Automatización en tareas de aprovisionamiento en entornos de virtualización OpenStack

Grado en Ingeniería Informática (Plan 2015)

Convocatoria septiembre 2016

Director del proyecto: Manuel Torres Gil

Autor: Juan Antonio Cabrera Gutierrez

Índice

1 Introducción 5
1.1 Objetivos 6
1.2 Descripción general del entorno 6
1.3 Planificación 8
1.4 Estructura de la memoria 9
2 Tecnología asociada para el desarrollo del proyecto 11
2.1 Tecnología base para el desarrollo del proyecto 12
2.1.1 Cloud Computing 12
2.1.2 OpenStack 14
2.1.3 Aprovisionamiento 17
2.1.4 Sistema de control de versiones 21
2.1.5 Base de datos 23
2.1.6 API RESTful 24
2.2 Herramientas utilizadas para el desarrollo del proyecto 25
2.2.1 Materiales utilizados 25
2.2.2 Herramientas de diseño, desarrollo y pruebas 26
2.2.3 Lenguaje y framework de programación 29
2.3 Conclusión 31
3 Sistema para la automatización de tareas 33
3.1 Diseño del sistema 34
3.1.1 Diseño de clases 34
3.1.2 Diseño de base de datos 37
3.1.3 Diagrama de casos de uso 40
3.1.4 Diseño de la interfaz de usuario 48
3.2 Construcción del sistema 51
3.2.1 Aplicación Web 51
3.2.2 Núcleo de la aplicación 53
3.3 Conclusión 54
4 Conclusiones y posibles mejoras 55
4.1 Posibles mejoras 56
Bibliografía 57

Figuras

Figura 1.1. Diagrama de red de dos laboratorios virtuales generados a través de nuestra plataforma. 7

Figura 2.2 Relación de los principales componentes de OpenStack 16

Figura 2.3. Diferencia entre los repositorios SVN y Git 22

Figura 2.4. Gráfica comparativa de MariDB 10.1 frente a MySQL 5.7 [25] 23

Figura 2.5. Diagrama de conexión entre aplicaciones Web tradicionales y con una interfaz API REST [7] 25

Figura 2.6. Interfaz gráfica de MySQL Workbench. 26

Figura 2.7. Entorno gráfico del editor de texto Atom 27

Figura 2.8. Ejecución del editor de texto Vim sobre gnome-terminal. 28

Figura 2.9. Herramienta de inspección de código de Google Chrome. 28

Figura 2.10. Interfaz gráfica de la herramienta Postman. 29

Figura 2.11. Gráfico de los lenguajes más populares de 2015 [12] 30

En esta grafica podemos ver como resalta Python sobre el resto de los lenguajes estudiados para la realización de la aplicación. 30

Figura 3.1. Diagrama de clases del proyecto. 34

Figura 3.2. Diagrama de base de datos del proyecto. 37

Figura 3.3. Diagrama de casos de uso 40

Figura 3.4. Imagen de la barra de navegación 48

Figura 3.5. Imagen de la pantalla de login 48

Figura 3.6. Imagen del error de autenticación 49

Figura 3.7. Imagen de la pantalla labs 49

Figura 3.8. Imagen de la pantalla lab 50

Figura 3.9. Imagen del menú desplegable de acciones sobre las máquinas virtuales. 50

Figura 3.10. Imagen del formulario de creación de laboratorios. 51

Figura 3.11. Diagrama de comunicación entre servicios 52

Figura 3.12. Diagrama de conexión de los diversos subsistemas de la aplicación. 53

1 Introducción

El departamento de informática de la universidad de Almería crea un cloud privado destinado a la docencia y la investigación. Este proyecto ofrece al grupo docente un soporte sobre el cual desarrollar laboratorios que permitan crear entornos de prueba reales. El cloud privada proporciona un nivel de abstracción sobre el hardware que permite reducir la carga de trabajo sobre el mismo.

Este Trabajo de Fin de Grado pretende crear e implantar una plataforma que permite gestionar y automatizar la creación, eliminación y aprovisionamiento de laboratorios.

En este capítulo analizaremos los distintos objetivos de este proyecto, el entorno sobre el que desarrollaremos nuestra plataforma y un llevaremos a cabo la planificación de las tares de desarrollo del proyecto.

1.1 Objetivos

Con este proyecto se pretende desarrollar una herramienta que permita la creación de laboratorios virtuales en el área de informática. La creación de dichos laboratorios se llevará a cabo de forma automática siguiendo las indicaciones previas del docente. Los objetivos principales de este proyecto se pueden definir como:

- Creación de infraestructura virtual de red. Dicha herramienta deberá crear todos los dispositivos necesarios para conectar las máquinas virtuales y darles acceso a internet. Para ello sera necesario crear una red física virtual, una subred lógica virtual y un router virtual que permita redirigir el tráfico de la red hacia internet.

- Creación de máquinas virtuales. Este es el objetivo principal de la herramienta, la creación de un número indeterminado de máquinas virtuales. Estas máquinas deberán ser añadidas a la red previamente descrita y asignarle en el router una IP publica, para que el router haga NAT sobre el tráfico generado por dichas máquinas.

- Aprovisionamiento de las máquinas virtuales. Una de las principales ventajas de la herramienta es la capacidad de aprovisionar las máquinas virtuales que hemos creado. Esta característica permitirá al docente seleccionar una serie de herramientas o paquetes que desee que sean preinstalados y configurados en las máquinas virtuales.

- Configuración de las fechas y horas de activación y desactivación de los laboratorios virtuales. Esta característica tiene como objetivo minimizar la carga de computo del Cloud-DI apagando las máquinas virtuales cuando su uso no esté justificado. Por ello y para abstraer estas tareas a los docentes, se automatizará estos procesos de tal forma que el docente solo deba especificar los periodos de tiempo en los que quiera que permanezcan activos los laboratorios.

Esta herramienta pretende dar soporte a la docencia minimizando el tiempo de implantación de laboratorios virtuales y mejorando la calidad de la docencia y por ende la experiencia del alumno.

1.2 Descripción general del entorno

Este proyecto surge del Cloud del Departamento de Informática (Cloud-DI). Así pues, surge como un apoyo a la docencia con el fin de mejorar la experiencia de los alumnos. Los docentes tendrán acceso a la herramienta para poder crear laboratorios. Esta plataforma que vamos a crear gestionara y planificara laboratorios virtuales sobre OpenStack, software open source orientado al cloud que gestiona recursos de computo, almacenamiento y red.

Esta plataforma no sera exclusivamente una interfaz de interacción con OpenStack, sino que configura de forma automática el sistema operativo de las máquinas, creando cuentas de usuario, instalar paquetes y configurar las credenciales de los usuarios.

Dada la configuración del Cloud del Departamento de Informática, estas máquinas virtuales solo son accesibles desde dentro de la red de la Universidad de Almería. Para acceder a ellas desde el exterior de la Universidad sera necesario conectarse al servidor de VPN de la UAL.

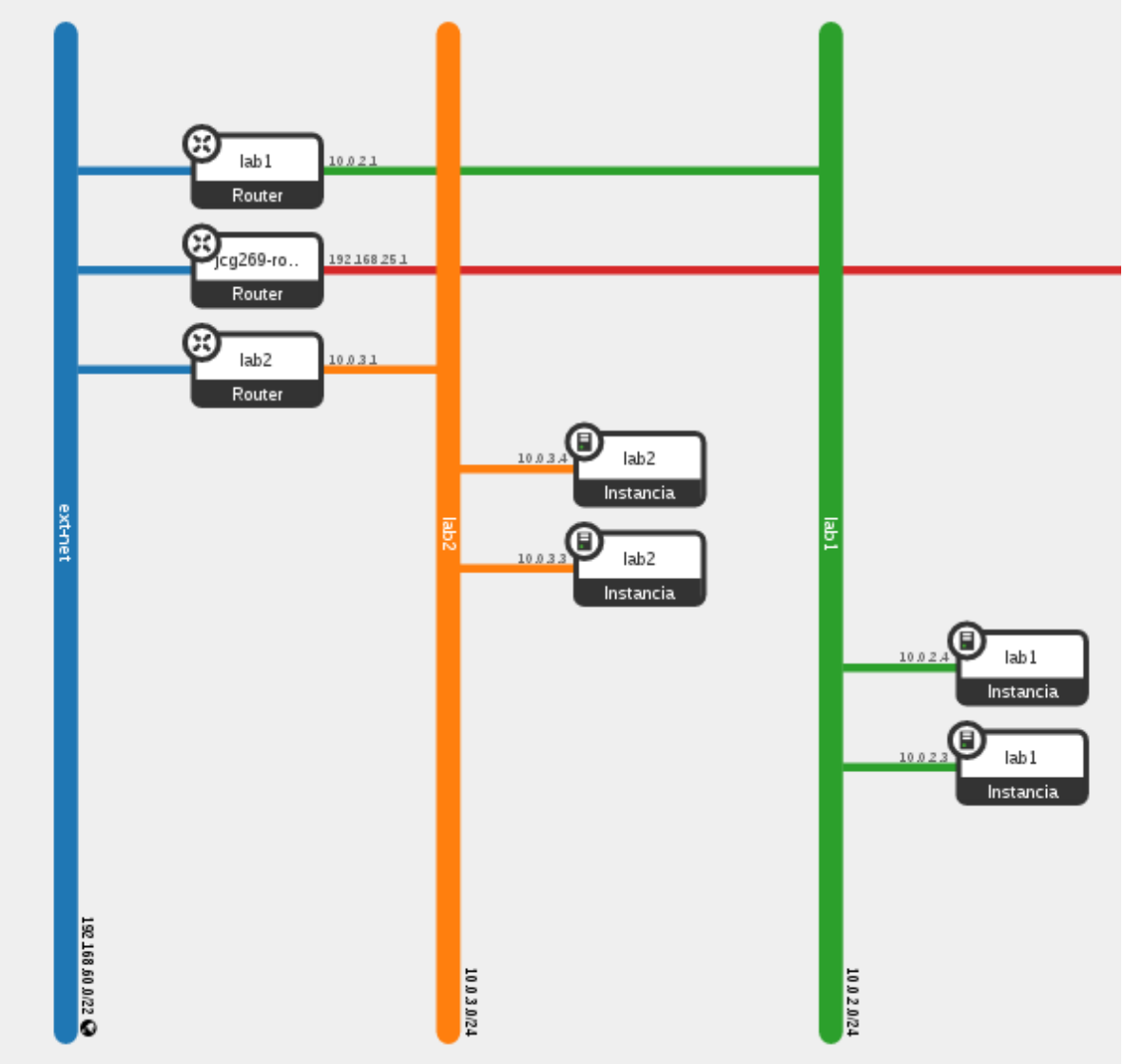

Figura 1.1. Diagrama de red de dos laboratorios virtuales generados a través de nuestra plataforma.

En la imagen anterior podemos ver la estructura de red que se genera al crear dos laboratorios con dos máquinas virtuales en cada uno de ellos.

En ella podemos comprobar que cada laboratorio cuenta con su propia red privada y su propio router que redirige los paquetes entre la red privada (10.0.X.0/24) y la subred de la universidad de Almería asignada al Cloud del departamento de Informática (192.168.62.2/22). De esta forma, las máquinas virtuales de un laboratorio no podrán interferir en las máquinas virtuales de otro laboratorio.

1.3 Planificación

Aunque este proyecto surge a mediados de febrero no es hasta mediados de junio que no se establece una rutina de trabajo, lo que supone un salto cualitativo en el desarrollo de la aplicación.

Hasta esta fecha todo el trabajo invertido en el proyecto consiste en obtener información sobre la materia, y de un alto número de entrevistas con el tutor del proyecto, el cual mostrará la infraestructura del departamento y los objetivos que desea alcanzar con dicho proyecto. Por todo ello nuestra planificación comienza a partir del día 13 de junio y finaliza el día 6 de septiembre. A continuación, mostraremos una tabla donde el lector podrá consultar la planificación del proyecto.

Semana	Mes	Horas	Descripción
13-17	Junio	20	Creación de un entorno de prueba con DevStack e instalación de nuestro propio servidor de base de datos.
20-24	Junio	20	Primera fase de desarrollo, conexión con OpenStack y toma de contacto con las librerías.
27-1	Junio	20	Desarrollo del core de la aplicación.
4-8	Julio	35	Desarrollo del core de la aplicación, así como la fase de pruebas del mismo.
11-15	Julio	35	Instalación configuración y documentación sobre la herramienta Ansible.
18-22	Julio	35	Desarrollo de los Playbooks de Ansible.
25-29	Julio	35	Refactorización y abstracción de los Playbooks de Ansible en roles.
1-5	Agosto	35	Desarrollo de la API RESTful.
8-12	Agosto	35	Desarrollo de la aplicación Web.
15-19	Agosto	35	Fase de pruebas de nuestra aplicación en desarrollo.
22-26	Agosto	28	Despliegue en producción y testeo de la aplicación.
27-6	Agosto y Septiembre	48	Desarrollo de la memoria.
	Total:	381	

1.4 Estructura de la memoria

Para el desarrollo de esta memoria hemos decidido dividir el documento en 4 capítulos, orientados a describir los distintos procesos llevados a cabo para el desarrollo del proyecto. A continuación, procedemos a enumerar y describir brevemente dichos capítulos.

- Introducción. En el llevamos a cabo una breve descripción del proyecto, los objetivos del mismo, la planificación de su desarrollo y el entorno.
- Tecnología asociada para el desarrollo del proyecto. En este capítulo se describen las tecnologías sobre las que se sustenta nuestro proyecto, así como las tecnologías empleadas en el desarrollo del mismo.
- Sistema para la automatización de tareas. En este tercer capítulo hablaremos tanto del diseño de nuestra plataforma como de la implementación de la misma. Para la exposición de su diseño, describiremos el diagrama de clase, diagrama de caso de uso, diseño de la base de datos y de la interfaz de usuario.
- Conclusiones y posibles mejoras. En este capítulo final expondremos brevemente las conclusiones de nuestro proyecto, así como las posibles mejoras aplicables a este.

2 Tecnología asociada para el desarrollo del proyecto

En este capítulo abordaremos las tecnologías que hemos empleado en el desarrollo de nuestro proyecto. Para ello trataremos de ofrecer al lector una visión de cada una de las tecnologías empleadas, así como una visión de lo que representa dicha tecnología en la actualidad. Podremos explorar en este capítulo las principales características de cada tecnología y discutiremos la elección de cada una de ellas.

Hemos dividido este capítulo en dos apartados. En el primer aparatado abordaremos las tecnologías sobre las que se basa el desarrollo de nuestro proyecto. Por el contrario, en el segundo aparatado hablaremos sobre las tecnologías y herramientas que nos han permitido desarrollar nuestra aplicación.

Con este capítulo esperamos que el lector comprenda el motivo por el cual hemos escogido cada una de las tecnologías que aquí describimos.

2.1 Tecnología base para el desarrollo del proyecto

En este apartado describiremos las tecnologías en las que está basada la plataforma desarrollada en este proyecto. Desde la tecnología del Cloud-DI hasta las tecnologías que nos permitirán desarrollar las funcionalidades de este proyecto.

2.1.1 Cloud Computing

Es un tipo de computación basado en internet, pretende compartir recursos como nodos de computación, nodos de almacenamiento y otros dispositivos. Este modelo permite ubicuidad en accesos sobre demanda a conjuntos compartidos de recursos de cómputo configurables, los cuales pueden ser rápidamente provisionados y lanzados con un mínimo esfuerzo de gestión.

A finales de los 60 y comienzo de los 70, la compañía IBM desarrolla la primera computadora diseñada específicamente para la virtualización (IBM S/360 Modelo 67). Con el descenso progresivo de los costes y la sobrecarga en el cómputo que supone la virtualización, en la década de los 80 esta tecnología cayó en el olvido y no es hasta finales de los 90 que resurgió debido al pequeño tamaño que alcanzan los transistores y la potencia que se alcanza con los nuevos procesadores. John McCarthy es quien acuña la idea de que la computación llegue a ser una utilidad pública. Pese a que el concepto de cloud computing fue creado en los sesenta, esta idea no puede ser llevada a cabo hasta que el ancho de banda de las conexiones de red no es suficientemente grande para dar una buena calidad de servicio. En 1999 la compañía Salesforce es pionera en ofrecer servicios de cloud computing a través de una aplicación web, pero no es hasta 2002 que se desarrolla Amazon Web Services y comienza el auge del cloud computing.

2.1.1.1 Tipos de Cloud Computing

En este apartado describiremos los distintos tipos de Cloud Computing que existen en el mercado.

2.1.1.1.1 Cloud público

El modelo Cloud público se basa sobre el estándar de cloud computing, en el cual un proveedor de servicios proporciona recursos, como aplicaciones y almacenamiento, a disposición el público en general a través de Internet. La mayoría de Cloud públicos ofrecen un modelo de pago por uso.

Los principales beneficios del cloud público son:

- Escalabilidad de los recursos, cuando estos sean requeridos.
- Bajo coste y facilidad de puesta en marcha.
- No se desperdician recursos dado que el pago es por el uso de estos.

Actualmente es un modelo en auge ya que permite abstraer a las medianas y pequeñas empresas de los costes derivados del mantenimiento y no requiere una inversión inicial tan elevada como en el caso de los Clouds privados.

2.1.1.1.2 Cloud privado

Es una infraestructura cloud similar al Cloud publico salvo por que dicha infraestructura esta operada únicamente por una organización, esta infraestructura es gestionada internamente o por un tercero. Este cloud puede ser alojado dentro de la organización o por un tercero, pero solo dicha organización tiene acceso al cloud. En ocasiones la escalabilidad es reducida o no se puede llevar a cabo de forma inmediata ya que esta está sujeta a una infraestructura privada y esta puede encontrarse en su mayor capacidad de carga de trabajo.

Por el contrario, ofrece ciertas ventajas frente al Cloud público. La principal ventaja que aporta una infraestructura de Cloud privada es permitir a la empresa propietaria diseñar dicha infraestructura a su medida, ofreciendo así un mayor control sobre los datos y permitiendo a dicha empresa aumentar la seguridad sobre los mismo.

2.1.1.1.3 Cloud hibrido

Es un entorno de cloud computing, el cual usa una mezcla de Cloud privado y Cloud público de terceros con orquestación entre ambas plataformas. Al permitir mover la carga de trabajo entre Cloud privadas y públicas atendiendo las necesidades de costo y computo, las nubes hibridas ofrecen a la empresa una mayor flexibilidad y más opciones de despliegue de sus datos.

El Cloud hibrido es particularmente valioso para cargas de trabajo dinámicas y con una gran variación. Se suelen emplear en aplicaciones que presentan una amplia fluctuación en su carga de trabajo temporalmente, permitiendo escalar rápidamente la aplicación sobre un cloud público.

2.1.1.2 Modelos de servicios de Cloud Computing

SPI Model es un término empleado para representar los modelos más comunes de servicios Cloud Computing. Entre los que destaca Software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS). A continuación, mostramos una figura que permite comparar los diversos modelos de Cloud Computing y como en cada uno de ellos varia la responsabilidad del proveedor de servicios.

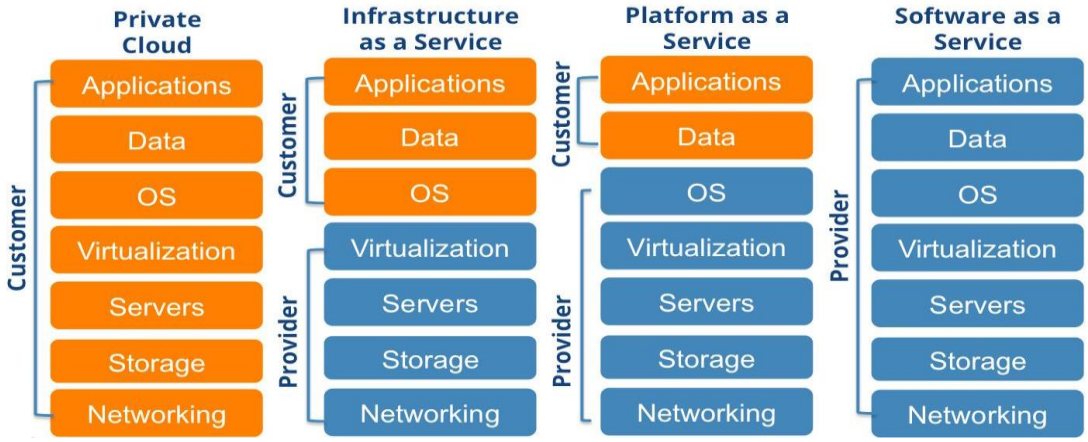

Figura 2.1 Modelos de los servicios de Cloud Computing [33]

2.1.1.2.1 SaaS

Es un modelo de distribución de software en el que una tercera parte hospeda la aplicación y pone a disposición de los clientes la aplicación a través de internet. Este modelo permite abstraer las tareas de instalación y configuración de equipos para la ejecución de dicha aplicación. Además, elimina los gastos de adquisición, aprovisionamiento y mantenimiento, así como la instalación y soporte de software licenciado.

2.1.1.2.2 PaaS

Este modelo de servicio Cloud Computing, proporciona una plataforma a medida para que los desarrolladores ejecuten y mantengan sus aplicaciones sin la complejidad de construir y mantener una infraestructura normalmente asociada con el despliegue y lanzamiento de una aplicación. De este modo tanto el mantenimiento de la infraestructura como el mantenimiento del sistema operativo corre a cargo de la empresa proveedora del servicio.

2.1.1.2.3 IaaS

Este término hace referencia al servicio online que abstrae al usuario de los detalles de la infraestructura como los recursos físicos de computación, localización, escalabilidad, seguridad, copias de seguridad, etc. Esta abstracción es posible gracias a la virtualización y la habilidad de escalar servicios acuerdo a las diferentes necesidades del cliente.

Este es el modelo que ofrece el departamento de Informática a los usuarios de Cloud-DI y sobre el cual desarrollaremos el contenido de este proyecto.

2.1.2 OpenStack

2.1.2.1 Descripción

Es una plataforma software open-source, desarrollada principalmente como una infraestructura como servicio (IaaS). La funcionalidad de este software es controlar conjuntos de hardware tanto de cómputo, como de almacenamiento y de red. Los usuarios pueden gestionar los recursos a través de una interfaz web (dashboard), mediante herramientas en línea y comandos o mediante una RESTful API. OpenStack fue lanzado bajo los términos de licencia Apache.

OpenStack comienza en 2010 como un proyecto conjunto de Rackspace Hosting y NASA. En la actualidad es gestionada por la fundación OpenStack, una organización sin ánimo de lucro establecida en 2012. Actualmente más de 500 compañías participan en el proyecto. OpenStack trabaja con unas de las empresas más conocidas y con tecnologías open source ideales para infraestructuras heterogéneas.

Cientos de las marcas más conocidas están basadas sobre OpenStack para controlar sus negocios todos los días, reduciendo costes y ayudándoles a moverse rápidamente. OpenStack tiene un fuerte ecosistema, y los usuarios que buscan soporte comercial pueden elegir entre diferentes proveedores de productos y servicios OpenStack en el mercado.

El software se descompone en diversos servicios encargados de gestionar los distintos recursos necesarios en un datacenter ofreciendo una capa de abstracción.

2.1.2.2 Servicios principales

Keystone es el servicio encargado de la autenticación e identificación tanto de los usuarios como de los servicios. En un proceso similar al empleado en el protocolo Kerberos, Keystone ofrecen token temporales a los usuarios, que se hayan autenticado con éxito, para que usen los distintos servicios. Además, este servicio se encarga de gestionar los diversos tenant y los recursos que se le han asignado a cada tenant.

Nova, este servicio genera una capa de abstracción sobre los nodos de cómputo permitiendo gestionar y ejecutar de máquinas virtuales. Este servicio abstrae los diversos hipervisores que ejecuta, lo cual permite emplear una misma interfaz de gestión a diversas tecnologías de virtualización. La arquitectura está diseñada para escalar horizontalmente el hardware, sin necesidad de emplear hardware o software propietario.

Neutron es el servicio encargado de gestionar y virtual izar recursos de red. Este servicio ofrece un amplio repertorio de recursos, tanto gestión de pool de direcciones IP como virtualización de dispositivos de red. Es a esta última característica a la cual se le puede sacar mayor partido, ya que permite crear complejas redes virtuales aumentando la seguridad, al aislar las máquinas virtuales en redes distintas con firewall virtuales, como aumentar el rendimiento, al construir balanceadores de carga virtuales que ofrecen escalabilidad.

Swift es un servicio de almacenamiento escalable y redundante, cuyo objetivo es almacenar de forma redundante los datos de los diversos servidores del centro de datos, lo que permite en caso de fallo recuperar cualquier servidor. Este servicio asegura tanto la replicación de datos como la integridad de los mismos.

Cinder es el servicio que proporciona volúmenes virtuales. Estos volúmenes son dispositivos de almacenamiento persistente que pueden ser asociados a diversas instancias de Nova.

Glance es el servicio encargado de la gestión de las imágenes. Estas imágenes pueden ser empleadas como plantillas para lanzar nuevas instancias o como copias de seguridad de instancias en ejecución.

Horizon es uno de los servicios más pequeños, pero por ello menos importante. Este servicio es el encargado de ejecutar el Dashboard, una amigable interfaz web desde la cual gestionar todos los recursos de OpenStack.

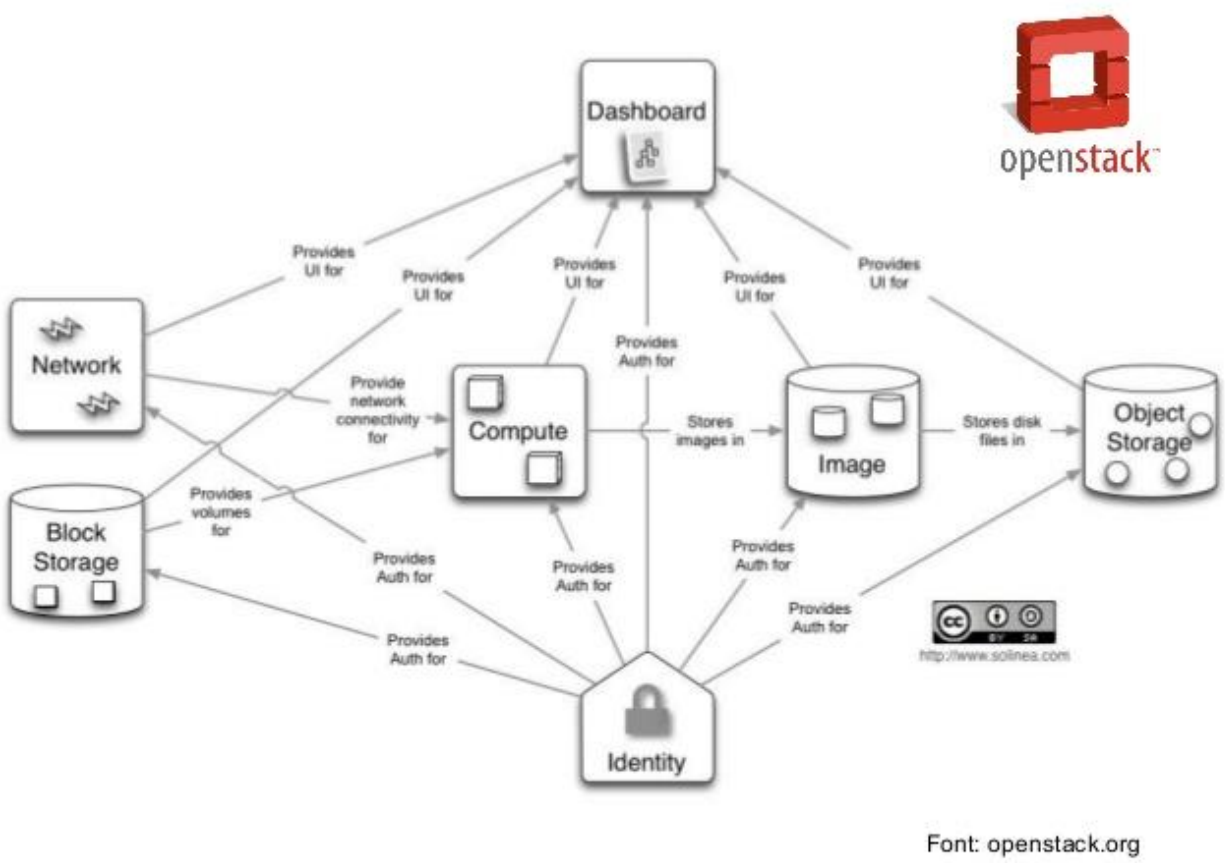

Figura 2.2 Relación de los principales componentes de OpenStack [34]

2.1.2.3 Entrono de trabajo

Para generar un entorno sobre el cual desarrollar y probar nuestra aplicación, instalamos sobre una única máquina DevStack. DevStack es herramienta usada para instalar una suite de los servicios de OpenStack para desarrollo y testing operacional. Además, demuestra y documenta ejemplos de configuración y ejecución de servicios tanto como la interfaz de comunicación con los usuarios.

DevStack [26] trae por defecto solo los servicios de Keystone, Nova y Horizon. A raíz de esta configuración por defecto tuvimos que añadir el servicio de Neutron, que nos permite configurar redes individuales a cada laboratorio.

Una vez hemos desarrollado la aplicación, hemos lanzado la primera versión de nuestra aplicación sobre la infraestructura OpenStack del departamento de informática. Esta infraestructura si cuenta con doce nodos de computo, un nodo de control, un nodo de red y un nodo de almacenamiento de bloques. Además, se pretende ampliar dicho Cloud con dos nodos de almacenamiento de objetos. A continuación, enumeramos las características de cada uno de los nodos que componen el cluster.

Nodo de computo	Procesador	AMD Opteron 6370P 16 núcleos a 2 GHz
	Memoria RAM	192 GB
	Memoria Almacenamiento masivo	SSD 160 GB HHD 1 TB
Nodo de almacenamiento de bloques	Procesador	AMD Opteron 6376P 16 núcleos a 2 GHz
	Memoria RAM	128 GB
	Memoria Almacenamiento masivo	HDD 1 TB
Nodo de control	Procesador	AMD Opteron 6370P 16 núcleos a 2 GHz
	Memoria RAM	128 GB
	Memoria Almacenamiento masivo	HDD 1 TB
Nodo de red	Procesador	AMD Opteron 6370P 16 núcleos a 2 GHz
	Memoria RAM	128 GB
	Memoria Almacenamiento masivo	HDD 1 TB

2.1.3 Aprovisionamiento

Una vez nuestro laboratorio virtual ha sido creado debemos aprovisionar dicho laboratorio con la configuración y el software que deseamos instalar por defecto en todas las máquinas del laboratorio. Para esta funcionalidad podemos encontrar en el mercado una gran variedad de herramientas de las cuales pasamos a estudiar para dilucidar cuál debemos usar.

2.1.3.1 Chef

Es una herramienta open source para administrar configuraciones, enfocada en el lado del desarrollador para su base de usuarios. Chef [27] opera con un modelo cliente servidor, con una estación de trabajo independiente necesaria para controlar el maestro. Está basada en Ruby, con Ruby puro usado para la mayoría de elementos. El diseño de Chef es transparente y basado en permitir las instrucciones que se le da, lo que significa que tú debes estar seguro de hacer que tus instrucciones sean claras.

A continuación, mostramos un ejemplo de una receta muy simple cuya finalidad es instalar el paquete nginx.

```
package ‘nginx’ do
        action :install
end
```

Para iniciar dicho servicio debemos crear la siguiente receta.

```
service ‘nginx’ do
        action [ :enable, :start ]
end
```

La compañía Chef Software, Inc. mantiene un portal web, Chef Supermarket [], en el cual podemos consultar las recetas creadas por los miembros de la comunidad.

Los principales pros de esta herramienta son:

- La rica colección de módulos y recetas de configuración.
- Code-driven da un enfoque de control y flexibilidad sobre la configuración.
- La centralización en torno a Git otorgando un gran control de versiones.

Por el contrario, las desventajas son:

- La curva de aprendizaje es muy inclinada.
- No es una herramienta muy sencilla.
- La necesidad de configurar previamente los servidores clientes.

Por estas desventajas desechamos esta herramienta como una opción viable para nuestro proyecto.

2.1.3.2 Puppet

Es una de las herramientas más extendidas de pleno derecho en la gestión de configuración. Es una herramienta open source, pero teniendo en cuenta el tiempo que ha estado presente, ha sido bien probada y desplegada en algunos de los más grandes y más demandados entornos. Puppet está basado en Ruby, pero usa un lenguaje de scripting personalizado cercano a JSON para trabajar con él. Se ejecuta como una configuración maestro-cliente y usa un enfoque basado en modelo. El diseño de código Puppet [28] trabaja como una lista de dependencias, la cuales pueden ser sencillas o más confusas dependiendo de nuestra configuración.

Como hicimos con Chef, mostramos un ejemplo de una receta muy simple cuya finalidad es instalar el paquete nginx he iniciar el servicio.

```
Class httpd {
   package { “nginx”: ensure => latest }
   service { “nginx::server”:
      name => nginx,
      ensure => running,
      enable = true
   }
}
```

La compañía Puppet, Inc. mantiene un portal web, puppetforge [], un repositorio de módulos escritos por la comunidad.

Las ventajas de Puppet son:

- Lo estable del soporte de la comunidad.
- Es la interface más madura y corre en casi todos los Sistemas Operativos.
- Una interfaz web más completa.

Por el contrario, las desventajas son:

- Para tareas complejas es necesario usar el CLI basado en Ruby.
- El soporte para versiones puras de Ruby está siendo reducida.
- El código Puppet tiende a ser muy extenso dificultando así su mantenimiento.
- La orientación basada en modelos ofrece un menor control que la orientación basada en código.

Fue por estas limitaciones no optamos por utilizar Puppet en nuestro proyecto.

2.1.3.3 SaltStack

SaltStack (Salt) [29] es una herramienta basada en CLI que puede ser preparada como un modelo cliente-servidor o como un modelo no centralizado. Basado en Python, Salt ofrece un método push y un método SSH de comunicación con el cliente. Salt permite agrupar cliente y templates de configuración simplificando el entorno de control.

A continuación, mostramos un ejemplo de una receta muy simple cuya finalidad es instalar el paquete nginx e iniciar dicho servicio.

```
nginx:
  pkg:
    - installed
  service.running:
    - watch:
      - pkg: nginx
```

Las principales ventajas de Salt son:

- Organización sencilla y su uso una vez que estás más allá de la fase de instalación.
- Las entradas, salidas y configuraciones son muy consistentes ya que emplea ficheros YAML.
- Alta escalabilidad y resiliencia en el modelo principal con niveles jerárquicos.

Por el contrario, las desventajas son:

- Difícil configurar un nuevo usuario.
- La documentación es difícil de entender en el nivel introductorio.
- La interfaz Web es nueva y poco completa frente al resto de interfaces Web.
- No soporta sistemas operativos no Linux.

Por todo ello nos parece una herramienta que aún es demasiado joven y no cumple con todas nuestras necesidades.

2.1.3.4 Ansible

Es una herramienta open source usada para desplegar aplicaciones en nodos remotos y aprovisionar servidores de un modo replicable. Proporciona un framework común para emprender múltiples niveles de aplicación y artefactos de aplicaciones usando un modelo de configuración push, a pesar de que puedes configurarlo como un modelo cliente-servidor si tú lo prefieres. Ansible [30] está construido sobre playbooks a los que puedes aplicar variables externas para desplegar tu aplicación.

Al igual que en las anteriores herramientas, mostramos un ejemplo de una receta muy simple cuya finalidad es instalar el paquete nginx e iniciar el servicio.

```
---
- name: Install nginx
  apt:
    name=nginx
    state=latest
    update_cache=yes
- name: Start nginx
  service:
    name=httpd
    state=start
```

La compañía Red Hat, Inc. mantiene un portal web, Ansible Galaxy [], en el cual podemos consultar, reutilizar y compartir contenido de Ansible.

Las principales ventajas de Ansibles son:

- Está basado en SSH, por lo que no requiere instalar ningún cliente en los nodos remotos.
- Presenta una fácil curva de aprendizaje gracias al uso de YAML.
- La estructura de Playbook es una estructura simple y clara, lo que facilita su mantenimiento.
- Tiene una función de registro variable que permite a las tareas registrar variables que afecten a tareas posteriores.
- El código base es mucho más ágil que el resto de herramientas.

Por el contrario, las desventajas de Ansible son:

- Registrar variables es necesario para funcionalidades básicas, las cuales pueden hacer tareas sencillas más complicadas.
- Difícil ver los valores de las variables dentro de los Playbooks.
- Presenta dificultades con los tiempos de rendimiento.

En nuestro caso hemos empleado la herramienta Ansible, ya que la curva de aprendizaje es muy inclinada; no requiere instalar cliente en los nodos destino; y la estructura permite que su mantenimiento sea muy sencillo. Por todo ello creemos que esta herramienta aporta mejores características para nuestro proyecto.

2.1.4 Sistema de control de versiones

Para almacenar y mantener las distintas versiones de nuestra aplicación creímos necesario la utilización de un repositorio. Es por ello que barajamos dos posibilidades.

La primera de ellas llamada Subversion [21] o más comúnmente conocida como SVN. Subversion es utilizado comúnmente en el desarrollo de software ya que permite mantener una histórico de las versiones de ficheros.

Las principales características de este repositorio es que está totalmente centralizado, basado en un modelo cliente-servidor, y los commits son operaciones atómicas, evitando así inconsistencias en los repositorios. El sistema mantiene un versionado de los directorios renombrados y con metadatos asociados, lo que permite mover o copiar dichas versiones en el árbol de directorios de una forma muy rápida. Otra de las características que lo convierte en una herramienta muy potente es la unión entre ramas trackeadas, permitiendo la unión automática entre ramas sin llamar a Subversion para que lo haga.

Por el contrario, la opción que escogimos, Git [22] es un sistema de control de versiones basado en flujo de trabajo no lineal, enfocado en una respuesta rápida, mantener integridad en los datos y soportar proyectos distribuidos.

Como muchos otros sistemas de control de versiones distribuidos, y a diferencia de la mayoría de los sistemas cliente-servidor, muchos directorios Git o equipos son miembros de pleno derecho del repositorio con completo historial y capacidad de controlar las versiones, independientemente del acceso a internet o al servidor central. Esta última ventaja fue decisiva para que nos decantásemos a utilizar dicha herramienta frente a Subversion.

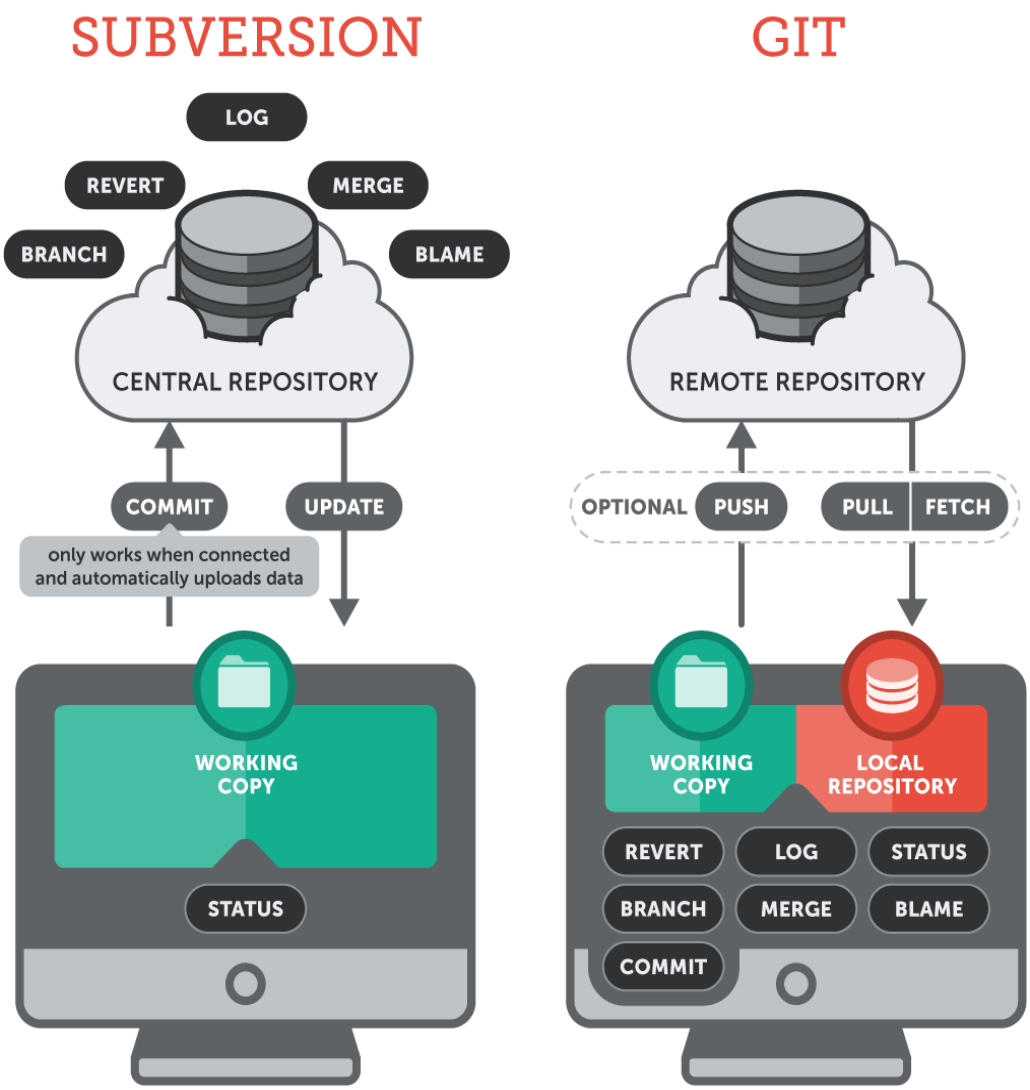

Figura 2.3. Diferencia entre los repositorios SVN y Git [35]

En la imagen anterior se muestra gráficamente las principales diferencias entre un sistema de control de versiones Subversion y un sistema de control de versiones Git.

2.1.5 Base de datos

Para poder conservar y consultar el estado de nuestra aplicación nos vemos en la necesidad de crear una base de datos. Dado que en nuestro caso deseamos almacenar el estado de nuestra aplicación entendimos que las operaciones con la base de datos deberían ser atómicas para evitar incongruencias en los datos. Por este motivo y dado que el rendimiento de la base de datos no jugaba un papel crucial en nuestra aplicación optamos por una base de datos SQL.

Una vez obtuvimos el modelo de base datos que utilizaríamos, tratamos de escoger la mejor opción para nuestro proyecto. De entre todos los posibles candidatos estudiamos la posibilidad de escoger MySQL [23] y MariaDB [24], dado que ambos sistemas de gestión de base de datos son gratuitos y ampliamente usados por la comunidad de desarrolladores.

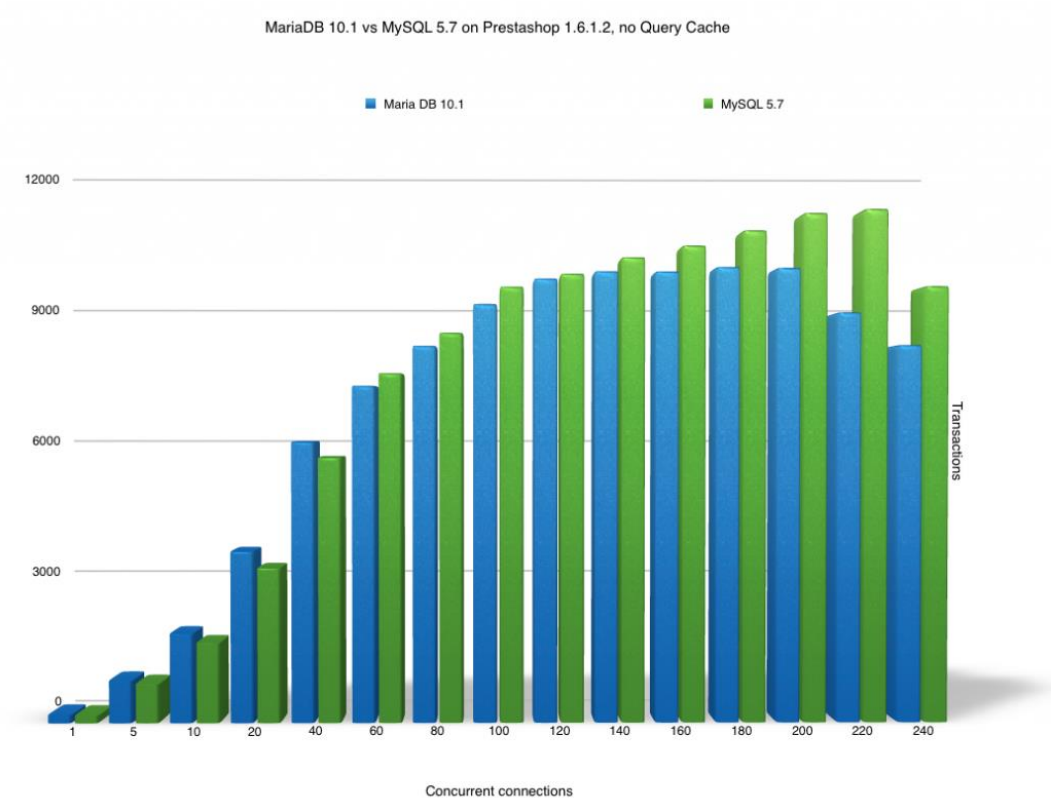

Figura 2.4. Gráfica comparativa de MariDB 10.1 frente a MySQL 5.7 [25]

En nuestro caso y entendiendo que en la actualidad MariaDB está teniendo un mejor soporte que MySQL y que presenta mejores tiempos de respuesta con pocas conexiones concurrentes, optamos por emplear MariaDB para nuestro Proyecto.

2.1.6 API RESTful

REST es el acrónimo de Representational State Transfer. Representa un tipo de arquitectura de interfaces de comunicación basada en un protocolo cliente servidor como Http, con operaciones bien definidas en el que los recursos están identificados de forma única por URLs. REST es solo una serie de principios de arquitectura. Ahora que el uso del cloud está en aumento, la mayoría de las interfaces de las aplicaciones explotan los servicios WEB y REST es la elección lógica para construir API que permiten al usuario final conectar e interactuar con los servicios cloud. RESTful APIs son usados por la mayoría de sitios, incluyendo Google, Amazon, Twitter, etc. [36]

Un RESTful API rompe una transacción al crear una serie de pequeños módulos, cada uno de los cuales aborda particularidades subyacentes de la transacción. Esta modularización proporciona a los desarrolladores mucha flexibilidad.

RESTful APIs especialmente toma ventajas de la metodología HTTP definida por el protocolo RFC 2616 [6]. El protocolo Http proporciona métodos, llamados verbos, que representan acciones sobre un recurso. Los métodos más utilizados por las API REST son las siguientes:

- GET: Se emplea para consultar a un recurso.
- POST: Se envía los datos necesarios para crear un nuevo recurso.
- PUT: Se emplea para editar un recurso.
- DELETE: Se emplea para eliminar un recurso.

Para que la aplicación del cliente detecte si las peticiones u operaciones han finalizado correctamente, el protocolo Http ofrece una serie de errores que permiten a los desarrolladores devolver errores específicos ante las peticiones. A continuación, mostramos una lista de los errores Http más utilizados:

- 200. Cuando la petición se ha resuelto correctamente.
- 201. Cuando una petición Post se resuelve correctamente.
- 401. La petición no está autorizada para acceder al recurso.
- 404. El recurso no se ha encontrado

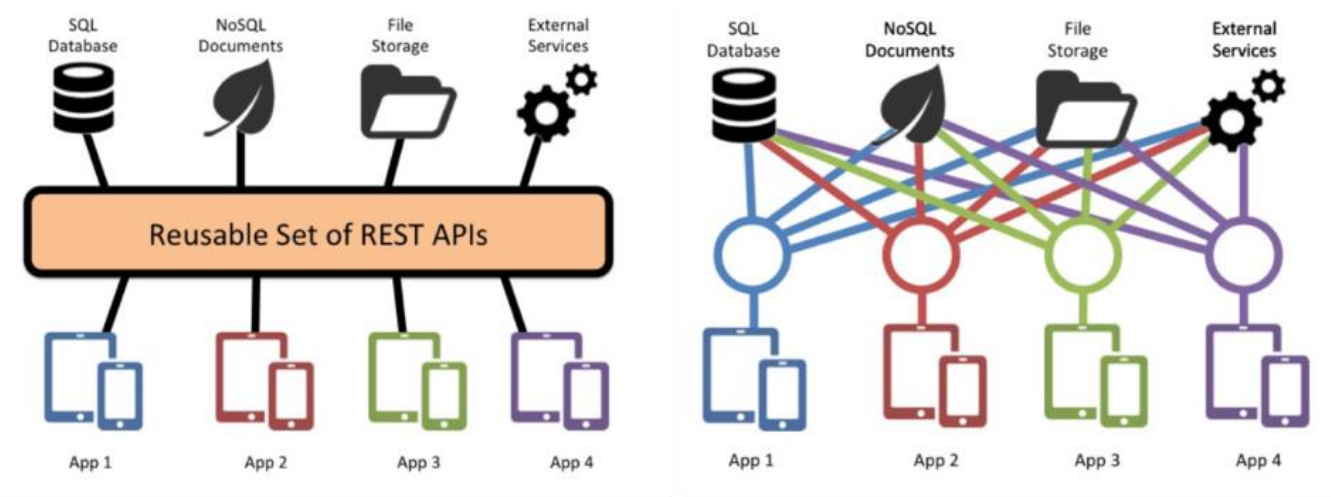

Figura 2.5. Diagrama de conexión entre aplicaciones Web tradicionales y con una interfaz API REST [7]

Esta arquitectura nos permite implementar nuestra aplicación con una estable interfaz de comunicación entre los diversos servicios, pudiendo abstraer la implementación de cada servicio al resto de servicio. Esta arquitectura permite modificar o remplazar algún servicio sin necesidad de modificar ningún otro, lo que reduce notoriamente el esfuerzo invertido en el mantenimiento de la aplicación.

2.2 Herramientas utilizadas para el desarrollo del proyecto

2.2.1 Materiales utilizados

Para este proyecto hemos hecho uso de un equipo con las siguientes características.

Procesador	Intel Core i5 4 núcleos 8 MB cache
Memoria RAM	16 GB
Almacenamiento masivo	512 GB HDD
Sistema Operativo	Fedora 24

Para la creación de un entorno de pruebas con DevStack hemos empleado una máquina virtual sobre este equipo. A continuación, procedemos a describir las características de dicha máquina virtual.

Procesador	2 núcleos
Memoria RAM	8 GB
Almacenamiento masivo	80 GB
Sistema Operativo	Ubuntu 14.04

2.2.2 Herramientas de diseño, desarrollo y pruebas

MySQL Workbench

Para el diseño de nuestra base de datos hemos empleado MySQL Workbench 6.3 [37], herramienta open source de diseño visual de base datos. Esta herramienta permite desde diseñar hasta interactuar con una base datos a través de una interfaz gráfica.

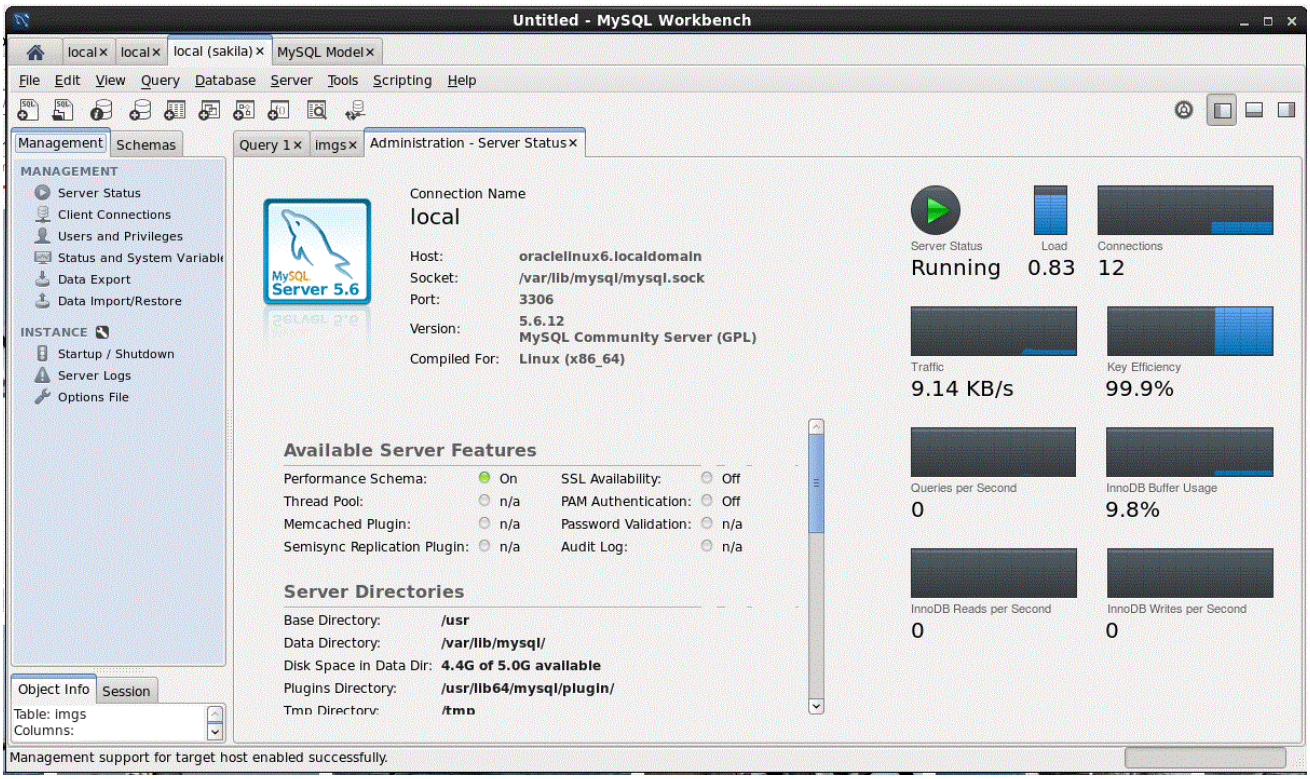

Figura 2.6. Interfaz gráfica de MySQL Workbench.

Las principales características de esta suite son:

- Editor visual SQL. Permite visualizar que tareas puedes llevar a cabo a través de una interfaz de usuario.
- Gestión de la conexión. Permite conexiones a base de datos, así como la ejecución de script a través de una interfaz gráfica.
- Ingeniería inversa. Permite obtener el modelo de una base datos a partir de un script SQL.
- Gestión de cambios. Permite tener una gestión de los cambios en un modelo de base datos, facilitando tu mantenimiento.
- Exportación e importación. Facilita la exportación e importación del modelo de base datos en scripts SQL.

Para el desarrollo de nuestra aplicación hemos utilizado dos editores de texto Atom [16] y Vim [17]. A continuación, explicamos los motivos que nos han llevado a usar ambos editores de texto.

Atom

Un editor de texto muy ligero y fácil de usar. Una de sus características estrella es el alto nivel de personalización y el elevado número de extensiones que se han desarrollado para él, pese al poco tiempo que tiene.

Atom está basado en Electron [18] un framework que permite aplicaciones de escritorio multiplataforma usando Chromium [19] y Node.js [20]. Atom puede ser usado como un IDE.

Es por su robustez, sencillez y su alto grado de configurabilidad que hemos escogido Atom.

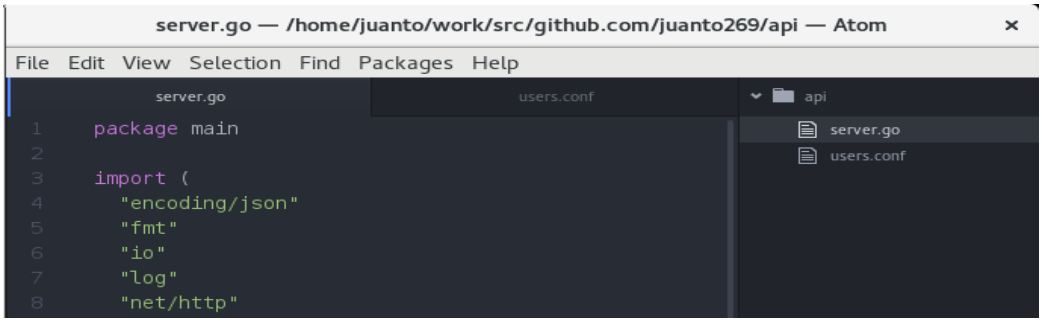

Figura 2.7. Entorno gráfico del editor de texto Atom.

Vim

Vim es un clon del editor de texto Vi. Este editor de texto fue diseñado para ser utilizado desde la interfaz de línea de comandos y como una aplicación autónoma. Vim es una aplicación open source.

Vim es uno de los más potentes editores de texto, con su propia línea de comando confiriendo una potente herramienta para la edición de texto. Es por dicha complejidad que la curva de aprendizaje presenta un crecimiento exponencial.

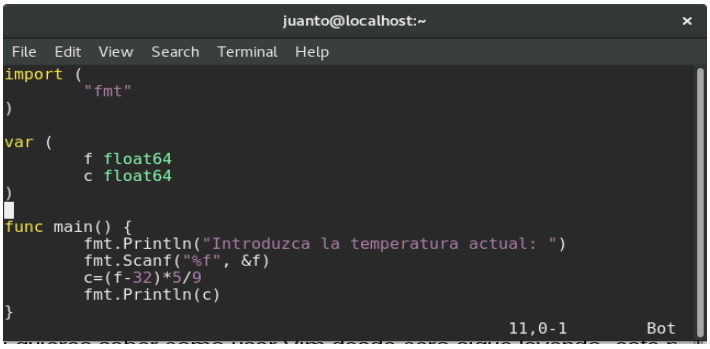

Figura 2.8. Ejecución del editor de texto Vim sobre gnome-terminal.

Google Chrome [38]

Para llevar a cabo las pruebas de nuestra aplicación hemos empleado Google Chrome. Google Chrome añade herramientas de desarrollo que nos permite modificar código en caliente, pudiendo observar de forma automática los cambios realizados.

Figura 2.9. Herramienta de inspección de código de Google Chrome.

Además, esta herramienta permite visualizar como se ejecutará nuestra aplicación en diversos dispositivos. De esta forma podemos desarrollar una aplicación responsive con gran facilidad y sin la necesidad de emplear múltiples dispositivos para las pruebas de la aplicación.

Postman

Es una gran herramienta para prototipado de APIs, y además posee las más potentes características de testeo. Esta herramienta permite configurar complejas peticiones HTTP a través de una cómoda interfaz gráfica. De este modo podemos testear nuestra API REST de una forma rápida y sencilla.

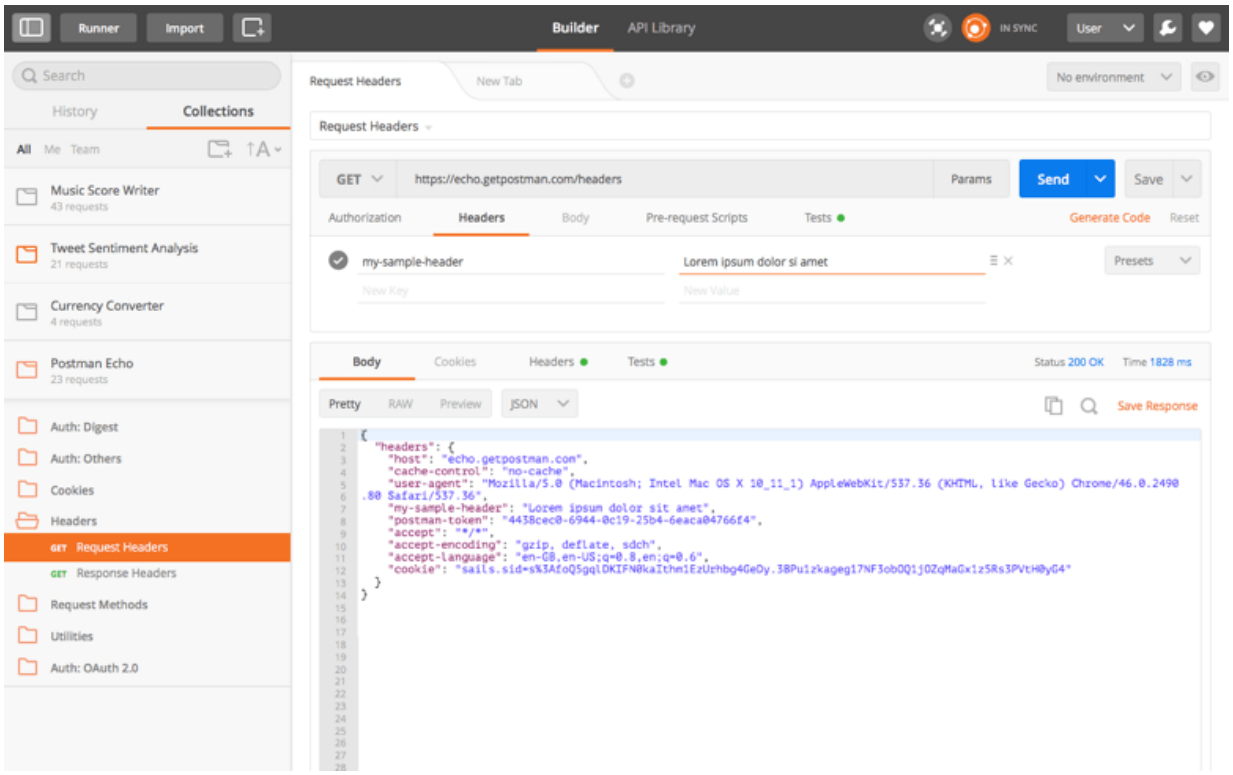

Figura 2.10. Interfaz gráfica de la herramienta Postman.

2.2.3 Lenguaje y framework de programación

2.2.3.1 Lenguaje de programación

Dada la complejidad del proyecto, tuvimos que tomar una gran decisión a la hora de seleccionar el lenguaje de programación. Puesto que nuestra aplicación requiere tanto programar la lógica de negocio de nuestra aplicación web cómo programar script de sistemas. Por estos motivos, realizamos a priori una lista de posibles candidatos justificando las ventajas y desventajas de cada uno de ellos.

En primer lugar, hablaremos C++ [8]. Pese a ser uno de los lenguajes más potentes, tanto en rendimiento como en versatilidad, tuvimos que declinar dicha opción por el elevado coste de mantenimiento del código desarrollado en C++ y la complejidad del mismo.

Como segunda opción barajamos la posibilidad de emplear Perl [9]. Pese a ser uno de los mejores lenguajes de script actuales y ser más sencilla su implementación, nos es muy potente y tiene fuertes carencias fuera del ámbito del scripting. Además, no es uno de los lenguajes más populares en la actualidad lo que impide encontrar documentación con tanta facilidad.

También estudiamos la opción de emplear Golang [10] como lenguaje de programación. Al igual que C++, Golang también es compilado lo que le confiere una gran potencia con la ventaja de ser un lenguaje más sencillo lo que facilita las tareas de mantenimiento de código. Pese a todas estas ventajas es un lenguaje relativamente joven y la comunidad de desarrolladores no es tan amplia como el resto de opciones.

Por último, escogimos como lenguaje de programación a Python [11]. Pese a ser un lenguaje interpretado, gran parte de su funcionalidad ha sido implementada en C para agilizar la ejecución de las aplicaciones. Es actualmente el lenguaje con una mayor comunidad de desarrolladores en todo el mundo. Esto se debe en parte a su versatilidad tanto para desarrollar script como para el desarrollo de aplicaciones complejas. Por todo ello consideramos esta opción como la más apropiada para nuestro proyecto.

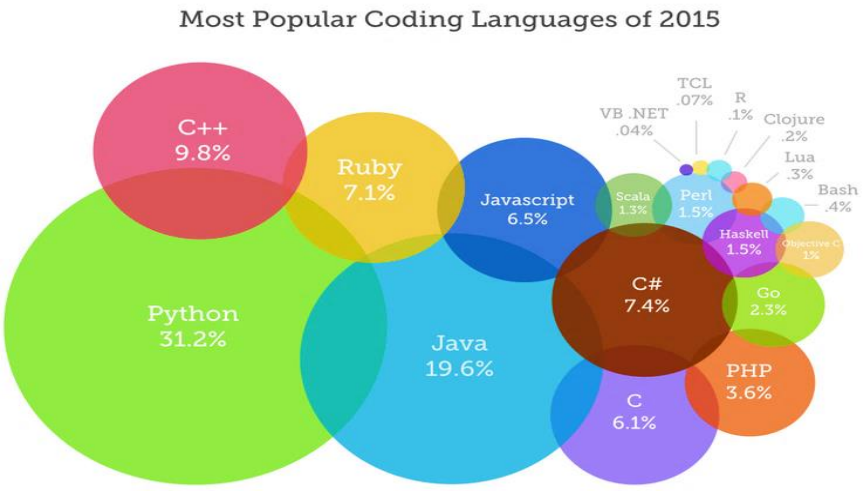

Figura 2.11. Gráfico de los lenguajes más populares de 2015 [12]

En esta grafica podemos ver como resalta Python sobre el resto de los lenguajes estudiados para la realización de la aplicación.

2.2.3.2 Framework

Una vez obtuvimos el lenguaje de programación que más se amoldaba a nuestro proyecto analizamos qué framework debíamos emplear para el desarrollo de nuestra aplicación Web.

La primera opción que estudiamos fue Pyramid [13], este framework fue concebido para proyectos grandes y muy complejos. Por ello, las herramientas de bootstrapping crean estructuras de proyecto más grandes que el resto. Además, incluye routing y autenticación, pero el templating y la administración de base de datos requiere de librerías externas.

Como segunda opción barajamos la posibilidad de emplear Django [14], pese a ser un framework muy completo y maduro es muy poco flexible y está diseñado para proyectos de media y gran envergadura.

Por su simplicidad escogimos Flask [15], el framework más joven pero cuya curva de aprendizaje es más pronunciada y cuya sencillez lo convierte en el framework ideal para pequeños proyectos como es nuestro caso.

2.3 Conclusión

Con este capítulo esperamos que el lector pueda tener una base de conocimiento sobre el entorno y la herramienta que hemos empleado para desarrollado el proyecto. Además, esperamos que comprenda las motivaciones que nos han impulsado a elegir cada una de las tecnologías y herramientas anteriormente descritas.

3 Sistema para la automatización de tareas

Este capítulo tiene como objetivo mostrar al lector como se ha llevado a cabo el desarrollo de la solución software. Desde el diseño de la solución hasta la implementación de la misma. Es por este motivo que hemos divido el capítulo en dos apartados.

El primero de ellos está destinado a mostrar el diseño de la aplicación. Comenzaremos mostrando el diseño de clases y continuaremos con el diseño de la base de datos, los diagramas de casos de uso y concluiremos con el diseño de la interfaz de usuario.

El segundo de los apartados pretende mostrar cómo se ha implementado la solución, describiendo los distintos componentes y la relación entre los mismos.

3.1 Diseño del sistema

3.1.1 Diseño de clases

En el este apartado pretendemos aclarar al lector las clases que se han desarrollado para el núcleo de nuestra aplicación. Esto le permitirá al lector comprender la relación entre ellas y a su vez simplificamos de este modo el entendimiento de las tareas llevadas a cabo por nuestra aplicación. Para poder obtener una visión global de la relación entre las clases que describiremos a continuación, facilitaremos un diagrama con las clases y la relación entre ellas.

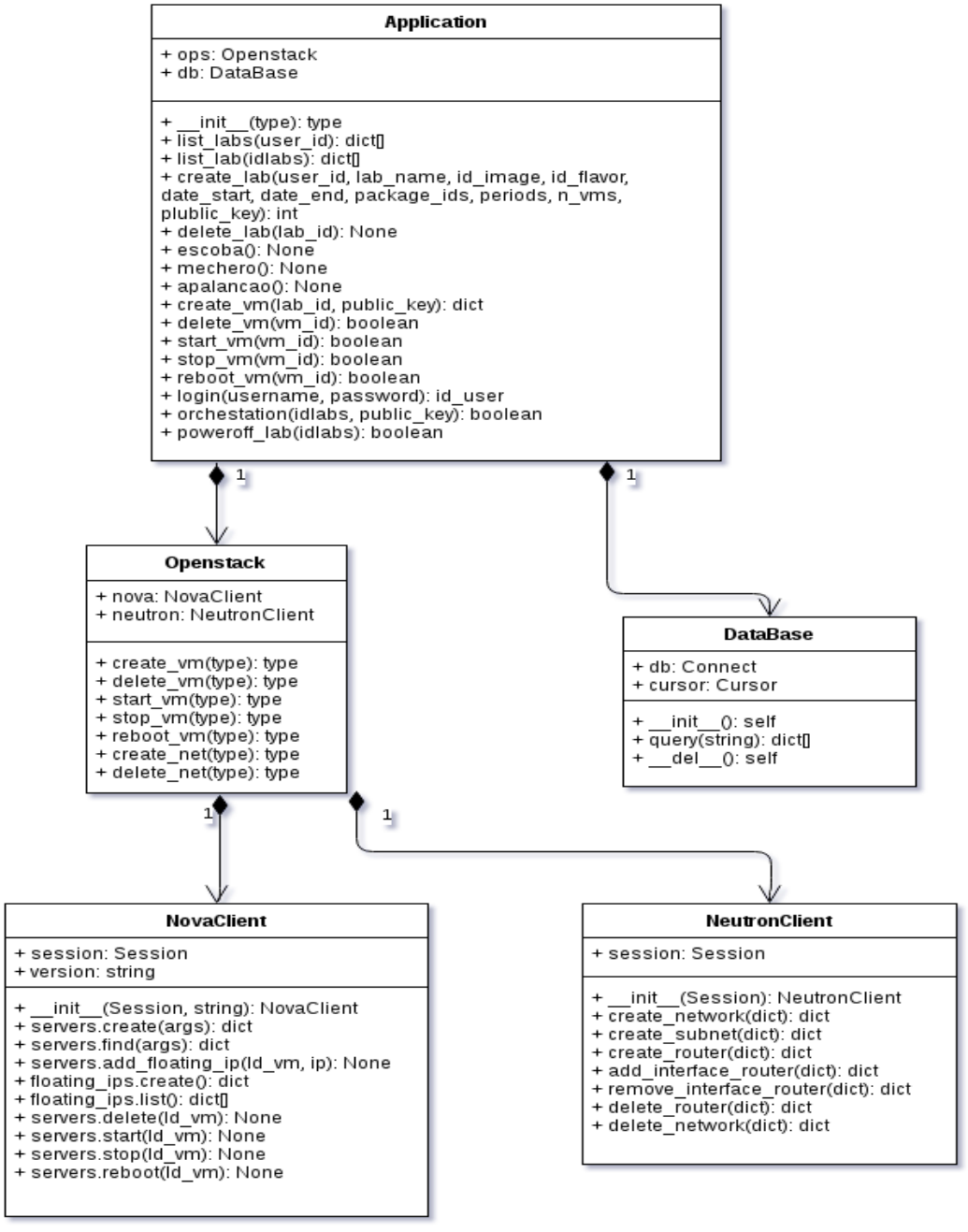

Figura 3.1. Diagrama de clases del proyecto.

Hemos optado por una estructura jerárquica de composición de clases. El objetivo de crear una estructura jerárquica es la abstracción de los procesos. De este modo los cambios en la implementación en las clases más bajas de la jerarquía no afectaran a la implementación de la Clase principal, mejorando así el mantenimiento de la aplicación.

Comenzaremos enumerando las clases que componen nuestra aplicación, explicando brevemente su finalidad, así como los atributos y métodos de la misma.

3.1.1.1 Application

Esta es la clase principal y alberga la funcionalidad completa de nuestra aplicación. El objetivo de la misma es la abstracción de todas las rutinas o funcionalidades de nuestra aplicación. La clase Application está compuesta por dos clases Openstack y DataBase, la descripción de estas clases se desarrollará más adelante en este mismo capítulo. A continuación, detallamos los métodos de esta clase con una breve descripción de los mismos.

- __init__. Este método es el constructor de la clase y no requiere ningún parámetro. Las únicas tareas que lleva a cabo es instanciar un objeto de la clase Openstack y otro de la clase DataBase.
- list_labs. Este método lista todos los laboratorios adscritos a un usuario. Para ello hace una consulta a la base de datos para obtener la id, el nombre de cada uno de los laboratorios.
- list_lab. Este método por el contrario lista las máquinas virtuales que están asociadas a un laboratorio, devolviendo la id, el estado y la dirección IP de cada una de las máquinas virtuales.
- create_lab. Este método es el encargado de la creación de un laboratorio para un usuario concreto. La relación entre usuarios y laboratorios es 1: n lo que implica que un laboratorio sólo puede pertenecer a un usuario. El objetivo principal de este método es el de abstraer al usuario de todo el proceso de creación y aprovisionado de las máquinas virtuales, así como de los elementos de red.
- delete_lab. Este método elimina un laboratorio existente. Esta eliminación conlleva la eliminación en cascada de todos los elementos de este laboratorio, abstrayendo las tareas de eliminación tanto de los dispositivos de red como de las máquinas virtuales.
- escoba. Este método elimina todos los laboratorios cuyo tiempo de vida haya expirado.
- mechero. Este método arrancara todos los laboratorios que se encuentre en un periodo de actividad. Esta tarea sólo se llevará a cabo sobre laboratorios que están actualmente apagados.
- apalancao. Este método apaga todos los laboratorios cuyo tiempo periodo de actividad haya finalizado. Esta tarea sólo se llevará a cabo sobre laboratorios que están actualmente encendidos.
- create_vm. Este método instancia una máquina virtual en OpenStack y esta es conectada a la red virtual de su laboratorio.
- delete_vm. Este método termina la instancia de una máquina virtual en OpenStack.
- start_vm. Este método está diseñado para arrancar una máquina virtual de OpenStack.

- stop_vm. Este método está diseñado para apagar una máquina virtual de OpenStack.
- reboot_vm. Este método lleva a cabo un reinicio hardware de una máquina virtual de OpenStack.
- login. Este método comprueba que los datos de autenticación de usuario que se le pasa por parámetro, corresponde a un usuario autorizado y le devuelve un identificador con el cual poder ejecutar el resto de métodos.
- orchestation. Este método aprovisiona todas las máquinas virtuales de un laboratorio. Este proceso será descrito con mayor detalle en el próximo capítulo.
- poweroff_lab. Este método tiene como objetivo apagar todas las máquinas virtuales de un laboratorio.

3.1.1.2 Openstack

Esta es la clase sobre la cual se lleva a cabo toda la interacción con OpenStack. El objetivo de esta clase es abstraer a la aplicación de la comunicación con OpenStack, de tal modo que en caso de cambiar la comunicación con OpenStack o migrar por completo el sistema a otras plataformas de cloud privado solo sería necesario modificar esta clase. Esta clase a su vez depende de NovaClient [31] y NeutronClient [32] clases desarrolladas por OpenStack que nos permite interactuar con nuestro cloud privado de OpenStack. A continuación, detallamos los métodos de esta clase con una breve descripción de los mismos.

- __init__.
- create_vm. Este método instancia una máquina virtual en OpenStack y esta es conectada a la red virtual de su laboratorio.
- delete_vm. Este método termina la instancia de una máquina virtual en OpenStack.
- start_vm. Este método está diseñado para arrancar una máquina virtual de OpenStack.
- stop_vm. Este método está diseñado para apagar una máquina virtual de OpenStack.
- reboot_vm. Este método lleva a cabo un reinicio hardware de una máquina virtual de OpenStack.
- create_net. Este método crea la red completa necesaria para un laboratorio, lo que conlleva la creación de una red, una subnet, con una máscara /24 y una dirección IP asignada de forma dinámica desde un pool de direcciones, y un router que permita la comunicación la red de la UAL.
- delete_net. Este método elimina una red, así como todos los elementos de la misma, subnet y router.

3.1.1.3 DataBase

Esta es la clase sobre la cual se lleva a cabo la interacción con la base de datos. El objetivo de esta clase es abstraer a la aplicación de la comunicación con la base de datos, de tal modo que en caso de cambiar de driver de base de datos no sea necesario modificar el core de la aplicación sino solo esta clase. A continuación, detallamos los métodos de esta clase con una breve descripción de los mismos.

- __init__. Este método es el constructor de la clase. En el momento de la instanciación de la clase se genera la conexión con la base de datos.
- query. Este método es el encargado de ejecutar la query, que se pasa por parámetro al método, en la base de datos. La ejecución de método devuelve un array de diccionarios con los resultados de la query.
- __del__. Este método es ejecutado cuando el objeto es eliminado. La única funcionalidad de este método es el cierre de la conexión con la base de datos.

3.1.2 Diseño de base de datos

Este apartado tiene como objetivo mostrar al lector el diseño de nuestra base de datos, tanto de las tablas que lo componen como de la relación entre ellas. Intentaremos que el lector comprenda porque tomamos la decisión de implementar una base de datos y el porqué del diseño de esta. Para facilitar al lector una visión clara y sencilla de nuestro diseño de la base de adjuntamos a continuación un el diagrama de nuestra base de datos.

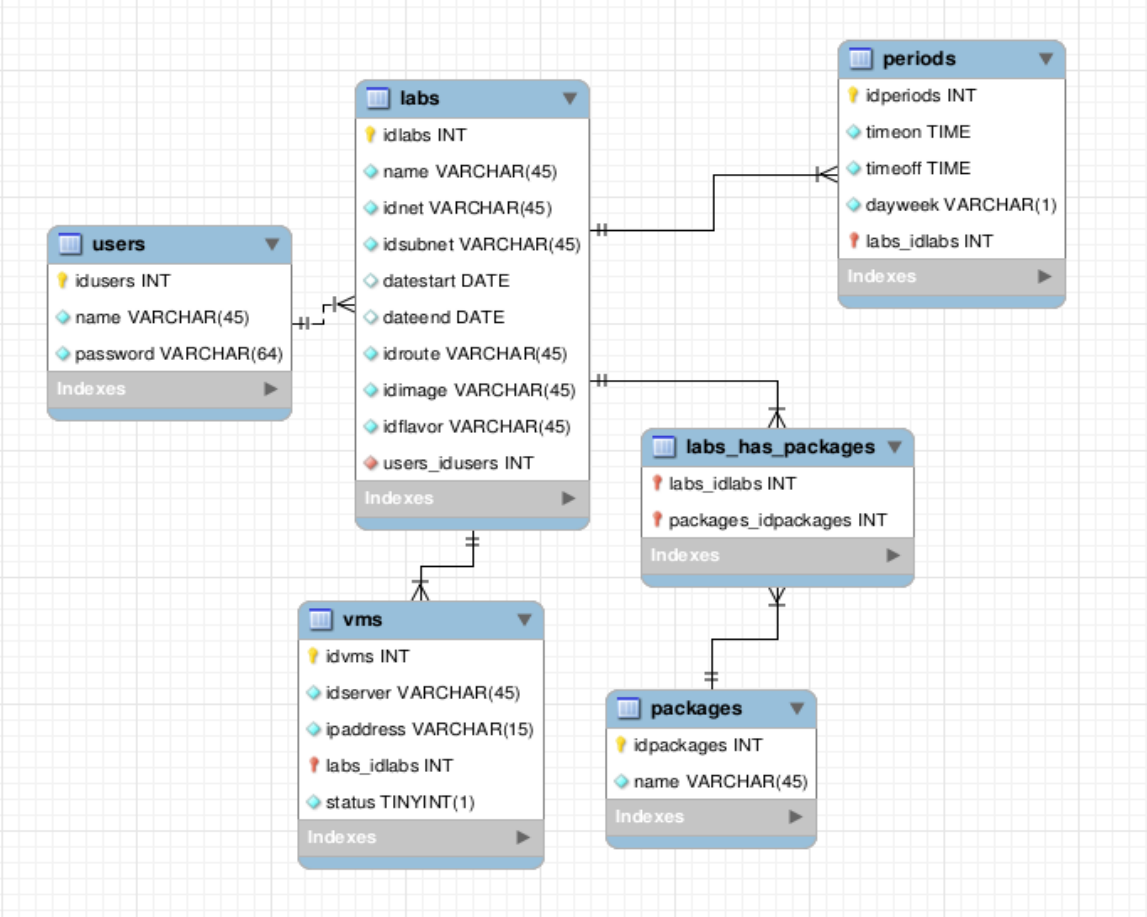

Figura 3.2. Diagrama de base de datos del proyecto.

En primer lugar, nos gustaría aclarar los motivos que nos llevaron a decidir crear una base de datos para nuestra aplicación. En la fase inicial de diseño de la aplicación nos planteamos una gran duda de diseño, en ella pretendemos dilucidar si sería necesario almacenar de forma interna el estado de nuestra aplicación o si por el contrario nuestra interfaz solo debería ser una interfaz de OpenStack y dejar que este guarde los metadatos de los laboratorios. Como respuesta a esta última alternativa, los laboratorios virtuales podrían ser definidos como Tenant en OpenStack, de tal forma que todas las instancias de un tenant representan las máquinas virtuales de un laboratorio y sería necesario crear tantos tenant como laboratorios. La principal desventaja que encontramos de este modelo es la necesidad de utilizar un usuario administrador para la interacción con OpenStack.

Por el contrario, con el uso de nuestra propia base de datos podremos alcanzar un grado de abstracción y de seguridad que de otro modo no podríamos alcanzar. Al encapsular nuestra aplicación en un único tenant nos permite usar un usuario que solo tiene privilegios en este tenant, limitando así el radio de acción de nuestra aplicación pudiendo evitar vulnerabilidades. Por todo ello consideramos que la implantación de una base de datos es la mejor opción para nuestra aplicación.

A continuación, intentaremos explicar la relación entre entidades y porque hemos optado por esta configuración. Para ello procedemos a enumerar las entidades que componen nuestra base de datos, así como una breve descripción de las mismas.

Nuestra entidad principal se llama labs, esta es la entidad central que está relacionada con el resto de entidades. Ha sido diseñada para albergar la información básica de un laboratorio. Gracias a esta entidad un laboratorio puede ser replicado con la misma información o se le puede añadir máquinas virtuales con la misma configuración inicial que el resto. A continuación, describimos brevemente los campos de esta entidad.

- idlabs. Es un identificador numérico único, se decidió optar por un id numérico para agilizar las consultas a base de datos.
- name. Este es el nombre que asigna cada usuario a su laboratorio. Este campo sera utilizado en la creación de todos los elementos que componen un laboratorio, desde redes hasta máquinas virtuales.
- idnet. Este identificador de la red del laboratorio coincide con el identificador asignado a la red por OpenStack.
- idsubnet. Este identificador de la subred del laboratorio coincide con el identificador asignado a la subred por OpenStack.
- idroute. Este identificador del router del laboratorio coincide con el identificador asignado a el router por OpenStack.
- datestart. Esta es la fecha en la que el laboratorio estará operativo. La creación del proyecto coincide con el momento de inserción en base de datos del laboratorio, pero las máquinas de este proyecto no podrán ser encendidas hasta la fecha indicada en este campo.
- dateend. Esta es la fecha en la cual será eliminado el laboratorio, así como todos los elementos que lo componen.
- idimage. Este es el identificador que le asigna OpenStack a la imagen desde la cual serán instanciadas todas las máquinas virtuales que componen este laboratorio.

- idflavor. Este es el identificador que le asigna OpenStack al sabor que especificará las características hardware de todas las instancias que compondrán este laboratorio.
- users_idusers (FK). Es la clave foránea que procede de la tabla usuarios y que permite relacionar cada laboratorio con su usuario propietario.

La tabla vms está diseñada para almacenar la información básica requerida de una instancia de un laboratorio. Esta entidad tiene como objetivo almacenar la información de interés sobre las instancias para así minimizar el número de consultas a OpenStack y agilizar de esta forma la obtención de la información para el usuario. A continuación, describimos brevemente los campos de esta entidad.

- idvms. Este es el identificador a través del cual es referenciada esta instancia en la base de datos. Se pretende con esto obtener un identificador ligero que minimice los procesos de cruzado de tablas.
- idserver. Este es el identificador usado por OpenStack para referirse a dicha instancia en concreto.
- ipaddress. Es la dirección IP flotante asignada a la instancia. Esta dirección pertenece al rango de direcciones de acceso público.
- status. Este campo tiene como objetivo almacenar en base de datos si dicha instancia está actualmente encendida o si por el contrario está apagada.
- labs_idlabs (FK). Esta clave foránea relaciona cada instancia con el laboratorio al que pertenece.

La tabla periods fue diseñada para almacenar la información referida a los periodos de actividad del laboratorio, es decir que permite al sistema determinar qué días de la semana y a qué horas permanecerán encendidas las instancias de un laboratorio. A continuación, describimos brevemente los campos de esta entidad que nos permitirán explicar al lector con mayor claridad la finalidad de dicha tabla.

- idperiods. Este es el identificador a través del cual es referenciado cada periodo en la base de datos.
- timeon. En este campo establece la hora a partir de la cual el periodo está activo.
- timeoff. En este campo por el contrario establece la hora en la cual el laboratorio al cual esté adscrito este periodo debe ser desactivado.
- dayweek. Este campo identifica el día de la semana al cual hace referencia este periodo.
- labs_idlabs (FK). Esta clave foránea relaciona cada periodo con el laboratorio al que pertenece.

La tabla packages tiene como objetivo almacenar en base datos todos los posibles paquetes o características que pueden ser añadidos a todas las instancias por defecto en el proceso de creación de los laboratorios. Esta tabla solo posee dos campos, uno de ellos un identificador numérico de cada uno de los paquetes disponibles y el segundo el nombre del mismo. Este último campo coincide con el nombre del rol asignado a dicha tarea en Ansible, aunque ahora pueda resultar confuso este tema será desarrollado en el próximo capítulo. La relación entre las tablas labs y packages es muchos a muchos por lo que de dicha relación surge una nueva identidad llamada labs_has_packages.

Por último, debemos hablar de la tabla users. Esta tabla contiene los datos más básicos de un usuario, su nombre y contraseña. Pese a la sencillez de la tabla, esta permite

agregar un nivel nuevo de seguridad asignando la gestión de cada laboratorio exclusivamente al usuario propietario de dicho laboratorio.

3.1.3 Diagrama de casos de uso

En este apartado hablaremos del caso de uso de nuestra aplicación. Para ello, en primer lugar, mostramos el diagrama y a continuación describiremos los casos de uso.

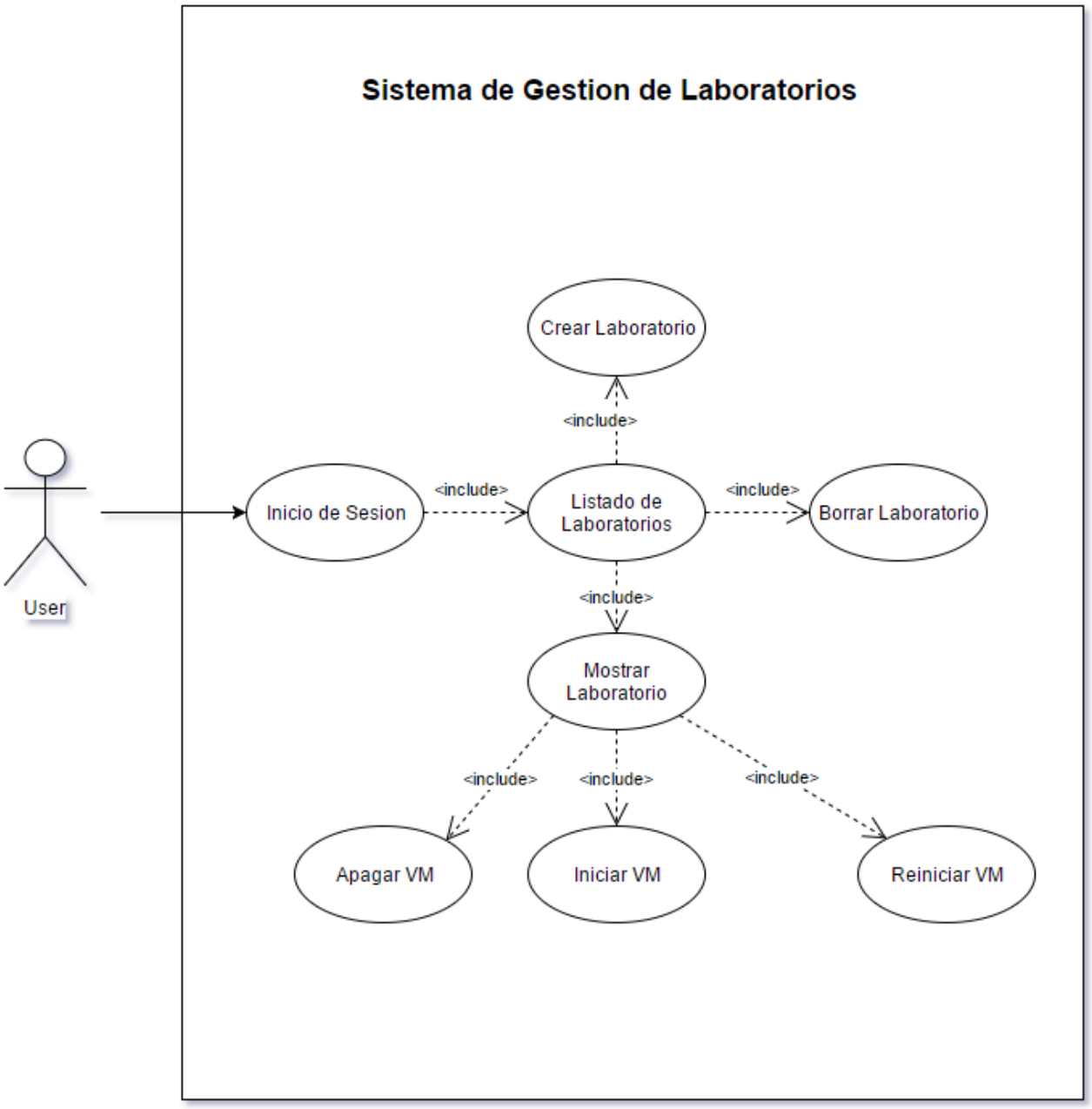

Figura 3.3. Diagrama de casos de uso

<table>
<tr><th colspan="4">Consultar los laboratorios de un usuario.</th></tr>
<tr><td>Versión</td><td colspan="3">1.0</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.</td></tr>
<tr><td rowspan="6">Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>1</td><td colspan="2">El usuario accede a la aplicación para gestionar sus laboratorios.</td></tr>
<tr><td>2</td><td colspan="2">El sistema solicita que el usuario se identifique.</td></tr>
<tr><td>3</td><td colspan="2">El usuario proporciona al sistema sus credenciales.</td></tr>
<tr><td>4</td><td colspan="2">El sistema comprueba si los datos proporcionados por el usuario son correctos.</td></tr>
<tr><td>5</td><td colspan="2">El sistema muestra al usuario una lista con los laboratorios del usuario.</td></tr>
<tr><td rowspan="3">Excepción</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>4</td><td colspan="2">Si el usuario proporciona credenciales incorrectas.</td></tr>
<tr><td></td><td>E.1</td><td>Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.</td></tr>
</table>

<table>
<tr><th colspan="4">Crear un nuevo laboratorio.</th></tr>
<tr><td>Versión</td><td colspan="3">1.0</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.</td></tr>
<tr><td rowspan="10">Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>1</td><td colspan="2">El usuario accede a la aplicación para gestionar sus laboratorios.</td></tr>
<tr><td>2</td><td colspan="2">El sistema solicita que el usuario se identifique.</td></tr>
<tr><td>3</td><td colspan="2">El usuario proporciona al sistema sus credenciales.</td></tr>
<tr><td>4</td><td colspan="2">El sistema comprueba si los datos proporcionados por el usuario son correctos.</td></tr>
<tr><td>5</td><td colspan="2">El sistema muestra al usuario una lista con los laboratorios del usuario.</td></tr>
<tr><td>6</td><td colspan="2">El usuario solicita crear un nuevo laboratorio.</td></tr>
<tr><td>7</td><td colspan="2">El sistema muestra al usuario un el formulario de creación de un nuevo laboratorio.</td></tr>
<tr><td>8</td><td colspan="2">El usuario completa el formulario con los datos del nuevo laboratorio y envía dicho formulario.</td></tr>
<tr><td>9</td><td colspan="2">El sistema vuelve al paso 5 una vez ha creado el laboratorio.</td></tr>
<tr><td rowspan="5">Excepción</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>4</td><td colspan="2">Si el usuario proporciona credenciales incorrectas.</td></tr>
<tr><td></td><td>E.1</td><td>Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.</td></tr>
<tr><td>8</td><td colspan="2">Si el usuario completa de forma incorrecta algún campo del formulario.</td></tr>
<tr><td></td><td>E.2</td><td>El sistema vuelve al paso 7 con un mensaje de error junto al campo incorrecto</td></tr>
</table>

<table>
<tr><th colspan="4">Eliminación de un laboratorio.</th></tr>
<tr><td>Versión</td><td colspan="3">1.0</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.</td></tr>
<tr><td rowspan="9">Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>1</td><td colspan="2">El usuario accede a la aplicación para gestionar sus laboratorios.</td></tr>
<tr><td>2</td><td colspan="2">El sistema solicita que el usuario se identifique.</td></tr>
<tr><td>3</td><td colspan="2">El usuario proporciona al sistema sus credenciales.</td></tr>
<tr><td>4</td><td colspan="2">El sistema comprueba si los datos proporcionados por el usuario son correctos.</td></tr>
<tr><td>5</td><td colspan="2">El sistema muestra al usuario una lista con los laboratorios del usuario.</td></tr>
<tr><td>6</td><td colspan="2">El usuario solicita eliminar un laboratorio determinado.</td></tr>
<tr><td>7</td><td colspan="2">El sistema vuelva al paso 5 una vez a eliminado el laboratorio.</td></tr>
<tr><td rowspan="3">Excepción</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>4</td><td colspan="2">Si el usuario proporciona credenciales incorrectas.</td></tr>
<tr><td></td><td>E.1</td><td>Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.</td></tr>
</table>

<table>
<tr><th colspan="4">Mostrar un laboratorio concreto.</th></tr>
<tr><td>Versión</td><td colspan="3">1.0</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.</td></tr>
<tr><td rowspan="9">Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>1</td><td colspan="2">El usuario accede a la aplicación para gestionar sus laboratorios.</td></tr>
<tr><td>2</td><td colspan="2">El sistema solicita que el usuario se autentifique.</td></tr>
<tr><td>3</td><td colspan="2">El usuario proporciona al sistema sus credenciales.</td></tr>
<tr><td>4</td><td colspan="2">El sistema comprueba si los datos proporcionados por el usuario son correctos.</td></tr>
<tr><td>5</td><td colspan="2">El sistema muestra al usuario una lista con los laboratorios del usuario.</td></tr>
<tr><td>6</td><td colspan="2">El usuario selecciona el laboratorio que desea visualizar.</td></tr>
<tr><td>7</td><td colspan="2">El sistema muestra al usuario una lista con todas las máquinas virtuales que componen el laboratorio.</td></tr>
<tr><td rowspan="3">Excepciones</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>4</td><td colspan="2">Si el usuario proporciona credenciales incorrectas.</td></tr>
<tr><td></td><td>E.1</td><td>Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.</td></tr>
</table>

<table>
<tr><th colspan="4">Arrancar una máquina virtual.</th></tr>
<tr><td>Versión</td><td colspan="3">1.0</td></tr>
<tr><td>Descripción</td><td colspan="3">El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.</td></tr>
<tr><td rowspan="12">Secuencia normal</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>1</td><td colspan="2">El usuario accede a la aplicación para gestionar sus laboratorios.</td></tr>
<tr><td>2</td><td colspan="2">El sistema solicita que el usuario se identifique.</td></tr>
<tr><td>3</td><td colspan="2">El usuario proporciona al sistema sus credenciales.</td></tr>
<tr><td>4</td><td colspan="2">El sistema comprueba si los datos proporcionados por el usuario son correctos.</td></tr>
<tr><td>5</td><td colspan="2">El sistema muestra al usuario una lista con los laboratorios del usuario.</td></tr>
<tr><td>6</td><td colspan="2">El usuario selecciona el laboratorio que desea visualizar.</td></tr>
<tr><td>7</td><td colspan="2">El sistema muestra al usuario una lista con todas las máquinas virtuales que componen el laboratorio.</td></tr>
<tr><td>8</td><td colspan="2">El usuario solicita al sistema una lista de las acciones posibles sobre una máquina virtual.</td></tr>
<tr><td>9</td><td colspan="2">El sistema muestra al usuario una lista con las posibles acciones: Arrancar, Apagar y Reiniciar.</td></tr>
<tr><td>10</td><td colspan="2">El usuario seleccionar la opción que desea.</td></tr>
<tr><td>11</td><td colspan="2">El sistema vuelve al paso 7 tras llevar acabo la acción que solicitud el usuario.</td></tr>
<tr><td rowspan="5">Excepciones</td><td>Paso</td><td colspan="2">Acción</td></tr>
<tr><td>4</td><td colspan="2">Si el usuario proporciona credenciales incorrectas.</td></tr>
<tr><td></td><td>E.1</td><td>Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.</td></tr>
<tr><td>10</td><td colspan="2">Si el usuario selecciona una acción inconsistente con el estado actual de la máquina.</td></tr>
<tr><td></td><td>E.3</td><td>El sistema regresa al paso 7 sin llevar a cabo ninguna acción.</td></tr>
</table>

Apagar una máquina virtual.			
Versión	1.0		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.		
Secuencia normal	Paso	Acción	
	1	El usuario accede a la aplicación para gestionar sus laboratorios.	
	2	El sistema solicita que el usuario se identifique.	
	3	El usuario proporciona al sistema sus credenciales.	
	4	El sistema comprueba si los datos proporcionados por el usuario son correctos.	
	5	El sistema muestra al usuario una lista con los laboratorios del usuario.	
	6	El usuario selecciona el laboratorio que desea visualizar.	
	7	El sistema muestra al usuario una lista con todas las máquinas virtuales que componen el laboratorio.	
	8	El usuario solicita al sistema una lista de las acciones posibles sobre una máquina virtual.	
	9	El sistema muestra al usuario una lista con las posibles acciones: Arrancar, Apagar y Reiniciar.	
	10	El usuario seleccionar la opción que desea.	
	11	El sistema vuelve al paso 7 tras llevar acabo la acción que solicitud el usuario.	
Excepciones	Paso	Acción	
	4	Si el usuario proporciona credenciales incorrectas.	
		E.1	Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.
	10	Si el usuario selecciona una acción inconsistente con el estado actual de la máquina.	
		E.3	El sistema regresa al paso 7 sin llevar a cabo ninguna acción.

Reiniciar una máquina virtual.			
Versión	1.0		
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando el usuario desee gestionar sus laboratorios virtuales.		
Secuencia normal	Paso	Acción	
	1	El usuario accede a la aplicación para gestionar sus laboratorios.	
	2	El sistema solicita que el usuario se identifique.	
	3	El usuario proporciona al sistema sus credenciales.	
	4	El sistema comprueba si los datos proporcionados por el usuario son correctos.	
	5	El sistema muestra al usuario una lista con los laboratorios del usuario.	
	6	El usuario selecciona el laboratorio que desea visualizar.	
	7	El sistema muestra al usuario una lista con todas las máquinas virtuales que componen el laboratorio.	
	8	El usuario solicita al sistema una lista de las acciones posibles sobre una máquina virtual.	
	9	El sistema muestra al usuario una lista con las posibles acciones: Arrancar, Apagar y Reiniciar.	
	10	El usuario seleccionar la opción que desea.	
	11	El sistema vuelve al paso 7 tras llevar acabo la acción que solicitud el usuario.	
Excepción	Paso	Acción	
	4	Si el usuario proporciona credenciales incorrectas.	
		E.1	Se muestra al usuario un mensaje de error y el sistema vuelve al paso 2.
	10	Si el usuario selecciona una acción inconsistente con el estado actual de la máquina.	
		E.3	El sistema regresa al paso 7 sin llevar a cabo ninguna acción.

3.1.4 Diseño de la interfaz de usuario

Para la interacción con el usuario hemos optado por implementar una interfaz Web. En este apartado intentaremos explicar al lector las decisiones de diseño que hemos tomado con respecto a la interfaz de nuestra aplicación y para ello haremos uso de diversas figuras que muestran la interfaz actual.

En primer lugar, hablaremos de la barra de navegación. La funcionalidad de nuestra aplicación está dividida en diversas ventanas. Cada una de estas ventanas ha sido diseñada para llevar a cabo una función específica. Pero antes de discutir la navegación y las diversas ventanas que componen nuestra aplicación mostraremos en la siguiente imagen la barra de navegación de nuestra aplicación.

Figura 3.4. Imagen de la barra de navegación

En esta barra de navegación podemos apreciar claramente que está divida en tres secciones. La primera de ellas situada en el margen izquierdo de la barra de navegación, contiene un enlace al resto de herramientas que ofrece el departamento de informática. Este enlace tiene como objetivo integrar conceptualmente nuestra aplicación en el resto de herramientas que ofrece actualmente el departamento de Informática, ofreciendo una sensación de continuidad.

La segunda sección de nuestra barra de navegación se encuentra situada en la zona central y en la actualidad solo contiene un enlace a la página de inicio de nuestra aplicación. Más adelante en este mismo apartado discutiremos el motivo porque nuestro menú de navegación solo tiene un único enlace.

Por último, el tercer elemento de nuestra barra de navegación es un botón que nos permite cerrar la sesión del usuario que esté actualmente logueado en nuestra aplicación. Una vez la aplicación ha cerrado la sesión del usuario este será redirigido a la página de login que mostramos a continuación.

Figura 3.5. Imagen de la pantalla de login

Esta imagen muestra la pantalla de login del usuario. Como podemos ver en el margen superior al igual que en el resto de pantalla de nuestra aplicación, encontramos la barra de navegación que hemos descrito anteriormente.

Esta pantalla contiene un pequeño formulario que permite al usuario introducir sus credenciales para hacer uso de nuestra aplicación. Si cualquier usuario pretende acceder a alguno de nuestros recursos sin haber iniciado sesión previamente será redirigido a esta ventana. En la imagen que mostramos a continuación mostramos el mensaje de error en la autenticación de un usuario.

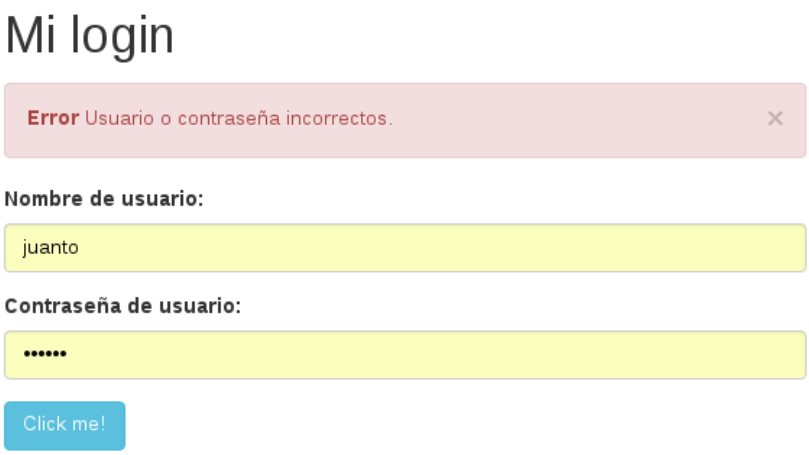

Figura 3.6. Imagen del error de autenticación

Una vez el usuario haya tenido éxito en la autenticación, este será redirigido a la ventana que hemos llamado Mis labs. A continuación, mostramos una muestra de la misma.

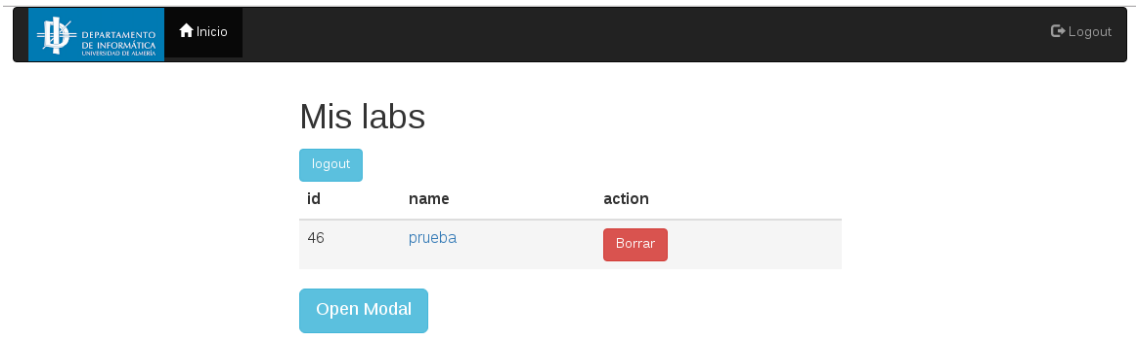

Figura 3.7. Imagen de la pantalla labs

El objetivo principal de esta ventana es mostrar todos los laboratorios asociados al usuario autenticado. Como podemos ver se muestra una lista de todos los laboratorios detallando el id, el nombre del laboratorio y la opción de eliminar dicho laboratorio. El nombre de cada laboratorio es el enlace que conduce a la ventana de lab, la cual será descrita más adelante.

En la actualidad esta es la ventana de inicio, y como comentamos con anterioridad, en este mismo apartado, es el único enlace en el menú de navegación. Esta decisión de diseño se debe a que toda entidad pertenece a un laboratorio por ende la única forma de accederá a un recurso es seleccionar previamente el laboratorio al que pertenece. De este modo simplificamos la interfaz y minimizamos la memoria cognitiva necesaria para acceder a cualquiera de los recursos.

Una vez seleccionado el laboratorio sobre el que vamos a trabajar, este te conduce a la siguiente ventana.

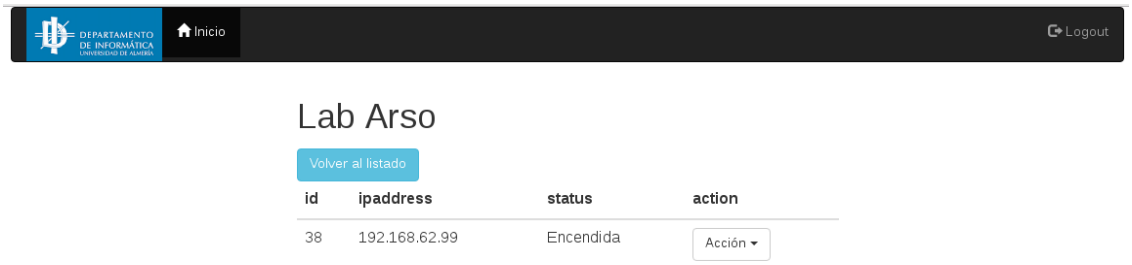

Figura 3.8. Imagen de la pantalla lab

Como podemos ver en la imagen, se muestra una tabla con todas las instancias de un laboratorio. Esta tabla muestra tanto el id de cada instancia como la dirección IP de la misma y su estado actual. Además, ofrecemos un menú desplegable para cada instancia que nos permita, apagar, encender o reiniciar cada máquina de forma individual. El objetivo de este menú, es ofrecer al docente la capacidad de gestionar las máquinas sin necesidad de acceder a OpenStack. Consideramos que esto puede ser útil en caso de que surja algún incidente o error durante el uso de las instancias por parte de los alumnos. A continuación, mostramos el menú contextual del que hemos hablado durante este párrafo.

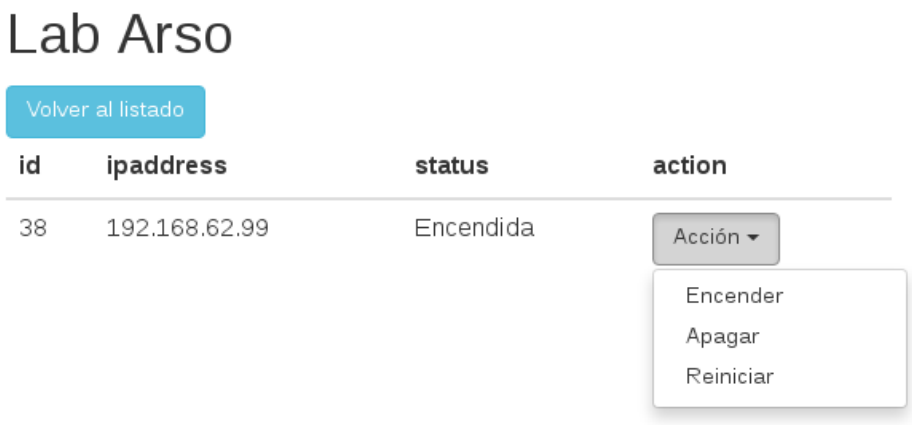

Figura 3.9. Imagen del menú desplegable de acciones sobre las máquinas virtuales.

Por último, hablaremos sobre el formulario que permitirá al usuario crear de forma sencilla laboratorios. A continuación, mostramos una imagen de dicho formulario.

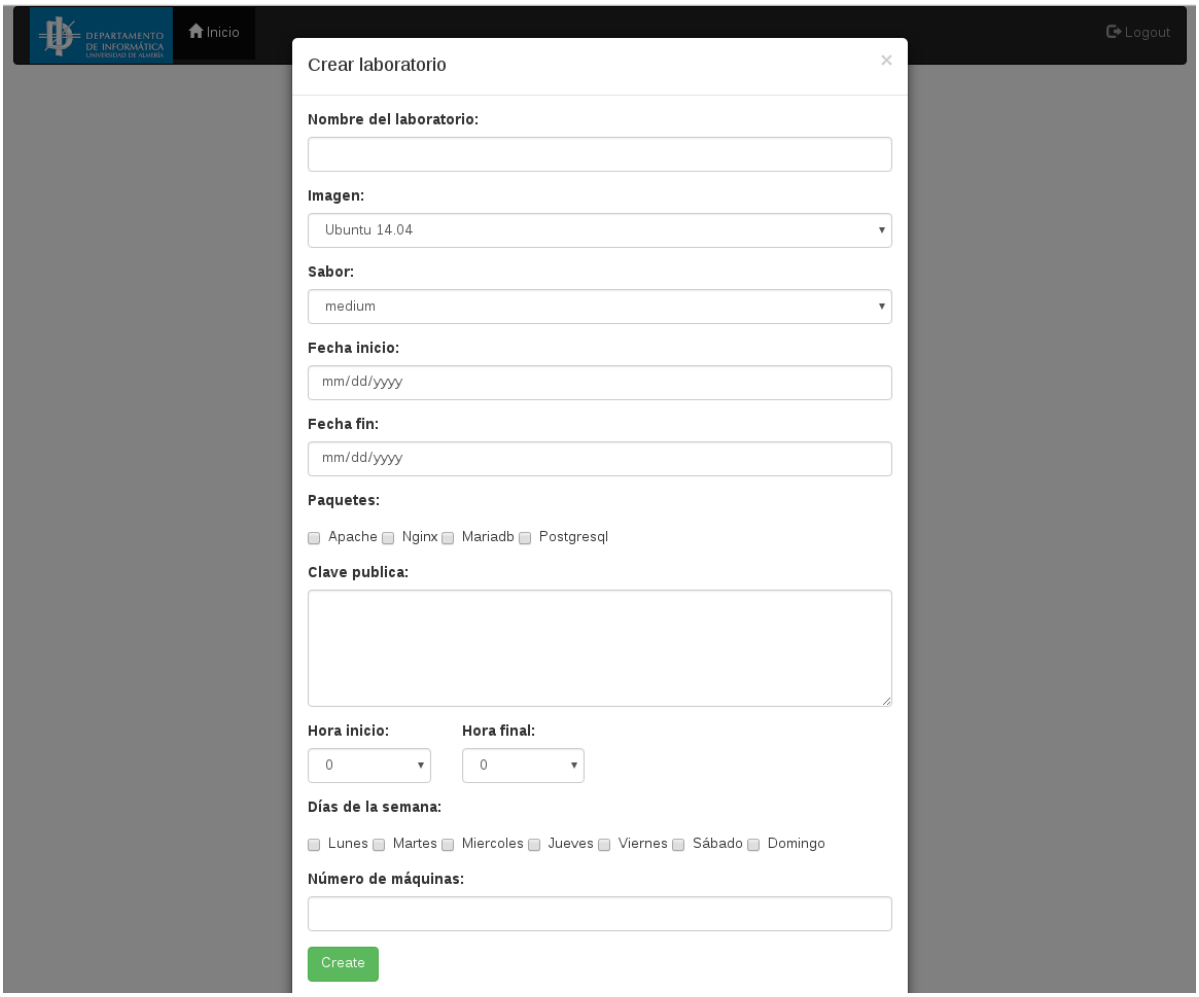

Figura 3.10. Imagen del formulario de creación de laboratorios.

Este formulario como se puede apreciar en la imagen tiene todos los campos necesarios para la creación de un laboratorio. Cabe destacar de este formulario que lleva a cabo la comprobación de errores de tipo con HTML 5 lo que permite ofrecer al usuario un ambiente común y confortable.

3.2 Construcción del sistema

3.2.1 Aplicación Web

Para la aplicación de este proyecto hemos decidido optar por un modelo de Aplicación Web, como hemos descrito en capítulos anteriores. Para esta aplicación Web hemos dividido el servicio en dos artefactos completamente autónomos. El primero es un servicio Web el cual atiende las peticiones que realizan los usuarios, el segundo servicio por el contrario tiene como finalidad crear una capa de abstracción sobre el núcleo de

la aplicación. A continuación, procederemos a describir más detenidamente cada uno de los servicios y la relación entre los mismos.

El servicio Web como hemos expuesto anteriormente atiende las peticiones de los usuarios. Estas peticiones pueden ser consultas de datos o modificaciones de los mismos. El servidor Web delega estas tareas sobre la API REST. Para ello le envía peticiones HTTP. Una vez que el servidor Web ha recibido la información desde la API REST, este genera una respuesta HTTP para el usuario. Para ilustrar dicho proceso hemos añadido a continuación un diagrama que esperemos clarifique dicho proceso al lector.

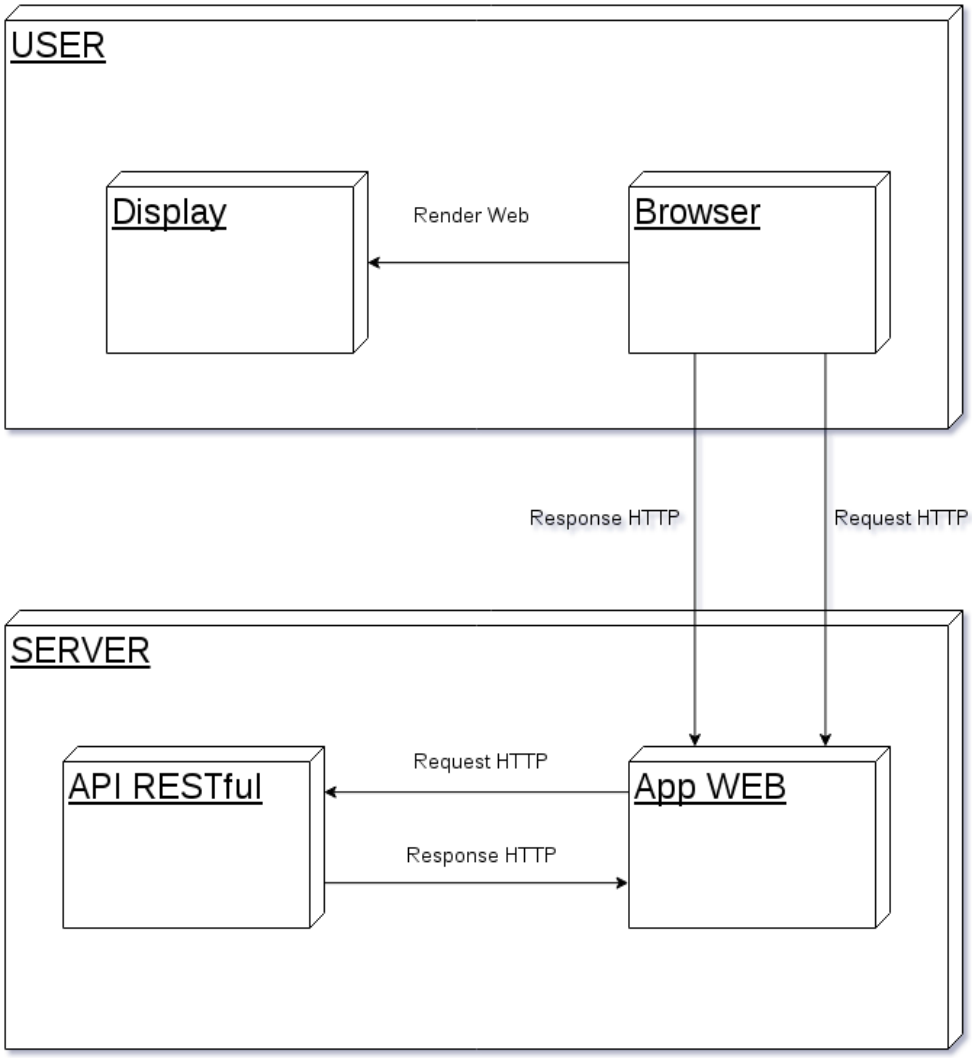

Figura 3.11. Diagrama de comunicación entre servicios

Tanto para nuestro servicio Web como nuestro Servicio API REST hemos empleado el framework Flask, el cual nos permite crear de forma sencilla servicios HTTP. El servicio API REST es el encargado de interactuar con el núcleo de la aplicación, del cual hablaremos en el próximo apartado.

3.2.2 Núcleo de la aplicación

En este apartado abordaremos el núcleo de nuestra aplicación. Hemos adjuntado un diagrama con las clases que componen el núcleo de la aplicación, con el cual esperamos que el lector pueda comprender mejor la relación entre las clases.

Comenzaremos hablando de la clase Application, esta contiene la totalidad de la funcionalidad de nuestra aplicación. El servicio API REST instancia la clase Application. A través de esta clase el servicio API REST se comunica con OpenStack y nuestra base de datos.

A su vez, la clase Application instancia un objeto de la clase OpenStack y otro de la clase DataBase. Estos objetos actuaran de drivers con los distintos servicios, permitiendo a nuestra aplicación comunicarse con ellos. En la siguiente imagen, mostramos un diagrama explicativo de la conexión entre los diversos subsistemas que componen la aplicación.

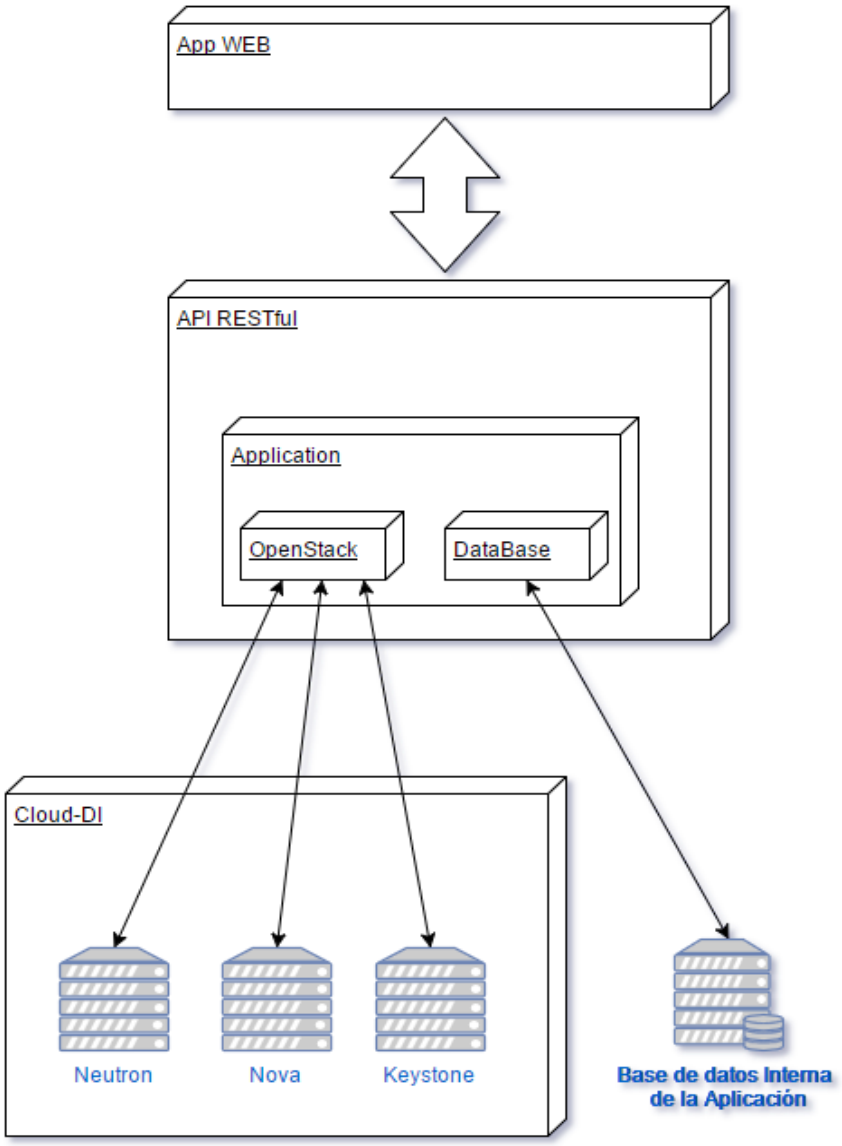

Figura 3.12. Diagrama de conexión de los diversos subsistemas de la aplicación.

En la imagen se puede observar que nuestro sistema está fuertemente modularizado. Esta jerarquía modularizada permite independizar fácilmente los diversos servicios. De esta forma un cambio en cualquiera de los subsistemas no requerirá modificaciones en el resto de la aplicación.

Un ejemplo de esto sería la sustitución del sistema de gestión de base de datos. En este caso solo sería necesario modificar la clase DataBase, aislando de este modo a la clase Application.

3.3 Conclusión

Como mostramos en este capítulo, hemos optado por un diseño modular para nuestra aplicación. Consideramos que desarrollar nuestra aplicación como un conjunto de servicios modulares nos permite mejorar el mantenimiento de la misma y simplificar la relación entre las diversas partes.

4 Conclusiones y posibles mejoras

La realización de este proyecto ha supuesto un gran reto, así como una gratificante experiencia. Este proyecto me ha permitido ampliar mis conocimientos en la rama de sistemas, permitiendo profundizar en áreas como la virtualización, la administración de redes y la administración de sistemas operativos. Además, la instalación y configuración de una plataforma cloud privada como es el caso de OpenStack, así como la interacción con dicho sistema me ha permitido ampliar mis áreas de conocimiento en la materia.

Este proyecto también me ha brindado la oportunidad de ampliar mis competencias en áreas de conocimiento que no he estudiado en profundidad durante estos cuatro años. Esta ampliación ha sido principalmente en la rama de ingeniería del software, permitiendo adquirir buenas prácticas y mejorando mis capacidades de diseño de software.

Por todo ello no me queda más que agradecer al departamento de informática de la universidad de Almería y en especial a mi tutor del proyecto por brindarme la oportunidad de llevar a cabo este proyecto que me ha hecho crecer profesionalmente.

4.1 Posibles mejoras

Como expusimos anteriormente este es un proyecto totalmente funcional, pero pese a ello consideramos que está en un estadio temprano de su desarrollo y por ello le convierte en proyecto aún moldeable y con mejoras que esperamos sean implementadas para mejorar la calidad de la experiencia de usuario. A continuación, enumeramos las mejoras que a nuestro juicio pueden suponer un avance en el proyecto.

- Implantar caducidad en los tokens de autenticación de usuarios.
- Implantar un sistema de colas como servicio (QaaS), que permita realizar las tareas más pesadas de forma asíncrona evitando así las largas esperas.
- Aumentar el número de paquetes que pueden ser instalados a cada laboratorio para hacer esta herramienta más versátil.
- Creación de test que mejoren drásticamente tanto el mantenimiento como la implantación de nueva característica a la aplicación.

Bibliografía

[1] Narayan Desai (abril 2016). OpenStack. Recuperado de: http://docs.openstack.org/

[2] MechIT Technologies (2016). Web Application. Recuperado de: http://www.mechittechnologies.com/webapp.php

[3] Margaret Rouse (septiembre 2005). JavaScript. Recuperado de: http://searchsoa.techtarget.com/definition/JavaScript/

[4] Margaret Rouse (septiembre 2005). Ajax (Asynchronous JavaScript and XML). Recuperado de: http://api.jquery.com/jquery.ajax/

[5] W3C (2014) HTML5. Recuperado de: https://www.w3.org/TR/html5/

[6] The Internet Society (junio 1999) RFC 2616. Recuperado de: https://www.ietf.org/rfc/rfc2616.txt

[7] Bill Appleton (agosto 2015). Build a reusable REST API back end. Recuperado de: http://www.infoworld.com/article/2960312/apis/build-a-reusable-rest-api-back-end.html

[8] Compute Hope (2016). C ++. Recuperado de: http://www.computerhope.com/jargon/c/cplus.htm/

[9] Margaret Rouse (septiembre 2006). Perl. Recuperado de: http://searchenterpriselinux.techtarget.com/definition/Perl

[10] Compute Hope (2016). Golang. Recuperado de: http://www.computerhope.com/jargon/g/golang.htm/

[11] Python Software Foundation. What is Python?: https://www.python.org/doc/essays/blurb//

[12] Codeeval (febrero 2015). Most Popular Coding Languages of 2015. Recuperado de: http://blog.codeeval.com/codeevalblog/2015#.V9GCuiiLSUk=

[13] Ryan Brown (enero 2016). Django vs Flask vs Pyramid: Choosing a Python Web Framework. Recuperado de: https://www.airpair.com/python/posts/django-flask-pyramid/

[14] Ryan Brown (enero 2016). Django vs Flask vs Pyramid: Choosing a Python Web Framework. Recuperado de: https://www.airpair.com/python/posts/django-flask-pyramid/ [15] Flask: http://flask.pocoo.org/

[16] Ryan Brown (enero 2016). Django vs Flask vs Pyramid: Choosing a Python Web Framework. Recuperado de: https://www.airpair.com/python/posts/django-flask-pyramid/

[17] Beltran de Heredia (mayo 2007). Why, oh WHY, do those #?@! Nutheads use vi?. Recuperado de: http://www.vim.org/

[18] Kristian Poslek (Agosto 2015). Building a desktop application with Electron. Recuperado de: http://electron.atom.io/

[19] The Chromium Projects. Chromium. Recuperado de: https://www.chromium.org/Home

[20] Node.js Foundation. About Node.js. Recuperado de: https://nodejs.org/en/about/

[21] Ben Collins-Sussman, Brian W. Fitzpatrick y C. Michael Pilato (mayo 2009). Version Control with Subversion. Recuperado de: http://svnbook.red-bean.com/en/1.6/svn.intro.whatis.html/

[22] Marcus Geduld (Agosto 2015). What is git and why should I use it? Recuperado de: https://www.quora.com/What-is-git-and-why-should-I-use-it

[23] Oracle. What is MySQL? Recuperado de: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html/

[24] MariaDB Foundation. About MariaDB. Recuperado de: https://mariadb.org/about/

[25] Jocelyn Fournier (noviembre 2015). MariaDB 10.1 vs MySQL 5.7: Real-world performances. Recuperado de: https://www.softizy.com/blog/mariadb-10-1-mysql-5-7-performances-ibm-power-8/

[26] OpenStack Fundation. DevStack: http://docs.openstack.org/developer/devstack/

[27] Chef. An Overview of Chef. Recuperado de: https://docs.chef.io/chef_overview.html/

[28] Puppet. How it works. Recuperado de: https://puppet.com/product/how-puppet-works

[29] Justin Ellingwood (octubre 2015). An Introduction to SaltStack Terminology and Concepts. Recuperado de: https://www.digitalocean.com/community/tutorials/an-introduction-to-saltstack-terminology-and-concepts/

[30] Ansible. Overview: How Ansible work. Recuperado de: https://www.ansible.com/how-ansible-works/

[31] OpenStack Foundation. The NovaClient Python API: http://docs.openstack.org/developer/python-novaclient/api.html

[32] OpenStack. NeutronClient Python API: http://docs.openstack.org/developer/python-neutronclient/usage/library.html

[33] SevOne. Monitoring Cloud Infrastructure Performance to Eliminate Visibility Gaps. Recuperado de: https://www.sevone.com/white-paper/monitoring-cloud-infrastructure-performance-eliminate-visibility-gaps

[34] OpenStack Foundation. Recuperada de: http://www.openstack.org/

[35] Tobias Günther (junio 2014). Switching from Subversion to Git. Recuperado de: https://www.git-tower.com/blog/switching-from-subversion-to-git/

[36] i2factory (febrero 2016). Qué es un servicio RESTFUL original de i2factory. http://www.i2factory.com/es/integracion/qu%C3%A9-es-un-servicio-restful

[37] Oracle. Getting Started Tutorial. Recuperado de: https://docs.oracle.com/cd/E19078-01/mysql/mysql-workbench/wb-getting-started-tutorial.html

[38] Google Inc. Herramientas para desarrolladores de Google Chrome https://support.google.com/dfp_premium/answer/4497389?hl=es

[39] Chef Software, Inc. Chef Supermarket: https://supermarket.chef.io/

[40] Puppet. Puppetforge: https://forge.puppet.com/

[41] Red Hat, Inc. Ansible Galaxy: https://galaxy.ansible.com/