Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería de Telecomunicación

Ingeniería de Telecomunicación
Licenciatura en Administración y Dirección de Empresas

# Proyecto Fin de Carrera

# Escalabilidad Automática de Aplicaciones en el *Cloud* PaaS


**Autor:** Beatriz Muñoz Manso
**Tutor:** Gregorio Robles Martínez
**Cotutor:** Henar Muñoz Frutos


Curso Académico 2012/2013

Proyecto Fin de Carrera
Escalabilidad Automática de Aplicaciones en el Cloud PaaS

Autor
Beatriz Muñoz Manso

Tutor
Gregorio Robles Martínez

Cotutor
Henar Muñoz Frutos

La defensa del presente Proyecto Fin de Carrera se realizó el día 20 de Junio de 2013, siendo evaluada por el siguiente tribunal:

Presidente:

Vocal:

Secretario:

y habiendo obtenido la siguiente Calificación:

Fuenlabrada, a 20 de Junio de 2013

*A mis padres*
*y mi hermano*

## Agradecimientos

Este Proyecto Fin de Carrera supone la culminación de una etapa de mi vida. Ha sido una experiencia realmente enriquecedora donde muchas personas habéis aportado vuestro granito de arena para que este camino llegue a su fin.

En primer lugar quiero agradecer a los profesores que he tenido en estos años. Gracias por toda la formación que me habéis dado. En particular, a Gregorio, mi tutor del proyecto, y uno de los mejores docentes que he conocido.

A todos mis compañeros que tanto me habéis enseñado durante mi estancia de prácticas en Telefónica. Gracias por acogerme, ha sido un auténtico lujo formar parte del equipo y no podría haber llevado este proyecto a cabo sin vosotros.

Gracias a todos mis amigos. Si no apareces en este párrafo (no puedo nombraros a todos), y eres uno de ellos, gracias. En especial, no puedo evitar nombrar a Rocío y Cris que habéis estado a mi lado aun en la distancia y os adoro, sois increibles. Mario y Javi sin vosotros este año hubiera sido mucho más complicado y me habéis tendido la mano. Gracias Fran, por todos los momentos juntos. Y Jesús, que decirte a ti... parte ya de mi familia. Mónica, Isa, Virgi y Abel, mis amigos segovianos, de esos que nunca falláis, y que me llevo en el corazón. A todos os pido que nunca cambiéis, sois extraordinarios.

Pablo, nunca sabré agradecerte todo lo que has hecho por mí a lo largo de estos años. *Por aguantar mis malos ratos y manías...*, por tu apoyo, cariño y comprensión. Siempre sabes sacarme una sonrisa. Gracias por quererme tanto. Te amo.

Agradecer a todos mis tíos y primos, y mis abuelas su apoyo incondicional. Gracias Henar, por darme la oportunidad de estar a tu lado, por enseñarme, ayudarme y por tu paciencia. Eres una de las mejores personas que he conocido nunca. Roberto, este año contigo me ha servido para conocerte mejor y quererte aún más si cabe. También formáis parte de este proyecto todos vosotros que ya no estáis aquí, porque desde esa estrellita en el cielo sé que me cuidáis y nunca dejareis de formar parte de mi vida, ¡os echo de menos!.

Y por último y más importante, mis padres. Gracias a vosotros he podido andar este camino, y me he convertido en la persona que soy hoy. Gracias por apoyarme, animarme y darme todo vuestro cariño. No me juzgáis, simplemente me queréis. Gracias por enseñarme todo, por los consejos y reprimendas, porque sin ellos no habría llegado hasta aquí. Me habéis enseñado el valor del trabajo y sacrificio, y sobre todo, siempre habéis estado a mi lado. Sois las personas más importantes de mi vida.

¡Gracias a todos!

# Contenido

# Índice de figuras

# Índice de tablas

# Abstract

The computing requirements for web applications are growing at high speed in the information technology age. Cloud computing is a computational paradigm which provides hardware and software infrastructure as a ubiquitous service based on a pay-per-use model. Besides, the cloud also offers software platforms that are suited to the demand at all times, and it means that they are scalable.

At platform level, scalability is a fundamental pillar since a service which does not satisfy the resource demand will not guarantee its availability to the users. Neither there are not so many convenient solutions in the market nor there is an easy way to do it.

**This project aims to provide a solution for PaaS level scalability** that is part of the *4CaaSt* project. Specifically, the work done in this project has involved solving several problems such as the contextualization of images, clonating of user-customized virtual machines, the monitoring for scalability driving and configuration of machines to allow it.

For automatic scalability of web applications on a cloud from a hardware infrastructure, this work develops a number of components and methods that are needed to achieve scalability at PaaS, based on IaaS capabilities. A service must be multi-tier to be scalable. Each virtual machine in the service will be deployed from an initial image that already allows monitoring and software installation severa.

The proposed solution has been developed in Java, using several technologies for data persistence, proper software development and the interaction of the with virtual machines with their management providers.

# Resumen

La necesidad de computación en las aplicaciones web crece a una gran velocidad en la era de la tecnología de la información. El *cloud computing* es un paradigma computacional donde se ofrece infraestructura de hardware y software como servicio de forma ubicua, bajo un modelo de pago por uso. El *cloud* ofrece además de servicios de infraestructura plataformas de software, que se adecúan a la demanda en cada momento, es decir, son escalables.

A nivel de plataforma, la escalabilidad es un pilar fundamental, ya que un servicio que no se adapte a la demanda, no garantizará su disponibilidad a los usuarios. A pesar de esta necesidad, no son muchas las soluciones viables para ello que actualmente están disponibles, ni existe una forma sencilla de llevarlo a cabo.

**En este proyecto se pretende ofrecer una solución para la escalabilidad a nivel PaaS** que se enmarca dentro del proyecto *4CaaSt*. Concretamente, el trabajo realizado en este proyecto se encarga de resolver varios problemas como son la contextualización de imágenes, el clonado de máquinas virtuales personalizadas por el usuario, monitorizar los servicios para saber en qué momento deben ser escalados y configurar las máquinas para permitirlo.

Para la escalabilidad automática de aplicaciones web en un *cloud* se desarrollan una serie de métodos y componentes necesarios para su gestión a nivel de plataforma a partir de una infraestructura provista en la nube. Para que un servicio sea escalable, debe ser multicapa. Las máquinas virtuales que forman el servicio partirán de una imagen base que permita la monitorización y la instalación de software.

La solución propuesta se ha desarrollado en Java, utilizado diversas tecnologías que permiten la persistencia de datos, el correcto desarrollo del software y la interacción con las máquinas virtuales de los proveedores junto con su gestión.

# Abreviaciones

| | |
|---|---|
| **API** | **A**pplication **P**rogramming **I**nterface − Interfaz de programación de aplicaciones |
| **AWS** | **A**mazon **W**eb **S**ervices |
| **BD** | **B**ase de **D**atos |
| **CPD** | **C**entro de **P**roceso de **D**atos |
| **CPU** | **C**entral **P**rocessing **U**nit − Unidad central de procesamiento |
| **DDL** | **D**ata **D**efinition **L**anguage − Lenguaje de definición de datos |
| **DML** | **D**ata **M**anipulation **L**anguage − Lenguaje de manipulación de datos |
| **EE** | **E**nterprise **E**dition − Edición de trabajo |
| **FQN** | **F**ully **Q**ualified **N**ame − Nombre completo cualificado |
| **HTTP** | **H**yper**t**ext **T**ransfer **P**rotocol − Protocolo de transferencia de hipertexto |
| **IaaS** | **I**nfrastructure **a**s **a** **S**ervice − Infraestructura como servicio |
| **IP** | **I**nternet **P**rotocol − Protocolo de Internet |
| **JDK** | **J**ava **D**evelopment **K**it − Kit de desarrollo en java |
| **JPQL** | **J**ava **P**ersistence **Q**uery **L**anguage − Lenguaje de Consultas Persistentes de Java |
| **JRE** | **J**ava **R**untime **E**nvironment − Entorno de ejecución de java |
| **JVM** | **J**ava **V**irtual **M**achine − Máquina virtual de Java |
| **KPI** | **K**ey **P**erformance **I**ndicator − Indicador Clave de Desempeño |
| **MAC** | **M**edia **A**ccess **C**ontrol − Control de acceso al medio |
| **NAS** | **N**etwork **A**ttached **S**torage − Almacenamiento en red |
| **ORM** | **O**bject-**R**elational **M**apping − Mapeo objeto-relacional |
| **OVF** | **O**pen **V**irtualization **F**ormat − Formato de virtualización abierto |
| **PaaS** | **P**latform **a**s **a** **S**ervice − Plataforma como servicio |
| **PFC** | **P**royecto **F**inal de **C**arrera |
| **POM** | **P**royect **O**bject **M**odel − Modelo de objeto de proyecto |
| **RAM** | **R**andom **A**ccess **M**emory− Memoria de acceso aleatorio |
| **REST** | **Re**presentational **S**tate **T**ransfer − Transferencia de estado representacional |
| **RIF** | **R**ule **I**nterchange **F**ormat − Formato de intercambio de reglas |
| **SaaS** | **S**oftware **a**s **a** **S**ervice − Software como un servicio |
| **SAN** | **S**torage **A**rea **N**etwork − Red de área de almacenamiento |
| **SCP** | **S**ecure **Co**p**y** − Copia segura |

| | |
|---|---|
| **SFTP** | **S**ecure **F**ile **T**ransfer **P**rotocol − Protocolo de transferencia segura de ficheros |
| **SGBD** | **S**istema de **G**estión de **B**ases de **D**atos |
| **SLA** | **S**ervice **L**evel **A**greement − Acuerdo de nivel de servicio |
| **SSH** | **S**ecure **Sh**ell − Intérprete de órdenes seguro |
| **SO** | **S**istema **O**perativo |
| **SOAP** | **S**imple **O**bject **A**ccess **P**rotocol − Protocolo de acceso simple a objetos |
| **SQL** | **S**tructured **Q**uery **L**anguage − Lenguaje de consulta estructurado |
| **SVN** | **S**ub**v**ersio**n** |
| **TI** | **T**ecnologías de la **I**nformación |
| **URI** | **U**niform **R**esource **I**dentifier − Identificador uniforme de recursos |
| **URL** | **U**niform **R**esource **L**ocator − Localizador de recursos uniforme |
| **VDC** | **V**irtual **D**ata **C**enter − Centro virtual de datos |
| **VLAN** | **V**irtual **L**ocal **A**rea **N**etwork − Red de área local virtual |
| **VM** | **V**irtual **M**achine − Máquina Virtual |
| **VPN** | **V**irtual **P**rivate **N**etwork −Red Privada Virtual |
| **XML** | e**X**tensible **M**arkup **L**anguage − Lenguaje de marcas extensible |

# Capítulo 1
# Introducción

El presente proyecto ha sido resultado de una beca de prácticas en Telefónica I+D, en un equipo de desarrollo encargado de trabajar en la innovación en el *cloud* mediante proyectos europeos. Entre ellos el proyecto *4CaaSt* [7].

*4CaaSt* tiene como objetivo crear una plataforma *cloud* multi−proveedor que aporte simplicidad al usuario tanto a la hora del despliegue de la propia plataforma, como al gestionar dinámicamente las aplicaciones que están alojadas sobre la misma.

El proyecto *4CaaSt* incorpora todas las características necesarias para facilitar la programación de aplicaciones complejas y permitir la creación de un ecosistema de negocio, en el que las aplicaciones que vienen de diferentes proveedores puedan ser adaptadas a distintos usuarios.

Concretamente, este Proyecto Fin de Carrera se encuadra dentro de ese proyecto, centrándose en la escalabilidad automática a nivel PaaS de aplicaciones alojadas en la nube.

## 1.1. Contexto

El *cloud computing* es un paradigma computacional que permite ofrecer servicios gracias a infraestructuras de hardware y software, que son accesibles al usuario a través de Internet. Ofrece un modelo de costes que consiste en pagar en función de la cantidad de recursos utilizados, es decir, acorde a la demanda del servicio. Además permite acceder desde cualquier lugar a los servicios alojados en la nube.

Dentro del *cloud* se ofrecen diferentes tipos de servicios. Un servicio IaaS proporciona un conjunto de recursos hardware, por ejemplo, almacenamiento de datos en la nube. Si además el servicio ofrece una plataforma que permita el desarrollo de aplicaciones se denominará servicio PaaS. Una página web de una empresa alojada en la nube sería un buen ejemplo de un servicio PaaS.

Una aplicación desplegada en el *cloud* sobre un servicio PaaS tiene asignados unos

recursos hardware y software distribuidos en una *Máquina Virtual* (VM) o varias. Para satisfacer la demanda del servicio los recursos necesarios pueden aumentar o disminuir en el *cloud* gracias a la capacidad de la nube de escalar.

La escalabilidad y el pago por uso son dos características muy importantes del *cloud computing*. La escalabilidad consiste en asignar los recursos necesarios a un servicio en función de las necesidades de capacidad, etc., que sean requeridas en cada momento. El usuario pagará en función de los recursos utilizados por el servicio.

### 1.1.1. Escalabilidad de un servicio

Muchos servicios tienen una demanda variable de usuarios que varía en función de la época del año, los días de la semana o a lo largo de las horas de un mismo día. Un ejemplo de ello es la gráfica de la figura 1.1, donde se muestra la demanda variable del servicio a través de la línea azul.

En un escenario mediante el que una empresa compra el hardware para mantener su servicio, puede suponer un coste excesivo de aprovisionamiento para obtener la infraestructura. Los recursos hardware necesarios deben garantizar la respuesta del servicio a todos los usuarios de cada momento. Observando la figura 1.1, los recursos deberían satisfacer la demanda máxima, que corresponde a la línea verde. Si no se pudiera garantizar esa demanda, no se ofrecería el servicio a los todos usuarios, como ocurre con la línea roja. Esto implica que si se pretende garantizar el servicio, sea cual sea la demanda, se realiza un sobre-coste, ya que cuando la demanda sea inferior, esos recursos no estarán completamente aprovechados.

Por lo tanto, o bien se garantiza la disponibilidad del servicio, con los costes que esto lleva asociados, sin utilizar en muchos casos todos los recursos disponibles; o bien se limitan los recursos, pero no se puede garantizar el servicio.

Supóngase ahora otro escenario en el cual, en lugar de la adquisición de los recursos hardware estos se alquilan. La empresa solamente pagará por los que utilice, adaptándose a la demanda en todo momento. En el la figura 1.1 los recursos alquilados variarían de forma proporcional con la demanda representada en la línea azul. Con esta opción se ahorrarían costes de infraestructura.

La solución consiste en desplegar el servicio en una plataforma *cloud*. En lugar de alojar el servicio en hardware físico adquirido por la empresa, se aloja la nube, que proporciona un acceso rápido y sencillo con la misma calidad.

Una de las características que hacen al *cloud* tan atractivo es el pago por uso. El coste dependerá exclusivamente de los recursos utilizados. Gracias a escalabilidad que proporciona la nube, los recursos variarán en función de la demanda.

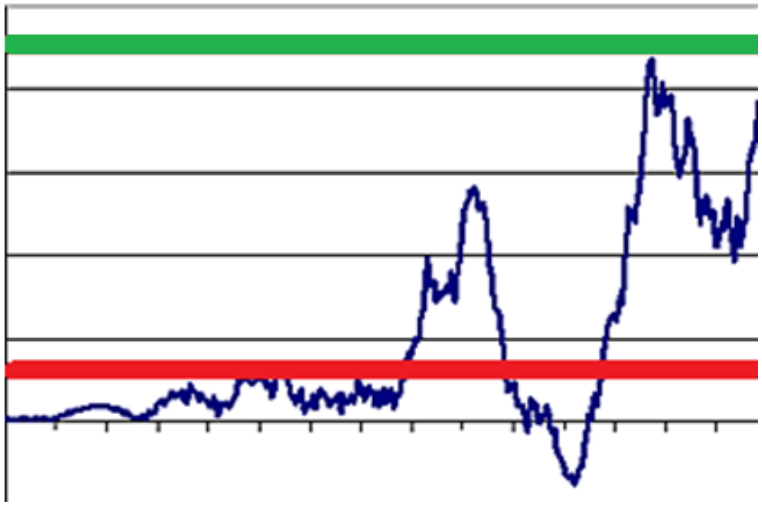


**Figura 1.1.** Demanda a lo largo del tiempo de un servicio

El cliente, que decide alojar un servicio en la nube, quiere ofrecer una calidad independiente de la demanda que exista en cada momento, y no se va a preocupar por los términos necesarios para conseguirlo. El servicio debe auto-gestionarse mediante la asignación de recursos automática, es decir, sin que el dueño del servicio intervenga cuando las necesidades varíen.

Para poder llevar a cabo la escalabilidad es necesario monitorizar cada una de las VMs que componen el servicio. Los datos obtenidos son analizados por el proveedor, y en función de ellos, dotará de más recursos al sistema o por el contrario se desasignarán. El uso de la CPU, por ejemplo, es uno de los indicadores que se utilizan para saber si es necesario añadir o quitar recursos, ya que si es muy elevado, será necesario tener más capacidad para dar el servicio, y si por el contrario el uso es muy bajo, se están infrautilizando los recursos, y parte de la capacidad sobra.

En un servicio IaaS, donde se busca obtener recursos típicamente hardware como almacenamiento, hay muchas soluciones para la escalabilidad. La escalabilidad basada en proporcionar nuevos recursos en forma de *Imágenes* funciona correctamente. Consiste en crear nuevas VMs a partir de imágenes existentes en el *cloud*, que tendrán las mismas características que las máquinas virtuales que ya forman parte del servicio. De esta forma, se obtiene mayor capacidad de una forma muy sencilla. Para reducirla simplemente se eliminarán las máquinas que supongan un exceso de recursos.

En un servicio PaaS, el alto grado de personalización de los recursos a utilizar supone un gran obstáculo a la hora de adaptarlos a la demanda del servicio en cada momento. Además de los recursos hardware, se ofrece software básico para la integración posterior de servicios desarrollados por el cliente, por lo que el escalado a través de imágenes no es válido.

## 1.2. Motivación

Un servicio PaaS ofrece, además de la infraestructura hardware propia de los servicios IaaS, servicios básicos para la escalabilidad de aplicaciones. Estos servicios se adaptarán a las necesidades de la aplicación que el usuario vaya a desarrollar y poner en funcionamiento sobre el sistema.

La escalabilidad en un servicio de plataforma es igual de importante que en un servicio IaaS. La adecuación de los recursos a la demanda del servicio permitirá el correcto funcionamiento de la aplicación.

Para la plataforma IaaS existen soluciones, como escalar a través de imágenes (ver sección 1.1.1). Obtener una solución que permita escalar un servicio PaaS no es tan intuitiva.

No se puede escalar las VMs a través de imágenes dadas por el proveedor, ya que será el propio usuario quien instale el software necesario para su servicio. Este software puede ser, por ejemplo, un gestor de bases de datos, o una aplicación desarrollada por el propio usuario. Si no es posible el escalado bajo demanda, no se podrá garantizar el servicio.

## 1.3. Problemas

Una solución que permita la escalabilidad de un servicio PaaS no es una tarea sencilla. Escalar a través de imágenes no es propio de un servicio de plataforma. No existen imágenes en el proveedor que se amolden específicamente a las necesidades de los usuarios.

Además, las métricas obtenidas para escalar un servicio tienen que ver habitualmente con los recursos hardware. En un servicio PaaS, es necesario tener en cuenta datos obtenidos de las VMs que correspondan a *Key Performance Indicators* (KPI) de la plataforma, como puede ser, en una aplicación web, el tiempo de respuesta, que resulta más representativo de la calidad que está ofreciendo el servicio. Por muy pequeño que sea el porcentaje de uso de la CPU (un KPI a nivel de infraestructura), si la aplicación web tiene un tiempo de respuesta muy grande, será que los recursos son insuficientes.

Los requerimientos de software hacen necesario separar en varias VMs el servicio, teniendo cada clase de ellas una funcionalidad concreta. Un servicio, por ejemplo, que aloje en una misma VM la aplicación y la base de datos encargada del almacenamiento persistente, al ser escalada, los datos serán replicados, pero en función de la máquina virtual que se utilice en cada momento para dar servicio al usuario, los datos almacenados variarán de una a otra, si se introducen tras el escalado.

La gestión de un servicio conformado por VMs con distintas funcionalidades, es posible solamente si las máquinas son capaces de interactuar entre sí. Además, al utilizar distintos tipos de VMs, es necesario que los datos de los KPIs obtenidos tengan una

máquina virtual asociada. Para ello debe considerarse la utilización de un identificador único en cada una de las máquinas que sea externo a los recursos hardware, ya que, por ejemplo, su dirección IP puede ser reasignada por el proveedor en algún momento.

A lo largo de este proyecto se proporciona y analiza una solución para la escalabilidad a nivel PaaS.

## 1.4. Objetivos

El objetivo de este proyecto final de carrera es, por tanto, desarrollar un software que permita escalar aplicaciones en plataformas *cloud* PaaS, independientemente del proveedor que se esté utilizando. Para conseguirlo se ha desglosado este objetivo principal en otros cuatro de menor envergadura, cuyo conjunto nos ofrezca la solución buscada. Los objetivos serían los siguientes:

- *Clonado de VMs configuradas.* Hasta ahora las VMs se clonaban utilizando una imagen dada por el proveedor. A nivel PaaS, se instalará software sobre una máquina que parte de la imagen base. El clonado se debe realizar una vez el software esté instalado.
- *Contextualización de las VMs.* Cada máquina tendrá que estar provista de un identificador único que ella misma conozca. Es necesario para el correcto funcionamiento de los sistemas de monitorización.
- *Interacción con los sistemas de monitorización.* La monitorización proporciona datos sobre el estado del servicio que se utilizarán para escalarlo. Será necesario que previamente las VMs a desplegar estén configuradas de forma que se puedan monitorizar. Las máquinas deben estar contextualizadas a través de un identificador único.
- *Configuración de las VMs.* Alojar un servicio en un *cloud* PaaS que sea escalable supone la utilización de varias máquinas virtuales que realicen diferentes funciones. Debe ser posible la comunicación entre ellas para ofrecer el servicio de forma satisfactoria.

## 1.5. Estructura de la memoria

La memoria del presente PFC consta de un total de siete capítulos y tres apéndices. El presente capítulo es el primero, seguido por el *Estado del arte* donde se expone desde la situación del *cloud computing* hasta la escalabilidad en la nube.

En los capítulos *Arquitectura general* e *Implementación y despliegue* se exponen la arquitectura necesaria para aplicaciones alojadas en el *cloud* y la solución propuesta. *Metodología y tecnologías utilizadas* es el capítulo donde se exponen los métodos y recursos utilizados para alcanzar la solución.

En el capítulo *Estudio de ahorro de costes*, como su propio nombre indica, se realiza una comparativa a nivel económico sobre el coste de alojar una aplicación web en

servidores físicos propios frente a alojarla en la nube. En las *Conclusiones y líneas futuras* se exponen los logros alcanzados, camino por el cual puede continuar el trabajo y una valoración personal sobre el proyecto.

Por último, los tres apéndices corresponden a la publicación del código (ver apéndice A), a la licencia de la memoria del proyecto (ver apéndice C), y a la planificación realizada (ver apéndice B).

No se ha realizado el presupuesto dado que, al ser un proyecto realizado en una empresa, los datos reales son confidenciales. Realizarlo de forma paralela podría llevar consigo la utilización de datos sensibles, que no deben ser publicados.

# Capítulo 2
# Estado del arte

En este capítulo se realiza una contextualización del presente Proyecto Final de Carrera para ilustrar al lector sobre los conceptos más importantes. Primeramente se explicará qué es el *cloud computing* al ser el pilar donde se asienta el trabajo realizado, así como tipos de servicios, entre ellos el PaaS, que se analizará en más profundidad y la escalabilidad, objetivo final de este estudio.

## 2.1. *Cloud Computing*

A modo de introducción, de una forma muy general y simple, se puede definir el *cloud computing* como una colección de recursos computacionales y de almacenamiento que puede crecer de forma «ilimitada». Entendiendo que la palabra ilimitada se utiliza en un contexto donde los recursos físicos tienen limitaciones, pero el usuario no los percibe.

Proporciona acceso ubicuo y bajo demanda a un conjunto compartido de recursos computacionales (redes, servidores, almacenamiento, aplicaciones y servicios) configurables, que pueden ser rápidamente aprovisionados con una gestión más sencilla. A través de esta tecnología se ofrecen servicios computacionales y de software. De esta forma, el usuario final no conoce la ubicación de los recursos ni se encarga de gestionarlos [8].

### 2.1.1. Orígenes del *Cloud Computing*

La evolución de las tecnologías de la información a lo largo de los últimos años ha provocado que las necesidades de las organizaciones hayan crecido a un mayor ritmo que la capacidad de procesamiento de los ordenadores físicos o personales. Por este motivo, se ha evolucionado a unas arquitecturas más complejas basadas en la ejecución simultánea de procesos en múltiples equipos informáticos.

A principios de la década de los noventa, las necesidades computacionales de alto rendimiento provocaron la creación de *clústeres*. Su desarrollo inicial fue posible gracias a los estándares abiertos, así como los sistemas *Unix*. Un *clúster* es una agrupación de ordenadores con componentes de hardware en común que se comportan como uno sólo.

A medida que fue creciendo la demanda de clústeres, éstos se fueron especializando para proporcionar servicios más concretos, y adaptándose a cada una de las organizaciones. Desde dichos centros se comenzaron a ofrecer servicios a usuarios ajenos, constituyendo la arquitectura de computación *Grid*, orientada al procesamiento en paralelo o a almacenar grandes cantidades de información. No obstante, la gran complejidad que suponía el uso de su infraestructura junto con la dificultad de utilizar diferentes *grids*, evitó su popularización a nivel comercial.

En paralelo, comenzaron a popularizarse tecnologías de *virtualización*, ver sección 2.2, que permitían implementar máquinas virtuales. En estas VMs, el hardware y el software no se encuentran en la misma máquina física. También permiten copiar o replicar el entorno sin tener que instalar de nuevo el software requerido. La virtualización permite reducir el coste total de propiedad del hardware y servicios. Los recursos se optimizan, ya que solamente se utilizan los que son estrictamente necesarios. El menor consumo de recursos implica una disminución de los costes de aprovisionamiento y gestión.

Esta nueva arquitectura permite distribuir carga de trabajo de forma sencilla, lo cual soluciona algunos de los problemas que tiene la arquitectura *grid*, abriendo una nueva puerta, el ***cloud computing***.

Este nuevo modelo surge como una solución capaz de proporcionar recursos de cálculo y de almacenamiento que, además, resulta especialmente apto para la explotación comercial de proveedores de servicios en Internet.

El *cloud computing* se refiere, por tanto, a un conjunto de software, aplicaciones y servicios que son accesibles al usuario a través de Internet. Está soportado por una infraestructura física que proporciona capacidad de almacenamiento, computación y comunicación. El *cloud computing* permite almacenar datos, procesar información y dar acceso para la comunicación con el exterior.

Una de las características más importantes del *cloud computing* es el pago por uso. Solamente se paga por lo que se utiliza. Esto supone una disminución de los costes muy importante para las organizaciones, como se analizó en el capítulo 1, y es lo que hace tan atractivo el *cloud* [8].

### 2.1.2. Características Esenciales

Como se ha podido ver, tanto a partir de la definición de *cloud computing*, como su evolución histórica, no es fácil determinar con precisión todos los conceptos y ventajas que supone la computación en la nube. Pero sí se pueden establecer seis características esenciales o pilares básicos:

**Auto-servicio bajo demanda.** El proveedor *cloud* ofrecerá la capacidad de aprovisionar recursos de forma transparente al cliente de un servicio. Para que se cumpla

el supuesto «bajo demanda», dicho proveedor utilizará una serie de componentes o softwares que eviten procesos de aprobación y configuración por parte del usuario. Un ejemplo de ello sería la virtualización. El cliente no se encarga de realizar ningún tipo de operación para adaptarse a las necesidades del servicio, sino que es el proveedor *cloud* quien se encarga de gestionar los recursos a disposición del usuario [9].

No puede haber, por tanto, demora en la provisión de servicios. La intervención del usuario se limita a especificar las características que desea. Para lograr estos objetivos, los proveedores ofrecen interfaces de gestión y de monitorización de los recursos de fácil utilización por el cliente [9].

**Acceso ubicuo y estándar.** Los servicios de *cloud* y los recursos computacionales usados, deben ser accesibles de forma ubicua (independiente de localización) al usuario a través de redes y tecnologías estándares, como SSH y vía web, para cualquier terminal o dispositivo (incluyendo los dispositivos móviles). Este es uno de los mantras del *cloud computing*: siempre disponible, desde cualquier sitio [9].

**Compartición de recursos.** El proveedor del servicio dispone de recursos que pueden ser compartidos a bajo nivel entre múltiples clientes. Un único servidor físico es compartido por varios usuarios. Se virtualiza en servidores virtuales, independientes entre sí, y cuya seguridad y rendimiento no afecta a los otros servidores virtuales. La compartición de recursos debe ser transparente al usuario, incluso bajo provisión dinámica, donde los recursos pueden ser reasignados a otros clientes en el caso de que dejen de usarse [9].

**Provisión automática.** A través de software se establecen procedimientos que sistematizan el arranque, parada y provisión de máquinas virtuales, a partir de recursos hardware o la *Infraestructura Física* [9].

**Elasticidad.** La elasticidad es la capacidad de una plataforma de crecer de forma planificada. Los recursos ofrecidos deben ser percibidos como ilimitados para el cliente. Y por ello, el cliente debe poder modificar su demanda elásticamente, esto es con facilidad y rapidez aumentando o disminuyendo los recursos contratados, incluso de forma automática. Es una de las características más apreciadas y buscadas de *cloud computing*, siendo un claro factor diferenciador. Contando con los suficientes recursos hardware es posible instanciar tantas máquinas virtuales como el usuario del servicio desee, con independencia del número de servidores físicos en los que se creen dichas VMs, gracias a la provisión automática [9].

**Pago por uso.** El consumo de cada uno de los recursos utilizados es medido adecuadamente por el proveedor y reportado al cliente para su conocimiento. Esto permite establecer mecanismos de pago como el *pago por uso*, donde el coste del servicio depende de la utilización de recursos realizada. Así, los modelos de costes crecen con el uso del servicio, por lo que no son necesarias inversiones en previsión de una determinada

demanda ni se desperdician recursos no utilizados. El usuario, en lugar de adquirir el hardware y el software, lo alquila [9].

### 2.1.3. Modelos de servicio

La clasificación en modelos de servicio es la más conocida y utilizada. Clasifica las soluciones de *cloud computing* en función de los servicios ofertados. Hay tres modelos que se ofertan como servicio, que están representados en la figura 2.1, también llamados *niveles* del *cloud* [9][8].

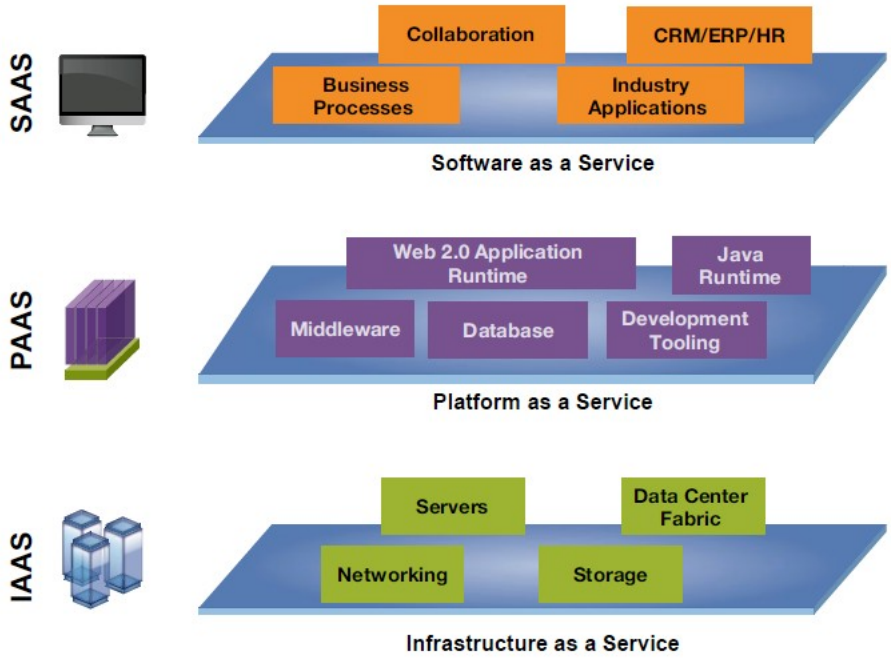



**Figura 2.1.** Modelos de servicio en el *cloud* [1].

**IaaS (Infrastructure as a Service - servicio de infraestructura:)** El proveedor pone a disposición de los usuarios una serie de recursos computacionales (CPU, memoria, ancho de banda, almacenamiento, etc.) que son utilizados por el cliente a bajo nivel; desde la gestión del sistema operativo, a la instalación y despliegue de las aplicaciones; teniendo libertad absoluta sobre el funcionamiento de las mismas. La infraestructura, si bien no se gestiona directamente por el usuario, está expuesta directamente al cliente para su uso.

**PaaS (Platform as a Service - servicio de plataforma:)** La infraestructura *cloud* en este modelo de servicio se ofrece al cliente junto con servicios básicos de aplicaciones. En este caso, la oferta de los proveedores de *cloud computing* son plataformas de software. No sólo consta de recursos sino también de servicios externos, ya configurados, para la simplificación de la creación de las infraestructuras soportadas sobre *cloud* como

sistemas de bases de datos, de mensajería asíncrona, balanceadores de carga, etc.

Estos servicios de plataforma se ofrecen como servicio mediante un *Application Programming Interface* (API), de forma que el usuario puede construir sus aplicaciones en base a ello. Es decir, a través de redes de servicio IP, el cliente puede acceder a todas las especificaciones necesarias para crear aplicaciones web.

El cliente no suele pagar en función de los recursos computacionales usados. En el modelo PaaS, se utilizan métricas de la plataforma, el número de conexiones a base de datos y el número de sesiones en el servidor web para establecer los costes del mantenimiento de las aplicaciones en el *cloud*.

**SaaS (Software as a Service - software como un servicio:)** El proveedor utiliza una infraestructura de *cloud* para crear un servicio completo (una aplicación final) y servirlo para su uso vía web. El cliente no suele pagar en función de los recursos computacionales usados, sino en a través de cálculos realizados con métricas asociadas a la aplicación usada, como número de usuarios, espacio en disco, etc. Es decir, el proveedor ofrece su aplicación a los clientes para su uso como un servicio bajo demanda. Las aplicaciones son provistas por el proveedor, y el usuario apenas puede personalizarlas o adaptarlas a sus necesidades.

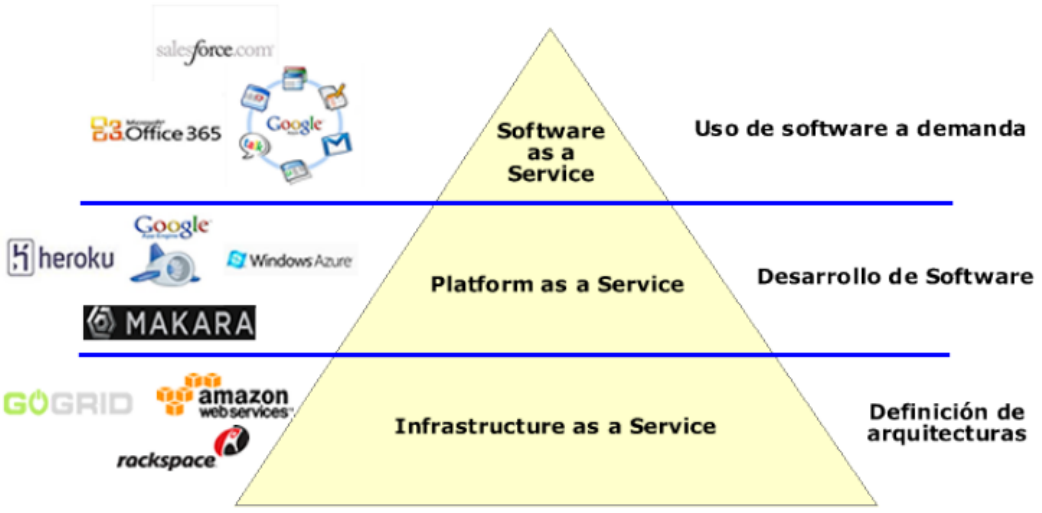



**Figura 2.2.** Algunos productos existentes en cada uno de los modelos de servicio en el *cloud* [2].

En la figura 2.2 se muestran productos relacionados con cada uno de los tres modelos de servicio. A nivel IaaS, se ofrecen los recursos físicos como memoria y redes. A nivel PaaS, los productos permiten el desarrollo de software en base a plataformas. El software bajo demanda se ubica en el nivel SaaS.

### 2.1.4. Configuraciones de *cloud* computing

Existen cuatro posibles configuraciones básicas para desplegar servicios de *cloud computing*:

- **Nube pública.** Un proveedor, que posee los recursos, ofrece los servicios al público en general. Debe haber mecanismos estrictos de separación en diferentes clientes usando la infraestructura. Los recursos son, por tanto, externos al usuario. Es la configuración más habitual.
- **Nube privada.** Con las mismas técnicas que una nube pública, se instalan un conjunto de recursos computacionales con características de provisión elástica y gestión dinámica, para uso de una organización con carácter privado, simplificando la infraestructura de TI.
- **Nube comunitaria.** Con objeto de consolidación de infraestructuras y tecnologías, una nube privada puede ser compartida por una agrupación de entidades. No es muy habitual. En su lugar, suele usarse una nube pública en estos casos.
- **Nube híbrida.** Es una combinación de nubes pública y privada. Permite conectar servicios propios de una infraestructura de TI con otros servicios en la nube. Típicamente se enlazan mediante mecanismos seguros como VPNs (redes privadas virtuales) para integrar todos los servicios [8][9].

## 2.2. Virtualización

La virtualización es una tecnología que permite ejecutar varios servidores virtuales dentro de un mismo servidor físico, de forma simultánea. Es uno de los principios básicos sobre los que se asientan las infraestructuras del *cloud*. La virtualización a nivel de *Hipervisor* es una tecnología que permite ejecutar varios servidores virtuales dentro de un mismo servidor físico, al mismo tiempo [9]. Crea un entorno simulado en el cual en una única máquina física se proveen servicios para:

- Crear entornos de ejecución aislados e independientes
- Dividir los recursos hardware
- Crear VM que ejecutan sistemas operativos potencialmente distintos
- Monitorización y gestión centralizada de todas las máquinas virtuales en ejecución, o al menos, todas estarán preparadas para ello.

### 2.2.1. Tecnologías de virtualización

Las tecnologías de virtualización emanan del mundo de la seguridad, donde para infraestructuras de TI, cada servicio ha de ejecutarse, por razones de aislamiento e independencia, en máquinas distintas. Este escenario crea, sin embargo, una multiplicidad de servidores físicos y en consecuencia, una multiplicación de los costes de adquisición y, sobre todo, de gestión de los mismos. En este escenario mencionado, los recursos hardware están desaprovechados, porque las cargas medias son bajas y los recursos no utilizados de

una máquina no pueden ser aprovechados por otros servicios. Existen básicamente dos tecnologías diferentes de virtualización, que son la paravirtualización y la virtualización completa.

**Paravirtualización.** Virtualización que requiere cooperación del sistema operativo (SO) virtualizado. Es necesario modificar el sistema operativo a virtualizar para que se coordine con el hipervisor. Sólo es posible con SOs de los que se disponga código fuente libre. La paravirtualización, a cambio, permite obtener rendimientos muy próximos al del sistema operativo nativo.

**Virtualización completa.** A través de esta tecnología se emulan todos los recursos hardware para permitir la ejecución de sistemas operativos sin modificar (requisito para SOs propietarios) con el coste de un rendimiento inferior o muy inferior a la paravirtualización (excepto con soporte hardware de virtualización).

La virtualización permite ejecutar en un mismo servidor físico máquinas virtuales independientes que maximizan el uso de recursos manteniendo el principio de aislamiento.

Una vez que explicada la virtualización y sus características principales, ya es posible trazar la sencilla relación que mantiene con *cloud computing*. La virtualización es la respuesta a las necesidades de compartición de recursos, provisión automática, elasticidad, monitorización y pago por uso, (ver sección 2.1.2) [10].

## 2.3. Proveedores *cloud*

Tras el gran *boom* del *cloud*, actualmente existen decenas de proveedores que ofrecen servicios en la nube, tendencia que sigue al alza. La mayoría de estos proveedores ofrecen las tres modalidades de servicio a través de productos diferenciados. Algunos de estos proveedores son IBM, Oracle, Salesforce, Microsoft, Google o Amazon.

**Amazon Web Services (AWS)** Es el pionero y, sin duda, el proveedor número uno mundial de servicios en la nube. Así Amazon, aprovechando la disponibilidad de recursos libres, se dedicó a buscar un nuevo modelo de negocio basado en la venta de este exceso de capacidad ofertando un *cloud* IaaS llamado *Elastic Compute Cloud* más conocido como *EC2*. Ha ido evolucionando y actualmente tiene más de una veintena de productos que se pueden contratar tanto de manera independiente como combinada para satisfacer las necesidades de cada usuario a bajo coste, con gran elasticidad y flexibilidad, y de forma segura. Dentro de los productos y servicios que ofrece, el más conocido y utilizado es EC2, a nivel IaaS. Proporciona capacidad informática escalable, utilizando además el sistema de pago por uso. A nivel PaaS ofrece, por ejemplo, el servicio *AWS Elastic Beanstalk* [11].

**Google** Google es una compañía estadounidense que comercializa varios productos *cloud*. Al igual que Amazon, tiene servicios a nivel IaaS, PaaS y SaaS. Además, Google utiliza sus propias tecnologías *cloud* para la gestión interna de publicidad y búsquedas,

entre otras cosas.

Ofrece servicios SaaS gracias a un conjunto de aplicaciones en la nube, que son accesibles por los usuarios. A este nivel, son muy conocidos los servicios ofertados al usuario doméstico como *Gmail* para el correo electrónico, *Google Calendar* para compartir un calendario entre usuarios, *Google Talk* para envío de texto y llamadas de voz, *Google Docs* para la creación, modificación y acceso a documentos. También ofrece servicios PaaS a través de *Google Apps*, donde se encuentra gran cantidad de software también accesible a través de Internet [12].

**Flexiant.** Antes Xcalibre, es una empresa dedicada a temas de hosting del Reino Unido. Tiene varios productos, entre ellos Flexiscale, que provee servicios a nivel IaaS, siendo uno de los más importantes a nivel europeo. Los usuarios de este servicio (Flexiscale) tienen la posibilidad de crear, arrancar y detener servidores según sus necesidades. Para realizar este tipo de acciones, proporciona una API al usuario, además de poder gestionarlo a través de un panel de control en su portal.

Flexiscale ofrece al usuario direcciones IP públicas para las VMs, y VLANs dedicadas a cada cliente, además de una recuperación automática ante fallos de hardware en menos de quince minutos (respaldada por *Service Level Agreement* (SLA)). Utiliza nodos redundantes mediante redes de almacenamiento [13].

Este proveedor no es de los más relevantes en lo que a cifra de negocio mundial se refiere. No obstante ha sido Flexiscale el producto *cloud* utilizado para el desarrollado este proyecto.

Además de éstos, hay infinidad de proveedores *cloud* como son Salesforce, Rackspace, Microsoft, Oracle, RightScale, Sun Microsystems, IBM, Cisco Systems y VMware entre otros.

### 2.3.1. Productos *cloud* en modelos de servicio PaaS

Como ya se ha abordado en este capítulo, la mayoría de los proveedores *cloud* ofrecen productos a los distintos niveles, IaaS, PaaS y SaaS. A lo largo de esta subsección se abordará la solución *cloud* ofertada tanto por Amazon Web Services, como por Google a nivel PaaS. En la figura 2.2 se muestran algunos productos existentes, clasificados en función del modelo de servicio que ofrecen.

**AWS Elastic Beanstalk:** *Elastic Beanstalk* es el producto a nivel PaaS que ofrece AWS. Permite al usuario desplegar aplicaciones ofreciendo un conjunto completo de recursos de infraestructura y software. Una vez que se carga una aplicación, Elastic Beanstalk se encarga de la implementación, del aprovisionamiento de capacidad, balanceado de carga y escalado de la misma. También se basa en EC2, por lo que tiene todas las funcionalidades ofertadas a nivel IaaS. Para proporcionar este servicio Elastic Beanstalk utiliza otros servicios de AWS.

- *EC2.* Para los recursos informáticos, como las máquinas virtuales.
- *Amazon Simple Storage service (Amazon S3).* Este producto se encarga del almacenamiento.
- *Elastic Load Balance.* Distribución de la cargar de Amazon EC2.
- *Auto Scaling.* Para escalar de forma automática la capacidad de Amazon EC2.

La funcionalidad que ofrece el servicio *Auto Scaling* es amplia, ya que permite desplegar nuevas versiones de las aplicaciones en los entornos ya desplegados, o volver a versiones anteriores. También monitoriza las VMs desplegadas y el entorno, obteniendo datos como medidas de CPU y latencia media. Todos los cambios que se produzcan en las aplicaciones y en el entorno son notificados por correo electrónico al cliente [14].

**Google App Engine:** *Google App Engine* es la solución a nivel PaaS que ofrece Google. Permite alojar aplicaciones Web en la infraestructura de la compañía de forma rápida y segura. El desarrollo de las aplicaciones es sencillo a través de servidores web dinámicos. Consta de una API para autenticar usuarios y acceder a otros servicios de Google. Proporciona un entorno de desarrollo local. Cabe destacar que tiene una zona de pruebas, donde correr las aplicaciones en la web desde el entorno local. Esto permite aislarlas en el propio equipo físico, proporcionando todas las medidas de seguridad necesarias para evitar conexiones indeseadas y escritura en el sistema de archivos.

Utiliza *BigTable* como motor de bases de datos, por lo que permite gran almacenamiento distribuido, lo que le proporciona una gran elasticidad. Gracias a la utilización de este *datastore*, se obtienen grandes ventajas de escalabilidad, control de versiones de las aplicaciones y optimización en la lectura de datos.

La reutilización de soluciones desarrolladas para el ámbito del *cloud*, así como la facilidad de interacción entre los servicios propios, es una de las grandes ventajas que ofrece [15].

## 2.4. Escalabilidad en el *cloud*

Se define *escalabilidad* como la capacidad de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejando el crecimiento continuo de trabajo de manera fluida, o bien estando preparado para hacerse más grande sin perder calidad en los servicios ofrecidos. La capacidad de ampliación de un sistema depende de las limitaciones que tenga el diseño. Permite reducir los costes y aumentar la flexibilidad y fiabilidad utilizando recursos hardware externos. Cabe destacar que tan importante es la escalabilidad para el aumento de recursos, como para la disminución de los mismos, para evitar un sobre-aprovisionamiento cuando la carga de trabajo disminuya. Existen dos tipos de escalabilidad que se pueden llevar a cabo en la nube, que son vertical y horizontal.

**Escalabilidad vertical.** La escalabilidad vertical consiste en aumentar la capacidad de una VM modificando los recursos hardware. Por ejemplo, como se observa en la figura 2.3, aumentar o reducir la memoria RAM, o el número de CPUs. La ampliación está limitada por la capacidad del hardware disponible. Al ser necesario modificar los recursos hardware, se producen tiempos de inactividad del software [16].

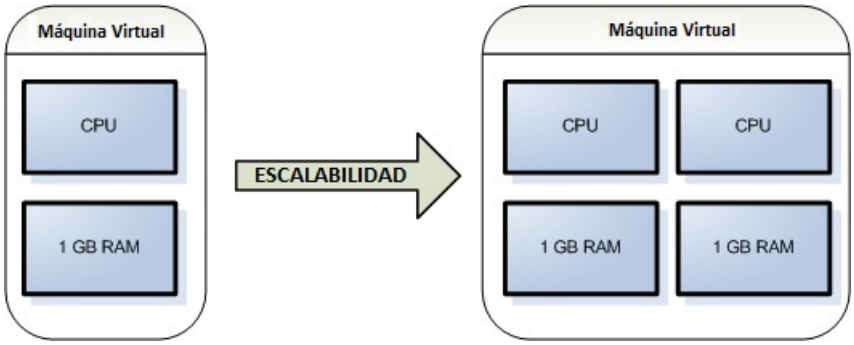



**Figura 2.3.** Escalabilidad vertical [3].

**Escalabilidad horizontal.** La escalabilidad horizontal aumenta o disminuye la capacidad del sistema añadiendo o quitando VMs. Si para dar funcionalidad a la aplicación tenemos una VM, y necesitamos mayor capacidad, se crea otra con las mismas características. Para poder realizar este tipo de escalado el software utilizado tiene que estar distribuido en capas.

Cada una de las VM pertenecientes a la misma capa, tendrá asignados los mismos recursos hardware y software. Los nodos serán homogéneos si están destinados a las mismas acciones. Gracias a esta similitud, el balanceo de carga que se realiza entre los nodos será más sencillo. Además las VMs serán independientes unas de otras.

La limitación que tiene la escalabilidad horizontal depende de la eficiencia de los nodos. Será eficiente mientras al añadir un nodo se cree un valor añadido para el sistema. En la figura 2.4 se observa un escalado horizontal de forma gráfica.

Es deseable que la capacidad de escalar de un sistema sea tanto a corto como a largo plazo. A corto plazo para permitir una rápida respuesta en función de la capacidad demandada, así como a largo plazo, para mantenerla en el tiempo.

Para que un sistema sea auto-escalable debe estar provisto por tres elementos fundamentales que son el balanceo de carga, la monitorización de recursos y el establecimiento de reglas de escalabilidad [16].

**Balancear** la carga significa repartirla entre varios nodos o VMs. Si tenemos una aplicación web desplegada en varios nodos del *cloud*, el balanceador se encargará de redirigir las peticiones a la VM que considere, en función de la carga que tenga en ese

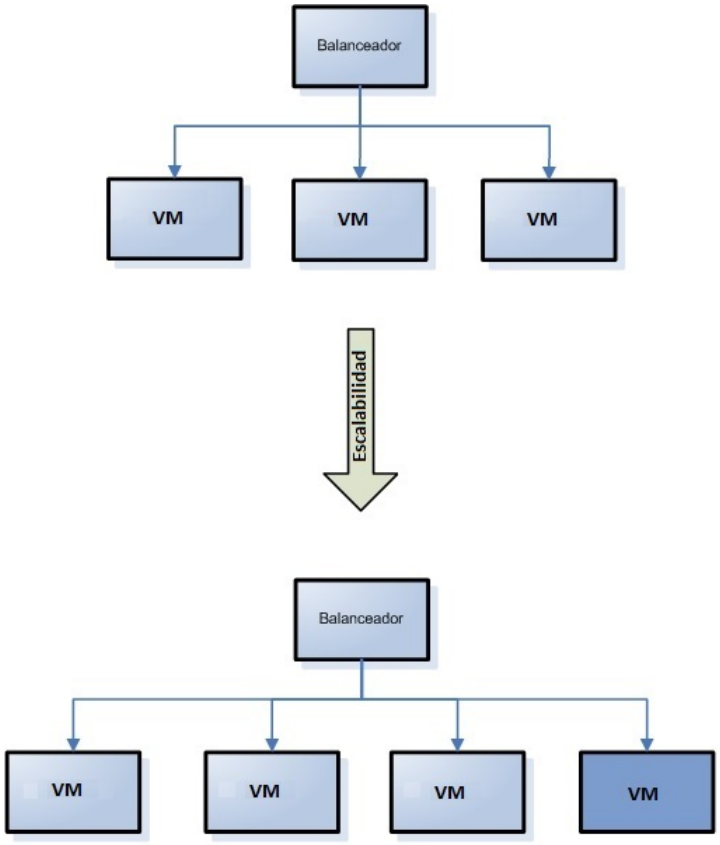



**Figura 2.4.** Escalabilidad horizontal [3].

momento cada una de ellas.

Los sistemas serán **monitorizados** para obtener datos sobre los recursos que se están utilizando y otros KPIs que sean significativos.

Estableciendo unas **reglas** con umbrales para el escalado, en función de los valores de las métricas monitorizadas se tomarán las decisiones de aumentar o disminuir la capacidad [17].

Una de las ventajas más importantes del *cloud computing* en el nivel **IaaS** es la facilidad y rapidez para poder escalar los sistemas en función de la demanda. Tan importante es esta propiedad como la posibilidad de "des-escalarlos". Cabe la posibilidad de que sea el propio cliente quien escale los recursos de forma manual, en función de sus necesidades, de forma que no sería el proveedor quien se encargara de ello, lo que en muchos casos entorpecería la eficiencia del sistema [18].

A día de hoy, el *cloud* IaaS es el que está más desarrollado en lo que a escalabilidad automática se refiere, permitiendo realizar un escalado horizontal añadiendo réplicas de un servicio, y escalado vertical añadiendo más recursos hardware a las VM ya desple-

gadas, o disminuyéndolos. Los mecanismos de escalabilidad a bajo nivel implementados por IaaS basan su eficiencia en la recogida y análisis de métricas propias de este nivel como pueden ser el uso de la CPU, datos almacenados, rendimiento de la RAM, etc. El proveedor *cloud* define una serie de reglas que permiten escalar el sistema en función de los datos recogidos por estas métricas. Estas reglas pueden ser establecidas también por los propios usuarios. No obstante, las métricas recogidas a nivel de infraestructura resultan insuficientes a la hora de escalar a nivel PaaS o SaaS, ya que sólo se centran en el consumo de los recursos físicos de las aplicaciones o del servicio en sí [17].

Por último, en el nivel del SaaS la escalabilidad también puede referirse a la posibilidad de aumentar el número de usuarios que pueden acceder a la aplicación, propiedad muy deseable y que se puede realizar con facilidad y rapidez gracias a las tecnologías desarrolladas actualmente para ello. En la siguiente sección se estudiará más en profundidad la escalabilidad para nivel PaaS, ya que es el objetivo de este trabajo [16].

### 2.4.1. Escalabilidad en un modelo de servicio PaaS

Los sistemas alojados en *clouds* PaaS se aprovechan de los recursos del nivel de infraestructura para poder escalar. Es necesario que los nodos estén separados en capas, ya que se preferirá un escalado horizontal de los recursos. El esquema básico de un sistema desplegado en un *cloud* a nivel PaaS se muestra en la figura 2.5. El almacenamiento de datos se encuentra separado de los balanceadores y de los contenedores de aplicaciones [16].

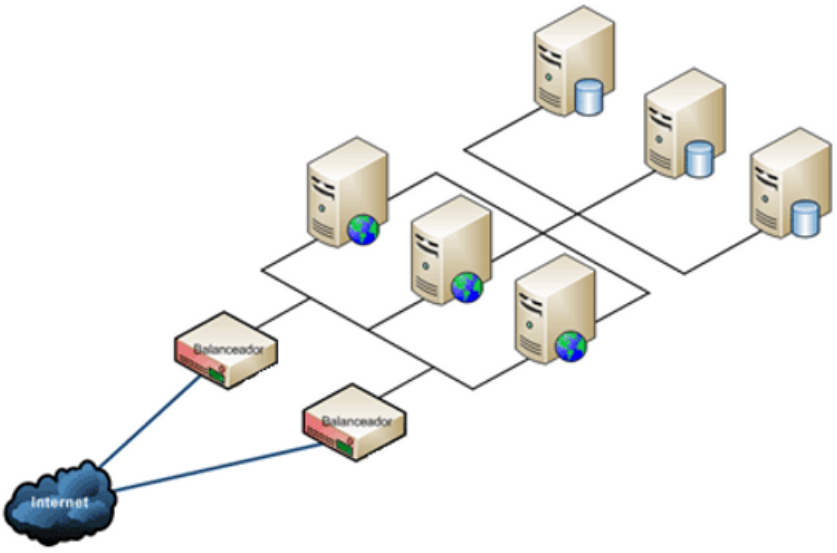



**Figura 2.5.** Sistema en un *cloud* a nivel PaaS

Los balanceadores son imprescindibles para la escalabilidad, como se explicó en la sección 2.4. Sin embargo, los KPIs monitorizados referentes a la utilización del hardware no resultan tan representativos para el funcionamiento de la plataforma, que es lo que se ofrece en un PaaS. Se tienen en cuenta otras métricas más representativas a nivel

de plataforma como el tiempo de respuesta y el número de usuarios simultáneos. El número de **usuarios simultáneos** se entiende como el número de usuarios activos en un intervalo de tiempo. El **tiempo de respuesta** indica el tiempo transcurrido desde que un usuario realiza una petición hasta que obtiene la respuesta. Para un gestor de bases de datos, por ejemplo, el **número de conexiones** sería un KPI muy representativo.

Estos KPIs son mucho más representativos del funcionamiento del servicio que ofrece el sistema. Para atender la demanda de los usuarios, con unos umbrales de calidad definidos, es preferible utilizar este tipo de métricas relacionadas con el funcionamiento del sistema. Las reglas que se establezcan para el escalado automático estarán referidas a los KPI a nivel PaaS definidos.

Un sistema como el de la figura 2.5 será, por tanto, escalable a través de las métricas recogidas y las reglas establecidas. Las reglas actúan a través de umbrales que pueden ser definidos por el usuario o por el proveedor.

La complejidad de la gestión de escalabilidad del sistema radica en las relaciones que existen entre las distintas VMs, ya que las que balancean deben conocer las máquinas a las que deben enviar la carga de trabajo. Las máquinas virtuales que se encargan del almacenamiento de datos han de estar registradas en las máquinas que van a acceder a ellas. De ahí surge la necesidad de que exista un manejador del escalado del sistema que sea capaz de interactuar con todas las VMs desplegadas [16].

# Capítulo 3
# Arquitectura general

En el tercer capítulo de esta memoria se expondrá la arquitectura desarrollada para la realización del proyecto. Se profundizará especialmente en los componentes necesarios para el despliegue y escalado automático de una aplicación web.

Se explicará también la arquitectura interna del proveedor *cloud* que se va a utilizar. Para contextualizar todo ello, en primer lugar se analizan una serie de conceptos básicos necesarios para la comprensión de todo el desarrollo.

## 3.1. Introducción a la solución propuesta

La solución propuesta para la escalabilidad automática de aplicaciones web en un *cloud* PaaS se va a basar en el trabajo desarrollado en el proyecto *4CaaSt* [7]. Principalmente como proveedor IaaS se tomará Flexiscale, (ver sección 2.3). El PaaS, asentado sobre el IaaS, se construirá principalmente mediante los componentes *Paas Manager* y *Gestión de instalación de software*. El trabajo realizado en este PFC, se basa en realizar funcionalidad avanzada IaaS sobre Flexicale y parte del *Paas Manager*, centrándose en la parte de escalabilidad. Finalmente para probar la escalabilidad PaaS se utilizará una aplicación multi-capa, como se verá en el capítulo *Implementación y despliegue*.

## 3.2. Conceptos básicos

Para poder comprender la arquitectura lógica, que se ha empleado para el desarrollo de los componentes que permiten desplegar y gestionar aplicaciones web en el *cloud*, es necesario tener claros algunos conceptos.

Se puede definir un ***entorno*** como el conjunto de recursos de hardware virtual y software necesarios para el correcto funcionamiento de un servicio, que en este caso será una aplicación web.

La aplicación deberá estar provista de todas las especificaciones necesarias para el correcto funcionamiento de la misma en la nube. En la figura 3.1 se muestra la arquitectura lógica utilizada para el despliegue de aplicaciones web.

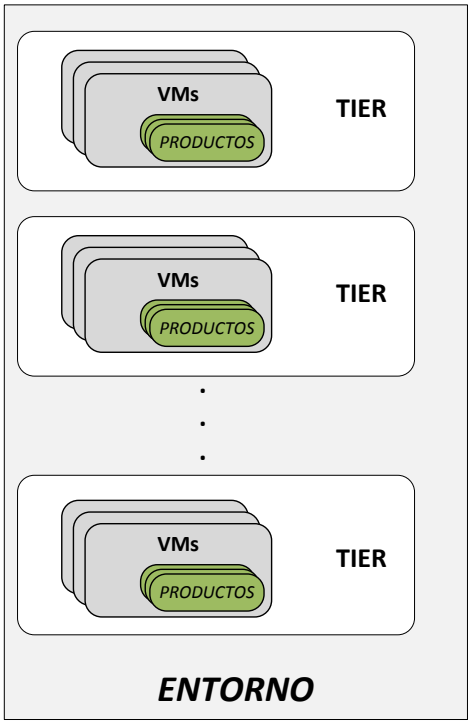



**Figura 3.1.** Arquitectura lógica

El ***entorno*** está constituido por un conjunto de ***tiers*** con diferente funcionalidad y relacionados entre sí. El conjunto permite alojar un **servicio** con la funcionalidad especificada por el usuario. El *entorno* está compuesto por un conjunto de VMs y está provisto de software adecuado a las especificaciones del servicio. Las características de hardware de una máquina virtual vendrán determinadas por el tipo de software que soporten o las necesidades de almacenamiento de las que se quiera proveerla. Dentro del *entorno* se aloja la aplicación web.

En función del tipo de funcionalidad de las VMs, éstas se agrupan en ***tiers***. Cada tier estará conformado por una o varias máquinas. El conjunto de VMs encargadas del almacenamiento de datos es un ejemplo de *tier*.

El software necesario en cada una de las VMs que pertene a un *tier* se denomina **producto.** Siguiendo con el ejemplo anterior, una máquina virtual necesitará estar provista de una base de datos si se va a encargar de almacenarlos.

Una vez claros estos términos, en la siguiente sección se analizará la arquitectura del proveedor *cloud* que se va a utilizar, y cómo se relaciona los conceptos descritos.

## 3.3. Arquitectura del proveedor en el *cloud*: Flexiscale

El desarrollo de un software para gestión en el *cloud* no puede llevarse a cabo sin conocer cómo funcionan los proveedores que se van a utilizar. Cada proveedor ofrece unas características diferentes que han de ser analizadas para poder desplegar servicios auto-escalables.

Para la realización de este proyecto final de carrera se utiliza el proveedor Flexiant, en concreto, su producto Flexiscale (ver sección 2.3). La página de inicio que nos ofrece este producto está en la figura 3.2. En ella aparecen todas las máquinas desplegadas por el usuario, indicando su estado, la imagen utilizada y las características de hardware virtual de cada una de ellas.

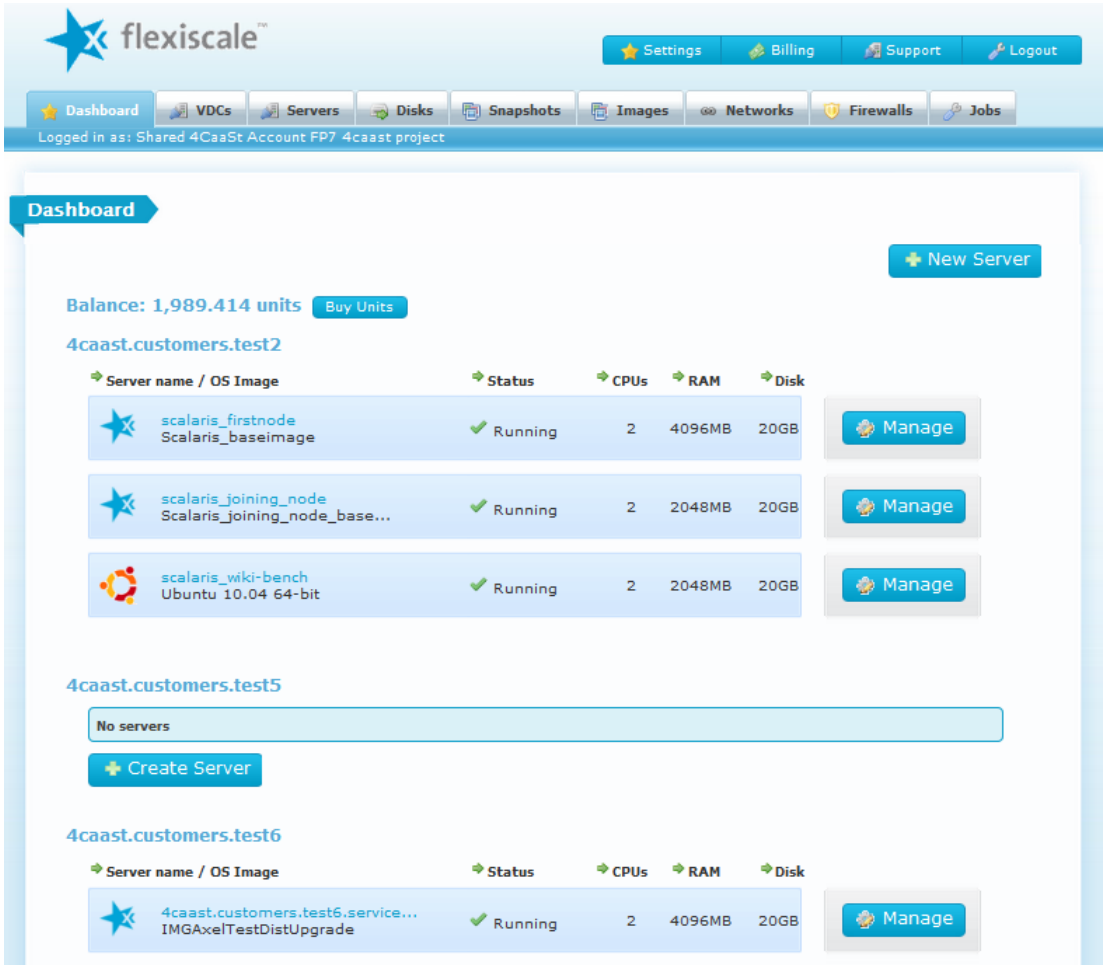



**Figura 3.2.** Página de inicio de un usuario de *Flexiscale* [4].

Es posible conocer todas las características de una máquina, como se especifica en la figura 3.3. A parte de las características que se podían observar en la pantalla inicial (figura 3.2), permite conocer los datos de red y acceso. Las direcciones IP que proporciona Flexiscale pueden ser públicas o privadas, quedando a elección del usuario.


**Figura 3.3.** Especificaciones de una VM en *Flexiscale* [4]

Un concepto muy importante a tener en cuenta es el *Virtual Data Center* (VDC). Los VDC se utilizan para clasificar las VMs en función de las necesidades del usuario. En el mismo VDC estarán, por ejemplo, todas las máquinas que se hayan desplegado en un entorno, ya que al pertenecer al mismo deben están relacionadas entre sí. En la figura 3.4 se muestran los VDC del usuario. Se indica para cada uno de ellos: el número de máquinas desplegadas, el número de imágenes y discos que se han creado en ese VDC, y el número de redes que lo conforman.

Se denominará **organización** al conjunto de recursos que utiliza un grupo de trabajo. Una organización estará formada por uno o más VDCs. Este concepto se tiene en cuenta a la hora de establecer el nombre de un VDC en Flexiscale, estando éste compuesto por el nombre de la organización y el propio del VDC [19].

Las VMs provistas de direcciones IP privadas, tendrán visibilidad únicamente dentro del VDC al que pertenezcan. Las redes privadas existentes en un VDC no son compartidas con otros centros de datos virtuales. Las máquinas con direcciones IP públicas son visibles desde cualquier punto de la red, desde su propio VDC y desde otros.

Flexiscale permite crear imágenes a partir de VMs desplegadas. Para ello se realiza un clonado del disco de una máquina, y se almacena. El proveedor Flexiant utiliza el hipervisor *Xen*, que no permite crear imágenes en caliente de una VM, es decir, es necesario que la máquina esté apagada. En la figura 3.5 se muestran todas las imágenes disponibles para un usuario, tanto las que son provistas por Flexiscale, como las creadas por el propio usuario. Las imágenes creadas por el usuario pueden ser borradas, mientras que las propias del proveedor no. Además, es posible modificar el nombre y los *Metadatos* de las imágenes del usuario, así como el nombre de usuario por defecto,


**Figura 3.4.** Especificaciones de un *VDC* en *Flexiscale* [4].

como se muestra en la figura 3.6.

Las características de hardware de una máquina virtual han de ser especificadas antes de la creación de la misma. No es posible modificarlas a posteriori. Es por esto que el tipo de escalabilidad que se va a utilizar para este proyecto es horizontal (ver sección 2.4).

A parte de la interacción web con el usuario, Flexiscale está provista de una API que utiliza la tecnología *Simple Object Access Protocol* (SOAP).

## 3.4. Arquitectura del Software

La puesta en marcha de una aplicación web auto-escalable en el *cloud* requiere un desarrollo de software complejo. La implementación de una estructura modular permite adecuarse de forma más intuitiva a los distintos requerimientos de los usuarios.

Cada uno de los componentes utiliza distintas tecnologías para ofrecer funcionalidades diferenciadas. La combinación de todos ellos da lugar a un software muy completo que permite nuestro objetivo final: escalar una aplicación web.

El software desarrollado se compone de cuatro componentes fundamentales, como se observa en la figura 3.7. El componente *Paas Manager* interactúa con los otros tres, ya que se encarga de orquestar el conjunto del software. *Claudia* se encargará de provisionar la infraestructura física del *entorno* y del análisis de las métricas obtenidas de los *Sistemas de Monitorización* para la escalabilidad. Los *Sistemas de Monitorización*


**Figura 3.5.** Imágenes de disco disponibles para un usuario en *Flexiscale* [4].

almacenan los datos obtenidos de las máquinas virtuales que indica el *Paas Manager* que son necesarios, y se los facilitan a *Claudia*. Los sistemas de *Gestión de instalación de software* se encargan de proveer correctamente el software que necesita cada una de las máquinas virtuales previa petición del *Paas Manager*.

A continuación, se explica la funcionalidad de cada uno de los componentes de forma más detallada. Vamos a distinguir los componentes en función del modelo de servicio que se vaya a utilizar. El componente *Paas Manager* se estudiará en más profundidad, dada su envergadura y relevancia para la solución propuesta.

### 3.4.1. Claudia

El primer componente que se va a analizar el *Claudia*. *Claudia* es un gestor de servicios, que se encarga de gestionar los entornos desplegados y de su escalabilidad. Además permite interactuar con los proveedores *cloud*. Provee a un servicio de toda la infraestructura hardware necesaria para su funcionamiento, es decir, proporciona todos los recursos necesarios a nivel IaaS.

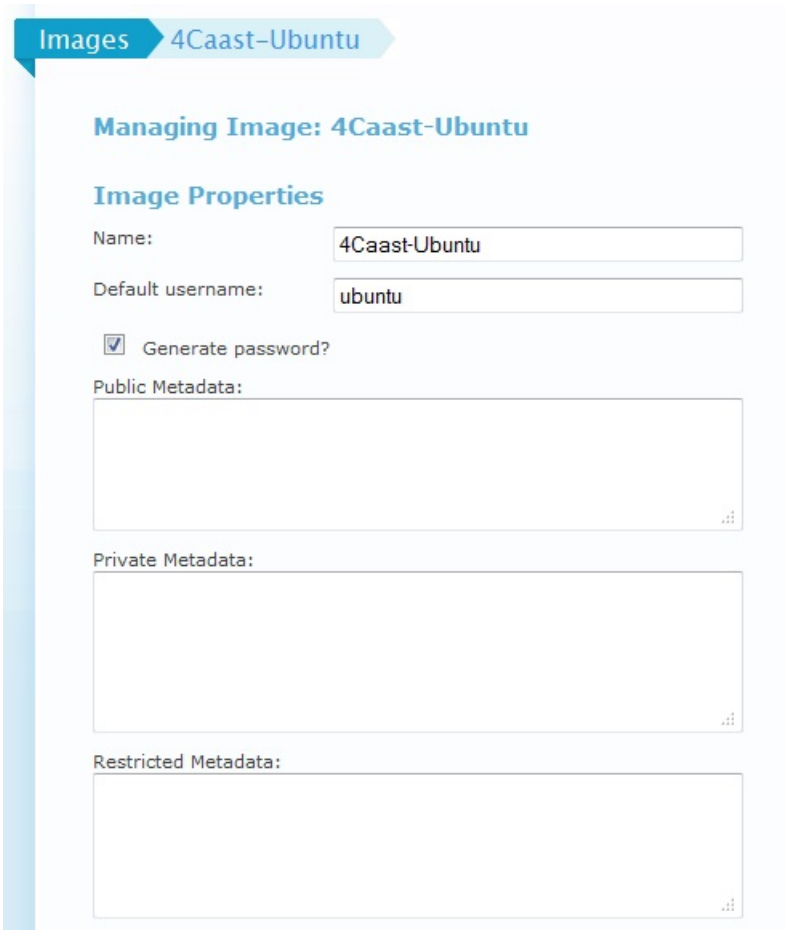



**Figura 3.6.** Configuración de una imagen creada por el usuario en *Flexiscale* [4].

La transferencia de datos e información con el *Paas Manager* se realiza a través de un *API REST* desarrollado específicamente para ellos.

Se comunica con los *Sistemas de Monitorización* gracias a una *API TCloud* [19]. La **API TCloud** es un *API REST* orientada que utiliza representaciones XML para el intercambio de mensajes. Permite la estandarización de las comunicaciones dentro del *cloud*.

Al encargarse de la provisión de los recursos hardware en la nube, interactúa con los proveedores *cloud* a través de *Drivers*. Es necesario desarrollar un *driver* por cada uno de los proveedores con los que se vaya a interactuar.

En el *driver* se especifican las acciones concretas a realizar con el proveedor elegido por el usuario. Las operaciones para los distintos proveedores *cloud* serán las mismas, y en función del *driver* que se utilice, se realizarán las acciones concretas necesarias. El *driver* para Flexiscale está integrado en *Claudia* junto con los correspondientes a otros

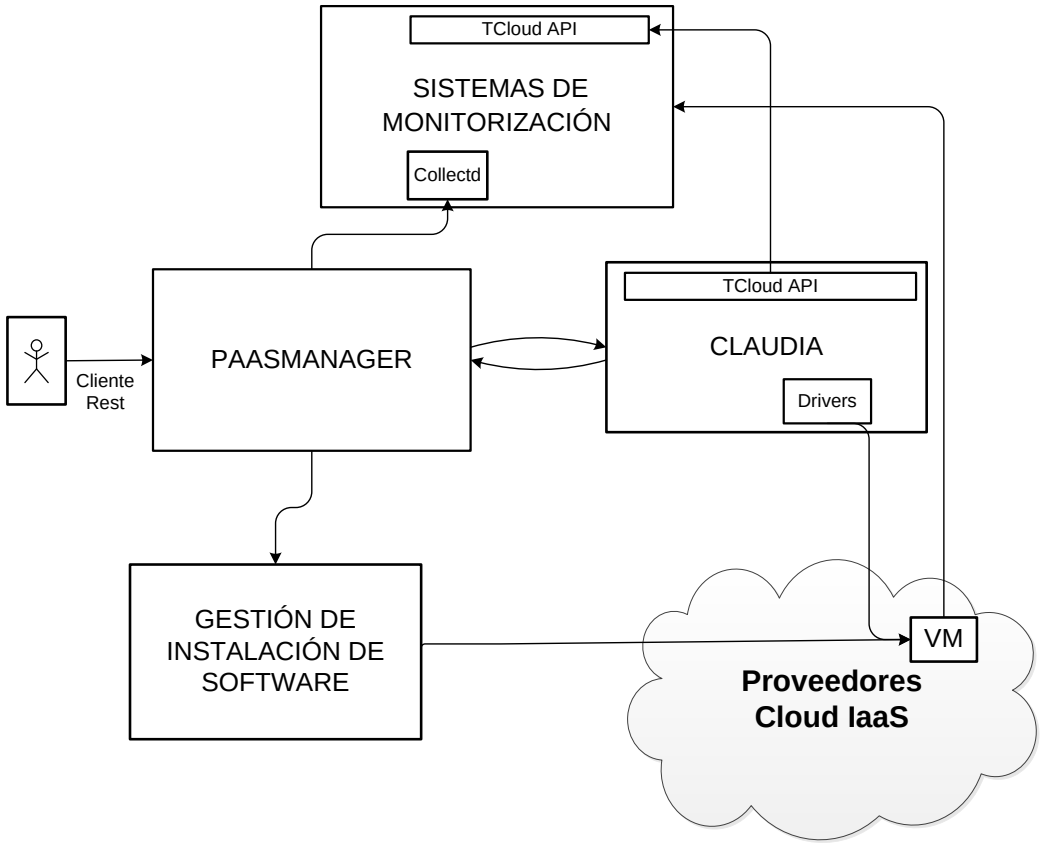



**Figura 3.7.** Componentes de la Arquitectura Software

proveedores.

A través de los *drivers* de *Claudia* es posible crear, borrar, apagar y encender máquinas virtuales en los proveedores *cloud*. Para el provisionamiento de las máquinas, se especifica con todo detalle las características de hardware necesarias como tamaño de la RAM o número de CPUs, el nombre que debe tener y la imagen base que se ha de utilizar para crear la máquina. Además *Claudia* puede acceder a los metadatos de cada una de las VMs, modificarlos, crear nuevos metadatos o borrar los que ya existen.

La comunicación con los *Sistemas de Monitorización* permite a *Claudia* obtener todos los datos de las métricas de las máquinas virtuales, que permiten realizar acciones de escalabilidad. La escalabilidad se realiza teniendo en cuenta umbrales establecidos para las métricas obtenidas, con máximos y mínimos. Estos parámetros se almacenan en reglas específicas para cada uno de los *tiers* que sean escalables dentro de un *entorno*.

El análisis de las métricas obtenidas y su comparación con las reglas se realiza a través de un motor de reglas, que permite realizar acciones si se producen violaciones de los umbrales. Las acciones asociadas a las reglas están previamente definidas y se refieren a la escalabilidad. A través de las acciones, las reglas con los umbrales de las

métricas y los datos de los KPIs es posible llevar a cabo la escalabilidad automática del *entorno*. Las acciones establecidas permiten realizar peticiones al *Paas Manager*, que se encargará de gestionar la escalabilidad.

### 3.4.2. Gestión de instalación de software

El componente de gestión de instalación de software se encarga del provisionamiento de software en máquinas virtuales, además de su gestión como actualizaciones, configuración y desinstalación. Está basado en un motor de configuración llamado *Chef* que ofrece una solución distribuida dividida entre un servidor central llamado *servidor Chef* y agentes en los nodos *clientes Chef*.

Las instrucciones de instalación y configuración se establecen en *récipes* o recetas. Las recetas son scripts de instalación de software que se almacenan en libros de recetas, o *CookBooks*, en función de las características del software. Es necesario que la máquina virtual con la cual se interactúe, tenga instalado un cliente *Chef* previamente.

A través de las recetas se pueden realizar diversas acciones sobre el software instalado en las VMs, desinstalarlo o reconfigurar parámetros. También se encarga de arrancar o parar el software instalado.

Las especificaciones de las operaciones a realizar y los datos de las máquinas involucradas son enviadas desde el *Paas Manager*. Es el usuario quien establece las especificaciones y se las indica al *Paas Manager* en una petición vía *API REST*.

### 3.4.3. Sistemas de Monitorización

Los sistemas de monitorización se encargan de recoger y almacenar en base de datos información referente a las métricas de monitorización tanto de las VMs como del software que se encuentre instalado dentro de las mismas. Dichos datos son recogidos a través del software de monitorización *Collectd*. Es necesario que un demonio de *Collectd* esté corriendo en la máquina virtual a monitorizar.

Estos datos son recogidos por *Claudia* para el análisis de los mismos, permitiendo la escalabilidad automática. Estas métricas son ofrecidas por los sistemas de monitorización mediante un API TCloud, que también utiliza *Claudia*.

La monitorización es un pilar necesario para la escalabilidad automática. Por tanto, este componente es necesario siempre que se quiera que el entorno sea auto-escalable, tanto en modelos de servicio IaaS como PaaS.

### 3.4.4. Paas Manager

El *Paas Manager* es el componente que se encarga de la gestión de los *entornos*, tanto en el despliegue como posteriormente su escalabilidad. Conoce todo el hardware virtual y el software comprendido en cada uno de los *entornos* desplegados, así como los

requisitos iniciales que establece el usuario.

La petición de creación de *entornos* es recibida por el *Paas Manager* a través de un *API REST*. Se encarga de la gestión del despliegue de dicho *entorno* ateniéndose a los requisitos del usuario.

Interactúa con todos los componentes que conforman el software. Se encarga de registrar cada una de las máquinas virtuales de un *entorno* en los *Sistemas de Monitorización* para que los datos de las métricas sean almacenados, para la posterior gestión de la escalabilidad automática. Las nuevas máquinas creadas para escalar también son registradas utilizando este mismo sistema.

Para la instalación y gestión del software, es el *Paas Manager* quien se encarga de realizar las peticiones a los *sistemas de Gestión de instalación de software*, de nuevo a través de un *API REST*. La comunicación se producirá siempre que sea necesaria la instalación de software, arrancarlo, pararlo y configurarlo en el despliegue, y posteriormente para la reconfiguración del *entorno* cuando sea escalado.

Para el despliegue y la gestión de la infraestructura física, el *Paas Manager* realiza peticiones a *Claudia* se encargará de proporcionar todos los recursos a nivel IaaS necesarios en el *entorno*. *Claudia*, al encargarse de la gestión de los datos de monitorización, ver sección 3.4.1, realizará las peticiones necesarias al *Paas Manager* para que se encargue de gestionar la escalabilidad para el correcto funcionamiento del *entorno*.

Toda la información acerca del *entorno* es almacenada de forma persistente en base de datos por el *Paas Manager*, donde se actualizan los cambios que se van produciendo para adecuar el servicio a la demanda.

En la figura 3.9 se muestran más detalladamente las distintas funciones que acaban de analizarse y las interacciones con los componentes junto con los distintos módulos que forman el *Paas Manager*. A continuación se analizará el modelo de datos utilizado para la gestión de *entornos* y la estructura del componente.

#### 3.4.4.1. Modelo de datos del *Paas Manager*

El *Paas Manager* tiene almacenados en su base de datos todos los recursos que puede manejar en un *entorno* cualquiera. Por ello, podemos distinguir entre dos módulos principales de datos: los recursos instanciados o provisionados y los recursos catalogados. Los **recursos catalogados** contienen todas las especificaciones necesarias que pueden ser gestionadas por el componente, y los **recursos instanciados** corresponden con los que actualmente están desplegados en el proveedor *cloud*.

El modelo de datos general que utiliza el Paas Manager corresponde con el analizado en la sección 3.2. Habrá *entornos*, *tiers*, *productos* y *aplicaciones*. Existirá, por tanto, una correspondencia entre los recursos instanciados y los recursos catalogados, para ca-

da uno de los tipos de datos del modelo. La relación entre ellos se observa en la tabla 3.1.

| **Recurso Catalogado** | **Recurso Instanciado** | **Características** |
|---|---|---|
| *Environment* | *EnvironmentInstance* | Un ***Environment*** es una representación completa del software instalado, incluyendo todos los *tiers* necesarios para una aplicación. El ***EnvironmentInstance*** corresponderá con el *entorno* desplegado en el proveedor. Por ejemplo: LAMP (Linux, Apache, MySQL, Php). |
| *Tier* | *TierInstance* | Un ***Tier*** es la especificación de los *productos* instalados en una VM, y los datos referentes a la escalabilidad de la capa. Un ***TierInstance*** corresponde a un *tier* que se encuentra desplegado en la nube dentro de un *entorno*. Por ejemplo: Tomcat. |
| *ProductRelease* | *ProductInstance* | El ***ProductRelease*** especifica el software instalable, que puede estar instalado antes del despliegue en la imagen o tener que ser instalado a posteriori. Un ***ProductInstance*** es el software que se encuentra instalado dentro de una máquina virtual. Por ejemplo: MySQL |
| *ApplicationRelease* | *ApplicationInstance* | En la ***ApplicationRelease*** se realiza la especificación de una aplicación donde se establecen los atributos de configuración y los artefactos. Una ***ApplicationInstance*** corresponde a la aplicación desplegada en el *cloud*. Por ejemplo: Mediawiki |

**Tabla 3.1.** Correspondencia entre Recursos Catalogados e Instanciados

En la figura 3.8 se muestra el diagrama de clases que implementa el *Paas Manager*. Quedan representados todos los recursos utilizados, tanto instanciados como catalogados. Las relaciones que se establecen entre ellos permiten la especificación de *entornos* de forma detallada.

Un *entorno* estará compuesto por una serie de *tiers* y cada uno de ellos tiene especificados o instalados unos *productos*. Además, se especificarán las *aplicaciones* que conforman un entorno. Las relaciones son simétricas para recursos catalogados e instan-

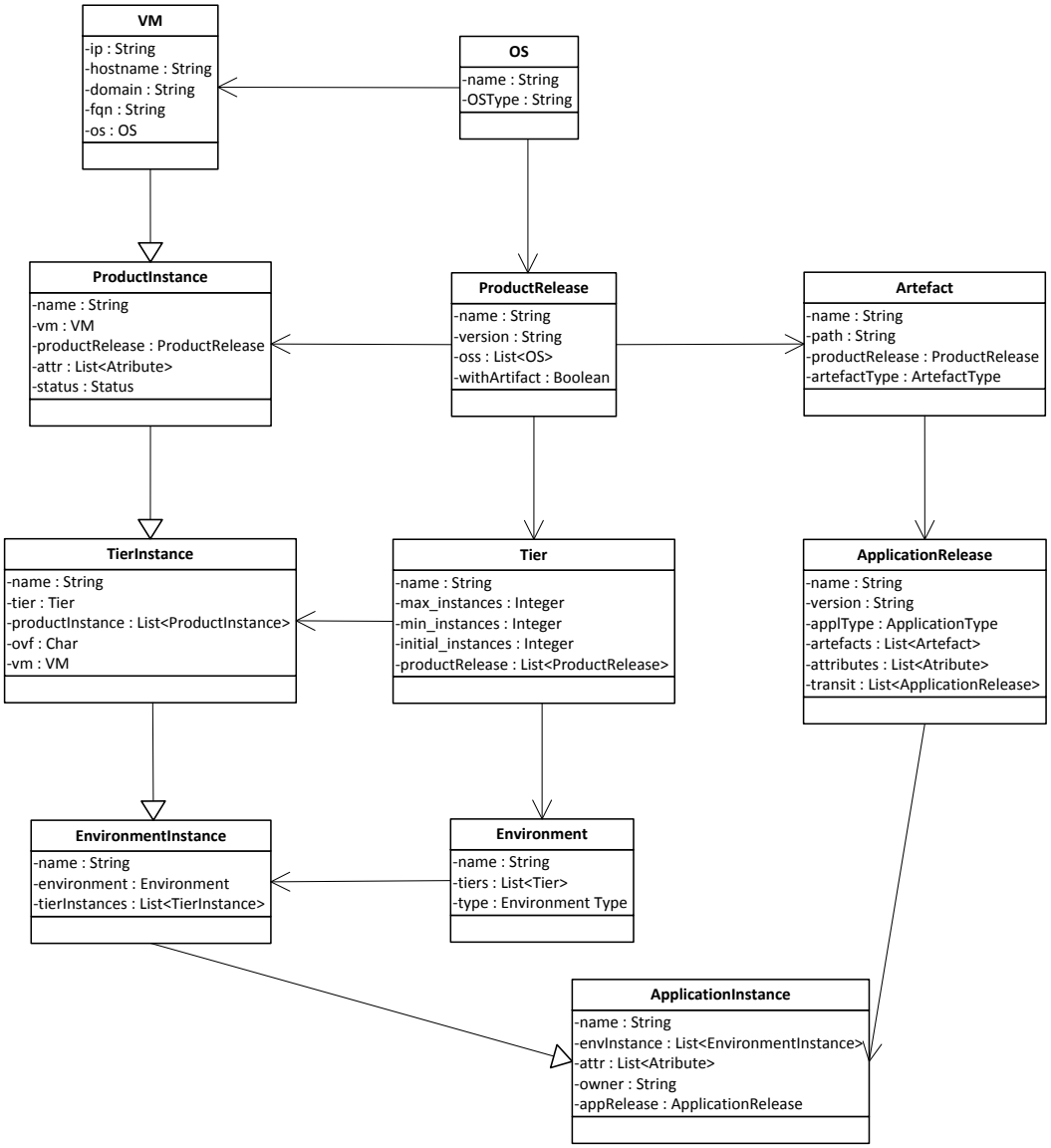



**Figura 3.8.** Diagrama de clases del *Paas Manager*

ciados. Cabe destacar la importancia del orden de preferencia: no puede existir un *tier* si no está en un *entorno*, y así sucesivamente.

#### 3.4.4.2. Módulos

El *Paas Manager* está formado por tres bloques principales y una base de datos asociada en la que se persisten los datos relacionados con los recursos catalogados e instanciados. En la figura 3.9 se muestran más detalladamente composición del *Paas Manager*, y el tipo de operaciones que realizan los distintos módulos.

Las acciones relacionadas con los recursos catalogados se ajustan solamente a

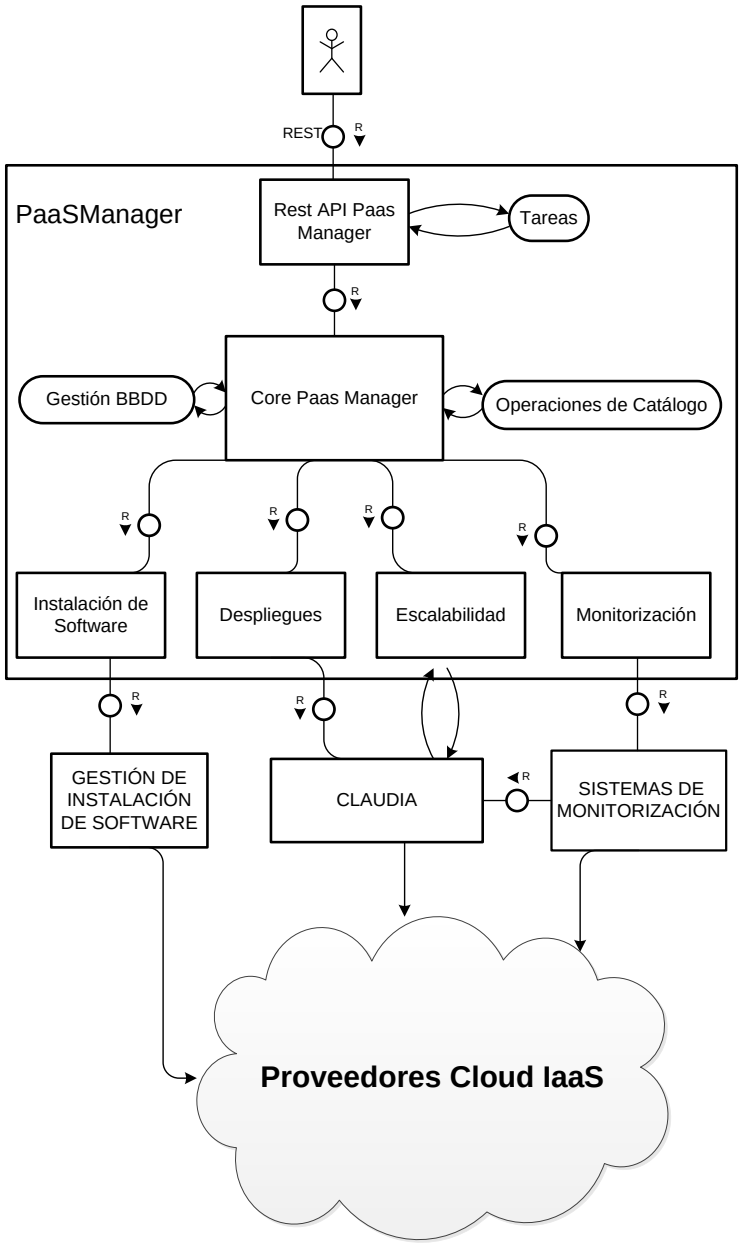



**Figura 3.9.** Arquitectura y operaciones del *Paas Manager*

operaciones con la base de datos. Sin embargo, para los recursos instanciados, además de acciones de persistencia de datos, se realizan las operaciones necesarias para el despliegue y la gestión de los mismos.

Dentro del *Paas Manager*, se distinguen tres bloques diferenciados, en función de la funcionalidad que proveen al componente:

- **Model Paas Manager:** En este bloque se establece el modelo de datos que se va a utilizar, tanto para los recursos instanciados como catalogados. Además se

establece el modelo para el almacenamiento en base de datos.

- **Rest API Paas Manager:** En este bloque se definen las operaciones a realizar tras las peticiones vía *RestFul*. También se convierten las especificaciones recibidas acerca de los entornos a desplegar o escalar a los datos del modelo.

- **Core Paas Manager:** Está compuesto por varios componentes que se encargan, cada uno de ellos, del manejo de los distintos recursos instanciados y catalogados. En el *Core Paas Manager*, se desarrollan las acciones a realizar para llevar a cabo la gestión del software que ha de ser instalado, despliegue de entornos, escalabilidad y monitorización. Además, se encarga de la gestión de la base de datos: consultas, insercciones, modificaciones, etc.

En la figura 3.9 no queda reflejado el bloque *Model Paas Manager*, ya que se encuentra implícito en todo el componente.

# Capítulo 4
# Implementación y despliegue

En este capítulo se recoge el trabajo realizado en este PFC para la escalabilidad de aplicaciones web a nivel PaaS. Para ello, primero se describirán las características principales que debe tener una aplicación para pode ser escalable en la nube. Teniendo en cuenta esto, posteriormente se definirá la plataforma para generar el *entorno* sobre el que se desplegará la aplicación. Finalmente se estudiarán los pasos realizados para conseguir la escalabilidad del *entorno* PaaS, para ofrecer a los usuarios finales el servicio con la calidad requerida, amoldándose a la demanda del mismo.

## 4.1. Características fundamentales de las aplicaciones desplegadas en el *cloud*

Las **aplicaciones web** son software complejo que se despliegan en *entornos* web, que son un conjunto de recursos de hardware virtual y software. El *entorno* estará compuesto de varias capas o *tiers*. Son necesarios, además de la infraestructura hardware proporcionada por un *cloud* IaaS, otros recursos adicionales como puede ser la aplicación web que se quiere desplegar. Además, es necesario conocer los datos que nos proporcionen información sobre el rendimiento de la aplicación. Esto permite ofrecer un servicio de calidad al usuario, gestionando la auto-escalabilidad.

La figura 4.1 corresponde al *entorno* desplegado para una aplicación web. Dentro del *entorno* existen tres *tiers*: el *tier* del balanceador para la distribución de la carga entre los diferentes servidores web, el *tier* de los servidores web con las aplicaciones que alojan, y el *tier* correspondiente a las bases de datos que se encargan de la persistencia de datos de la aplicación. Por tanto, el conjunto de los tres *tiers* desplegados, y relacionados entre sí, forman el *entorno* necesario para alojar una aplicación web en el *cloud* y gestionarla.

Una aplicación web está alojada en un entorno. Cuando se habla de escalar una aplicación web quiere decir escalar el *entorno* sobre el que se despliega la aplicación web. Para que la aplicación web sea escalable, el *entorno* debe tener cinco características fundamentales.

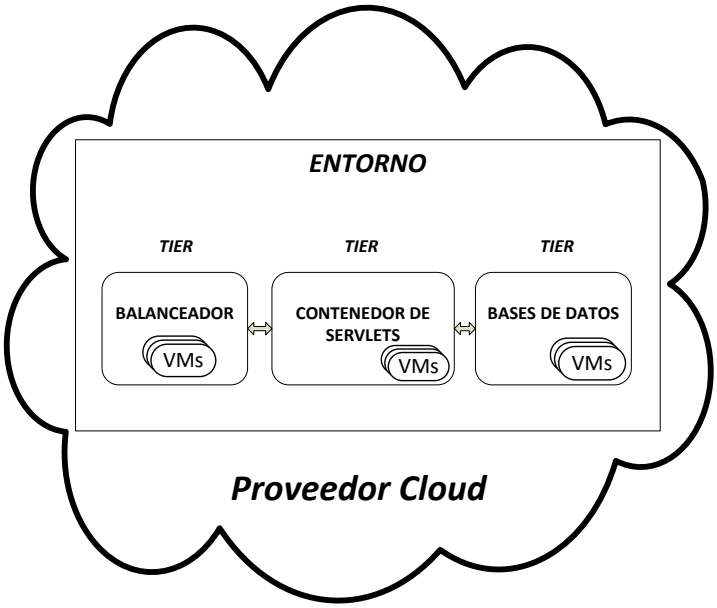



**Figura 4.1.** Arquitectura en el *cloud* de una aplicación web

**Gestión de varios tiers.** La aplicación se implementa en distintos *tiers*. Cada uno de ellos está compuesto por un tipo particular de VM. Los distintos componentes desarrollados nos permiten el despliegue de distintas máquinas dentro de un *entorno*, y la configuración dinámica de los *tiers*, obteniendo una estructura compleja del *entorno*.

**Tiers sin estado.** Para escalar de forma sencilla, añadiendo o eliminando máquinas virtuales, es necesario que los *tiers* a escalar no tengan estado, es decir, no tengan datos almacenados. Separando el *entorno* en *tiers* con y sin estado, podemos escalar un servicio sin que se pierda ningún tipo de información necesaria para el correcto funcionamiento del *entorno*. Es decir, para escalar un *tier* sin estado, creamos y borramos réplicas idénticas, teniendo varias máquinas virtuales clonadas entre sí. Para una aplicación web esto implica que la capa de base de datos no pueda ser balanceada, ya que contiene datos de la aplicación y que la única capa a escalar sea la que contiene la aplicación web.

**Balanceadores.** Un balanceador nos permite repartir la carga entre los distintos servidores web que forman los *tier* sin estado, en nuestro ejemplo, el servidor web, equilibrando la carga entre ellos. Para distribuir la carga, es necesario utilizar algoritmos de balanceo, más simples o más complejos, en función de las necesidades del *entorno* o de la aplicación.

**Persistencia de Datos de la aplicación.** Al escalar se crean y destruyen máquinas virtuales. Para evitar la pérdida de datos, y el acceso a los mismos desde cualquier réplica de un *tier*, es necesario almacenarlos de manera persistente. Para ello, se utiliza un *tier* diferente para esta persistencia, al que se puede acceder desde los *tiers* del *entorno* que estén configurados para ello.

**Métricas de escalado.** La escalabilidad se centra en los indicadores de la demanda del servicio o calidad de servicio, como son el tiempo de respuesta o el número de usuarios. Esto permite conocer la sobre-carga del *entorno* o la infra-utilización de los recursos, clave para la escalabilidad de un sistema.

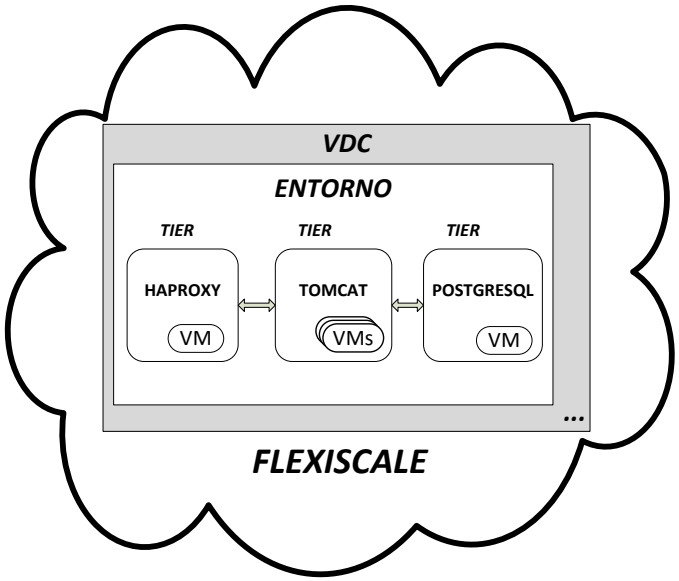



**Figura 4.2.** *Entorno* de la aplicación web desplegada en *Flexiscale*

El *entorno* necesario para alojar la aplicación web que se va a desplegar se muestra en la figura 4.2. El proveedor *cloud* a utilizar será Flexiscale. Todas las máquinas virtuales desplegadas en un *entorno* estarán en el mismo VDC.

La aplicación web estará desplegada en el ***tier* tomcat**. Este *tier* es el único de los tres *tiers* del entorno que no tiene estado, ya que los datos serán almacenados en el ***tier* postgreSQL**, que es donde se alojan las bases de datos necesarias. El *tier* tomcat, por tanto será escalable, y la carga se repartirá entre las distintas VMs desplegadas bajo demanda, por el ***tier* haproxy**. El *tier* haproxy está formado por un balanceador que tiene configurado un algoritmo de reparto de carga *Round Robin*, es decir, la carga se repartirá de manera equitativa entre todas las VMs que esté balanceado. Al almacenar información sobre cada una de las máquinas que forman el *tier* tomcat, también tiene estado, y por tanto, no será escalable.

Cada uno de los tres *tiers* será monitorizado, obteniendo las métricas de los KPIs necesarios para gestionar la escalabilidad.

## 4.2. Definición del *entorno*

Una vez que tenemos la arquitectura del *entorno* de la aplicación web, el siguiente paso es definir el *entorno* que se quiere desplegar en la nube de forma entendible por los

componentes *cloud*, para comenzar a operar con ella.

### 4.2.1. Definición de un *entorno* no escalable a nivel IaaS

Un *entorno* puede ser definido de cero, o utilizar especificaciones ya existentes para hacerlo. Para este proyecto queremos definir un *entorno* propio desde el inicio. Las características que queremos otorgar a un *entorno* son especificadas en la petición de despliegue a través del estándar *Open Virtualization Format* (OVF).

El estandar OVF utiliza el formato XML. A continuación se explican las secciones más importantes que forman el OVF y que son necesarias para la creación de un *entorno*.

**Definición de los *tiers*.** En primer lugar es necesario definir los *tiers* en los que se quiere dividir el *entorno*. Cada *tier* estará asociado a una imagen para las máquinas que lo formen, y será la misma para todas. Para ello se utiliza la sección *References*.

```
<ovf:References>
        <ovf:File ovf:id="haproxy"
                ovf:href="4CaastHaproxyNoSetup"
                rsrvr:digest="8f1643c4fdf83ab3827190ab771f76e1"/>
        <ovf:File ovf:id="tomcat"
                ovf:href="4Caast-Ubuntu-NoSetup"
                rsrvr:digest="8f1643c4fdf83ab3827190ab771f76e1" />
        <ovf:File ovf:id="posgresql"
                ovf:href="4CaastChefClientCollectdScriptUbuntu"
                rsrvr:digest="8f1643c4fdf83ab3827190ab771f76e1"/>
</ovf:References>
```

Se indica en el cuadro, por tanto, que existirán tres *tiers*: haproxy, tomcat y postgresql, donde las VMs que conformen cada uno de ellos, utilizarán las imágenes definidas.

**Especificación de red** Es necesario establecer la red a la que va a pertenecer el *entono* que se quiere desplegar. Se especifica en la sección *NetworkSection*.

```
<ovf:NetworkSection>
        <ovf:Network ovf:name="public" rsrvr:public="true">
        </ovf:Network>
</ovf:NetworkSection>
```

Por defecto, la red que se va a utilizar siempre es la pública. Esto permitirá que las VMs sean visibles desde fuera del VDC al que pertenecen.

**Especificación de las máquinas virtuales de un *tier*** En la sección *VirtualSystemCollection*, se especifican las características del *entorno* general a desplegar. Estará compuesta por tantas secciones *VirtualSystem* como *tiers* existan. Dentro de las especificaciones de cada *tier* aparecen otras secciones:

```
<ovf:VirtualSystemCollection ovf:id="demoPFC">
        <ovf:VirtualSystem ovf:id="haproxy" rsrvr:min="1"
                rsrvr:max="1"   rsrvr:initial="1"
                rsrvr:balancer="true"   rsrvr:lbport="8088">
                <ovf:OperatingSystemSection ovf:id="76">
                        <ovf:Description>
                                4CaastHaproxyNoSetup
                        </ovf:Description>
                </ovf:OperatingSystemSection>
        </ovf:VirtualSystem>
</ovf:VirtualSystemCollection>
```

En el *VirtualSystemCollection* se establece el nombre del *entorno* que se va a desplegar. Para este ejemplo, el *VirtualSystem* especificado es para el *tier* haproxy. Además se restringe el número máximo, mínimo e inicial de VMs que conformarán el *tier*. Es muy importante indicar estos campos ya que, en función de ellos, la aplicación será escalable o no. Si el número máximo y mínimo de VMs en el *tier* es el mismo, no será escalable.

Además, al tratarse del *tier* encargado de balancear la carga, ha de indicarse que es un balanceador, y el número del puerto en el que se encuentra.

En el *OperatingSystemSection* se especifica de nuevo la imagen que va a servir de base en ese *tier*, así como el sistema operativo de la misma. El sitema operativo se establece de forma numérica, siguiendo las reglas establecidas por Flexiscale.

Dentro de la definición del *VirtualSystem*, aun no está definida la infraestructura física necesaria. Por ello añadimos una nueva sección en su interior:

```
<ovf:VirtualHardwareSection>
        <Info>
                Virtual Hardware Requirements: 512Mb,1CPU,1disk
        /Info>
        <Item>
                <rasd:Description>
                        Number of virtual CPUs
                </rasd:Description>
                <rasd:ElementName>1 virtual CPU</rasd:ElementName>
                <rasd:InstanceID>1</rasd:InstanceID>
                <rasd:ResourceType>3</rasd:ResourceType>
                <rasd:VirtualQuantity>1</rasd:VirtualQuantity>
        </Item>
        ...
        ...
</ovf:VirtualHardwareSection>
```

En la sección *VirtualHardwareSection* de nuestro OVF, se especifican el número de CPUs, el tamaño de la RAM y la red a la que pertenece. Estará definido un *Item* por

cada uno de los recursos hardware.

Con estas secciones queda definido un *entorno* a nivel IaaS. Se desplegarán tantas VMs como hayan especificado para componer cada uno de los *tiers*. Las imágenes utilizadas deberán estar provistas del software que sea requerido en cada *tier*. También deberán tener instalada nuestra aplicación web.

Cabe destacar la importancia de establecer un orden en el despliegue de los *tiers*. Se debe tener en cuenta la relación que existe en entre el software de los distintos *tiers*. Para el ejemplo de la figura 4.2, en primer lugar debemos desplegar el *tier* que contiene el almacenamiento persistente de datos mediante PostgreSQL, ya que los servidores web Tomcat necesitan conocer las direcciones IP de las máquinas virtuales para poder acceder a las bases de datos. El *tier* del bananceador Haproxy sería el siguiente en crearse, ya que las máquinas que conforman el *tier* del servidor web han de registrarse en el mismo para poder equilibrar la carga. Finalmente se desplegarán las VM que conforman el *tier* del servidor web. El orden de despliegue de cada uno de los *tiers* se establece en la sección *References*, explicada previamente.

### 4.2.2. Definición de un *entorno* no escalable a nivel PaaS

El despliegue de un *entorno* a nivel IaaS obliga a utilizar imágenes existentes en el proveedor. Esto limita las características que puede tener, ya que el software para cada *tier* ha de elegirse entre un conjunto de imágenes con software pre-instalado y pre-configurado.

Sin embargo, la solución propuesta en este proyecto permite instalar en cada *tier* el software requerido. No es necesario utilizar imágenes que te proporcionen combinaciones de software, sino que es el usuario quien elige.

Los *tiers* de un *entorno* pueden ser una combinación de ambas soluciones. Si existe una imagen en el proveedor que se ajusta a los requerimientos de un *tier* concreto, será utilizada. Habrá para otros *tiers* donde no exista la imagen adecuada, por lo que se partirá de una imagen base para desplegar las VMs, y posteriormente serán configuradas.

El software que nos encontramos dentro de un *entorno* puede ser clasificado como *Product Instance Component* (PIC), o *Application Component* (AC).

Un **PIC** es el software necesario para el correcto desempeño de las funciones de un *tier*. Por ejemplo, un Haproxy, en el *tier* del balanceador, o un tomcat o postgreSQL.

Un **AC** es el software de aplicación que se despliega en un *tier*, y que necesita de un PIC para su funcionamiento. Por ejemplo, en el *tier* tomcat del ejemplo, tenemos una aplicación web, que sería un AC, y que para su desempeño necesita tener instalado Tomcat, un PIC.

Para poder realizar la configuración del software, es necesario añadir en el OVF una nueva sección. Será diferente para cada *tier*, por lo que se especifica dentro de la sección *VirtualSystem*.

```
<ovfenvelope:ProductSection>
        <ovfenvelope:Info>tomcat_PIC</ovfenvelope:Info>
        <ovfenvelope:Product>tomcat_PIC</ovfenvelope:Product>
        <ovfenvelope:Version>0.0.3</ovfenvelope:Version>
        <ovfenvelope:Property ovfenvelope:value="tomcat_PIC"
                ovfenvelope:key="fourcaast.instancecomponent.id"/>
        <ovfenvelope:Property .../>
                ...
</ovfenvelope:ProductSection>
```

Dentro del *ProductSection* se especifica el producto, la versión y una serie de propiedades referentes al *CookBook* que debe utilizar el componente para la *Gestión de instalación de software* así como las recetas concretas. Habrá recetas para la instalación del software, y también para arrancarlo o pararlo. Las distintas recetas necesarias para un mismo producto se especifican en el mismo *ProductSection*. Para cada producto o software necesario para alojar la aplicación se utilizará un *ProductSection* distinto.

Para instalar la aplicación web en el *tier*, es necesario un nuevo *ProductSection* en el OVF, que corresponda a un AC. En las propiedades se especificará la ubicación de la aplicación a instalar, que puede estar en el componente para la *Gestión de instalación de software* o en algún repositorio accesible para este. Otras características necesarias para el funcionamiento de la aplicación también se añaden como propiedades. Al igual que para los PIC, se debe indicar si se quiere arrancar o no la aplicación una vez instalada.

### 4.2.3. Definición de un *entorno* escalable

El *entorno* a desplegar quedaría definido con las dos secciones anteriores. Sin embargo, la idea de desplegar la aplicación web en el *cloud* es que los recursos utilizados vayan adaptándose a la demanda.

Para ello es necesario que el *entorno* sea escalable, y se añadan o quiten máquinas a los *tiers* de acuerdo a los valores de las métricas obtenidas.

Para que el *entorno* sea escalable debe definirse en el OVF. En primer lugar, los *tiers* sin estado que se quieran escalar deben tener indicado un número máximo de VMs, que debe ser mayor que su número mínimo.

```
<ovf:VirtualSystemCollection ovf:id="demoPFC">
        <ovf:VirtualSystem ovf:id="tomcat" rsrvr:min="1"
rsrvr:max="5" rsrvr:initial="1" rsrvr:balanced="haproxy">
</ovf:VirtualSystemCollection>
```

Para la aplicación web que queremos desplegar, el *tier* escalable será tomcat. Indicamos que en el despliegue del *entorno* estará compuesto por una única máquina virtual. Sin embargo, si es necesario, se desplegarán nuevas máquinas virtuales hasta un máximo de 5. En ningún caso el *tier* no tendrá ninguna VM. La carga será balanceada entre todas las máquinas por el *tier* haproxy.

El *entorno*, por tanto, será escalable. No obstante, se debe indicar cuándo se deben borrar o crear nuevas VMs en el *tier*. Para establecer las reglas de escalado se crea otra sección dentro del *VirtualSystem*. En este caso, para definirlas, no se utilizará el estándar OVF, sino que se empotrará una sección definida en *Rule Interchange Format* (RIF). En el despliegue de la aplicación, *Claudia* realizará la traducción de las reglas RIF a un lenguaje que comprenda el motor de reglas, quedando almacenadas en un fichero. Un ejemplo muy sintetizado de una regla definida en el OVF se muestra en el siguiente cuadro.

```
<rsrvr:GovernanceRuleSection>
<rif:sentence>
<rif:if>
 ...
 <rif:formula>
  ...
  <rif:Const type="xsd:string">requestDelay</rif:Const>
   ...
 </rif:formula>
 ...
 <rif:formula>
  <rif:op>
   <rif:Const type="rif:iri">
    http://www.w3.org/2007/rif...predicate#numeric-greater-than
   </rif:Const>
  </rif:op>
  ...
  <rif:op>
   <rif:Const type="rif:iri">
    http://www.w3.org/2007/rif...predicate#numeric-divide
   </rif:Const>
   ...
   <rif:args ordered="yes">
    <rif:Var>?kpiValue</rif:Var>
    <rif:Var>?actualCount</rif:Var>
   </rif:args>
   ...
  </rif:op>
  ...
  <rif:Const type="xsd:integer">20</rif:Const>
   ...
 </rif:formula>
</rif:if>
```

```
<rif:then><rif:Do>
 <rif:op>
  <rif:Const type="rif:iri">
   http://claudia.telefonica.com#actions.scalePaaSUp
  </rif:Const>
 </rif:op>
 ....
 <rif:Var>?vee</rif:Var>
</rif:Do></rif:then>
</rif:sentence>
</rsrvr:GovernanceRuleSection>
```

En esta regla se establece que la métrica que se va a utilizar para escalar el *entorno* es el tiempo de respuesta. Esta métrica no se refiere a los recursos hardware, como debería ocurrir en un *entorno* desplegado a nivel IaaS, sino que se utiliza un indicador del servicio ofrecido por la aplicación web.

Según la regla del ejemplo, se realizará la acción *scalePaaSUp* definida en *Claudia*, siempre que el tiempo de respuesta de una VM del *tier* sea superior a 20. Al utilizar el algoritmo de balanceo *Round Robin*, conociendo el tiempo de respuesta de una VM, conocemos el de todas las que forman el *tier*, ya que la carga se distribuye uniformemente entre ellas.

## 4.3. Imágenes creadas para la monitorización de VMs

Al definir un *entorno* escalable es necesario conocer los KPIs de los *tier* para poder escalarlos. Los datos de las métricas han de estar disponibles en la VM para que sean almacenadas por los sistemas de monitorización.

Cada imagen base que se va a utilizar en cada *tier* tendrá instalado el demonio *Collectd*, que utiliza sondas para recoger los datos de la máquina.

La configuración de las sondas de recogida de métricas del *Collectd* se realiza desde la propia VM. El valor de los KPI ha de ser enviado a los *Sistemas de monitorización* para su almacenado. No es suficiente con la instalación de dichas sondas, ya que si la VM no es caracterizada de alguna manera, no se podrá distinguir la procedencia de los datos. Los *Sistemas de monitorización* almacenan los datos recibidos por muchas VMs, y cada una de ellas debe tener un identificador único que la distinga del resto.

La solución que se propone para solventar la identificación de cada VM es crear un *script* de configuración en las imágenes. Dicho script realiza todas las acciones necesarias para poder llevar a cabo la correcta monitorización.

- Es necesario identificar la máquina, es decir, cada VM debe tener un nombre único que permita ser reconocida desde el exterior y por ella misma. Como identificador único se utiliza el *Fully Qualified Name* (FQN), que es introducido por *Claudia* en el despliegue de las máquinas.

- Para ello, se obtienen los metadatos de la VM a través de una petición HTTP especificada en la API de Flexiscale. El XML de respuesta se parsea obteniendo el FQN. Dicho FQN se almacenará en un fichero, que tendrá la misma ruta en cualquier máquina virtual desplegada desde cualquier imagen.

- Una vez obtenido el FQN, se añade el *md5* de ese valor a un fichero de configuración del *Collectd*, y que utilizarán las sondas para enviar los datos de las métricas asociadas a cada VM. El *md5* del FQN será, por tanto, el identificador unívoco que permite que los sistemas de monitorización conozcan la procedencia de los KPIs.

- Toda esta configuración se realiza con el *Collectd* parado, por lo que es necesario arrancarlo, quedando configurado para el envío de datos. Una vez configurado comienzan a enviarse la métricas desde la VM.

En Flexiscale, cuando se crea una VM a partir de una imagen, la contraseña de dicha VM será la misma que tuviera la VM de la que se creó la imagen. El acceso a ella, se complica bastante, teniendo que buscar la VM *«de la que hereda»* para obtenerla. Para solucionarlo, se borra un fichero de configuración en el que se establece dicha contraseña por defecto, por lo que el *password* pasaría a ser el propio de la máquina. Esta operación también se realiza dentro del *script*.

Los *tiers* que requieren de la instalación de software por parte del componente de *Gestión de instalación de software* deben estar provistos de un cliente *Chef*. A nivel PaaS se utilizan recetas *Chef*, lo que implica que la máquina sobre la que se actúe debe estar preparado para ello. Las imágenes base utilizadas para estos *tiers*, además de estar configuradas para la monitorización, tendrán el cliente *Chef* instalado.

## 4.4. Despliegue del *entorno*

Una vez realizadas todas las especificaciones necesarias en el OVF, se realiza la petición al *Paas Manager* para el despliegue del *entorno* a través de un cliente REST. El proceso de despliegue del *entorno* tiene una duración variable, en función del número de *tiers*, de máquinas virtuales que componen cada uno de ellos y de los requerimientos de software a instalar en el caso de despliegue PaaS. Por ello, esta petición se procesa de forma asíncrona por el *Paas Manager*, creando una tarea con un identificador conocido por el usuario, que permite su seguimiento.

El *Paas Manager* será el encargado de interactuar con el resto de componentes; por ello se va a analizar más detenidamente cada una de las acciones que se realizan.

Las distintas operaciones realizadas para crear el *entorno* escalable en el *cloud* PaaS se muestran en la figura 4.3. Sólo se ha analizado la petición realizada por el cliente. A lo largo de esta sección se analizarán el resto de operaciones que ha de realizar el *Paas Manager* para cumplir con las especificaciones del usuario.

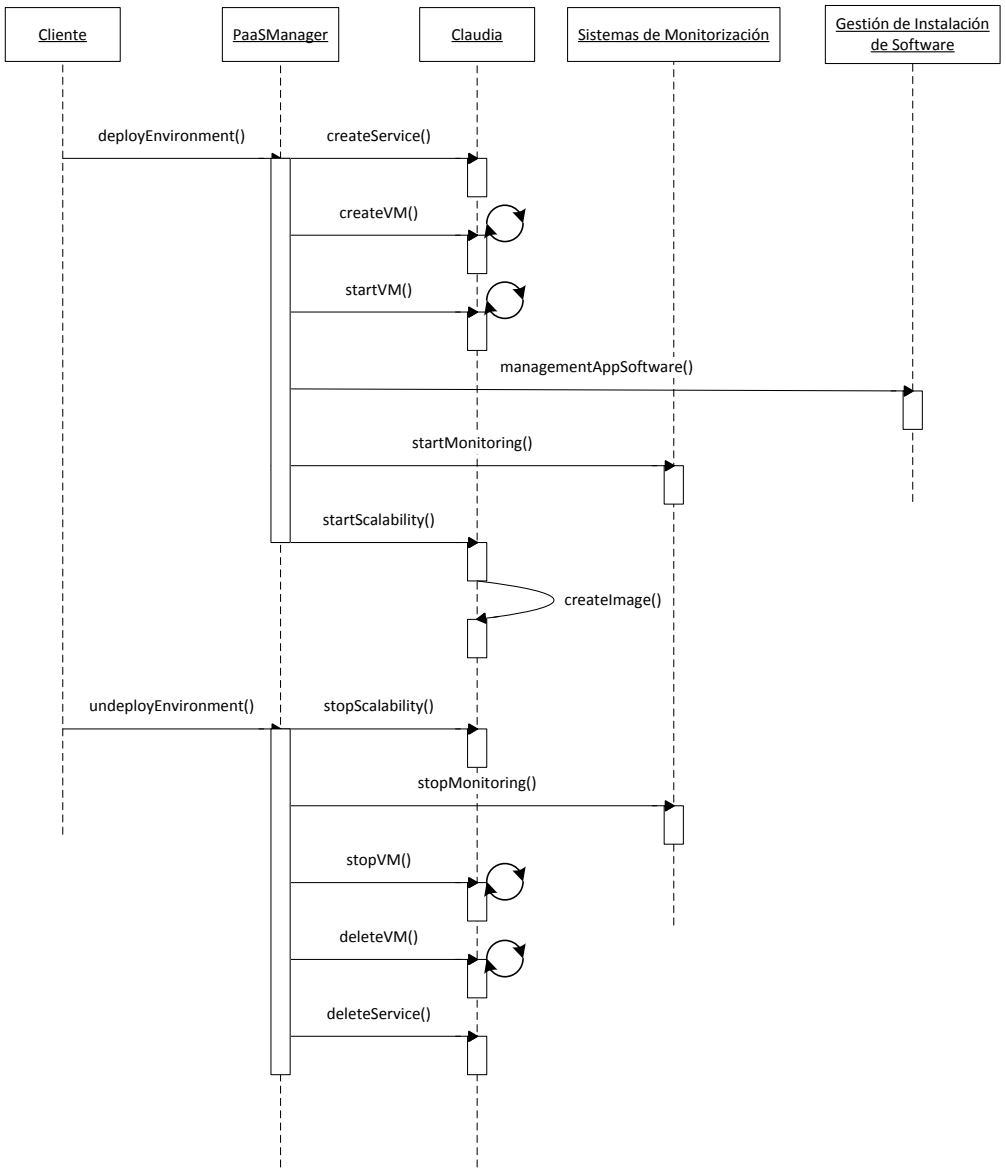



**Figura 4.3.** Diagrama de secuencia para el despliegue de un *entorno* escalable a nivel PaaS

### 4.4.1. Interacción con Claudia

*Claudia* se encarga de proveer al entorno de toda la infraestructura física necesaria para alojar una aplicación web en la nube.

#### 4.4.1.1. Despliegue

Una vez realizada la petición de despliegue vía Restfull, el *Paas Manager* debe gestionarlo. *Claudia* es el componente encargado de interactuar con Flexiscale a través

de su *driver*. La comunicación del *Paas Manager* con *Claudia* se realiza también con Rest.

En la *URL* de la petición realizada por el cliente, se especifica el VDC en el cual se va a desplegar la aplicación. El *Paas Manager* indica a *Claudia* que si no existe ese VDC, lo cree a través de la petición *startService()* (figura 4.3). *Claudia* almacena en su base de datos todas las especificaciones del *entorno* que son enviadas desde el *Paas Manager* con el formato OVF.

Los parámetros de despliegue de un *tier* que indican el número máximo, mínimo e inicial en el OVF se almacenan de forma persistente en el *Paas Manager*. Será éste quien se encargue de modificarlos en el OVF que se envía a *Claudia* para el despliegue de la infraestructura física, cambiando cada uno de los tres valores por 1, de forma que *Claudia* no conozca si el entorno es o no escalable.

Completada esta acción, el *Paas Manager* indica a *Claudia* que se deben desplegar las máquinas iniciales que conforman cada *tier* y y arrancarlas (ver figura 4.3).

*Claudia* realiza estas tareas de forma asíncrona, por lo que el *Paas Manager* se quedará esperando a que finalice para realizar la siguiente petición.

La escalabilidad puede llevarse a cabo si cada una de las VMs desplegadas en un *tier* tiene un identificador único dentro de Flexiscale. Con las VMs creadas, antes de ser arrancadas, *Claudia* introduce el metadato FQN que crea a partir del VCD, el nombre del *entorno*, el nombre del *tier*, y el número de réplica. Cada una de las máquinas de un *tier* tendrá un número de réplica que estará definido entre 1 y el número máximo de nodos que puedan formar el *tier*.

Para poder introducir el FQN, es necesario desarrollar en el *driver* de Flexiscale una API correspondiente a los metadatos. También será utilizada para obtener la dirección IP de cada máquina virtual.

Desplegadas las VMs, el *Paas Manager* recopila la información sobre la dirección IP y el FQN de cada una de ellas. Con estos datos será posible la gestión del *entorno*.

Las reglas RIF establecidas en el OVF para el *tier* tomcat son traducidas por *Claudia* y almacenadas. El nombre de ese fichero corresponde con el FQN de la máquina del *tier* cuyo número de réplica es 1.

Instalado el software necesario para los *tiers* desplegados a nivel PaaS, el *Paas Manager* realiza una nueva petición a *Claudia* para que genere una imagen de una máquina que se encuentre dentro de cada uno de los *tiers* de estas características (ver figura 4.3). La VM es parada para crear la imagen, por lo que después debe ser arrancada de nuevo. La imagen se creará siempre que el *tier* sea escalable. Si no lo es, no tiene sentido crear una imagen que nunca se va a utilizar.

La creación de imágenes se realiza para simplificar posteriormente la escalabilidad. El coste temporal de la instalación de software puede hacer que nuestra aplicación web no proporcione la calidad de servicio deseada cada vez que se escala.

El *Paas Manager* almacena de forma persistente todas las características del *entorno* y de las VMs que se encuentran desplegadas. También guarda en base de datos los OVF correspondientes al despliegue de cada uno de los *tier*. Si se ha creado una imagen de una VM de un *tier*, en el OVF se sustituirá el nombre de la imagen dada por el cliente inicialmente, por la que se acaba de crear.

Finalmente, en un *entorno* escalable, el *Paas Manager* indica a *Claudia* que comience a recoger de los sistemas de monitorización las métricas de los KPIs establecidos en las reglas.

#### 4.4.1.2. Retirada del entorno

Para borrar un *entorno* desplegado, se realizarán las operaciones inversas a las que se realizan en el despliegue (ver figura 4.3).

A petición del *Paas Manager*, *Claudia* apagará todas las VMs desplegadas en cada uno de los *tiers* para su posterior eliminación. Además borrará de su base de datos la información relacionada con el *entorno*.

### 4.4.2. Interacción con los sistemas de Gestión de Instalación de Software

Tras el despliegue de todas las máquinas virtuales, el *Paas Manager* comprueba si existe algún *tier* en el que esté especificado el software a instalar. Para cada una de las VM de los *tier* que cumplan ese requisito, se realiza una petición a los sistemas de instalación de software. Realmente, se realiza una petición por cada *ProductSection* especificado en el OVF inicial por el cliente. En cada petición se especifican las propiedades dadas en esa sección así como la dirección IP de la VM en la que se va a instalar el software.

El componente para la *Gestión de instalación de software* se encarga de la instalación y configuración del software en la VM. Es necesario que la VM esté arrancada. También se procede a la instalación de la aplicación web.

Al existir un *tier* que está balanceado en el *entorno*, el *Paas Manager* también se encarga de realizar una petición al componente para la *Gestión de instalación de software* para que configure el balanceador. Se añadirán las direcciones IP de las VMs que vayan a ser balanceadas, para poder repartir la carga entre ellas.

La interacción se lleva a cabo solamente en el despliegue de un *entorno*, ya que al borrar el *entorno* todas las VMs serán borradas, y no es necesaria la desinstalación previa del software y las aplicaciones instaladas.

### 4.4.3. Interacción con los sistemas de Monitorización

En un entorno escalable, es un requisito fundamental y necesario, monitorizar las máquinas virtuales y el software que corre sobre ellas.

#### 4.4.3.1. Despliegue

Tras la creación y arranque de cada una de las VMs que conforman los *tiers*, el *Paas Manager* realiza peticiones a los *Sistemas de Monitorización* indicando que se debe comenzar a recoger datos de cada una de ellas. Los *Sistemas de Monitorización* necesitan conocer el FQN de las máquinas para almacenarlas correctamente. Las VM quedan registradas en la base de datos de los sistemas de monitorización.

En petición realizada para el registro de una VM por el *Paas Manager* se indica el *md5* del FQN. Las imágenes base configuradas previamente, ya están enviando datos a los *Sistemas de Monitorización* (ver sección 4.3). No obstante estos lo están descartando, ya que no se ha registrado la máquina previamente. Tras el registro, los KPIs que reciban los *Sistemas de Monitorización* que correspondan a una VM registrada, quedarán almacenados.

#### 4.4.3.2. Retirada del entorno

Un *entorno* que va a ser borrado no necesita estar monitorizado. Por tanto, el *Paas Manager* realiza una petición a los *Sistemas de Monitorización* indicando que deje de esperar métricas de cada una de las VMs que se habían registrado.

No es posible des-registrar una VM. Por tanto, si en un futuro se va a desplegar el mismo *entorno* de nuevo, el *Paas Manager* tiene en cuenta esto, y no realiza la petición de registro de nuevo. Tampoco es posible que automáticamente se borren las métricas monitorizadas por un entono.

Para la escalabilidad, los valores de los KPIs que se tienen en cuenta son los últimos recibidos, por lo que los datos anteriores almacenados para una VM no interferirían en el correcto funcionamiento del nuevo *entorno* auto-escalable.

## 4.5. Escalabilidad de un *entorno* desplegado

Alojar una aplicación en el *cloud* permite ofrecer una calidad lineal en el tiempo, adaptando los recursos. El *entorno* sobre el que se despliega la aplicación web está preparado para amoldarse a la demanda.

Mientras el *entorno* esté desplegado en el *cloud*, *Claudia* tiene un hilo en ejecución que recoge los datos de los KPIs de monitorización.

Los umbrales de las métricas establecidos por el usuario, se almacenan en *Claudia* (ver sección 4.4.1.1). La violación de los umbrales lleva asociado consigo una acción. Para la optimización de los recursos se llevarán a cabo dos acciones relacionadas con la escalabilidad. Si se supera un umbral máximo, se creará una nueva réplica o VM dentro

del *tier* escalable. Por el contrario, si el valor de un KPI está por debajo del umbral mínimo, se borrará una réplica. El KPI que se utilizará es el tiempo de respuesta de la aplicación web.

El usuario ha establecido en el OVF para la creación del *entorno* un número máximo y mínimo de VMs que pueden formar un *tier*. El número de réplicas creadas o borradas estará limitado por las especificaciones.

En *entorno* de la figura 4.4, se utilizará como apoyo para la correcta explicación de la escalabilidad. Este *entorno* está compuesto por tres *tiers*. Cada uno de los *tiers* está formado por una única VM. El *tier* haproxy se encarga de balancear la carga. El *tier* tomcat aloja la apliciación web. Los datos de la aplicación web se alamacenarán en el *tier* postgreSQL.

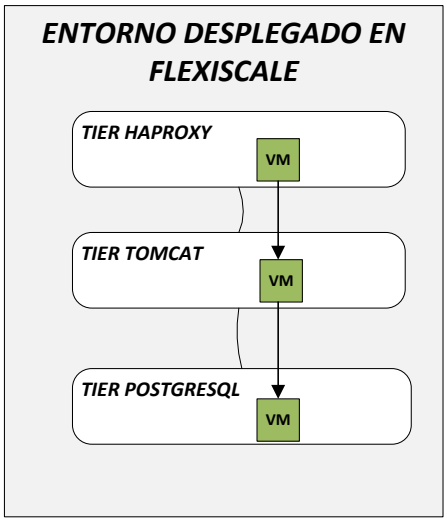



**Figura 4.4.** *Entorno* desplegado en el *cloud* para una aplicación web

El *Paas Manager* se encargará de gestionar la escalabilidad. En este caso será *Claudia* quien realice inicialmente las peticiones de crear o borrar una nueva VM. El diagrama de secuencia de las peticiones realizadas entre los componentes se encuentra en la figura 4.5.

### 4.5.1. Escalabilidad: aumento de los recursos de un *tier*

En la figura 4.4 tenemos en *entorno* desplegado donde se aloja la aplicación web en el *cloud*. *Claudia* recibe constantemente los eventos de monitorización de cada una de las VMs.

El *tier* tomcat es escalable en este *entorno*. *Claudia* analiza constantemente los eventos de monitorización de las VMs y los compara con las reglas definidas con los

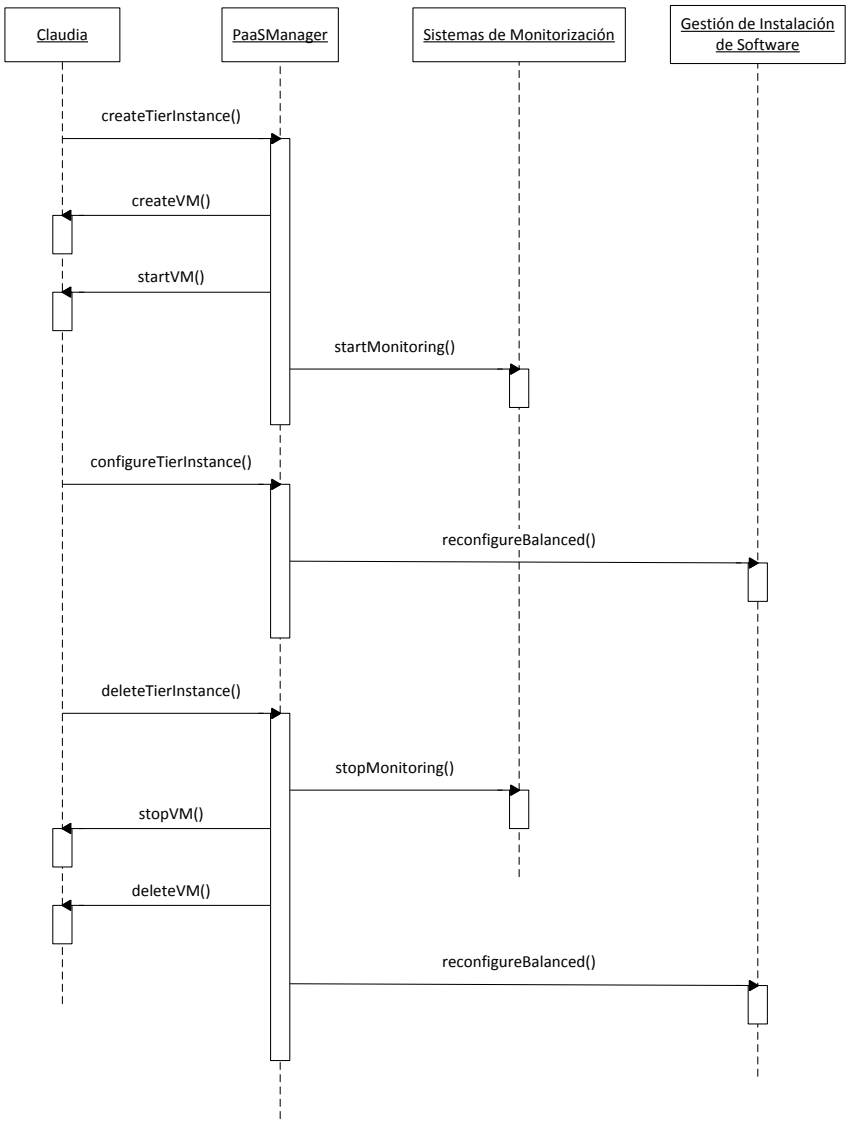



**Figura 4.5.** Diagrama de secuencia para la escalabilidad de un *entorno* a nivel PaaS

umbrales. Cuando un KPI supera el umbral, *Claudia* realiza la acción establecida en las reglas. La realización de la acción lleva consigo envíar al *Paas Manager* la instrucción de que escale el *tier* tomcat (ver figura 4.5).

Cuando el *Paas Manager* recibe esta petición, comprueba mediante los datos almacenados del *tier*, si es posible crear una nueva VM, teniendo en cuenta el número máximo de instancias en ese *tier*, y el número actual de réplicas desplegadas. La petición se procesa como una tarea asíncrona ya que, aunque no suponga tanto tiempo como el

despliegue de un entorno, sí tarda lo suficiente como para bloquear otras peticiones que puedan llegar al componente.

Al no ser una acción inmediata, *Claudia* seguirá recibiendo eventos de monitorización que superen el umbral, y realizando la acción de pedir escalar el *tier*. El *Paas Manager* se encarga de descartar las peticiones que llegan si la acción ya está en proceso de realización.

Realizadas todas las comprobaciones oportunas, el *Paas Manager*, realiza a *Claudia* las peticiones de crear una nueva VM y arrancarla, al igual que ocurre en el despliegue del *entorno* (ver figura 4.5). En la petición se indica el número de réplica para que *Claudia* pueda generar correctamente el FQN de la máquina.

En el OVF enviado para la creación de la VM ya aparece la imagen creada en el despliegue como imagen base, ya que el *tier* tomcat ha sido creado instalando la aplicación a posteriori, gracias al componente para la *Gestión de instalación de software*.

La máquina es creada y se procede a su registro en los sistemas de monitorización (ver sección 4.4.3.1). El *Paas Manager*, almacena los datos de la nueva máquina virtual y los cambios asociados producidos en el *entorno* (ver figura 4.5).

Una vez el *tier* tomcat ha sido escalado, *Claudia* comprueba si dicho *tier* es balanceado. El *tier* que se encarga de balancear la carga debe conocer la dirección de la nueva VM creada, para distribuir la carga. Se realiza, por tanto, una segunda acción asociada a esa regla. El *Paas Manager* recibe la petición de reconfigurar el balanceador. Es el componente para la *Gestión de instalación de software* el que se encarga de añadir el nuevo nodo o réplica al balanceador con la información recibida desde el *Paas Manager* (ver figura 4.5).

Completada la acción de reconfigurado, el *entorno* queda perfectamente escalado. El *entorno* resultante sería el que se muestra en la figura 4.6.

### 4.5.2. Escalabilidad: disminución de los recursos de un *tier*

La disminución de la demanda en la aplicación web provoca que el *entorno* donde se encuentra alojada esté sobre-aprovisionado. Siguiendo con el ejemplo utilizado en esta sección, el *entorno* está configurado como se indica en la figura 4.6.

Los valores de los KPIs recibidos por *Claudia* comienzan a disminuir de forma cualitativa, llegando al umbral mínimo establecido en las reglas por el usuario. Si el valor del tiempo de respuesta es demasiado pequeño es porque tendremos exceso de recursos en el *entorno*.

Al superar el umbral mínimo, el *Paas Manager* recibe la petición del escalar el *tier*

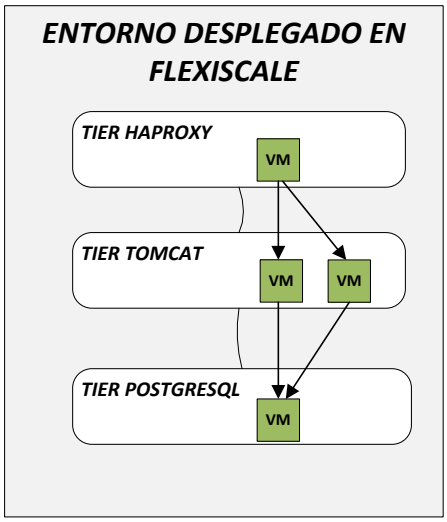



**Figura 4.6.** *Entorno* escalado en el *cloud* para una aplicación web

tomcat, disminuyendo una VM. Al igual que para el escalado incremental, el *Paas Manager* comprueba que el número de réplicas desplegadas en el *tier* sea mayor que el número mínimo establecido, ya que en otro caso, no se podrá borrar. Igualmente tiene en cuenta que la operación no esté en proceso.

*Claudia* recibirá la petición de parar la VM que se quiere borrar (ver figura 4.5). El *Paas Manager* indicará el número de réplica que debe desaparecer. Utiliza el criterio de borrar la VM de un *tier*, cuyo número de réplica sea mayor. Una vez parada la VM, ésta es borrada por *Claudia*.

Los *Sistemas de Monitorización* reciben la petición de dejar de almacenar los datos correspondientes a la VM que se ha borrado (ver sección 4.4.3.1).

Se realizan las comprobaciones en *Claudia* para saber si el *tier* que se está escalando está siendo balanceado. El *tier* balanceador debe conocer que ha desaparecido la VM, ya que toda la carga que se redireccione a máquinas que no existen, no podrá obtener respuesta, y nuestra calidad del servicio disminuirá.

El *Paas Manager* recibe la petición de reconfigurar el balanceador, para que elimine la VM del *tier* tomcat. Al igual que para la escalabilidad hacia arriba, es el componente para la *Gestión de instalación de software* el que se encarga de borrar la dirección IP en el balanceador de la VM que se ha borrado del *entorno* (ver figura 4.5).

En la figura 4.4 queda representado el *entorno* tras la escalabilidad hacia abajo realizada, teniendo en cuenta que el *entorno* antes de este escalado era el mostrado en la figura 4.6.

# Capítulo 5
# Metodología y tecnologías utilizadas

En este quinto capítulo se describe tanto la metodología de trabajo realizada a lo largo del desarrollo del proyecto así como unas nociones básicas de las tecnologías implicadas y utilizadas.

## 5.1. Metodología de trabajo

Para el desarrollo de este Proyecto Final de Carrera, se han utilizado dos metodologías ágiles, *SCRUM* asociada a la gestión de proyectos, y *Programación extrema* relacionada con buenas prácticas de desarrollo de software. A lo largo de esta sección se realizará una retrospectiva, explicando en qué consisten y las herramientas que se han utilizado para su aplicación.

### 5.1.1. Metodología Ágil

El desarrollo ágil de software son métodos de ingeniería del software basados en el desarrollo iterativo e incremental. Las necesidades y soluciones evolucionan gracias a la autoorganización de equipos multifuncionales. Fomenta la comunicación continua entre los propios miembros del equipo y con los clientes. Promueve la planificación adaptativa, el desarrollo y entrega de software funcional, en cortos periodos de tiempo que permiten una respuesta rápida y flexible a los cambios. Alienta a mantener una constante atención a la excelencia técnica y al buen diseño [5].

#### 5.1.1.1. SCRUM

Para el desarrollo de este proyecto se ha utilizado SCRUM como metodología de gestión ágil. Su simplicidad se basa en la adaptación continua a las circunstancias de la evolución del proyecto.

La idea del SCRUM es dividir el proyecto en varios *sprints*, que son periodos de tiempo de entre dos y cuatro semanas. Los objetivos del proyecto se definen en el *product backlog*, al inicio del mismo. Todo *sprint* comienza con una reunión de *sprint planning* para definir el *sprint backlog*, priorizan las tareas a llevar a cabo durante este tiempo.

Tras la finalización de un *sprint*, se realiza una reunión llamada *demo meeting*, para que los desarrolladores muestren al resto del equipo las tareas terminadas, realizando demostraciones del software desarrollado. En la figura 5.1 se muestra de forma esquemática el proceso a seguir en la metodología SCRUM que acaba de ser explicado.

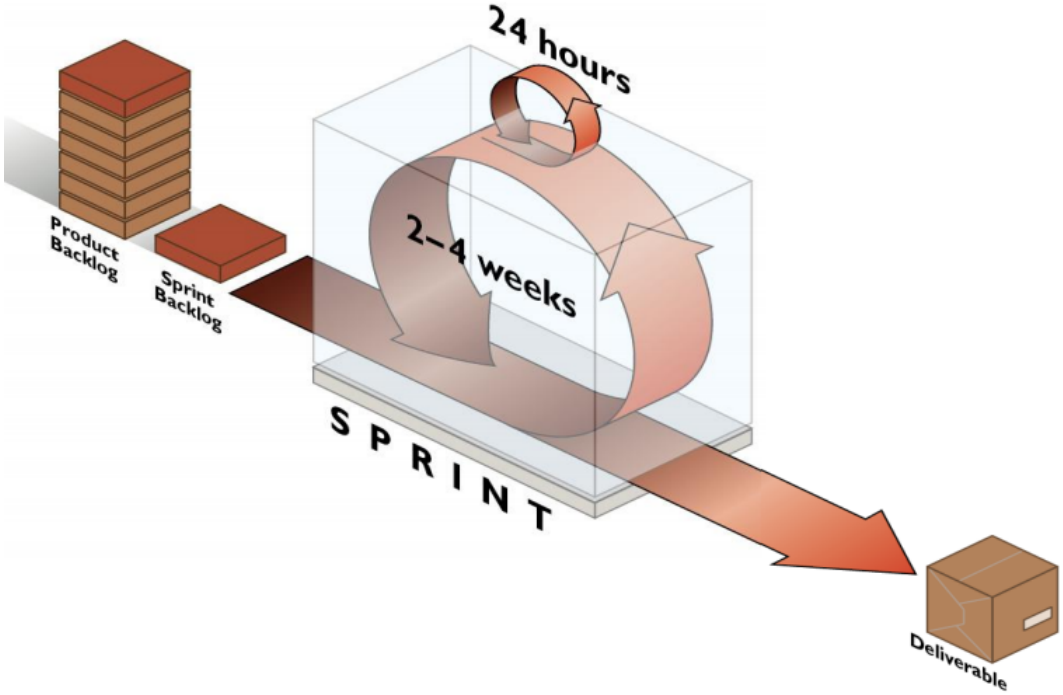



**Figura 5.1.** Reuniones metodología *SCRUM* [5].

Diariamente se realiza una reunión llamada *Daily Sprint*, en la cual cada miembro del equipo explica brevemente las tareas que han sido realizadas durante el día anterior y la previsión para ese día. Estas reuniones no duran más de 15 minutos y se realizan al inicio de la jornada laboral.

La persona encargada del correcto funcionamiento del equipo es el **Scrum Master.** Su trabajo consiste en eliminar las dificultades que se presenten para la consecución de los objetivos de un *sprint*. No es el líder del equipo, simplemente es un *facilitador* que hace que se cumpla la metodología de trabajo y se encarga de tareas administrativas.

El resto del equipo de trabajo está compuesto por desarrolladores, que se encargan de la realización del software de forma escalonada y adecuándose a los objetivos definidos en cada *sprint planning*.

Para la correcta realización de la metodología de trabajo SCRUM se utiliza el software *Jira*. En esta herramienta queda especificado el *product backlog* y el *spring backlog*. Permite que cada persona pueda consultar las tareas que tiene asignadas para un *sprint*.

#### 5.1.1.2. Jira

JIRA es una aplicación basada en web que se puede utilizar para gestionar el SCRUM. Permite definir diferentes *backlogs*, el del proyecto y los de los *sprints*. Los *backlogs* se formalizarán en *features* que pueden ser historias de usuario, tareas, *Bugs*, documentación, etc. Da soporte al usuario a que se gestione su trabajo a realizar.

La figura 5.2 es una captura de pantalla de la herramienta Jira. Se observan las features asignados a un miembro del equipo y la descomposición del mismo. Esta tabla tiene varias columnas:

- **To Do.** Se muestran las *features* pendientes de realizar
- **Implementation.** *Features*, que se están desarrollando
- **Impeded.** Aquí se colocan las tareas que por algún motivo en concreto, que queda especificado, no se puede continuar con su desarrollo
- **Verification.** Fase de pruebas.
- **Done.** Se encuentran en esta columna las *features* y las partes en que han sido desglosadas, que ya han sido terminadas.

Cada una de las *features* puede ser descompuesta en actividades, para facilitar la organización, de este modo, una tarea se puede dividir en sub−tareas, o incluir algún *sub−bug* que haya surgido durante la fase de pruebas. Esto corre a cargo de cada uno de los miembros del equipo, usando su propio criterio, que le servirá de guía a lo largo del *sprint*.

Cualquier miembro del equipo de trabajo puede consultar el desarrollo que lleva cualquiera de sus compañeros en el *sprint*, lo que permite la reasignación de tareas. Además, es necesario incluir las horas de trabajo diarias que lleva la realización de cada una de las *feautes*, por lo que es más sencilla la planificación del siguiente *sprint*, repartiendo adecuadamente la carga de trabajo en función de la complejidad de las tareas y de la productividad de cada uno de los componentes.

#### 5.1.1.3. Programación extrema

La *programación extrema* es una metodología ágil que permite a los desarrolladores dar respuestas más rápidas a los cambios que vayan surgiendo en el proyecto. Esta metodología está muy orientada a aumentar la productividad de un trabajo en equipo. Algunas de sus principales características son:

- **Comunicación**. Existe un contacto permanente entre todos los actores partícipes en el proyecto
- **Sencillez**. Permite mantener el diseño limpio y sencillo.
- **Retroalimentación**. El código puede ser revisado por los miembros del equipo, ya sean todos, o sólo a los cuales les compete.

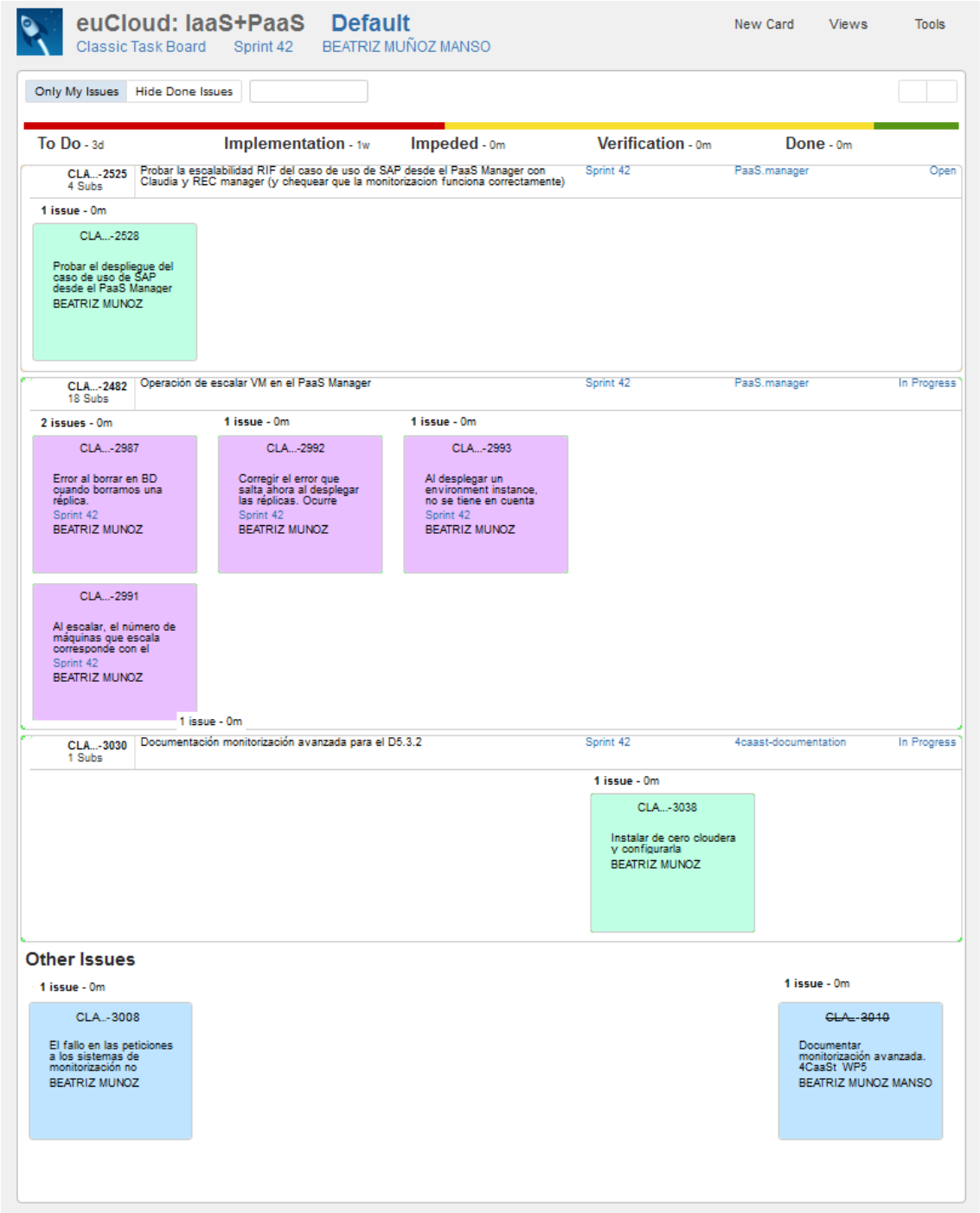



**Figura 5.2.** *Features* detallados para un *sprint* en Jira

Se establecen normas entre los miembros del equipo, que promueven la colaboración entre ellos y a su vez la autonomía de cada miembro. La interacción con los compañeros, la revisión de código vía *ReviewBoard* y las pruebas unitarias son ejemplos de buenas prácticas de desarrollo [20].

**ReviewBoard** es una herramienta que se utiliza para la metodología ágil de programación extrema. Varios miembros del equipo de desarrollo realizan modificaciones simultáneas sobre el código, por lo que es necesario seguir unas reglas establecidas. El *ReviewBoad* permite que el resto de desarrolladores revisen el código antes de subirlo al repositorio y realicen los comentarios y correcciones que consideren sobre el mismo. En el momento que todos los desarrolladores implicados den el visto bueno, el código será subido al repositorio, mientras tanto, no interferirá con el resto [21].

## 5.2. Tecnologías utilizadas

En esta sección se describen brevemente las tecnologías utilizadas para el desarrollo de este Proyecto Fin de Carrera, así como las herramientas necesarias para el desempeño del mismo. El contenido se desarrollará en varias secciones a lo largo de las cuales se analizará en el lenguaje de programación Java, la persistencia de datos, inversión de control, servicios web y empaquetado de aplicaciones. Además se incluyen unas pequeñas nociones sobre el control de versiones y otras herramientas que han sido indispensables para el desempeño del proyecto.

Dado que ya se ha hablado de la solución propuesta (ver capítulo 4), para el desarrollo del código y la aplicación web desarrollada en java desplegada en un tomcat donde la persistencia se almacena en una base de datos. Se ha utilizado *Hibernate* para persistencia de datos y *Spring* para inyección de dependencias.

### 5.2.1. Tecnologías de desarrollo: Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos multiplataforma desarrollado por *Sun Microsystems*.

En la actualidad, es el lenguaje de programación más extendido del mundo y dispone de multitud de entornos de programación *Open Source* y de pago. Crece constantemente gracias a su comunidad de desarrolladores que continuamente extienden su funcionalidad con nuevas librerías y utilidades. Existe una amplia documentación al respecto, así como están desarrolladas infinidad de librerías de clases. Java se ejecuta sobre una JVM que además necesita un compilador. Algunas de las características fundamentales de Java son las definidas a continuación.

**Orientación a Objetos.** Para que un lenguaje pueda considerarse orientado a objetos debe soportar como mínimo las características de encapsulación, herencia, polimorfismo y enlace dinámico; algo que Java proporciona.

**Robustez.** En lugar de los punteros se emplean referencias a objetos, los cuales son identificadores simbólicos. El gestor de memoria de Java lleva una contabilidad de las referencias a los objetos.

**Portabilidad.** Las aplicaciones, una vez compiladas, pueden ser ejecutadas en cualquier máquina, sobre cualquier sistema operativo siempre que la máquina virtual de Java (JMV) se encuentre instalada. Es un lenguaje interpretado, es decir, al compilar su código fuente no se genera código máquina, sino un pseudocódigo que es interpretado por la JVM.

**Simplicidad.** Las facilidades básicas del lenguaje son muy pocas, sin embargo a través de sus librerías de clases se proporciona un gran número de funcionalidades extra. Las librerías se pueden considerar como módulos que proporcionan funcionalidades concretas.

**Seguridad.** Al no utilizar punteros para manipular memoria por parte del desarrollador, no es posible acceder a recursos del sistema de forma no controlada. El compilador de Java efectúa una verificación sistemática de conversiones [22].

#### 5.2.1.1. Entornos de desarrollo: Eclipse

Para desarrollar código Java, se ofrecen diferentes entornos de desarrollo para el desarrollo de código. El más utilizado es *Java Development Kit* (JDK), compuesta por un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java.

Existe también una versión reducida del JDK, la máquina denominada JRE (Java Runtime Environment) que únicamente es una VM de Java. Gracias a los distintos *entornos de desarrollo integrado* (IDEs), en un mismo entorno se puede escribir, compilar y ejecutar el código. También permiten depurar el código en un entorno gráfico. Se consigue un desarrollo de aplicaciones más rápido ya que permiten incorporar librerías de componentes desarrollados por el usuario de forma más sencilla. El IDE utilizado ha sido Eclipse [22].

**Eclipse** es un IDE de código abierto y multiplataforma para el desarrollo de proyectos software, desarrollado actualmente por la Fundación Eclipse, organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La principal función de Eclipse es proporcionar herramientas integradas útiles, y reglas a seguir, que permiten el desarrollo de software. A través de APIs definidas, se facilita el trabajo del desarrollador [23].

#### 5.2.1.2. Máquinas Virtuales de Java

Cualquier desarrollo de software realizado en Java requiere de la traducción del código para que pueda ser interpretado, y de su compilación. Para ello, se utilizan dos máquinas virtuales Java.

**Java Virtual Machine** La *Java Virtual Machine* (JVM), es una máquina virtual de proceso nativo, es decir, se ejecuta como un proceso normal dentro de un sistema operativo específico y soporta un único proceso.

En Java, tras compilar los archivos, en los ficheros **.class* generados, están las instrucciones en «bytecode», que es un código binario especial. Estos ficheros son ejecutados por la JVM, convirtiéndolos al código particular requerido por la CPU [24].

**Compilador de Java** Se trata de una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java (con extensión *.Java). Si no encuentra errores en genera los ficheros compilados (con extensión *.class). En otro caso, indica las líneas donde se ha incurrido en errores.

Para trabajar con aplicaciones web con acceso a bases de datos, ver sección 5.2.4, es necesario implementar mecanismos de persistencia de datos.

### 5.2.2. Persistencia de datos

Al desarrollar una aplicación web que utiliza bases de datos para el almacenamiento, es necesario que dichos datos persistan en la misma. En Java los objetos tienen un estado y comportamiento determinado. Sin embargo, las bases de datos relacionales almacenan la información mediante tablas, filas, y columnas, de manera que para almacenar un objeto hay que realizar una correlación entre el sistema orientado a objetos de Java y el sistema relacional de nuestra base de datos. Para realiza la conversión de los datos se utilizan técnicas basadas en *Object-Relational mapping* (ORM), que permite dicha conversión utilizando motores de persistencia [25].

#### 5.2.2.1. API de persistencia de datos de Java

*Java Persistence API*, también conocido como JPA, es la API de persistencia desarrollada por *Sun Microsystems* para la plataforma Java EE. Es una API del lenguaje de programación Java que gestiona datos relacionales en las aplicaciones, preservando la información de forma permanente.

JPA es una abstracción que nos permite realizar dicha correlación de forma sencilla, realizando por nosotros toda la conversión entre nuestros objetos y las tablas de una base de datos. Por supuesto, JPA también nos permite seguir el sentido inverso, creando objetos a partir de las tablas de una base de datos, y también de forma transparente. Los servidores de aplicaciones web permiten el acceso a recursos externos como sistemas de gestión de bases de datos o de colas de mensajes. El servidor registra los recursos, asociándoles un nombre accesible. JPA establece una interfaz común que es implementada por un proveedor de persistencia de nuestra elección (como *Hibernate* en nuestro caso).

El *Lenguaje de Consultas de Persistencia de Java* (JPQL) permite especificar consultas a bases de datos relacionales en forma parecida a SQL desde un programa Java.

Además la API de persistencia permite especificar que las acciones de inserción y/o borrado de objetos deben propagarse a través de determinadas relaciones [26].

#### 5.2.2.2. Hibernate

*Hibernate* es el motor de persistencia de objetos relacionales para la plataforma Java más utilizado. Soluciona el problema de la diferencia de modelos de datos de la aplicación, ya que facilita el mapeo entre el modelo de objetos de la memoria y los atributos de una base de datos relacional. Para ello utiliza anotaciones en las entidades declaradas en el código, a través de las cuales establece relaciones entre los atributos de una base de datos y el modelo de objetos de la aplicación.

Hibernate permite que la aplicación manipule la base de datos operando sobre objetos. Además realiza las conversiones necesarias en los datos entre los tipos utilizados en Java y los definidos en la base de datos. Está diseñado para adaptarse a una base de datos ya existente, aunque también puede crearla a partir de la información de la que dispone.

Se realizan anotaciones en las clases para cada uno de los atributos definidos. Estas anotaciones se referirán a tamaños, relaciones entre atributos, etc. En el siguiente cuadro se muestran, a modo de ejemplo, tres anotaciones realizadas.

```
@ManyToOne
private Tier tier ;

@Column(length=30000)
private String ovf ;

@Embedded
private VM vm;
```

La anotación *@ManyToOne* permite que se creen relaciones de muchos objetos a un mismo *tier*. La variable *ovf* se almacenará en una columna de la tabla, y no podrá estar compuesta por más de 30.000 caracteres como se indica en la anotacion de la longitud de la columna.

Con la anotación *@Embedded*, indicamos que el campo *vm* puede ser integrable. Para que funcione, la clase *VM* tendrá la anotación *@Embeddable*, es decir, que será integrable dentro de otras entidades.

Con Hibernate, a partir de los ficheros de configuración .hbm.xml, podemos construir automáticamente los fuentes Java correspondientes a los beans de las tablas. Es decir, por cada tabla construye una clase Java cuyos atributos son las columnas de la tabla y con los métodos set() y get() correspondientes a esos atributos. Se debe tener en cuenta que el método setId() correspondiente al atributo id que se usará como clave primaria en la base de datos. Es un método privado, aparecerá en la clase de la que forme parte

de manera obligatoria [27].

### 5.2.3. Inversión de control

El método de programación *inversión de control* invierte el flujo de ejecución de un programa respecto a la programación tradicional. En lugar de indicarse las acciones de forma secuencial, la inversión de control permite especificar las respuestas que se deben obtener a solicitudes concretas [28]. Para su realización, es necesario utilizar algún tipo de arquitectura que permita gestionarlo, como por ejemplo *spring*.

#### 5.2.3.1. Spring

*Spring* es un *Framework* para simplificar el desarrollo de aplicaciones web en Java, permitiendo la inversión de control.

La inversión de control es un modelo donde es la biblioteca la que invoca al código, y no al revés, como se hace en la programación tradicional. Para implementar la inversión de control, se utiliza la inyección de dependencias, que es un patrón de diseño en el cual los objetos no son creados por la clase en que se van a utilizar, sino que los objetos son suministrados a esa clase.

Para implementar la *inyección de dependencias*, es necesario implementar un contenedor, que inyecta a cada objeto los objetos necesarios según se establece en un fichero de configuración. Este contenedor es implementado por Spring. Un ejemplo de este fichero sería el siguiente:

```
<?xml version ="1.0" encoding ="UTF-8"?>
-<beans default-autowire ="no"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.springframework.org/schema/beans">

<bean class="manager.impl.TierInstanceManagerImpl"
lazy-init="true" id ="tierInstanceManager">
<property ref="tierInstanceDao" name="tierInstanceDao"/>
<property ref="tierDao" name="tierDao"/>
<property ref="productInstanceDao" name="productInstanceDao"/>
<property ref="productReleaseDao" name="productReleaseDao"/>
</bean>

</beans>
```

El fichero de configuración está compuesto por *Beans*. En cada *bean* se indican las dependencias que se pueden inyectar a una clase en concreto [29].

En este *bean* se indica que la clase *TierInstanceManagerImpl* del paquete *paasmanager.manager.impl* tiene el nombre de referencia *tierInstanceManager* para cuando

sea referenciado por algún *bean* correspondiente a otra clase. También se indica que se inyectarán objetos de las clases *tierInstanceDao*, *tierDao*, *productInstanceDao* y *productReleaseDao*.

Para cada uno de los objetos que se inyecten las dependencias, se creará un método set en la clase de la cual estamos creando el *bean*, de forma que no es necesario crear un objeto para esas clases ya que los objetos quedan ya referenciados.

### 5.2.4. Tecnologías de gestión de bases de datos

Al utilizar bases de datos para el almacenamiento persistente, es necesario utilizar un lenguaje que nos permita acceder a las mismas para realizar operaciones así como un gestor de bases de datos.

#### 5.2.4.1. Lenguaje de consultas SQL

El lenguaje de consulta estructurado (SQL), es un lenguaje que permite el acceso a bases de datos relacionales, permitiendo realizar operaciones sobre los datos, como pueden ser consultas, inserciones, modificaciones, y eliminación de datos. Los comandos básicos para consultas en la base de datos son:

- SELECT: indica los atrubutos a los que se quiere acceder de la *Base de Datos* (BD).
- FROM: especifica todas las relaciones o tablas que se necesitan para la consulta
- WHERE: especifica las condiciones para seleccionar la *Tupla* o tuplas, de las relaciones especificadas en FROM.

Las operaciones se pueden dividir en dos grupos que son, el DDL (Data Definition Language, Lenguaje de definición de datos) para manipular objetos de la base de datos como crear una base de datos y especificar su estructura; y DML (Data Manipulation Language, Lenguaje de manipulación de datos), para manipular los datos de la base de datos [30].

#### 5.2.4.2. Gestor de bases de datos: PostgreSQL

PostgreSQL es uno de los sistemas de gestión de bases de datos relacional más potentes del mercado *open source*. Un Sistema de gestión de Bases de Datos (SGBD) es una aplicación informática que permite a los usuarios definir, crear, mantener y consultar una base de datos; así como proporciona acceso controlado a la misma. Utiliza como lenguaje de consulta SQL, ver sección 5.2.4.1.

Soporta distintos tipos de datos. Además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, etc. También permite la creación de tipos propios. Para ello utiliza arrays como estructura para los datos.

Permite la declaración de funciones propias, así como la definición de disparadores (ejecución de un procedimiento almacenado, basado en una determinada acción sobre una tabla específica). Incluye herencia entre tablas por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.

Incorpora la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit.

Para el acceso a este gestor de Bases de Datos se puede utilizar Hibernate, (ver sección 5.2.2.2). PostgreSQL permite el almacenamiento de grandes cantidades de datos durante largos periodos de tiempo, manteniéndolos seguros de accidentes o uso no autorizado. Los datos que se almacenan pueden llegar a tener un gran nivel de complejidad, y aun así, utilizando este sistema se reduce el espacio de almacenamiento y las redundancias [31].

### 5.2.5. Empaquetado de aplicaciones

El empaquetado de aplicaciones consiste en proporcionar las aplicaciones en forma de paquetes. Estos paquetes están formados por los programas ejecutables de la aplicación, así como por todas las bibliotecas de las que depende y cualquier otro fichero que sea necesario para la correcta ejecución de la aplicación. Permite evitar los problemas de las dependencias tanto a la hora de instalar la aplicación como al usarla, ya que cada paquete lleva consigo sus dependencias [32].

#### 5.2.5.1. Apache Maven

*Apache Maven* es un software de gestión de proyectos basado en el concepto de un modelo de objeto de proyecto (POM) para proyectos desarrollados en Java. Es similar en funcionalidad a *Apache Ant* pero tiene un modelo de configuración de construcción más simple, basado en un formato XML.

Una característica clave de Maven es que está listo para usarse en red. Puede descargar dinámicamente *Plugins* de un repositorio que también provee acceso a muchas versiones de diferentes proyectos *open source* en Java, Apache y otras organizaciones. Maven provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios.

Maven se contruye usando una arquitectura basada en *plugins* que permite que utilice cualquier aplicación a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir *plugins* para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etc. para cualquier otro lenguaje.

*Project Object Model* (POM) es utilizado por Maven para describir el proyecto de

software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Un ejemplo de POM sería el siguiente:

```
<?xml version="1.0" encoding ="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.telefonica.euro_iaas</groupId>
 <artifactId>paas-manager-core</artifactId>
 <packaging>jar</packaging>
 <name>paas-manager-core</name>
 <parent>
  <groupId>com.telefonica.euro_iaas</groupId>
  <artifactId>paas-manager-server</artifactId>
  <version>0.0.3-SNAPSHOT</version>
 </parent>

 <dependencies>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>3.3.0.SP1</version>
        </dependency>
        <dependency>
                <groupId>com.telefonica.euro_iaas</groupId>
                <artifactId>commons-dao</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>\${spring.version}</version>
        </dependency>
        <dependency>
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-client</artifactId>
                <version>1.6</version>
        </dependency>
 </dependencies>
 <build>
  <plugins>
         <plugin>
          <groupId>org.apache. maven.plugins</groupId>
          <artifactId>maven-javadoc-plugin</artifactId>
         </plugin>
  </plugins>
 </build>
</project>
```

En este POM se definen las características que queremos dar al proyecto, indicando el número de versión que soporta, el grupo al cual pertenece (*groupId*), el formato de empaquetamiento que se va a utilizar (*artifactId*), el nombre por el que es conocido el proyecto, las dependencias y los *plugins* a utilizar y el proyecto padre al que pertenece.

Cada proyecto va a depender de otros para su correcto funcionamiento, por lo que indicando las relaciones que existen, a través de las dependencias, Maven se encarga de enlazar y compilar los proyectos que se indiquen, es decir, gestiona las relaciones entre los distintos proyectos. A través de los *plugins* especificados en el POM, será posible la correcta compilación y empaquetado de la aplicación.[33]

### 5.2.6. Servicio web

Un sistema software diseñado para soportar interacciones entre máquinas sobre una red se denomina **servicio web**. Los Servicios web suelen ser APIs que pueden ser accedidas vía web y son ejecutados en el sistema que los aloja. La arquitectura de software que se utiliza como API en este proyecto es *Representational State Transfer* (REST).

Los Servicios web basados en REST intentan emular al protocolo HTTP o protocolos similares mediante la restricción de establecer la interfaz a un conjunto conocido de operaciones. Cabe destacar que REST no es un estándar, ya que es tan sólo un estilo de arquitectura. No obstante está basado en estándares como HTTP o URL [34].

Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial. Además REST permite la asincronía entre cliente y servidor, ya que no es necesario que estén en constante comunicación mientras alguno de ellos está realizando una operación [26].

*Jersey* es la librería de Java que será utilizada para integrar REST en el software desarrollado.

#### 5.2.6.1. Jersey

Jersey es un software libre que permite la creación de servicios web REST en Java. Para proveer a nuestra aplicación de este servicio, se realizan anotaciones en las clases. Algunas de las anotaciones que se realizan son:

- *@GET:* El método anotado corresponde a una petición HTTP GET.
- *@POST:* el método anotado corresponde a una petición HTTP POST.
- *@Path:* Sirve para indicar la URI de una clase o método que sirve las peticiones.

En el siguiente recuadro se muestran, a modo de ejemplo, algunas de las anotaciones realizadas en las clases implementadas en el software de este proyecto.

```
@Path("/envInst/org/{org}/vdc/{vdc}/environmentInstance/
{environmentInstance}/tierInstance")
@Scope("request")

@GET
@Path("/{tierInstance}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})

@POST
@Path("/")
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
```

En la implementación de la clase, se escribe el *path* completo que debe ser llamado, así como se indica a través de la anotación *@Scope* que se devolverá una única instancia por cada solicitud recibida.

El tipo de método que debe ser invocado para llamar a un método público concreto de la clase, se especifica en la interfaz, indicando además el *path* relativo. La anotación *@Consumes* especifica el formato que debe tener el *Payload* de la petición, y la anotación *@Produces* el formato que tendrá el cuerpo de la respuesta que se envíe.

Además, se crea un fichero XML dentro de la aplicación para configurar el Servlet Jersey de la aplicación. A continuación se muestra un ejemplo de un fichero de configuración. [35]

```
<web-app xsi:schemaLocation="http://Java.sun.com/xml/ns/javaee
http://Java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<servlet>
<servlet-name>Paas Manager Server Rest Servlet</servlet-name>
<servlet-class>
com.sun.jersey.spi.spring.container.servlet.SpringServlet
</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>Paas Manager Server Rest Servlet</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

### 5.2.7. Control de versiones

Un sistema de control de versiones permite gestionar los diversos cambios que se realizan sobre un desarrollo software. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado.

Un sistema de control de versiones proporciona un mecanismo de almacenamiento donde se permite realizar cambios sobre los elementos almacenados. Realiza un registro sobre todas las acciones realizadas de manera detallada, permitiendo volver a versiones anteriores del software [36].

#### 5.2.7.1. Git

*Git* es un software de control de versiones, que se gestiona los proyectos de forma ramificada. Cada desarrollador de un mismo proyecto puede tener su rama local independiente. Git permite la fusión del código. Pueden crearse varios niveles en el repositorio, que permiten la gestión del código de manera independiente o conjunta según las necesidades que vayan surgiendo a lo largo del desarrollo. Cuando se quiere actualizar el repositorio remoto, no es necesario subir todas las ramas. Se puede elegir cuales se comparten. Es decir, un desarrollador puede crear su propio código, subirlo a su rama y probarlo, sin que interfiera con el resto de código, y una vez que se desarrolle la funcionalidad completa, se podrá compartir con el resto. De esta forma, se puede desarrollar sin preocuparse de la planificación constantemente, y de la fusión de código.


**Figura 5.3.** Ramificación Git [6].

Existen tres tipos de ramas que se pueden crear en git, atendiendo a la figura 5.3:

- **Master.** Es la rama principal.
- **Develop.** Es la rama de desarrollo de cada usuario.
- **Topic.** Ramas que sirven para añadir algo concreto.

Cualquiera de los tres tipos de ramas puede ser creada, modificada, compartida y borrada [6].

#### 5.2.7.2. Subversion

Subversion es un sistema de control de versiones *open source*. El código desarrollado se encuentra en un repositorio en el cual se va actualizando a medida que el desarrolla-

dor quiera, permitiendo volver a versiones anteriores en caso de que sea necesario. Las acciones más básicas que se pueden realizar sobre el repositorio son las siguientes:

- **Check-out**. Descargar del repositorio una copia de la última versión en local
- **Update**. Actualiza la versión local, con la última versión del repositorio
- **Commit**. Actualizar el repositorio de versiones con los cambios realizados en local
- **Merge**. Integrar cambios realizados sobre un fichero desde varias versiones locales. Esto es muy importante realizarlo si el desarrollo corre a cargo de varias personas.

De esta forma, se facilita el desarrollo de código en proyectos con envergadura considerable [37].

**TortoiseSVN:** Para la simplificación del uso de subversion, se utiliza TortoiseSVN. Es un cliente Windows, que proporciona una interfaz gráfica tanto para las distintas actuaciones que se quieran realizar en el repositorio SVN, como a través de imágenes en archivos y carpetas que indican si un archivo ha sido modificado o no por el desarrollador respecto a la última versión descargada del repositorio. También se muestran las incompatibilidades de código que se producen al actualizar la versión. Esta herramienta es compatible con el uso de línea de comandos de la shell de Windows [38].

### 5.2.8. Otras Herramientas utilizadas

En este apartado se explican brevemente algunas de las herramientas que han sido necesarias a lo largo del desarrollo del proyecto.

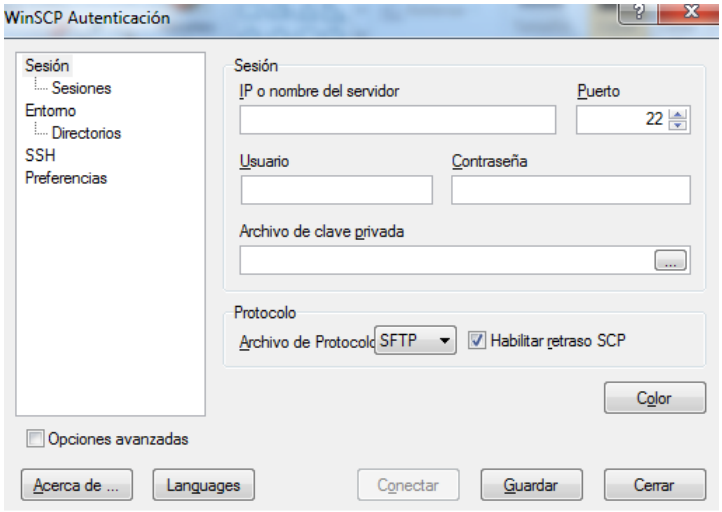
#### 5.2.8.1. Putty

PuTTY es un cliente SSH y telnet, para la plataforma Windows. Su nombre proviene de las siglas Pu: Port unique TTY: terminal type. Su traducción al castellano sería: Puerto único para terminales de teletipo. En el desarrollo en un entorno *cloud*, se hace necesaria la utilización de protocolos como SSH para poder acceder a máquinas virtuales para la gestión de la mismas, la instalación de software y resto de acciones necesarias, que, a fin de cuentas, son las mismas que se realizan en cualquier máquina física.

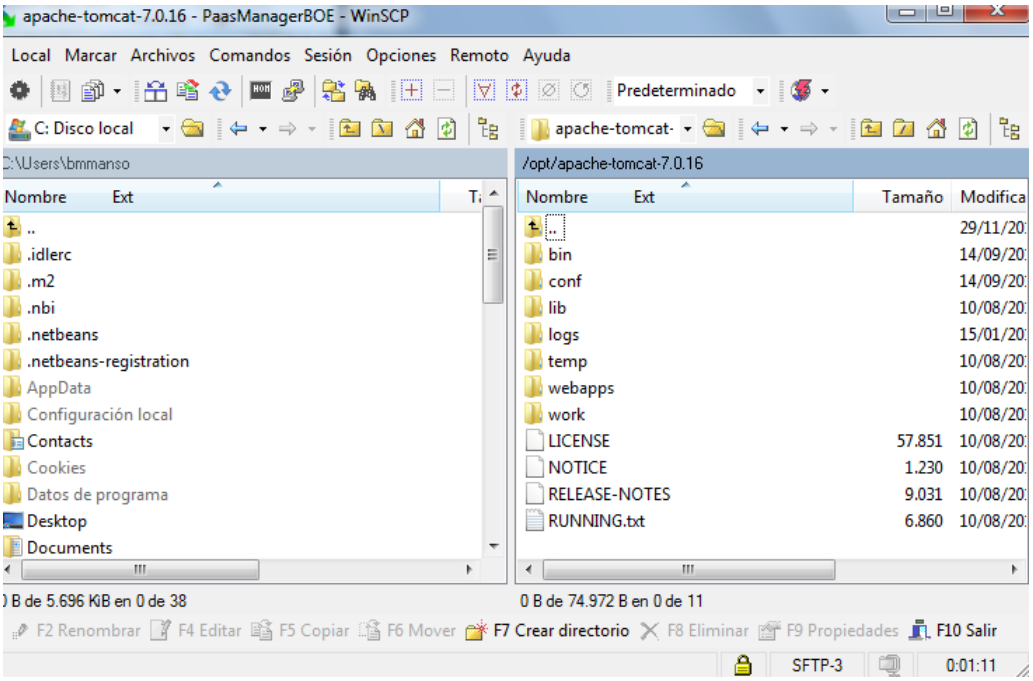
Es un software de código abierto, que proporciona un acceso más sencillo a la *Shell*, aunque SSH también puede ser invocado por línea de comandos [39].

#### 5.2.8.2. WinSCP

WinSCP es una aplicación cliente SCP y SFTP que incorpora interfaz gráfica, y emplea SSH. Para la transferencia de archivos y software que se desarrolla en local, el uso de esta herramienta permite copiarlos a máquinas virtuales de forma segura. Por supuesto también se pueden descargar archivos remotos. La interfaz gráfica simplifica las acciones a realizar, tanto sobre contenidos como para la gestión de los protocolos empleados en lo que ha sesiones se refiere. En la figura 5.4 está la interfaz gráfica de usuario [40].




(a) Interfaz gráfica de usuario de *WinSCP* antes de la conexión




(b) Interfaz gráfica de usuario de *WinSCP* después de la conexión

**Figura 5.4.** Interfaz gráfica de usuario de *WinSCP*

#### 5.2.8.3. JMeter

*JMeter* es una herramienta, desarrollada por Apache, que se utiliza para probar el rendimiento de los recursos. Se utiliza para realizar pruebas unitarias, ya que permite simular una carga pesada en un servidor y en una red entre otras cosas. Gracias a ello se puede analizar el rendimiento general de una aplicación web, por ejemplo, ante distintos tipos de carga. Para esto último, Jmeter genera hilos que se inyectan como eventos en la aplicación de forma que se simulan accesos a la misma. Aunque la generación de carga es su uso más habitual, también se utiliza para comprobar la veracidad de los datos recibidos, entre otros muchos datos que se monitorizan [41].

# Capítulo 6
# Estudio de ahorro de costes

Alojar aplicaciones web en el *cloud*, puede suponer una gran disminución de costes a las empresas, si su demanda es variable en función de las épocas del año. A lo largo de este capítulo se demostrará con cifras numéricas, el ahorro que puede suponer para una organización el despliegue de su página web en la nube.

La cantidad de recursos hardware necesarios para que una aplicación web ofrezca servicio a todos los usuarios está directamente relacionado con el número de visitas que recibe.

Los costes de mantenimiento del hardware serán, por tanto, proporcionales a la carga. Será deseable que la aplicación web tenga una alta disponibilidad, ofreciendo servicio a todos los usuarios.

En la figura 6.1 se muestra un gráfico con el número de visitas diarias reales recibidas por la página web *java-spain.com*. El horizonte temporal analizado para el estudio debe de ser suficientemente significativo para permitir la comparativa. Se utilizan datos recogidos durante dos años.

El número de visitas mostradas en la figura 6.1 es muy irregular, esto es que la demanda de la página web *java-spain.com* es muy cambiante, con periodos de fluctuación en muchas ocasiones diarios.

## 6.1. Alojar la aplicación en la infraestructura hardware de la empresa

La decisión de alojar la aplicación en los propios recursos hardware de la empresa, supone que estos deben ser suficientes para dar servicio a todos los usuarios en cualquier momento.

Es decir, se debe realizar un aprovisionamiento de infraestructura y software que permita ofrecer el servicio independientemente de la demanda del mismo. La línea verde de la figura 6.2 correspondería con la capacidad que debe ser adquirida para la

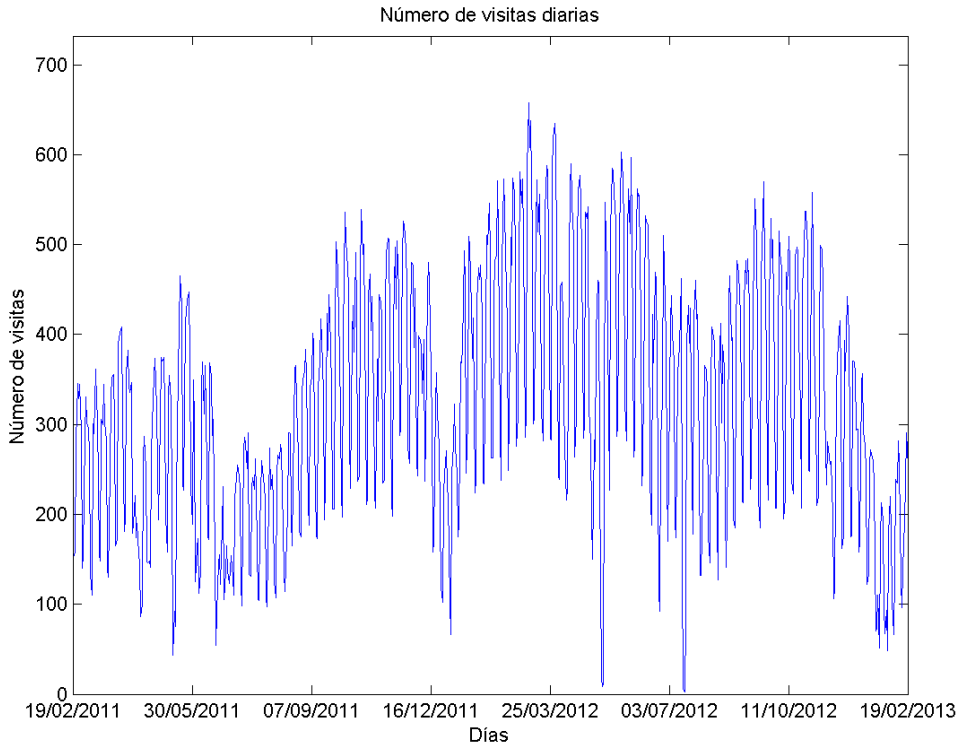



**Figura 6.1.** Visitas diarias recibidas por la página web *java-spain.com*

aplicación web, si se pretende cubrir toda la demanda.

Toda la demanda será asumida a cambio de una infrautilización de la infraestructura en la mayoría del tiempo. Si los recursos fueran menores no se podría ofrecer el servicio a todos los usuarios siempre. El coste inicial que asumirá la empresa, por tanto, será equivalente al coste de los recursos necesarios para satisfacer la demanda máxima.

Todo esto en el mejor de los casos, ya que la demanda puede ser estimada a priori, pero no es conocida, por lo que se puede realizar un mayor aprovisionamiento de infraestructura para dar servicio a todos los usuarios potenciales. Provocaría un gran incremento en los costes iniciales de puesta en funcionamiento de la aplicación.

La infraestructura que pueda ser adquirida por la empresa se volverá obsoleta con el paso de los años. Deberá ser renovada periódicamente, por lo que además de los costes iniciales que supone la inversión en infraestructura existirán futuros costes de renovación de los recursos hardware.

El mantenimiento de la infraestructura hardware, además, lleva asociados consigo costes de mantenimiento, para permitir la gestión de la aplicación y su correcto funcionamiento.

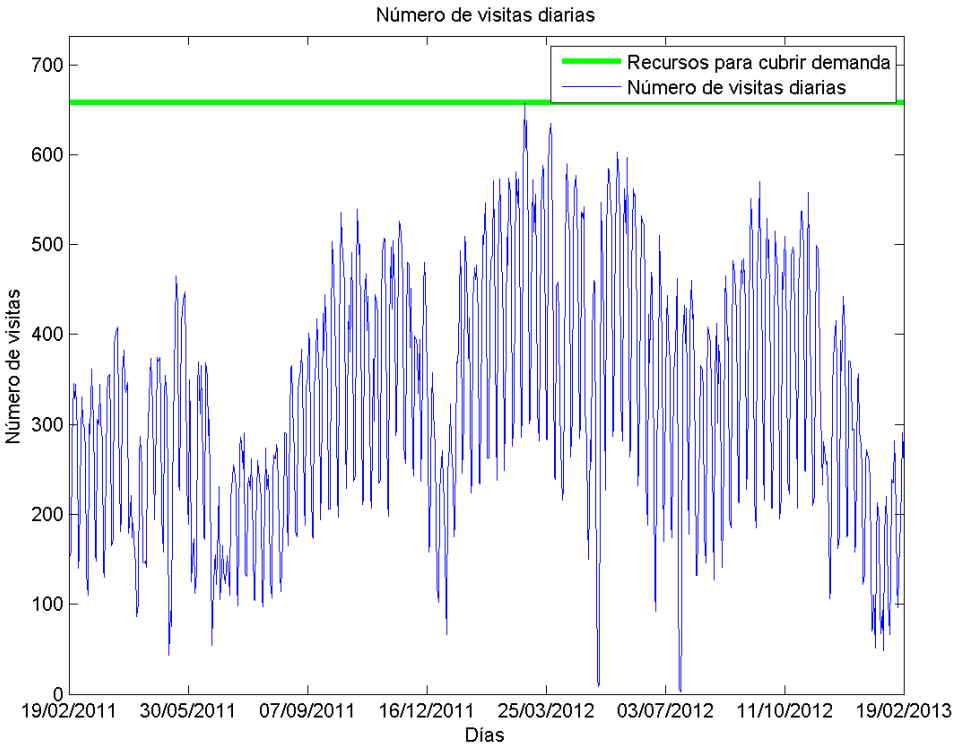



**Figura 6.2.** Recursos disponibles con adquisición de infraestructura hardware por parte de la empresa

## 6.2. Alojar la aplicación en la nube

Gracias al modelo de pago por uso, el coste asociado a los servicios en el *cloud* es variable, y previsiblemente inferior al incurrido con el uso de tecnología tradicional. La importancia del concepto **bajo demanda** asociado al uso de soluciones en la nube radica en que, a diferencia de lo que ocurre en el caso de la infraestructura convencional, la empresa tan sólo paga por el uso que hace del servicio de *cloud computing* en cada momento.

Existen muchos proveedores que ofrecen servicios relacionados con el despliegue de aplicaciones en el *cloud*, como Amazon, a través de su producto *AWS Elastic Beanstalk*, estudiado en la sección 2.3.1, que será utilizado para realizar la comparativa.

El alquiler de recursos para alojar la aplicación en *Amazon* se realiza a través de máquinas virtuales. El número de VMs utilizadas por la aplicación se amoldará a la demanda del servicio, pagando únicamente por los recursos que son necesarios.

Se realiza la suposición de que todas las visitas a la página web *java-spain.com* suponen la misma carga para la aplicación. Esto permitirá simplificar los cálculos, ya que el objeto de este capítulo es realizar una aproximación sobre el ahorro de costes que supone alojar la aplicación en la nube, y no obtener unos datos complemente fidedignos.

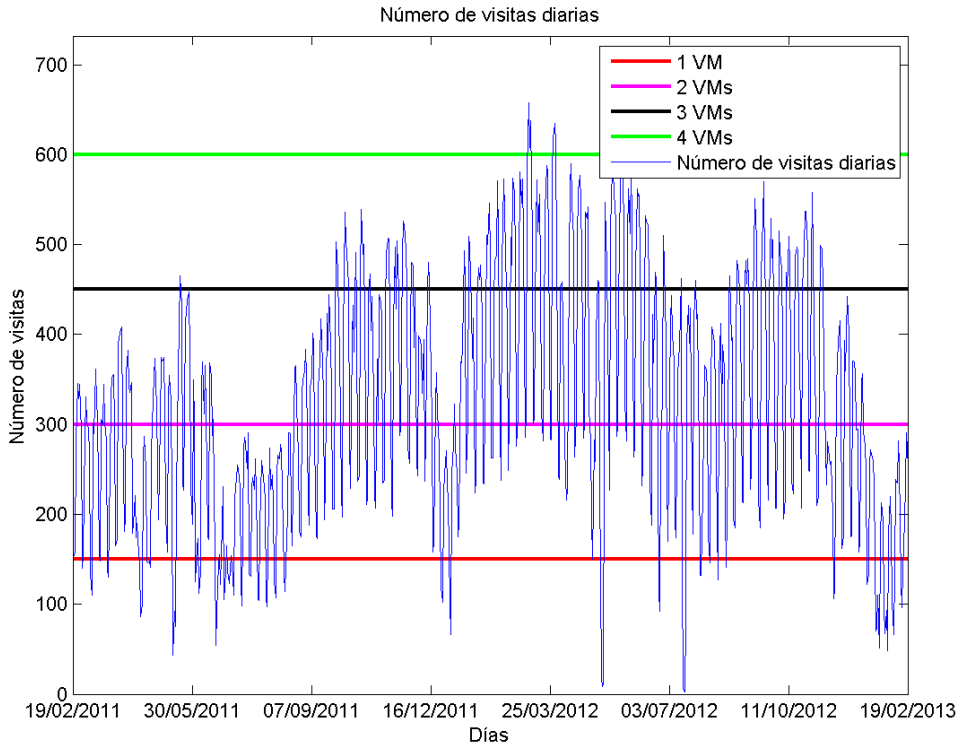



**Figura 6.3.** Infraestructura necesaria para alojar la aplicación de la empresa en el *cloud*

De acuerdo a la suposición anterior, se estimará que una VM de *AWS Elastic Beanstalk* es capaz de soportar la carga que generan 150 visitas. En la figura 6.3, se muestran el número de máquinas virtuales necesarias para satisfacer la demanda en cada momento.

Como máximo, teniendo en cuenta la representación de la figura 6.3, será necesaria la infraestructura provista por 5 máquinas virtuales. No obstante, si la demanda tuviera picos mayores, se incrementarían los recursos para ofrecer el servicio a todos los usuarios. La aplicación siempre estará desplegada, aunque no esté siendo visitada, por lo que, al menos, la infraestructura ofrecida por una VM debe estar disponible. En cualquier caso, la demanda quedará garantizada.

La empresa, no incurrirá en costes iniciales de aprovisionamiento de recursos, ni tampoco en los relacionados con la gestión de los mismos. La infraestructura se alquila al proveedor, pagando en función de los recursos utilizados, y la gestión va incluida en el precio. Además tampoco es necesaria la renovación del hardware, ya que es el proveedor quien se encarga de administrar y gestionar todos sus recursos.

## 6.3. Costes asociados a la aplicación

Para cada uno de los dos posibles modelos que puede elegir la empresa, los costes a tener en cuenta serán distintos. Se realizará el estudio a lo largo de dos años, tanto

si se aloja la aplicación en la nube como si la empresa se provee de la infraestructura necesaria para hacerlo.

Se calcularán los costes específicos a cada uno de los modelos, dejando de lado aquellos que suponen la misma cuantía independientemente de la opción elegida por la empresa.

### 6.3.1. Costes asociados a alojar la aplicación en el *cloud*

*AWS Elastic Beanstalk* ofrece a sus usuarios todos los servicios necesarios para alojar una aplicación en la nube. Provee al servicio de un balanceador y red para cada una de las instancias de VM que se realice. Además incorpora la autogestión de la elasticidad, permitiendo que el número de máquinas instanciadas varíe de forma automática, adecuándose a la demanda de la aplicación en tiempo real.

Para realizar los cálculos, se supondrá que por cada 150 visitas diarias, será necesaria una nueva instancia de una VM en Amazon, con Linux instalado como sistema operativo. Todos los servicios necesarios para el correcto funcionamiento de la aplicación se incorporan en el precio del alquiler diario de cada una de las instancias.

| | **1 VM** | **2 VM** | **3 VM** | **4 VM** | **5 VM** | **Total** |
|---|---|---|---|---|---|---|
| **Número de días** | 87 | 261 | 221 | 156 | 7 | 732 |
| **Alquiler diario** | 5€ | 10€ | 15€ | 20€ | 25€ | |
| **Alquiler total** | 435€ | 2610€ | 3315€ | 3120€ | 175€ | 9655€ |
| **Inversión inicial** | 0 | 0 | 0 | 0 | 0 | 0 |
| **Mantenimiento** | 0 | 0 | 0 | 0 | 0 | 0 |
| **Gestión** | 0 | 0 | 0 | 0 | 0 | 0 |
| ***COSTE TOTAL*** | | | | | | **9.655€** |
| ***COSTE ANUAL*** | | | | | | **4.828€** |

**Tabla 6.1.** Costes asociados a alojar la aplicación en la nube

En la tabla 6.1 están reflejados los costes asociados a alojar la aplicación en la nube. El monto final corresponde únicamente al alquiler de los recursos en el proveedor, ya que solamente se analizan los costes específicos a la solución.

### 6.3.2. Costes asociados a alojar la aplicación a través de los recursos propios de la empresa

Para satisfacer la demanda es necesario que la empresa realice un aprovisionamiento de infraestructura que permita dar servicio a todos los usuarios que lo demanden. Como se muestra en la figura 6.2, será necesario que los recursos soporten 658 visitas diarias, que corresponde con la línea verde. Por lo tanto, será necesario disponer de los mismos recursos hardware que proporcionan cinco instancias de VMs en el *cloud*.

La vida útil de la infraestructura hardware es de 5 años, que corresponde con el máximo permitido para la amortización en el Plan General de Contabilidad de 2007.

La persona contratada para el mantenimiento del hardware y la gestión de la aplicación tendrá la formación adecuada para el puesto y trabajará la mitad de su jornada en estas tareas, suponiendo un coste para la empresa de 2000 euros mensuales, más dos pagas extras anuales por la misma cuantía (incluyendo las cotizaciones sociales).

El software necesario para desplegar la aplicación será Linux, lo que no supondrá un coste adicional para la empresa. La empresa deberá estar provista de una red que permita dar el servicio a toda la demanda.

| | **Total** |
|---|---|
| **Inversión inicial** | 4000€ |
| **Coste aprovisionamiento** | 1600€ |
| **Mantenimiento y Gestión mensual** | 1000€ |
| **Costes Mantenimiento y Gestión** | 28000€ |
| **Red Mensual** | 100€ |
| **Coste de Red** | 2400€ |
| ***COSTE TOTAL*** | **32.000€** |
| ***COSTE ANUAL*** | **16.000€** |

**Tabla 6.2.** Costes asociados a alojar la aplicación en la infraestructura de la empresa

Los costes asociados a alojar la aplicación en la infraestructura de la empresa se muestran en la tabla 6.1. Los costes totales propios del servicio de la aplicación están compuestos por el coste anual de la infraestructura hardware adquirida por la empresa y el alquiler de red, junto con los costes de mantenimiento y gestión. Estos últimos suponen, con gran diferencia, el mayor incremento de los costes.

## 6.4. Elección del modelo

En la sección anterior se han calculado los costes asociados a cada una de las dos posibles opciones que puede elegir la empresa para ofrecer el servicio. Dentro de los costes no se han tenido en cuenta, por ejemplo, los referentes al desarrollo de la aplicación, que tendrían el mismo monto en ambos modelos. El objetivo de este capítulo es calcular el ahorro de costes que supone un modelo frente al otro, no los costes que ha de asumir la empresa en cada uno de ellos.

En la tabla 6.3, se muestra los costes anuales concretos asociados a la utilización de cada uno de los dos modelos propuestos. Con los datos obtenidos, resulta obvia la elección de alojar la aplicación en la nube alquilando los recursos. El ahorro de costes que supone alojar la aplicación en el *cloud* asciende a 11.172 euros.

Al alojar la aplicación en la nube, además de suponer un gran ahorro para la empresa, se produce una externalización de la gestión al proveedor, lo que implica

| **Coste anual en la nube** | **Coste anual con recursos propios** | **Ahorro de Costes** |
|---|---|---|
| 4.828€ | 16.000€ | **11.172€** |

**Tabla 6.3.** Costes anuales asociados al servicio ofrecido a través de la aplicación

una mayor velocidad de respuesta a cualquier cambio en la infraestructura necesaria tanto hardware como software. Además, la empresa no debe preocuparse si existe algún problema de pérdida de datos, ya que será el proveedor quien copie, almacene y gestione las copias de seguridad, y en consecuencia se encargará de la recuperación de la información en caso necesario.

Los proveedores cuentan con altas medidas de seguridad que protegen su infraestructura física. La arquitectura lógica es muy segura y por ende cualquier aplicación allí alojada, gracias a la inversión realizada por el proveedor en software de seguridad. Cuentan con todos los elementos y funcionalidades fundamentales como el análisis de vulnerabilidades, *antimalware*, firewall, proxy, etc.

En conclusión, tras realizar el estudio, para una aplicación con demanda variable, lo más aconsejable tanto a nivel económico como de gestión es alojarla en la nube.

# Capítulo 7
# Conclusiones y líneas futuras

Este último capítulo presenta, a modo de conclusión, el logro de los objetivos alcanzados en este proyecto, así como las futuras líneas de trabajo que se pueden llevar a cabo para mejorar la escalabilidad de aplicaciones web alojadas en la nube.

## 7.1. Logros alcanzados

Al inicio de esta memoria, en el capítulo *Introducción*, se establecieron cuatro objetivos principales para la consecución del objetivo principal de este proyecto, el autoescalado de aplicaciones web en un *cloud* PaaS. A partir de la arquitectura explicada en el capítulo *Arquitectura general*, y la solución expuesta en el capítulo *Implementación y despliegue* se puede extraer el grado de consecución de cada uno de los objetivos.

El grado de consecución de los objetivos específicos se expone en las siguientes subsecciones.

### 7.1.1. Clonado de máquinas virtuales configuradas

Uno de los problemas, que se presentaban para lograr escalar una aplicación web alojada en un *cloud* PaaS, era que no se podrían utilizar los métodos tradicionales de escalado a través de imágenes dadas por el proveedor.

En el *Despliegue del entorno* se instalará todo el software necesario en los *tiers* que así lo requieran. El proceso de instalación de software es costoso y requiere un tiempo, que en muchos casos puede resultar excesivo para adaptarse a la demanda del servicio.

Se crearán por tanto, imágenes de las máquinas virtuales con el software instalado, en los *tiers* escalables. De esta forma, se consigue utilizar los mecanismos simples del escalado a nivel IaaS, clonando las máquinas virtuales de un *tier* a partir de una imagen configurada con el software necesario.

### 7.1.2. Contextualización de las máquinas virtuales

Cada una de las máquinas virtuales que conforman un *entorno* deben estar provistas de un identificador único que pueda ser conocido por ellas mismas.

*Claudia* introduce en cada una de las máquinas virtuales, un metadato llamado FQN que corresponde con un identificador único que además proporciona información sobre el nombre del *entorno*, el VDC al que pertenece y el *tier* del que forma parte, como se explica en la sección 4.4.1.

A partir de un *script* creado en las imágenes base que van a ser utilizadas para el despliegue, la máquina obtiene su FQN y lo almacena en un fichero. De esta forma, la máquina queda perfectamente contextualizada, ver sección 4.3.

### 7.1.3. Interacción con los sistemas de monitorización

La monitorización de las máquinas virtuales que forman el *entorno* que aloja una aplicación web es fundamental para el escalado.

Para obtener los datos, es necesario que sea la propia máquina la que los recolecte, por lo que la imagen base utilizada para crear una VM tendrá instalado un software que se encarge de esta acción, ver sección 4.3.

La contextualización de las máquinas virtuales permite asociar los datos a cada una de ellas de manera inequívoca, de forma que el *Paas Manager*, conociendo el FQN de la máquina, es capaz de indicar cuáles son las VMs de las que se deben almacenar las métricas.

*Claudia*, por su parte, se encargará del análisis de los datos monitorizados, permitiendo realizar acciones de escalabilidad cuando sea necesario, ver sección 4.5.

### 7.1.4. Configuración de las máquinas virtuales

La solución propuesta pasa por separar el *entorno* que aloja la aplicación web en *tiers*. Las máquinas virtuales interactúan para el funcionamiento del servicio, por lo que deben conocerse entre ellas.

El componente para la *Gestión de instalación de software* se encarga de que las máquinas que deben interactuar con otras, registren los datos que permitan la comunicación. Al ser escalable, el número de VMs de cada *tier* variará en función de la demanda, y por tanto, también se encargarán de ir actualizando los registros.

## 7.2. Futuras líneas de trabajo

La solución propuesta permite la escalabilidad automática de aplicaciones web desplegadas en un *cloud* PaaS, cumpliendo el objetivo final de este proyecto.

A lo largo de esta memoria se ha explicado como la escalabilidad permite amoldarse a la demanda de un servicio, atendiendo a unas métricas de monitorización obtenidas, que resultan representativas del rendimiento de la aplicación.

En ningún momento se tiene en cuenta que la variación de la demanda de un servicio a lo largo del tiempo es una función correlada, que sigue tendencias. El entorno social, geopolítico, tecnológico, económico y cultural afecta notablemente a estas variaciones. Una página web dedicada a la venta de ropa, por ejemplo, tendrá una mayor demanda todos los años en la campaña de Navidad, o la página web de un cine será más visitada los fines de semana.

Escalar una aplicación web es un trabajo que requiere un tiempo, y la solución propuesta es a posteriori. Esto es, una vez que la demanda aumenta, y el servicio deja de ofrecer la calidad deseada, se escala para retornar a los umbrales establecidos. Por el contrario, cuando se detecta la infrautilización de los recursos, se elimina el exceso.

Lo ideal sería poder realizar un estudio del histórico almacenado de las métricas para analizar las tendencias y obtener predicciones que permitan anticiparse a las fluctuaciones de la demanda.

Sería deseable conocer las tendencias alcistas y bajistas para poder adecuar los recursos a priori, y mantener la calidad del servicio constante a lo largo del tiempo, sin incurrir en sobre-aprovisionamiento.

Además, estimar si los datos obtenidos de las métricas de monitorización corresponden a tendencias o a simples picos de demanda, puede evitar que se desasignen recursos que en un pequeño intervalo de tiempo serán necesarios de nuevo. En ocasiones, es más conveniente un exceso de recursos durante un pequeño intervalo de tiempo, que adecuarlos exactamente a la demanda.

Un ejemplo que muestra la necesidad del análisis a priori sería el anunciado lanzamiento del *smartphone Nexus 4* por parte de *Google*, que hizo que miles de personas accedieran a su página web en los pocos minutos que tardaron en agotarse las existencias. Esto supuso un incremento altísimo de las visitas recibidas. Escalar los recursos según la variación real de la demanda, supondría que durante el tiempo que se tarda en escalar la infraestructura de la aplicación, el servicio estaría colapsado para los usuarios finales.

El trabajo debe centrarse por tanto, en realizar algoritmos capaces de analizar las tendencias de la demanda de una aplicación web, y extrapolar los resultados a al software que permita realizar una escalabilidad a priori, siempre que sea posible.

## 7.3. Valoración Final

La realización de este PFC ha supuesto para mí una experiencia personal muy gratificante. La oportunidad que me fue brindada para desarrollar mis aptitudes en esta beca, me ha enriquecido tanto a nivel de conocimiento proporcionándome una grata experiencia laboral, y por supuesto como persona.

A lo largo de todos los años de universidad he aprendido muchos conocimientos que siempre me preguntaba si el día de mañana cuando trabajara en una empresa me servirían para algo. Ahora puedo decir con rotundidad que sí, me han sido muy útiles, al igual que he adquirido mucha formación a lo largo de mi estancia en Telefónica.

La mayoría del desarrollo de software ha sido en Java, lenguaje que conocía gracias a asignaturas cursadas, y que he seguido aprendiendo en el desarrollo de este PFC. El empaquetado de aplicaciones e inyección de dependencias, eran cosas que me sonaban ligeramente en las que he profundizado a lo largo de estos meses.

La elección de mi tutor, Gregorio, no fue casual. Gracias a él la programación dejó de resultarme una tarea casi imposible y que no terminaba de entender, a convertirse en una forma de crear «cosas útiles» a partir de un fichero en blanco. Me enseñó las primeras nociones de persistencia de datos, y algunas de las buenas prácticas para el desarrollo de código que he utilizado además de mi primer contacto con el control de versiones.

La verdad, que la mayoría de los conocimientos que he aprendido en la carrera, y que he utilizado en este PFC me los ha impartido el departamento *GSyC*. Los conocimientos de *Shell*, que nos han acompañado a lo largo de toda la carrera, me han facilitado enormemente el trabajo con máquinas virtuales.

Creo, sinceramente, que he mejorado bastante mis habilidades como programadora, aprendiendo a resolver cuestiones por mis propios medios, a la vez que adquiría nuevos conocimientos. He aprendido como los distintos tipos de licencias que se utilizan sobre el código, limitan el acceso o la distribución de los resultados.

Trabajar en equipo en un proyecto como éste me ha enseñado mucho. He podido aprender de grandes profesionales que no han escatimado esfuerzos en enseñarme, he aprendido nuevas metodologías de trabajo que ni siquiera conocía y además, la satisfacción de que no es un trabajo que se ha realizado sólo para una evaluación posterior, sino que otras personas se benefician de él.

Los amigos y compañeros de la universidad me han apoyado muchísimo y ayudado en todo momento a que los estudios llegaran a su fin, y el PFC es la culminación de ello. La gran dificultad que entrañan estos estudios hace que entre todos hagamos piña y decidamos ayudarnos al máximo, si todos aportamos nuestro granito de arena a los demás el camino es un poco más sencillo, y en eso no tengo queja alguna, sino todo lo contrario.

Me siento muy orgullosa del trabajo que he realizado y todo el esfuerzo que ha llevado la consecución de este proyecto. La realización del proyecto durante mi beca, en otra ciudad ha sido difícil de compaginar con las asignaturas que estaban pendientes este año en la universidad. Ha sido realmente duro, y el sacrificio muy grande, pero escribiendo estas líneas no dejo de pensar que ha merecido la pena, y que no lo cambiaría.

# Apéndices

# Apéndice A
# Publicación del código

La publicación del código se realiza a través de la licencia *AGPL* que se puede encontrar en la página web:

`http://www.gnu.org/licenses/agpl.html`.

El código desarrollado para Claudia se encuentra en el repositorio de *GitHub*:

`https://github.com/StratusLab/claudia`

El Paas Manager se encuentra en el repositorio privado de Subversión que me muestra a continuación:

`https://barricada.hi.inet/repositorio/boi/europ-iaas/paas-manager/`

Este código se liberará en cuestión de semanas desde la escritura de esta memoria, a través de una migración a *GitHub*.

# Apéndice B
# Planificación del Proyecto

Este proyecto comenzó a desarrollarse en verano de 2012, hasta la primavera de 2013, es decir, desde mediados del mes de agosto hasta primeros del mes de mayo.

El PFC es el resultado de una beca Telefónica I+D, donde se marcaron los objetivos y la planificación del mismo. El desarrollo de software fue realizado hasta Enero de 2013. En las siguientes secciones se explicará con más detalle la realización del proyecto, atendiendo a los componentes utilizados en cada fase con más peso.

## B.1. Claudia y Sistemas de Monitorización (Agosto-Septiembre de 2012)

Durante los dos primeros meses del proyecto el trabajo se centró inicialmente en la familiarización con *Claudia* y los *Sistemas de monitorización* y realizar las acciones necesarias para la escalabilidad.

### B.1.1. Claudia

*Claudia*, al igual que el resto de componentes del proyecto *4CaaSt* ya estaban desarrollados antes del inicio de este PFC. No obstante, era necesario desarrollar todo el software adicional que permitiera la escalabilidad de aplicaciones en un *cloud* PaaS.

El desarrollo consistió en el acceso a los metadatos de las máquinas virtuales desplegadas en los proveedores *cloud* desde *Claudia*. Es necesario que la propia VM almacenara su FQN, por lo que desde *Claudia* debía introducirse en los metadatos y la propia máquina, a través de un script que se ejecutara en el arranque, guardar dicho FQN en un fichero accesible.

### B.1.2. Sistemas de monitorización

Las máquinas debían estar monitorizadas de forma externa para conocer su rendimiento, y en función de éste realizar las acciones oportunas de escalado. Por tanto fue necesario crear una imagen base que tuviera instalado el demonio *Collectd* y que además se encargara de indicar a los sistemas de monitorización que debían recoger sus datos. Esta interacción se desarrolla también en un *script* que se ejecuta en el

arranque, y que realiza una petición a la base de datos de los sistemas de monitorización.

En la imagen creada, también se instala un agente *Chef*, que permitirá la instalación de software.

## B.2. Paas Manager (Octubre 2012-Enero 2013)

Esta fase ha sido la que mayor carga de trabajo ha supuesto. En el mes de octubre fue muy importante la familiarización y conocimiento exhaustivo del desarrollo del componente *Paas Manager* que ya había sido realizada, a través del código y de pruebas con casos de uso para comprenderlo exactamente. Este proceso llevó consigo además, la creación de imágenes nuevas basadas en la creada en la fase anterior que permitieran desplegar aplicaciones sin utilizar los *Sistemas de Gestión de Instalación Software*.

Posteriormente, se desarrollaron los métodos necesarios para la gestión de la monitorización de las máquinas virtuales desde el *Paas Manager*, siendo este componente el que controlara que VMs debían ser monitorizadas y en qué momento deben dejar de serlo. Fue necesaria la creación de una nueva imagen base que permitiera la instalación de software, basada en la realizada en la fase anterior, pero que no realizara peticiones a los sistemas de monitorización en lo que al inicio de la recogida de datos se refiere.

El desarrollo de todos los métodos necesarios para la escalabilidad fue a continuación. Fue necesaria la modificación de algunos modelos de datos para el correcto funcionamiento del software, y realizar acciones de persistencia de datos que permitieran la escalabilidad, además de la creación de un *API REST* y del desarrollo de software en sí.

En *Claudia* se crearon las acciones a realizar para la escalabilidad, permitiendo que el *Paas Manager* gestionara el aprovisionamiento o retirada de recursos de los *entornos*.

## B.3. Memoria y últimos detalles (Febrero 2013 - Mayo 2013)

En esta última fase principalmente la carga de trabajo recayó sobre la escritura de la presente memoria.

Además, bien por pequeños *bugs* encontrados, bien por el cambio de algunas especificaciones para otros desarrollos que no afectan a este PFC pero sí a la escalabilidad de determinados *entornos*, ha sido necesario realizar cambios en el desarrollo del software.

Quedan ciertos aspectos a concluir a lo largo del mes de mayo de 2013 antes de la presentación:

- Conversión de la presente memoria al formato de libro electrónico *epub*.
- Elaboración de la presentación.

Con esto, el proyecto quedaría terminado para el día de la presentación ante el tribunal, prevista para el mes de junio de 2013.

# Apéndice C
# Licencia Creative Commons

CREATIVE COMMONS CORPORATION NO ES UN DESPACHO DE ABOGADOS Y NO PROPORCIONA SERVICIOS JURÍDICOS. LA DISTRIBUCIÓN DE ESTA LICENCIA NO CREA UNA RELACIÓN ABOGADO-CLIENTE. CREATIVE COMMONS PROPORCIONA ESTA INFORMACIÓN TAL CUAL (ON AN AS-IS BASIS). CREATIVE COMMONS NO OFRECE GARANTÍA ALGUNA RESPECTO DE LA INFORMACIÓN PROPORCIONADA, NI ASUME RESPONSABILIDAD ALGUNA POR DAÑOS PRODUCIDOS A CONSECUENCIA DE SU USO.

**Licencia**

LA OBRA O LA PRESTACIÓN (SEGÚN SE DEFINEN MÁS ADELANTE) SE PROPORCIONA BAJO LOS TÉRMINOS DE ESTA LICENCIA PÚBLICA DE CREATIVE COMMONS (CCPL O LICENCIA). LA OBRA O LA PRESTACIÓN SE ENCUENTRA PROTEGIDA POR LA LEY ESPAÑOLA DE PROPIEDAD INTELECTUAL Y/O CUALESQUIERA OTRAS NORMAS QUE RESULTEN DE APLICACIÓN. QUEDA PROHIBIDO CUALQUIER USO DE LA OBRA O PRESTACIÓN DIFERENTE A LO AUTORIZADO BAJO ESTA LICENCIA O LO DISPUESTO EN LA LEY DE PROPIEDAD INTELECTUAL.

MEDIANTE EL EJERCICIO DE CUALQUIER DERECHO SOBRE LA OBRA O LA PRESTACIÓN, USTED ACEPTA Y CONSIENTE LAS LIMITACIONES Y OBLIGACIONES DE ESTA LICENCIA, SIN PERJUICIO DE LA NECESIDAD DE CONSENTIMIENTO EXPRESO EN CASO DE VIOLACIÓN PREVIA DE LOS TÉRMINOS DE LA MISMA. EL LICENCIADOR LE CONCEDE LOS DERECHOS CONTENIDOS EN ESTA LICENCIA, SIEMPRE QUE USTED ACEPTE LOS PRESENTES TÉRMINOS Y CONDICIONES.

## C.1. Definiciones

a. La ***obra*** es la creación literaria, artística o científica ofrecida bajo los términos de esta licencia.

b. En esta licencia se considera una ***prestación*** cualquier interpretación, ejecución,

fonograma, grabación audiovisual, emisión o transmisión, mera fotografía u otros objetos protegidos por la legislación de propiedad intelectual vigente aplicable.

c. La aplicación de esta licencia a una ***colección*** (definida más adelante) afectará únicamente a su estructura en cuanto forma de expresión de la selección o disposición de sus contenidos, no siendo extensiva a éstos. En este caso la colección tendrá la consideración de obra a efectos de esta licencia.

d. El ***titular originario*** es:

   i. En el caso de una obra literaria, artística o científica, la persona natural o grupo de personas que creó la obra.

   ii. En el caso de una obra colectiva, la persona que la edite y divulgue bajo su nombre, salvo pacto contrario.

   iii. En el caso de una interpretación o ejecución, el actor, cantante, músico, o cualquier otra persona que represente, cante, lea, recite, interprete o ejecute en cualquier forma una obra.

   iv. En el caso de un fonograma, el productor fonográfico, es decir, la persona natural o jurídica bajo cuya iniciativa y responsabilidad se realiza por primera vez una fijación exclusivamente sonora de la ejecución de una obra o de otros sonidos.

   v. En el caso de una grabación audiovisual, el productor de la grabación, es decir, la persona natural o jurídica que tenga la iniciativa y asuma la responsabilidad de las fijaciones de un plano o secuencia de imágenes, con o sin sonido.

   vi. En el caso de una emisión o una transmisión, la entidad de radiodifusión.

   vii. En el caso de una mera fotografía, aquella persona que la haya realizado.

   viii. En el caso de otros objetos protegidos por la legislación de propiedad intelectual vigente, la persona que ésta señale.

e. Se considerarán ***obras derivadas*** aquellas obras creadas a partir de la licenciada, como por ejemplo: las traducciones y adaptaciones; las revisiones, actualizaciones y anotaciones; los compendios, resúmenes y extractos; los arreglos musicales y, en general, cualesquiera transformaciones de una obra literaria, artística o científica. Para evitar la duda, si la obra consiste en una composición musical o grabación de sonidos, la sincronización temporal de la obra con una imagen en movimiento (synching) será considerada como una obra derivada a efectos de esta licencia.

f. Tendrán la consideración de ***colecciones*** la recopilación de obras ajenas, de datos o de otros elementos independientes como las antologías y las bases de datos que por la selección o disposición de sus contenidos constituyan creaciones intelectuales. La mera incorporación de una obra en una colección no dará lugar a una derivada a efectos de esta licencia.

g. El ***licenciador*** es la persona o la entidad que ofrece la obra o prestación bajo los términos de esta licencia y le concede los derechos de explotación de la misma conforme a lo dispuesto en ella.

h. ***Usted*** es la persona o la entidad que ejercita los derechos concedidos mediante esta licencia y que no ha violado previamente los términos de la misma con respecto a la obra o la prestación, o que ha recibido el permiso expreso del licenciador de ejercitar los derechos concedidos mediante esta licencia a pesar de una violación anterior.

i. La ***transformación*** de una obra comprende su traducción, adaptación y cualquier otra modificación en su forma de la que se derive una obra diferente. La creación resultante de la transformación de una obra tendrá la consideración de obra derivada.

j. Se entiende por ***reproducción*** la fijación directa o indirecta, provisional o permanente, por cualquier medio y en cualquier forma, de toda la obra o la prestación o de parte de ella, que permita su comunicación o la obtención de copias.

k. Se entiende por ***distribución*** la puesta a disposición del público del original o de las copias de la obra o la prestación, en un soporte tangible, mediante su venta, alquiler, préstamo o de cualquier otra forma.

l. Se entiende por ***comunicación pública*** todo acto por el cual una pluralidad de personas, que no pertenezcan al ámbito doméstico de quien la lleva a cabo, pueda tener acceso a la obra o la prestación sin previa distribución de ejemplares a cada una de ellas. Se considera comunicación pública la puesta a disposición del público de obras o prestaciones por procedimientos alámbricos o inalámbricos, de tal forma que cualquier persona pueda acceder a ellas desde el lugar y en el momento que elija.

m. La ***explotación*** de la obra o la prestación comprende la reproducción, la distribución, la comunicación pública y, en su caso, la transformación.

n. Los ***elementos de la licencia*** son las características principales de la licencia según la selección efectuada por el licenciador e indicadas en el título de esta licencia: Reconocimiento, CompartirIgual.

ñ. Una ***licencia equivalente*** es:

   i. Una versión posterior de esta licencia de Creative Commons con los mismos elementos de licencia.

   ii. La misma versión o una versión posterior de esta licencia de cualquier otra jurisdicción reconocida por Creative Commons con los mismos elementos de la licencia (ejemplo: Reconocimiento-CompartirIgual 3.0 Japón).

   iii. La misma versión o una versión posterior de la licencia de Creative Commons no adaptada a ninguna jurisdicción (*Unported*) con los mismos elementos de la licencia.

   iv. Una de las licencias compatibles que aparece en http://creativecommons.org/compatiblelicenses y que ha sido aprobada por Creative Commons como esencialmente equivalente a esta licencia porque, como mínimo:

      a. Contiene términos con el mismo propósito, el mismo significado y el mismo efecto que los elementos de esta licencia.

b. Permite explícitamente que las obras derivadas de obras sujetas a ella puedan ser distribuidas mediante esta licencia, la licencia de Creative Commons no adaptada a ninguna jurisdicción (*Unported*) o una licencia de cualquier otra jurisdicción reconocida por Creative Commons, con sus mismos elementos de licencia.

## C.2. Límites de los derechos

Nada en esta licencia pretende reducir o restringir cualesquiera límites legales de los derechos exclusivos del titular de los derechos de propiedad intelectual de acuerdo con la Ley de propiedad intelectual o cualesquiera otras leyes aplicables, ya sean derivados de usos legítimos, tales como la copia privada o la cita, u otras limitaciones como la resultante de la primera venta de ejemplares (agotamiento).

## C.3. Concesión de licencia

Conforme a los términos y a las condiciones de esta licencia, el licenciador concede, por el plazo de protección de los derechos de propiedad intelectual y a título gratuito, una licencia de ámbito mundial no exclusiva que incluye los derechos siguientes:

a. Derecho de reproducción, distribución y comunicación pública de la obra o la prestación.

b. Derecho a incorporar la obra o la prestación en una o más colecciones.

c. Derecho de reproducción, distribución y comunicación pública de la obra o la prestación lícitamente incorporada en una colección.

d. Derecho de transformación de la obra para crear una obra derivada siempre y cuando se incluya en ésta una indicación de la transformación o modificación efectuada.

e. Derecho de reproducción, distribución y comunicación pública de obras derivadas creadas a partir de la obra licenciada.

f. Derecho a extraer y reutilizar la obra o la prestación de una base de datos.

g. Para evitar cualquier duda, el titular originario:

   i. Conserva el derecho a percibir las remuneraciones o compensaciones previstas por actos de explotación de la obra o prestación, calificadas por la ley como irrenunciables e inalienables y sujetas a gestión colectiva obligatoria.

   ii. Renuncia al derecho exclusivo a percibir, tanto individualmente como mediante una entidad de gestión colectiva de derechos, cualquier remuneración derivada de actos de explotación de la obra o prestación que usted realice.

Estos derechos se pueden ejercitar en todos los medios y formatos, tangibles o intangibles, conocidos en el momento de la concesión de esta licencia. Los derechos mencionados incluyen el derecho a efectuar las modificaciones que sean precisas técnicamente para el ejercicio de los derechos en otros medios y formatos. Todos los derechos no concedidos

expresamente por el licenciador quedan reservados, incluyendo, a título enunciativo pero no limitativo, los derechos morales irrenunciables reconocidos por la ley aplicable. En la medida en que el licenciador ostente derechos exclusivos previstos por la ley nacional vigente que implementa la directiva europea en materia de derecho sui generis sobre bases de datos, renuncia expresamente a dichos derechos exclusivos.

## C.4. Restricciones

La concesión de derechos que supone esta licencia se encuentra sujeta y limitada a las restricciones siguientes:

a. Usted puede reproducir, distribuir o comunicar públicamente la obra o prestación solamente bajo los términos de esta licencia y debe incluir una copia de la misma, o su Identificador Uniforme de Recurso (URI). Usted no puede ofrecer o imponer ninguna condición sobre la obra o prestación que altere o restrinja los términos de esta licencia o el ejercicio de sus derechos por parte de los concesionarios de la misma. Usted no puede sublicenciar la obra o prestación. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías. Usted no puede reproducir, distribuir o comunicar públicamente la obra o prestación con medidas tecnológicas que controlen el acceso o el uso de una manera contraria a los términos de esta licencia. Esta sección 4.a también afecta a la obra o prestación incorporada en una colección, pero ello no implica que ésta en su conjunto quede automáticamente o deba quedar sujeta a los términos de la misma. En el caso que le sea requerido, previa comunicación del licenciador, si usted incorpora la obra en una colección y/o crea una obra derivada, deberá quitar cualquier crédito requerido en el apartado 4.c, en la medida de lo posible.

b. Usted puede distribuir o comunicar públicamente una obra derivada en el sentido de esta licencia solamente bajo los términos de la misma u otra licencia equivalente. Si usted utiliza esta misma licencia debe incluir una copia o bien su URI, con cada obra derivada que usted distribuya o comunique públicamente. Usted no puede ofrecer o imponer ningún término respecto a la obra derivada que altere o restrinja los términos de esta licencia o el ejercicio de sus derechos por parte de los concesionarios de la misma. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías cuando distribuya o comunique públicamente la obra derivada. Usted no puede ofrecer o imponer ningún término respecto de las obras derivadas o sus transformaciones que alteren o restrinjan los términos de esta licencia o el ejercicio de sus derechos por parte de los concesionarios de la misma. Usted no puede reproducir, distribuir o comunicar públicamente la obra derivada con medidas tecnológicas que controlen el acceso o uso de la obra de una manera contraria a los términos de esta licencia. Si utiliza una licencia equivalente debe cumplir con los requisitos que ésta establezca cuando distribuya o comunique públicamente la obra derivada. Todas estas condiciones se aplican a una obra derivada en tanto que incorporada a una colección, pero no implica que ésta tenga que estar sujeta a los términos de esta licencia.

c. Si usted reproduce, distribuye o comunica públicamente la obra o la prestación, una colección que la incorpore o cualquier obra derivada, debe mantener intactos todos los avisos sobre la propiedad intelectual e indicar, de manera razonable conforme al medio o a los medios que usted esté utilizando:

   i. El nombre del autor original, o el seudónimo si es el caso, así como el del titular originario, si le es facilitado.

   ii. El nombre de aquellas partes (por ejemplo: institución, publicación, revista) que el titular originario y/o el licenciador designen para ser reconocidos en el aviso legal, las condiciones de uso, o de cualquier otra manera razonable.

   iii. El título de la obra o la prestación si le es facilitado.

   iv. El URI, si existe, que el licenciador especifique para ser vinculado a la obra o la prestación, a menos que tal URI no se refiera al aviso legal o a la información sobre la licencia de la obra o la prestación.

   v. En el caso de una obra derivada, un aviso que identifique la transformación de la obra en la obra derivada (p. ej., "traducción castellana de la obra de Autor Original,.º "guión basado en obra original de Autor Original").

   Este reconocimiento debe hacerse de manera razonable. En el caso de una obra derivada o incorporación en una colección estos créditos deberán aparecer como mínimo en el mismo lugar donde se hallen los correspondientes a otros autores o titulares y de forma comparable a los mismos. Para evitar la duda, los créditos requeridos en esta sección sólo serán utilizados a efectos de atribución de la obra o la prestación en la manera especificada anteriormente. Sin un permiso previo por escrito, usted no puede afirmar ni dar a entender implícitamente ni explícitamente ninguna conexión, patrocinio o aprobación por parte del titular originario, el licenciador y/o las partes reconocidas hacia usted o hacia el uso que hace de la obra o la prestación.

d. Para evitar cualquier duda, debe hacerse notar que las restricciones anteriores (párrafos 4.a, 4.b y 4.c) no son de aplicación a aquellas partes de la obra o la prestación objeto de esta licencia que únicamente puedan ser protegidas mediante el derecho sui generis sobre bases de datos recogido por la ley nacional vigente implementando la directiva europea de bases de datos

## C.5. Exoneración de responsabilidad

A MENOS QUE SE ACUERDE MUTUAMENTE ENTRE LAS PARTES, EL LICENCIADOR OFRECE LA OBRA O LA PRESTACIÓN TAL CUAL (ON AN AS-IS BASIS) Y NO CONFIERE NINGUNA GARANTÍA DE CUALQUIER TIPO RESPECTO DE LA OBRA O LA PRESTACIÓN O DE LA PRESENCIA O AUSENCIA DE ERRORES QUE PUEDAN O NO SER DESCUBIERTOS. ALGUNAS JURISDICCIONES NO PERMITEN LA EXCLUSIÓN DE TALES GARANTÍAS, POR LO QUE TAL EXCLUSIÓN PUEDE NO SER DE APLICACIÓN A USTED.

## C.6. Limitación de responsabilidad

SALVO QUE LO DISPONGA EXPRESA E IMPERATIVAMENTE LA LEY APLICABLE, EN NINGÚN CASO EL LICENCIADOR SERÁ RESPONSABLE ANTE USTED POR CUALESQUIERA DAÑOS RESULTANTES, GENERALES O ESPECIALES (INCLUIDO EL DAÑO EMERGENTE Y EL LUCRO CESANTE), FORTUITOS O CAUSALES, DIRECTOS O INDIRECTOS, PRODUCIDOS EN CONEXIÓN CON ESTA LICENCIA O EL USO DE LA OBRA O LA PRESTACIÓN, INCLUSO SI EL LICENCIADOR HUBIERA SIDO INFORMADO DE LA POSIBILIDAD DE TALES DAÑOS.

## C.7. Finalización de la licencia

a. Esta licencia y la concesión de los derechos que contiene terminarán automáticamente en caso de cualquier incumplimiento de los términos de la misma. Las personas o entidades que hayan recibido de usted obras derivadas o colecciones bajo esta licencia, sin embargo, no verán sus licencias finalizadas, siempre que tales personas o entidades se mantengan en el cumplimiento íntegro de esta licencia. Las secciones 1, 2, 5, 6, 7 y 8 permanecerán vigentes pese a cualquier finalización de esta licencia.

b. Conforme a las condiciones y términos anteriores, la concesión de derechos de esta licencia es vigente por todo el plazo de protección de los derechos de propiedad intelectual según la ley aplicable. A pesar de lo anterior, el licenciador se reserva el derecho a divulgar o publicar la obra o la prestación en condiciones distintas a las presentes, o de retirar la obra o la prestación en cualquier momento. No obstante, ello no supondrá dar por concluida esta licencia (o cualquier otra licencia que haya sido concedida, o sea necesario ser concedida, bajo los términos de esta licencia), que continuará vigente y con efectos completos a no ser que haya finalizado conforme a lo establecido anteriormente, sin perjuicio del derecho moral de arrepentimiento en los términos reconocidos por la ley de propiedad intelectual aplicable.

## C.8. Miscelánea

a. Cada vez que usted realice cualquier tipo de explotación de la obra o la prestación, o de una colección que la incorpore, el licenciador ofrece a los terceros y sucesivos licenciatarios la concesión de derechos sobre la obra o la prestación en las mismas condiciones y términos que la licencia concedida a usted.

b. Cada vez que usted realice cualquier tipo de explotación de una obra derivada, el licenciador ofrece a los terceros y sucesivos licenciatarios la concesión de derechos sobre la obra objeto de esta licencia en las mismas condiciones y términos que la licencia concedida a usted.

c. Si alguna disposición de esta licencia resulta inválida o inaplicable según la Ley vigente, ello no afectará la validez o aplicabilidad del resto de los términos de esta licencia y, sin ninguna acción adicional por cualquiera las partes de este acuerdo, tal disposición se entenderá reformada en lo estrictamente necesario para hacer que tal disposición sea válida y ejecutiva.

d. No se entenderá que existe renuncia respecto de algún término o disposición de esta licencia, ni que se consiente violación alguna de la misma, a menos que tal renuncia o consentimiento figure por escrito y lleve la firma de la parte que renuncie o consienta.

e. Esta licencia constituye el acuerdo pleno entre las partes con respecto a la obra o la prestación objeto de la licencia. No caben interpretaciones, acuerdos o condiciones con respecto a la obra o la prestación que no se encuentren expresamente especificados en la presente licencia. El licenciador no estará obligado por ninguna disposición complementaria que pueda aparecer en cualquier comunicación que le haga llegar usted. Esta licencia no se puede modificar sin el mutuo acuerdo por escrito entre el licenciador y usted.

## C.9. Aviso de Creative Commons

Creative Commons no es parte de esta licencia, y no ofrece ninguna garantía en relación con la obra o la prestación. Creative Commons no será responsable frente a usted o a cualquier parte, por cualesquiera daños resultantes, incluyendo, pero no limitado, daños generales o especiales (incluido el daño emergente y el lucro cesante), fortuitos o causales, en conexión con esta licencia. A pesar de las dos (2) oraciones anteriores, si Creative Commons se ha identificado expresamente como el licenciador, tendrá todos los derechos y obligaciones del licenciador.

Salvo para el propósito limitado de indicar al público que la obra o la prestación está licenciada bajo la CCPL, ninguna parte utilizará la marca registrada Çreative Commons.º cualquier marca registrada o insignia relacionada con Çreative Commons"sin su consentimiento por escrito. Cualquier uso permitido se hará de conformidad con las pautas vigentes en cada momento sobre el uso de la marca registrada por Çreative Commons", en tanto que sean publicadas su sitio web (website) o sean proporcionadas a petición previa. Para evitar cualquier duda, estas restricciones en el uso de la marca no forman parte de esta licencia.

Puede contactar con Creative Commons en: `http://creativecommons.org/`.

# Glosario

**A**

**Acuerdo de nivel de servicio** Un acuerdo de nivel de servicio o *Service Level Agreement*, es un contrato escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio [42].

**Application Programming Interface** Un interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos software ofrecidos para ser utilizado por otro software como una capa de abstracción. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general.

**B**

**Base de Datos** Una base de datos es una colección de información organizada. También se puede definir como un conjunto de datos pertenecientes a un mismo contexto, que se almacenan de forma persistente.

**Bean** Los *beans* son clases de Java, de entidad o de servicio. Cuado se declara un *bean*, se le pueden declarar propiedades. En Spring todos los *beans* son *singleton*, es decir, sólo se realiza una intancia de esa clase.

**BigTable** *BigTable* es un sistema de gestión de base de datos creado por Google, distribuido, de alta eficiencia y propietario. Comenzó a ser desarrollado a principios de 2004. *BigTable* almacena la información en tablas multidimensionales cuyas celdas están, en su mayoría, sin utilizar. Además, estas celdas disponen de versiones temporales de sus valores, con lo que se puede hacer un seguimiento de los valores que han tomado históricamente [43].

**Bug** Un *bug* es un error o fallo en un sistema de software que desencadena un resultado indeseado.

**C**

**Chef** *Chef* es una herramienta de gestión de la configuración de software. Utiliza un lenguaje de dominio específico basado en *Ruby* para la escritura de *recetas*, que se almacenan en *CookBooks*. *Chef* es una herramienta utiliza para configurar servicios en la nube o para simplificar la tarea de configurar los servidores internos de la empresa. *Chef* configura automáticamente los

ajustes de los sistemas operativos y los programas que se ejecutan dentro de las máquinas. Funciona utilizando un modelo cliente−servidor y se puede integrar con algunas plataformas de la nube [44].

**Collectd** *Collectd* es un demonio de *Unix* que se encarga de la recolección de datos sobre el rendimiento de los recursos hardware y software de una máquina, y es capaz de transferirlos. Los datos recogidos resultan útiles para mantener una visión general de los recursos disponibles. Es capaz de realizar estadísticas sobre las métricas obtenidas de una máquina [45].

**D**

**Driver** Un *driver* es un software informático que permite interactuar con otros realizando una abstracción del hardware y proporcionando una interfaz estándar para la comunicación.

**F**

**Framework** Un *framework* es una estructura de tecnología para un soporte definido que se compone de software concreto, para facilitar el desarrollo de otro proyecto de software.

**Fully Qualified Name** El FQN es un nombre que caracteriza a una máquina virtual de forma inequívoca.

**G**

**Grid** *Grid* es un modelo para resolver problemas de computación masiva utilizando un gran número de ordenadores organizados como una maya en un sistema distribuido. De ahí que su nombre, en castellano red o poder de la red.

**H**

**Hipervisor** Un hipervisor permite que diferentes sistemas operativos, tareas y configuraciones de software coexistan en una misma máquina física. El hipervisor abstrae los recursos físicos de la máquina anfitriona para las distintas máquinas virtuales. Garantizan un nivel de aislamiento entre los invitados. También proporcionan una interfaz única para el hardware [46].

**I**

**Imágenes** Una imagen de disco es un archivo que contiene la estructura y los contenidos de un dispositivo. Contiene una copia completa del orginal, de forma que este puede ser replciado.

**Infraestructura Física** La infraestructura física engloba los recursos hardware necesarios para el almacenamiento, computación y comunicaciones.

**K**

**Key Performance Indicators** Los indicadores clave de desempeño son métricas, utilizadas para cuantificar objetivos que reflejan el rendimiento de un sistema. Deben ser representativos, medibles, permitir la comparación con datos pasados, específicos y su obtención ha de poder llevarse a cabo a tiempo. A través de los KPIs se puede realizar un análisis completo de la situación de un sistema.

**M**

**Metadatos** Un grupo de metadatos se refiere a un grupo de datos, llamados recursos. Metadatos es un término que se refiere a datos sobre los propios datos. En los metadatos de una máquina virtual se encuentran todos los datos referentes a la propia máquina.

**Máquina Virtual** Software que simula una máquina y puede ejecutar programas como si se tratara de un sistema de computación físico.

**O**

**Open Source** *Open source* es el término con el que se conoce al software distribuido y desarrollado libremente.

**Open Virtualization Format** OVF es un estándar abierto para empaquetar y distribuir servicios virtualizados o de forma más general software a ejecutar en máquinas virtuales. El estándar describe un formato abierto, seguro, portable, eficiente y extensible para empaquetación y distribución de software a ejecutar en máquinas virtuales. El estándar OVF no está vinculado a ninguna arquitectura de procesador [47].

**P**

**Payload** Es la parte de los datos de una transmisión que corresponden al propósito fundamental, excluyendo la información que se encarga únicamente de la correcta realización de la transmisión, como pueden ser las cabeceras.

**Plugin** Un *plugin* es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

**S**

**Shell** El termino *shell* engloba a todos aquellos programas que proveen una interfaz de usuario para acceder a los servicios del sistema operativo. Estos pueden ser gráficos o de texto simple, dependiendo del tipo de interfaz que empleen.

**Simple Object Access Protocol** SOAP es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML [48].

**T**

**Tupla** Función finita que permite mapear objetos a través del nombre de algunas variables.

**U**

**Unix** *Unix* es un sistema operativo portable, multitarea y multiusuario desarrollado por los laboratorios BELL en 1969. Existen muchas familias dentro de *Unix* que han evolucionado de forma independiente, como es el sistema operativo *Linux*.

**V**

**Virtual Data Center** Un centro de datos virtual corresponde a la ubicación virtual donde se concentran los recursos necesarios para el procesamiento de la información de una organización.

# Bibliografía


[1] (Última consulta, Abril 2013) Aspectos profesionales: Protección de Datos, Cloud Computing y Sistemas de Gestión. [Online]. Available: http://www.aspectosprofesionales.info/2012/05/cloud-computing-y-proteccion-de-datos.html

[2] (Última consulta, Abril 2013) Cloud Computing Latinoamérica. [Online]. Available: http://www.cloudcomputingla.com/

[3] (Última consulta, Abril 2013) Escalabilidad y alta disponibilidad. [Online]. Available: http://blog.0balla.net/2012/04/escalabilidad-y-alta-disponibilidad

[4] (Última consulta, Mayo, 2013) Flexiscale Website. [Online]. Available: http://www.flexiscale.com

[5] (Última consulta, Abril, 2013) Manifesto for Agile Software Development. [Online]. Available: http://www.agilemanifesto.org/

[6] (Última consulta, Febrero, 2013) Git Website. [Online]. Available: http://git-scm.com

[7] (Última Consulta, Febrero, 2013) Morfeo Proyect. 4CaaSt. [Online]. Available: http://4caast.morfeo-project.org

[8] (Mayo, 2012) Cloud Computing. Retos y Oportunidades. [Online]. Available: http://www.ontsi.red.es/ontsi/sites/default/files/1-_estudio_cloud_computing_retos_y_oportunidades_vdef.pdf

[9] B. Stiller, T. Bocek, F. Hecht, G. Machado, P. Racz, and M. Waldburger, "Unidad didáctica 1. Introducción al cloud computing," Colegio de Ingenieros, Tech. Rep., Segunda Edición.

[10] (Última consulta, Abril 2013) Vmware website. [Online]. Available: http://www.vmware.com

[11] (Última consulta, Abril 2013) Amazon Web Services website. [Online]. Available: http://aws.amazon.com

[12] (Última consulta, Abril 2013) Google Cloud Platform website. [Online]. Available: https://cloud.google.com/

[13] (Última consulta, Abril 2013) Flexiant website. [Online]. Available: http://flexiant.com/

[14] (Última consulta, Abril 2013) AWS Elastic Beanstalk website. [Online]. Available: http://aws.amazon.com/es/elasticbeanstalk/

[15] (Última consulta, Abril 2013) Google App Engine website. [Online]. Available: https://developers.google.com/appengine/

[16] B. Wilder, *Cloud Architecture Patterns*. O'Reilly Media, 2012.

[17] D. M. Luis M. Vaquero, Fermín Galán, "Elastically Ruling the Cloud: Specifying Application's Behavior in Federated Clouds," TID, Tech. Rep., White Paper.

[18] R. B. Luis M. Vaquero, Luis Rodero-Merino, "Dynamically Scaling Applications in the Cloud," TID, Tech. Rep., White Paper.

[19] (Enero, 2010) Telefónica's TCloud API Specification. [Online]. Available: http://www.tid.es/files/doc/apis/TCloud_API_Spec_v0.9.pdf

[20] (Última consulta, Febrero, 2013) Extreme Programming: A gentle introduction. [Online]. Available: http://www.extremeprogramming.org

[21] (Última consulta, Febrero, 2013) Review Board Website. [Online]. Available: http://www.reviewboard.org/

[22] (Enero, 2000) Aprende java. [Online]. Available: http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

[23] (Febrero, 2003) Eclipse platform technical overview. [Online]. Available: http://eclipse.org/whitepapers/eclipse-overview.pdf

[24] (Última consulta, Febrero, 2013) JVM language summit 2012 sessions. [Online]. Available: http://www.oracle.com/technetwork/java/javase/community/jvmls2012-1840099.html

[25] (Última consulta, Febrero, 2013) Hibernate Overview. What is Object/Relational Mapping? [Online]. Available: http://www.hibernate.org/about/orm

[26] (Última consulta, Febrero, 2013) Articles about java technology. [Online]. Available: http://www.oracle.com/technetwork/articles/java/index.html

[27] (Última consulta, Abril, 2013) Hibernate. JBoos Comunity web site. [Online]. Available: http://www.hibernate.org

[28] (Última consulta, Abril, 2013) Inversion of Control Containers and the Dependency Injection pattern. [Online]. Available: http://martinfowler.com/articles/injection.html

[29] (Última consulta, Febrero, 2013) Spring Website. [Online]. Available: http://www.springsource.org

[30] (Última consulta, Febrero, 2013) Tutorial de SQL. [Online]. Available: http://www.desarrolloweb.com/manuales/9/

[31] (Última consulta, Febrero, 2013) PostgreSQL website. [Online]. Available: http://www.postgresql.org

[32] M. Clark, *Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps*. The Pragmatic Programmers, 2004.

[33] (Última consulta, Febrero, 2013) Maven Website. [Online]. Available: http://maven.apache.org/

[34] (Última consulta, Abril 2013) Rest vs Web Services. [Online]. Available: http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

[35] (Última consulta, Febrero, 2013) Jersey Website. [Online]. Available: http://jersey.java.net/

[36] L. Wingerd, *Practical Perforce*. O'Reilly Media, 2005.

[37] (Última consulta, Febrero, 2013) Subversion Website. [Online]. Available: http://subversion.apache.org

[38] (Última consulta, Febrero, 2013) TortoiseSVN Website. [Online]. Available: http://tortoisesvn.net

[39] (Última consulta, Febrero, 2013) OpenSSH Website. [Online]. Available: http://www.openssh.org

[40] (Última consulta, Febrero, 2013) WinSCP Website. [Online]. Available: http://winscp.net

[41] (Última consulta, Febrero, 2013) JMeter Website. [Online]. Available: http://jmeter.apache.org

[42] (Última consulta, Abril 2013) The Service Level Agreement. [Online]. Available: http://www.sla-zone.co.uk/

[43] (Última consulta, Abril 2013) Google App Engine website. [Online]. Available: http://google.dirson.com/post/1827-bigtable-sistema-almacenamiento-google

[44] (Última consulta, Abril, 2013) Chef Website. [Online]. Available: http://www.opscode.com/chef/

[45] (Última consulta, Abril, 2013) Collectd Website. [Online]. Available: http://collectd.org

[46] (Última consulta, Abril 2013) Soluciones Robustas de Virtualización con Software Libre. [Online]. Available: http://forge.morfeo-project.org/wiki/images/9/97/Virtualizacion.pdf

[47] (Última consulta, Mayo, 2013) The Open Virtual Machine Format. Whitepaper for OVF Specification. [Online]. Available: http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf

[48] E. Cerami, *Web services essentials: distributed applications with XML-RPG, SOAP, UDDI & WSDL.* O'Reilly Media, 2002.