida, la instancia será creada en la red interna, tal y como vemos en la *figura 12*.

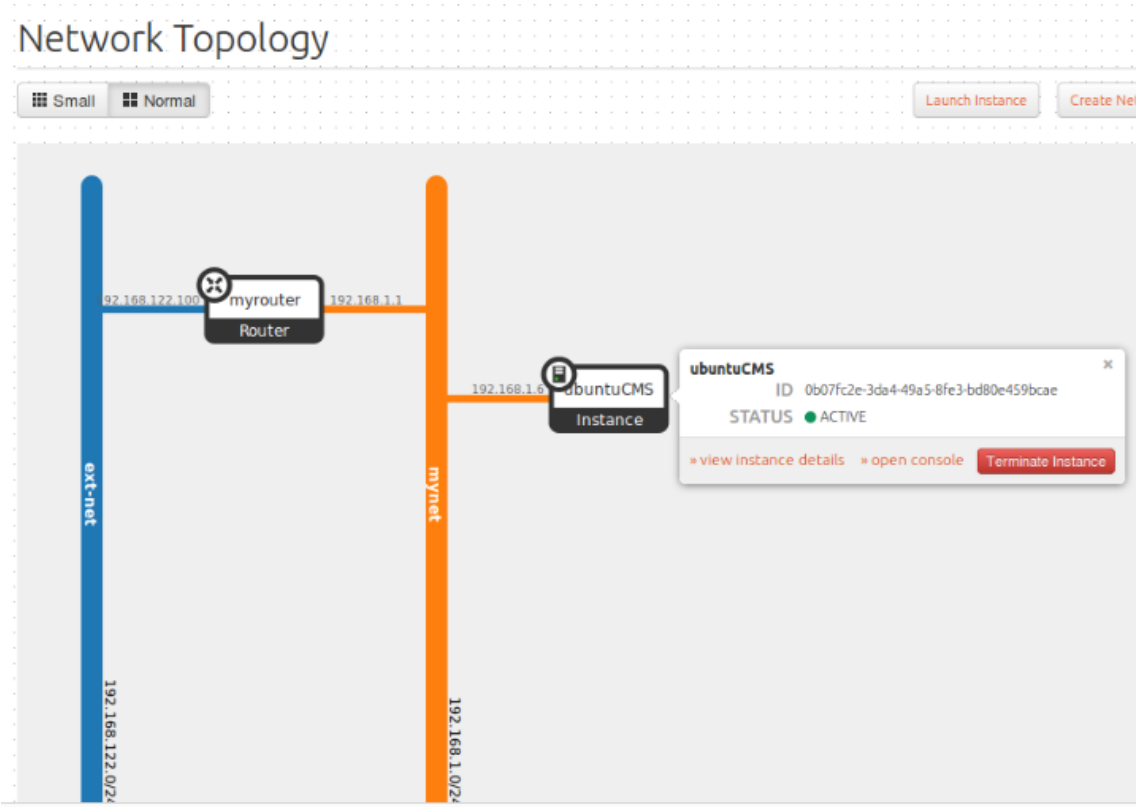

Figura 12: Topología de red con instancia servidor

A la hora de acceder a la instancia tenemos dos opciones: una de ellas es que la máquina tenga predefinida un usuario y contraseña, y acceder desde la interfaz (o si está preparada para recibir conexiones SSH a través de un terminal) y otra forma a través de SSH con clave pública/privada, que se puede generar desde la interfaz de Horizon. Nos descargaríamos la clave y accederíamos con ella de la siguiente forma:

ssh –i nombreClave.pem nombreUsuario@IP

Observación: la clave debe tener privilegios mínimos, es decir, 400 (sólo lectura por el propietario).

Dado que la instancia no es accesible desde fuera (está en una subred interna) debemos asignare una IP flotante. Esta IP sí que es accesible cara al exterior (suponiendo que tuviésemos un conjunto de IPs públicas. En nuestro caso, “público” lo entendemos como que desde una máquina de la subred podemos acceder con el navegador para comprobar su

funcionamiento). En la *figura 13* se ve la asociación entre la máquina llamada “ubuntuCMS” y la IP “192.168.122.102”.

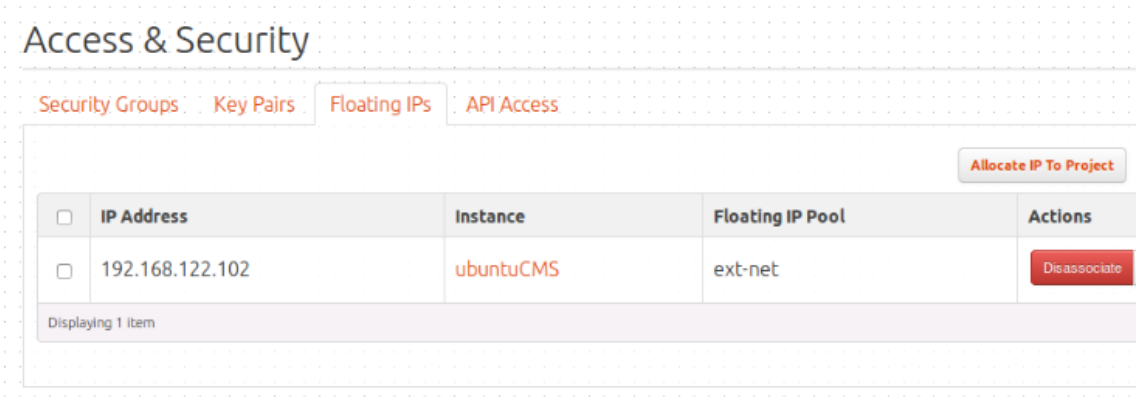

Figura 13: asociación IP-CMS (IP flotante = 192.168.122.102)

Los pasos seguidos para la instalación del CMS WordPress están en el “*Anexo 4: WordPress*”.

A continuación, probaremos a conectarnos a través de un navegador a dicha IP (recordando que tenemos que estar en la misma subred para acceder a ella, es decir, desde mi portátil con IP 192.168.122.1 puedo acceder al CMS). Vemos en la *figura 14* que sí que podemos entrar a la interfaz y además entrar a través del *login* al usuario administrador.

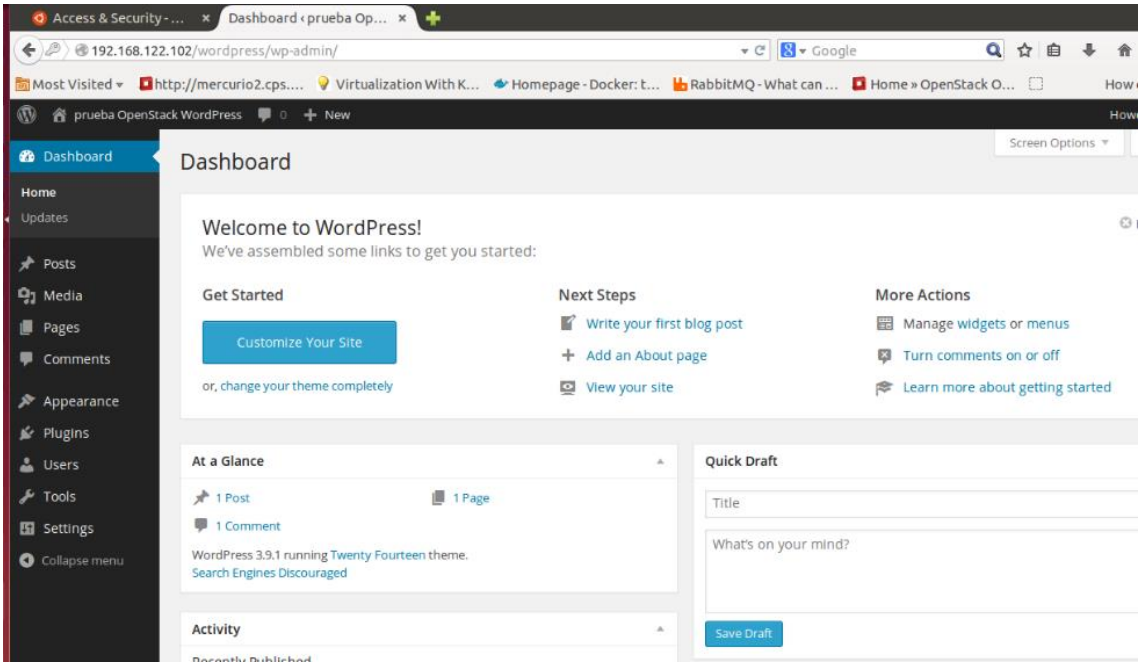

Figura 14: Dashboard administración WordPress (conexión IP flotante)

Para desplegar el servidor, debimos antes cumplir ciertos requisitos. Por ejemplo, creamos un *flavor* (recursos que se le ofrece a una instancia) con 7VCPUs y 2GB de RAM. También nos aseguramos espacio del disco duro en el nodo donde se fuese a desplegar la máquina (por lo menos, la máquina debe tener unos 12GB de ROM).

El tiempo de despliegue llevó aproximadamente 4 minutos, cuando con las CirrOS suele ser inferior a 10 segundos.

Conclusiones

Conclusiones del proyecto

A lo largo del proyecto, se tuvieron en cuenta muchas variaciones y decisiones que marcarían el resto del proyecto. Entre otros, el más importante para el proyecto fue la decisión de cambiar de versión, tanto en Ubuntu como en OpenStack. A pesar de ser un cambio arriesgado, dado que podían surgir problemas todavía no documentados, a partir de los resultados, se determina que fue acertada la decisión.

Se consiguió realizar un análisis exhaustivo de un *cloud* tipo *IaaS*, tanto a nivel de administración (análisis de componentes, diferentes tipos implementaciones, etc.) como a más alto nivel (diferentes escenarios de despliegue, pruebas a nivel de usuario consumidor del servicio, etc.).

Se consiguió el objetivo principal de diseñar un sistema de configuración dinámica de OpenStack, así como la implementación y pruebas de diferentes casos de uso. Se ofrece al administrador la posibilidad de desplegar entornos OpenStack de forma trivial con una configuración determinada.

Uno de los objetivos interesantes que se buscaba en el proyecto era el de realizar cambios en caliente (es decir, mientras OpenStack esté funcionando, sin dejar que el usuario final pierda servicio), pero dada la complejidad del proyecto (sobretodo en el conocimiento de los componentes y su implementación) y el poco tiempo del que se disponía, esta parte no ha sido testeada como se debería.

En cuanto a problemas relacionados con la configuración de servicios, han aparecido de todo tipo y en todos los servicios. Algunos servicios fueron menos problemáticos de implantar, pero algunos, en especial la red (Neutron), son muy complicados y nada triviales de configurar. De hecho, hasta finales del proyecto no se llegó a tener una versión estable de la red, lo que implica que se deberá en un futuro testear con más profundidad.

Como se ha podido ver, este TFG toca una cantidad relativamente alta de conceptos vistos a lo largo del grado: redes, almacenamiento, administración, bases de datos, ingeniería del software, etc. Es por ello que se considera un proyecto muy útil a nivel didáctico y a nivel de uso posterior, dado que será empleado para la docencia de dos asignaturas, como se cita en la introducción del proyecto.

Conclusiones generales

Montar un entorno *cloud*, en este caso mediante OpenStack, realizar su configuración y control de recursos no es nada trivial. Requiere tener altos conocimientos informáticos, desde un nivel de administración alto hasta controlar la gestión de redes y almacenamiento. Es por ello que están surgiendo tantísimas empresas cuya única finalidad es la de configurar entornos *cloud* privados y realizar su mantenimiento, quitándole este pesado trabajo a los informáticos de las empresas, ya que el sólo hecho de conocer los componentes y las implicaciones que tiene el uso de dichos componentes puede llevar incluso meses. Luego la primera conclusión es que montar OpenStack en una empresa es una ardua tarea, tanto económicamente como temporalmente.

La complejidad informática cada vez es mayor, y los usuarios no están por la labor de invertir una cantidad importante de su tiempo en realizar despliegues manuales (en una empresa supone gastar tiempo y por lo tanto dinero). Por lo tanto, es importante reflejar como conclusión que debemos abstraer lo máximo posible al usuario final de la complejidad que supone estos tipos de configuración, y proporcionarles mecanismos automáticos que les faciliten dichas tareas.

Otra conclusión que se obtiene de este trabajo es que hoy en día, los entornos informáticos están cambiando de manera muy rápida. Tanto que no llega a establecerse una versión, en este caso de OpenStack Havana, y ya se prepara la salida de una nueva versión (apenas 6 meses de margen entre cada versión que ha ido apareciendo). La velocidad de estos desarrollos demuestra el interés que tienen las grandes empresas (RackSpace, Mirantis, HP, Ubuntu, etc.) en conseguir una versión Open Source de tipo *cloud* competitiva, que llegue a estar a la altura de grandes servidores de *IaaS* como Amazon o Google.

Ligado a esto último, tanto los rápidos cambios en la tecnología como su alta complejidad, nos obliga a los informáticos a tener que tomar decisiones mucho más rápidas, barajando diferentes alternativas y tomando decisiones acordes a un criterio. En este proyecto, se tomaron varias decisiones en poco tiempo (¿Migrar a la nueva versión OpenStack/Ubuntu? ¿Qué lenguajes utilizamos, Puppet, Chef o Ansible? ¿Python o Ruby? ¿YAML o JSON? Etc), así que como última conclusión me gustaría destacar la importancia de tomar decisiones rápidas y con conocimiento.

- ## Trabajo futuro

Para perfeccionar este TFG, se podría trabajar en un futuro con la realización de un tipo de interfaz (ya sea desde un terminal o incluso a través de un servidor web) que pida al administrador los campos necesarios para realizar el despliegue OpenStack, sin necesidad de tocar el fichero de configuración (el fichero se rellenaría automáticamente a partir de los datos introducidos por el administrador).

También se podría realizar un despliegue de todos los componentes que ofrece actualmente OpenStack, es decir, no únicamente los servicios asociados a computación, red y almacenamiento de bloques, sino también otros servicios como Swift, Heat, Ceilometer, etc.

Algo que sería interesante realizar en un futuro es la comprobación de ciertos componentes antes de lanzar el despliegue. Por ejemplo, comprobar que la red es la que debe ser para que el entorno funcione correctamente, y en caso de no serlo no ejecutar el despliegue (al estilo de Mirantis).

Se podría tener en cuenta también diferentes tipos de implementaciones de los servicios que ofrece OpenStack. Por ejemplo, que se le dé a elegir al usuario si quiere implementar Cinder con GlusterFS o Ceph por debajo, o si quiere que las redes usen GRE o VLAN.

En cuanto a nivel de validación, hubiese sido interesante realizar más pruebas: mover instancias entre servidores, como se realizó en DevStack, crear zonas, comprobar los límites del sistema mediante sobrecarga, etc. También se podría validar hasta qué punto nuestro sistema admite modificaciones de la configuración en caliente, parando lo mínimo el servicio, y ver qué repercusión tiene estos cambios cara al usuario final.

Se deja para un futuro el diseño e implementación de alta disponibilidad. Es uno de los conceptos más interesantes a corto plazo, dado que el *cloud* debería proveer siempre servicio funcional al usuario final (Algunos ya están controlados, por ejemplo en el caso de que un nodo quede fuera de servicio las instancias migrarían a otros nodos gracias a Nova, pero ¿y si se cae por ejemplo la base de datos?). Algunos ideas de componentes que se podrían redundar se pueden ver en el "*Anexo 2: RackSpace: alta disponibilidad*".

Bibliografía

Nota: El último acceso de la totalidad de los links es el 25 de junio de 2014.

1. Proyecto OpenStack

OpenStack, *OpenStack Main Page* http://www.openstack.org/
OpenStack, *OpenStack docs* http://docs.openstack.org/
OpenStack, *OpenStack wiki* https://wiki.openstack.org/wiki/Main_Page
OpenStack, *ask OpenStack* https://ask.openstack.org/en/questions/
OpenStack, *installation Guide for Ubuntu 12.04/14.04 (LTS)*
http://docs.openstack.org/icehouse/install-guide/install/apt/content/
OpenStack, *installation Guide for Ubuntu 12.04 (LTS)* http://docs.openstack.org/havana/install-guide/install/apt/content/
OpenStack, *OpenStack GitHub* https://github.com/openstack/

2. Tecnologías y aspectos de OpenStack

Victoria Martínez de la Cruz, *En pocas palabras: ¿Cómo funciona OpenStack?*
http://vmartinezdelacruz.com/en-pocas-palabras-como-funciona-openstack/
Ken Pepple, *OpenStack Folsom Architecture*
http://ken.pepple.info/openstack/2012/09/25/openstack-folsom-architecture/
Wikipedia, *OpenStack* http://es.wikipedia.org/wiki/OpenStack
Roberto Orayen, *Introducción a OpenStack Havana y RedHat*
http://www.elblogdenegu.com/introduccion-a-openstack-havana-y-redhat
Emujicad, *Presentación OpenStack Comunidad Venezuela*
http://www.slideshare.net/emujicad/presentacion-openstack-comunidad-venezuela
Lorin Hochstein, Python APIs: *The best-kept secret of OpenStack*
http://www.ibm.com/developerworks/cloud/library/cl-openstack-pythonapis/index.html?ca=drs-
SwiftStack, OpenStack Swift Architecture https://swiftstack.com/openstack-swift/architecture/
Gonzalo Nazareno, *Introducción a OpenStack Horizon*
http://www.gonzalonazareno.org/cloud/material/intro-horizon.pdf
Flux7 Labs, *What is Neutron? How to install and use it?*
http://flux7.com/blogs/openstack/tutorial-what-is-neutron-how-to-install-and-use-it/
Sebean Hsiung, *Block-based vs object-based storage systems*
http://www.datarecoverytools.co.uk/2009/11/13/block-based-vs-object-based-storage-systems/
Wikipedia, *Object storage* http://en.wikipedia.org/wiki/Object_storage#Object-based_storage_devices
Wikipedia, *AMQP* http://es.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol
Jeus Lara, *Instalando y usando OpenVSwitch en Debian GNU/Linux*
http://phenobarbital.wordpress.com/2012/12/08/instalando-y-usando-openvswitch-en-debian-gnulinux/
Ceph, *Ceph* http://ceph.com/
Kenneth Hui, *OpenStack Compute For vSphere Admins*
http://cloudarchitectmusings.com/2013/06/24/openstack-for-vmware-admins-nova-compute-with-vsphere-part-1/
Karan Singh, *Ceph Storage :: Next Big Thing* http://karan-mj.blogspot.com.es/2013/12/ceph-storage-part-2.html
Wikipedia, *iSCSI* http://es.wikipedia.org/wiki/ISCSI
Wikipedia, *LVM* http://es.wikipedia.org/wiki/Logical_Volume_Manager
Inktank, *Ceph for OpenStack* http://www.inktank.com/openstack-storage-solutions/

3. Análisis

DevStack, *DevStack* http://devstack.org/
Pablo Seminario, *Jugando con OpenStack… localmente!* http://blog.insert-coin.org/post/38072557449/openstack-local

DevStack, *proyecto DevStack* https://github.com/openstack-dev/devstack
DevStack, Multi-Node Lab: Serious Stuff http://devstack.org/guides/multinode-lab.html
Sébastien Han, *DevStack in 1 minute*, http://www.sebastien-han.fr/blog/2013/08/08/devstack-in-1-minute/
Mirantis, *Welcome to Mirantis OpenStack Documentation* http://docs.mirantis.com/fuel/fuel-4.0/
RackSpace, *Cloud Privado* http://www.rackspace.com/es/cloud/private/
RedHat, *RDO Quickstart* http://openstack.redhat.com/Quickstart

4. Implementación

Kotov, *Automatización con Puppet (I): Condeptos e instalación*
http://rooteando.com/automatizacion-con-puppet
Dean Wilson, *Puppet CookBook* http://www.puppetcookbook.com/
Python for Beginners, *How to use Fabric in Python*
http://www.pythonforbeginners.com/systems-programming/how-to-use-fabric-in-python/

5. Libro referencia

Kevin Jackson, *OpenStack Cloud Computing Cookbook*

Anexo 1: Gestión del proyecto

A continuación se expone el diagrama de Gantt asociado a este proyecto (*figura 15*).

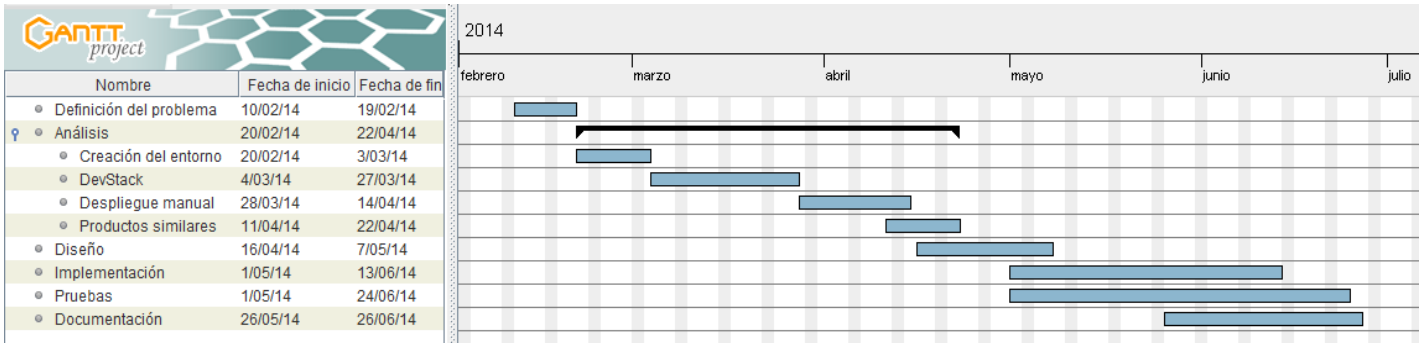

Figura 15: Diagrama de Gantt

Cabe destacar que este proyecto no tiene una definición clara de análisis, diseño, implementación y validación o pruebas, sino que muchas de ellas se solapan. Por ejemplo, mientras estamos analizando los productos similares del mercado, indirectamente estamos diseñando nuestra aplicación; o mientras estábamos desplegando manualmente OpenStack sobre Ubuntu, ya estábamos decidiendo aspectos de diseño, como sobre qué sistema operativo se va a desplegar y la versión tanto del sistema operativo como de OpenStack.

A continuación se muestra las horas consumidas de forma aproximada en cada fase:

Fase	Número de horas
Definición del problema	18
Análisis	
Creación del entorno	28
Devstack	70
Despliegue manual	36
Productos similares	22
Diseño	44
Implementación	86
Pruebas	30
Documentación	45
TOTAL	379

Anexo 2: Aspectos tecnológicos de herramientas

- DevStack
  - Fichero de Configuración

[[local|localrc]]
HOST_IP=192.168.122.50
LOGFILE=/opt/stack/logs/stack.sh.log
VERBOSE=True
LOG_COLOR=True
SCREEN_LOGDIR=/opt/stack/logs
FLAT_INTERFACE=eth0
FIXED_NETWORK_SIZE=4096
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
RABBIT_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=password

#habilitamos servicio Swift
enable_service s-proxy s-object s-container s-account
SWIFT_HASH=password
SYSLOG=True

#deshabilitamos servicio nova network para poner Neutron
disable_service n-net

#habilitamos Neutron (Quantum)
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron

 # Ceilometer
enable_service ceilometer-acompute,ceilometer-acentral,ceilometer-anotification,ceilometer-collector
enable_service ceilometer-alarm-evaluator,ceilometer-alarm-notifier
enable_service ceilometer-api
enable_service heat h-api h-api-cfn h-api-cw h-eng

#cargamos imágenes de Fedora
IMAGE_URLS+=",http://fedorapeople.org/groups/heat/prebuilt-jeos-images/F17-i386-cfntools.qcow2"
IMAGE_URLS+=",http://fedorapeople.org/groups/heat/prebuilt-jeos-images/F17-x86_64-cfntools.qcow2"

Añadiendo Nodo Computacional

El fichero de configuración para el nodo computacional tendrá la siguiente estructura:

```
HOST_IP=192.168.42.12
FLAT_INTERFACE=eth0
FIXED_RANGE=10.4.128.0/20
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.42.128/25
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecrete
SERVICE_PASSWORD=supersecrete
SERVICE_TOKEN=xyzpdqlazydog
DATABASE_TYPE=mysql
SERVICE_HOST=192.168.42.11
MYSQL_HOST=192.168.42.11
RABBIT_HOST=192.168.42.11
GLANCE_HOSTPORT=192.168.42.11:9292
ENABLED_SERVICES=n-cpu,n-net,n-api,c-sch,c-api,c-vol
```

Esto indicará a DevStack que la IP del nodo computacional es la que viene en la variable HOST_IP, la interfaz sobre la que se va a desplegar la red interna del *cloud* (FLAT_INTERFACE), el rango de IPs que se asociará a las instancias en la red privada (FIXED_RANGE) y las que se ofrecerán como IPs flotantes o elásticas (FLOATING_RANGE). Establecemos contraseñas y qué IP tiene el nodo controlador, y definimos los servicios que queremos que estén operativos en el nuevo nodo computacional (n-cpu = nova computer, n-net = nova network, n-api = nova api, c-sch = cinder scheduler, etc).

Modificará entre otras cosas el fichero /etc/nova/nova.conf, diciéndole a quién debe preguntar en el caso de que necesite ciertos servicios (por ejemplo, para acceder a la base de datos MySQL).

Lanzamos el script y vemos, si ejecutamos desde el nodo controlador el siguiente comando:

nova-manage service list

que se ha añadido la máquina como nodo computacional. Desde Horizon también podemos verlo (si hacemos *login* como administrador).

Pruebas: Migración y Zonas de disponibilidad

Migración de instancias

Para la migración:

- Sólo podemos mover máquinas que estén en la misma zona. En caso contrario, la máquina se migrará con estado ERROR.
- Los pasos son:
    1. Paramos proceso nova-compute, sin destruir la instancia (por ejemplo, un *shutdown* mataría la istancia y no funcionaría).
    2. Vemos cómo aparece XXX en nova-compute del que acabamos de matar con "*nova-manage service list*".
    3. nova host-evacuate --target_host <nodo al que migrar> < nodo problemático>
    4. virsh list : vemos la migración

Esto nos puede servir si por ejemplo queremos realizar el mantenimiento de un nodo. Veríamos las instancias que tiene en ejecución y las migraríamos a otros nodos.

¿Cómo hacer que si un nodo se apaga por cualquier tipo de problema, se migre automáticamente?

Tenemos varios comandos involucrados en la migración:

nova live-migration	*Migrate running server to a new machine.*
nova migrate	*Migrate a server. The new host will be selected by the scheduler.*
nova host-servers-migrate	*Migrate all instances of the specified host to other available hosts.*
nova evacuate	*Evacuate server from failed host to specified one.*
nova host-evacuate	*Evacuate all instances from failed host to specified one.*

Hay dos tipos de migración:

- Migración “en frío”: la instancia a migrar debe ser parada para poder cambiar a otro hipervisor.
- Migración “en caliente”: la instancia sigue funcionando mientras se migra, sin perder disponibilidad.

Ha su vez, hay dos tipos de migración en caliente:

- Con almacenamiento compartido: los dos hipervisores tienen acceso al almacenamiento compartido.
- Migración del bloque: no existe almacenamiento compatido.

Moverse a través de DevStack y ubicación de los logs:

Para moverse en el rejoin-stack.sh -> cntr + a + (n ó p, dependiendo si queremos ir hacia delante o hacia atrás).

Para los logs: configurarlos desde local.conf. Problema: no sale por ejemplo en /var/log/nova... como dicen, lo máximo que se a conseguido es que salga el log que se ejecuta al lanzar el script.

Para rearrancar los procesos desde la ventana que aparece con rejoin-stack.sh -> ir a la ventana del proceso que queramos rearrancar (el que aparece en la esquina inferior izquierda es el proceso en cuestión), pulsar cntrl + c para matar el proceso. Después pulsamos el botón de "arriba" para ver última instrucción ejecutada y la ejecutamos. Se rearrancará el proceso (podemos verlo con ps aux cuando matemos el proceso y cuando lo arrancamos de nuevo, cómo vuelve a aparecer).

Elegir dónde lanzar una instancia:

Creamos el nuevo agregado, que creará como zona de disponibilidad "new_zone"

nova aggregate-create test new_zone

Con el ID que nos mostrará el anterior comando (si se ha creado correctamente=, pasamos a añadir un host a dicho agregado. El host lo podemos coger si ejecutamos por ejemplo "nova service-list".

nova aggregate-add-host 1 node-vm

Ya podemos ver cómo se ha añadido el nodo a la nueva zona:

nova availability-zone-list

Lanzamos una instancia de ejemplo en dicha zona:

nova boot ejemplo --image 778d0d68-4e98-40a5-ad01-97ab60ca9035 --flavor 1 --availability-zone new_zone

○ Ejemplos de errores y soluciones

Error al lanzar instancia:

Error que aparece en la interfaz Horizon:

Failed to launch instance "cirros": Please try again later [Error: Unexpected vif_type=binding_failed].

Solución: http://lists.openstack.org/pipermail/openstack/2013-November/003049.html

Fallo de Openvswitch.agent: está caído. Lo podemos ver con *neutron agent-list* (aparece la siguiente línea: 14dc0611-9239-4a9e-80a0-6f0e04572b99 | Open vSwitch agent | controller-vm | XXX | True)

Vemos que falta en los procesos en ejecución (ps aux| grep neutron) este proceso:

python /usr/local/bin/neutron-openvswitch-agent --config-file /etc/neutron/neutron.conf --config-file /etc/neutron/plugins/ml2/ml2_conf.ini

Y si miramos con *ifconfig* vemos que el bridge no está disponible: br-ex no aparece.

La solución rápida: re arrancar strack.sh (no ./rejoin-stack.sh. Mantendría el error).

Error al lanzar nova-api:

2014-02-21 12:13:43.244 CRITICAL nova [-] ProcessExecutionError: Error inesperado al ejecutar orden.
Orden: sudo nova-rootwrap /etc/nova/rootwrap.conf iptables-save -c
C\xf3digo de salida: 1
Stdout: ''
Stderr: 'Traceback (most recent call last):\n File "/usr/bin/nova-rootwrap", line 59, in <module>\n from nova.rootwrap import wrapper\nImportError: No module named rootwrap\n'

2014-02-21 12:13:43.244 TRACE nova Traceback (most recent call last):

[…]

2014-02-21 12:13:43.244 TRACE nova ProcessExecutionError: Error inesperado al ejecutar orden.
2014-02-21 12:13:43.244 TRACE nova Orden: sudo nova-rootwrap /etc/nova/rootwrap.conf iptables-save -c
2014-02-21 12:13:43.244 TRACE nova C\xf3digo de salida: 1
2014-02-21 12:13:43.244 TRACE nova Stdout: ''
2014-02-21 12:13:43.244 TRACE nova Stderr: 'Traceback (most recent call last):\n File "/usr/bin/nova-rootwrap", line 59, in <module>\n   from nova.rootwrap import wrapper\nImportError: No module named rootwrap\n'
2014-02-21 12:13:43.244 TRACE nova

Solución: Por alguna razón intenta usar el directorio /usr/bin/nova-rootwrap por defecto. Hay que borrar este directorio y usará directamente /usr/local/bin

Errores asociados a Heat:

Cuidado al lanzar heat con devstack. Muchas páginas hablan de localrc. Con estas líneas no me funcionó. Para que funcionara seguí este enlace:

https://github.com/openstack-dev/devstack (Sección Heat).

Otro error:

Al lanzar un stack:

heat stack-create teststack -f WordPress_Single_Instance.template -P "InstanceType=m1.tiny;DBUsername=wp;DBPassword=verybadpassword;KeyName=heat_key; LinuxDistribution=F17"

ERROR: Property error: WikiDatabase: ImageId "F17-i386-cfntools" does not validate glance.image

Solución:

1) mirar el error por internet: no hay soluciones.
2) Mirar logs: no aportan mucha más información.
3) Postear en el foro de Openstack: no hubo respuesta.
4) Volver a bajar el repositorio de devstack: vemos que era un problema interno de Devstack.

Errores con las variables de entorno:

Si hay algún problema con las variables de entorno, pueden aparecer los siguientes errores:

keystone role-list
User 19cb45fae8c44b87beed5065dbb92508 is unauthorized for tenant admin (HTTP 401)

nova list
ERROR: Invalid OpenStack Nova credentials.

keystone role-list
Invalid user / password (HTTP 401)

Muy importante las variables de entorno: si no están o tienen algún problema de inconsistencia, los comandos de las APIs no funcionarán. Suele ocurrir cuando se han asignado mal las variables de entorno (por ejemplo, se intenta acceder a un *tenant* que el usuario no debe).

Para ello lo mejor es eliminar las variables de entorno y volveras a poner (quitar se hace asignando valor "nulo" : *export OS_TENANT_NAME=).*

Si un usuario no tiene el rol de admin, no puede hacer ciertas tareas. Por ejemplo, no podrá crear/listar nuevos usuarios:

keystone user-list
You are not authorized to perform the requested action, admin_required. (HTTP 403)

Errores con Swift:

En la documentación pone que con poner enable_service swift es suficiente. No funciona. Si se hace eso el script fallará.

Para configurarlo correctamente hay que añadir en el fichero local.conf: *enable_service s-proxy s-object s-container s-account*

Otro error: desde Horizon no deja borrar objetos. Hay que hacerlo desde línea de comandos:
Swift delete <nombreContenedor>

- ## Instalación manual OpenStack Havana en Ubuntu 12.04
  - ### Configuración Básica

1) Instalar NTP en el nodo controlador. El resto de nodos se sincronizarán su hora editando el fichero de configuración:
   - Controlador: *sudo apt-get install ntp*
   - Resto de nodos:
     - *sudo apt-get install ntp*
     - editar /etc/ntp.conf y cambiar el servidor para que apunte al controlador
2) Instalar MySQL en el nodo controlador, y MySQL cliente en el resto de nodos.
   - Controlador:
     - *sudo apt-get install python-mysqldb mysql-server*
     - Editar /etc/mysql/my.cnf para permitir acceso desde el exterior:
       [mysqld]
       ...
       bind-address = 0.0.0.0
     - reiniciar el servicio: *sudo service mysql restart*
   - Resto de nodos : *sudo apt-get install python-mysqldb*
3) Sincronizar paquetes OpenStack (Havana):
   sudo apt-get install python-software-properties
   sudo add-apt-repository cloud-archive:havana
   sudo apt-get update
   sudo apt-get dist-upgrade
   sudo reboot
4) Instalar el servidor de cola de mensajes (RabbitMQ):
   sudo apt-get install rabbitmq-server

Observación: Por defecto, usa el usuario "guest" que instala de forma predefinida. Para cambiar su contraseña:

sudo rabbitmqctl change_password guest RABBIT_PASS

Keystone

Instalamos keystone: *sudo apt-get install keystone*

Editamos /etc/keystone/keystone.conf y cambiamos en la sección [sql]:

connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone

Siendo el primer keystone: el nombre del usuario en MySQL (por ejemplo, podría ser root), la password para acceder con ese usuario (password de root), controller sería la IP del nodo controlador, y keystone sería el nombre de la tabla.

Por defecto, en la instalación de keystone se crea una SQLite. Borrarla: *sudo rm /var/lib/keystone/keystone.db*

Creamos la tabla "keystone" en la BD y damos privilegios al usuario creado para el acceso a su manipulación (tanto localmente como remotamente):

mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> GRANT ALL PRIVILEGES ON keystone. TO 'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';*
mysql> GRANT ALL PRIVILEGES ON keystone. TO 'keystone'@'%'IDENTIFIED BY 'KEYSTONE_DBPASS';*

OBS: el "keystone" que hay tras el "ON" es el nombre de la base de datos creada. El "keystone" que hay tras el "TO" es el nombre del usuario (en mi caso, root).

Sincronizamos el repositorio*: sudo keystone-manage db_sync*.

Ya podemos ver que se han creado las tablas en "keystone"

Creamos un *token* para manipular el servicio keystone como administrador (esto es sólo para manipular keystone al principio, ya que no tenemos ningún admin definido en el keystone. Cuando los definamos, debemos borrar este *token* y usar el admin). Editamos /etc/keystone/keystone.conf en la sección [DEFAULT] tal que así:

[DEFAULT]
 # A "shared secret" between keystone and other openstack services
admin_token = ADMIN_TOKEN

El "ADMIN_TOKEN" podemos generarlo mediante el siguiente comando: *openssl rand -hex 10*

Reiniciamos el servicio keystone: *service keystone restart*

Definimos usuarios, proyectos y roles:

Primero definimos las variables de entorno necesarias para acceder al keystone:

export OS_SERVICE_TOKEN=ADMIN_TOKEN
export OS_SERVICE_ENDPOINT=http://IP:35357/v2.0

Creamos *tenants*:

keystone tenant-create --name=admin --description="Admin Tenant"

keystone tenant-create --name=service --description="Service Tenant"

Creamos el usuario administrador:

keystone user-create --name=admin --pass=adminp --email=admin@admin.com

Creamos el rol de administrador. Los roles que se crean deben estar especificados en el fichero "policy.json" que se encuentra en los directorios de cada módulo. En este caso, podemos encontrarlo en /etc/keystone/.

keystone role-create --name=admin

Ahora asociamos usuario, rol, y tentant:

keystone user-role-add --user=admin --tenant=admin --role=admin

Definir servicios y API *endpoints*:

keystone service-create: describe un servicio
keystone endpoint-create: asocia API endpoints con un servicio

Vamos a registrar el servicio de autenticación. Para ello primero creamos el servicio keystone identity service:
keystone service-create --name=keystone --type=identity --description="Keystone Identity Service"

Este comando devolverá un ID, que usaremos para asociar al *endpoint*.
Y asociamos el servicio al *endpoint*:

keystone endpoint-create \
--service-id=fcc69784b2bf4ad4b21d75fa051f1021 \
--publicurl=http://192.168.122.19:5000/v2.0 \
--internalurl=http://192.168.122.19:5000/v2.0 \
--adminurl=http://192.168.122.19:35357/v2.0

Verificamos la instalacion:

Primero quitamos las variables de entorno definidas al principio que usamos para editar keystone cuando no tenía un usuario administrador:

unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT

Podemos crear un fichero .sh con las variables de entorno que utilizaremos para las autenticaciones. Creamos openrc.sh y editamos lo siguiente:

export OS_USERNAME=admin
export OS_PASSWORD=adminp
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://192.168.122.19:35357/v2.0

Para lanzarlo: *source openrc.sh*

Para comprobar que están las variables fijadas: *printenv* | *grep OS*

Verificamos el funcionamiento: *keystone token-get*. Este comando devuelve un ID. Si es así, se ha configurado correctamente. *keystone user-list* también nos puede ayudar a esta verificación.

Glance

Observación: debemos tener bien configurado keystone para configurar Glance.

Instalamos Glance: *sudo apt-get install glance python-glanceclient*

Configuramos la localización de la base de datos. El servicio de imágenes provee los servicios de "glance-api" y "glance-registry", ambas con sus propios ficheros de configuración. Editamos tanto /etc/glance/glance-api.conf y /etc/glance/glance-registry.conf, cambiando la sección [DEFAULT]:

```
[DEFAULT]
...
# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
sql_connection = mysql://root:rootp@192.168.122.19/glance
```

Borramos la BD SQLite que Glance crea por defecto: *sudo rm /var/lib/glance/glance.sqlite*

Creamos la BD glance, y le damos privilegios al usuario (en mi caso root) para poder acceder a dicha base de datos tanto localmente como desde el exterior:

```
# mysql -u root -p
mysql> CREATE DATABASE glance;
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'root'@'localhost' IDENTIFIED BY
'rootp';
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'root'@'%' IDENTIFIED BY 'rootp';
```

Sincronizamos la base de datos para que se creen las tablas que correspondan: *sudo glance-manage db_sync*

Creamos en keystone un usuario glance que se usará para autenticar glance contra keystone. Le daremos el rol de admin, y el *tenant* "service":

```
keystone user-create --name=glance --pass=glancep --email=glance@glance.com
keystone user-role-add --user=glance --tenant=service --role=admin
```

Configuramos Glance para que use Keystone como medio de autenticación. Editamos los ficheros de los dos servicios mencionados anteriormente, /etc/glance/glance-api.conf y /etc/glance/glance-registry.conf:

```
[keystone_authtoken]
...
auth_uri = http://192.168.122.19:5000
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = glancep

[paste_deploy]
...
flavor = keystone
```

Editamos los ficheros /etc/glance/glance-api-paste.ini y /etc/glance/glance-registry-paste.ini para añadirle los credenciales de autenticación contra keystone:

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
```

auth_host=192.168.122.19
admin_user=glance
admin_tenant_name=service
admin_password=glancep

Creamos el servicio y el *endpoint* en keystone:

keystone service-create --name=glance --type=image --description="Glance Image Service"
 # usamos el ID que nos devuelve para el endpoint
*keystone endpoint-create *
*--service-id=26132787dc17449c81efdd51fc7d224c *
*--publicurl=http://192.168.122.19:9292 *
*--internalurl=http://192.168.122.19:9292 *
--adminurl=http://192.168.122.19:9292

Reiniciamos los dos servicios:

sudo service glance-registry restart
sudo service glance-api restart

Verificamos la instalación:

Descargamos una imagen. En este caso, CirrOS, dado que es muy ligera (apenas 24MB):

mkdir images
cd images/
wget http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img

Subimos la imagen a Glance:
*glance image-create --name=cirrOS --disk-format=fileFormat *
--container-format=containerFormat --is-public=accessValue < imageFile

Donde:
- *fileFormat* lo podemos mirar con el comando file: file cirros-0.3.1-x86_64-disk.img. Nos dice que es una imagen con formato (hay diferentes formatos permitidos: qcow2, raw, vhd, vmdk, vdi, iso, aki, ari, and ami)
- *containerFormat*: se usa para especificar si la imagen tiene metadatos. Lo mejor es usar siempre "bare" (diferentes formatos: bare, ovf, aki, ari and ami).
- accessValue: true si queremos que todos los usuarios puedan ver la imagen, false en caso contrario.
- *imageFile*: nombre de la imagen descargada.

En mi caso: *glance image-create --name=cirrOS --disk-format=qcow2 --container-format=bare --is-public=true < cirros-0.3.1-x86_64-disk.img*

Podemos ver que se ha creado correctamente mediante *glance image-list*

Nova

Instalamos Nova en el nodo controlador:

sudo apt-get install nova-novncproxy novnc nova-api nova-ajax-console-proxy nova-cert nova-conductor nova-consoleauth nova-doc nova-scheduler python-novaclient

Editamos el fichero /etc/nova/nova.conf y añadimos las siguientes líneas para configurar la localización de la BD:

[database]

 # The SQLAlchemy connection string used to connect to the database
connection = mysql://root:rootp@192.168.122.19/nova
[keystone_authtoken]
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = novap

Configuramos Nova para que use el servicio RabbitMQ, añadiendo en el fichero anterior lo siguiente (en la sección [DEFAULT]):

rpc_backend = nova.rpc.impl_kombu
rabbit_host = 192.168.122.19
rabbit_password = rabbitmqp

Borramos la BD de SQLite que habrá creado por defecto en la instalación de los paquetes: *sudo rm /var/lib/nova/nova.sqlite*

Creamos la BD nova y le damos permisos para acceder desde el exterior:

 # mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON nova. TO 'root'@'localhost' IDENTIFIED BY 'rootp';*
mysql> GRANT ALL PRIVILEGES ON nova. TO 'root'@'%' IDENTIFIED BY 'rootp';*

Sincronizamos la BD para que cree las tablas en la BD nova: *sudo nova-manage db sync*

Editamos el fichero nova.conf añadiendo nuestra IP a my_ip, vncserver_listen, y vncserver_proxyclient_address. OBS: Openstack recomienda tener dos tarjetas de red, una cara al exterior, para administración y salida al exterior, y otra para la comunicación entre nodos, que sería interna. No obstante, se puede configurar todo como si estuviese directamente conectado con el exterior. Añadimos las siguientes líneas:

[DEFAULT]
...
my_ip=192.168.122.19
vncserver_listen=192.168.122.19
vncserver_proxyclient_address=192.168.122.19

Autenticación:

Creamos el usuario "nova", para autenticarse contra keystone, añadiendole el rol de admin y el *tenant service*:

keystone user-create --name=nova --pass=novap --email=nova@nova.com
keystone user-role-add --user=nova --tenant=service --role=admin

Editamos nova.conf añadiendo que la autenticación se hará a través de keystone:

[DEFAULT]
...
auth_strategy=keystone

Añadimos los credenciales para autenticarnos en el fichero /etc/nova/api-paste.ini:

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory

auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.122.19:5000/v2.0
admin_tenant_name = service
admin_user = nova
admin_password = novap

Registramos el servicio y el *endpoint* para que sea visible por otros módulos de Openstack:

keystone service-create --name=nova --type=compute --description="Nova Compute service"
keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://192.168.122.19:8774/v2/%\(tenant_id\)s \
--internalurl=http://192.168.122.19:8774/v2/%\(tenant_id\)s \
--adminurl=http://192.168.122.19:8774/v2/%\(tenant_id\)s

Observación: sustituir *the_service_id_above* por el ID que nos da el primer comando (*service-create*).

Reiniciamos los servicios:

sudo service nova-api restart
sudo service nova-cert restart
sudo service nova-consoleauth restart
sudo service nova-scheduler restart
sudo service nova-conductor restart
sudo service nova-novncproxy restart

Para nodos computacionales:

Primero deberemos tener todo bien configurado (NTP, red, MySQL, etc). De momento lo intentaremos lanzar sobre la misma máquina, así que esto no será necesario, sólo será necesario en el caso de hacer otro *compute node*.

Instalamos los paquetes necesarios (instalará entre otras cosas KVM para lanzar las instancias): *sudo apt-get install nova-compute-kvm python-guestfs*

Nos preguntará si queremos crear un "supermin appliance". Respondemos que sí.

Debido a un bug, hay que hacer el kernel leíble. Para ello hay que lanzar: *sudo dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-$(uname -r)*

Para que esto se mantenga en futuras actualizaciones, crear fichero /etc/kernel/postinst.d/statoverride y escribir:

#!/bin/sh
version="$1"
 # passing the kernel version is required
[-z "${version}"] && exit 0
dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-${version}

Hacemos dicho fichero ejecutable: *sudo chmod +x /etc/kernel/postinst.d/statoverride*

Editamos nova.conf, añadiendo el método de autenticación y la BD contra la que actuaremos (la creada en el nodo controlador):

[DEFAULT]
...
auth_strategy=keystone

...
[database]
 # The SQLAlchemy connection string used to connect to the database
connection = mysql://root:rootp@192.168.122.19/nova

Añadimos en nova.conf, en la parte de DEFAULT, el message broker (RabbitMQ):

rpc_backend = nova.rpc.impl_kombu
rabbit_host = 192.168.122.19
rabbit_password = rabbitmqp

Configuramos el nodo computacional para poder acceder remotamente a las instancias. Añadimos en nova.conf:

[DEFAULT]
...
my_ip=192.168.122.19
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=192.168.122.19
novncproxy_base_url=http://192.168.122.19:6080/vnc_auto.html

También debemos especificar el host donde corre el servidor de imágenes glance. En nuestro caso, es el mismo que el controlador:
[*DEFAULT]*
...
glance_host=192.168.122.19

Añadimos los credenciales de keystone en el fichero /etc/nova/api-paste.ini:

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = novap

Borramos la BD SQLite que crea por defecto: *sudo rm /var/lib/nova/nova.sqlite*

Rearrancamos el servicio: *sudo service nova-compute restart*

Observación: Borrar instancias en estado de error:

nova reset-state instancia01
nova delete instancia01

○ Neutron

Dado que la documentación es bastante enrevesada, primero detallamos el mapa de navegación a base de links:

Primero en el nodo controlador:
http://docs.openstack.org/havana/install-guide/install/apt/content/neutron-install-network-node.html

Segundo en el controlador: http://docs.openstack.org/havana/install-guide/install/apt/content/neutron-install.dedicated-network-node.html (a mitad de las

instrucciones habrá que hacer: http://docs.openstack.org/havana/install-guide/install/apt/content/install-neutron.install-plug-in.ovs.html
Y al final habrá que hacer las VLANs: http://docs.openstack.org/havana/install-guide/install/apt/content/install-neutron.install-plug-in.ovs.vlan.html)

Tercero en el controlador: http://docs.openstack.org/havana/install-guide/install/apt/content/install-neutron.dedicated-controller-node.html

Cuarto en los computadores: http://docs.openstack.org/havana/install-guide/install/apt/content/install-neutron.dedicated-compute-node.html

Quinto en el controller: http://docs.openstack.org/havana/install-guide/install/apt/content/install-neutron.configure-networks.html

Definir base de datos y acceso a Keystone:

mysql -u root -p
mysql> CREATE DATABASE neutron;
mysql> GRANT ALL PRIVILEGES ON neutron. TO 'root'@'localhost' IDENTIFIED BY 'rootp';*
mysql> GRANT ALL PRIVILEGES ON neutron. TO 'root'@'%' IDENTIFIED BY 'rootp';*

keystone user-create --name=neutron --pass=neutronp --email=neutron@neutron.com

keystone user-role-add --user=neutron --tenant=service --role=admin

keystone service-create --name=neutron --type=network --description="OpenStack Networking Service"

keystone endpoint-create \
--service-id the_service_id_above \
--publicurl http://192.168.122.19:9696 \
--adminurl http://192.168.122.19:9696 \
--internalurl http://192.168.122.19:9696

Instalar Open vSwitch (OVS):

sudo apt-get install neutron-plugin-openvswitch-agent

Configurar VLANs (no GRE):

Decimos a OVS que use VLANs:

Editamos /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini:

[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-eth0

Añadimos el bridge a OVS y asociamos la tarjeta de red eth0 al bridge que actuará de red exterior en OpenStack:

sudo ovs-vsctl add-br br-eth0
sudo ovs-vsctl add-port br-eth0 eth0

Editamos fichero /etc/network/interfaces fijando las IPs y asociando la IP al br-eth0

Instalamos la parte de Neutron servidor:

sudo apt-get install neutron-server neutron-dhcp-agent neutron-plugin-openvswitch-agent neutron-l3-agent

Editar /etc/sysctl.conf para configurar el enroutamiento:

net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0

Reiniciar para que se vean los cambios (la red o la máquina virtual entera).

Configurar autenticación de Neutron contra Keystone:

[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = neutron
admin_password = neutronp

Así como el acceso RabbitMQ (en neutron.conf):

[DEFAULT]
...
rabbit_host = 192.168.122.19
rabbit_userid = guest
rabbit_password = rabbitmqp

Y la conexión con la BD:

[database]
connection = mysql://root:rootp@controller/neutron

Editar /etc/neutron/api-paste.ini y añadir autenticación:

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 192.168.122.19
auth_uri = http://192.168.122.19:5000
admin_tenant_name = service
admin_user = neutron
admin_password = neutronp

Asociamos el bridge interno a OVS:

sudo ovs-vsctl add-br br-int

Editamos /etc/neutron/l3_agent.ini y /etc/neutron/dhcp_agent.ini

interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True

Editamos /etc/neutron/neutron.conf:

core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2

Editamos /etc/neutron/dhcp_agent.ini:

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
```

Editamos nova.conf:

```
[DEFAULT]
neutron_metadata_proxy_shared_secret = metadatap
service_neutron_metadata_proxy = true
```

Editamos /etc/neutron/metadata_agent.ini:

```
[DEFAULT]
auth_url = http://192.168.122.19:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = neutronp
nova_metadata_ip = 192.168.122.19
metadata_proxy_shared_secret = metadatap
```

Configurar firewall en /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini:

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Añadir en nova.conf:

```
network_api_class = nova.network.neutronv2.api.API
neutron_url = http://192.168.122.19:9696
neutron_auth_strategy = keystone
neutron_admin_tenant_name = service
neutron_admin_username = neutron
neutron_admin_password = neutronp
neutron_admin_auth_url = http://192.168.122.19:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.libvirt.firewall.IptablesFirewallDriver
```

Validación:

Creamos red externa y le asociamos, suponiendo que *br-eth0* tenga IP 10.0.0.1:

```
neutron net-create ext-net --shared --router:external=True
neutron subnet-create ext-net \
  --allocation-pool start=10.0.0.5,end=10.0.0.200 \
  --gateway=10.0.0.1 --enable_dhcp=True \
  10.0.0.0/24
```

Creamos un router que asociaremos a la red externa

```
neutron router-create myrouter
neutron router-gateway-set myrouter ext-net
```

Creamos red privada y le asociamos interfaz al router en la nueva subred:

```
neutron net-create mynet
neutron subnet-create --name mynet-subnet mynet 192.168.1.0/24
neutron router-interface-add myrouter mynet-subnet
```

Horizon

Instalamos paquetes necesarios: *sudo apt-get install memcached libapache2-mod-wsgi openstack-dashboard*

Borramos el tema que instala Ubuntu, para que nos aparezca el tema OpenStack: *sudo apt-get remove --purge openstack-dashboard-ubuntu-theme*

Con esto sería suficiente. 'Tenemos que tener en cuenta que el valor de 'LOCATION' del fichero /etc/openstack-dashboard/local_settings.py (en la parte de CACHES), tiene que ser igual tanto en IP como en puerto al valor que se encuentra en el fichero /etc/memcached.conf.

En el caso de que quisiésemos permitir a ciertas IPs el acceso a la interfaz, hay que modificar el valor de ALLOWED_HOSTS del fichero local_settings.py.

Si la interfaz la quisiésemos poner en otro servidor distinto al controlador, hay que modificar el valor OPENSTACK_HOST de local_settings.py: OPENSTACK_HOST = "controller"

Reiniciamos servicios:

sudo service apache2 restart
sudo service memcached restart

Entramos en http://192.168.122.19/horizon con los credenciales de los usuarios de keystone.

Cinder

Instalamos paquetes necesarios: *sudo apt-get install cinder-api cinder-scheduler*

Editamos /etc/cinder/cinder.conf. Lo más probable es que la sección database no exista, así que la añadimos al final:

[database]
...
connection = mysql://root:rootp@192.168.122.19/cinder

Creamos la base de datos "cinder" con los permisos cara al exterior:

 # mysql -u root -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL PRIVILEGES ON cinder. TO 'root'@'localhost' IDENTIFIED BY 'rootp';*
mysql> GRANT ALL PRIVILEGES ON cinder. TO 'root'@'%' IDENTIFIED BY 'rootp';*

Creamos las tablas: *sudo cinder-manage db sync*

Creamos el usuario cinder, que será el que use el servicio de almacenamiento de bloques (Cinder) para autenticarse frente al servicio de identidad (keystone).

keystone user-create --name=cinder --pass=cinderp --email=cinder@cinder.com
keystone user-role-add --user=cinder --tenant=service --role=admin

Añadimos los credenciales en /etc/cinder/api-paste.ini:

[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=192.168.122.19
auth_port = 35357

auth_protocol = http
auth_uri = http://192.168.122.19:5000
admin_tenant_name=service
admin_user=cinder
admin_password=cinderp

Editamos /etc/cinder/cinder.conf para definir el uso de RabbitMQ:

[DEFAULT]
...
rpc_backend = cinder.openstack.common.rpc.impl_kombu
rabbit_host = 192.168.122.19
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = rabbitmqp

Creamos servicio y *endpoint* (de la versión 1 y 2):
keystone service-create --name=cinder --type=volume --description="Cinder Volume Service"
keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://192.168.122.19:8776/v1/%\(tenant_id\)s \
--internalurl=http://192.168.122.19:8776/v1/%\(tenant_id\)s \
--adminurl=http://192.168.122.19:8776/v1/%\(tenant_id\)s

keystone service-create --name=cinderv2 --type=volumev2 --description="Cinder Volume Service V2"
keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://192.168.122.19:8776/v2/%\(tenant_id\)s \
--internalurl=http://192.168.122.19:8776/v2/%\(tenant_id\)s \
--adminurl=http://192.168.122.19:8776/v2/%\(tenant_id\)s

Reiniciamos servicios para aplicar las nuevas configuraciones:

sudo service cinder-scheduler restart
sudo service cinder-api restart

Después de configurar el nodo controlador, configuramos el servicio de almacenamiento de bloques (esto se puede hacer en un nuevo nodo que contendrá el disco que sirve los volúmenes, pero para pruebas lo haré en el nodo controlador).

Si hacemos un segundo nodo, hay que tener cuidado con que estén sincronizados los nodos por NTP.

Instalamos LVM (lo normal es que esté ya instalado):

sudo apt-get install lvm2

Creamos en la máquina virtual un segundo disco que usaremos para Cinder (vdb).

Después creamos el volumen físico y lógico:

sudo pvcreate /dev/vdb
sudo vgcreate cinder-volumes /dev/vdb

Instalamos cinde-volume (necesario para nodos Cinder computacionales): *sudo apt-get install cinder-volume*

En nuestro caso, ya no se necesitará seguir ningún paso más. En el caso de que tengamos Cinder en otro nodo, debemos indicar cuál es el nodo controlador al que sirve.

Swift

Credenciales Keystone (omitimos creación base de datos ya que es idéntico al resto de servicios con la salvedad de cambiar los nombres de los servicios por Swift):

```
keystone user-create --name=swift --pass=swiftp --email=swift@swift.com
keystone user-role-add --user=swift --tenant=service --role=admin
keystone service-create --name=swift --type=object-store --description="Object Storage Service"

keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl='http://192.168.122.19:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl='http://192.168.122.19:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl=http://192.168.122.19:8080
```

Creamos directorio Swift:

```
sudo mkdir -p /etc/swift
```

Crear /etc/swift/swift.conf y añadir:

```
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix = fLIbertYgibbitZ
```

```
sudo apt-get install swift-account swift-container swift-object xfsprogs
```

Creamos otro disco (vdc), con extensión XFS y lo montamos:

```
sudo mkfs.xfs /dev/vdc
sudo echo "/dev/vdc /srv/node/vdc xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
sudo mkdir -p /srv/node/vdc
sudo mount /srv/node/vdc
sudo chown -R swift:swift /srv/node
```

Creamos el fichero /etc/rsyncd.conf con lo siguiente:

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.122.19
[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock
[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock
[object]
max connections = 2
```

path = /srv/node/
read only = false
lock file = /var/lock/object.lock

Editamos /etc/default/rsync: *RSYNC_ENABLE=true*

Iniciamos servicio rsync: *sudo service rsync start*

Creamos directorio caché y establecemos permisos:

sudo mkdir -p /var/swift/recon
sudo chown -R swift:swift /var/swift/recon

Instalación y configuración del nodo proxy (en el controlador todo):

sudo apt-get install swift-proxy memcached python-keystoneclient python-swiftclient python-webob

Cambiamos en el fichero /etc/memcached.conf*: -l 127.0.0.1 por -l 192.168.122.19* (en nuestro caso da igual, porque es el mismo nodo).

Reiniciamos servicio: *sudo service memcached restart*

Heat

Instalamos paquetes necesarios: *sudo apt-get install heat-api heat-api-cfn heat-engine*

Definimos dónde está la base de datos, la creamos y damos permisos:

En /etc/heat/heat.conf:

[DEFAULT]
 # The SQLAlchemy connection string used to connect to the database
sql_connection = mysql://root:rootp@192.168.122.19/heat

 # mysql -u root -p
mysql> CREATE DATABASE heat;
mysql> GRANT ALL PRIVILEGES ON heat. TO 'root'@'localhost' IDENTIFIED BY 'rootp';*
mysql> GRANT ALL PRIVILEGES ON heat. TO 'root'@'%' IDENTIFIED BY 'rootp';*

Borramos la BD SQLite: *sudo rm /var/lib/heat/heat.sqlite*

Cremaos las tablas: *sudo heat-manage db_sync*

Editamos heat.conf para modificar valores para activar *login*:

[DEFAULT]
...
 # Print more verbose output (set logging level to INFO instead
# of default WARNING level). (boolean value)
verbose = True
...
 # (Optional) The base directory used for relative --log-file
 # paths (string value)
log_dir=/var/log/heat

Configuramos para usar RabbitMQ, en heat.conf:

[DEFAULT]
...
rabbit_host = 192.168.122.19
rabbit_password = rabbitmqp

Autenticación: crear usuario, servicio, *endpoint*, y editar heat.conf para definir autenticación:

keystone user-create --name=heat --pass=heatp --email=heat@heat.com
keystone user-role-add --user=heat --tenant=service --role=admin

[keystone_authtoken]
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.122.19:5000/v2.0
admin_tenant_name = service
admin_user = heat
admin_password = heatp
[ec2_authtoken]
auth_uri = http://192.168.122.19:5000/v2.0
keystone_ec2_uri = http://192.168.122.19:5000/v2.0/ec2tokens

keystone service-create --name=heat --type=orchestration --description="Heat Orchestration API"

*keystone endpoint-create *
*--service-id=the_service_id_above *
*--publicurl=http://192.168.122.19:8004/v1/%\(tenant_id\)s *
*--internalurl=http://192.168.122.19:8004/v1/%\(tenant_id\)s *
--adminurl=http://192.168.122.19:8004/v1/%\(tenant_id\)s

keystone service-create --name=heat-cfn --type=cloudformation --description="Heat CloudFormation API"

*keystone endpoint-create *
*--service-id=the_service_id_above *
*--publicurl=http://192.168.122.19:8000/v1 *
*--internalurl=http://192.168.122.19:8000/v1 *
--adminurl=http://192.168.122.19:8000/v1

Reiniciamos servicios:

sudo service heat-api restart
sudo service heat-api-cfn restart
sudo service heat-engine restart

Ejemplo de uso:

Copiar lo siguiente en la interfaz (Stacks -> Launch Stack -> direct input):

 # This is a hello world HOT template just defining a single compute instance
heat_template_version: 2013-05-23

description: >
Hello world HOT template that just defines a single compute instance.
Contains just base features to verify base HOT support.

parameters:
KeyName:
type: string

```
    description: Name of an existing key pair to use for the instance
  InstanceType:
    type: string
    description: Instance type for the instance to be created
    default: m1.small
    constraints:
      - allowed_values: [m1.tiny, m1.small, m1.large]
        description: Value must be one of 'm1.tiny', 'm1.small' or 'm1.large'
  ImageId:
    type: string
    description: ID of the image to use for the instance
  # parameters below are not used in template, but are for verifying parameter
  # validation support in HOT
  db_password:
    type: string
    description: Database password
    hidden: true
    constraints:
      - length: { min: 6, max: 8 }
        description: Password length must be between 6 and 8 characters
      - allowed_pattern: "[a-zA-Z0-9]+"
        description: Password must consist of characters and numbers only
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: Password must start with an uppercase character
  db_port:
    type: number
    description: Database port number
    default: 50000
    constraints:
      - range: { min: 40000, max: 60000 }
        description: Port number must be between 40000 and 60000

resources:
  my_instance1:
    # Use an AWS resource type since this exists; so why use other name here?
    type: AWS::EC2::Instance
    properties:
      KeyName: { get_param: KeyName }
      ImageId: { get_param: ImageId }
      InstanceType: { get_param: InstanceType }

  my_instance2:
    # Use an AWS resource type since this exists; so why use other name here?
    type: AWS::EC2::Instance
    properties:
      KeyName: { get_param: KeyName }
      ImageId: { get_param: ImageId }
      InstanceType: { get_param: InstanceType }

outputs:
  instance_ip1:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance1, PublicIp] }
  instance_ip2:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance2, PublicIp] }
```

Rellenar el formulario: Observación: para que funcione, la contraseña de la base de datos tiene que ir en mayúsculas. Si hay algún problema, mirar logs.

Ceilometer

Instalar paquetes necesarios: *sudo apt-get install ceilometer-api ceilometer-collector ceilometer-agent-central python-ceilometerclient*

Ceilometer, dado que tiene que procesar rápidamente miles de datos, usa una base de datos no relacional (MongoDB) que pasaremos a instalar: *sudo apt-get install mongodb*

Editar /etc/mongodb.conf:

bind_ip = 0.0.0.0

Reiniciamos servicio para que establezca nueva configuración: *sudo service mongodb restart*

Observación: para enganchar con MongoDB: lanzar el servicio de este modo (indicando el path donde va a almacenar los datos y diciéndole que ocupe menos espacio):

sudo mongod --dbpath=/var/lib/mongodb/data/db --smallfiles &

Creamos la base de datos y el usuario de la BD:

```
mongo --host controller
use ceilometer
db.addUser( { user: "ceilometer",
 pwd: "ceilometerp",
 roles: [ "readWrite", "dbAdmin" ]
} )
```

Editar /etc/ceilometer/ceilometer.conf:

```
[database]
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection = mongodb://ceilometer:ceilometer@192.168.122.19:27017/ceilometer
```

Editar /etc/ceilometer/ceilometer.conf y añadir:

```
[publisher_rpc]
# Secret value for signing metering messages (string value)
metering_secret = ADMIN_TOKEN
```

Siendo ADMIN_TOKEN generado con el siguiente comando: *openssl rand -hex 10*

Acceso a RabbitMQ (ceilometer.conf):

```
rabbit_host = controller
rabbit_password = rabbitmqp
```

Decimos dónde queremos que se depositen los logs (ceilometer.conf):

```
[DEFAULT]
log_dir = /var/log/ceilometer
```

Autenticación:

```
keystone user-create --name=ceilometer --pass=ceilometerp --
email=ceilometer@ceilometer.com
keystone user-role-add --user=ceilometer --tenant=service --role=admin
```

En ceilometer.conf:

[keystone_authtoken]
auth_host = 192.168.122.19
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.122.19:5000
admin_tenant_name = service
admin_user = ceilometer
admin_password = ceilometerp

[service_credentials]
os_username = ceilometer
os_tenant_name = service
os_password = ceilometerp

Creamos servicio y *endpoint* en Keystone:

keystone service-create --name=ceilometer --type=metering --description="Ceilometer Telemetry Service"

keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://192.168.122.19:8777 \
--internalurl=http://192.168.122.19:8777 \
--adminurl=http://192.168.122.19:8777

Reiniciamos servicios:

sudo service ceilometer-agent-central restart
sudo service ceilometer-api restart
sudo service ceilometer-collector restart

Instalamos la parte computacional: *sudo apt-get install ceilometer-agent-compute*

Añadimos en /etc/nova/nova.conf:

[DEFAULT]
...
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = nova.openstack.common.notifier.rpc_notifier
notification_driver = ceilometer.compute.nova_notifier

Reiniciamos servicio:

sudo service ceilometer-agent-compute restart

Añadir servicio de imágenes a la telemetría:

En glance-api.conf añadir:

notifier_strategy = rabbit
rabbit_host = 192.168.122.19
rabbit_password = rabbitmqp

Añadir servicio de almacenamiento de bloques a la telemetría:

En /etc/cinder/cinder.conf añadir en DEFAULT:

control_exchange = cinder

notification_driver = cinder.openstack.common.notifier.rpc_notifier

Reiniciamos servicio:

sudo service cinder-volume restart
sudo service cinder-api restart

Añadir servicio de almacenamiento de objetos a la telemetría:

keystone role-create --name=ResellerAdmin
keystone user-role-add --tenant service --user ceilometer --role ID_DEL_COMANDO_ANTERIOR

Editar /etc/swift/proxy-server.conf:

[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth ceilometer proxy-server

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

- ## RackSpace: alta disponibilidad

Rackspace Private *cloud* tiene la capacidad de implementar el soporte para alta disponibilidad de todos los componentes Nova y APIs, Cinder, Keystone y Glance, así como del scheduler, RabbitMQ y MySQL. La alta disponibilidad está sustentada gracias a *Keepalived* y *HAProxy* .

Rackspace Private *cloud* utiliza los siguientes métodos para implementar alta disponibilidad en el clúster:

Replicación maestro-maestro de MySQL y tolerancia a fallos activo-pasivo: MySQL está instalado en ambos nodos del controlador, y la replicación maestro-maestro se configura entre los nodos. *Keepalived* gestiona las conexiones entre los dos nodos, de manera que sólo un nodo recibe peticiones de lectura/escritura en cualquier momento.

Tolerancia a fallos activo-pasivo de RabbitMQ: RabbitMQ está instalado en ambos nodos del controlador. *Keepalived* gestiona las conexiones entre los dos nodos, de manera que sólo un nodo recibe peticiones de lectura/escritura en cualquier momento.

Equilibrio de carga de la API: Todos los servicios que son sin estado y pueden equilibrar la carga (esencialmente todas las APIs y algunos otros) se han instalado en ambos nodos del controlador. *HAProxy* es entonces instalado en ambos nodos, y *Keepalived* administra las conexiones a *HAProxy*, lo que hace a *HAProxy* tener alta disponibilidad. Los *endpoints* de Keystone y todos los accesos a la API pasan a través de *Keepalived*.

Zonas de disponibilidad:

Las zonas de disponibilidad permiten gestionar y aislar diferentes nodos en el entorno. Por ejemplo, es posible que se desee aislar diferentes conjuntos de nodos de computación para proporcionar diferentes recursos a los clientes. Si una zona de disponibilidad experimenta tiempos de inactividad, otras zonas del clúster no se verán afectadas.

Cuando se crea un clúster Nova, es creado en una zona de disponibilidad por defecto, y todos los nodos están asignados a esa zona. Se pueden crear zonas de disponibilidad adicionales dentro del clúster según sea necesario.

- Puppet: Arquitectura
    - Arquitectura Cliente – Servidor

Servidor: envía, periódicamente configuraciones a todas las máquinas requeridas con aspectos personalizados por grupos o individuales. Utilizan protocolos cifrados para seguridad en envío de configuraciones. Es opcional en muchas de las herramientas.

Clientes: ejecutan periódicamente (ej. 5 minutos) su configuración. Establece lo mínimo que puede ponerse en funcionamiento.

En la *figura 16*, vemos una topología propia de Puppet, en la que tenemos el servidor Puppet en la parte superior con los recursos necesarios (base de datos, lista de nodos involucrados, etc.) y en la parte inferior los clientes, que tendrán un agente (observador) que se encargará de vigilar que la configuración de la máquina en la que está es correcta. En caso de que el servidor cambie la información, serán los agentes los que se encarguen de darse cuenta del cambio y de actuar en consecuencia.

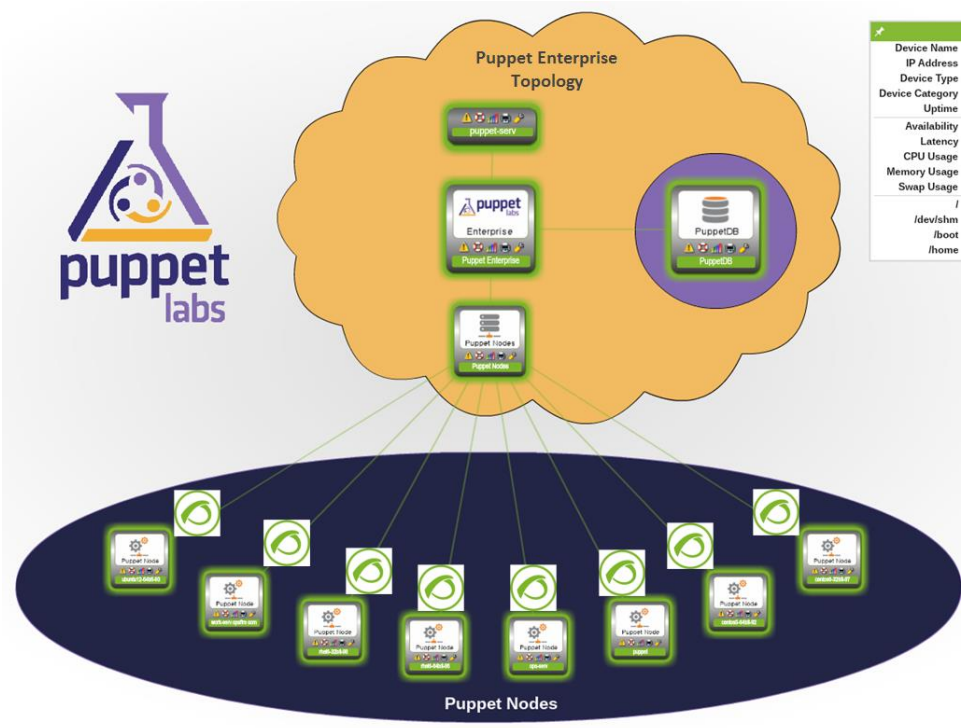

Figura 16: topología Puppet cliente-servidor

- Arquitectura usada en este proyecto

Dado que nuestro proyecto no requiere esta infraestructura, se puede ejecutar localmente cada manifiesto, es decir, que no dependan de un servidor Puppet. Cada nodo aplicará los manifiestos Puppet que crea necesario, y se ejecutará una vez. El comando para lanzarlo sigue la siguiente estructura:

puppet apply [nombre-manifiesto].pp

Se podría programar que se ejecutase el manifiesto cada X tiempo, por ejemplo, definiendo el comando en el cron del sistema, y se conseguiría un resultado similar a si tuviésemos un servidor. En nuestro caso, no es necesario definirlo en el cron.

Anexo 3: Aspectos adicionales OpenStack

Cambios versión OpenStack Icehouse

Aproximadamente 350 nuevas características centradas en testeo y estabilidad del *release*. La lista entera se puede encontrar en http://status.openstack.org/release/

2.902 bugs corregidos.

Nuevos idiomas añadidos en la interfaz Horizon (16 en total).

En Keystone se añade identidad federada (CERN).

DaaS (*Database as a service*).

Almacenamiento en bloque escalonado.

AMQP

AMQP (*Advanced Message Queuing Protocol*) es un protocolo estándar utilizado para sistemas de comunicación. Está orientado a mensajes y define las características necesarias para encolamiento, enrutamiento, exactitud y seguridad en los mensajes. AMQP define el comportamiento que debe seguir tanto el servidor de mensajería como el cliente.

OpenStack utiliza esta tecnología de mensajes a través de su implementación mediante RabbitMQ (es el principal que se suele utilizar, con lo cual nos centraremos en él).

Los componentes Nova utilizan llamadas a procedimiento remoto (RPC) para comunicarse entre sí. Sin embargo, este paradigma se construye sobre el paradigma publicación/suscripción que ofrece RabbitMQ.

Por consiguiente, RabbitMQ es un middleware de mensajería que implementa el protocolo AMQP.

Lista Hipervisores

- Baremetal
- Docker
- Hyper-V
- Kernel-based Virtual Machine (KVM)
- Linux Containers (LXC)
- Quick Emulator (QEMU)
- User Mode Linux (UML)
- VMWare vSphere
- Xen

○ Nova-Network

Existen tres formas distintas de gestionar la red a través de nova-network: FlatManager, FlatDHCPManager, VlanManager.

FlatManager y FlatDHCPManager son muy parecidas. Ambas se basan en un concepto de red a modo bridge, con un único bridge de entrada/salida. Como su nombre indica, se crea una red plana, todas las instancias están en la misma subred.

FlatManager proporciona el conjunto más sencillo de operaciones. Su objetivo se basa en asociar la instancia con el bridge. Por defecto, no hace configuración alguna sobre la IP asociada a la instancia. Esta tarea se deja para el administrador, haciendo uso de servidores DHCP externos u otros medios. En la *figura 17* se ve detallado lo citado, 2 instancias (vm_1 y vm_2) conectadas directamente a través del bridge (br100), con salida al exterior, siendo el DHCP externo el que provea la información necesaria de configuración de red.

FlatDHCPManager realiza la misma función, con la salvedad de que provee un servidor DHCP interno. Un proceso servidor DHCP *dnsmasq* se genera y escucha en la IP de la interfaz de puente. Es decir, los hosts que tienen el servicio nova-network actúan como router para las instancias. Podemos ver esto en la *figura 18*.

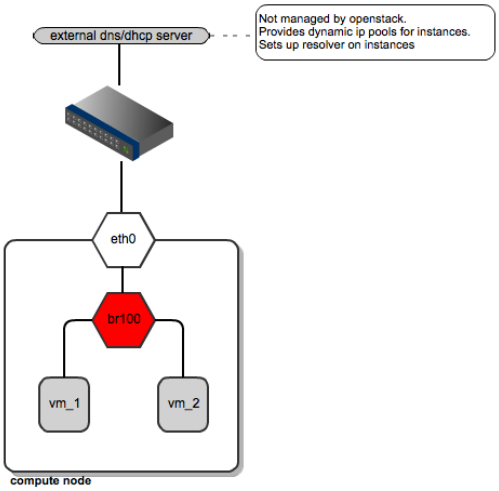

Figura 17: FlatManager

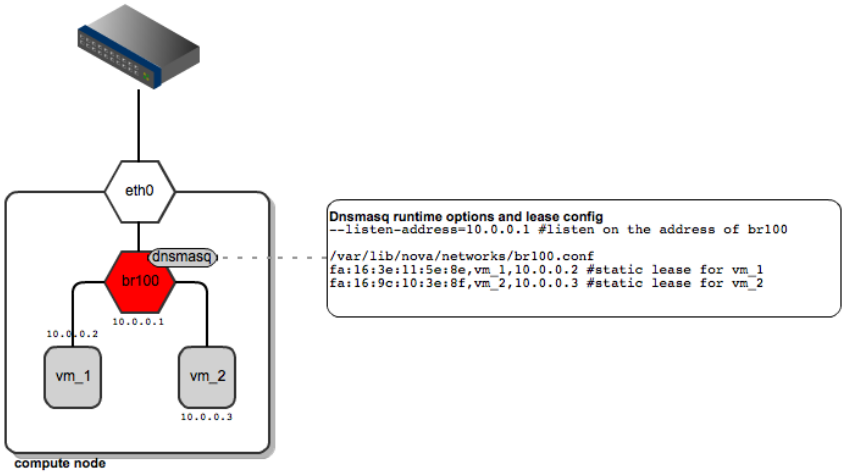

Figura 18: FlatDHCPManager

Si bien los dos anteriores están orientados para despliegues a escala simple, VlanManager es una buena opción para los *clouds* internos a gran escala y los *clouds* públicos. Como su nombre lo indica, VlanManager se basa en el uso de redes VLAN ("LAN virtuales"). El propósito de las VLAN es dividir una red física en dominios de difusión distintos (de manera que los grupos de hosts que pertenecen a diferentes VLAN no pueden verse entre sí). Como se ve en la *figura 19*, podemos crear diferentes VLANs para que los *tenants* tengan sus comunicaciones independientes al resto de *tenants*.

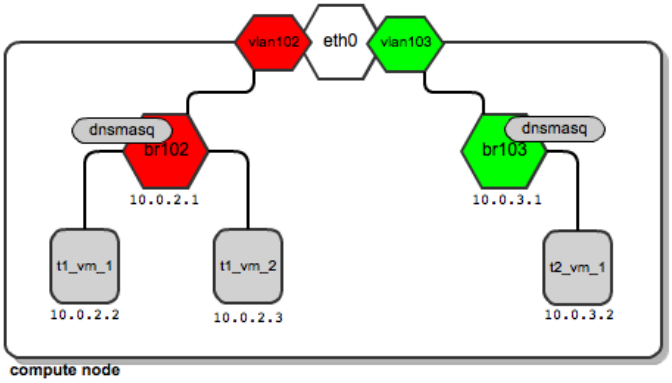

Figura 19: VlanManager

○ Open VSwitch

Open VSwitch es un sistema de switch virtual, diseñado específicamente para habilitar automatización y despliegue de interfaces de red de manera programática

OpenVSwitch permite más capacidades que los módulos regulares del *kernel* Linux, contando con características como QoS, LACP, etc. Es el modo de gestión de redes virtuales en soluciones como OpenNebula, OpenStack y XenCenter (XCP).

El sistema OpenStack que vamos a desplegar utiliza OpenVSwitch del modo que vemos en la *figura 20*:

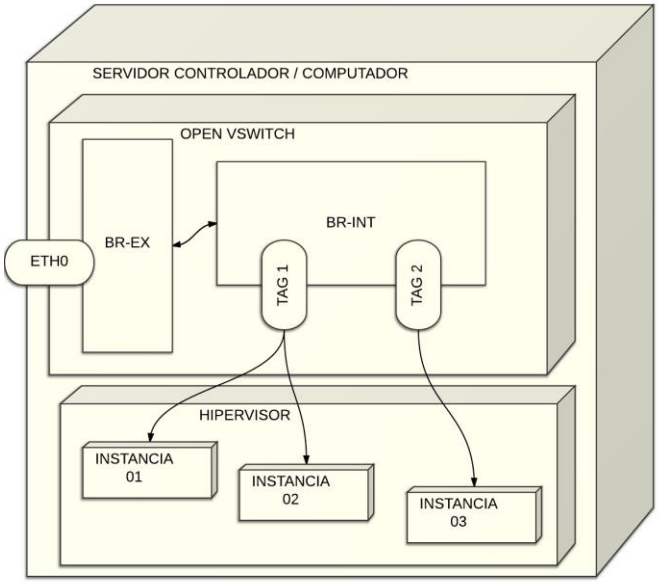

Figura 20: esquema OVS

Podemos observar que tiene asociado dos bridges, uno para la red interna y otro para la red externa. Además, estará asociado a su vez con la interfaz de red eth0, que permitirá tener acceso al exterior. Las instancias de los *tenants* tendrán a su vez puertos comunicados con el bridge interno, que estarán etiquetados (VLAN) para diferenciar el tráfico.

iSCSI

iSCSI (*internet SCSI*) es un estándar que permite usar el protocolo SCSI (*Small Computer System Interface*) sobre redes TCP/IP para la transferencia de datos (es decir, para almacenamiento).

LVM

LVM (Logical Volumen Manager): La gestión de volúmenes lógicos proporciona una vista de alto nivel sobre el almacenamiento, en vez discos y particiones. LVM permite redimensionar y mover los volúmenes a nuestra voluntad. En particular en Cinder, tenemos un grupo volumen físico, el cual tendrá asociado varias unidades de almacenamiento, y sobre él se crearán unidades de volumen lógicas, que serán creadas y destruidas como el usuario OpenStack decida (es lo que se pone a disposición del usuario final).

Ceph

Ceph es una plataforma de almacenamiento diseñado para presentar almacenamiento de bloques, archivos y objetos en un clúster distribuido. Los principales objetivos de Ceph son: evitar cualquier punto de fallo, escalabilidad y tolerante a errores a través de la replicación de los datos.

Como vemos en la *figura 21,* Ceph provee tanto servicio de almacenamiento de bloques como de objetos. Por ello puede ser utilizado por todos los servicios de almacenamiento de OpenStack. Es por esto que está cogiendo tanta fuerza, dado que con una implementación podemos tener el *backend* de todos los servicios, añadiendo además otro tipo de aspectos como alta disponibilidad.

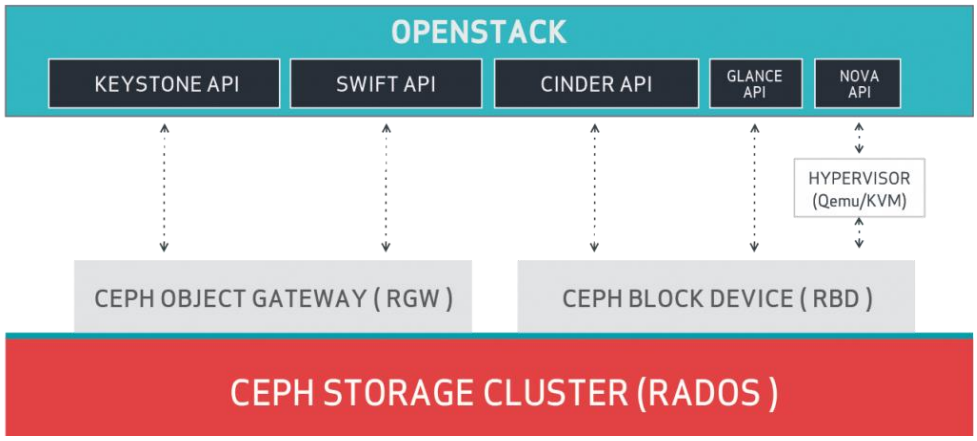

Figura 21: OpenStack y Ceph

Anexo 4: Manual del administrador

- Manual del administrador

Antes de empezar

Instalar fabric y la librería python-yaml en la máquina desde donde se lanzará todo el despliegue.

Creación y configuración del entorno

Crearemos una máquina virtual que actuará de nodo controlador, a la que llamaremos "controller", y dos nodos dedicados a computación, que llamaremos "comp1" y "comp2". Todas ellas tendrán 5GB de almacenamiento. "controller" tendrá a su vez 4GB de RAM y los computacionales ("comp1" y "comp2") tendrán 2GB de RAM. Además, el nodo "controller" tendrá asociado un disco externo virtual, de tamaño 3GB (para pruebas es suficiente, será usado para almacenamiento Cinder).

Ambas máquinas deberán tener el mismo usuario y contraseña de *login*, y poder realizar operaciones con privilegios de administrador (sudo).

"controller": editaremos el fichero /etc/network/interfaces para añadir de forma estática la IP, máscara, red y router por defecto. Dado que estamos en una configuración NAT, libvirt suele tener asociada la IP 192.168.122.1 como router por defecto. Esto se puede comprobar ejecutando "ifconfig" en la máquina anfitriona:

```
virbr0  Link encap:Ethernet  HWaddr fe:54:00:64:29:e3
        inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:50588 errors:0 dropped:0 overruns:0 frame:0
TX packets:210748 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:4054313 (4.0 MB)  TX bytes:1799349173 (1.7 GB)
```

Al "controller" le asociamos la IP = 192.168.122.200, y estará en la subred 192.168.122.0/24 (es decir, network 192.168.122.0 y máscara 255.255.255.0).

Editamos el fichero /etc/hosts para definir la resolución de nombres interna. En nuestro caso:

```
192.168.122.200         controller
192.168.122.201         comp1
192.168.122.202         comp2
```

Editaremos también el fichero /etc/hostname para cambiar el nombre de la máquina (le pondremos como nombre "controller").

En los nodos computacionales ("comp1" y "comp2) realizaremos los mismos pasos, asociando a comp1 la IP = 192.168.122.201 y a comp2 = 192.168.122.202.

En las 3 máquinas virtuales vamos a instalar ntp y openssh-server si no viene instalado por defecto.

Reiniciamos las 3 máquinas y comprobamos que tenemos acceso desde el host anfitrión al resto de máquinas (probamos a conectarnos a través de ssh en todas las máquinas).

Despliegue de OpenStack

En el directorio "OpenstackDespliegue", editamos el fichero "configOpenstackUser.yaml" tanto las IPs como el nombre, contraseña y nombre del proyecto que creamos conveniente. El usuario y contraseña deberá ser el de acceso a la máquina virtual, ya que se utilizarán esas credenciales para acceder a las máquinas. El *tenant* puede ser aleatorio, pero se deberá tener cuidado de que no exista en el "home" del usuario un directorio con el mismo nombre que el asignado al *tenant*, ya que el despliegue crea una carpeta con el nombre del *tenant* para alojar

ahí toda la información necesaria (módulos Puppet a utilizar y ficheros de texto usados para Facter). El rango de IPs que asociemos en el campo "computer" de nova deberá ser tal que así: "computer: 192.168.122.200-202", mientras que en el de cinder sólo utilizaremos como nodo de almacenamiento el controlador, luego: "computer: 192.168.122.200-200".

Una vez guardado el fichero con los cambios, pasaremos a ejecutar el script "run.sh" (sh run.sh). Esperamos a que se realice el despliegue (puede llevar bastante tiempo dependiendo de la conexión de red).

Despliegue posterior

Reiniciamos TODAS una vez terminado el despliegue (sudo shutdown -r now).

Entramos en la máquina virtual "controller" y en la ruta $HOME/nombreTenant/afterDeployment podemos encontrar scripts de ejemplo para el despliegue. Landamos con privilegios de administración (con sudo) el script "trasDespliegue.sh", con los siguientes parámetros: nombreUsuario passUsuario nombreTenantUsuario directorioDisco. Los tres primeros son para crear un nuevo usuario, con la contraseña fijada en el segundo parámetro y crear el *tenant* que queramos asociar al nuevo usuario. El cuarto parámetro se utilizará para crear un grupo de volúmenes físico, usado para Cinder.

Suponiendo que se le ha añadido un segundo disco a la máquina virtual, vamos a comprobar que se ha creado correctamente:

Para ver la ruta del nuevo disco virtual y otros aspectos, podemos ejecutar el siguiente comando:

```
sudo lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT,LABEL
```

Con "pvs" podemos ver que se ha creado el grupo correctamente.

Reinicamos la máquina "controller" (sudo shutdown -r now).

En los nodos computacionales, debido a un error, debemos pausar el servicio neutron-server (sudo service neutron-server stop).

Validación de la instalación

En el nodo "controller", ejecutamos ifconfig: debemos ver que se ha creado dos bridges: br-int y br-tun. br-tun a su vez deberá contener la IP "192.168.122.200".

Definimos las variables de entorno ejecutando el siguiente comando desde el home del usuario: source $HOME/openstack/openrc.sh

Ejecutamos sudo nova-manage service list. Debemos ver que los servicios tienen una cara sonriente en el estado ":-)" y que además podemos ver 3 nodos "nova-compute", cada uno de ellos con un nombre de host distinto (controller, comp1 y comp2).

Ejecutamos neutron agent-list. Todos deberían tener la cara sonriente en el estado. Si vemos XXX, podemos esperar unos minutos y volver a comprobar el estado (puede tardar un tiempo hasta que entra en el estado correcto).

Entramos a la interfaz de usuario OpenStack: 192.168.122.200/horizon. Hacemos *login* con el usuario creado con el script.

Creamos una red interna en la sección "network", por ejemplo podemos llamarla net-int y como nombre de subred subnet-int. La subred podría estar por ejemplo en el rango 192.168.1.0/24.

Creamos un router y definimos mediante el botón "set gateway" la red externa (es decir, deberá tener acceso al exterior). Añadimos como interfaz de red una conexión a nuestra red interna. Por defecto le establecerá la IP 192.168.1.1 (siguiendo nuestro ejemplo).

Dentro de la red interna, lanzaremos una instancia tipo CirrOS, con el nombre que queramos (por ejemplo "inst01") y que esté en la subred 192.168.1.0/24. El DHCP se encargará de definirle una IP. Si todo ha ido bien, deberá aparecer tras unos segundos de despliegue el estado ACTIVE.

Para comprobar dónde se ha creado, ejecutamos "virsh list" en cada máquina virtual. Aquella que veamos un ID asociado a un estado "running" será el host de la instancia lanzada.

Podemos acceder a la instancia a través de la consola que ofrece OpenStack desde la propia web.

Podemos crear también un volumen de ejemplo. Hay que asignar un tamaño al nuevo volumen inferior al que tengamos en total en el grupo que hayamos creado (podemos ver el tamaño total con el comando "pvg"). Al crearlo, podemos ver que entra en estado disponible. Con el comando "lvs" ejecutado en el nodo "controller", vemos que se ha creado correctamente el volumen lógico.

Salida al exterior

Pasaremos a convertir nuestra máquina virtual "controller" en un router, para que las instancias tengan conectividad al exterior. Para ello, en el nodo "controller", editamos el fichero /etc/sysctl.conf y definimos:

```
net.ipv4.ip_forward=1

net.ipv4.conf.all.rp_filter=0

net.ipv4.conf.default.rp_filter=0
```

Guardamos y ejecutamos: sudo sysctl -p

Además, debemos cambiar todos los paquetes que salgan de las instancias al exterior con la IP del nodo controlador, es decir, con la IP 192.168.122.200 (ya que sino saldrían con la IP por ejemplo 10.0.0.5 y a la hora de realizar la devolución, el router 192.168.122.1 no sabría a quién enviar el paquete). Luego con esto todos los paquetes saldrían con IP = 192.168.122.200 y sí que sabrían volver al destino con los paquetes de respuesta. Esto lo hacemos ejecutando:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Observación: si se reinicia la máquina, habría que volver a lanzar este comando.

Probamos la conectividad entrando en una instancia y ejecutando desde esa instancia por ejemplo ping 8.8.8.8 (ping a un DNS de Google).

- Fichero de configuración del administrador

```
---
admin:  #user y password de las máquinas donde desplegamos los servicios.
        #también son el user, password y tenant del administrador OpenStack.
        #path donde se almacenarán los ficheros en las máquinas remotas: $HOME/openstack
   user: as2admin
   password: as2
   tenant: openstack

mysql:
   ip: 192.168.122.200
   root_password: toor
   keystone:
      user: keystoneu
      password: keystonep
      db_name: keystone
   glance:
      user: glanceu
      password: glancep
      db_name: glance
   nova:
      user: novau
      password: novap
      db_name: nova
   cinder:
      user: cinderu
      password: cinderp
      db_name: cinder
   neutron:
      user: neutronu
      password: neutronp
      db_name: neutron

rabbitmq:
   user: rabbitmq
   password: rabbitmqp
   ip: 192.168.122.200

keystone: 192.168.122.200

glance: 192.168.122.200

nova:
  controller: 192.168.122.200
  computer: 192.168.122.200-201

cinder:
  controller: 192.168.122.200
  computer: 192.168.122.200-201

horizon: 192.168.122.200

neutron: 192.168.122.200
```

- ## Ejemplo fichero de configuración Facter

Keystone.txt:

root_password=toor
keystone_user_mysql=keystoneu
keystone_password_mysql=keystonep
keystone_db_mysql=keystone
ip_mysql=192.168.122.200
keystone_ip=192.168.122.200
user=as2admin
password=as2
tenant=openstack
path_openrc=/home/as2admin/openstack

- ## WordPress

Instalar en la máquina virtual WORDPRESS:

Instalamos paquetes MySQL, PHP y Apache:

sudo apt-get install mysql-client mysql-server
sudo apt-get install apache2
sudo apt-get install php5
sudo apt-get install php5-mysql
sudo apt-get install php5-gd

En la ruta pública de Apache, descargamos el CMS de la web y lo decomprimimos:

cd /var/www/
wget http://wordpress.org/latest.tar.gz
tar xfz latest.tar.gz

En la base de datos, nos autenticamos con el usuario root y creamos la BD del CMS, así como su usuario. Editamos el fichero my.conf para permitir acceso cara al exterior a la BD:

mysql -u root -ptoor
create user 'adminwordpress'@'localhost' identified by 'adminwordpressp';
create database wordpress;
grant all privileges on wordpress. to 'adminwordpress'@'localhost' identified by 'adminwordpressp';*
grant all privileges on wordpress. to 'adminwordpress'@'%' identified by 'adminwordpressp';*
*grant all privileges on *.* to 'root'@'localhost' identified by 'toor';*
*grant all privileges on *.* to 'root'@'%' identified by 'toor';*
sudo vi /etc/mysql/my.cnf #bind-address 0.0.0.0

Creamos el fichero de configuración a partir del fichero de ejemplo que proporciona WordPress. Editamos el fichero con los accesos a la BD que hemos creado en el paso anterior:

sudo cp /var/www/wordpress/wp-config-sample.php /var/www/wordpress/wp-config.php
sudo vi wp-config.php

Anexo 5: Código implementado

run.sh

```
#!/bin/bash
cd python
#ejecutamos el entorno OpenStack, obteniendo los logs tanto por pantalla como guardados en un fichero
fab startOpenstack | tee -a ../openstack.log
```

python/fabfile.py

```
from fabric.api import *
import openstack

def startOpenstack():
    openstack.startDeployment()
```

python/openstack.py

```
import yaml
from fabric.api import *
import configFiles, keystone, glance, nova, cinder, horizon, neutron

#cargamos el fichero Openstack modificado por el usuario
stream = open("../configOpenstackUser.yaml",'r')
configOpenstack = yaml.load(stream)

#recogemos las variables
user = configOpenstack["admin"]["user"]
password = configOpenstack["admin"]["password"]
tenant =  configOpenstack["admin"]["tenant"]
mysql_ip = configOpenstack["mysql"]["ip"]
ip_keystone = configOpenstack["keystone"]
ip_glance = configOpenstack["glance"]
ip_novaController = configOpenstack["nova"]["controller"]
ips_novaComputer = configOpenstack["nova"]["computer"]
ip_cinderController = configOpenstack["cinder"]["controller"]
ips_cinderComputer = configOpenstack["cinder"]["computer"]
ip_horizon = configOpenstack["horizon"]
ip_neutron = configOpenstack["neutron"]
ipNovaList = []
ipCinderList = []

#iniciamos variables de entorno (Fabric)
env.user = user
env.password = password
env.roledefs = {
    'keystone': [ip_keystone],
    'glance': [ip_glance],
    'novaController': [ip_novaController],
    'novaComputer': ipNovaList,
    'cinderController': [ip_cinderController],
    'cinderComputer': ipCinderList,
    'horizon': [ip_horizon],
    'neutron': [ip_neutron],
}
```

```
#inicializamos los arrays de IPs llamando a la funcion encargada de parsear las del fichero
YAML
def initIPs():
    setIPs(ips_novaComputer,ipNovaList)
    setIPs(ips_cinderComputer,ipCinderList)

#pone en marcha un entorno OpenStack
def startDeployment():
    execute(initIPs)
    execute(startConfigFiles)
    execute(startKeystone)
    execute(startGlance)
    execute(startNovaController)
    execute(startNovaComputer)
    execute(startCinderController)
    execute(startCinderComputer)
    execute(startHorizon)
    execute(startNeutron)
    execute(moveAfterConfigFiles)

#inicializa la creacion de los ficheros de configuracion para Facter
def startConfigFiles():
    configFiles.configOpenstack = configOpenstack
    configFiles.create()

#para la IP que se haya asignado con rol 'keystone': lanzamos configuracion Keystone
@roles('keystone')
def startKeystone():
    keystone.path_files = "/home/"+user+"/"+tenant
    keystone.pushKeystone()

#para la IP que se haya asignado con rol 'glance': lanzamos configuracion Glance
@roles('glance')
def startGlance():
    glance.path_files = "/home/"+user+"/"+tenant
    glance.pushGlance()

#para la IP que se haya asignado con rol 'novaController': lanzamos configuracion Nova
Controlador
@roles('novaController')
def startNovaController():
    nova.path_files = "/home/"+user+"/"+tenant
    nova.pushNovaController()

#para la IP que se haya asignado con rol 'novaComputer': lanzamos configuracion Nova
Computer
@parallel
@roles('novaComputer')
def startNovaComputer():
    nova.path_files = "/home/"+user+"/"+tenant
    nova.pushNovaComputer()

#para la IP que se haya asignado con rol 'cinderController': lanzamos configuracion Cinder
Controller
@roles('cinderController')

def startCinderController():
    cinder.path_files = "/home/"+user+"/"+tenant
    cinder.pushCinderController()
```

```
#para la IP que se haya asignado con rol 'cinderComputer': lanzamos configuracion Cinder
Computer
@parallel
@roles('cinderComputer')
def startCinderComputer():
    cinder.path_files = "/home/"+user+"/"+tenant
    cinder.pushCinderComputer()

#para la IP que se haya asignado con rol 'horizon': lanzamos configuracion Horizon
@roles('horizon')
def startHorizon():
    horizon.path_files = "/home/"+user+"/"+tenant
    horizon.pushHorizon()

#para la IP que se haya asignado con rol 'neutron': lanzamos configuracion Neutron
@roles('neutron')
def startNeutron():
    neutron.path_files = "/home/"+user+"/"+tenant
    neutron.pushNeutron()

#movemos los ficheros de configuracion posteriores al nodo controlador
@roles('novaController')
def moveAfterConfigFiles():
    put("../afterDeployment","/home/"+user+"/"+tenant)

#funcion que parsea el rango de IPs pasadas del fichero YAML para definir un array de IPs
def setIPs(ips,listaIps):
    ip_1,ip_2,ip_3,ip_4 = ips.split(".")
    ip_init,ip_end = ip_4.split("-")
    for i in range(int(ip_init),int(ip_end)+1):
        listaIps.append(ip_1+"."+ip_2+"."+ip_3+"."+str(i))
```

python/configFiles.py

```
from fabric.api import *

#define los ficheros de configuracion (para Facter) de cada servicio
configOpenstack=None

#funcion que ejecuta el resto de funciones con una unica llamada
def create():
    execute(createKeystone)
    execute(createGlance)
    execute(createNovaController)
    execute(createNovaComputer)
    execute(createCinderController)
    execute(createCinderComputer)
    execute(createNeutron)

#creamos las variables Facter necesarias para un nodo Keystone
def createKeystone():
    f = open ("../configFiles/keystone.txt", "w")
    f.write("root_password="+configOpenstack["mysql"]["root_password"]+"\n")
    f.write("keystone_user_mysql="+configOpenstack["mysql"]["keystone"]["user"]+"\n")

f.write("keystone_password_mysql="+configOpenstack["mysql"]["keystone"]["password"]+"\n")
    f.write("keystone_db_mysql="+configOpenstack["mysql"]["keystone"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
```

```
    f.write("user="+configOpenstack["admin"]["user"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.write("tenant="+configOpenstack["admin"]["tenant"]+"\n")

f.write("path_openrc=/home/"+configOpenstack["admin"]["user"]+"/"+configOpenstack["admin"]["
tenant"])
    f.close()

#creamos las variables Facter necesarias para un nodo Glance
def createGlance():
    f = open ("../configFiles/glance.txt", "w")
    f.write("root_password="+configOpenstack["mysql"]["root_password"]+"\n")
    f.write("glance_user_mysql="+configOpenstack["mysql"]["glance"]["user"]+"\n")
    f.write("glance_password_mysql="+configOpenstack["mysql"]["glance"]["password"]+"\n")
    f.write("glance_db_mysql="+configOpenstack["mysql"]["glance"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("glance_ip="+configOpenstack["glance"]+"\n")
    f.write("user="+configOpenstack["admin"]["user"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
    f.write("tenant="+configOpenstack["admin"]["tenant"])
    f.close()

#creamos las variables Facter necesarias para un nodo Nova controlador
def createNovaController():
    f = open ("../configFiles/novaController.txt", "w")
    f.write("root_password="+configOpenstack["mysql"]["root_password"]+"\n")
    f.write("nova_user_mysql="+configOpenstack["mysql"]["nova"]["user"]+"\n")
    f.write("nova_password_mysql="+configOpenstack["mysql"]["nova"]["password"]+"\n")
    f.write("nova_db_mysql="+configOpenstack["mysql"]["nova"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("nova_ip="+configOpenstack["nova"]["controller"]+"\n")
    f.write("ip_glance="+configOpenstack["glance"]+"\n")
    f.write("user_rabbitmq="+configOpenstack["rabbitmq"]["user"]+"\n")
    f.write("password_rabbitmq="+configOpenstack["rabbitmq"]["password"]+"\n")
    f.write("ip_rabbitmq="+configOpenstack["rabbitmq"]["ip"]+"\n")
    f.write("user="+configOpenstack["admin"]["user"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.write("tenant="+configOpenstack["admin"]["tenant"])
    f.close()

#creamos las variables Facter necesarias para un nodo Nova computacional
def createNovaComputer():
    f = open ("../configFiles/novaComputer.txt", "w")
    f.write("nova_user_mysql="+configOpenstack["mysql"]["nova"]["user"]+"\n")
    f.write("nova_password_mysql="+configOpenstack["mysql"]["nova"]["password"]+"\n")
    f.write("nova_db_mysql="+configOpenstack["mysql"]["nova"]["db_name"]+"\n")
    f.write("nova_ip="+configOpenstack["nova"]["controller"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("ip_glance="+configOpenstack["glance"]+"\n")
    f.write("user_rabbitmq="+configOpenstack["rabbitmq"]["user"]+"\n")
    f.write("password_rabbitmq="+configOpenstack["rabbitmq"]["password"]+"\n")
    f.write("neutron_ip="+configOpenstack["neutron"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
    f.write("user="+configOpenstack["admin"]["user"]+"\n")
    f.write("tenant="+configOpenstack["admin"]["tenant"]+"\n")
    f.write("neutron_user_mysql="+configOpenstack["mysql"]["neutron"]["user"]+"\n")
```

```
    f.write("neutron_password_mysql="+configOpenstack["mysql"]["neutron"]["password"]+"\n")
    f.write("neutron_db_mysql="+configOpenstack["mysql"]["neutron"]["db_name"]+"\n")
    f.write("ip_rabbitmq="+configOpenstack["rabbitmq"]["ip"])
    f.close()

#creamos las variables Facter necesarias para un nodo Cinder controlador
def createCinderController():
    f = open ("../configFiles/cinderController.txt", "w")
    f.write("root_password="+configOpenstack["mysql"]["root_password"]+"\n")
    f.write("cinder_user_mysql="+configOpenstack["mysql"]["cinder"]["user"]+"\n")
    f.write("cinder_password_mysql="+configOpenstack["mysql"]["cinder"]["password"]+"\n")
    f.write("cinder_db_mysql="+configOpenstack["mysql"]["cinder"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("ip_cinder="+configOpenstack["cinder"]["controller"]+"\n")
    f.write("user_rabbitmq="+configOpenstack["rabbitmq"]["user"]+"\n")
    f.write("password_rabbitmq="+configOpenstack["rabbitmq"]["password"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
    f.write("ip_rabbitmq="+configOpenstack["rabbitmq"]["ip"])
    f.close()

#creamos las variables Facter necesarias para un nodo Cinder computacional
def createCinderComputer():
    f = open ("../configFiles/cinderComputer.txt", "w")
    f.write("cinder_user_mysql="+configOpenstack["mysql"]["cinder"]["user"]+"\n")
    f.write("cinder_password_mysql="+configOpenstack["mysql"]["cinder"]["password"]+"\n")
    f.write("cinder_db_mysql="+configOpenstack["mysql"]["cinder"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("ip_cinder="+configOpenstack["cinder"]["controller"]+"\n")
    f.write("user_rabbitmq="+configOpenstack["rabbitmq"]["user"]+"\n")
    f.write("password_rabbitmq="+configOpenstack["rabbitmq"]["password"]+"\n")
    f.write("ip_rabbitmq="+configOpenstack["rabbitmq"]["ip"])
    f.close()

#creamos las variables Facter necesarias para un nodo Neutron
def createNeutron():
    f = open ("../configFiles/neutron.txt", "w")
    f.write("neutron_user_mysql="+configOpenstack["mysql"]["neutron"]["user"]+"\n")
    f.write("neutron_password_mysql="+configOpenstack["mysql"]["neutron"]["password"]+"\n")
    f.write("neutron_db_mysql="+configOpenstack["mysql"]["neutron"]["db_name"]+"\n")
    f.write("ip_mysql="+configOpenstack["mysql"]["ip"]+"\n")
    f.write("neutron_ip="+configOpenstack["neutron"]+"\n")
    f.write("user_rabbitmq="+configOpenstack["rabbitmq"]["user"]+"\n")
    f.write("password_rabbitmq="+configOpenstack["rabbitmq"]["password"]+"\n")
    f.write("ip_rabbitmq="+configOpenstack["rabbitmq"]["ip"]+"\n")
    f.write("keystone_ip="+configOpenstack["keystone"]+"\n")
    f.write("user="+configOpenstack["admin"]["user"]+"\n")
    f.write("password="+configOpenstack["admin"]["password"]+"\n")
    f.close()
```

python/cinder.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""

#definimos el metodo que lanzara Cinder controller
def pushCinderController():
    #instalamos Puppet
```

```
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/cinder",path_files)
    put("../puppet/puppet-labs/rabbitmq",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/keystone",path_files)

    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/cinder/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/rabbitmq/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")

    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/cinderController.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/cinderController.txt /etc/facter/facts.d")

    #movemos los plantillass a la ruta especificada
    put("../puppet/puppet-plantillas/cinder-plantillas",path_files)

    sudo("puppet apply "+ path_files + "/cinder-plantillas/mysqlDB.pp")
    sudo("puppet apply "+ path_files + "/cinder-plantillas/cinder.pp")

#definimos el metodo que lanzara Cinder computer
def pushCinderComputer():
    #instalamos Puppet
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/cinder",path_files)
    put("../puppet/puppet-labs/rabbitmq",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/keystone",path_files)

    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/cinder/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/rabbitmq/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")

    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/cinderComputer.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/cinderComputer.txt /etc/facter/facts.d")

    #movemos los plantillass a la ruta especificada
    put("../puppet/puppet-plantillas/cinder-plantillas",path_files)

    sudo("puppet apply "+ path_files + "/cinder-plantillas/mysqlClient.pp")
    sudo("puppet apply "+ path_files + "/cinder-plantillas/cinderCompute.pp")
```

python/glance.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""

#definimos el metodo que lanzara Glance
def pushGlance():
   #instalamos Puppet
   sudo("apt-get -y install puppet")

   #movemos los manifiestos de Puppet de la maquina local a la remota
   run("mkdir -p "+ path_files +" 2>/dev/null")
   put("../puppet/puppet-labs/mysql",path_files)
   put("../puppet/puppet-labs/glance",path_files)
   sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
   sudo("cp -R " + path_files + "/glance/ /etc/puppet/modules/")

   # movemos el fichero de configuracion a la ruta de Facter
   put("../configFiles/glance.txt",path_files)
   sudo("mkdir -p /etc/facter/facts.d")
   sudo("cp -R " + path_files + "/glance.txt /etc/facter/facts.d")

   #movemos los templates a la ruta especificada
   put("../puppet/puppet-plantillas/glance-plantillas",path_files)

   sudo("puppet apply "+ path_files + "/glance-plantillas/mysqlDB.pp")
   sudo("puppet apply "+ path_files + "/glance-plantillas/glance.pp")
```

python/horizon.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""

#definimos el metodo que lanzara nova controller
def pushHorizon():
   #instalamos Puppet
   sudo("apt-get -y install puppet")

   #movemos los templates a la ruta especificada
   run("mkdir -p "+ path_files +" 2>/dev/null")
   put("../puppet/puppet-plantillas/horizon-plantillas",path_files)

   sudo("puppet apply "+ path_files + "/horizon-plantillas/horizonPackage.pp")
```

python/keystone.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""
#definimos el metodo que lanzara keystone
def pushKeystone():
```

```
    #instalamos Puppet
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota, al dir "path_files"
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/keystone",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/apache",path_files)
    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/apache/ /etc/puppet/modules/")

    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/keystone.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/keystone.txt /etc/facter/facts.d")

    #movemos los plantillass a la ruta especificada
    put("../puppet/puppet-plantillas/keystone-plantillas",path_files)

    sudo("puppet apply "+ path_files + "/keystone-plantillas/mysqlDB.pp")
    sudo("puppet apply "+ path_files + "/keystone-plantillas/keystone.pp")
    sudo("puppet apply "+ path_files + "/keystone-plantillas/keystone-createthings.pp")
```

python/nova.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""

#definimos el metodo que lanzara nova controller
def pushNovaController():
    #instalamos Puppet
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/nova",path_files)
    put("../puppet/puppet-labs/rabbitmq",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/glance",path_files)
    put("../puppet/puppet-labs/cinder",path_files)
    put("../puppet/puppet-labs/keystone",path_files)

    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/nova/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/rabbitmq/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/glance/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/cinder/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")
```

```
    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/novaController.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/novaController.txt /etc/facter/facts.d")

    #movemos los plantillass a la ruta especificada
    put("../puppet/puppet-plantillas/nova-plantillas",path_files)

    sudo("puppet apply "+ path_files + "/nova-plantillas/mysqlDB.pp")
    sudo("puppet apply "+ path_files + "/nova-plantillas/rabbitmq.pp")
    sudo("puppet apply "+ path_files + "/nova-plantillas/nova.pp")

#definimos el metodo que lanzara nova computer
def pushNovaComputer():
    #instalamos Puppet
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/nova",path_files)
    put("../puppet/puppet-labs/rabbitmq",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/glance",path_files)
    put("../puppet/puppet-labs/cinder",path_files)
    put("../puppet/puppet-labs/keystone",path_files)
    put("../puppet/puppet-labs/neutron",path_files)
    put("../puppet/puppet-labs/vswitch",path_files)

    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/nova/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/rabbitmq/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/glance/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/cinder/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/neutron/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/vswitch/ /etc/puppet/modules/")

    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/novaComputer.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/novaComputer.txt /etc/facter/facts.d")

    #movemos los plantillass a la ruta especificada
    put("../puppet/puppet-plantillas/nova-plantillas",path_files)

    sudo("puppet apply "+ path_files + "/nova-plantillas/mysqlClient.pp")
    #sudo("puppet apply "+ path_files + "/nova-plantillas/rabbitmq.pp")
    sudo("puppet apply "+ path_files + "/nova-plantillas/novaCompute.pp")
    sudo("puppet apply "+ path_files + "/nova-plantillas/neutronCompute.pp")
```

python/neutron.py

```
from fabric.api import *

#definimos parametros de la clase
path_files = ""

#definimos el metodo que lanzara Neutron
def pushNeutron():
    #instalamos Puppet
    sudo("apt-get -y install puppet")

    #movemos los manifiestos de Puppet de la maquina local a la remota
    run("mkdir -p "+ path_files +" 2>/dev/null")
    put("../puppet/puppet-labs/mysql",path_files)
    put("../puppet/puppet-labs/neutron",path_files)
    put("../puppet/puppet-labs/nova",path_files)
    put("../puppet/puppet-labs/inifile",path_files)
    put("../puppet/puppet-labs/stdlib",path_files)
    put("../puppet/puppet-labs/vswitch",path_files)
    put("../puppet/puppet-labs/keystone",path_files)

    sudo("cp -R " + path_files + "/mysql/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/neutron/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/nova/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/inifile/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/vswitch/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/keystone/ /etc/puppet/modules/")
    sudo("cp -R " + path_files + "/stdlib/ /etc/puppet/modules/")

    # movemos el fichero de configuracion a la ruta de Facter
    put("../configFiles/neutron.txt",path_files)
    sudo("mkdir -p /etc/facter/facts.d")
    sudo("cp -R " + path_files + "/neutron.txt /etc/facter/facts.d")

    #movemos los plantillas a la ruta especificada
    put("../puppet/puppet-plantillas/neutron-plantillas",path_files)

    #creamos en Facter la variable con el ID del tenant admin de nova (parche)
    sudo("bash "+path_files+"/neutron-plantillas/parcheConfig.sh")

    sudo("puppet apply "+ path_files + "/neutron-plantillas/mysqlDB.pp")
    sudo("puppet apply "+ path_files + "/neutron-plantillas/neutron.pp")
```

puppet/puppet-plantillas/cinder-plantillas/cinder.pp

```
class { 'cinder':
  sql_connection  =>
"mysql://${cinder_user_mysql}:${cinder_password_mysql}@${ip_mysql}/${cinder_db_mysql}",
  rabbit_userid       => "${user_rabbitmq}",
  rabbit_password     => "${password_rabbitmq}",
  rabbit_host         => "${ip_rabbitmq}",
  verbose         => true,
}

class { 'cinder::keystone::auth':
  password         => "${password}",
  email            => "${user}@${user}.com",
  public_address   => "${ip_cinder}",
```

```
  admin_address    => "${ip_cinder}",
  internal_address => "${ip_cinder}",
  region           => 'regionOne',
}


class { 'cinder::api':
  keystone_password       => "${password}",
  enabled                 => true,
  keystone_auth_host      => "${keystone_ip}",
}

class { 'cinder::scheduler':
  scheduler_driver => 'cinder.scheduler.simple.SimpleScheduler',
}


#class { 'cinder::volume': }

#class { 'cinder::volume::iscsi':
#  iscsi_ip_address => "${ip_cinder}",
#}
```

puppet/puppet-plantillas/cinder-plantillas/cinderCompute.pp

```
class { 'cinder':
  sql_connection  =>
"mysql://${cinder_user_mysql}:${cinder_password_mysql}@${ip_mysql}/${cinder_db_mysql}",
  rabbit_userid       => "${user_rabbitmq}",
  rabbit_password     => "${password_rabbitmq}",
  rabbit_host         => "${ip_rabbitmq}",
  verbose         => true,
}

class { 'cinder::volume': }

class { 'cinder::volume::iscsi':
  iscsi_ip_address => "${ip_cinder}",
}
```

puppet/puppet-plantillas/cinder-plantillas/mysqlClient.pp

```
package {
   'mysql-client-core-5.5': ensure => present
}
```

puppet/puppet-plantillas/cinder-plantillas/mysqlDB.pp

```
# MySQL Server
class { 'mysql::server':
   config_hash => {
     'root_password' => "${root_password}",
          bind_address => '0.0.0.0',
   },
}

mysql::db { "${cinder_db_mysql}":
```

```
    user     => "${cinder_user_mysql}",
    password => "${cinder_password_mysql}",
    host     => '%',
    grant    => ['all'],
    charset => 'utf8',
    require => File['/root/.my.cnf'],
}
```

puppet/puppet-plantillas/glance-plantillas/glance.pp

```
class { 'glance::api':
  verbose           => true,
  keystone_tenant   => "${tenant}",
  keystone_user     => "${user}",
  keystone_password => "${password}",
  auth_host                                  => "${keystone_ip}",
  sql_connection =>
"mysql://${glance_user_mysql}:${glance_password_mysql}@${ip_mysql}/${glance_db_mysql}",
}

class { 'glance::registry':
  verbose           => true,
  keystone_tenant   => "${tenant}",
  keystone_user     => "${user}",
  keystone_password => "${password}",
  sql_connection =>
"mysql://${glance_user_mysql}:${glance_password_mysql}@${ip_mysql}/${glance_db_mysql}",
}

class { 'glance::backend::file': }

class { 'glance::keystone::auth':
  password         => "${password}",
  email            => "${user}@${user}.com",
  public_address   => "${glance_ip}",
  admin_address    => "${glance_ip}",
  internal_address => "${glance_ip}",
  region           => 'regionOne',
}
```

puppet/puppet-plantillas/glance-plantillas/mysqlDB.pp

```
# MySQL Server
class { 'mysql::server':
    config_hash => {
      'root_password' => "${root_password}",
           bind_address => '0.0.0.0',
    },
}

mysql::db { "${glance_db_mysql}":
    user     => "${glance_user_mysql}",
    password => "${glance_password_mysql}",
    host     => '%',
    grant    => ['all'],
    charset => 'utf8',
    require => File['/root/.my.cnf'],
}
```

puppet/puppet-plantillas/horizon-plantillas/horizonPackage.pp

```
package { 'apache2':
 ensure => present,
}
package { 'memcached':
 ensure => present,
}
package { 'libapache2-mod-wsgi':
 ensure => present,
}
package { 'openstack-dashboard':
 ensure => present,
}
```

puppet/puppet-plantillas/keystone-plantillas/keystone.pp

```
class { 'keystone':
  verbose        => True,
  catalog_type   => 'sql',
  admin_token    => 'random_uuid',
  sql_connection =>
"mysql://${keystone_user_mysql}:${keystone_password_mysql}@${ip_mysql}/${keystone_db_
mysql}",
}

# Installs the service user endpoint.
class { 'keystone::endpoint':
  public_address   => "${keystone_ip}",
  admin_address    => "${keystone_ip}",
  internal_address => "${keystone_ip}",
  region           => 'regionOne',
}
```

puppet/puppet-plantillas/keystone-plantillas/keystone-createthings.pp

```
class { 'keystone::roles::admin':
  admin        =>  "${user}",
  admin_tenant => "${tenant}",
  email        => "${user}@${user}.com",
  password     => "${password}",
}

# Create an openrc file
file { "${path_openrc}/openrc.sh":
  ensure    => present,
  owner     => "${user}",
  group     => "${user}",
  mode      => '0600',
  content   =>
"
export OS_TENANT_NAME=${tenant}
export OS_USERNAME=${user}
export OS_PASSWORD=${password}
export OS_AUTH_URL=\"http://${keystone_ip}:5000/v2.0/\"

"
}
```

puppet/puppet-plantillas/keystone-plantillas/mysqlDB.pp

```
# MySQL Server
class { 'mysql::server':
   config_hash => {
     'root_password' => "${root_password}",
         bind_address => '0.0.0.0',
   },
}

mysql::db { "${keystone_db_mysql}":
   user     => "${keystone_user_mysql}",
   password => "${keystone_password_mysql}",
   host     => '%',
   grant    => ['all'],
   charset => 'utf8',
   require => File['/root/.my.cnf'],
}
```

puppet/puppet-plantillas/neutron-plantillas/neutron.pp

```
# enable the neutron service
class { '::neutron':
   enabled         => true,
   allow_overlapping_ips => true,
   rabbit_host     => "${ip_rabbitmq}",
   rabbit_user     => "${user_rabbitmq}",
   rabbit_password => "${password_rabbitmq}",
   verbose         => false,
   debug           => false,
}

# configure authentication
class { 'neutron::server':
   auth_host => "${keystone_ip}",
   auth_password   => "${password}",
   agent_down_time  => '75',
   report_interval  => '30',
   database_connection  =>
"mysql://${neutron_user_mysql}:${neutron_password_mysql}@${ip_mysql}/${neutron_db_mysq
l}",
}

# Various agents
class { 'neutron::agents::dhcp': }

class { 'neutron::agents::l3':
   external_network_bridge => 'br-tun',
}

#setups neutron load balancing agent
class { 'neutron::agents::lbaas': }

#setups VPN agent
class { 'neutron::agents::vpnaas': }

#setups network metering agent
class { 'neutron::agents::metering': }
```

```
#setups OVS neutron agent
class { 'neutron::agents::ovs':
  local_ip         => "${neutron_ip}",
  enable_tunneling => true,
  polling_interval => 2,
}

class { 'neutron::plugins::ml2':
  type_drivers          => ['vxlan'],
  tenant_network_types  => ['vxlan'],
  vxlan_group           => '239.1.1.1',
  mechanism_drivers     => ['openvswitch'],
  vni_ranges            => ['0:300']
}

class { 'nova::network::neutron':
  neutron_admin_password  => "${password}",
  neutron_url                => "http://${neutron_ip}:9696",
  neutron_admin_auth_url  => "http://${neutron_ip}:35357/v2.0"
}

class { 'neutron::server::notifications':
  nova_url                    => "http://${nova_ip}:8774/v2",
  nova_admin_auth_url   => "http://${nova_ip}:35357/v2.0",
  nova_admin_password   => "${password}",
  nova_admin_tenant_id  => "${nova_admin_tenant_id}",
}

class { 'neutron::keystone::auth':
  password         => "${password}",
  email            => "${user}@${user}.com",
  public_address   => "${neutron_ip}",
  admin_address    => "${neutron_ip}",
  internal_address => "${neutron_ip}",
  region           => 'regionOne',
}
```

puppet/puppet-plantillas/neutron-plantillas/parcheConfig.sh

```
#! /bin/bash

IP=$(sed -n '11p' < $PWD/openstack/novaComputer.txt | cut -d "=" -f2)
NAME=$(sed -n '12p' < $PWD/openstack/novaComputer.txt | cut -d "=" -f2)
PASSWORD=$(sed -n '10p' < $PWD/openstack/novaComputer.txt | cut -d "=" -f2)
TENANT=$(sed -n '13p' < $PWD/openstack/novaComputer.txt | cut -d "=" -f2)

ID=$(keystone  --os-auth-url="http://$IP:5000/v2.0/" --os-tenant-name=$TENANT --os-
username=$NAME --os-password=$PASSWORD tenant-get services |  sed -n '6p' | cut -d " " -
f14)

echo "nova_admin_tenant_id=$ID" >> /etc/facter/facts.d/neutron.txt
```

puppet/puppet-plantillas/neutron-plantillas/mysqlDB.pp

```
# MySQL Server
class { 'mysql::server':
   config_hash => {
     'root_password' => "${root_password}",
```

```
            bind_address => '0.0.0.0',
    },
}

mysql::db { "${neutron_db_mysql}":
    user     => "${neutron_user_mysql}",
    password => "${neutron_password_mysql}",
    host     => '%',
    grant    => ['all'],
    charset => 'utf8',
    require => File['/root/.my.cnf'],
}
```

puppet/puppet-plantillas/neutron-plantillas/nova.pp

```
class { 'nova':
  database_connection =>
"mysql://${nova_user_mysql}:${nova_password_mysql}@${ip_mysql}/${nova_db_mysql}?chars
et=utf8",
  rabbit_userid       => "${user_rabbitmq}",
  rabbit_password     => "${password_rabbitmq}",
  image_service       => 'nova.image.glance.GlanceImageService',
  glance_api_servers  => "${ip_glance}:9292",
  verbose             => false,
  rabbit_host         => "${ip_rabbitmq}",
}

class { 'nova::rabbitmq':
 userid              => "${user_rabbitmq}",
 password            => "${password_rabbitmq}",
}

class { 'nova::client':
}

class { 'nova::api':
 enabled           => true,
 admin_password => "${password}",
 auth_host         => "${keystone_ip}",
}

class { 'nova::conductor':
 enabled => true,
}

class { 'nova::scheduler': enabled => true }

class { 'nova::cert': enabled => true }

class { 'nova::vncproxy':
 enabled => true,
 host      => "${nova_ip}"
 }

class { 'nova::consoleauth': enabled => true }

class { 'nova::keystone::auth':
  password         => "${password}",
  email            => "${user}@${user}.com",
```

```
  public_address   => "${nova_ip}",
  admin_address    => "${nova_ip}",
  internal_address => "${nova_ip}",
  region           => 'regionOne',
}
```

puppet/puppet-plantillas/neutron-plantillas/novaCompute.pp

```
class { 'nova':
  database_connection =>
"mysql://${nova_user_mysql}:${nova_password_mysql}@${ip_mysql}/${nova_db_mysql}?chars
et=utf8",
  rabbit_userid       => "${user_rabbitmq}",
  rabbit_password     => "${password_rabbitmq}",
  image_service       => 'nova.image.glance.GlanceImageService',
  glance_api_servers  => "${ip_glance}:9292",
  verbose             => false,
  rabbit_host         => "${ip_rabbitmq}",

}

class { 'nova::compute':
  enabled                       => true,
  vnc_enabled                   => true,
  vncserver_proxyclient_address => "${nova_ip}",
  vncproxy_host                 => "${nova_ip}"
}

class { 'nova::compute::libvirt':
  vncserver_listen => '0.0.0.0',
  migration_support => true,
}
```

puppet/puppet-plantillas/neutron-plantillas/rabbitmq.pp

```
class { 'rabbitmq::server':}

rabbitmq_user { "${user_rabbitmq}":
  admin    => true,
  password => "${password_rabbitmq}",
  provider => 'rabbitmqctl',
}
```

puppet/puppet-plantillas/neutron-plantillas/mysqlClient.pp

```
package {
   'mysql-client-core-5.5': ensure => present
}
```

puppet/puppet-plantillas/neutron-plantillas/mysqlDB.pp

```
# MySQL Server
class { 'mysql::server':
   config_hash => {
     'root_password' => "${root_password}",
          bind_address => '0.0.0.0',
```

```
    },
}

mysql::db { "${nova_db_mysql}":
    user     => "${nova_user_mysql}",
    password => "${nova_password_mysql}",
    host     => '%',
    grant    => ['all'],
    charset => 'utf8',
    require => File['/root/.my.cnf'],
}
```

afterDeployment/trasDespliegue.sh

```
#! /bin/bash

#Ejemplo uso: sudo sh trasDespliegue.sh user password tenantName /dev/vdb
sh create_image.sh
sh create_volume.sh $4
sudo sh create_network.sh
sh restartServices.sh
sh create_user.sh $1 $2 $3
```

afterDeployment/create_image.sh

```
#! /bin/bash

. ../openrc.sh
glance image-create --name=cirrOS --disk-format=qcow2 --container-format=bare --is-
public=true < ../glance-plantillas/images/cirros-0.3.1-x86_64-disk.img
```

afterDeployment/create_network.sh

```
#! /bin/bash

. ../openrc.sh
sudo echo "auto lo
iface lo inet loopback
auto eth0
iface eth0 inet manual
up ifconfig '$IFACE' 0.0.0.0 up
up ip link set '$IFACE' promisc on
down ip link set '$IFACE' promisc off
down ifconfig '$IFACE' down
auto br-tun
        iface br-tun inet static
        address 192.168.122.200
        netmask 255.255.255.0
        network 192.168.122.0
        broadcast 192.168.122.255
        dns-nameservers 192.168.122.1" > /etc/network/interfaces
sudo ovs-vsctl add-port br-tun eth0
neutron net-create ext-net --shared --router:external=True
neutron subnet-create ext-net --allocation-pool start=192.168.122.100,end=192.168.122.199 --
gateway=192.168.122.1 --enable_dhcp=True 192.168.122.0/24
neutron router-create myrouter
neutron router-gateway-set myrouter ext-net
```

```
neutron net-create mynet
neutron subnet-create --name mynet-subnet mynet 192.168.1.0/24
neutron router-interface-add myrouter mynet-subnet
```

afterDeployment/create_user.sh

```
#! /bin/bash

#uso: ./create_user.sh nombreUsuario passUsuario nombreTenant
. ../openrc.sh
keystone tenant-create --name=$3
keystone user-create --name=$1 --pass=$2 --email=$1@$1.com
keystone user-role-add --user=$1 --role=_member_ --tenant=$3
```

afterDeployment/create_volume.sh

```
#! /bin/bash

sudo cinder-manage db sync
sudo pvcreate $1
sudo vgcreate cinder-volumes $1
```

afterDeployment/modify-external-network.sh

```
#! /bin/bash

#tocamos tabla de encaminamiento, para decirle al nodo central cual debe ser su gateway
sudo route add -net 192.168.1.0 netmask 255.255.255.0 gw 10.0.0.1

#tocamos fichero sysctl para que la maquina actue de router
sudo echo "net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0" > /etc/sysctl.conf
sudo sysctl -p

#modificamos iptables para crear una NAT y salir cara al exterior
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

afterDeployment/restartServices.sh

```
sudo service ntp restart
sudo service rabbitmq-server restart
sudo service mysql restart
sudo service openvswitch-switch restart
sudo service keystone restart
sudo service glance-registry restart
sudo service glance-api restart
sudo service nova-api restart
sudo service nova-cert restart
sudo service nova-consoleauth restart
sudo service nova-scheduler restart
sudo service nova-conductor restart
sudo service nova-novncproxy restart
sudo service nova-compute restart
sudo service neutron-dhcp-agent restart
```

```
sudo service neutron-l3-agent restart
sudo service neutron-metadata-agent restart
sudo service neutron-plugin-openvswitch-agent restart
sudo service neutron-server restart
sudo service apache2 restart
sudo service memcached restart
sudo service cinder-scheduler restart
sudo service cinder-api restart
sudo service cinder-volume restart
sudo service tgt restart
sudo service rsync restart
sudo service memcached restart
sudo swift-init all start
sudo service swift-proxy restart
sudo service heat-api restart
sudo service heat-api-cfn restart
sudo service heat-engine restart
sudo service mongodb restart
#sudo mongod --dbpath=/var/lib/mongodb/data/db --smallfiles &
sudo service ceilometer-agent-central restart
sudo service ceilometer-api restart
sudo service ceilometer-collector restart
sudo service ceilometer-agent-compute restart
```