Escuela Técnica Superior de Ingenieros
Industriales y de Telecomunicación

Universidad de Cantabria

*Trabajo Fin de Grado*

# DESPLIEGUE Y CONFIGURACIÓN DE UN BALANCEADOR DE CARGA PARA SERVICIOS THREDDS MEDIANTE FILOSOFÍA DEVOPS

(Deployment and configuration of a load balancer for THREDDS services using DevOps philosophy)

Para acceder al Título de

***Graduado en***
***Ingeniería de Tecnologías de Telecomunicación***

**Autor: Sonia Fernández Tejería**

**Julio - 2017**

UNIVERSIDAD DE CANTABRIA

E.T.S de Ingenieros Industriales y de Telecomunicación

## Graduado en Ingeniería de Tecnologías de Telecomunicacion

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

**Realizado por:** Sonia Fernández Tejería

**Director del TFG:** Antonio S. Cofiño González

**Título:** "Despliegue y configuración de un balanceador de carga para servicios THREDDS mediante filosofía DevOps"

**Title:** "Deployment and configuration of a load balancer for THREDDS services using DevOps philosophy"

**Presentado a examen el día:** 31 de Julio de 2017

para acceder al Título de

## Graduado en Ingeniería de Tecnologías de Telecomunicación

Composición del Tribunal:

Presidente (Apellidos, Nombre): Crespo Fidalgo, José Luis

Secretario (Apellidos, Nombre): Irastorza Teja, José Ángel

Vocal (Apellidos, Nombre): Fernández Ibañez, Tomás

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

VºBº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

UNIVERSIDAD DE CANTABRIA

# *Resumen*

## Grado en Ingeniería en Tecnologías de Telecomunicación

por *Sonia Fernández Tejería*

El entorno de análisis de datos JASMIN tiene una parte de su almacenamiento destinada a los proyectos de un grupo de científicos conocidos como *Group Workspace* (GWS). Normalmente, estos proyectos necesitan compartir sus datos con otras comunidades (públicamente o con un grupo restringido). Con objeto de que el resto de usuarios puedan acceder a esta información, es necesario, en primer lugar, descubrirlos. Por ello, tener catalogados los datos es una característica muy importante. Actualmente, las soluciones llevadas a cabo dentro de estos GWSs no resuelven de una manera eficiente estas funcionalidades. Además, las soluciones implementadas son diferentes para ambos, es decir, ofrecen una solución para el descubrimiento de los datos y otra para el acceso a los mismos.

Con el fin de eliminar esta dispersión en el proceso de la publicación de los datos, cabe destacar la necesidad de facilitar e integrar estas capacidades en un mismo marco y proceso. Esta es la razón por la cual, en este trabajo, se propone la utilización de THREDDS Data Server (TDS) como marco común. TDS es un servidor *web* que integra el servicio de catálogo de datos junto con la capacidad de acceso al dato.

Debido a que el volumen y concurrencia de los datos utilizados en este sector puede ser muy elevado, si en un momento dado un solo servidor *web* recibe un aumento considerable de peticiones, este puede llegar a convertirse en un cuello de botella. Por ello, con objeto de mantener el rendimiento y la calidad del servicio, se plantea la utilización de un sistema de balanceo de carga en el cual se integra el servicio TDS. Se trata de un servidor que distribuye la carga entrante entre un grupo de servidores que la procesan. Para llevar a cabo este sistema, se utilizan un servidor *reverse proxy* con el *software* Apache HTTP Server, y un clúster de servidores con el *software* Apache Tomcat.

Por otra parte, con objeto de optimizar el proceso de despliegue del sistema de balanceo de carga se sigue la filosofía *DevOps*. Esta proporciona una automatización en el proceso de despliegue *software*, permitiendo una reproducción exacta tanto en las fases de construcción, prueba y producción.

***Palabras clave* — THREDDS, DevOps, Ansible, Balanceo de carga, JASMIN, Reverse Proxy, Tomcat, Apache**

UNIVERSIDAD DE CANTABRIA

# *Abstract*

## Grado en Ingeniería en Tecnologías de Telecomunicación

por *Sonia Fernández Tejería*

Large projects are given access to an allocation of shared disk known as a Group Workspace (GWS) on JASMIN. Many projects usually want to share their data with other communities (publicly or to a restricted group). Firstly, the users need to discover the data in order to access them. Therefore, having the data cataloged is a very importante feature. Currently the solutions implemented don't solve these requirementes efficiently. Moreover, the solutions for both of them are differents, in other words, they implement a solution to discover the data and another one to access the content.

In order to eliminate this dispersion of publishing services processes, it highlights the need to facilitate and integrate all these capabilities in a single process and framework. This is why in this work the use of THREDDS Data Server (TDS) is been proposed as a proof of concept for that purpose. TDS is a web server that combines catalog services with integrated data-access capabilities.

Because the data volume and concurrency can be very large, if a web server handles an increase of request at one time, it could become a bottle-neck. Thus, in order to keep the performance and the quality of the service, this raises the use of a load balancing system where TDS service is implemented. This is a server that distributes the load among a group of servers that process the data. In order to implement the system, it is used a reverse proxy server with Apache HTTP Server as a software, and a server cluster with Apache Tomcat as a software.

Additionally, in order to optimize the deployment of the load balancing system, it follows the *DevOps* philosophy. It provides an automated software deployment process, allowing an exact replica in the built, testing and production process.

***Key words* — THREDDS, DevOps, Ansible, Load Balancing, JASMIN, Reverse Proxy, Tomcat, Apache**

UNIVERSIDAD DE CANTABRIA

# *Estructura del Trabajo*

## Grado en Ingeniería en Tecnologías de Telecomunicación

por *Sonia Fernández Tejería*

Este trabajo puede dividirse en dos partes principales: desarrollo teórico y desarrollo práctico. Dentro del desarrollo teórico existen, a su vez, dos partes bien diferenciadas. Por un lado, se encuentra la explicación de la parte técnica de este trabajo, relacionada con los sistemas de balanceo de carga. Por el otro, la segunda parte se basa en el desarrollo teórico de la parte funcional de este trabajo, es decir, la implementación del servicio THREDDS. Por otro lado, el desarrollo práctico recoge una descripción sobre el despligue del sistema de balanceo de carga, así como del servicio THREDDS sobre dicho sistema. Además se realiza una breve explicación teórica sobre la filosofía *DevOps*, mencionada en el título de este trabajo. Si bien esta parte se trata del desarrollo práctico, la explicación teórica sobre qué es *DevOps* cobra sentido en esta parte debido a su relación con el proceso de despligue del sistema. Además, este trabajo lo inician y finalizan los capítulos correspondientes a la introducción y líneas futuras, respectivamente. Así mismo, se describe a continuación el contenido principal presente en cada uno de los capítulos de este trabajo:

- **Capítulo 1**
  El primer capítulo se corresponde con la introducción del trabajo. En él, se lleva a cabo una breve contextualización de los principales problemas en torno a los cuales gira este trabajo: implementación de un sistema de balanceo de carga y despliegue del servicio THREDDS. Además, se describen los objetivos del mismo.
- **Capítulo** 2
  El segundo capítulo aborda la descripción detallada de los sistemas de balanceo de carga. En este, se realizará un análisis sobre las diferentes tecnologías existentes para llevar a cabo un sistema de balanceo de carga, detallando cuales son sus ventajas e inconvenientes. De esta manera, se llegará a la elección de una de estas tecnologías, la cual será implementada en este trabajo.
- **Capítulo 3**
  El tercer capítulo trata de forma detallada la justificación de la implementación del servicio THREDDS en este trabajo. Con el fin de llegar a entender esta justificación, resulta necesario conocer quién será el destinatario de este servicio. De esta forma, como se verá en este capítulo, los usuarios a los que este servicio es ofrecido pertenecen a una comunidad denominada JASMIN. Por esta razón, también se abordará el concepto JASMIN y su infraestructura, así como algunos conceptos teóricos acerca de la computación *cloud*.

- **Capítulo 4**
  En este capítulo se comienza con la parte práctica. En este trabajo, el desarrollo práctico está relacionado con el despliegue, tanto del sistema de balanceo de carga, como del servicio THREDDS sobre dicho sistema. Además, se describe como este sistema se integra dentro de la infraestructura JASMIN, descrita en el tercer capítulo. Con el fin de llevar a cabo el despliegue, se ha utiliza una herramienta conocida como *Ansible*. Dicha herramienta forma parte de la filosofía *DevOps*, motivo por el cual, en este capítulo se realiza una descripción teórica sobre dicha filosofía.
- **Capítulo 5**
  El quinto y último capítulo de este trabajo recoge los puntos planteados como líneas de futura exploración, de forma que su consecución proporcione continuidad a todo el trabajo experimental que ha sido llevado a cabo.

# *Agradecimientos*

En primer lugar me gustaría agradecer a mi director de trabajo Antonio, por la confianza y la ayuda que ha depositado en mí. Gracias por darme la oportunidad de vivir esta experiencia tan increíble y por haberme enseñado tanto, no solo en lo profesional sino en lo personal. Ha pasado un año desde que empezó todo esto y sin duda, será algo de lo que tendré un grato recuerdo.

Además, agradecer a todo el Grupo de Meteorología de Santander de la Universidad de Cantabria, tanto a los que estaban antes como a los de ahora, por el buen ambiente de trabajo y humano, así como también por ayudarme siempre en todo lo que he necesitado.

Por otro lado, no quería olvidarme de todos los compañeros que he tenido durante estos 5 meses en el CEDA, en especial, Phil y Ruth, por guiarme durante todo este tiempo, así como también a mis compañeros de despacho, Neil, Richard y William, con los cuales aprender su lengua y cultura ha resultado realmente divertido.

Por último, pero no menos importante, agradecer a mi familia y amigos por todo el apoyo y el cariño recibido desde la distancia, y como no, a ti Alberto, por ser la persona que mejor me entiende, contigo empezó todo hace 4 años y contigo termina.

MUCHAS GRACIAS

# Índice general

# Índice de figuras

# Índice de códigos

# Acrónimos

**ACK** ACKnowledgement

**AJP** Apache JServ Protocol

**ASIC** Application-Specific Integrated Circuit

**CDM** Common Data Model

**CEDA** Centre for Enviromental Data Archival

**CEMS** Climate and Enviromental Monitoring from Space

**CPU** Central Processing Unit

**DevOps** Development and Operations

**DNS** Domain Name System

**DSR** Direct Server Return

**FTP** File Transfer Protocol

**GWS** Group WorkSpace

**HPC** High Performance Computing

**HTTP** Hypertext Transfer Protocol

**IaC** Infrastructure as Code

**IaaS** Infrastructure as a Service

**ICI** Interface Control Information

**ICMP** Internet Control Message Protocol

**IDU** Interface Data Unit

**IP** Internet Protocol

**ISO** International Organization for Standardization

**IT** Information Technology

**JASMIN** Joint Analysis System Meeting Infrastructure Needs

**J2EE** Java platform, Enterprise Edition

**MAC** Media Access Control

**MAT** MAC Address Translation

**MTU** Maximum Transmission Unit

**NAT** Network Address Translation

**NCAS** National Centre for Atmospheric Science

**NERC** Natural Enviroment Research Council

**NIST** National Intitute of Standards and Technology

**OPeNDAP** Open-source Project for a Network Data Access Protocol

**OSI** Open System Interconnection

**PaaS** Platform as a Service

**PCI** Protocol Control Information

**PDU** Protocol Data Unit

**RAL** Rutherford Appleton Laboratory

**RAM** Random Access Memory

**SaaS** Software as a Service

**SDU** Service Data Unit

**SLB** Server Load Balancing

**SO** Sistema Operativo

**SSH** Secure SHell

**SSL** Secure Sockets Layer

**STFC** Science and Technology Facilities Council

**TAP** THREDDS Access Portal

**TCP** Transmission Control Protocol

**TDS** THREDDS Data Server

**THREDDS** THematic Real-time Earth Distributed Data Servers

**TLS** Transport Layer Security

**UDP** User Data Protocol

**URI** Uniform Resource Identifier

**URL** Uniform Resource Locator

**VIP** Virtual IP

**VM** Virtual Machine

**XML** eXtensible Markup Language

**YAML** Yet Another Markup Language

**WAR** Web Application Archive

# Capítulo 1

# Introducción

En este primer capítulo se realiza una introducción del trabajo. En él, se trata de contextualizar el tema principal sobre el cual se desarrollará el mismo: integración de técnicas de balanceo de carga ofreciendo el servicio TDS a las comunidades relacionadas con la meteorología, mediante la aplicación de la filosofía DevOps. Además, a esta parte le acompañará una sección dedicada a enumerar y detallar los principales objetivos de este trabajo.

## 1.1. Contexto

Con el paso del tiempo, las comunidades científicas en el campo de la meteorología, clima y medioambientales han incrementado su acceso a grandes volúmenes de datos. Además, cada comunidad ha utilizado sus propios estándares y herramientas con el fin de promover el acceso libre a sus datos. Con objeto de implementar dicho acceso, las estrategias llevadas a cabo han pasado desde, instalaciones basadas en *web* que permitían a los usuarios descargarse los datos que les resultasen más interesantes, hasta la entrega de dichos datos en tiempo real. También se han desarrollado alternativas cliente-servidor que permitían a los usuarios ejecutar aplicaciones en servidores remotos con objeto de acceder a los datos como si estuviesen almacenados en sus discos locales. A pesar de ello, ninguno de estos servicios ha permitido el descubrimiento de los datos. Por ello, los usuarios debían conocer a priori donde estaban alojados dichos datos permitiéndoles de esta manera acceder a ellos. La búsqueda, el acceso y la modificación de los datos supone, para los científicos de estas comunidades una pérdida de tiempo, ya que su objetivo principal es analizarlos. Por ello, estas comunidades necesitaban mejores herramientas y procedimientos para realizar este proceso de una manera más eficiente.

En este contexto, para hacer este proceso más eficiente, surge una herramienta que implementa todas estas funcionalidades, denominada THematic Real-time Earth Distributed Data Servers (THREDDS) Data Server (TDS). TDS proporciona una capa de servicio intermedia entre los usuarios de los datos y los proveedores de dichos datos. Para ello,

TDS aplica la tecnología de la biblioteca digital[1] al descubrimiento y uso de datos. De esta manera, TDS proporciona un servicio al usuario que le permite encontrar, acceder, visualizar y utilizar los conjuntos de datos relacionados con sus necesidades. Para ello, es necesario que los proveedores de datos escriban sus datos en alguno de los formatos estándar que TDS entiende. Además, el servicio TDS ofrece un acceso a los datos a través de diferentes protocolos, proporcionando un acceso más versátil y homogéneo.

Con objeto de proporcionar este servicio a dichas comunidades, el servicio TDS se puede desplegar en un simple servidor *web*. Sin embargo, con el rápido crecimiento tanto de la cantidad y el volumen de la información, como de los usuarios que acceden a dicha información, es necesario garantizar una calidad en el servicio que se le ofrece al usuario. Por este motivo, en lugar de utilizar técnicas tradicionales para la mejora de la calidad en el servicio (aumento CPU, almacenamiento, etc ...), se presenta la solución del balanceo de carga. Esta se trata de añadir más servidores, encargados de implementar el servicio TDS, los cuales no son accesibles directamente por los usuarios. Para ello, se despliega un servidor *web* que se encarga de recibir todas las peticiones de los usuarios y distribuirlas entre los servidores con el servicio TDS desplegado. De esta manera, se puede mejorar el rendimiento del sistema y, por consiguiente, mantener o incluso mejorar la calidad del servicio entregado al usuario.

En este trabajo se identifican las diferentes comunidades meteorológicas como un grupo de usuarios que trabajan en un mismo proyecto, denominados *Group Workspaces* (GWSs). Estos grupos forman tienen acceso a un entorno asociado de análisis de datos conocido como JASMIN, detallado en el *Capítulo 3*. En particular los GWS que se beneficiarán de este servició serán aquellos que tengan, entre sus objetivos, la provisión de datos a otros GWS o usuarios externos. El motivo de proporcionar el servicio TDS a los GWS no es otro que el hecho de haber realizado una estancia durante 5 meses en el centro de análisis de datos ambientales, también conocido como (CEDA), encargado de administrar el entorno de análisis JASMIN.

Aunque, a lo largo de este trabajo, se haga mención a los GWSs, este trabajo se ha desarrollado con un propósito general, con el fin de que el objetivo principal del trabajo (integración de técnicas de balanceo de carga ofreciendo el servicio TDS a las comunidades relacionadas con la meteorología, mediante la aplicación de la filosofía DevOps) pueda ser implementado en cualquier entorno.

Por ello, la utilización de la herramienta *Ansible* ha permitido que se pueda crear un código general, el cual pueda ser desplegado en cualquier sistema. Como se verá en el *Capítulo 4*, *Ansible* pertenece a la fase de despliegue de la filosofía *DevOps*. En esta, la infraestructura es tratada como código (IaC), es decir, se trata la configuración de cada infraestructura como el *software* de programación. De esta manera, permite a las máquinas, donde se despliega este código, gestionarse de manera programada. Por consiguiente, esto lo convierte en una infraestructura "elástica", es decir, escalable y replicable.

[1] Se trata de un sistema para el procesamiento, acceso y tratamiento técnico de la información digital. Dicha información se estructura mediante una colección de documentos digitales, sobre los cuales se pueden ofrecer otros servicios de valor añadido para el usuario final [1].

## 1.2. Objetivos

Una vez contextualizado el motivo de este trabajo, se define su objetivo como la integración de técnicas de balanceo de carga en la infraestructura JASMIN, mediante la aplicación de la filosofía *DevOps* ofreciendo el servicio TDS a los GWSs. Para la consecución de este objetivo se definen las siguientes tareas:

- Analizar y evaluar las distintas tecnologías de balanceo de carga utilizadas para aplicaciones sobre TCP/IP y HTTP.
- Desplegar y configurar un sistema de balanceo de carga para los diferentes servicios *web* ofrecidos por el servidor THREDDS.
- Desplegar tanto el sistema de balanceo de carga como el servicio THREDDS mediante la utilización de la filosofía *DevOps*.
- Integrar el sistema de balanceo de carga en una infraestructura *cloud* como JASMIN.

# Capítulo 2

# Estado del arte del balanceo de carga

Desde mediados del siglo XX, Internet ha experimentado un período de desarrollo de gran intensidad. El crecimiento de usuarios y tráfico de red de Internet requiere de unos requisitos más altos en el rendimiento y escalabilidad de los servicios web. Aunque hoy en día un solo servidor puede gestionar fácilmente los requisitos de procesamiento de los portales de Internet más populares, la capacidad de cualquier servidor es finita y, cuando ésta llega a su límite, es necesario mejorar el servidor de alguna manera [2].

Tradicionalmente, se utilizaban opciones como añadir más memoria RAM (*Random Access Memory*), añadir más procesadores, incrementar la memoria, etc. Sin embargo, estas soluciones, aparte de ser muy costosas, solo pueden escalar [1] hasta un punto determinado. Por ello, una aplicación web tiene que ser capaz de ejecutarse en múltiples servidores para poder aceptar un incremento de peticiones de los usuarios. En las aplicaciones de Intranet la escalabilidad no es un gran problema ya que el número de usuarios no es muy elevado. En contraposición, las aplicaciones de Internet sufren un incremento de la carga a medida que la disponibilidad del ancho de banda aumenta. En esta situación, se presenta la solución del balanceo de carga como la mejor opción para hacer frente a este problema.

El término balanceo de carga se define como el proceso y la tecnología de distribuir numerosas peticiones entre varios servidores utilizando dispositivos de red. Estos dispositivos interceptan el tráfico destinado de una página y lo redireccionan a varios servidores. La utilización de esta técnica evita la concentración de la carga en un solo servidor, siendo distribuida entre múltiples servidores, los cuales funcionan, desde el punto de vista del usuario, como un solo servidor, es decir, de manera transparente. El balanceo de

[1] Se define escalabilidad como la medida de la capacidad de crecimiento de un servicio o una aplicación para satisfacer demandas de rendimiento cada vez mayores. Cuando la carga de un sistema aumenta, este debe poder expandirse para poder satisfacer la demanda sin perder calidad de servicio [3].

carga debe alcanzar un equilibrio entre factores como el coste, el rendimiento y la escalabilidad, empleando para ello un conjunto de servidores a un precio relativamente bajo. Este equilibrio no se podría lograr utilizando un sistema monolítico.

A pesar de las ventajas que los balanceadores de carga presentan, el riesgo de fallo incrementa al mismo tiempo que crece el número de dispositivos utilizados. Por ello, es de gran importancia tener en cuenta la disponibilidad [2] de nuestro sistema. Aunque esta característica se suele asociar a los sistemas de balanceo de carga, el uso de algunas técnicas no proporciona una alta disponibilidad y esto puede resultar contraproducente.

La forma más sencilla de desplegar un sistema de balanceo de carga es asignar ciertos servidores a un grupo predefinido de usuarios. Aunque esta técnica se puede implementar para servicios en una intranet, no es así en los servicios de Internet.

En este capítulo se van a explicar las diferentes tecnologías existentes para implementar un sistema de balanceo de carga.

## 2.1. Conceptos relacionados con el balanceo de carga

Existe una terminología confusa e inconsistente en torno al mundo del balanceo de carga, ya que es una tecnología relativamente joven. Por ello, en esta sección se tratará de aclarar algunos términos que se utilizarán a lo largo de este trabajo.

### 2.1.1. Modelo OSI

El modelo de referencia OSI (*Open System Interconnection*) es un proyecto que fue desarrollado por el ISO (*International Organization for Standardization*) con el fin de proporcionar un marco y un conjunto de estándares de comunicación de datos para conectar dispositivos de diferentes tipos y arquitecturas de red. Este modelo sirve como esqueleto para la definición de estándares que permitan la interconexión de sistemas heterogéneos. Se trata de un modelo universal que define siete capas y las funciones que se realizan en cada una de ellas. Cada capa representa un nivel de abstracción separado e interactúa únicamente con sus capas adyacentes [4].

En la *Figura 2.1* se representa el funcionamiento del modelo OSI. La entidad de la capa N+1 pasa una unidad de datos de interfaz (IDU) a la entidad de la capa N a través de un punto de acceso al servicio de la capa N. La IDU consta de una unidad de datos de servicio (SDU) y de una información de control de la interfaz (ICI) en la que se envía la información a las capas inferiores. La SDU puede ser fragmentada por la entidad de capa N, dependiendo del tamaño máximo de unidad (MTU). A cada fragmento se le asignará una cabecera, llamada información de control de protocolo (PCI) y formará una unidad de datos de protocolo (PDU). Las cabeceras de las PDUs se utilizan para controlar el intercambio de información entre entidades pares (de la misma capa) [5].

[2] Se define disponibilidad como el porcentaje de tiempo que un sistema es capaz de realizar las funciones para las que está diseñado [3].

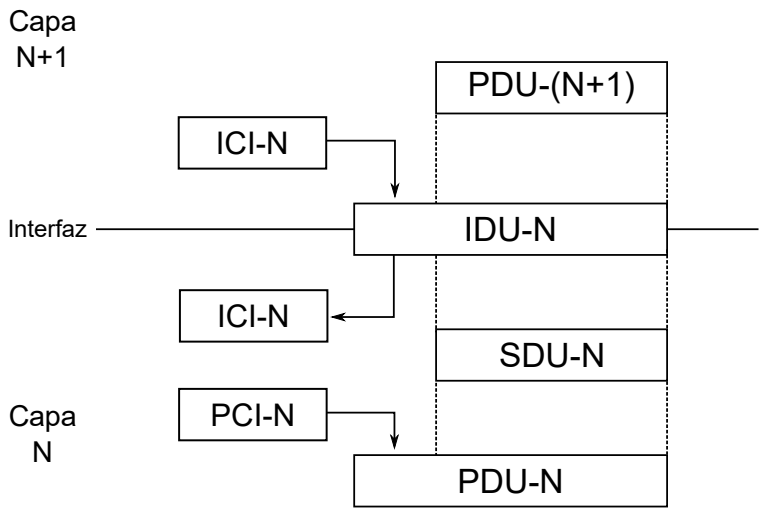



**Figura 2.1.** Intercambio de datos entre entidades según el modelo OSI

A continuación, se describe brevemente las funcionalidades de cada capa:

- **Capa 1**: también conocida como **nivel físico**. Es la capa más baja de la arquitectura OSI. Abarca la interfaz física entre dispositivos y las normas por las que los bits pasan de unos dispositivos a otros. Se ocupa de la transmisión de secuencias de bits a través del medio físico, detallando las características mecánicas, eléctricas, funcionales y de procedimiento de las interfaces de conexión al medio físico. (Protocolos: *V.x*, *X-21*, *RS-232*, *etc*)
- **Capa 2**: se conoce como el **nivel de enlace de datos**. Proporciona una transferencia segura de la información a través del enlace físico, así como un medio para activar, mantener y desactivar el enlace. Envía bloques de datos, denominados tramas (PDU del nivel de enlace), con la sincronización, control de errores y control de flujo necesarios. En esta capa se deben resolver los problemas causados por daño, pérdida o duplicidad de tramas. Los dispositivos de red que trabajan en esta capa son conocidos como bridges. (Protocolos: *Ethernet*, *HDLC*, *etc*)
- **Capa 3**: conocida como el **nivel de red**. Proporciona independencia a las capas superiores con respecto de la transmisión de datos y tecnologías de conmutación utilizadas para conectar los sistemas. Es responsable de establecer, mantener y terminar las conexiones, efectuando el encaminamiento de los mensajes desde el origen al destino a través de los nodos de la red. Este nivel no garantiza la conexión extremo a extremo, sino solo entre dispositivos adyacentes. Los dispositivos asociados a este nivel son los enrutadores, los cuales realizan el encaminamiento de forma inteligente, a diferencia de aquellos de nivel 2. El protocolo al que se va a hacer referencia en este trabajo es IP, el protocolo inter-red utilizado en la red de Internet. (Protocolos: *PLP*, *IP*, *etc*)
- **Capa 4**: a esta capa se la conoce como el **nivel de transporte**. A diferencia de la capa de red, este nivel proporciona un transporte de los datos extremo a extremo asegurando que las unidades de datos lleguen libres de error, ordenadas, sin pérdidas ni duplicados. Fragmenta los datos que llegan del nivel superior en unidades

más pequeñas y los pasa al nivel de red. En este nivel, se pueden multiplexar varias conexiones de transporte sobre la misma conexión de red o bien distribuir una conexión de transporte sobre varias conexiones de red. En la cabecera que añade este nivel se envía la información que identifica la conexión a la que pertenece cada mensaje (puerto). (Protocolos: *TCP*, *UDP*, *SPX*, *etc*)

- **Capa 5**: también llamado **nivel de sesión**. Proporciona una sincronización y gestión de los diálogos entre las entidades extremas. Establece, gestiona y termina sesiones entre aplicaciones. Se trata de una capa importante en el comercio digital ya que todos los datos de una sesión, como por ejemplo el carrito de compra en un servidor web, deben permanecer en un mismo servidor (persistencia) cuando nos referimos al uso del balanceo de carga, como se explicará más adelante. (Protocolos: *NetBIOS*, *RPC*, *NCP*)
- **Capa 6**: conocida como **nivel de presentación**. Recibe peticiones de servicio de la capa de aplicación y envía peticiones de servicio a la capa de sesión. Proporciona un lenguaje común que permite a los diferentes sistemas comunicarse los unos con los otros, aunque estos utilicen un lenguaje propio diferente (representación interna de caracteres), es decir, define la sintaxis utilizada entre las entidades de aplicación. (Protocolos: *ASN.1*, *MIME*, *XML*, *etc*)
- **Capa 7**: se la denomina **nivel de aplicación**. Proporciona un medio para que los procesos de aplicación accedan al entorno OSI. Contiene funciones de gestión y mecanismos útiles para soportar aplicaciones distribuidas. (Protocolos: *FTP*, *HTTP*, *SMTP*, *etc*)

### 2.1.2. Virtual IP

Virtual IP (VIP) es la instancia del balanceo de carga que se utiliza para que los usuarios accedan a una página web. Una VIP tiene que tener asociada una dirección IP real, la cual tiene que se accesible públicamente. Además, debe existir al menos un servidor real asignado a cada VIP. Normalmente suelen haber múltiples servidores reales de modo que la VIP distribuye el tráfico entre todos los servidores utilizando diferentes métricas y métodos [6].

### 2.1.3. Servidor

La RAE define el término servidor como: "*Unidad informática que proporciona diversos servicios a computadoras conectadas con ella a través de una red*" [7], es decir, se trata de un dispositivo que ejecuta un servicio que comparte la carga entre otros servicios.

### 2.1.4. Grupo

Se utilizará este término indistintamente junto con "*granja*", "*granja de servidores*" o "*clúster*", haciendo referencia al conjunto de servidores entre los cuales el balanceador distribuirá la carga.

### 2.1.5. Redundancia

Se puede utilizar el término redundancia en los casos en que, cuando un dispositivo falla, otro ocupará su lugar y funcionalidad, con un impacto muy pequeño o inexistente en las operaciones del sistema. Existen múltiples maneras de implementar esta funcionalidad [6].

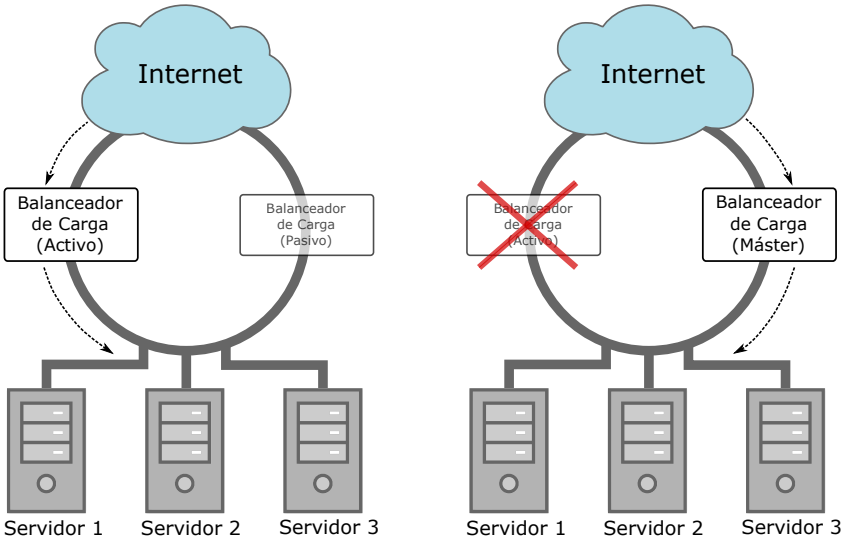
Normalmente, se suelen utilizar dos dispositivos, utilizando un protocolo en uno de ellos que controle el estado del otro. En algunos escenarios es posible que ambos dispositivos estén en funcionamiento y acepten tráfico, mientras que en otros solo se utiliza un dispositivo y el otro se encuentra inactivo a la espera de que el primero falle.

Sin embargo, en cualquiera de los escenarios, si un dispositivo deja de funcionar, todas las conexiones activas se resetean y los números de secuencia de la información se pierden.

#### Escenario Activo-Pasivo

Este escenario es el más fácil de entender e implementar. También es conocido como la relación "maestro/esclavo". En este, el maestro hace referencia al dispositivo activo y el esclavo al dispositivo pasivo. El primero de ellos es el encargado de recoger todo el tráfico mientras que el segundo está esperando a que el maestro falle. En caso de que esto suceda, el dispositivo pasivo pasaría a convertirse en el activo, aceptando todo el tráfico que tenía el otro servidor.

En la *Figura 2.2* se representa de forma gráfica el modo de funcionamiento de los servidores en este escenario: modo normal (*Figura 2.2(a)*) y con fallo del servidor maestro (*Figura 2.2(b)*).



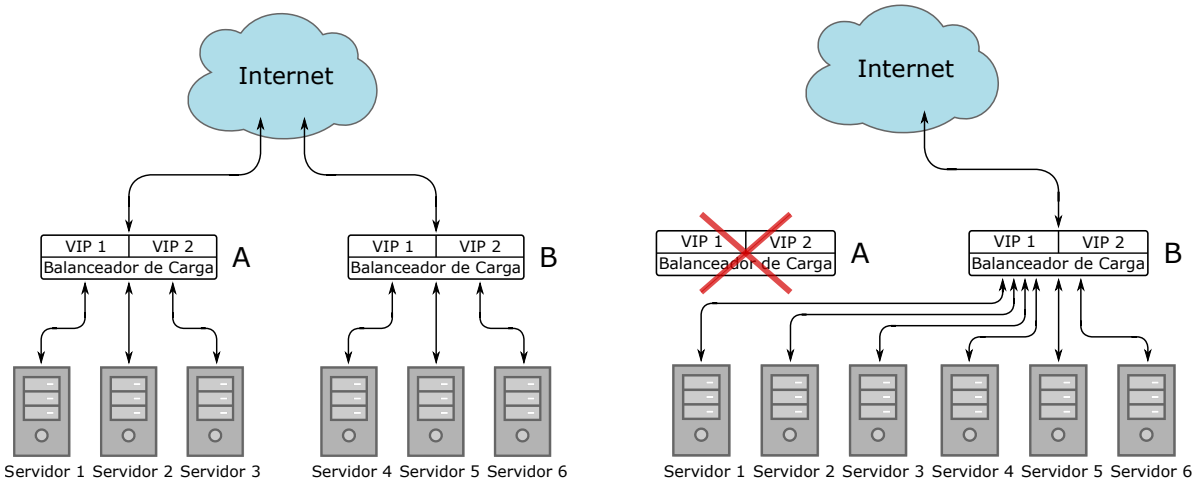

(a) Funcionamiento normal (b) Fallo del servidor maestro

**Figura 2.2.** Escenario Activo-Pasivo
**Fuente**: "Server load balancing" [6]

#### Escenario Activo-Activo

Este escenario sigue la relación "maestro/maestro". Existen múltiples variantes de este escenario, aunque en cualquier caso ambas unidades aceptan tráfico.

En el ejemplo de la *Figura 2.3* se crean dos VIPs que responden a ambos balanceadores de carga. Como muestra la *Figura 2.3(b)*, en el caso de que uno de los dos balanceadores falle , su VIP correspondiente sigue estando disponible hacia el usuario, aunque sea el otro balanceador el que responda a sus peticiones. Además, este balanceador tomará el control de todas las funcionalidades que tenía el balanceador que ha fallado.




(a) Funcionamiento normal (b) Fallo de un servidor maestro

**Figura 2.3.** Escenario Activo-Activo
**Fuente**: "Server load balancing" [6]

### 2.1.6. Persistencia de sesión

También conocido como sesión "pegajosa" (*sticky session*). La persistencia es el acto de mantener el tráfico de un usuario específico en el mismo servidor en el que comenzó la sesión. Esta característica es especialmente importante en las aplicaciones de comercio digital, como se comentó en el nivel de sesión del modelo OSI [8].

### 2.1.7. Comprobación del estado

Para seleccionar un servidor, el balanceador de carga necesita conocer cuáles de ellos están disponibles. Por ello, una de las tareas más importantes del balanceador de carga se basa en conocer el estado de los servidores a los cuales distribuye el tráfico para poder quitarlos del ciclo si estos se encuentran fuera de servicio. Este tipo de tareas son conocidas como "*revisiones de salud*". Se pueden implementar mediante el envío de un *ping*[3],

[3]Herramienta utilizada en redes de computación para comprobar el estado de la comunicación del *host* local con uno o varios equipos remotos por medio del envío de paquetes ICMP.

comprobación del puerto o una comprobación del contenido [8].

Comprobar el estado de los servidores resulta una tarea muy complicada, ya que las comprobaciones tienen que estar suficientemente espaciadas en el tiempo como para no sobrecargar los servidores, pero lo suficientemente cerca como para detectar rápidamente un servidor que ha dejado de funcionar. Por ello es bastante común implementar un balanceador de carga encargado de realizar las comprobaciones del estado de los servidores.

## 2.2. Evolución

Una de las primeras tecnologías utilizadas en el balanceo de carga fue la basada en *DNS Round Robin*, la cual se explicará en la *Subsección 2.3.1*. En vista de que esta solución no era capaz de manejar por sí solo los problemas de redundancia, escalabilidad y configuración, surgió otra técnica llamada balanceo de carga de servidor (SLB), explicada en las *Subsecciones 2.3.2*, *2.3.4*, *2.3.5 y 2.3.6* . Esta técnica puede ser implementada bien a través de un equipamiento hardware dedicado o bien a través de un desarrollo software [8], como se explicará en la *Sección 2.6*. El balanceador de carga realiza algunas modificaciones en los paquetes que se envían desde y hacia el usuario para distribuir la carga. Esta solución aporta varios beneficios, entre los cuales destacan tres principalmente [6]:

1. **Flexibilidad**: permite añadir y eliminar servidores de un lugar en cualquier momento con un efecto inmediato. Entre otras ventajas, esto permite realizar el mantenimiento de cualquier dispositivo, incluso en las horas más concurridas sin tener ningún impacto en el sistema.

2. **Alta disponibilidad**: los balanceadores de carga pueden comprobar el estado de los servidores, eliminar de los servidores disponibles cualquiera que se encuentre en mal estado, modificarlo y volver a ponerlo a disposición. Esto se puede hacer de manera automática, sin ningún tipo de intervención del administrador. Además, como se ha comentado al principio de este capítulo, desplegar nuevos componentes en el sistema incrementa el riesgo de fallos del mismo. Por lo que, con objeto de satisfacer la disponibilidad del sistema, es recomendable implementar un segundo servidor de balanceo de carga como respaldo del servidor principal.

3. **Escalabilidad**: sabiendo que los balanceadores distribuyen la carga entre varios servidores, el incremento del tráfico puede solventarse simplemente añadiendo más servidores a nuestro sistema. Cuando el tráfico en una página web incrementa, los servidores se pueden desplegar de inmediato, de manera dinámica, con objeto de satisfacer ese incremento en la carga.

## 2.3. Tecnologías existentes en el balanceo de carga

### 2.3.1. Balanceo de carga basado en DNS

Una de las primeras técnicas utilizadas para implementar el balanceo de carga se llama *DNS Round Robin*. Esta tecnología implementa la función del DNS (*Domain Name System*), la cual permite asociar múltiples direcciones IP a un mismo nombre de dominio. Cada entrada DNS mapea un nombre de dominio a una dirección IP [6].

Utilizando la tecnología DNS, se suele asociar una sola dirección IP a un nombre de dominio dado. Pero con el *DNS Round Robin*, se puede asociar múltiples direcciones IP a un mismo nombre de dominio, distribuyendo el tráfico entre las direcciones IP asociadas. Si un servidor DNS tiene numerosas entradas IP para un nombre de dominio dado, este devuelve todos ellos de manera ordenada. De esta manera, los usuarios verán diferentes direcciones IP para el mismo nombre de dominio, siendo capaces de llegar a los diferentes servidores.

Aunque esta técnica parezca una manera muy simple de distribuir el tráfico entre los servidores, presenta algunas limitaciones. Para poder comprenderlas resulta necesario explicar como funciona el DNS en sí mismo.

#### Funcionamiento DNS

Como se ha explicado anteriormente, el DNS asocia direcciones IP con nombres de dominio con el fin de que el usuario solo tenga que memorizar un nombre en vez de una combinación de números. Aun así, los dispositivos necesitan conocer la dirección IP para establecer las conexiones. Cada terminal conectado a Internet tiene uno o más servidores DNS configurados con objeto de realizar esta traducción.

En Internet, las direcciones IP son utilizadas para identificar a los diferentes terminales. A pesar de ello, las direcciones IP están formadas por una combinación de números expresadas en 4 octetos (IPv4) u 8 octetos (IPv6). Esto hace que la memorización de las direcciones sea realmente complicada. Por ello, se utilizan nombres de dominio para identificar a los terminales, resultando esto más fácil de recordar para las personas. Un sistema de DNS gestiona la relación entre los nombres de dominio y las direcciones IP, proporcionando una dirección IP en respuesta a una petición del cliente relacionado con un nombre de dominio.

A continuación, se describe paso por paso el proceso de resolución de nombres desde que el usuario realiza la consulta hasta que esta es resuelta (*Figura 2.4*):

**(1)** Cuando un usuario escribe una URL (*Uniform Resource Locator*) en el navegador, el Sistema Operativo (SO) envía una petición al servidor DNS local preguntándole por la dirección IP asociada a dicho nombre. Este servidor no suele tener dicha información almacenada, a no ser que el usuario haya realizado anteriormente la misma consulta, y esta se encuentre almacenada en la caché del servidor DNS.

**(2)** Por ello, el servidor busca el nombre de dominio en uno de los servidores raíz.

**(3)** Aunque los servidores raíz no tienen la información asociada a la dirección IP, saben quién tiene parte de esa información. Por este motivo, envían al usuario la información de a qué servidor debe preguntar.

**(4)** El usuario pregunta a dicho servidor, conocido como servidor DNS de dominio superior.

**(5)** Los servidores de dominio superior conocen a los servidores DNS de autoridad, por lo que reenvía al usuario a este servidor.

**(6)** Una vez se conoce el DNS de autoridad, el usuario reenvía la petición.

**(7)** El servidor DNS de autoridad devuelve al usuario la resolución de la consulta si este es el último nivel del nombre de dominio a resolver.

> *Los pasos 6 y 7 se realizarán tantas veces como sea necesario para llegar al servidor final ya que los nombres de dominio son estructuras jerárquicas, es decir, pueden haber tantos servidores DNS autorizados como nombres existan (en el ejemplo:* `www.dominio1.dominio2.es`*, el usuario tendrá que realizar 4 peticiones hasta llegar al servidor final (servidor raíz [*`.es`*], servidor de dominio superior* `.es` *[*`dominio2.es`*], servidor autorizado* `dominio2` *[*`dominio1.dominio2.es`*], y servidor autorizado* `dominio1` *[*`www.dominio1.dominio2.es`*] ).*

**(8)** Finalmente, el servidor DNS del usuario realiza la petición al servidor final, apareciendo la página web solicitada por el usuario en su pantalla.

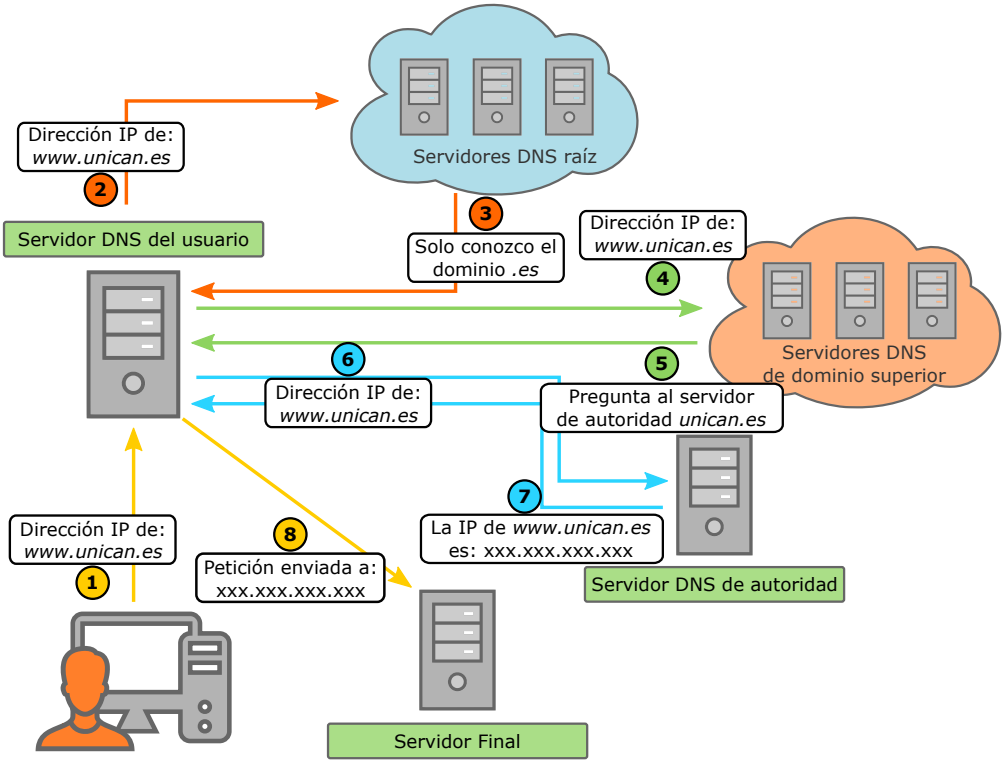



**Figura 2.4.** Diagrama de resolución de nombres de una petición realizada por un usuario

**Limitaciones**

Como se ha mencionado brevemente en el punto anterior, el DNS tiene una memoria caché donde almacena las direcciones IP de las peticiones recientes que ha realizado el usuario. Cuando el servidor DNS realiza una petición al servidor raíz y este le responde, el servidor DNS almacena esta información en su caché hasta que dicha entrada expire. Esta característica supone una de las limitaciones del método de *DNS Round Robin*.

Uno de los problemas causados por el almacenamiento de las entradas en el servidor es la distribución del tráfico. Utilizando la tecnología *DNS Round Robin*, una petición se debe distribuir entre todas las direcciones IP asociadas a un nombre de dominio. Teóricamente, si existiesen cuatro direcciones IP para un mismo nombre de dominio, la petición sería dividida en cuatro partes y distribuida por dichos servidores. Sin embargo, en la práctica puede ocurrir que el usuario realice una petición a su servidor de nombre configurado acerca de un nombre de dominio que ya está registrado en su caché, y alguna de las direcciones IP asociadas no se encuentre disponible ya que el servidor correspondiente está caído.

Cuando la demanda aumenta repentinamente, la falta de velocidad en la actualización del servidor DNS es otro problema. Las nuevas entradas en la caché del servidor DNS tardan un tiempo en propagarse, limitando esto la escalabilidad del sistema.

Por estas limitaciones, la técnica *DNS Round Robin* no se utiliza como solución por si sola, sino que se implementa junto con otras técnicas de balanceo de carga que se explicarán a continuación.

### 2.3.2. Enrutamiento directo

También conocido como DSR (*Direct Server Return*). Este método se basa en la omisión del balanceador de carga en la conexión de salida entre el usuario y el servidor final, es decir, el servidor devuelve la información directamente al usuario sin necesidad de pasar por el balanceador de carga. Para ello, es necesario que el clúster de servidores y el balanceador se encuentren en la misma red física. Este proceso se implementa mediante la traducción de la dirección MAC (MAT) [6].

La dirección MAC (*Media Access Control*) es la dirección *hardware* o física de las tarjetas de red de los dispositivos formada por un conjunto de 6 *bytes* expresada en formato hexadecimal. Esta dirección identifica inequívocamente a cada dispositivo, es decir, es única para cada uno de ellos y opera en la capa 2 del modelo OSI.

Para poder implementar esta tecnología es necesario configurar los servidores reales del clúster y el balanceador de carga con una dirección IP diferente a cada uno y una dirección VIP común. Como la dirección MAC es la encargada de dirigir los paquetes IP al dispositivo físico correcto, no se pueden tener dos dispositivos de una red con la misma dirección IP. Para hacer esto posible, en vez de asignarle la VIP a una interfaz de red, se le asigna a una interfaz de tipo *loopback*. Esta, es una interfaz lógica utilizada por el propio dispositivo para acceder a sus propios servicios independientemente de la configuración de red. La dirección universal de esta interfaz es la 127.0.0.1. Sin embargo, esta dirección

puede ser modificada, asignando la dirección que se crea conveniente. Teniendo la dirección VIP configurada en esta interfaz, se soluciona el problema tener dos dispositivos de la misma red con la misma dirección IP.

En la siguiente figura se representa el funcionamiento del balanceo de carga basado en DSR:

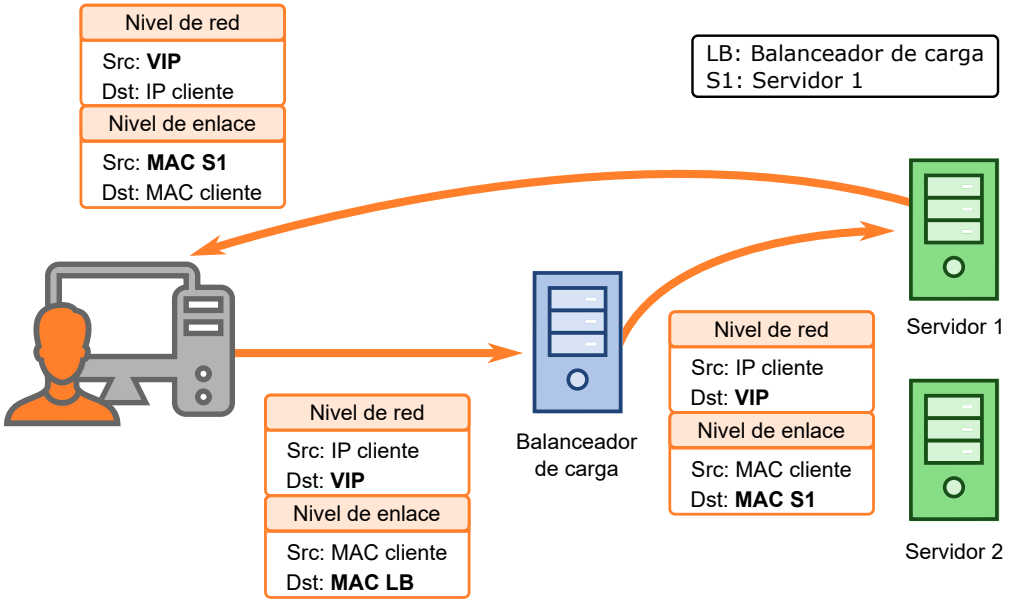



**Figura 2.5.** Diagrama de balanceo de carga basado en *Direct Server Return*

Como puede apreciarse, el usuario accede al balanceador de carga a través de la dirección VIP en vez de la dirección IP real. Cuando el balanceador recibe tráfico en la dirección VIP, en vez de cambiar la dirección IP destino a la dirección IP real del servidor, se utiliza el MAT para realizar la traducción de la dirección MAC de destino.

Como el balanceador cambia la dirección MAC de destino, la dirección de destino sigue siendo la dirección VIP, pero ahora hace referencia al servidor final en vez de al balanceador. Por esta razón el servidor puede enviar directamente el tráfico a la dirección IP del cliente sin necesidad de pasar por el balanceador.

Como es de esperar, esta implementación supone un incremento importante del rendimiento de la red. Además, la potencia de procesamiento necesaria para implementar esta técnica es mínima, por lo que este método es utilizado en los servidores expuestos al exterior con grandes cantidades de tráfico.

### 2.3.3. IP *Tunneling*

Esta tecnología se basa en la implementación del método Direct Server Return (DSR) mediante el uso de tunelización IP, es decir, encapsular el protocolo IP sobre él mismo. Esta configuración permite al servidor final, al igual que con el método DSR, responder directamente al usuario sin necesidad de pasar por el balanceador.

Con esta configuración, el balanceador no necesita estar en la misma red que los servidores finales. El funcionamiento es el siguiente: el usuario envía una petición a la dirección VIP del balanceador. A continuación, el balanceador encapsula el paquete recibido en otro paquete IP antes de enviarlo al servidor seleccionado. Después el servidor final recibe el paquete, con la dirección IP origen correspondiente al balanceador, desencapsula el paquete y lee el paquete original que envió el usuario. Y por último, el servidor le envía la respuesta directamente al usuario.

La ventaja de utilizar este método es que el balanceador de carga no se tiene una sobrecarga debida a la modificación de los paquetes.

### 2.3.4. Traducción de la dirección de red

El NAT (*Network Address Translation*) se basa en la manipulación de la dirección de red (nivel 3 del modelo OSI) del origen o del destino. Este método permite que un dispositivo con una dirección interna tenga acceso a la red externa. De la misma manera, cuando un terminal perteneciente a la red interna quiere acceder a algún dispositivo de la red interna, el balanceador de carga mapea la dirección externa en una dirección interna [2].

Con objeto de entender cómo funciona este método, se va a explicar cuál es el flujo de datos, el cual se representa de forma gráfica en la *Figura 2.6*. El usuario accede desde su navegador a la dirección VIP del balanceador (1). Éste en vez de responder a la petición, reescribe la dirección IP de destino con la correspondiente al servidor real que ha elegido para realizar el balanceo (2). El servidor recibe la petición y manda de vuelta la respuesta al usuario por el camino que tiene configurado por defecto, es decir, el balanceador de carga (3). Teóricamente, la dirección de origen de la respuesta debiera ser la dirección IP del servidor, pero si el usuario recibiera una respuesta con esta dirección, la ignoraría ya que este no inició una conexión con esa dirección. Para solventar este problema, es necesario cambiar la dirección IP de origen por la dirección VIP, para que así el usuario reciba una respuesta con la misma dirección origen con la que él estableció la conexión (4).

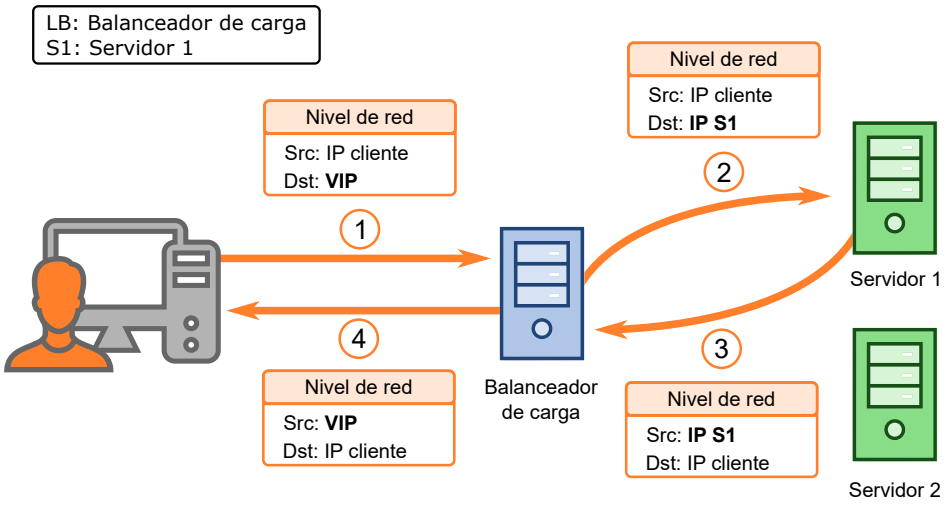



**Figura 2.6.** Diagrama de balanceo de carga basado en la traducción de direcciones de red

Una de las grandes limitaciones de esta tecnología se encuentra en el cuello de botella que supone el balanceador de carga. A diferencia del método anterior, el tráfico de vuelta tiene que pasar por el balanceador, por lo que este tiene que realizar tanto las traducciones de ida como las de vuelta. Además, si los tiempos de espera de sesión son demasiado cortos en el balanceador de carga provocan un efecto conocido como la tormenta de ACKs (descrita en la *Subsección 2.3.5*). En este caso, la única solución es incrementar los tiempos de espera con el riesgo de saturar la tabla de sesión del balanceador de carga [8].

### 2.3.5. Conmutación de capa 4

En esta tecnología se utiliza la información correspondiente al nivel 4 para decidir cómo se va a distribuir la petición del cliente entre el grupo de servidores. Cuando se habla de la información del nivel 4, se hace referencia a las direcciones de red (nivel 3) de origen y destino y a los puertos (nivel 4) que se describen en la cabecera del paquete de esta capa [2]. Por ello, también se la conoce como "balanceo de carga de la capa 3/4". Es decir, no es más que la utilización de los métodos de balanceo de nivel 2 o 3 (DSR o NAT) añadiendo algunas mejoras mediante la utilización de protocolos de nivel 4.

Como se explicó en la *Subsección 2.1.1*, este nivel proporciona una transmisión segura entre los dispositivos extremos, asegurando el control de errores, de flujo y el ordenamiento de los paquetes. Si bien son funcionalidades definidas para esta capa, algunos protocolos, como por ejemplo UDP, no cumplen estas características. UDP es un protocolo no orientado a la conexión, es decir, no realiza ninguna de las funcionalidades descritas anteriormente, siendo la aplicación que confíe en su uso, responsable de gestionar el problema de la fiabilidad [5].

Al ser UDP un protocolo no orientado a la conexión, su utilización para este método no tiene ningún sentido. Por ello, el protocolo utilizado es TCP. Se trata de un servicio de transferencia de segmentos (PDUs) orientado a la conexión. Asegura la reordenación de los datos, independientemente de que los datagramas IP no se reciban en orden. Realiza control de flujo, control de errores, reconoce los segmentos mediante el envío de paquetes ACK y se realizan retransmisiones en caso de pérdida o errores. Al ser un protocolo orientado a la conexión, requiere un procedimiento explícito de establecimiento y finalización de la comunicación, identificando cada conexión mediante un número de secuencia diferente.

Por otro lado, debido al uso del protocolo TCP, los balanceadores de nivel 4 tienen que afrontar algunos de los problemas que se describen a continuación [8]:

- **Cabeceras multi-paquete**: el balanceador busca cadenas de caracteres dentro de los paquetes. Cuando los datos esperados no se encuentran en el primer paquete, este debe ser almacenado, suponiendo un consumo importante de memoria.
- **Fragmentación**: cuando un paquete es demasiado grande para ser enrutado en una sola pieza, este es dividido en paquetes más pequeños. Como los dispositivos encargados de enviar los paquetes a través de la red son los enrutadores (nivel 3), es muy probable que los fragmentos sigan caminos diferentes hasta llegar al destino (no se garantiza la conexión extremo a extremo). Por eso, es muy probable que el orden

de llegada al destino sea aleatorio y desordenado. Además, para que el destino sea capaz de ensamblarlos, los fragmentos deberán ser almacenados.

- **Paquetes en orden inverso**: cuando un paquete es enviado a través de Internet, es muy probable que tenga que ser fragmentado para poder atravesar la red, como se ha mencionado en el punto anterior. El balanceador de carga tiene que ser capaz de reconstruir el mensaje original, aunque tenga los paquetes desordenados.
- **Paquetes perdidos y retransmisiones**: los paquetes que se pierden entre el origen y el destino o que llegan erróneamente, son retransmitidos de manera transparente después de un tiempo de espera. Esto supone que el procesado realizado en un paquete tiene que realizarse de nuevo, por lo que el balanceador de carga no puede dar por hecho que ha terminado el procesamiento hasta que se retransmite el paquete y se realiza de nuevo el procesado.
- **Inserción de datos**: cuando el balanceador inserta datos, los valores correspondientes a los números de secuencia TCP así como a los checksums tienen que volver a ser creados para todos los paquetes.

Además, la memoria también es un recurso limitado: los balanceadores basados en red necesitan memoria para almacenar los segmentos y las sesiones. Cuando el servidor se encuentra en la desconexión pasiva, un usuario que intente establecer una conexión no puede utilizar el puerto asociado a la conexión en estado pasivo. Esto puede afectar a la escalabilidad del sistema, ya que el tiempo de espera puede estar en torno a los 4 minutos. Si se cierran muchas conexiones simultáneamente, la lista de conexiones en estado de espera en la tabla de sesiones del balanceador, es considerablemente grande. Además, existe un número finito de conexiones que pueden estas activas simultáneamente. Esta limitación es debida a que el número de puertos disponibles es limitado. Es probable que sea complicado establecer una nueva conexión si existen muchas conexiones en el estado de espera.

Previamente se ha hablado de la tormenta de ACKs que puede sufrir el balanceador cuando realiza el NAT. Esto ocurre por un reúso temprano de las sesiones recién terminadas, causando una saturación de la red entre un usuario dado y un servidor. Esto no ocurre cuando se realiza el balanceo de nivel 4 ya que al tener implementado TCP, las sesiones se cierran completamente entre el cliente y el servidor después del TIME_WAIT.

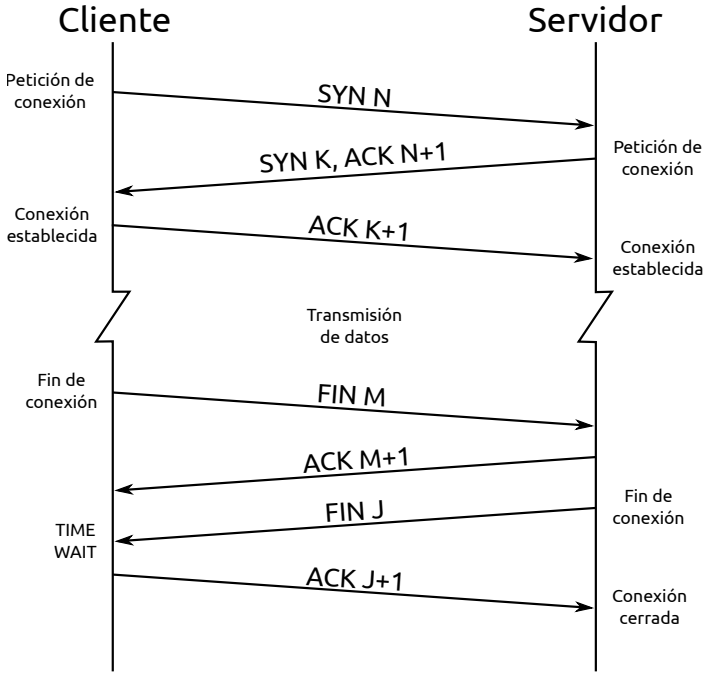



**Figura 2.7.** Establecimiento y cierre de la conexión TCP
**Fuente**: "http://www.see-my-ip.com/tutoriales/imagen/tcp_abrir_cerrar_conexion.jpg"

### 2.3.6. *Reverse Proxy*

Este método basa la decisión de la distribución de la carga en el tipo de aplicación o protocolo de aplicación que el usuario utiliza. Por eso, se conoce también como balanceo de carga de capa 7 [9].

En la *Figura 2.8* se representa el diagrama de estados del balanceo de carga cuando se hace uso de este método. Así, un servidor *reverse proxy* recibe peticiones de los clientes de la red pública, los envía a los servidores pertenecientes a una red interna y, a continuación, envía al usuario las respuestas que se generan en los servidores. El *proxy* evalúa las peticiones y determina, según la información incluida en la petición del usuario y el algoritmo de distribución de carga, cuál es el mejor servidor al que redireccionar esta petición. El balanceador se comunica con el servidor en nombre de la aplicación del cliente, es decir, envía las peticiones del usuario al servidor final y devuelve las respuestas desde el servidor hasta el usuario. De esta manera, el *proxy* proporciona al cliente la ilusión de que se está comunicando directamente con el servidor final, aunque no ocurra así [10].

Esto implica que los servidores nunca acceden directamente a los usuarios. Aunque este método necesita más recursos de procesamiento que aquellos que trabajan en el nivel de red, el hecho de separar la comunicación entre los servidores y los usuarios proporciona un nivel de seguridad adicional [8].

En contraposición a los balanceadores de nivel de transporte, los *proxies* no "ven" ninguna sobrecarga en el procesado de capa 7, ya que todos los problemas citados en el

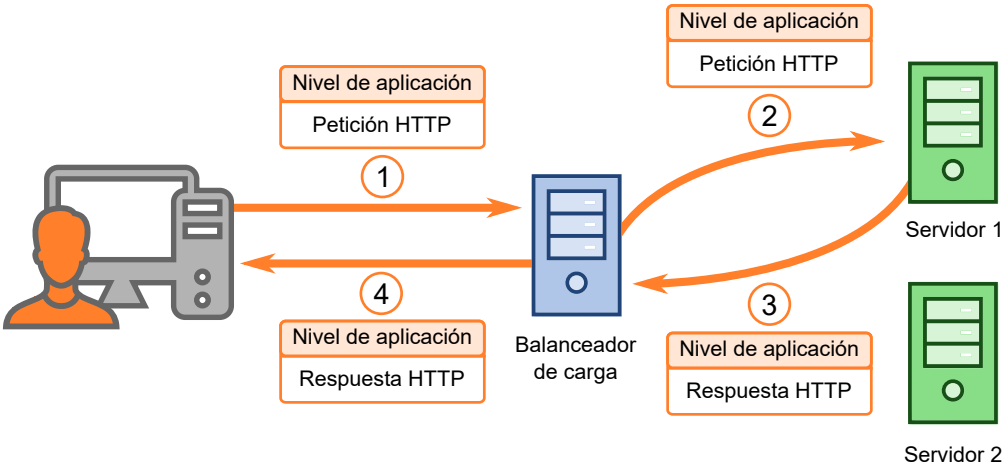



**Figura 2.8.** Diagrama de estados del balanceo de carga mediante el uso de un *Reverse Proxy*

balanceo de nivel 4 son solventados de forma natural por las capas inferiores. Por ello, el balanceador de carga solo tiene que centrarse en el contenido y realizar las acciones correctas. Además, ofrecen un registro muy detallado sin ningún tipo de coste extra.

Este método presenta algunas limitaciones. El contenido se puede identificar erróneamente, enviando las peticiones al servidor equivocado. En otras ocasiones, la respuesta no es procesada antes de ser devuelta al usuario. Además, esta tecnología solo suele soportar un protocolo o un número definido de protocolos. Por ello, si, por ejemplo, una gran empresa quiere desplegar múltiples servicios con protocolos diferentes, se deberían desplegar varios *proxies*, resolviendo cada uno de ellos un protocolo diferente.

## 2.4. Algoritmos utilizados en la distribución de la carga

El balanceador de carga tiene varias formas de distribuir la carga entre los diferentes servidores. Una comúnmente utilizada consiste en enviar la petición al primer servidor que responda. Esta práctica es errónea ya que, si el servidor tiene cualquier razón para responder un poco más rápido, esto descompensará el clúster de servidores asumiendo la mayor parte de las peticiones.

Una segunda aproximación consiste en enviar la petición al servidor que esté menos cargado[4]. Esta aproximación puede ajustarse correctamente en entornos en los cuales el tiempo de la sesión[5] no sea muy corto, es decir, las conexiones perduran durante un largo periodo de tiempo. Sin embargo, esta no se ajusta bien en los casos en los cuales los servidores reciben muchas peticiones en un periodo de tiempo corto.

Para clústeres de servidores homogéneos, el método más utilizado es *Round Robin*. Este método alterna las peticiones entre los servidores. Si cada servidor tiene una capacidad

[4]El término cargado se refiere a la cantidad de peticiones que un servidor está llevando a cabo.
[5]Se conoce el término sesión como la duración de una conexión entre un usuario y un servidor.

diferente, esto se conoce como el método *Round Robin* ponderado, es decir, el balanceador de carga asignará la petición entre los diferentes servidores acorde a la capacidad de carga relativa configurada en cada uno.

Estos algoritmos presentan algunas desventajas. Una de ellas es que no son deterministas, es decir, dos peticiones consecutivas del mismo usuario tienen una probabilidad alta de ser gestionada por dos servidores diferentes. Si la sesión del usuario es almacenada en los servidores, esta se perderá entre las dos peticiones consecutivas. Además, si se inicia una conexión un poco más complicada, como por ejemplo una de tipo SSL/TLS [6], esta conexión se tendrá que negociar con cada petición.

Para resolver este problema, se puede utilizar el algoritmo conocido como *dirección hash*. En él, la dirección IP del cliente es dividida entre el número de servidores disponibles. El resultado determina el servidor correcto para ese usuario particular. Esto funciona correctamente mientras el número de servidores no cambie y la dirección IP del usuario sea estática. En el caso de que un servidor falle, todos los usuarios son rebalanceados y pierden sus sesiones. Este método no se puede aplicar siempre debido a que, para tener una buena distribución, se requieren un elevado número de direcciones IP. La solución a ese problema es utilizar la persistencia de sesión.

## 2.5. Persistencia de sesiones

La persistencia de sesiones es una forma de asegurar que un cierto usuario mantiene todas sus peticiones en un mismo servidor. El uso de este método surge sobretodo en los balanceadores de carga de nivel 7, donde HTTP es el protocolo por excelencia en las comunicaciones *web* que tienen lugar en este nivel. Sin embargo, HTTP es un protocolo sin estado, es decir, no tiene ningún conocimiento de la solicitud o respuesta que el cliente o servidor ha realizado previamente. Esto supone una desventaja a la hora de implementar la persistencia de sesiones.

En un escenario donde no hay balanceo de carga no supone ningún problema, ya que todas las conexiones son enviadas al único servidor existente. Sin embargo, en un sistema con balanceo de carga es importante que el servidor tenga consciencia de las conexiones que tiene establecidas.

Una solución fácil es enviar una respuesta HTTP con código 302, es decir, una redirección. La principal desventaja que esto tiene es que cuando el servidor falla, el usuario no tiene forma de volver hacia atrás y ser redirigido a otro servidor.

Una segunda solución consiste en que el balanceador de carga aprenda las asociaciones entre usuario y servidor. La forma más fácil de implementar esto es que el balanceador almacene a qué servidor fue enviada la petición de usuario la última vez. Generalmente, esto supone un problema cuando el tamaño del clúster de servidores varía debido a fallos. Sin embargo, esta solución no resuelve el caso en el que el usuario tenga una dirección IP variable o se esté utilizando un balanceo mediante la traducción de direcciones de red (NAT).

[6] Protocolo criptográfico que proporciona comunicaciones seguras a través de la red.

Con el fin de solucionar el inconveniente del protocolo HTTP y mantener la conexión de un usuario, se utilizan las *cookies*. Estas fueron creadas con el fin de enviar información al usuario, la cual será devuelta por este en posteriores accesos. Esto requiere que el usuario acepte las *cookies*, pero generalmente esto sucede en las aplicaciones que necesitan persistencia.

Si el balanceador de carga pudiese identificar un servidor basándose en la *cookie* enviada por el usuario, esto resolvería la mayoría de los problemas. Existen dos aproximaciones principales: el balanceador de carga puede almacenar la sesión *cookie* asignada por los servidores o pueden anotar la cookie para la identificación del servidor.

Si bien esta sección hace referencia al concepto de persistencia citado anteriormente, existe otro concepto más general denominado "replicación de la sesión" (*session replication*). Se trata de replicar la información de una sesión de un usuario con un servidor, entre los múltiples servidores que forman parte del clúster. De esta manera, si el servidor con el que el usuario está manteniendo la comunicación falla, el usuario no perderá la sesión al conectarse a otro servidor.

### 2.5.1. Almacenamiento de la cookie

Es la solución menos intrusiva. El balanceador de carga es configurado para aprender la *cookie* de aplicación. Cuando recibe la petición del usuario, chequea si contiene esta *cookie* y un valor conocido. Si no se da este caso, envía la petición a cualquier servidor de acuerdo con el algoritmo utilizado para la distribución de la carga. Después, extraerá el valor de la *cookie* que el servidor envía de vuelta en la respuesta y lo añade a la tabla de sesión junto con la información del servidor. Cuando el usuario vuelve a realizar una petición, el balanceador de carga analiza la *cookie*, consulta su tabla de sesión y envía la petición a ese servidor.

Aunque este método es muy fácil de desplegar, tiene dos desventajas. En primer lugar, el balanceador de carga tiene memoria finita, por lo que se podría saturar. La única solución para evitar esto es modificar el límite del tiempo de vida de la *cookie* en la tabla. Esto implica que, si el usuario vuelve después del tiempo de expiración de la *cookie*, será enviado al servidor equivocado.

En segundo lugar, si el balanceador de carga se cae y vuelve otra vez a funcionar, no conocerá ninguna asociación entre los servidores y las *cookies*, por lo que volverá a redirigir a los usuarios al servidor equivocado. Por supuesto, la utilización de un escenario con la combinación Activo-Activo (ver *Figura 2.3*) no resuelve este problema ya que, como se detalló anteriormente, todas las conexiones existentes en el servidor que deja de funcionar desaparecen, aunque el otro servidor activo tome todas sus funciones. La solución a este problema es la utilización del método *session replication* mencionado anteriormente. Para ello es necesario, o bien configurar una base de datos, o bien implementar un sistema de archivos compartidos entre todos los servidores del clúster con objeto de tener almacenada la información relativa a todas las sesiones.

Una solución alternativa a estas desventajas es utilizar un algoritmo determinístico, como un "hash"basado en la dirección IP. De esta manera, la *cookie* se pierde en el balan-

ceador de carga, pero al menos los usuarios que tienen una dirección IP estática pueden mantener la sesión en el servidor.

### 2.5.2. Anotación de la cookie

Este método consiste en insertar el identificador del servidor en la *cookie* de la respuesta HTTP. De esta manera, el balanceador de carga no necesita aprender nada, simplemente reutiliza el valor presentado por el usuario para seleccionar el servidor correcto. Esto soluciona los problemas presentes en el método de aprendizaje de la *cookie*: limitación de memoria y la toma de control del balanceo de carga. Sin embargo, requiere un mayor esfuerzo por parte del balanceador de carga, el cual debe abrir la transmisión para manipular los datos.

### 2.5.3. Limitaciones

Hasta el momento se han descrito métodos que implican decodificar los intercambios entre usuario y servidor, y a veces la necesidad de realizar alguna modificación, pero el balanceador de carga siempre tiene acceso al contenido de HTTP. Pero con el aumento de las aplicaciones que confían en SSL/TLS para mantener su seguridad, los balanceadores de carga no son capaces de acceder al contenido HTTP. Por esta razón, cada vez existen más balanceadores de carga que dan soporte a SSL/TLS.

El balanceador de carga actúa como *reverse proxy* y sirve como un servidor SSL/TLS final. Mantiene los certificados de los servidores, descifra las peticiones, accede al contenido y envía la petición directamente a los servidores (bien como texto en claro o sin cifrar). Esto conlleva a una nueva aproximación en el que los servidores de aplicación no necesitan administrar SSL/TLS. Sin embargo, el balanceador de carga se convierte en un cuello de botella: un solo balanceador de carga basado en *software* (ver *Subsección 2.6.2*) no es capaz de procesar el contenido SSL/TLS tan rápido como un clúster de varios servidores.

Debido a esta limitación en la arquitectura, el balanceador de carga se satura mucho antes que los servidores de aplicación, y la única solución para esto, es añadir otro balanceador de carga para manejar la carga SSL/TLS. La solución más escalable cuando las aplicaciones requieren SSL/TLS es tener un clúster de *proxies* inversos solo para manejar la carga SSL/TLS.

## 2.6. Tipos de soluciones

Los balanceadores de carga se pueden dividir en dos tipos: los basados en conmutación, también conocidos como basados en *hardware*, o los basados en *software*. La elección de un tipo u otro depende de las características que requiere el sistema a implementar.

### 2.6.1. Soluciones basadas en Hardware

Se trata de dispositivos que confían en chips ASIC[7] (*Application-Specific Integrated Circuit*) para desarrollar las funciones de manipulación de los paquetes. Estos dispositivos son procesadores mucho más especializados que los procesadores usados para propósitos generales, los cuales permiten implementar sobre ellos una variedad más amplia de *software*. Gracias a su especialización, son procesadores que desarrollan las tareas mucho más rápido y de manera más eficiente que los procesadores generales.

Aunque parece que el número de ventajas es mucho mayor, este tipo de balanceadores presentar una gran desventaja: son muy poco flexibles. Si se requiere una nueva tarea que no está implementada, se debe crear un nuevo diseño ASIC. Como el protocolo IP es una tecnología muy estable, la elección de este tipo de balanceadores es perfecta cuando se quieren implementar las tecnologías de balanceo hasta el nivel 3 (DSR, NAT).

### 2.6.2. Soluciones basadas en Software

Se trata de dispositivos que ejecutan un SO estándar. Las técnicas de balanceo de carga son implementadas mediante la ejecución de un *software* en dicho sistema operativo. Este tipo de balanceadores son muy fáciles de desarrollar debido a la cantidad de recursos de código que existen para los sistemas operativos.

En contraposición a los dispositivos basados en *hardware*, estos balanceadores son muy flexibles. Añadir nuevas características, o modificar las existentes, al dispositivo se solventa modificando el código en la configuración.

Por lo general, este tipo de balanceadores suele implementar la tecnología de balanceo basada en la capa de nivel 7.

## 2.7. Conclusión

Una vez analizadas todas las alternativas posibles, se procede a seleccionar aquella más adecuada para los propósitos de este trabajo.

Los parámetros más comunes en la elección del sistema de balanceo son: el rendimiento, la fiabilidad y la escalabilidad, así como el espacio entre sesiones. La evaluación de este último depende la implementación o no del protocolo TCP en el balanceador. Esto es debido a que TCP requiere que, una vez que la sesión ha terminado, esta permanezca en la tabla de sesión en el estado TIME_WAIT lo suficiente como para poder realizar la retransmisión en caso necesario. Después de este tiempo, la sesión es automáticamente eliminada liberando memoria en el balanceador. En la práctica, los retrasos son del orden de entre 15 y 60 segundos. Esto supone un problema real en los sistemas que dan soporte a sesiones con altas tasas de velocidad, ya que todas las sesiones terminadas tienen que estar almacenadas en la tabla.

[7] Circuito integrado hecho a medida para un uso en particular, en vez de ser concebidos para propósitos de uso general.

Para los balanceadores de carga que implementan el balanceo en la capa 7, esta operación se realiza de manera transparente por el SO, por lo que el balanceador omite estas sesiones que se encuentran en espera y solo tiene en cuenta aquellas que aún no han terminado. Sin embargo cuando el balanceador trabaja en la capa de nivel 3-4, las tablas de sesión son tan extensas que el balanceador podría llegar a saturarse.

Como ya se ha mencionado anteriormente, las soluciones basadas en *software* ofrecen una mayor flexibilidad y escalabilidad a los usuarios que los basados en *hardware*. Esta escalabilidad permite una mayor capacidad de satisfacer unas necesidades en tiempo real bajo demanda. Por otro lado, si un balanceador de carga falla, se puede desplegar rápidamente otro balanceador con las mismas características que el anterior, sin tener un gran impacto sobre el sistema.

Además, el objetivo de este trabajo es proporcionar un servicio THREDDS para cada Group WorkSpace (GWS), como se explicará en el siguiente capítulo. Sin embargo, aunque este sea el objetivo funcional, en relación a la implementación técnica se utiliza un sistema de balanceo de carga. El balanceador de carga distribuye el tráfico entre los diferentes clústeres que tiene configurados, los cuales pertenecen a un GWS diferente. En este escenario, la diferencia de acceder al servicio THREDDS de un GWS u otro se basa en la petición HTTP que el usuario haga al balanceador. Por esta razón, se va a implementar un balanceador de carga de nivel 7.

# Capítulo 3

# Infraestructura y servicios

El objetivo de este trabajo, como ya se ha mencionado al final del capítulo anterior, consiste en desplegar un servicio THREDDS, para un grupo de científicos que colaboran entre ellos, en un clúster de servidores, utilizando un sistema de balanceo de carga. Dicho grupo utiliza la infraestructura JASMIN para llevar a cabo sus proyectos, razón por la cual en este capítulo se explica qué es JASMIN y cuáles son sus características principales. Esta infraestructura integra conceptos relacionados con la computación *cloud*, como por ejemplo IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) o SaaS (*Software as a Service*). Es por ello que la primera parte de este capítulo contiene una explicación de la computación *cloud*. Finalmente, se concluye el capítulo con una justificación sobre por qué el uso del servicio THREDDS es el más apropiado para satisfacer los requisitos de los *Group Workspaces*.

## 3.1. Computación *cloud*

La computación *cloud* es una tecnología emergente que utiliza los conceptos de virtualización[1], conectividad, potencia de cálculo, almacenamiento y compartición de los recursos de computación, y todo ello a través de Internet. Es un término genérico para todo aquello que esté relacionado con la provisión de servicios a través de Internet [11].

De acuerdo al Instituto Nacional de Estándares y Tecnología (NIST): "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [12].

Es decir, la computación *cloud* es un modelo para el acceso a recursos de computación compartidos que pueden ser rápidamente desplegados. Surge como una alternativa a los servidores locales que manejan las aplicaciones. Los usuarios finales de una red de

[1] Creación a través de *software* de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de *hardware*, un sistema operativo, un dispositivo de almacenamiento o cualquier otro recurso de red.

computación basada en *cloud* desconocen la localización física de los servidores que les proporcionan el servicio.

El término *cloud* se utiliza como una metáfora de Internet, basada en el dibujo de una nube que se utilizaba en el pasado para representar la red telefónica, y más tarde la utilizada para representar Internet en los diagramas de computación de la red [13]. Algunas de las ventajas de este modelo de computación, por las cuales cada vez tiene más importancia en las empresas y administraciones públicas, son las siguientes [11]:

- ✓ Ahorro de costes con respecto a la inversión de capital que necesitarían para llegar a tener los mismos recursos físicos.
- ✓ Reducción de costes con respecto al desarrollo y la entrega del servicio IT.
- ✓ Reducción de la responsabilidad de gestión, permitiendo al personal centrarse más en la producción e innovación.
- ✓ Incremento en la agilidad del negocio permitiendo a las empresas conocer las necesidades del mercado, el cual se encuentra en un constante cambio.

A pesar de los beneficios que presenta, existen algunas preocupaciones sobre los proveedores de servicios y los usuarios finales a la hora de utilizar este servicio tan moderno y potente. Estas preocupaciones están relacionadas con la seguridad y la privacidad en los entornos de computación *cloud* [10].

Asimismo, las características esenciales de la computación basada en la nube son [12]:

- ▹ **Autoservicio bajo demanda**: los usuarios son capaces de modificar capacidades como su tiempo o el almacenamiento de red, de forma automática y cuando sea necesario, por motivos de demanda.
- ▹ **Amplio acceso a la red**: los usuarios pueden acceder a las capacidades a través de la red mediante mecanismos estandarizados que promueven el uso heterogéneo de las plataformas de los clientes.
- ▹ **Uso común de los recursos**: se establece un grupo de recursos que combina diferentes medios físicos y virtuales para dar servicio a múltiples usuarios de manera dinámica, de acuerdo a la demanda requerida.
- ▹ **Elasticidad rápida**: los recursos pueden ser suministrados y eliminados de manera dinámica, en algunos casos de forma automática, con objeto de escalar rápidamente, expandiéndose y contrayéndose en relación a la demanda.
- ▹ **Métricas del servicio**: los sistemas *cloud* deben optimizar la localización de los recursos y proporcionar métricas de uso con propósitos de facturación.

### 3.1.1. Tipos de *cloud*

En función de la forma en la que se ha desplegado la nube, existen cuatro tipos principales. A continuación se procede a describir cada uno de ellos [10].

#### *Cloud* pública

También conocida como la nube externa. Es un servicio en la nube proporcionado por terceros, alojado y administrado por los proveedores de servicios. La infraestructura es ofrecida de manera libre, aunque no gratuita, para el público general. El proveedor asume las responsabilidades de instalación, configuración, aprovisionamiento y mantenimiento. Suelen estar disponibles mediante una conexión de red segura y restringida y proporcionan servicios de pago-por-uso bajo demanda. A pesar de las ventajas que presenta, se enfrenta a problemas de privacidad y seguridad.

#### *Cloud* privada

También conocida como nube interna, permite acceder a los recursos de computación en la nube en exclusiva a una única organización. A diferencia de la nube pública, en este caso la compañía es responsable del aprovisionamiento y mantenimiento de la infraestructura. Sin embargo, esta proporciona una privacidad mayor que la anterior. Se trata de redes propietarias, a menudo centros de datos, que residen dentro de una empresa para el uso exclusivo de la organización o de un grupo conocido de usuarios. Aunque normalmente se encuentren dentro de la organización, también es posible que estén fuera, donde los recursos están solo disponibles para los miembros de la empresa y son inaccesibles para el resto de usuarios.

#### *Cloud* comunitaria

Es una nube semi-privada utilizada por una comunidad de usuarios con unos conocimientos y requisitos comunes. Esta comunidad específica de usuarios puede, a su vez, pertenecer a alguna organización física o virtual. En este caso, una o más compañías de la comunidad deben gestionar la infraestructura. De esta manera, se convierte en una nube privada para la comunidad, pero la responsabilidad de la gestión es compartida entre todos los miembros de la comunidad.

#### *Cloud* híbrida

Es una combinación de la nube pública y privada. En este caso, las responsabilidades de gestión son divididas entre la empresa y los proveedores de la nube pública. Para los procesos críticos, este tipo de infraestructura puede resultar muy eficiente debido a la mejora en el control y la gestión por parte de la empresa en sí. Por ejemplo, las organizaciones pueden mantener datos sensibles dentro de la parte privada y el resto en la parte pública.

### 3.1.2. Servicios *cloud*

Como se observa en la *Figura 3.1*, la arquitectura de estos entornos de computación se encuentra dividida en cinco capas principales. A continuación se enumeran estas cinco capas:

**(1)** Infraestructura física

**(2)** Infraestructura virtual

**(3)** Plataforma

**(4)** Aplicación

**(5)** Red

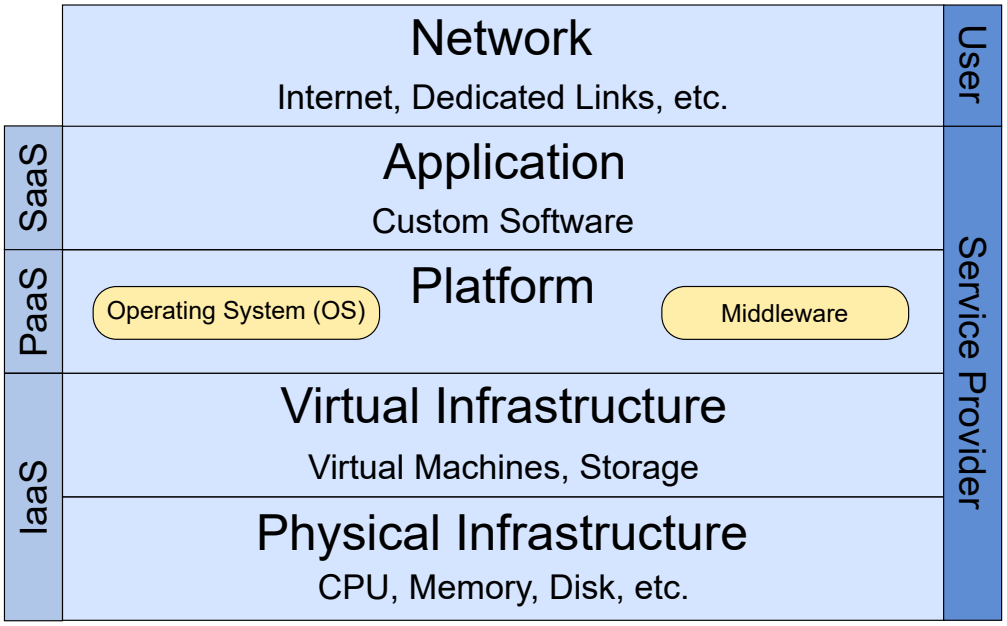



**Figura 3.1.** Arquitectura de la Computación Cloud
**Fuente**: "*Cloud computing: Vision, architecture and Characteristics*" [11]

De acuerdo a las 5 capas de la *Figura 3.1*, se definen tres modelos diferentes de arquitectura que hacen referencia a tres tipos de servicios genéricos.

#### Infraestructura como Servicio (IaaS)

Este nivel es, esencialmente, de servicios *hardware*, es decir, un proveedor de una IaaS invierte en infraestructura, la despliega y la mantiene para ofrecer un *hardware* físico o virtual. Se trata de la entrega del *hardware* y el *software* asociado como un servicio donde el consumidor es capaz de desplegar y ejecutar un *software*. Los usuarios tienen un control total sobre la infraestructura de servicio y pueden desplegar y mantener los servicios

*software* ellos mismos. En Infrastructure as a Service (IaaS), el usuario busca adquirir recursos físicos de computación sin realizar una inversión para tener esos mismos recursos bajo su responsabilidad. Los proveedores de servicios IaaS ofrecen nubes públicas, privadas virtuales y herramientas para nubes privadas. La infraestructura se ofrece con la filosofía "paga-lo-que-uses", por lo que a veces se la conoce como "computación útil", ya que existe una similitud entre el aprovisionamiento y el uso de los servicios como con los servicios de gas o electricidad. Algunos productos y proveedores de IaaS son Amazon EC2, IBM Blue House, VMWare, GoGrid, RigthScale and Linode [11].

**Plataforma como Servicio (PaaS)**

Este nivel se encarga de proporcionar al consumidor la capacidad de desplegar sus propias aplicaciones sin el coste y la complejidad de comprar y gestionar la infraestructura inferior. Por ello, este servicio se refiere a las herramientas de desarrollo *software* y productos que los clientes compran para que puedan construir y desplegar sus propias aplicaciones, proporcionando así una mayor flexibilidad y control. De hecho, se elimina el coste asociado a la configuración, gestión y supervisión de la infraestructura de la nube. Además, se ofrece una plataforma para implementar y cargar aplicaciones personalizadas por los desarrolladores. Este nivel se puede interpretar también como el "nivel del desarrollador". El precio de este servicio se basa en el uso de computación por horas, la transferencia y almacenamiento de datos por *gigabytes*, las peticiones de almacenamiento de datos, etc... Algunos ejemplos de productos y proveedores de PaaS son el motor de aplicaciones de Google, Heroku y Mosso.

***Software* como Servicio (SaaS)**

Se trata del nivel más alto del entorno de computación *cloud*. Suprime la instalación del *software*, el pago de licencias, las configuraciones del *middleware*y la administración del sistema. Además, mejora la rapidez de la instalación del *software*, la configuración y la personalización. Se trata de aplicaciones preconstruidas e integradas que se entregan a los clientes como un servicio. En este nivel, el consumidor no tiene que preocuparse por desarrollar o programar el *software*, aunque puede configurarlo. SaaS abarca un mercado muy amplio donde los servicios pueden ser desde correos electrónicos basados en *web*, hasta servicios bancarios *online*, además de procesamiento de la base de datos. Algunos productos y proveedores de SaaS son Gmail, Hotmail, IBM WebSphere y Boomi.

## 3.2. Infraestructura JASMIN

La infraestructura JASMIN (*Joint Analysis System Meeting Infrastructure Needs*) es un "*super-data-cluster*" creado para suplir los requisitos de análisis de datos de la comunidad de modelización de sistemas climáticos y terrestres del Reino Unido y Europa. Esta infraestructura está desplegada en el departamento de e-Ciencia del Consejo de Instalaciones de Ciencia y Tecnología (STFC) perteneciente al Laboratorio Rutherford Appleton (RAL) en nombre del Centro Nacional de la Ciencia Atmosférica (NCAS), uno de los seis centros de investigación del Consejo de Investigación del Medio Ambiente del Reino Unido (NERC). El departamento e-Ciencia del STFC gestiona toda la infraestructura y el entorno de red de JASMIN, dejando la funcionalidad científica en manos del Centro de Archivo de Datos Ambientales (CEDA) del STFC. En resumen, JASMIN es un entorno de análisis de datos científicos administrado por el CEDA que da soporte a una amplia variedad de flujos de trabajo científicos en el ámbito de las ciencias ambientales [14].

La infraestructura JASMIN proporciona computación y almacenamiento unidos entre sí por una red de gran ancho de banda. Utiliza una topología única con unas capacidades mucho mayores que las de un centro normal de datos. Esta infraestructura está formada por un sistema central (el súper clúster de datos) y tres sistemas satelitales en las universidades de *Bristol*, *Leeds* y *Reading*. Cada sistema satelital consiste en discos de 150, 100 y 500 *terabytes* de almacenamiento, respectivamente, y en recursos de computación.

La arquitectura técnica fue escogida tanto como para facilitar la gestión como para proporcionar un entorno de almacenamiento y análisis de alto rendimiento muy flexible, y consta esencialmente de seis componentes:

**(1)** Una red central de baja latencia.

**(2)** Un subsistema de almacenamiento basado en la tecnología *Panasas*.

**(3)** Un sistema de computación basado en colas (*batch*).

**(4)** Unos sistemas de cálculo de datos que proporcionan una infraestructura basada en hypervisores para las máquinas virtuales.

**(5)** Un sistema con gran capacidad de memoria.

**(6)** Sistema redundante de almacenamiento para dar soporte a discos privados de las máquinas virtuales.

Los sistemas de computación que conforman la infraestructura JASMIN no solo se utilizan para apoyar las funciones del centro de datos del CEDA, sino también para ayudar a las distintas comunidades científicas, permitiendo el desarrollo de herramientas de análisis de datos en paralelo. Estas herramientas pueden ser utilizadas tanto en un número pequeño de CPUs, típico en máquinas virtuales personalizadas, como en un número mayor (hasta cientos de CPUs). Asimismo, estas son utilizadas para configurar un clúster de nodos de alto rendimiento (HPC).

Adicionalmente, JASMIN proporciona tres tipos de servicios: un entorno virtualizado de computación (aunque no es exactamente una nube privada, utilizan ese término para referirse a este servicio), un entorno físico de computación y un servicio de computación de alto rendimiento (LOTUS). Como se puede apreciar en la *Figura 3.2* los servicios se pueden dividir en dos clases diferentes. Por un lado, se encuentran los servicios de computación y análisis (color naranja) y, por otro lado, se encuentra los servicios de datos (color verde). Como se ha comentado previamente, ambos servicios son gestionados por el CEDA [15].

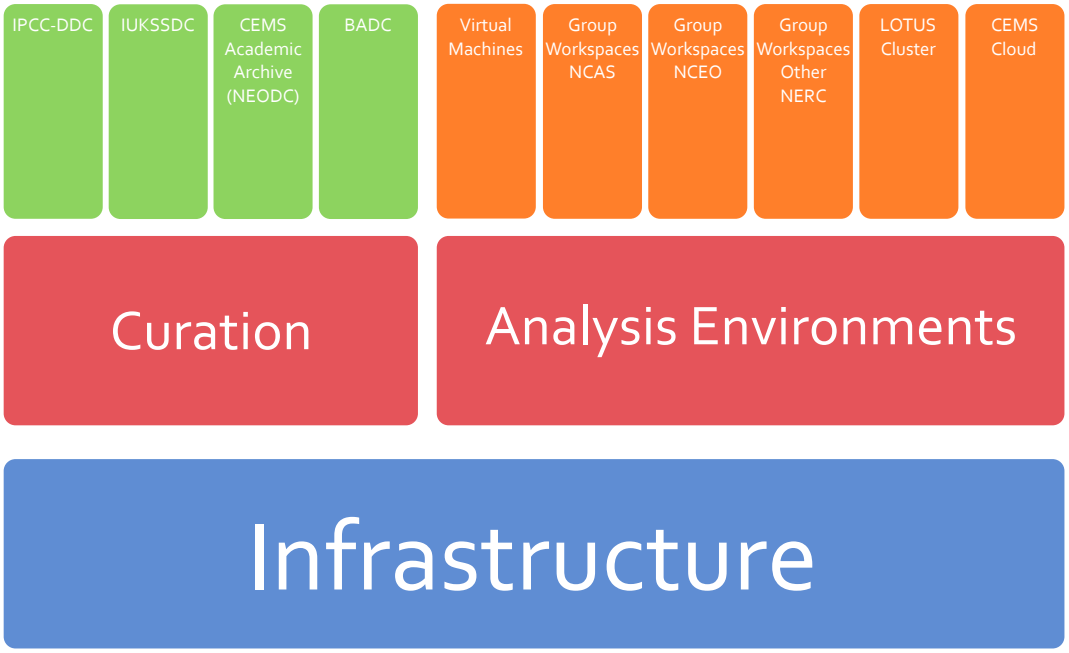



**Figura 3.2.** Diagrama de los niveles y tipos de servicios de JASMIN
**Fuente**: *"http://www.jasmin.ac.uk/services/"* [16]

La comunidad de usuarios a los que se les quiere proporcionar el servicio THREDDS son aquellos correspondientes a los Group Workspaces (GWS). Por ello, es necesario explicar brevemente en qué consisten estos GWSs.

### 3.2.1. Group Workspaces

Los GWSs son porciones de disco del sistema de almacenamiento basado en *Panasas* asignadas para proyectos particulares, permitiendo a los científicos colaboradores de dichos proyectos compartir la red de almacenamiento integrada en JASMIN. Los usuarios pueden extraer datos de sitios externos a una caché común, procesarlos y analizarlos [16].

Además, estos GWSs pueden utilizar los recursos computacionales compartidos, como los servidores de análisis científico y el clúster de procesamiento *batch* LOTUS. Adicionalmente, tienen su propia infraestructura de computación. Cuando estos recursos no son suficientes para cumplir los requisitos de un proyecto en particular, los usuarios pueden crear máquinas virtuales específicas del proyecto en la infraestructura JASMIN. En este punto entran en juego los servicios *cloud* de JASMIN.

### 3.2.2. Servicios *cloud* de JASMIN

Junto con la cola HPC *batch* y los servicios de almacenamiento, JASMIN también proporciona un servicio de computación basado en la nube. La nube de JASMIN es similar al concepto general de nube en cuanto a que permite a las instituciones o proyectos consumir recursos computaciones como servicio sin necesidad de aprovisionar y mantener la infraestructura física. Además, los usuarios pueden crear sus propias máquinas virtuales dentro de la infraestructura JASMIN.

La característica más importante de la nube de JASMIN es que tiene acceso al sistema de almacenamiento basado en *Panasas* (archivo de datos del CEDA, GWSs). Por este motivo, esta es ideal para proyectos que trabajan con esos datos, permitiendo soluciones novedosas para la manipulación y presentación de los datos a los usuarios finales.

Con la finalidad de proporcionar una mayor flexibilidad a los usuarios, preservando la seguridad del sistema, la infraestructura se encuentra dividida en dos partes: *JASMIN Unmanaged Cloud* y *JASMIN Managed Cloud*. La *Figura 3.3* representan los componentes que forman la estructura JASMIN [17]:

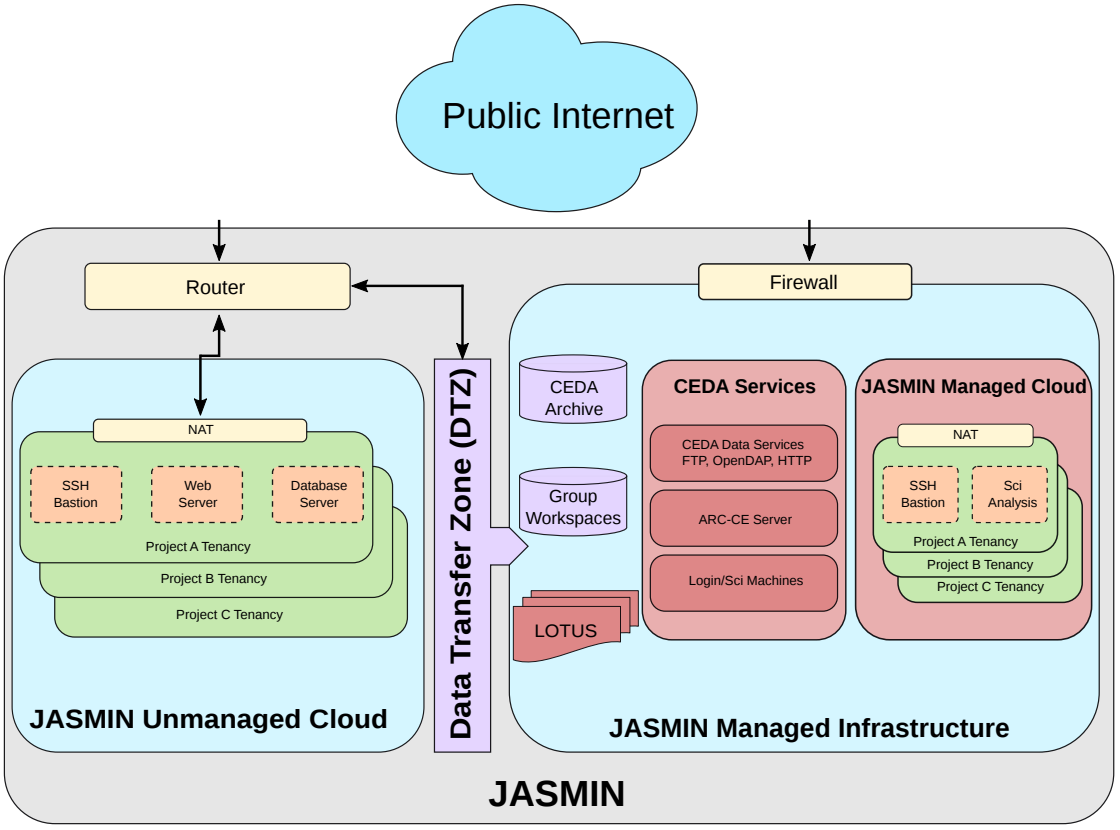



**Figura 3.3.** Infraestructura JASMIN
**Fuente**: "*http://help.ceda.ac.uk/article/282-introduction-to-the-jasmin-cloud*" [17]

En esta puede observarse la división previamente mencionada. Por un lado, se encuentra la parte no administrada (*unmanaged*), y, por el otro, la administrada (*managed*). La primera de ellas se ofrece como una IaaS, y se encuentra fuera del *firewall* de JASMIN.

Los usuarios tienen acceso como *root* a las máquinas, y recae en ellos toda la responsabilidad de la administración del sistema. Además, por encontrarse fuera del *firewall*, los usuarios no tienen acceso directo al sistema de almacenamiento de JASMIN. Por el contrario, estos usuarios sí tienen acceso directo a la zona de transferencia de datos (DTZ) que les conecta con los servicios integrados del CEDA.

La segunda parte que conforma la infraestructura (*managed*) se ofrece como una PaaS. Se encuentra dentro del *firewall* de JASMIN, por lo que los usuarios tienen acceso al sistema de almacenamiento basado en *Panasas*. Para prevenir que dichos usuarios realicen cambios en el sistema, no se les asignan privilegios de acceso *root* a sus máquinas y solo pueden desplegar máquinas virtuales con una plantilla predefinida. Sin embargo, estos no tienen que hacerse cargo de la seguridad de las máquinas. Actualmente solo hay dos plantillas disponibles: *SSH bastion* o una máquina de inicio de sesión y un servidor de análisis científico (*Sci Analysis*) con una configuración similar a los servicios compartidos de análisis científico de JASMIN.

Ambas partes comparten una estructura de red similar. Cada grupo de máquinas tiene su propia red local, donde las máquinas tienen direcciones IP privadas de clase C en el rango 192.168.3.0/24, por lo que todas las máquinas de la *tenancy*[2] pueden comunicarse entre ellas. Además, cada *tenancy* tiene un enrutador virtual. Esto permite a las máquinas dentro de la *tenancy* comunicarse con las máquinas fuera de su *tenancy* asegurando que los paquetes se envían a la máquina correcta. También proporcionan una traducción de direcciones de red (NAT) que permite asignar a las máquinas una dirección Internet Protocol (IP) visible fuera de la *tenancy*. En la nube administrada, esto se traduce en una dirección IP visible en la red JASMIN. En la nube no administrada, se traduce en una dirección IP pública.

La nube de JASMIN, descrita en este apartado, tiene una gran importancia en este trabajo, ya que se utilizarán los recursos de la nube para poder desplegar el sistema de balanceo de carga a implementar.

## 3.3. Servicio THREDDS para los GWS

### 3.3.1. Motivación

Hoy en día las comunidades científicas y, en particular, las ciencias medioambientales, meteorológica y climática requieren colaborar entre ellas. Para ello, necesitan soluciones que les permitan gestionar los datos. En estas disciplinas el uso de modelos y sus simulaciones necesitan acceder y analizar una gran cantidad de colecciones de datos distribuidos. Además, estas actividades involucran un análisis de la cantidad de esos datos mediante la intercomparación de las salidas de estos modelos con observaciones provenientes de un amplio rango de fuentes y formatos [18].

[2]Se trata de una asignación de recursos, es decir, CPUs virtuales, RAM y almacenamiento dentro de la nube.

Un usuario que trabaje en proyectos relacionados con el clima, la meteorología o el medio ambiente debe afrontar tres problemas principales:

(1) **Descubrimiento del dato**: localizar los datos adecuados es el primer paso para su utilización. Para ello, estos datos deben estar recogidos en catálogos, de forma que sean accesibles y, consecuentemente, se maximice su exposición.

(2) **Acceso al dato**: normalmente el metadato[3] proporciona un enlace a la localización de los datos, bien sean físicos o digitales. Un usuario de los datos climáticos debe tener acceso a los mismo. Así como una biblioteca entrega un libro a un usuario remoto a través del servicio postal, un servicio de red es utilizado para la entrega de datos digitales. Además, existen diferentes clases de servicios dependiendo del tipo de datos. Algunos están especializados en una entrega rápida, otros en grandes volúmenes de datos y otros pueden proporcionar una funcionalidad añadida.

(3) **Uso del dato**: una vez descubierto y obtenido el acceso al dato, el usuario tiene que ser capaz de utilizarlo. La cantidad de formatos diferentes existentes supone un reto importante. Los formatos pueden definirse bien en la caracterización de la semántica y la estructura de los datos, o bien en el formato de los ficheros, es decir, contenido en vez de contenedor. Para ello, se han establecido algunas convenciones para la entrega de diferentes formatos de archivos.

Actualmente, los GWSs en JASMIN solo proporcionan dos funcionalidades muy limitadas con respecto a la compartición del contenido de sus proyectos [15]:

- **Compartir información sobre datos en un GWS**: la motivación que ha llevado a los GWSs a implementar esta funcionalidad se basa en que, como se ha mencionado previamente, es muy común que diferentes proyectos o científicos necesiten algún conjunto de datos perteneciente a otro proyecto en concreto. Por ello, implementan una especie de "catálogo", que es accesible a través de una simple página *web*. Esta funcionalidad no permite el acceso al dato, simplemente publica que ese conjunto de datos está disponible dentro de ese GWS. Una vez que el usuario descubre y decide acceder a esos datos, tiene que pedir acceso a ese GWS.
- **Acceso a datos vía HTTP**: a diferencia de la funcionalidad anterior, esta se basa en el acceso al dato en sí. Cuando un usuario perteneciente a un GWS cree de utilidad compartir un conjunto de datos con el resto de usuarios que no tienen acceso al sistema de ficheros de ese GWS, puede hacerlo mediante la implementación de esta funcionalidad. Aunque esto permita a los usuarios descargarse los datos, existen otro tipo de servicios de acceso al dato más eficientes y utilizados en el sector de la meteorología, como es OPeNDAP. Sin embargo, en este caso, el acceso a los datos solo está disponible a través de HTTP.

[3] Engloba todos los atributos de los datos que lo describen, proporcionan contexto, indican la calidad o simplemente documentan las características de un objeto o un dato.

Sin entrar en detalles de cómo son implementadas ambas funcionalidades, se puede apreciar que estas soluciones no resuelven los tres problemas citados anteriormente de manera eficiente. Por ello, en este trabajo se plantea el uso de THREDDS como la mejor alternativa. Se trata de un sistema que simplifica el descubrimiento y uso de datos científicos, permitiendo a los investigadores publicar, contribuir, encontrar e interactuar con los datos relacionados con el sector meteorológico de una manera efectiva e integrada. Por este motivo, en el siguiente subapartado se procede el servicio THREDDS.

### 3.3.2. THREDDS

El proyecto THREDDS (*THematic Real-time Earth Distributed Data Servers*) Unidata proporciona un servicio intermedio entre los proveedores de datos y los consumidores. THREDDS Data Server (TDS) combina los servicios de catálogo con las capacidades integradas de acceso al dato, incluyendo OPeNDAP, HTTP y servicio OGC (WCS). Además, TDS es capaz de generar catálogos automáticamente, algo que es de gran utilidad sobre todo cuando los datos se actualizan frecuentemente, como es el caso del sector de la meteorología [19].

Los catálogos THREDDS son documentos en formato XML que permiten a los proveedores de datos listar *online* los conjuntos de datos disponibles. El creador del catálogo agrupa los conjuntos de datos en un esquema de clasificación jerárquico simple que convierte un catálogo en un directorio de datos lógicos. Se trata de archivos estáticos o generados dinámicamente por servidores *web* para hacer un seguimiento de los conjuntos de datos que cambian continuamente. Como mínimo, un catálogo debe especificar un nombre legible y una URL de acceso para cada conjunto de datos. Mientras que los catálogos THREDDS por sí solos son, en general, agnósticos con respecto al significado y contenido de los conjuntos de datos, TDS trata de procesarlos utilizando un modelo común de datos (CDM). A continuación se muestra un ejemplo básico de un catálogo THREDDS:

```
<?xml version="1.0" ?>
<catalog xmlns="http://www.unidata.ucar.edu/namespaces/thredds/InvCatalog/v1.0" >
  <service name="odap" serviceType="OpenDAP" base="/thredds/dodsC/" />
  <dataset name="SAGE III Ozone 2006-10-31" serviceName="odap"
  ↪  urlPath="sage/20061031.nc" ID="20061031.nc"/>
</catalog>
```

**Código 1.** Ejemplo de un catálogo THREDDS.

Por su parte, CDM es un modelo de datos abstracto utilizado por la librería netCDF-Java y el proyecto netCDF-4 [20]. La forma y la disponibilidad de los diferentes tipos de información depende del formato de datos considerado. Además, CDM es capaz de leer conjuntos de datos con los formatos netCDF, OPeNDAP y HDF5, así como también formatos binarios como GRIB-1, GRIB-2, GINI, NEXRAD y DORADE. Finalmente, CDM también permite que la información sea accesible, sin importar la forma en que esa información ha sido obtenida.

Por otro lado, TDS integra un servidor OPeNDAP [21], el cual proporciona acceso a cualquier conjunto de datos que pueda ser leído a través de la librería netCDF del CDM. Se trata esencialmente de una adaptación de CDM en OPeNDAP utilizando la librería Java-DODS [22]. Adicionalmente, mediante la utilización de TDS se puede hacer disponible un mismo *dataset* bien vía HTTP, o bien vía OPeNDAP, sin necesidad de tener duplicada la información. En la siguiente figura se muestra un diagrama con los componentes más relevantes del servicio TDS:

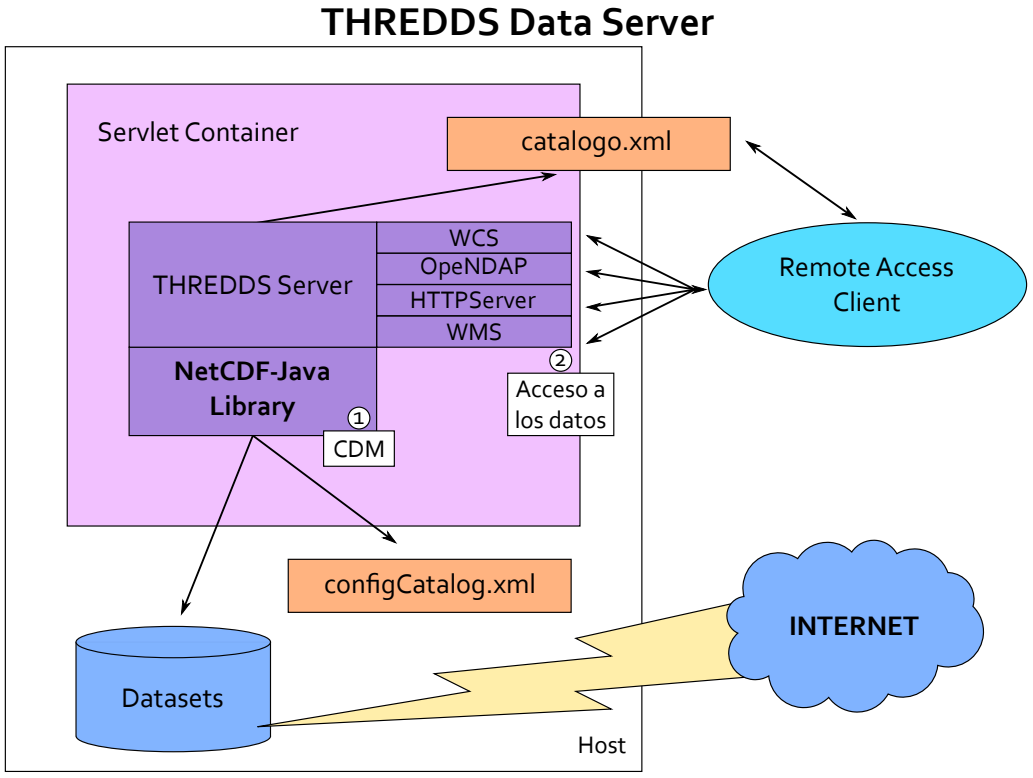



**Figura 3.4.** Diagrama de los elementos más importantes de TDS
**Fuente**: *"http://www.unidata.ucar.edu/software/thredds/current/tds/tutorial/TDSOverview.pdf"*

HTTP es el servicio más simple que ofrece THREDDS. Este, permite al usuario descargar directamente los archivos a través de un cliente HTTP. El procedimiento de descarga es muy sencillo: el usuario accede a la página *web* del fichero que le interesa. Entre los servicios de descarga disponibles, selecciona el correspondiente a HTTP y, a continuación, la descarga comienza a realizarse. También se puede acceder al archivo a través de un gestor de descargas (ejemplo: `wget`) o incluyéndolo dentro de un *script* (Matlab, Python, etc...). El mayor inconveniente que presenta HTTP es que se realiza la descarga total del fichero, esto es, no se puede descargar solo una parte de él. Debido a que el tamaño de los archivos suele ser bastante grande,en ocasiones en torno a los *gigabytes*, la utilización de HTTP puede resultar ineficiente si solo se desea una parte del fichero.

Por otro lado, OPeNDAP proporciona un *software* que permite acceder a los datos a través de Internet. Está diseñado para integrarse en otras aplicaciones y que estas lo implementen como una funcionalidad propia. Herramientas como Matlab, Octave o Python disponen de módulos que permiten obtener datos a través de este servicio. La principal

ventaja de utilizar este frente a HTTP es que se puede acceder a una parte de los datos sin necesidad de descargar el archivo completo. Es decir, se pueden filtrar los datos antes de realizar la descarga. Esta es otra de las razones por las cuales la utilización de TDS es la mejor solución para los GWSs.

Es posible utilizar TDS solo para generar los catálogos o solo para dar acceso a los datos. Sin embargo, la integración de ambas funcionalidades reduce significativamente la configuración y el mantenimiento del servidor.

Dejando a un lado los detalles de la funcionalidad de TDS, lo importante acerca de TDS en este trabajo es su despliegue. TDS es una aplicación *web* J2EE [23] en la que su contenido se encuentra en un único archivo denominado WAR (*Web Application Archive*), lo que implica que su despliegue en un servidor de aplicaciones *web* J2EE como Apache Tomcat sea realmente sencilla. Por su parte, Tomcat es un contenedor Java y un servidor *web* al mismo tiempo [24], el cual permite desplegar diferentes aplicaciones *web* entre las que se encuentra TDS. Una aplicación *web* se define como una jerarquía de directorios y archivos en un diseño estándar. Dicha jerarquía es accesible mediante archivos comprimidos con la extensión WAR. Por esta razón, en el *Capítulo 4*, relativo a la implementación del sistema, se utilizarán diferentes contenedores Tomcat para desplegar el servicio TDS.

# Capítulo 4

# Implementación técnica

En este capítulo se va a tratar todo el desarrollo que ha sido implementado con el fin de proveer un servicio THREDDS a los GWSs. Para ello se van a describir las tecnologías *software* utilizadas, el despliegue del sistema de balanceo de carga y la integración de este sistema dentro de la infraestructura JASMIN. Además, se profundizará un concepto que no se ha mencionado hasta ahora: *DevOps*, el cual toma gran importancia en la fase del despliegue de las tecnologías *software*.

## 4.1. Tecnologías utilizadas

Con objeto de alcanzar el escenario final, a continuación se procede a realizar un recorrido por las tecnologías *software* que han sido utilizadas tanto para desplegar el servicio THREDDS, como para desplegar el sistema de balanceo de carga en sí mismo.

### 4.1.1. Apache HTTP

El proyecto Apache HTTP Server (*httpd*), lanzado en el año 1995, surge para desarrollar y mantener un servidor HTTP de código abierto para diversos sistemas operativos, incluidos UNIX y Windows. Segun [25] el objetivo es proporcionar un servidor seguro, eficiente y extensible que proporcione servicios HTTP en sintonía con los estándares de existentes de HTTP.

En este trabajo se utiliza Apache HTTP Server, bien en su versión 2.2 o en la 2.4, dependiendo de las preferencias del administrador. El rol que juega este servidor dentro del sistema es el de balanceador de carga, distribuyendo el tráfico entre los diferentes clústeres[1]. En cuanto al papel que ocupa dentro de la infraestructura JASMIN, se trata de una máquina gestionada por un administrador de sistemas, con un SO CentOS 6.9, expuesta

[1]En este punto, cabe destacar que cada GWS va a formar un clúster de *workers*. Esto implica que el balanceador de carga va a tener varios clústeres configurados entre los que distribuir la carga.

a Internet, a la cual acceden los usuarios que desean utilizar el servicio THREDDS correspondiente a cada GWS. Adicionalmente, para poder implementar el balanceo de carga se pueden utilizar o bien los módulos *mod_proxy* y *mod_proxy_balancer* o bien el módulo *mod_jk*.

### 4.1.2. Apache Tomcat

El *software* Apache Tomcat no es más que un contenedor[2] de aplicaciones J2EE [23] gratuito y de código abierto. Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket son las aplicaciones J2EE que implementa [24].

Como ya se comentó en la *Sección 3.3.2*, se va a proceder a desplegar el servicio THREDDS sobre un servidor Tomcat. Por ello, en este trabajo se utiliza Apache Tomcat bien en su versión 7 o en la versión 8. La implementación de esta tecnología permite tener en un mismo servidor múltiples instancias con su propio espacio de aplicaciones y configuraciones. Esto es posible gracias a que Tomcat hace uso de dos variables de entorno que permiten separar el despliegue del propio servidor Tomcat (**$CATALINA_HOME**) de la configuración de las instancias (**$CATALINA_BASE**). El primero representa la raíz de la instalación del servidor Tomcat, mientras que el último hace referencia a la configuración específica de cada instancia Tomcat.

Si bien el nombre utilizado para referirse a cada *"servidor"* Tomcat dentro de una misma máquina es *instancia*, también se puede utilizar la palabra ***worker***, la cual será empleada a partir de este punto. Sin embargo, cabe destacar que cada instancia Tomcat puede tener configurados diferentes conectores[3]. Esto supone que cada conector dentro de una instancia es equivalente a la definición de *worker* citada previamente. Aun así, en este trabajo solo se ha configurado un conector de cada instancia Tomcat que tenga presencia en el balanceo de carga, por lo que, desde el punto de vista del *proxy*, la primera definición se ajusta perfectamente a este escenario.

En relación al rol que toma el servidor Tomcat en la infraestructura JASMIN, este consiste en una máquina con un SO CentOS 6.9 no accesible desde Internet, pero sí desde el interior de la infraestructura desplegada por el administrador de cada GWS, proporcionando el servicio TDS a dicho grupo. En cuanto al papel del servidor Tomcat en el sistema de balanceo de carga, este consiste en un conjunto de *workers*, entre los cuales el balanceador de carga distribuye el tráfico. Es decir, se trata de un clúster de *"servidores"*.

[2]Se trata de un entorno de ejecución que gestiona unos componentes, es decir, unidades de *software*.

[3]Cada conector representa un puerto diferente en el que la instancia Tomcat está escuchando para recibir peticiones.

## 4.2. Despliegue del sistema de balanceo de carga

Una vez descritas las dos tecnologías de las cuales se va a hacer uso en el sistema de balanceo y cuáles son sus roles, se procede a explicar el funcionamiento de dicho sistema.

La decisión acerca de qué tipo de sistema de balanceo de carga utilizar está basada en el contenido presente en la petición HTTP, es decir, un balanceo de nivel de aplicación, ya abordado en la *Subsección 2.3.6*. Esto es así debido a que, como se ha comentado previamente, el usuario accede al servidor *httpd* para llegar a la instancia THREDDS de cada GWS. En los *Códigos 2 y 3* se recoge un ejemplo de configuración del balanceador de carga, utilizando el módulo *mod_jk*. En este ejemplo, se hace uso de dicho módulo puesto que, una vez entendida la configuración con el *mod_jk*, configurar los módulos *mod_proxy* y *mod_proxy_balancer* resulta mucho más sencillo.

La configuración del módulo *mod_jk* se basa en tres archivos diferentes. El primero es relativo a la activación del módulo, el cual no es relevante a la hora de entender el funcionamiento, motivo por el que no se pondrá un ejemplo sobre ello. En el *Código 2* se puede ver un ejemplo de configuración del segundo archivo. Aunque estos nombres se pueden modificar, su nombre por defecto es *uriworkermap.properties*. En él se escriben las reglas de asignación del reenvío de solicitudes, es decir, a qué clúster debe reenviar la petición en función del contenido de la petición HTTP.

```
# Pagina de control para el sistema de balanceo de carga
/status-jk=jk-manager
/status-jk/*=jk-manager

# Patron URI entrante = Cluster asociado
# GWS 1
/thredds/*/gws1=GWS_1
/thredds/*/gws1/*=GWS_1

# GWS 2
/thredds/*/gws2=GWS_2
/thredds/*/gws2/*=GWS_2
```

**Código 2.** Ejemplo de configuración del fichero *uriworkermap.properties*

Además, se ha configurado un catálogo estático en el balanceador de carga, de tal manera que, cuando el usuario hace una petición a la URL *www.balanceador/thredds*, el balanceador devuelve una lista de todos los servicios TDS integrados en único catálogo. De esta manera, se le facilita al usuario el acceso al servicio TDS del GWS que desee. La configuración utilizada en el *Código 2* realiza el mapeo de la URI */thredds/*/**gwsN*** y de toda la sub-URI de */thredds/*/**gwsN**/*. El primer * utilizado en la URI corresponde al servicio de TDS que se está utilizando. Si no se especifica, TDS utiliza, por defecto, el servicio de catálogo.

Finalmente, el *Código 3* recoge la configuración del último fichero, continuando el ejemplo anterior. El nombre utilizado es, en este caso, *worker.properties*. En este, se describen los *workers* que pertenecen a un clúster específico y sus características, como por

ejemplo: dirección IP, puerto, nombre de dominio, tipo de protocolo utilizado para realizar la comunicación entre el balanceador y los *workers*, etc... Además, se activa el módulo "*status*", el cual permite realizar una gestión básica del sistema de balanceo de carga.

```
# Configurar jk-manager como tipo "status"
worker.jk-manager.type=status

# Definicion de los GWS_1 y GWS_2 de tipo "load balancing"
worker.list=GWS_1
worker.GWS_1.type=lb
worker.GWS_2.type=lb

# Definicion de los workers asociados al cluster
worker.GWS_1.balance_workers= workerA, workerB
worker.GWS_2.balance_workers= workerC, workerD

# Configuracion de los workers A, B, C y D
worker.workerA.type=ajp13
worker.workerA.host=GWS1-host
worker.workerA.port=18009

worker.workerB.type=ajp13
worker.workerB.host=GWS1-host
worker.workerB.port=28009

worker.workerC.type=ajp13
worker.workerC.host=GWS2-host
worker.workerC.port=18009

worker.workerD.type=ajp13
worker.workerD.host=GWS2-host
worker.workerD.port=28009

# Activar el jk-manager y los cluster GWS_1 y GWS_2
worker.list= jk-manager, GWS_1, GWS_2
```

**Código 3.** Ejemplo de configuración del fichero *workers.properties*

En este ejemplo, cada clúster tiene configurado 2 *workers*, entre los cuales distribuye la carga. El método que utiliza para seleccionar el mejor *worker* está basado en el método *Round Robin*, ya detallado en la *Sección 2.4*. Como se puede observar, la configuración llevada a cabo es la más sencilla posible, dejando la mayor parte de características con su valor por defecto. Sin embargo, se pueden configurar diferentes parámetros para mantener la persistencia, el número de intentos que tiene que realizar el balanceador si no puede conectarse con un *worker*, el factor de balanceo de cada *worker*; para, de esta forma, llevar a cabo una distribución basada en el método *Round Robin ponderado*, el tiempo entre *pings* para comprobar el estado de los *workers* (Subsección 2.1.7) y muchos más.

En relación al protocolo utilizado para realizar la comunicación entre el *proxy* y los *workers*, este difiere ligeramente entre los 2 módulos que se han implementado. Como se ha visto en el ejemplo, el protocolo utilizado por el módulo *mod_jk* es Apache JServ Pro-

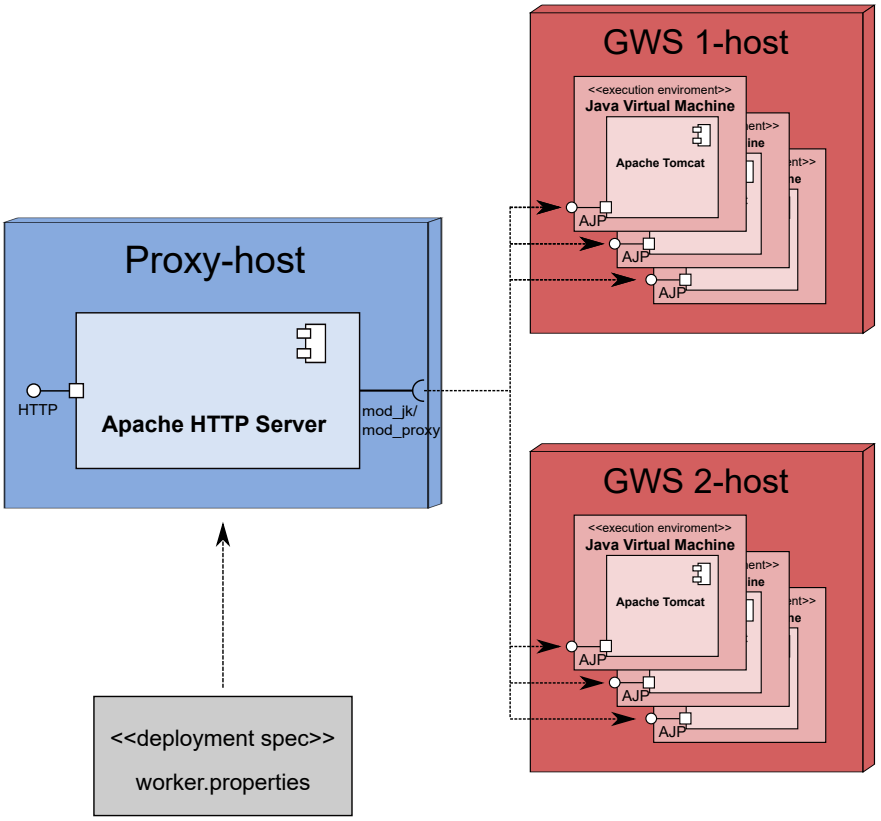



**Figura 4.1.** Ejemplo de sistema de balanceo de carga con 2 GWS configurados

tocol (AJP)[4]. Para este módulo, no existe ninguna otra posibilidad de implementar otro protocolo de comunicación. En cambio, para los módulos *mod_proxy* y *mod_proxy_balancer* se puede configurar tanto el protocolo HTTP, como el AJP o el FTP. En este trabajo se ha configurado el protocolo AJP en ambos módulos.

El protocolo AJP es un protocolo binario que trabaja sobre HTTP. Su función principal es retransmitir a otro servidor solo la parte esencial de las peticiones de HTTP. Esto hace que el envío de información entre servidores sea mucho más rápido de lo que podría ser mediante el uso del protocolo HTTP. La razón de que la comunicación sea más lenta mediante el uso del protocolo HTTP es porque HTTP está basado en los mensajes, es decir, el texto no va codificado, sino que se trata de un texto plano. Por ello, la extensión de los mensajes es mucho mayor y el *proxy* necesita más tiempo para poder analizarlo. En cambio, utilizando un protocolo binario, como es AJP, los mensajes se transmiten codificados (en formato binario 1/0), siendo su extensión mucho menor. De esta manera, se consigue una velocidad de transmisión mayor.

Además, existen algunos problemas, como el mencionado a continuación, que son gestionados de manera automática por el protocolo AJP y los conectores AJP configurados en los *workers*. Un servidor que funcione como *reverse proxy* no es totalmente transparente a la aplicación alojada en los servidores finales. Es decir, un cliente se conecta al *host* y

[4]Protocolo binario que permite enviar solicitudes desde un servidor web a un servidor de aplicaciones que se encuentra detrás del servidor web

al puerto perteneciente al servidor *reverse proxy* y no al servidor final. Para poder llegar hasta el servidor final, el *proxy* realiza la comunicación con un *host* y un puerto diferentes a los que el usuario utilizó en su petición. Cuando la aplicación del servidor final devuelve el contenido al *proxy*, este utiliza los valores correspondientes a su *host* y su puerto, los cuales no pueden ser utilizados por el usuario.

Aun así existen casos en los cuales se tienen que configurar algunas variables específicas en los conectores del servidor Tomcat para que así, el protocolo AJP funcione correctamente. Se supone el caso de tener un *reverse proxy* delante del servidor *httpd*, por ejemplo, un balanceador de carga HTTP o algo similar. En este caso, la dirección IP del cliente que ve el último servidor Apache, es la dirección IP del primer servidor *reverse proxy* y no la del cliente original. Normalmente, estos servidores generan una cabecera HTTP adicional, que contiene la dirección IP del cliente original (o una lista de direcciones IP, si hay más *proxies* inversos en cascada en frente). En esta situación, sería conveniente utilizar el contenido de esta cabecera como la dirección IP del cliente que se le pasa al servidor final. Para resolver este problema, el conector AJP de Tomcat permite configurar unos parámetros, denominados *proxyName*, *proxyPort*, permitiendo configurar correctamente el nombre del *frontend*[5].

Estas son las razones por las cuales se ha decidido implementar el protocolo AJP para realizar la comunicación entre el *proxy* y los *workers*.

## 4.3. Filosofía *DevOps*

El propio título del trabajo contiene el término *DevOps*. Sin embargo, hasta este punto aún no se había mencionado. Antes de explicar cuál es su rol dentro en este trabajo, se va a describir brevemente en que consiste esta filosofía y qué herramienta se va a utilizar a la hora de implementar este trabajo.

### 4.3.1. Contexto

*DevOps* es una metodología, método, proceso o manera de pensar, que extiende la filosofía *ágil* para producir rápidamente productos y servicios *software*. De esta manera se puede mejorar el rendimiento y calidad en las operaciones. Integra dos mundos, el desarrollo software (Dev) y los despliegues *software* en las infraestructuras operacionales (Ops), utilizando para ello un desarrollo, despliegue e infraestructura de gestión y de monitorización automatizado [26].

Si bien se ha menciona la filosofía *ágil*, es necesario explicar cuales son los fundamentos de dicha filosofía. Esta nace como respuesta a las maneras tradicionales del desarrollo *software*. Su principal objetivo es ver como incrementar la eficiencia y la flexibilidad de la gestión de un proyecto. En el *"Agile Manifesto"*hay 12 principios básicos que describen esta filosofía [27]:

[5]Parte del software que interactúa con los usuarios.

**(1)** Satisfacer al cliente a través de entregas tempranas y continuas del *software*.

**(2)** Aceptar que los requisitos cambien, incluso en etapas tardías del desarrollo.

**(3)** Entregar *software* funcional frecuentemente.

**(4)** Los empresarios y los desarrolladores trabajan juntos de forma cercana durante todo el proyecto.

**(5)** Los participantes en un proyecto tienen que ser personas motivadas. Para ello se les tiene que proporcionar el entorno y el apoyo que necesitan y confiar en ellos para la consecución de ello.

**(6)** El método más eficiente y efectivo de comunicar información, al equipo de desarrollo y entre sus miembros, son reuniones presenciales físicamente.

**(7)** El funcionamiento del *software* es la métrica principal del progreso.

**(8)** Mediante la utilización de esta filosofía se promueve el continuo desarrollo, es decir, los participantes tienen que ser capaces de mantener un ritmo constante de forma indefinida.

**(9)** Para mejorar la agilidad es necesario prestar una atención continua a la excelencia técnica y al buen diseño.

**(10)** Es esencial buscar la simplicidad en el desarrollo.

**(11)** Los mejores proyectos surgen de equipos auto-organizados.

**(12)** El equipo se debe reunir periódicamente para reflexionar sobre cómo ser más efectivo para ajustar y perfeccionar su trabajo.

*DevOps* surge para resolver el conflicto existente durante el despliegue, entre el desarrollo y la gestión de operaciones. De esta manera, se pueden reducir o evitar algunos problemas que existen en la industria *software*. Según [28], algunos de esos problemas son:

- **Miedo al cambio**: los usuarios suelen ser reacios a realizar algún cambio una vez que la aplicación está funcionando.
- **Despliegues arriesgados**: en el proceso del despliegue de *software*, existe la preocupación de si este funcionará correctamente en un entorno real.
- **Jugar a culpar**: cuando entre los desarrolladores es detectado un problema, se empieza el juego de echar la culpa al otro entre todos ellos.
- **Separación**: el equipo de los proyectos suele estar dividido en dos partes:

  **(1)** **Equipo de desarrollo**: programadores, evaluadores y controladores de calidad.

  **(2)** **Equipo de operaciones**: administradores de bases de datos, administradores de sistemas, administradores de red y operadores.

### 4.3.2. Fases y herramientas

Como era de esperar, para llevar a cabo la filosofía *DevOps* es necesario tener un conjunto de herramientas. Y además, es muy importante elegir la herramienta correcta dependiendo del entorno y el proyecto. Dependiendo de la fase *DevOps* en la que se encuentre, existen diferentes herramientas de desarrollo. Según [29], existen diferentes fases de un proceso *DevOps*:

- **Construcción**: el proceso comienza por organizar los entornos para el desarrollo. En esta fase, las herramientas deben soportar flujos de trabajo rápidos. Existen dos tipos de herramientas:
  1. **Herramientas de construcción**: gestionan el desarrollo *software* y el ciclo de vida del servicio. La utilidad de estas herramientas está en la automatización de tareas comunes de desarrollo como la compilación de código fuente en código binario o la creación de ejecutables. (Ejemplo: *Apache Ant*, *Rake*, *Gradle*)
  2. **Herramientas para la integración continua**: integran el código del desarrollador y comprueban si existe algún error. De esta manera, el sistema se prueba constantemente. (Ejemplo: *Jenkins*, *TeamCity*, *Bamboo*)
- **Despliegue**: en esta fase, la infraestructura debe ser tratada como código (IaC), es decir, el entorno de operación se administra de la misma manera que se hace con el código de las aplicaciones. (Ejemplo: *Puppet*, *Chef*, *Ansible*)
- **Operación**: aunque las dos fases anterior son muy importantes, no se debe olvidar esta fase ya que se necesitan herramientas para mantener la estabilidad y el rendimiento de la infraestructura. Se puede distinguir entre dos tipos de herramientas diferentes:
  1. **Herramientas de registro**: los registros se utilizan para analizar el rendimiento del sistema y las tendencias de uso. Además, los desarrolladores encargados de realizar el proceso de depuración utilizan los datos que se encuentran en estos registros. (Ejemplos: *Loggly*, *Graylog*, *Splunk*)
  2. **Herramientas para la monitorización**: permiten a las organizaciones identificar y resolver los problemas en las infraestructuras IT antes de que afecten de manera crítica a los procesos. (Ejemplos: *Nagios*, *New Relic*, *Cacti*)

### 4.3.3. Fase de despliegue

Una vez explicadas las fases de la filosofía *DevOps*, se va a explicar más en detalle la fase de despliegue ya que es la más acorde al trabajo que se va a desarrollar.

La infraestructura como código (IaC) es uno de los principios de *DevOps*. Se refiere a la gestión de configuración automatizada. Permite ahorrar tiempo al no realizar manualmente tareas repetitivas. Además, proporciona fiabilidad. Por lo general, cada vez que se despliega un nuevo servidor, se necesita repetir la misma tarea que se ha hecho,

anteriormente, en otro servidor. Antes del concepto de IaC, este proceso se realizaba manualmente. Sin embargo ahora, todo lo que se necesita es escribirlo en un código y ejecutarlo en ese servidor. IaC hace que la configuración de los entornos sea más dinámica, proporcionando así, los servicios más rápidamente a los clientes.

Todas las herramientas de esta fase son herramientas de "*gestión de la configuración*", es decir, están diseñadas para instalar y administrar *software* en servidores existentes. En comparación con la provisión de infraestructura manual, estas herramientas pueden reducir la complejidad de mantenimiento de las configuraciones. La entrega continua mejora el tiempo de entrega y permite prácticas *ágiles* con una rápida reacción de los consumidores. Dichas herramientas mejoran el ciclo de vida de las aplicaciones.

#### *Puppet*

La primera herramienta es *Puppet*. Permite definir los estados de la infraestructura. Se compone de un servicio maestro y servicios agentes en cada cliente. *Puppet* se puede ejecutar de manera autónoma, pero lo normal es utilizar el servidor maestro. El servidor maestro se conecta a una base de datos que mantiene el seguimiento del sistema y del *software* que hay en él. Su función principal es proporcionar información de configuración necesaria a todos los clientes dentro del sistema. Los clientes informan al maestro de su estado actual, para que así el maestro pueda tener un seguimiento del estado de todo el sistema. Esta herramienta automatiza las tareas que un administrador de sistemas realiza normalmente de manera manual. El administrador solo necesita entrar en el servidor máster y realizar los cambios que crea oportunos. Dependiendo de la configuración que tengan los clientes, estos cambios llegarán a ellos de una manera u otra. En el modelo "*pull*", los agentes instalados en los clientes se conectan al servidor maestro para preguntar por la nueva configuración, mientras que en el modelo "*push*", el maestro tiene la capacidad de depositar la información de configuración en los clientes en cualquier momento. La sintaxis que utiliza *Puppet* está basada en Ruby.

#### *Chef*

La segunda herramienta es Chef. Se trata de una plataforma de gestión de configuración y automatización de Opscode. Es una plataforma poderosa de automatización que transforma una infraestructura compleja en código, sin importar su tamaño. El método *Chef* es muy similar a *Puppet* en cuanto a que existe un servidor maestro y varios agentes instalados en los nodos a gestionar. Sin embargo, difieren en que en la instalación de Chef se requiere una estación de trabajo para controlar el servidor maestro.

#### *Ansible*

La última herramienta, pero no menos importante, es *Ansible*. Es la más diferente a las dos anteriores ya que no necesita instalar agentes en cada cliente. Se configura el sistema con un modelo autónomo. Este modelo se basa en: desde una máquina de control, conectarse a los servidores con mediante SSH. Por ello, *Ansible* utiliza un método "*push*".

Se trata de una herramienta de código abierto. Aunque actualmente da soporte para Windows, inicialmente *Ansible* no fue diseñado con ese fin, por lo que su implementación en servidores Windows no es estable. Su utilización esta pensada para utilizarlo en terminales OSX, Linux o Unix, ya que se necesita Python 2.4 en los nodos [30]. El lenguaje de gestión de *Ansible* se escribe en unos *scripts* denominados *playbooks*. Están escritos en formato YAML y describen el estado en el que el sistema debe estar. En los *playbooks* se escribe el estado en el que se desea que se encuentre el sistema y después *Ansible* transforma el sistema al estado descrito en dichos *playbooks*. Cada *playbooks* consiste en una o más tareas descritas en una lista. Las tareas descritas en los *playbooks* se ejecutan secuencialmente sobre el grupo de nodos definidos. A estas tareas se las llama también "modulos de *Ansible*". La mayor parte de estos módulos son idempotentes, es decir, aunque se ejecute varias veces el mismo *playbook*, este no realiza ningún cambio en el sistema si este ya se encuentra en dicho estado. *Ansible* proporciona un sistema simple con dependencias mínimas. Además, a diferencia de las dos herramientas anteriormente explicadas, no mantiene un control sobre el estado del servidor una vez se ha terminado de ejecutar los *playbooks*. Esto hace que *Ansible* sea la mejor opción para el desarrollo de nuestro trabajo.

## 4.4. Integración de la filosofía *DevOps* en la Computación *Cloud*

Hoy en día es vital para muchos proveedores de *software*, especialmente en el campo de aplicaciones *web* y aplicaciones de SaaS, implementar frecuente y continuamente actualizaciones de una aplicación. Esto se debe a que los usuarios y los clientes esperan que las nuevas características y correcciones de errores estén disponibles lo más rápido posible. La utilización de una filosofía *DevOps* con la computación *cloud* permite el aprovisionamiento rápido y bajo demanda de recursos subyacentes, como por ejemplo servidores virtuales, almacenamiento, etc. [31].

Este trabajo no se centra tanto en la entrega de nuevas versiones de un producto, sino más bien en el aprovisionamiento[6] de una máquina virtual que se ofrece al usuario como un SaaS. Como se explicó en la *Subsección 3.1.2*, el SaaS se basa en aplicaciones preconstruidas e integradas que se entregan a los clientes como un servicio, permitiendo al usuario olvidarse de la instalación del servicio, pero proporcionándole la libertad de configurar el servicio como crea conveniente.

En la *Figura 4.2* se puede ver la relación entre la filosofía *DevOps* y la computación *cloud* existente en este trabajo. En primer lugar, el administrador del GWS que decida integrar el servicio THREDDS en su grupo, accede al portal *web* de JASMIN *cloud*. Una vez solicite la creación de una máquina virtual, se inicia un proceso de aprovisionamiento de la máquina. En este paso, que se corresponde con el paso 2 en la *Figura 4.2*, se ejecuta un *playbook* de *Ansible* de manera automática, desplegando el servicio THREDDS con su/s instancia/s Tomcat correspondiente/s. Cuando la configuración de la VM finaliza, el administrador del GWS accede a la VM mediante una conexión SSH para configurar[7] el

[6] En este contexto, el término aprovisionar significa configurar un entorno de desarrollo.

[7] Los administradores de los GWS son personas que tienen un conocimiento en la utilización de este

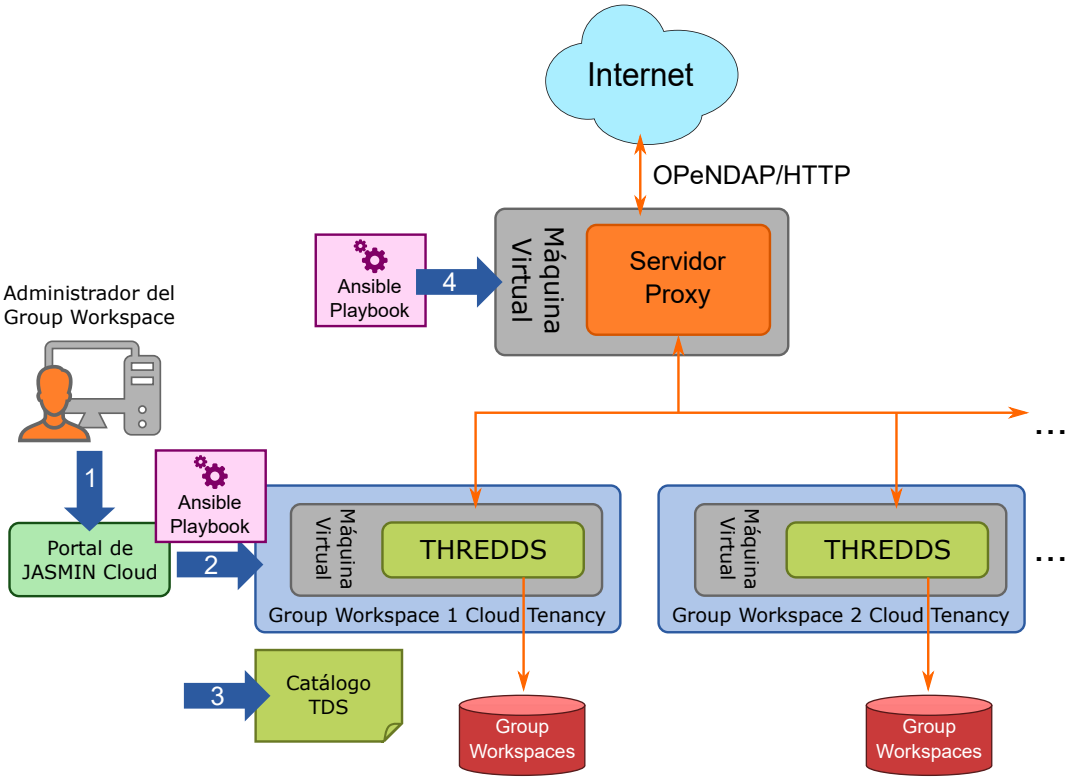



**Figura 4.2.** Proceso de solicitud del servicio THREDDS para un GWS dado

servicio THREDDS de la manera que crea más conveniente para su grupo. En este punto, el servidor *proxy* no tiene conocimiento de que ese GWS tiene dado de alta el servicio THREDDS. Por eso, el siguiente y último paso, se basa en que el administrador del GWS actualice esta información en el *proxy*. El problema que surge en este momento consiste en realizar estos cambios teniendo en cuenta que dicho administrador no dispone de acceso al servidor *proxy*[8].

Por este motivo, la solución implementada es la siguiente: el administrador del *proxy* proporciona al administrador[9] del GWS un acceso restringido a la VM del *proxy*, de tal manera que solo le permita ejecutar un *script* alojado en este *host*. De este modo, el administrador del *proxy* se asegura de que no pueda realizar ningún cambio no deseado en la configuración de la máquina. El servidor *proxy*, al igual que los servidores Tomcat, es creado a partir de un *playbook* de *Ansible*. En ese proceso, se crean 3 archivos de configuración diferentes, los cuales se enumeran y describen a continuación:

1. *Playbook* de *Ansible* denominado *update_proxy_GWS.yml*. Se encarga de dar de alta el clúster en los archivos de configuración, bien empleando el módulo *mod_jk* o bien el *mod_proxy* (sin dar de alta los *workers*).

2. *Playbook* de *Ansible* denominado *update_proxy_TDS.yml*. Se encarga de dar de alta a los *workers* asociados a su GWS.

servicio.

[8]Por motivos de seguridad, la única persona que tiene acceso a esta VM es el administrador del sistema.

[9]Estos administradores son usuarios pertenecientes a la comunidad de JASMIN

3. *Script* de Python denominado *update_proxy.py*. Este último es el *script* que el administrador del GWS puede ejecutar al realizar la conexión SSH. Dentro de este archivo se realiza, en primer lugar, un filtrado de los parámetros que el administrador pasa mediante el comando SSH y, por último, se realiza una llamada a los dos *playbooks* de *Ansible*[10] citados anteriormente.

A continuación, el administrador del GWS, que se encuentra conectado a su VM, ejecuta un *script* de Python (*Código 5*), creado en el proceso de aprovisionamiento de la máquina. Este *script* lee el fichero *tomcat_instances.yaml* (ver *Código 4*), también creado durante el proceso de aprovisionamiento, el cual contiene toda la información necesaria para configurar el clúster en el servidor *proxy* (*host*, puerto, nombre de la instancia, GWS al que pertenece, etc.).

```
- name: workerA
    base_port: 1800
    ip_address: GWS1-host
    ajp:
      port: 18009
      proxyName: Proxy-host
      proxyPort: 8008
    gws_instance:
      - name: GWS1
- name: workerB
    base_port: 2800
    ip_address: GWS1-host
    ajp:
      port: 28009
      proxyName: Proxy-host
      proxyPort: 8008
    gws_instance:
      - name: GWS1
```

**Código 4.** Ejemplo del fichero *tomcat_instances.yaml* para el *host* del GWS1

Una vez esta información es leída, se le pregunta al administrador, instancia a instancia, qué acción quiere realizar en el servidor *proxy*. Las acciones que se han implementado son: añadir el *worker*, borrarlo o no hacer nada. Por último, el *script* realiza la conexión SSH al servidor *proxy*, pasándole los parámetros necesarios para realizar la configuración. Adicionalmente, se pensó en la idea de dar la opción al administrador de elegir la acción que quería llevar a cabo ya que, aunque en un primer momento se quieran añadir todas las instancias desplegadas, puede darse el caso en el que alguna de ellas tenga algún tipo de error y el administrador quiera eliminarla del sistema de balanceo de carga.

[10]Se utilizan *playbooks* de *Ansible* en vez de otro tipo ya que, como se comentó en la *Sección 4.3.3*, los módulos de *Ansible* comprueban el estado en el que se encuentra el *host*, por lo que solo realiza cambios si el estado al que se le quiere llevar es diferente al estado en el que se encuentra.

```
#!/usr/bin/python
import [...]

def update_proxy(proxyName, options):
  paramiko.util.log_to_file('paramiko.log')
  try:
    ssh = paramiko.SSHClient()
    ssh.load_system_host_keys()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname = proxyName, username='root')
    try:
      stdin,stdout,stderr = ssh.exec_command(command="%s" %options)
      outlines=stdout.readlines()
      resp=''.join(outlines)
      print(resp)
  except [...]
    sys.exit(1)

def get_content():
  with open("tomcat_instances.yaml", 'r') as info:
    try:
      info_dict = yaml.load(info)
      for instance in info_dict:
        tomcat_instance = instance
        workerName = (" --workerName " + tomcat_instance["name"] + " ")
        workerPort = ("--workerPort " + str(tomcat_instance["ajp"]["port"]) + " ")
        proxyName = tomcat_instance["ajp"]["proxyName"]
        if "ip_internal_address" in tomcat_instance:
          workerHost = ("--workerHost " + tomcat_instance["ip_internal_address"] + " ")
          ip_address = tomcat_instance["ip_internal_address"]
        elif "ip_address" in tomcat_instance:
          workerHost = ("--workerHost " + tomcat_instance["ip_address"] + " ")
          ip_address = tomcat_instance["ip_address"]
        for gws in instance["gws_instance"]:
          gwsName = ("--gwsName " + gws["name"])
          action = raw_input("Action for %s tomcat instance: %[···])
          if action == "addWorker":
            options = action + workerName + workerPort + workerHost + gwsName
          elif action == "removeWorker":
            options = action + workerName + gwsName
          else:
            break
          update_proxy(proxyName,options)
    except yaml.YAMLError as exc:
        print(exc)

def main():
  print "Help:"
  print "Action = addWorker | removeWorker (by default: Nothing to do) \n"
  get_content()
if __name__ == "__main__":
  main()
```

**Código 5.** *Script update_proxy.py* alojado en el *host* del GWS1

Una vez finaliza este proceso, el sistema de balanceo de carga ya estaría implementado sobre la infraestructura JASMIN.

## 4.5. Cambios en la infraestructura JASMIN

En la *Sección 3.2* se explicó la infraestructura JASMIN, así como también los servicios *cloud* que ofrecía. Además, se citó la importancia que esta tenía en el trabajo a desarrollar. Por esta razón, es necesario mencionar los cambios que supone la implantación del servicio THREDDS y del sistema de balanceo de carga dentro de dicha infraestructura. Para ello, en la *Figura 4.3* se muestra la nueva configuración de la infraestructura JASMIN.

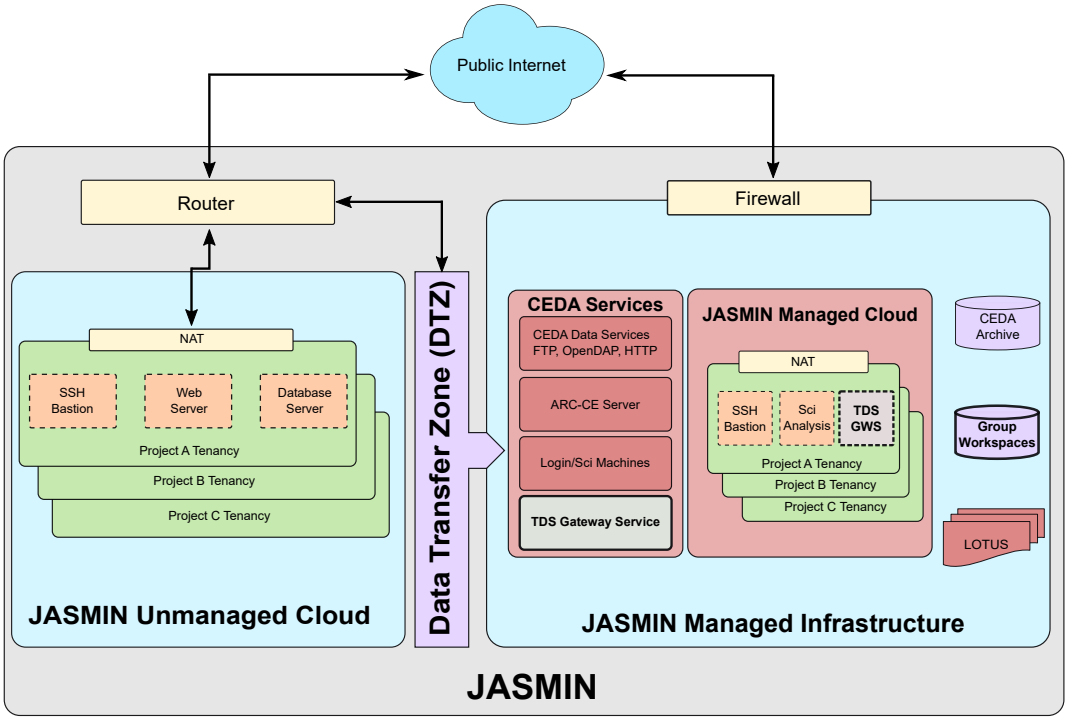



**Figura 4.3.** Propuesta de la nueva infraestructura JASMIN

En primer lugar, se puede apreciar la nueva plantilla creada en la parte JASMIN *Managed Cloud*, denominada *TDS GWS*. Esta tiene como finalidad desplegar el servicio THREDDS para los GWSs. Aunque el proceso de creación de laVM se ha descrito en la sección anterior, cabe destacar que, cuando los administradores acceden al portal *web* de JASMIN *cloud*, estos tendrán disponibles tres plantillas diferentes: *SSH bastion*, *Sci Analysis* y *TDS GWS*. Sin embargo, recordando la explicación sobre las plantillas existentes inicialmente (ver *Subsección 3.2.2*) y la descripción sobre cómo se ofrece esta nueva plantilla (ver *Sección 4.4*), existe una diferencia entre ellas. Esta radica en el tipo de servicio *cloud* que se ofrece al usuario. Mientras que las primeras plantillas ofrecen al usuario una VM como una PaaS, esta última se ofrece como un SaaS.

Además, recordando las diferencias entre JASMIN *managed cloud* y JASMIN *unmanaged cloud*, estas radican en los privilegios que se le otorgan al usuario que crea la VM

y el alcance que esta tiene al sistema de almacenamiento basado en *Panasas*. De hecho, estas son las principales razones por las cuales se ha decidido crear la plantilla en la nube administrada en vez de en la no administrada. Debido a que el servicio THREDDS está orientado a los GWSs, las VMs necesitan tener acceso al sistema de ficheros basado en *Panasas*, en el cual se encuentra almacenada toda la información de los proyectos de cada GWS. Sin embargo, los usuarios no tienen acceso como *root* a las VMs. Por ello, en el proceso de aprovisionamiento de la VM, mediante el *playbook* de *Ansible*, es necesario ejecutar el servicio THREDDS con los privilegios adecuados. Es decir, si el servicio se ejecutase como usuario *root*, el administrador no sería capaz de pararlo, ejecutarlo y realizar los cambios necesarios. Por otro lado, es necesario destacar también que las VM pertenecientes a este *cloud* no son accesibles desde Internet, sino solo desde dentro de la red de JASMIN.

Para solventar este último problema, se propone añadir a los servicios del CEDA ya existentes, un servicio denominado "Pasarela TDS" (o "TDS *Gateway*" en la *Figura 4.3*). De esta manera, tanto los usuarios de Internet como los que tengan una VM en la nube no administrada, pueden acceder al servicio THREDDS de cada GWS. Como se puede deducir, el servicio "TDS *Gateway*" no es más que el servidor *reverse proxy* encargado de distribuir las peticiones entrantes entre los *workers*. Es decir, utilizando los términos utilizados en la *Figura 4.3*, este se encarga de agrupar todas las *tenancy* que se han creado mediante la plantilla *TDS GWS*.

Por último, se muestra en la *Figura 4.4* la integración del sistema de balanceo de carga (ver *Figura 4.1*) sobre la infraestructura JASMIN (ver *Figura 4.3*).

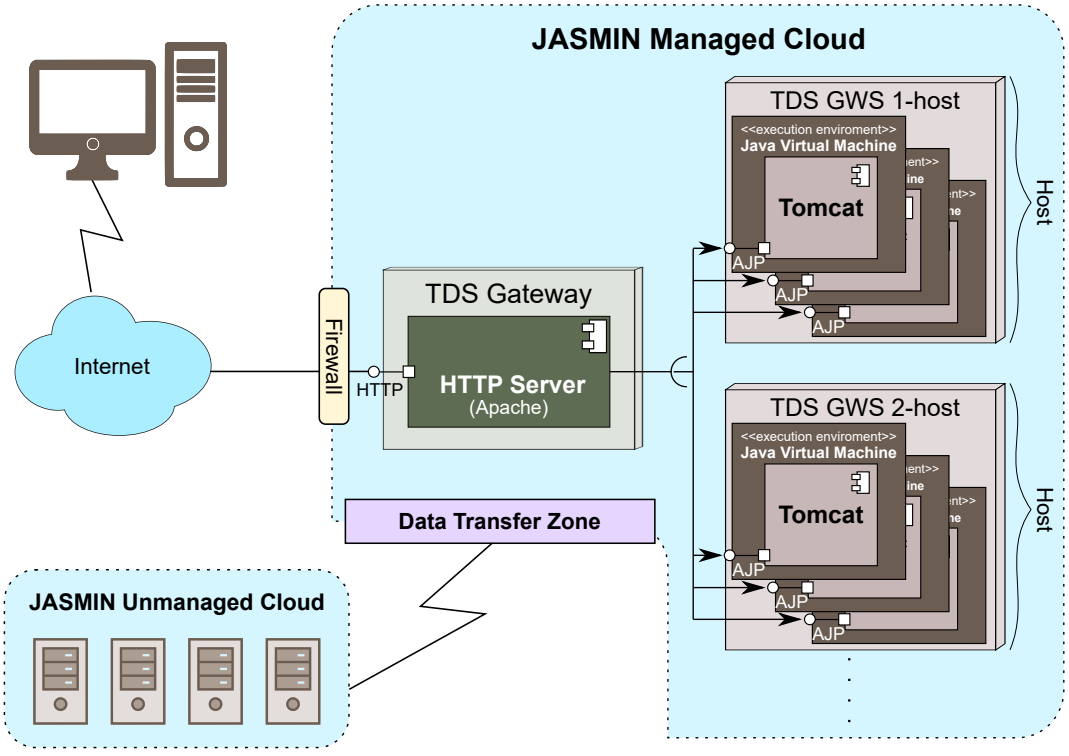



**Figura 4.4.** Sistema de balanceo de carga en la infraestructura JASMIN

A pesar de la descripción realizada en este apartado, es necesario aclarar que dichos cambios aún no se han llevado a cabo. Es decir, este trabajo no ha alcanzado todavía la fase de producción. Esto es debido a que la fase de construcción se ha implementado en una *tenancy* dedicada al uso exclusivo de pruebas. Por ello, el trabajo aquí desarrollado tiene que pasar todavía por la fase de evaluación, en la cual, el sistema opere en un entorno real. Aún así, la idea inicial para implantar el sistema es la descrita en este apartado.

## 4.6. Desarrollo del código mediante *Ansible*

Aunque el objetivo desarrollado en este trabajo ha sido la implantación del sistema en la infraestructura JASMIN, el meta-objetivo no es solo este, si no una solución genérica en la que se pueda desplegar un sistema de balanceo de carga con el servicio TDS implementado en cualquier otro escenario. Para ello, el trabajo realizado se ha puesto a disposición de la comunidad en un repositorio de *GitHub*[11] [32].

Adicionalmente a lo que se comentó en la *Subsección 4.3.3*, *Ansible* se trata de una herramienta modular, es decir, implementa mecanismos que permiten descomponer tareas complicadas en otras más simples. Su principal mecanismo es conocido como *role*, el cual simplifica la creación de *playbooks* complejos y facilita la reutilización del código. El concepto *role* se debe utilizar cuando un conjunto de tareas relacionadas entre sí se quieren desplegar en diferentes *hosts*.

Además, como se mencionó en el *Capítulo 1*, la utilización de la herramienta *Ansible* ha permitido la creación de un código de propósito general en vez de uno específico para este escenario. Esto ha sido posible gracias al uso del concepto *role*. Para ello, se han creado cuatro roles con propósitos bien diferenciados:

**(1)** *common*: utilizado para la creación de un entorno virtual en el cual se instala un servicio de control de procesos conocido como *supervisord* [33], además de otras dependencias necesarias para el funcionamiento del sistema. Este rol se activa cuando el desarrollo se quiere implementar en un entorno sin acceso *root*.

**(2)** *httpd*: es el encargado de instalar y configurar el servidor *reverse proxy*.

**(3)** *tds*: tiene como finalidad desplegar y realizar una configuración básica del servicio THREDDS.

**(4)** *lib*: se trata de un rol que agrupa tareas comunes a los 3 roles anteriores, como por ejemplo comprobar que un servicio está instalado, seleccionar las variables específicas de la versión que se quiere desarrollar, etc.

[11] Plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

Por otro lado, para el desarrollo del código, se han tenido en cuenta tres casos de uso diferentes:

**(1)** Utilizar solo los paquetes disponibles en el repositorio de *RedHat/CentOS*, además de realizar el control de los procesos mediante el servicio de inicio de sistema del SO. La utilización de esta opción implica que el usuario tiene privilegios de acceso *root*.

**(2)** Este se trata del mismo caso anterior pero se diferencia en que los servicios son instalados desde los archivos binarios en vez de desde el repositorio oficial.

**(3)** En este caso, se utilizan los archivos binarios pero se integra el sistema de control de procesos implementado en el rol *common*, es decir, el *supervisord*. La elección de esta opción implica que el usuario no tiene por que tener privilegios *root*, por lo que todo se despliega en el *home* del usuario.

Para seleccionar entre un caso de uso u otro, el usuario no tiene más que definir unas variables en el *playbook* que desarrolle. En los *Códigos 6 y 7*, se recogen ejemplos de los *playbooks* que despliegan el primer y tercer caso de uso:

```
- name: Deploy httpd
    hosts: host-proxy
    vars_files:
      - proxy-secret.yml
    vars:
      load_balancer_module: mod_proxy
    roles:
      - role: httpd
```

**Código 6.** *Playbook* para el despliegue de un servidor *httpd* con el primer caso de uso

```
  - name: Deploy httpd
    hosts: host-proxy
    vars_files:
      - proxy-secret.yml
    vars:
      virtualenv_name: TDS4GWS
      httpd_install_from_source: true
      httpd_system: false
      httpd_version: '2.2.32'
      httpd_port: 8001
    roles:
      - role: httpd
```

**Código 7.** *Playbook* para el despliegue de un servidor *httpd* con el tercer caso de uso

Como se observa, no existe gran diferencia entre un código y otro. En función de los parámetros que configure el usuario, *Ansible* ejecuta internamente unos roles u otros y ,a su vez, estos presentan diferentes configuraciones. Es decir, existen algunas variables que definen los diferentes casos de uso, mencionados anteriormente, pero también existen otras variables que varían la configuración dentro de un mismo rol. Este es el caso de la variable *load_balancer_module* (ver *Código 6*), la cual configura el conector utilizado en la comunicación entre el servidor *reverse proxy* y los *workers*. Aunque en el *Código 7* no se especifique dicha variable, se han descrito unas variables para todas aquellas susceptibles de ser configuradas por el usuario. En este caso, la variable por defecto para *load_balancer_module* es la configuración del módulo *mod_jk*. Lo mismo ocurre con las variables *virtualenv_name*, *httpd_version* y *httpd_port*.

Por otro lado, otro de los parámetros importantes dentro de *Ansible* es el correspondiente a los *hosts*. Este se trata de un documento en el cual se detallan las características de las máquinas donde se quieren ejecutar los *playbooks*. De hecho, este es el archivo que realmente diferencia unos escenarios de otros. En el siguiente código se presenta una configuración del archivo *hosts* utilizado por *Ansible* en la fase de pruebas:

```
gws1-tds4gws ansible_ssh_host=192.168.33.12 ansible_ssh_user=root
gws2-tds4gws ansible_ssh_host=192.168.33.13 ansible_ssh_user=root

host-proxy ansible_ssh_host=192.168.33.10 ansible_ssh_user=root

[proxy]
host-proxy

[tds-hosts]
gws1-tds4gws
gws2-tds4gws
```

**Código 8.** Ejemplo del archivo *hosts* utilizado en la fase de pruebas

Atendiendo a estos ejemplos, se puede apreciar la versatilidad que ofrece el uso de la herramienta *Ansible*. De esta manera, ha sido posible crear un código de propósito general, portable a cualquier escenario que se quiera llevar a cabo. Es decir, gracias al uso de la filosofía *DevOps*, junto con los beneficios de las infraestructuras tipo *cloud*, se convierte este trabajo/producto en algo "elástico", es decir, escalable y reproducible.

# Capítulo 5

# Líneas futuras

Una vez descrito todo el proceso experimental que ha sido llevado a cabo a lo largo del trabajo, así como todos los conceptos teóricos ligados al mismo, es turno de presentar algunas líneas futuras susceptibles de ser exploradas. Aunque el objetivo inicial del trabajo abarcaba también la realización de las líneas definidas en este último capítulo, estas se plantean como puntos futuros que continúen con el trabajo aquí desarrollado.

Así, las líneas futuras aquí planteadas surgen de la realización de un estudio del rendimiento de la red, el cual evite degradar la calidad final percibida por el usuario incluso en un momento de alta demanda en el sistema. Así, a través de dicho estudio, se pretenden cubrir los objetivos que se enumeran y describen en las siguientes líneas:

**(1)** Determinar el número óptimo de *workers* a implementar dentro de un clúster. De esta forma, será posible llegar a desarrollar un sistema dinámico en el que los *workers* se crean o eliminan en función de la carga de tráfico existente en cada momento.

**(2)** Definir el número de servidores *reverse proxy* para que el balanceador no se convierta en un cuello de botella.

**(3)** Precisar el escenario más adecuado para el servidor *reverse proxy*, es decir, la utilización de un escenario activo/pasivo o activo/activo.

Con todo ello, se propone como continuación de este proyecto la evaluación de los puntos previamente mencionados, con el fin de trasladar la implementación aquí realizada a un escenario práctico de uso.

Finalmente, también se plantea la migración del sistema desarrollado en el CEDA al Servicio de Datos Climáticos de la Universidad de Cantabria. Esto supone enlazar dicho sistema con los recursos computacionales existentes en el grupo, así como también con el TAP (*THREDDS Access Portal*) [34]. Este último se basa en la integración de servicios de autenticación frente al servicio TDS, permitiendo la restricción del acceso a este mediante la definición de grupos de usuarios.

# Bibliografía


[1] J. Tramullas. Propuestas de concepto y definición de la biblioteca digital. 2002.

[2] Y. Jiao and W. Wang. Design and implementation of load balancing of distributed-system-based web server. In *2010 Third International Symposium on Electronic Commerce and Security*, pages 337–342, July 2010.

[3] Descripción de la disponibilidad, la confiabilidad y la escalabilidad. https://technet.microsoft.com/es-es/library/aa996704(v=exchg.65).aspx.

[4] N. Briscoe. Understanding the OSI 7-layer model. *PC Network Advisor*, 120(2), 2000.

[5] M. García Arranz. *Fundamentos de la Transmisión de Datos*. Apuntes de clase, Universidad de Cantabria, 2014-2015. Asignatura Comunicación de Datos.

[6] T. Bourke. *Server load balancing*. O'Reilly, Beijing ; Sebastopol, Calif, 1st ed edition, 2001.

[7] Real academia española. http://www.rae.es/rae.html.

[8] W. Tarreau. Making applications scalable with load balancing. *Willy Tarreau's articles*, 2006.

[9] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen. SRLB: The Power of Choices in Load Balancing with Segment Routing. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2011–2016. IEEE, 2017.

[10] Y. Tao and G. Chen. An Extensible Universal Reverse Proxy Architecture. In *2016 International Conference on Network and Information Systems for Computers (ICNISC)*, pages 8–11, April 2016.

[11] F. Fatemi Moghaddam, M. B. Rohani, M. Ahmadi, T. Khodadadi, and K. Madadipouya. Cloud computing: Vision, architecture and Characteristics. In *2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC)*, pages 1–6, August 2015.

[12] P. Mell, T. Grance, and others. The NIST definition of cloud computing. 2011.

[13] S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[14] B. N. Lawrence, V. Bennett, J. Churchill, M. Juckes, P. Kershaw, P. Oliver, M. Pritchard, and A. Stephens. The JASMIN super-data-cluster. *arXiv preprint arXiv:1204.3553*, 2012.

[15] Group Workspaces | JASMIN. http://www.jasmin.ac.uk/services/group-workspaces/.

[16] Overview | JASMIN. http://www.jasmin.ac.uk/services/.

[17] Introduction to the JASMIN Cloud - Centre for Environmental Analysis (CEDA) and JASMIN Help Docs Site. http://help.ceda.ac.uk/article/282-introduction-to-the-jasmin-cloud.

[18] e-Science All Hands Meeting, Simon J Cox, and Engineering and Physical Sciences Research Council, editors. *Proceedings of UK e-science all hands meeting: Nottingham 2-4 September 2003*. The Engineering and Physical Sciences Research Council (EPSRC), Swindon, 2004.

[19] J. Caron, E. Davis, Y. Ho, and R. Kambic. *THREDDS Data Server (TDS) version 4 and 5 [software]*. Unidata, Boulder, CO, 2017.

[20] *Network Common Data Form (NetCDF) version 4 [software]*. Unidata, Boulder, CO, 2017.

[21] J. Gallagher, N. Potter, T. Sgouros, S. Hankin, and G. Flierl. The Data Access Protocol - DAP 2.0. Standard ESE-RFC-004.1.1, NASA, October 2007.

[22] FunctionLibrary (Java-DODS Documentation). https://www.opendap.org/pub/dods/DODS-Java-1.1/1.1.3/Java-DODS/doc/dods/dap/Server/FunctionLibrary.html.

[23] Java Platform, Enterprise Edition (Java EE) | Oracle Technology Network | Oracle. http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[24] Apache Tomcat. https://tomcat.apache.org/.

[25] The Apache HTTP Server Project. https://httpd.apache.org/.

[26] M. Rajkumar, A. K. Pole, V. S. Adige, and P. Mahanta. DevOps culture and its impact on cloud delivery and software development. In *2016 International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring)*, pages 1–6, April 2016.

[27] M. Molhanec. Agile Product Design - A modern approach to quality improvement. In *Proceedings of the 36th International Spring Seminar on Electronics Technology*, pages 178–182, May 2013.

[28] A. Wahaballa, O. Wahballa, M. Abdellatief, H. Xiong, and Z. Qin. Toward unified DevOps model. In *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 211–214, September 2015.

[29] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. DevOps. *IEEE Software*, 33(3):94–100, May 2016.

[30] L. Hochstein. *Ansible: Up and Running*. O'Reilly.

[31] J. Wettinger, V. Andrikopoulos, and F. Leymann. Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications. In *2015 IEEE International Conference on Cloud Engineering*, pages 60–65, March 2015.

[32] ansible-thredds-cluster: Ansible playbooks for deployment of THREDDS server cluster. https://github.com/SantanderMetGroup/ansible-thredds-cluster, July 2017.

[33] Supervisor: A Process Control System — Supervisor 3.3.3 documentation. http://supervisord.org/.

[34] UDG-TAP Home. http://meteo.unican.es/udg-tap/home.