Diseño de una Arquitectura Dinámica para Set-Top Box Multi Proveedor de Servicios

Iván Bernabé Sánchez

en cumplimiento parcial de los requisitos para el grado de
Doctor en
Ingeniería Telemática

Universidad Carlos III de Madrid

Director:
Daniel Díaz Sánchez

Fecha deposito Junio 2021

Esta tesis se distribuye bajo licencia "Creative Commons
Atribución-NoComercial-CompartirIgual"

CONTENIDOS PUBLICADOS Y PRESENTADOS

Esta tesis doctoral contiene contenidos publicados y presentados en varias contribuciones anteriores. A continuación se muestran las publicaciones del autor las cuales han sido mencionadas durante la tesis. Estas publicaciones tienen un cierto grado de de relación con la tesis debido a que algunas de ellas han sido el origen del trabajo desarrollado de esta tesis y han sentado la base del trabajo de investigación. A continuación se va a mostrar un listado de las publicaciones junto a los autores, título, el año de publicación y el nombre de revista o conferencia, los capítulos en donde se localiza cada publicación yel rol del autor para cada publicación.

A. Publicaciones en revistas indexadas en el JCR
En este apartado se presentan los artículos que han sido publicados en alguna revista.

1. Iván Bernabé-Sánchez, Daniel Díaz-Sánchez y Mario Muñoz-Organero, **"Specification and Unattended Deployment of Home Networks at the Edge of the Network"**, IEEE Transactions on Consumer Electronics (Volume: 66, Issue: 4, Nov. 2020). DOI: 10.1109/TCE.2020.3018543

    - Esta contribución está incluida completamente en la tesis, en los capítulos 5, 6, 7 y 9.
    - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura general. También diseñó los mecanismos y modelos de información necesarios para el correcto funcionamiento de los componentes que componen la arquitectura. EL desarrollo del prototipo, su ejecución y el análisis realizado a partir de las medidas tomadas forman parte del autor. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
    - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.

B. Publicaciones en congresos
En este apartado se presentan las publicaciones que han sido publicados en algún congreso o conferencia de ámbito internacional.

1. Iván Bernabe Sanchez; Daniel Diaz Sanchez y Mario Muñoz-Organero, **"Optimizing OSGi Services on Gateways"**, Ambient Intelligence-Software and Applications, Springer International Publishing, 2013. p. 155-162. DOI: 10.1007/978-3-319-00566-9_20
    - Esta contribución está incluida completamente en la tesis, en el capítulo 4.
    - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura de distribución de contenidos propuesta. El autor también propuso los mecanismos de optimización de software para dispositivos limitados. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
    - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.
2. Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero, **"Flex-box: A flexible software architecture for IPTV set-top boxes"**, Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on. 10.1109/ICCE-Berlin.2012.6336480, 2012 , p. 121 - 125. DOI: 10.1109/ICCE-Berlin.2012.6336480
    - Esta contribución está incluida completamente en la tesis, en el capítulo 4.
    - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura multiproveedor de servicios. El autor también propuso el modelo de datos semántico que permite registrar la información de los bloques funcionales. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
    - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.
3. Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero, **"Optimizing resouces on gateways using OSGi"**, 2012 International Conferences on Consumer Electronics. Las Vegas, January 2012. DOI: 10.1109/ICCE.2012.6161957.
    - Esta contribución está incluida completamente en la tesis, en el capítulo 4.
    - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal del diseño de los mecanismos de

optimización de software para dispositivos limitados. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.

- El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.

C. Publicaciones en congresos nacionales

En este apartado se presentan las publicaciones que han sido publicados en algún congreso o conferencia de ámbito nacional.

1. Iván Bernabé, Daniel Díaz-Sánchez y Mario Muñoz-Organero, **"La Nube y su Papel como Plataforma de Distribución de Contenidos de TV"**, JARCA Conference, Rota, June 2014. ISBN: 978-84-606-6085-9.
   - Esta contribución está incluida completamente en la tesis, en el capítulo 4.
   - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
   - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.
2. Ivan Bernabe Sanchez; Daniel Diaz Sanchez; Mario Munoz-Organero, **"Arquitectura IPTV para STB Basada en Capas de Control Jerárquicas"**, JARCA 2013, p. 9.
   - Esta contribución está incluida parcialmente en la tesis, en el capítulo 4.
   - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
   - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.
3. Iván Bernabé, Mario Muñozl, **"An OSGi platform for Eco-Driving systems"**, Actas JARCA 2012. pp. 11-16. ISBN 978-84-616-2007-4 2012. Junio 2012. Salou.
   - Esta contribución está incluida completamente en la tesis, en el capítulo 4.

- El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
- El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.

4. Iván Bernabé Sánchez, Daniel Díaz Sánchez, Mario Muñoz Organero, **"Using XMPP to Federate Multimedia UPnP Device"** Jornadas JARCA 2011 , 27, 28 y 29 de Junio de 2011 , Huelva
   - Esta contribución está completamente incluida en el capítulo 4 de esta tesis.
   - El rol del autor de esta tesis en esta publicación fue la contribución en la idea principal y diseño de la arquitectura de distribución de contenidos propuesta. El autor también fue el responsable de liderar la escritura del articulo y fue el contribuidor principal de su escritura.
   - El material de esta fuente incluida en la tesis no está señalado por medios tipográficos ni referencias.

OTROS MÉRITOS DE INVESTIGACIÓN

Parte del contenido se desarrolló como línea de investigación en varios proyectos de I+D, denominados:

- RAUDOS2 Red Interactiva Multiplataforma de Distribución de Contenidos Audiovisuales (TSI-020302-2010-67) (Proy. Nacional)
- HAUS-Hogar digital y contenidos Audiovisuales adaptados a los USuarios (IPT-2011-1049-430000) (Proy. Nacional)
- IRENE - Incentivación del reciclaje de envases con NFC en España (PT-2012-1036-370000) (Proy. Nacional)
- COMINN - Centro comercial interactivo con interacción natural (IPT-2012-0883-430000) (Proy. Nacional)
- REMEDISS - Red médica social sensorizada (IPT-2012-0882-430000)
- OSAMI-Commons Open Source Infrastructure Ambient Inteligence Commons (TSI-020400-2009-92) (Proy. Internacional)

ABSTRACT

El ecosistema de servicios digitales disponibles para los dispositivos domésticos ha estado creciendo durante los últimos años. Ese crecimiento no solo es causado por el aumento significativo de los contenidos digitales disponibles, sino también por el aumento de los dispositivos conectados a Internet que son capaces de consumir estos contenidos. La mayoría de estos dispositivos suelen ser dispositivos móviles y otros dispositivos personales de consumo los cuales se están convirtiendo en los clientes preferidos. Actualmente, la mayoría de los servicios dirigidos a dispositivos heterogéneos son servicios Over-the-top (OTT). Estos servicios OTT permiten que varios dispositivos consuman el mismo contenido, ya que no hay necesidad de una infraestructura de servicio subyacente. Sin embargo, estos servicios funcionarían mejor utilizando una infraestructura dedicada que proporciona una mayor experiencia de usuario mientras que requiere menos recursos. Tomando en cuenta este panorama se puede pensar en un futuro escenario caracterizado por una gran diversidad de dispositivos, de usuarios potenciales y un gran número de servicios disponibles por lo que sería conveniente disponer de infraestructuras fiables y robustas y además poder dar soporte a cualquier tipo de servicio.
Dado que la infraestructura física puede ser costosa de implementar, lo más lógico sería considerar que este ecosistema de servicios digitales va a requerir el uso de infraestructuras dinámicas y escalables. Afortunadamente, el paradigma del Cloud Computing, y sus alternativas como el Edge Computing y el Fog Computing, permiten crear instancias dinámicas de servicios y elementos de red haciendo frente a los requisitos antes mencionados mientras reduce los costes de inversión y operación. Sin embargo, el paradigma de cloud computing tiene varios problemas que requieren especial atención como: Data Lock-In (o bloqueo de proveedor) y la interoperabilidad entre proveedores de infraestructura.
Hoy en día, los orquestadores de computación en la nube son bastante heterogéneos y carecen de interfaces estándar que permitan realizar despliegues automatizados. Por lo tanto, una definición de infraestructura diseñada para un proveedor de infraestructura en la nube determinado es más que probable que no se pueda crear una instancia en otros. Por lo tanto, se requieren mecanismos de configuración flexibles, administración avanzada, procesamiento descentralizado y auto-organización.
En esta tesis, los autores proponen una arquitectura que amplía las funciones de red definidas en las especificaciones y estándares de NFV para

incorporar funcionalidades de mayor nivel. El objetivo es ampliar la virtualización de las funciones de red con la creación de instancias de servicios en dispositivos de consumo. Los autores proponen ampliar la idea de NFV en lugar de proponer un nuevo diseño porque NFV tiene una gran aceptación en la industria y por eso contribuiremos ampliando la arquitectura NFV. De esta manera, los proveedores de servicios tendrán un control adicional del software del sistema al mover algunos servicios, tradicionalmente ubicados en los dispositivos, a la nube.
Uno de los mayores problemas encontrados para extender la idea al mayor número de infraestructuras se debe a las particularidades de cada una de ellas que las hacen diferentes. Para superar esto, proponemos un modelo estructurado de información para declarar componentes, conexiones y configuraciones, que permite a la solución de arquitectura propuesta en la tesis crear instancias de un determinado conjunto de servicios en diferentes proveedores de infraestructuras en la nube. Para que esta configuración pueda ser aplicada a varios niveles en las infraestructuras, este trabajo propone utilizar un modelo segmentado por capas de software. Por lo tanto, los orquestadores pueden implementar e interconectar partes de software de una manera sencilla, independientemente del proveedor del entorno.
En esta tesis se propone un modelo arquitectónico que permite construir un ecosistema virtual en entornos domésticos que requieren muchas y complejas tareas de configuración. El uso de mecanismos automáticos de gestión de software garantiza la correcta configuración y despliegue de servicios en diferentes plataformas acelerando el proceso y reduciendo la probabilidad de cometer errores de configuración. A continuación se presentan las principales contribuciones en este contexto:

- Diseño de un ecosistema digital doméstico e integrado.
- Modelo de arquitectura flexible y adaptable.
- Automatización en la configuración de infraestructuras y despliegue de servicios.
- Optimización de dependencias y del despliegue de servicios software.
- Modelado de la información del software de infraestructura.
- Orquestación del software basado en herramientas de la IaC.

En resumen, el trabajo presentado en esta tesis contribuye a avanzar en el desarrollo de los ecosistemas domésticos digitales con el objetivo de proporcionar a los proveedores de servicios nuevas herramientas que les haga más fácil la gestión de los sistemas a un precio más bajo. Los consumidores

de los servicios podrán gestionar de manera sencilla ecosistemas digitales donde la frontera entre lo virtual, los servicios contratados y sus dispositivos domésticos y móviles forman un todo integrado y perfectamente orquestado.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE CUADROS

ACRÓNIMOS

AMI Aplicación de Mensajeria Instantánea

API Application Programming Interface

APS Application Provisioning Service

ARM Azure Resource Manager

AS Servicio Analizador

AWS Amazon Web Services

BME Bare Metal Evaluator

BS Broker de Servicios

CACS Cloud Application Catalogue Service

CAMP Cloud Application Management for Platforms

CAPEX Capital Expenditure

CCE Cloud Computing Evaluator

CD Centro de Datos

CE Container Evaluator

CeCE Centralized Computing Evaluator

CIMI Cloud Infrastructure Management Interface

CLI Command Line Interface

CMS Control and Monitoring Service

CNCF Cloud Native Computing Foundation

CORD Central Office Re-architected as a Datacenter

CPE Customer Premises Equipment

CSAR ToSCA Cloud Service Archive

DACS Device Application Catalogue Service

DBaaS Data Base as a Service

DCE Distributed Computing Evaluator

Dev Device

DLNA Digital Living Network Alliance

DMTF Distributed Management Task Force

DNS Domain Name Service

DS Servicio de Distribución

DSL Domain Specific Language

DU Deploy Unit

DUL Deploy Unit layer

EMS Sistema de Gestión de Elementos

ETSI European Telecommunication Standards Institute

GUI Graphical User Interface

HGC Herramientas de Gestión de Configuracion

HL Host Layer

HOT Heat Orchestration Template

IaaS Infrastructure as a Service

IaC Infrastructure as Code

IL Interface Layer

IMOM Infrastructure Monitoring and Management system

IMS IP Multimedia Subsystem

INFORD INFrastructure Orchestration and Deployment system

IS Ingenieros del Software

ISS Infrastructure Selection Service

JSON JavaScript Object Notation

KVM Kernel-based Virtual Machine

LDS Location Discriminator Service

LNM Local Network Manager

LSE Location Selection Service

LXC Linux Containers

MaaS Metal as a Service

MANO MANagement and Orchestration

MCS MicroServices Container Service

MEC Mobile Edge Computing

NAS Network Attached Storage

NAT Network Address Translation

NFV Network Function Virtualization

NFVI NFV Infraestructure

NFVO NFV Orchestrator

NL Node Layer

NMS Network Manager Service

OASIS Organization for the Advancement of Structured Information Standards

OCCI Open Cloud Computing Interface

OGF Open Grid Forum

ONOS Open Networking Operating System

OPEX OPerational EXpenditures

OSGi Open Services Gateway initiative

OSM Open Source MANO

OSS/BSS Operations/Business Support System

OTT Over-The-Top

OVF Open Virtualization Format

P2P Peer to Peer

PaaS Platform as a Service

PAC Private Access Channel

PDP Platform Deployment Package

PDS Platform Discriminator Service

PL Platform Layer

QoS Quality of Service

RGW Residential GateWay

RTP Real-time Transport Protocol

SaaS Service as a Service

SC Servicio de Gestión de la Conexión

SDMV Servicio de Dispositivo Multimedia Virtual

SDN Software Defined Network

SeCI Service Configuration Interface

SGIC Servicio de Gestión de Intercambio de Contenidos

SIP Session Initiation Protocol

SO Sistema Operativo

SOA Service Oriented Architecture

SS Servicio de almacenamiento

STB Set-Top Box

STUN Simple Transversal de UDP sobre NATs

SVC Scalable video coding

SVS Smart Video Service

SyCI System Configuration Interface

TDS Technologies Discriminator Service

TI Tecnologias de la Informacion

TOSCA Topology and Orchestration Specification for Cloud Applications

TSS Tecnology Selection Service

TURN Traversal Using Relay NAT

UPnP Universal Plug and Play

VE VPS Evaluator

VIM Virtual Infrastructure Manager

VM Virtual Machine

VMM Virtual Machine Manager

VNF Virtual Network Functions

VNFM Virtual Network Function Manager

VPS Virtual Private Server

VUN Red Virtual UPnP

XaaS Anything as a Service

XML eXtensible Markup Language

XMPP eXtensible Messaging and Presence Protocol

YAML Yet Another Markup Language

1 INTRODUCCIÓN

Este capítulo tiene el objetivo de poner en contexto al lector sobre la organización que se ha seguido para este documento de tesis. En primer lugar, en la sección 1.1 se presenta la motivación que ha servido para iniciar el trabajo de investigación de esta tesis. A continuación en la sección 1.2 se describen los objetivos establecidos al inicio del trabajo y las tareas realizadas para conseguirlos. En la sección 1.4 se comentan los diferentes capítulos que componen el documento de tesis y el trabajo realizado en cada uno de ellos. Finalmente, en la sección 1.5 se describen las actividades de investigación y difusión realizadas a través de participación en proyectos de investigación nacionales e internacionales y las contribuciones a congresos y revistas con los resultados obtenidos de la investigación realizada.

1.1 MOTIVACIÓN

El ecosistema de servicios digitales disponible para dispositivos de consumo ha estado creciendo durante años. Por ejemplo, el streaming de vídeo ha seguido creciendo y se espera que su tráfico alcance el 78 % del tráfico total consumido por los consumidores de Internet en 2021 [1]. Ese crecimiento no sólo es causado por el aumento significativo del contenido digital disponible, sino también por el aumento de los dispositivos conectados a Internet que son capaces de consumir estos contenidos, además hay que tener en cuenta que los dispositivos móviles y otros dispositivos de consumo personales se están convirtiendo en los clientes preferidos [1]. Actualmente, la mayoría de los servicios dirigidos a dispositivos heterogéneos son servicios Over-the-top (OTT), por lo que los datos se envían al usuario a través de Internet a diferencia de otros mecanismos de transmisión basados en Real-time Transport Protocol (RTP), Scalable Video Coding (SVC), etc. los cuales no son apropiados para funcionar sobre Internet debido a que requieren información del estado de la red para gestionar la calidad de la transmisión a diferencia de OTT que sí es capaz de hacerlo [2]. Sin embargo, los servicios OTT permiten que varios dispositivos sin configuraciones especiales consuman el mismo contenido, ya que no hay necesidad de una infraestructura de servicio subyacente. Sin embargo, estos servicios proporcionarán una mejor calidad utilizando una infraestructura dedicada que repercutirá en una mayor experiencia de usuario [2].

Un escenario futuro, caracterizado por una gran diversidad de dispositivos con un gran número de usuarios potenciales y multitud de servicios disponibles, requiere utilizar infraestructuras robustas y fiables capaces de soportar la combinación de diferentes mecanismos de transmisión por servicio. Dado que la infraestructura física es costosa de implementar, es sensato tener en cuenta que este ecosistema de servicios digitales necesitará usar infraestructuras dinámicas y escalables. Afortunadamente, el paradigma de Cloud Computing, y sus diferentes variantes como Edge Computing y Fog Computing, proporcionan una infraestructura de este tipo permitiendo crear instancias dinámicas de servicios y elementos de red que hace frente a los requisitos antes mencionados reduciendo los costes de inversión y operación. Sin embargo, el paradigma de Cloud Computing tiene varios problemas que requieren una atención especial como privacidad y Data Lock-In[1] [3]. En cuanto a la privacidad, deben desarrollarse nuevos mecanismos que garanticen que los datos de los usuarios no se utilicen de forma incorrecta una vez cargados en la nube. El problema del bloqueo de datos es especialmente preocupante con respecto a la implementación dinámica de la infraestructura.

A pesar de las ventajas de la computación en la nube y de su desarrollo, el creciente número de dispositivos virtualizados en la nube complica su administración ya que por lo general cuanto mayor es el número de servicios y software a gestionar la probabilidad de cometer errores y de que surjan problemas es mayor. Por lo tanto, se requieren mecanismos de configuración flexibles, administración avanzada, procesamiento descentralizado y auto-organización. La distribución de los servicios entre la nube y los dispositivos permite trasladar las tareas de procesamiento a infraestructuras de procesamiento más adecuadas ya que proporcionan una alta disponibilidad de los servicios desplegados sobre ellas, permiten adaptarse a la demanda de cada momento, los costes son más bajos, etc. De esta manera, las tareas que exigen altos recursos informáticos se descargan en la nube, manteniendo los servicios básicos, como la interfaz gráfica de usuario (GUI) en los dispositivos que se conectan a esos servicios.

Un modelo que permite realizar offloading[2] permite distribuir dispositivos de consumo con menos recursos computacionales y, por tanto, más baratos y más pequeños, lo que reduce los costes de transporte y el consumo

1 El Data Lock-In o bloqueo de datos, es el correspondiente en datos al problema de Vendor Lock-In, en el que una entidad utiliza una tecnología de terceros particular realizando un esfuerzo económico y/o humano cuyo resultado no es posible usarlo con otras tecnologías de otros fabricantes.

2 Offloading se refiere comúnmente a la capacidad de trasladar trabajos concretos a otros lugares para descargar al dispositivo de ciertas tareas. Puede hacerse por rendimiento (en los casos en que la infraestructura que los realiza tanga un mayor rendimiento) e incluso para ahorrar recursos locales como memoria, cpu o batería.

de energía (ya que las tareas complejas se ejecutan en la nube). Además, convertir los dispositivos en versiones más simples, tiene sus ventajas ya que no requiere software complejo para controlarlos y por lo tanto es más fácil realizar tareas comunes de actualización y mantenimiento incrementando su seguridad y reduciendo los tiempos de desarrollo.

Una manera de gestionar el software sobre la nube es a través de orquestadores [4]. Hoy en día, los orquestadores de los sistemas de información en la nube son bastante heterogéneos y carecen de interfaces estándar fiables que permitan despliegues automatizados independientemente del proveedor de infraestructura. Por lo tanto, una definición de infraestructura diseñada para un framework diseñado para un entorno en la nube concreto, es probable que no pueda ser instanciada en otros entornos en la nube por problemas de compatibilidad. Esto se debe a que los proveedores de infraestructura en la nube están optimizados según sus propios criterios e intereses lo que condiciona apostar más por un tipo de tecnologías que por o otros o dar más importantes a algunos aspectos de configuración lo que condiciona la configuración de las plataformas y frameworks que proporcionan a sus clientes.

En esta tesis, el autor propone una arquitectura y mecanismos de despliegue y configuración que amplía las funciones de red definidas en las especificaciones y estándares de NFV [5] para la incorporación de nuevas características. Entre los objetivos de la investigación se encuentra el de ampliar la tecnología de virtualización de las funciones de red integrando esta con la creación de instancias de servicios para dispositivos de consumo. En esta tesis se propone ampliar la idea de NFV en lugar de proponer un nuevo diseño debido a que NFV tiene una gran aceptación en la industria y por lo tanto se va a proponer la expansión de la arquitectura NFV.

El trabajo desarrollado en esta tesis proporciona mecanismos de administración de servicios a los proveedores de servicios que les facilita el despliegue y la gestión de cualquier dispositivo. Además, propone mecanismos para incorporar de manera sencilla funcionalidades desplegadas normalmente sobre dispositivos físicos e incluso desplegar el software del propio dispositivo sobre una infraestructura **proporcionando un control adicional del software del sistema moviendo algunos servicios, tradicionalmente ubicados en los dispositivos, a la nube**. El problema principal al que se enfrenta esta tesis es la posibilidad de poder desplegar aquellas funcionalidades sobre múltiples infraestructuras lo que necesita crear mecanismos que proporcionen la independencia respecto a la infraestructura base.

Como solución se propone una arquitectura capaz de configurar infraestructuras a medida sobre la cual poder desplegar arquitecturas de servicios en la cual los proveedores de servicios podrán desplegar sus servicios. Esta

tesis se basa principalmente en tecnologías de virtualización[3] y en el paradigma de la Infraestructura como Código (IaC) como tecnologías claves.

Como la solución propuesta en este trabajo contempla la configuración de diferentes niveles sobre las infraestructuras, este trabajo propone utilizar un modelo segmentado por capas de software. El modelo de capas software[4] en este trabajo se entiende como una pila de 4 capas apiladas una encima de otra donde cada una de ellas se corresponde con algún nivel de infraestructura. La capa más abajo llamada Capa de Infraestructura representa a los recursos de una infraestructura, la capa inmediatamente encima de la capa de Infraestructura contiene los elementos relacionados con máquinas virtuales y sistemas operativos. La siguiente capa se corresponde con el software de infraestructura y frameworks y finalmente la última capa que se compone de los servicios software de interés para los proveedores de servicios. Agrupar los elementos involucrados en cada una de las capas es importante ya que permite gestionarlos a través de mecanismos adecuados para ellos y así obtener configuraciones óptimas de las que se beneficien los elementos de la capa inmediatamente superior. Una capa bien configurada y optimizada aporta beneficios de rendimiento y fiabilidad a las capas superiores.

El objetivo del diseño de las capas es para que el orquestador de la arquitectura presentada en este trabajo pueda organizar su trabajo dependiendo de la capa que vaya a configurar en cada momento facilitando a los orquestadores implementar e interconectar piezas de software de una manera sencilla independientemente del proveedor del entorno. El modelo de capas software también incluye el uso de mecanismos de virtualización para garantizar implementaciones software en varias infraestructuras. La autoconfiguración también es interesante para este trabajo, ya que proponemos introducir el concepto de ecosistema virtual en entornos domésticos y estos entornos requieren muchas y complejas tareas de configuración. El uso de mecanismos automáticos de gestión de software garantiza la correcta configuración y replicación de servicios en diferentes plataformas. También acelera el proceso. No usar estos mecanismos podría tomar más tiempo para completar cada proceso y cometer errores de configuración.

1.2 OBJETIVOS

El principal objetivo de la investigación propuesta en esta tesis esta orientada a desarrollar una arquitectura programable capaz de instanciar servicios y dispositivos multimedia sobre múltiples infraestructuras. El trabajo

3 Más adelante, en la sección 2.2 se discutirá las ventajas y problemas de cada una de las tecnologías de virtualización y cuales son las más adecuadas para cada tipo de aplicación

4 En la sección 6 se describe este modelo

va más allá de una simple arquitectura ya que también propone mecanismos capaces de configurar los recursos sobre múltiples infraestructura para posteriormente adaptar los entornos de ejecución sobre los cuales se despliegan los servicios requeridos. Por lo tanto, para lograr este objetivo se han desarrollado las tareas siguientes:

1. Realizar un estudio del arte de las tecnologías actuales utilizadas para desplegar software. Entre ellos se incluyen tecnologías de virtualización, herramientas orientadas a la Infraestructura como Código (IaC) y proyectos de investigación.
2. Analizar investigaciones actuales, estándares desarrollados y propuestas en desarrollo sobre el despliegue de servicios y software en entornos en la nube.
3. Analizar mecanismos de interoperabilidad sobre diferentes entornos en la nube dependientes de aspectos técnicos.
4. Diseñar mecanismos capaces de gestionar tecnologías de desarrollo para facilitar el despliegue y adaptación a los diferentes entornos de la nube.

1.3 PLAN DE DESARROLLO

Esta sección describe la metodología de trabajo llevado a cabo para la elaboración de la tesis doctoral. A continuación se muestran las principales tareas realizadas:

1. Obtener la bibliografía relacionada con las arquitecturas de servicios, los sistemas de distribución de contenidos, las herramientas de gestión de configuración y proyectos actuales relacionados con la gestión dinámica de servicios con el objetivo de estudiar y analizar sus problemas.
2. Diseñar y proponer una arquitectura con los elementos necesarios para solucionar los problemas identificados en el paso 1.
3. Desarrollar las pruebas necesarias para verificar el modelo de arquitectura propuesto en el paso 2.
4. Obtener las conclusiones desde las medidas obtenidas en la fase de pruebas realizadas.
5. Identificar futuras líneas de investigación.
6. Escribir y publicar artículos científicos para dar a conocer los avances y resultados obtenidos en las fases anteriores

1.4 ORGANIZACIÓN DE LA TESIS

Con el objetivo de alcanzar los objetivos comentados en las secciones anteriores, la organización de esta tesis se distribuye de la siguiente manera:

El **capitulo 1** se presentan los aspectos introductorios al trabajo realizado en la tesis y a la organización de la misma.

El **capitulo 2** presenta el estado del arte de las tecnologías implicadas en esta tesis. La idea es dar una visión general de cada una de estas tecnologías, analizar sus capacidades y sus límites ya que en el capítulo siguiente se mostrará cómo se complementan unas con otras para formar la solución final. El capítulo comienza dando una visión de las actuales arquitecturas de servicios, luego comentan las tecnologías de virtualización, posteriormente analizará el concepto de la softwarización y su implicación en los sistemas actuales. Finalmente, analiza aspectos de la computación en la nube además de tecnologías y estándares para gestionar los recursos de infraestructura de los proveedores de entornos en la nube.

A continuación, el **capítulo 3** analiza la Infraestructura como Código (IaC) para facilitar software en la nube. El capítulo presenta una introducción a la IaC y presenta una categorización sobre las herramientas del mercado que pueden ser aplicadas a la IaC y qué herramientas son las más adecuadas dependiendo del software a configurar. El capítulo presenta análisis de: 1) las diferentes herramientas de gestión de la configuración; 2)herramientas para la orquestación de recursos de infraestructuras; 3) herramientas para la orquestación de servicios NFV y 4) proyectos que pueden incluir algunas de las herramientas incluidas en las categorías enumeradas y que les permiten proporcionar funcionalidades más completas.

El **capítulo 4** presenta una primera aproximación de arquitectura de servicios que mejora los sistemas de distribución de contenidos multimedia basados en dispositivos con tecnologías UPnP. La arquitectura permite extender el dominio desde redes locales a redes overlay. Parte del trabajo mostrado en este capítulo fue publicado en [6]. Además, se proponen mecanismos para optimizar el software desplegado sobre cada dispositivo. Estos mecanismos de optimización fueron publicados en [7] y [8].

El **capítulo 5** propone una solución arquitectónica respecto a los problemas descritos en el capítulo 2, en el que se integran ideas innovadoras junto con las tecnologías descritas en el capítulo 2 y 3. Básicamente se presenta la arquitectura principal de la solución, los mecanismos de gestión que configurarán el funcionamiento del sistema y los módulos que permitirán optimizar las configuraciones resultantes. Esta arquitectura se ha publicado en la revista [9].

El **capítulo 6** propone mecanismos que permiten configurar los recursos y los diferentes niveles software que se pueden encontrar en una infraes-

tructura en la nube orientada a servicios. Estos mecanismos se basan en un modelo de capas software que ayudan a organizar el software a través de niveles formando así una pila de capas software las cuales se apilan unas sobre otras. Este modelo de capas facilita la labor de las herramientas de orquestación las cuales realizarán las tareas correspondientes en las capas a configurar. Parte de estos mecanismos se publicaron en la revista [9].

El **capítulo 7** realiza la validación de los capítulos 5 y 6 a través de la realización de un prototipo que implementa parte de la arquitectura mostrada en el capítulo 5 y los mecanismos de configuración mostrados en el capítulo 6. El prototipo se ejecuta correctamente y como resultado se muestran las medidas realizadas y se realiza un análisis de ellos.

El **capítulo 8** muestra unos mecanismos avanzados para optimizar el despliegue de servicios sobre diferentes modelos de arquitecturas de servicios. El capítulo presenta algunas configuraciones de parámetros y de funcionamiento que son aplicados a diferentes escenarios de aplicación y durante el desarrollo de esta tesis se han registrado. Como resultado de estas observaciones se ha diseñado un componente de optimización y un modelo de información que permite proporcionar configuraciones y recomendaciones sobre los despliegues de los servicios.

El **capítulo 9** presenta un prototipo para validar el componente de optimización presentado en el capítulo 8. El capítulo muestra como un mismo servicio puede ser desplegado sobre diferentes entornos tecnológicos e infraestructuras y en el texto se presentan las medidas tomadas en cada uno de estos entornos tecnológicos. El análisis de estas mediciones se publicaron en [9]. Además, el capítulo también presenta una validación general del trabajo realizado a lo largo de la tesis a través de la verificación de los requisitos cumplidos. Estos requisitos se establecieron en el capítulo 2 y han servido de guía durante el diseño de las diferentes soluciones propuestas en la tesis.

Finalmente, el **capítulo 10** presenta las conclusiones obtenidas después de la investigación realizada durante la tesis y los posibles trabajos futuros que podrían realizarse a partir de los resultados obtenidos durante este trabajo.

1.5 ACTIVIDADES DE INVESTIGACIÓN Y DIFUSIÓN RELACIONADAS

En cuanto a la publicación y difusión de los resultados obtenidos en el desarrollo de la tesis, parte del contenido se desarrolló como línea de investigación en varios proyectos nacionales de I+D que se enumeran en la sección 1.5.1.

Además, la difusión también se llevó a cabo mediante la publicación de artículos científicos. Los principales artículos que apoyan el interés de

la investigación presentada en esta tesis se detallan a continuación en las secciones 1.5.2, donde se enumeran las revistas y las secciones 1.5.3 y 1.5.4 donde se enumeran los congresos. En donde para cada contribución, se va a explicar brevemente el tipo de publicación, la fecha de publicación y su contenido.

1.5.1 *Participación en proyectos de I+D*

En cuanto a la publicación y difusión de los resultados obtenidos en el desarrollo de la tesis, parte del contenido se desarrolló como línea de investigación en varios proyectos de I+D, denominados:

- RAUDOS2 Red Interactiva Multiplataforma de Distribución de Contenidos Audiovisuales (TSI-020302-2010-67) (Proy. Nacional)
- HAUS-Hogar digital y contenidos Audiovisuales adaptados a los USuarios (IPT-2011-1049-430000) (Proy. Nacional)
- IRENE - Incentivación del reciclaje de envases con NFC en España (PT-2012-1036-370000) (Proy. Nacional)
- COMINN - Centro comercial interactivo con interacción natural (IPT-2012-0883-430000) (Proy. Nacional)
- REMEDISS - Red médica social sensorizada (IPT-2012-0882-430000)
- OSAMI-Commons Open Source Infrastructure Ambient Inteligence Commons (TSI-020400-2009-92) (Proy. Internacional)

1.5.2 *Contribuciones a revistas*

Specification and Unattended Deployment of Home Networks at the Edge of the Network [9]
Autores: Iván Bernabé-Sánchez, Daniel Díaz-Sánchez y Mario Muñoz-Organero.
Revista: IEEE Transactions on Consumer Electronics (Volume: 66, Issue: 4, Nov. 2020).
Abstract: Consumer devices continue to expand their capabilities by connecting to digital services and other devices to form information-sharing ecosystems. This is complex and requires meeting connection requirements and minimal processing capabilities to ensure communication. The emergence of new services, and the evolution of current technologies, constantly redefine the rules of the game by opening up new possibilities and increasing competition among service providers. Paradigms such as edge computing, softwarization of physical devices, self-configuration mechanisms,

definition of software as a code and interoperability between devices, define design principles to be taken into account in future service infrastructures. This work analyzes these principles and presents a programmable architecture in which services and virtual devices are instantiated in any computing infrastructure, as cloud or edge computing, upon request according to the needs specified by service providers or users. Considering that the target computing infrastructures are heterogeneous, the solution defines network elements and provides network templates to ensure it can be deployed on different infrastructures irrespectively of the vendor. A prototype has been developed and tested on a virtualized cloud-based home network relying on open source solutions.

1.5.3 *Contribuciones a congresos*

Optimizing OSGi Services on Gateways[7]
Autores: Ivan Bernabe Sanchez; Daniel Diaz Sanchez y Mario Muñoz-Organero.
Conferencia: Ambient Intelligence-Software and Applications, Springer International Publishing, 2013. p. 155-162.

Flex-box: A flexible software architecture for IPTV set-top boxes [10]
Autores: Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero
Conferencia: Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on. Digital Object Identifier: 10.1109/ICCE-Berlin.2012.6336480 Publication Year: 2012 , Page(s): 121 - 125 **Abstract:** IPTV services are available in many flavors. There are two main trends: vertical IPTV platforms on the Internet with user unrestricted access and standard platforms that belong to closed systems (users need to contract the access). The substantial differences and incompatibilities among the different types of platforms have brought the appearance of a plethora of different user technologies and systems that require non-professional users to deal with different devices and hampers the interchange between vendors. This work approaches the problem of dealing with vertical platforms providing a flexible horizontal self-configuring STB architecture.

Optimizing resouces on gateways using OSGi[8]
Autores: Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero
Conferencia: 2012 International Conferences on Consumer Electronics. Las Vegas, January 2012
Abstract: There are lots of devices and services inside a home which have to be managed. OSGi platforms alleviate this management by installing some services on a gateway located at home. A service provider might manage those services as part of a package of services. In general, OSGi platforms are connected to centralized repositories facilitating the maintenance by service providers. However, when various service providers work on a given gateway the number of services and components installed in it increases. This sometimes leads to inconsistencies as duplicate components that might cause performance problems or service interruptions. This paper presents a system for analyzing and automatically optimizing the components deployed on a OSGi home gateway.

1.5.4 *Contribuciones a congresos nacionales*

La Nube y su Papel como Plataforma de Distribución de Contenidos de TV[11]
Autores: Iván Bernabé, Daniel Díaz-Sánchez y Mario Muñoz-Organero
Conferencia: JARCA Conference, Rota, June 2014

Arquitectura IPTV para STB Basada en Capas de Control Jerárquicas[12]
Autores: Ivan Bernabe Sanchez; Daniel Diaz Sanchez; Mario Munoz-Organero
Conferencia: JARCA 2013, p. 9.

An OSGi platform for Eco-Driving systems[13]
Autores: Iván Bernabé, Mario Muñoz.
Conferencia: Actas JARCA 2012. pp. 11-16. ISBN 978-84-616-2007-4 2012. Junio 2012. Salou.

Using XMPP to Federate Multimedia UPnP Device[6]
Autores: Iván Bernabé Sánchez, Daniel Díaz Sánchez, Mario Muñoz Orga-

nero
Conferencia: Jornadas JARCA 2011 , 27, 28 y 29 de Junio de 2011 , Huelva

2

ESTADO DEL ARTE

A lo largo de este capítulo se muestra el estado del arte de aquellas tecnologías que son de interés para el desarrollo de esta tesis. El trabajo de esta tesis esta orientado a mejorar y optimizar las infraestructuras de servicios software utilizadas por los proveedores de servicios para ofrecer su oferta de servicios. Por esta razón, este capítulo tiene el objetivo de poner al lector en el contexto de las tecnologías involucradas en este trabajo además de presentar un análisis y clasificación de dichas tecnologías. El capítulo comienza con la sección 2.1 en la que describen los diferentes tipos de de arquitecturas existentes que los proveedores de servicios pueden utilizar. A continuación la sección 2.2 presenta las tecnologías relacionadas con la virtualización de sistemas y como puede ser utilizada para encapsular software. La sección 2.3 describe la importancia y ventajas de convertir cualquier dispositivo físico en un dispositivo virtual y finalmente, la sección 2.4 describe las diferentes arquitectura y tecnologías analizadas en el capítulo y que pueden convivir en la nube formando un complejo ecosistema digital. Además, se muestran y describen estándares y soluciones que permiten no sólo gestionar los recursos en la nube sino también las tecnologías analizadas en 2.2.

2.1 TIPOS DE MODELOS DE ARQUITECTURAS PARA PROVEEDORES DE SERVICIOS DE DISTRIBUCIÓN DE CONTENIDOS

El tráfico en los sistemas de distribución de contenidos por Internet ha crecido exponencialmente en los últimos años. Estos se debe a que el ecosistema de los servicios digitales ha cambiado debido principalmente al aumento en el número de dispositivos móviles, al incremento en los anchos de banda para satisfacer la calidad de los servicios y a la aparición de nuevas aplicaciones que tienen que coexistir con las anteriores ya asentadas en el mercado generando tráfico adicional. Estos nuevos requisitos generan nuevas necesidades de computación y de servicios que tienen que ser gestionadas eficazmente para satisfacer la calidad. Para hacer frente a la gestión de los sistemas de tecnologías de la información (TI) se propone el uso de las arquitecturas orientadas a servicios (SOA) ya que este tipo de arquitectura permite organizar y gestionar fácilmente los sistemas a través de servicios. Trabajar con servicios permite organizar un sistema con servicios independientes proporcionando una abstracción del sistema en

la que se muestra sólo información básica, dando una visión simplificada de la estructura y por lo tanto facilitando la comprensión del sistema (ver apartado 2.1.5).

Los beneficios de trabajar con servicios también se caracterizan por trabajar con componentes y servicios software con acoplamientos bajos los cuales facilitan la reusabilidad del software y además permiten crear composiciones formadas por varios servicios con el objetivo de crear aplicaciones o servicios más complejos. Es decir, la arquitectura de servicios permiten configurar estructuras de servicios independientes donde cada servicio se encarga de un aspecto particular y son capaces de comunicarse entre sí para realizar servicios más complejos. En las siguientes secciones se van a describir los principales modelos de arquitecturas diseñadas para solucionar diferentes problemas existentes en múltiples escenarios de aplicación.

Estos modelos de arquitecturas son soluciones evolucionadas del modelo tradicional 2.1.1 y han evolucionado para cubrir nuevas necesidades surgidas para satisfacer la demanda y las nuevas necesidades provenientes del desarrollo tecnológico. Algunas de las principales aportaciones del desarrollo tecnológico a los sistemas de TI compuesto por servicios ha promovido:

1. La expansión de las tecnologías móviles, la reducción de los costes de los dispositivos inteligentes y el incremento de contenidos audiovisuales de alta calidad ha impulsado el consumo de ancho de banda de las redes de comunicaciones empujando a los ingenieros a investigar nuevos mecanismos que permitan optimizar y descargar la carga de trabajo del core de las redes. Como resultado de tales investigaciones, una de las más efectivas es la de desplazar los contenidos más demandados a centros de datos distribuidos geográficamente. Estos centros de datos perimetrales son parte de centros datos centralizados que se extienden geográficamente y que distribuyen sus servicios a través de todos sus centros de datos dando lugar a las arquitecturas distribuidas geográficamente 2.1.2.

2. El surgimiento de nuevos SPs que han dado lugar a un amplio abanico de servicios disponibles en el mercado de las TIs. Este mercado de servicios formado por multitud de de SPs independientes integra en ocasiones servicios con funcionalidades similares pero a precios y características diferentes que podrían satisfacer la demanda de clientes con necesidades particulares. La incorporación de nuevos SPs al mercado proporciona una oferta de servicios mayor y asegura la existencia de soluciones heterogéneas capaces de satisfacer la demanda de cualquier cliente. Sin embargo supone un problema, ya que el cliente puede perderse en el mercado y no escoger los servicios más adecuados a sus necesidades. Los brokers de servicios vienen a solucionar

esa problemática desde un nuevo modelo de consumo de servicios que modifica ligeramente la arquitectura de servicios tradicionales. En la sección 2.1.3 se detalla dicha arquitectura.

3. El desarrollo de mecanismos para la auto-configuración de servicios y sistemas capaces de organizarse de manera automática. Uno de los ejemplos más populares de los últimos años es la aparición de nuevas tecnologías como las redes definidas por software (SDN) y las funciones de red (NFV) que ha permitido cambiar el concepto de infraestructuras de red drásticamente donde las funciones de red son conectados entre sí para proporcionar servicios de red. La posibilidad de poder adaptar la infraestructura de manera ad-hoc a través de mecanismos software abre nuevas necesidades que deben ser tomadas en cuenta. En la sección 5.2 se comenta dicha arquitectura.

4. La creación del paradigma de los microservicios que ha permitido llevar el concepto de arquitecturas de servicios a un nivel más local proponiendo llevar el concepto de arquitecturas de servicios al desarrollo de aplicaciones tradicionales. El uso de microservicios en el dearrollo de aplicaicones tradicionales permite a los desarrolladores beneficiarse de las ventajas de las arquitecturas de servicios a la hora de desarrollar aplicaciones o servicios software. La sección 2.1.5 describe dicha arquitectura.

2.1.1 *Modelo de arquitectura de servicios tradicional*

Para comprender los modelos de arquitecturas de servicios las tecnologías de la información, primero es necesario comprender qué es un servicio debido a que las arquitecturas de servicios se componen de éstos. Los servicios, en el ámbito de los sistemas de telecomunicación, forman parte de aplicaciones y proporcionan funcionalidades particulares que pueden ser consumidas por una o más de una aplicación. Los servicios de telecomunicación permiten el intercambio de datos entre dos entidades (puede que distanciadas geográficamente) y cada servicio esta diseñado para implementar una funcionalidad específica. Los servicios de telecomunicación se integran en aplicaciones de telecomunicaciones formando un conjunto de servicios que permiten a las aplicaciones desarrollar las tareas de comunicación.

Los sistemas dedicados a la distribución de contenidos multimedia están compuestos por un conjunto de servicios que proporcionan funcionalidades únicas e independientes relacionados con la distribución de contenidos multimedia. Por ejemplo, servicios de almacenamiento en caché localizados en la red, servicios de transcodificación, de transmisión de contenidos, etc.

La Figura 1 muestra el modelo de una arquitectura de servicios tradicional en la que un proveedor de servicios (en este caso proveedor de servicios 1, SP1) dispone de un conjunto de servicios los cuales están desplegados en una infraestructura de TI. Los servicios desplegados son desarrollados por los ingenieros del software (IS1). IS1 pueden trabajar para SP1 para que desarrollen los servicios o SP1 los puede adquirir a terceros o por otra vía. El administrador de sistemas (A1), trabajador de SP1, es el encargado de las tareas de gestión y configuración necesarias para que dichos servicios puedan ser desplegados sobre la infraestructura base. Finalmente, los clientes de SP1 consumen los servicios contratados.

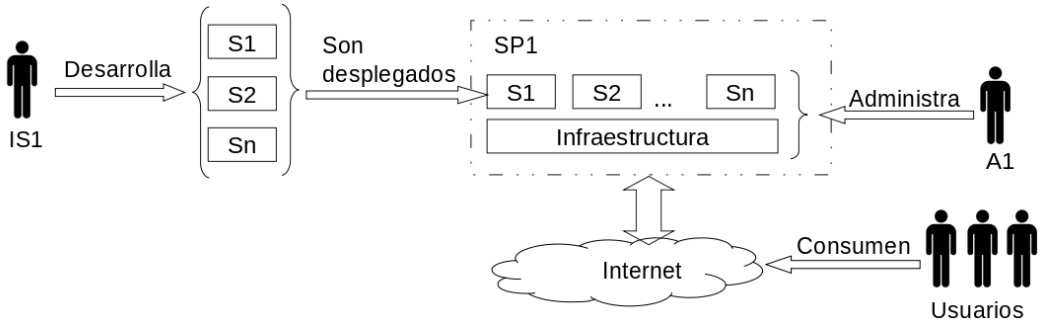

Figura 1: Estructura general de un modelo de arquitectura de servicios donde se aprecia la relación entre los diferentes roles de usuarios que interactúan en el consumo de servicios proporcionados por un proveedor de servicios (SP1).

2.1.2 *Arquitectura de servicios distribuidos geográficamente*

La Figura 2 muestra el modelo de una arquitectura de servicios distribuida sobre centros de datos remotos en la que un proveedor de servicios (en este caso el proveedor de servicios 1, SP1) dispone de un conjunto de servicios desplegados sobre varios centros de datos (Centro de datos 1 (CD1) y centro de datos 2 (CD2)) los cuales están separados geográficamente. A diferencia del modelo comentado en el apartado anterior, SP1 distribuye los servicios sobre diferentes los centros de datos distribuidos geográficamente para cumplir necesidades de diferente índole (por ejemplo: localizaciones mejor posicionadas en relación a los usuarios finales). Al igual que en el caso anterior, los servicios desplegados son desarrollados por los ingenieros del software (IS1) que pueden estar contratados por SP1 para que los desarrollen, o SP1 los puede adquirir aquellos servicios por otra vía. El administrador de sistemas (A1) de SP1 es el encargado de las tareas de gestión y configuración necesarias para que dichos servicios puedan ser desplegados correctamente sobre los diferentes centros de datos. Finalmen-

te, los clientes de SP1 consumen los servicios contratados accediendo a los servicios ubicados en los diferentes centros de datos.

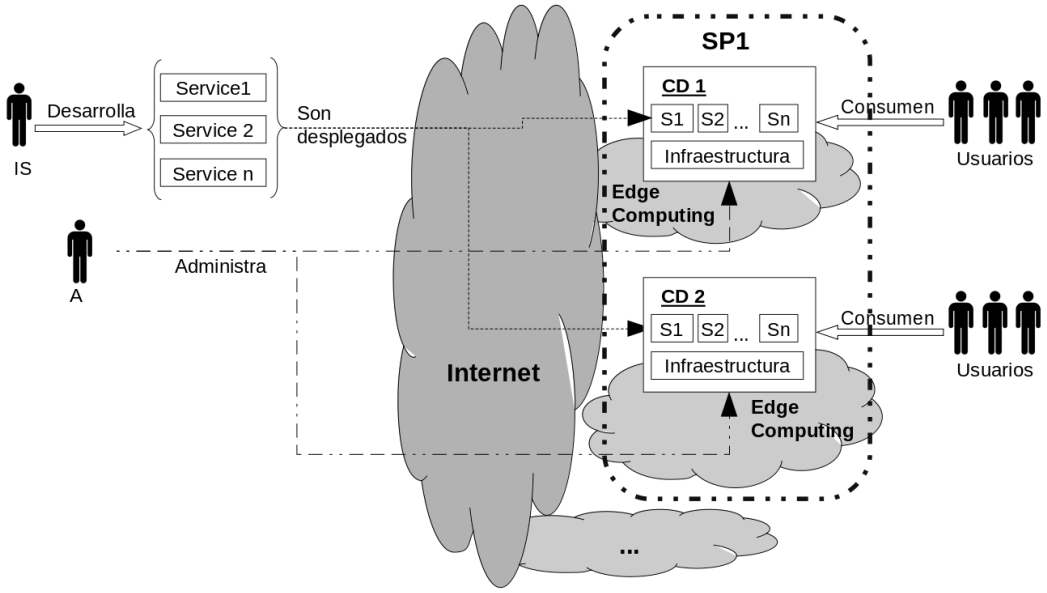

Figura 2: Estructura general de un modelo de arquitectura de servicios distribuida. La figura muestra como un proveedor de servicios (SP1) puede distribuir geográficamente sus servicios en diferentes centros de datos (Centro de datos 1 (CD1) y centro de datos 2 (CD2)). La figura también muestra la relación entre los diferentes roles de usuarios que colaboran con el SP1 para permitir consumir servicios a los usuarios clientes.

El origen de esta arquitectura surge como solución a varios problemas debido a que las arquitecturas básicas de servicios de telecomunicación (descritas en el apartado anterior) presenta limitaciones técnicas que no permiten satisfacer ciertos requisitos para algunas aplicaciones. Gran parte de estos problemas están relacionados con el consumo del tráfico de datos de las aplicaciones, ya que éstas cada vez demanda más tráfico que debe ser intercambiado por las redes de comunicaciones. Un ejemplo claro se puede observar en la distribución de contenidos multimedia debido a que los requisitos de transmisión para la reproducción son más exigentes demandando más anchos de banda para ofrecer mayor calidad.

Como solución a esta nueva situación, se propuso un modelo de arquitectura de servicios distribuida geográficamente sobre centros de datos cercanos a los usuarios finales. La idea es mover o duplicar servicios sobre los centros de datos que se encuentran distribuidos geográficamente. El objetivo es mejorar la calidad de los servicios a través de la reducción de la latencia de las comunicaciones.

Un ejemplo ampliamente extendido lo podemos encontrar en las redes de distribución de contenidos (CDNs) que cachean contenidos en varios

lugares de la red para que los contenidos no sean descargados desde el core del sistema sino que los descargue del centro de datos más cercano del usuario final.

En esta línea, la European Telecommunications Standards Institute (ETSI) definió un modelo de arquitectura de servicios distribuida conocida como Multi-access Edge Computing [14] [15], anteriormente conocida como Mobile Edge Computing (MEC) [1], la cual está estandarizada por el Grupo de Especificación de la Industria (ISG) del European Telecommunications Standards Institute[2]. MEC es el paradigma de computación distribuida más importante, sin embargo existen otras alternativas como el Fog Computing desarrollado por Cisco [16], que permite que las aplicaciones se ejecuten directamente en el borde de la red a través de los dispositivos inteligentes conectados. Otro modelo de computación en la nube es el Cloudlet [17] que nació para resolver el problema de latencia en el acceso a la nube y proporcionar servicios de aplicaciones, procesamiento de carga y almacenamiento cerca de los usuarios finales.

2.1.3 *Modelo basado en arquitecturas de Brokers de servicios*

Un Broker de servicios (BS) es un servicio mediador que conecta a los clientes con los proveedores de servicios (SPs) gestionando las peticiones realizadas por los clientes para consumir algún servicio proporcionado por alguno de los SPs. El Broker de servicios indexa varios servicios provenientes de diferentes SPs y cuando le llegan las peticiones de algún cliente, el BS se encarga de proporcionar el servicios más adecuando tomando en cuenta aspectos relevantes para los clientes tales como el precio del servicio, el ancho de banda requerido, la calidad del servicio, etc.

La Figura 3 muestra el modelo de una arquitectura de servicios que integra un broker de servicios, en este caso el modelo de negocio varia bastante a los modelos comentados anteriormente. En este escenario, los usuarios clientes consumen servicios desde el proveedor de servicios SP4 pero en este caso los servicios no pertenecen a SP4 sino que SP4 dispone de una broker de servicios que se encarga de indexar y catalogar los servicios desplegados en otros proveedores de servicios (SP1, SP2 y SP3). SP4 recopila, registra y guarda información relevante de los servicios proporcionados por cada proveedor de servicios y así puede ofrecer los servicios más adecuados a la demanda además de crear aplicaciones compuestas por varios servicios. Cuando SP4 recibe una petición para consumir alguno de los servicios, el broker de servicios procesa la petición y selecciona el servicio más adecuado a consumir acorde a unas políticas predefinidas (tales como

1 https://www.etsi.org/technologies/multi-access-edge-computing
2 https://www.etsi.org/

el precio, grado de adecuación, compatibilidad, etc). De la misma manera que en los escenarios anteriores, los clientes consumen los servicios a través de SP4 de manera transparente y sin que sean consciente de la complejidad del sistema. SP1, SP2 y SP3 pueden tener un modelo de arquitectura del tipo al que se muestra en la sección 2.1.1 o 2.1.2.

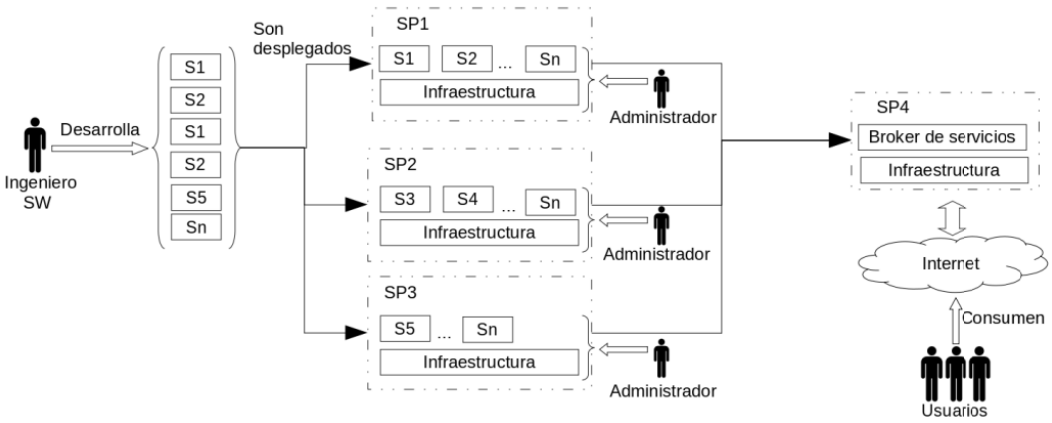

Figura 3: Estructura general de un modelo de arquitectura basada en un broker de servicios. La figura muestra como un broker de servicios (SP4) que se encarga de indexar y registrar los servicios proporcionados por los proveedores de servicios SP1, SP2 y SP3 con el objetivo de formar un catalogo de servicios. El broker de servicios gestiona estos servicios y se los ofrece a los usuarios que necesitan consumir alguno de esos servicios. La figura también muestra la relación entre los diferentes roles de usuarios que interactúan con alguno de los proveedores de servicios para asegurar su funcionamiento.

Satisfacer la demanda de los usuarios no es fácil y en ocasiones los brokers de servicios tienen dificultades para encontrar proveedores de servicios con las capacidades adecuadas para satisfacer la demanda. En la bibliografía existen varios trabajos que proponen soluciones para abordar diferentes dificultades técnicas. Por ejemplo: en [18] han abordado el problema de una arquitectura de servicios basada en brokers para permitir la interoperabilidad entre servicios en la nube heterogéneos.

[19] presenta un trabajo basado en un broker de servicios para un marketplace en la nube el cual decide qué recursos de infraestructura se deben asignar para minimizar los costos de usuario. En [20] se propone un framework intermedio para proporcionar servicios en la nube de diferentes proveedores de infraestructuras a los usuarios para el procesamiento de big data en streaming. En [21] se propone un framework basado en un broker el cual asume la responsabilidad de recomendar los mejores servicios disponibles a los usuarios. En [22] analizan como maximizar los beneficios para un broker en la nube a través de la compra de instancias de máquinas

virtuales reservadas a varios proveedores de infraestructura y así poder ofrecer servicios a los usuarios.

A pesar de que estos trabajos solucionan problemas en dominios diferentes, cada uno tiene carencias que no les hace idóneas para otros ámbitos. En [19] [20] [21] no tomaron en cuenta la heterogeneidad del tipo de usuario y el tiempo de respuesta, en [23] tampoco tuvieron en cuenta la heterogeneidad del usuario, el factor de precios y la estrategia de equilibrio de carga. Estos parámetros que pueden ser de interés para unas soluciones, no lo son para otros y las preferencias a tener en cuenta depende de los diferentes requisitos y el escenario de aplicación. Por lo que es complicado de disponer de una solución genérica que pueda aplicarse a múltiples entornos.

Los brokers de servicios tienen funciones de intermediación, agregación y arbitraje [24]. Por lo tanto, los brokers de servicios agregan valor a los servicios que intermedian mediante la entrega de servicios agregados nuevos y/o personalizados.

2.1.4 *Modelos de arquitectura de servicios encadenados*

En la Figura 4 se muestra un modelo de arquitectura orientado a desarrollar sistemas de servicios compuestos o aplicaciones a través de la composición de servicios más sencillos. El escenario está compuesto por un proveedor de servicios SP1 que dispone de un catalogo de servicios disponibles para ser desplegados.

En el modelo de la Figura 4, SP1 se compone por una arquitectura distribuida y flexible donde el componente principal es el orquestador situado en el core del sistema de SP1. El orquestador es un componente software encargado de configurar, desplegar y configurar cada uno de los servicios disponibles sobre múltiples centro de datos. Este modelo de arquitectura es mucho más flexible y escalable que los modelos anteriores, además el control del sistema recae por completo en el orquestador. Sin embargo, requiere procesos de configuración complejos haciendo que las tareas de mantenimiento y de operación sean más complicadas por lo que requiere una cierta formación del personal de administración de sistemas.

A pesar de las complejidades que presenta la gestión de servicios de este modelo de arquitectura, su aplicación trae enormes beneficios ya que permite: alcanzar un rendimiento óptimo de las redes, satisfacer las demandas del usuario adaptando los sistemas en tiempo real y también permite garantizar el cumplimiento con los acuerdos de nivel de servicio y minimizar los costes de funcionamiento.

Principalmente, hay dos enfoques para implementar una cadena de servicios: coreografía y orquestación [25]. La coreografía define las secuencias y condiciones en las que diferentes servicios independientes se conectan.

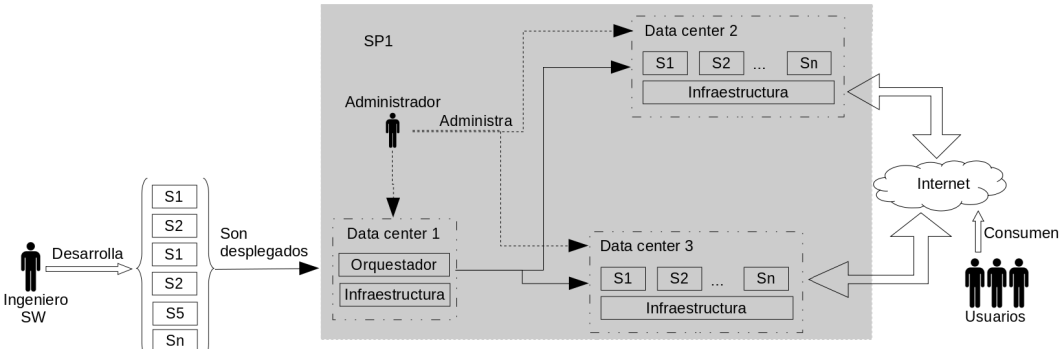

Figura 4: Modelo de arquitectura de servicios en la cual los servicios cooperan entre ellos para proporcionar funcionalidades complejas a partir de servicios más simples.

En el caso de la orquestación se definen las secuencias y condiciones en las que un servicio invoca otros servicios [26]. Es decir, la orquestación permite a una entidad central controlar diferentes servicios y sus interacciones, mientras que la coreografía permite a los servicios organizarse de forma descentralizada.

En ambos enfoques, la tarea de seleccionar cada uno de los servicios es de vital importancia, por esta razón, en las arquitecturas orientadas a servicios [27], se propone que cada proveedor de servicios publique información de cada uno de ellos creando un directorio con los servicios disponibles. Finalmente, a través de algoritmos de selección, se pueden escoger los servicios más adecuados para componer la aplicación o servicio deseado. Como ejemplo, en [28] y [29] se presentan algoritmos que deciden los servicios más adecuados tomando en cuenta diferentes parámetros como el precio y el tiempo de respuesta. A parte de tener en cuenta parámetros técnicos que determinen el mejor servicio a ser utilizado, también es necesario asegurarse que los servicios a evaluar son capaces de proporcionar la funcionalidad adecuada al objetivo final. Es decir, si se desea crear una nueva aplicación de videollamadas, se de deben seleccionar servicios de vídeo adecuados con las características adecuadas para formar parte de la aplicación final.

2.1.5 *Arquitectura basadas en microservicios*

La arquitectura de microservicios es un nuevo enfoque emergente utilizado para diseñar y desarrollar aplicaciones que requieren alta escalabilidad y permiten realizar cambios importantes de manera simple y rápida. La idea detrás de las arquitecturas de microservicios no es completamente nueva, ya que es una evolución de la arquitectura orientada a servicios. Una de las principales características de la arquitectura de microservicios

es la estructura de las aplicaciones compuestas por un cierto número de servicios independientes donde cada servicio está se encarga de un aspecto particular, por lo tanto, los microservicios se comunican entre sí para realizar servicios más complejos.

Hay varias ventajas que se podrían obtener al aplicar una arquitectura de microservicios en el dominio de los proveedores de servicios. Principalmente, el uso de microservicios es que el desacoplo de componentes permite configurar un entorno óptimo para crear y mantener aplicaciones que requieren una gran escalabilidad. Una característica importante para el desarrollo de microservicio es el poder desarrollarlos de forma independiente permitiendo gestionar cambios importantes del software que implementan. A continuación se muestra una lista de beneficios adicionales que podrían obtenerse al adoptar los microservicios:

- Al separar los componentes de una aplicación es menos probable que un error o un problema afecte a todo el sistema. Los posibles servicios defectuosos podrían aislarse, repararse y volver a implementarse sin necesidad de detener toda la aplicación.
- El código fuente es mucho más fácil de entender debido a que se reduce la complejidad del microservicio ya que éste provee un menor número de funcionalidades que lo que sería un servicio tradicional. Por lo tanto, los desarrolladores podrían concentrarse en tareas específicas sin afectar a la aplicación general y sin la necesidad coordinarse con otros desarrolladores.
- Al desarrollar un microservicio, los desarrolladores son libres de elegir cualquier lenguaje de programación que se adapte mejor a ese servicio en particular. El encapsulamiento de servicios permite desarrollar software que tan sólo requiera entenderse con otros servicios a través de un punto de conexión sin importar el lenguaje de programación con el que ha sido desarrollado.

2.1.6 *Resumen*

En esta sección se ha mostrado varias alternativas de modelos de arquitecturas de servicios utilizadas para satisfacer las necesidades de diferentes escenarios de aplicación. Las plataformas de servicios son una forma eficaz de gestionar sistemas complejos de TI, dependiendo del tamaño, el tipo de aplicación y los requisitos del sistema a implementar, los diseñadores optarán por uno de los modelos de arquitecturas comentados.

Todos estos modelos han ido evolucionando a partir del modelo de arquitectura de servicios tradicional, las diferentes variaciones han surgido

con el objetivo de solucionar necesidades encontradas sobre los sistemas reales. A partir de este análisis surge la idea de pensar cuáles serán las futuras necesidades y cambios que deberán afrontar las futuras arquitecturas de servicios para hacer frente a las necesidades de las aplicaciones actuales. Retos como la reducción de la latencia, incremento y simplificación de tareas de gestión de las redes, optimización en la colocación de servicios de red, la gestión de servicios de manera desatendida y el desarrollo de herramientas avanzadas. Esto facilita la gestión de todos estos procesos sobre grandes sistemas complejos que hacen la necesidad de diseñar mecanismos y herramientas para abordar los futuros escenarios de sistemas orientados a servicios.

Conseguir una alta personalización en las futuras arquitecturas de servicios será uno de los grandes desafíos a conseguir ya que implicará que los recursos deben configurarse dinámicamente y agruparse para ser capaces de proporcionar diferentes perfiles de servicio a petición [30].

A partir de estas necesidades se ha desarrollado la tabla 1 que contiene los requisitos que deberían ser satisfechas por las futuras arquitecturas de servicios.

A continuación se describe cada una de los requisitos presentadas en la tabla 1:

- R1. Mecanismos para gestión de servicios. Requisito que recoge la necesidad de proporcionar mecanismos que permitan añadir, indexar modificar y eliminar los servicios disponibles en la plataforma.
- R2. Catalogo de servicios. Se requiere la necesidad de publicar y describir los servicios disponibles en los sistemas para que puedan ser desplegados en cualquier momento.
- R3. Soporte Multiplataforma. Hace la referencia a la capacidad que tendrán los servicios de trabajar sobre diferentes infraestructuras base de manera transparente.
- R4. Encapsulado de servicios. Con el objetivo de facilitar el despliegue y la portabilidad de los servicios, la encapsulación de los servicios serán clave a la hora de gestionar los servicios.
- R5. Empaquetado de servicios. Por lo general, todo servicio requiere de otras componentes (librerías, servicios de terceros, etc.) para poder realizar su trabajo correctamente. La capacidad de trabajar con servicios empaquetados permite incorporar todas las dependencias que el servicio va a necesitar para que funcione correctamente.

Requisitos	Descripción
R1	Mecanismos para gestión de servicios.
R2	Catalogo de servicios.
R3	Soporte Multiplataforma.
R4	Encapsulado de servicios.
R5	Empaquetado de servicios.
R6	Soporte para la configuración de enlaces virtuales.
R7	Softwarización de cualquier tipo de funcionalidad.
R8	Mecanismos de interoperabilidad con otros sistemas.
R9	Gestión de recursos de infraestructura.
R10	Mecanismos para desplegar y configurar software de manera desatendida.
R11	Administración del ciclo de vida de los servicios
R12	Adaptar/configura infraestructura en función de los requisitos de funcionamiento de los servicios.
R13	Gestionar encadenamiento de servicios.
R14	Mecanismos para optimizar el despliegue de servicios.
R15	Curva de aprendizaje moderada
R16	Requisitos de funcionamiento mínimos
R17	Autoconfiguración
R18	Aislamiento
R19	Integración personalizable

Tabla 1: Requisitos a tomar en cuenta en el desarrollo de las futuras infraestructuras de servicios.

- R6. Conexiones a través de enlaces virtuales. Los enlaces virtuales permite realizar comunicaciones de componentes software sin necesidad de gestionar el medio físico dedicado a las redes de telecomunicaciones. Satisfacer este requisito no sólo agiliza el proceso de conectar diferentes elementos sino que facilita y permite su automatización mejorando el CAPEX y el OPEX.

- R7. Softwarización de cualquier tipo de funcionalidad o servicio. La transformación de cualquier dispositivo físico en un dispositivo software capaz de realizar las tareas de su homologo físico permite agilizar, no sólo su puesta en funcionamiento, sino también las tareas de mantenimiento y actualizaciones mejorando considerablemente el CAPEX y el OPEX.

- R8. Mecanismos de interoperabilidad con otros sistemas. La posibilidad de poder comunicarse con otros sistemas permite la expansión y compartición de servicios y recursos entre diferentes infraestructuras. Esto añade robustez y escalabilidad a los sistemas haciéndolos más seguros a través de mecanismos de balanceo de carga. Además permite organizar sistemas complejos en sistemas más simples que intercambia información por un objetivo en común.

- R9. Gestión de recursos de infraestructura. La capacidad de poder manipular los recursos asignados a una infraestructura es importante ya que afecta directamente sobre su capacidad de escalado y liberación de recursos.

- R10. Mecanismos para desplegar y configurar software de manera desatendida. Una capacidad importante para los sistemas del futuro es la posibilidad de poder realizar tareas de gestión de componentes software de manera desatendida. Los sistemas del futuro deberán proveer herramientas y mecanismos para proveer esta funcionalidad.

- R11. Administración del ciclo de vida de los servicios. Otro requisitos importante es proveer herramientas y mecanismos para gestionar el ciclo de vida de los servicios que permitirá instalar, arrancar, parar o desinstalar cualquier servicio que deba estar (o no) activo en un momento dado.

- R12. Adaptar/configurar infraestructura en función de las necesidades de funcionamiento de los servicios. Antes de instalar un servicio sobre una infraestructura concreta puede que sea necesario realizar ajustes de configuración previos para que los nuevos servicios puedan funcionar sin problema. Por ejemplo, resolver dependencias de

software necesario y no instalado previamente (por ejemplo otro servicio).

- R13. Gestionar encadenamiento de servicios. A medida que la complejidad de los sistemas aumente, será necesario incorporar mecanismos que permitan orquestar todos los elementos involucrados en la cadena de servicios con la que se pretenda satisfacer la demanda en un momento dado.

- R14. Mecanismos para optimizar el despliegue de servicios. Los sistemas deberán tratar de optimizar el despliegue de los servicios en función de la arquitectura subyacente y de los requisitos de funcionamiento del servicio.

- R15. Curva de aprendizaje moderada. Es recomendable que todo sistema de TI sea sencillo y comprensible para facilitar su adopción por parte de los administradores e ingenieros de sistemas a través del uso de estándares y tecnologías ampliamente extendidas y adoptadas por la industria.

- R16 Requisitos de funcionamiento mínimos. La solución debe ser capaz de proporcionar configuraciones mínimas y suficientes con el objetivo de satisfacer las necesidades completas de los servicios a ser desplegados manteniendo un footprint bajo. Esto permitirán desplegar infraestructuras responsables con el medio ambiente.

- R17 Autoconfiguración. La solución debe proporcionar a los usuarios la capacidad de implementar aplicaciones bajo demanda sin tener que esperar a la acción humana.

- R18 Aislamiento. La solución debe ser capaz de soportar un modelo multiusuario capaz de proporcionar servicios de forma segura a varios usuarios (o grupos de usuarios) y sin ninguna supervisión humana. Además, los recursos deben aislarse entre los usuarios del sistema para que los recursos controlados por un usuario no tengan acceso ni impacto en los recursos controlados por otros usuarios.

- R19. Integración personalizable. La solución no debe imponer un modelo de implementación o arquitectura en particular a las aplicaciones. Cada aplicación tiene requisitos únicos que deben ser aclimatados permitiendo que los servicios se conecten entre sí en el "espacio de usuario" de manera automática. Esto permite al desarrollador de aplicaciones personalizar cualquier cosa del lado cliente, pero no debe requerirlo. Los servicios deben estar lo suficientemente integrados

como para que puedan ser conectados por el consumidor de la nube sin necesidad de ninguna interacción del lado cliente.

2.2 LA VIRTUALIZACIÓN Y ENCAPSULAMIENTO DE SERVICIOS

2.2.1 *Introducción*

En 2.1 se ha presentado los diferentes tipos de arquitecturas y sus particularidades que la hacen adecuadas a ciertos casos prácticos. Una característica importante de este tipo de sistemas es la capacidad de tener en cuenta algunos aspectos cuando es necesario modificar los recursos requeridos por los servicios o mover un mismo servicio entre infraestructuras basadas en tecnologías diferentes. Estos es debido a que diferentes infraestructuras pueden trabajar con distintos entornos tecnológicos e incluso en algunos casos, estos entornos pueden estar adaptados a necesidades particulares que les hace únicos dificultando la compatibilidad al moverlo a otras infraestructuras o modificar los recursos que tiene asignados. Por esta razón, no es fácil desplegar un mismo servicio sobre diferentes entornos de computación, además, en ocasiones cada servicio esta diseñado para ser ejecutado en el entorno (o plataforma cloud) para el que fue diseñado y esto dificulta moverlo a otras infraestructuras.
Una manera de solucionar este problema es a través de herramientas capaces de desplegar software sobre diferentes infraestructuras haciendo uso de herramientas de gestión de la configuración como Puppet [31], Ansible [32], etc. Otra manera de solucionarlo es a través del uso de contenedores software que encapsulan los servicios y aplicaciones a ejecutar junto a las dependencias y librerías requeridas por dicho software, de este modo, se facilita la migración de servicios entre infraestructuras. Esta última iniciativa está muy relacionada con la tecnología de la virtualización. Ambas soluciones son válidas [9], pero cada una de ellas presentan características particulares que las hacen diferentes y adecuadas para diversas situaciones. Por ejemplo, desplegar software a través de herramientas de gestión de la configuración da como resultado un sistema optimizado (sólo se instala el software estrictamente necesario y no se duplica dependencias o librerías previamente instaladas en el sistema) con baja sobrecarga adicional. Para esto, se requiere que los archivos de configuración puedan ser procesados correctamente por herramientas de configuración, si no es así podría haber problemas de compatibilidad y por lo tanto el despliegue del software no se llevaría cabo o incluso podría desplegarse incorrectamente.
En caso de las soluciones basadas en mecanismos de virtualización, generalmente, reducen los problemas de despliegue de software ya que se mueve todo el software encapsulado en un entorno virtualizado pero aumenta

la sobrecarga global del sistema. Esto es debido a que mover una máquina virtual (VM) más el software instalado sobre ella, por lo general será más pesado y requerirá más tiempo que si se instalase el software sobre una VM genérica ubicada en el lugar deseado con el entorno de ejecución ya configurado y listo para ejecutar aquel software (esto se analizará más adelante en el capítulo 7).

En los apartados 2.2.3 y 2.2.4 se describen mecanismos de virtualización que permiten mover servicios entre infraestructuras y en la sección 3 se presenta algunos mecanismos para desplegar servicios entre infraestructuras a través de herramientas orientadas al despliegue de infraestructura por código.

2.2.2 *Tecnología de virtualización*

La virtualización representa la capacidad de ejecutar varias instancias lógicas (software) sobre instancias físicas [33]] tales como hardware y máquinas de computación físicas. Inicialmente la virtualización se consideró un método para dividir lógicamente grandes sistemas de computación para permitir que varias aplicaciones se ejecuten simultáneamente [34]. En otras palabras, es la abstracción del hardware físico para que pueda aparecer como varias instancias lógicas. Esto es posible debido a que la tecnología de virtualización permite compartir (dividir) o combinar recursos informáticos a través de la creación de otros virtuales.

Generalmente la virtualización ha sido empleada sobre servidores pero también se se aplica a diferentes dominios, incluidas las redes [35] [36] y el almacenamiento. La virtualización del servidor permite la ejecución de varios kernels del sistema operativo sobre el mismo hardware [37] permitiendo abstraer los recursos físicos y exponiendo interfaces para acceder a dichos recursos a los sistemas operativos instanciados instanciados. La virtualización se comporta de una manera similar a los sistemas operativos ya que abstrae los componentes hardware a través una interfaz que gestiona su acceso a las aplicaciones y usuarios. La tecnología de virtualización se basa principalmente en el hypervisor.

El hypervisor, también llamado Virtual Machine Manager (VMM), es el componente que administra los sistemas operativo de un servidor físico, presentando una vista virtualizada de los recursos físicos disponibles a los sistemas operativos (SO) invitados (SO instanciados sobre la máquina física). Básicamente hay 5 tipos de virtualización dependiendo del nivel de abstracción proporcionado y como se comportan [34] [37]:

- Virtualización completa. El sistema operativo invitado trabaja como si estuviese instalado sobre una máquina física y no requiere de configuraciones especiales, todas las llamadas al sistema son interceptadas

por el hipervisor que las está emulando. Es un enfoque genérico y flexible, ya que separa completamente el sistema operativo invitado de los componentes de hardware debajo ya que el hipervisor virtualiza todos los recursos físicos. Sin embargo, esto produce una sobrecarga de trabajo general a la máquina física que influye negativamente en el desempeño global de la máquina física. Los principales hipervisors de este tipo son: XEN [38] y KVM [39].

- Paravirtualización. Este tipo de virtualización tiene un enfoque diferentes a los hipervisores vistos anteriormente ya que trabajan a través de unas modificaciones requeridas sobre el sistema operativo invitado. En este tipo de virtualización las llamadas del sistema se sustituyen por llamadas al hipervisor, el cual abstrae el hardware de debajo proporcionando interfaces similares. Es una solución utilizada principalmente en los casos en que el hardware no admite la virtualización y como solución alternativa, las tecnologías de hipervisores se instalan en la parte superior del sistema operativo.
- Hosted Virtualization. Los hipervisores se instalan encima del sistema operativo host. Ejemplos de hipervisores de este tipo son VMWare [40] y Virtualbox [41].
- Emulado. Este tipo de virtualización emula un sistema informático completo a través de la emulación de los recursos de hardware y permitiendo que los SO puedan ser ejecutados sobre arquitecturas diferentes a la arquirectura física. QEMU [42] es un ejemplo de este tipo de hipervisor.
- Virtualización basada en contenedores. Es una tecnología de virtualización que no se basa está basada en un hipervisor [43]. Si no que su funcionamiento se basa en una modificación del kernel que permite compartir el sistema operativo host.

El desarrollo de los contenedores software ha transformado drásticamente la forma en que se implementa y gestiona el software. Un contenedor de software es una instancia aislada en el espacio de usuario de un sistema operativo. Esto hace que la tecnología de los contenedores a penas introduzcan sobrecarga al sistema operativo host y se puedan ejecutar un gran número de contenedores en la misma máquina física, todos con un sistema de archivos dedicado, aislado y sus procesos correspondientes en ejecución. Es decir, el contenedor se comporta como una máquina virtual, sin embargo, no se basa en la capa intermedia proporcionada por el hipervisor, por lo tanto, tiene menos sobrecarga [43]. La Figura 5 muestra las diferencias arquitectónicas entre las tecnologías de virtualización estándar y las de contenedores. Como se puede observar en la Figura 5, arquitectonicamente hay

una menor cantidad de capas requeridas por las tecnologías de contenedores para ejecutar los sistemas operativo invitados respecto a las soluciones basadas en hipervisores. Algunas de las implementaciones de contenedores más relevantes son actualmente: Linux Containers (LXC) [44] y Docker [45].

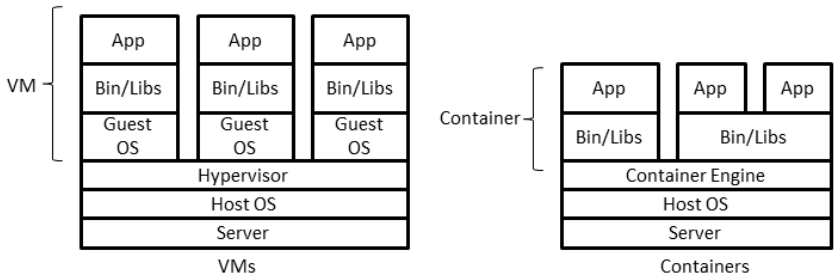

Figura 5: Diferencias entre las tecnologías de virtualización basadas en máquinas virtuales y la virtualización basada en la tecnología de contenedores.

Una de las principales ventajas que aporta la virtualización a las arquitecturas de servicios es que permite encapsular servicios en entornos virtualizados. Esto es útil para moverlos o desplegarlos sobre otras infraestructuras haciendo más fácil la migración de dichos servicios, ya que si los servicios se encuentran encapsulados en VMs o contenedores, tan sólo será necesario disponer de la configuración necesaria en la infraestructura destino para poder ejecutar la VM o contenedor además de los servicios contenidos en su interior. La idea es encapsular los servicios en entornos configurados óptimamente asegurando el correcto funcionamiento de éstos, junto a las librerías y dependencias necesarias. De esta manera no importa los aspectos de configuración de la infraestructura donde se este ejecutando el servicio ya que la tecnología de virtualización enmascarará aquellos aspectos tecnológicos de cara al servicio, emulando al entorno tecnológico con el que el servicio fue diseñado inicialmente.

2.2.3 *Virtualización de servicios a través de máquinas virtuales*

Como se ha comentado, la virtualización permite crear capas de abstracción de recursos que permiten emular completamente sistemas de computación. Esto permite ejecutar varios sistemas operativos y sus aplicaciones compartiendo recursos hardware desde un mismo sistema de computación.

Esta emulación del sistema de computación es lo que se llama máquina virtual (VM). Sobre un mismo sistema de computación puede haber una o varias VMs gestionadas por un hipervisor [46], el cual es el responsable de crear y gestionar cada una de las VMs. Para cada VM, el hypervisor emula y reserva los recursos físicos hardware requeridos para cada una de

las VMs lo que puede suponer una enorme carga computacional para el sistema host que alberga todas las VMs. El uso de máquinas virtuales esta muy extendido en sistemas de computación en la nube, ya que permiten desplegar instancias de máquinas virtuales sobre las cuales se despliegan servicios software. Este modelo de gestión de servicio es llamado Infraestructura como servicio (Infrastructure as a Service, IaaS). Algunos de los hypervisores más importantes de código libre que ofrecen IaaS son: Xen [38], KVM [39], VirtualBox [41], VMWare vSphere [40], etc.

2.2.4 *Encapsulado de servicios con tecnologías de contenedores*

La encapsulación de los servicio en contenedores presenta una opción rentable y fiable para la virtualización [47]. La virtualización de contenedores hace referencia a la virtualización a nivel de sistema operativo (SO) el cual ejecuta varios sistemas en un único host. La mayoría de los contenedores se construyen en base al kernel de Linux y se conocen como contenedores Linux (LXC) [48] (también se conoce como virtualización ligera del sistema operativo). La tecnología de contenedores logra el aislamiento a través de cgroups y espacios de nombres. Los espacios de nombres y cgroups son unas características nuevas del kernel de Linux que le permiten aislar los procesos entre sí. Cuando utiliza esas características, la llama "contenedores". Básicamente, estas características le permiten comportarse como si tuviese una máquina virtual, excepto que no es una máquina virtual en absoluto, son sólo procesos que se ejecutan en el mismo kernel de Linux. Los contenedores obtienen un aislamiento completo desde el punto de vista de la aplicación, incluidos los árboles de proceso, la red, los identificadores de usuario y los sistemas de archivos montados. Esta tecnología ha sido implementada principalmente para Linux pero está disponible para otros sistemas operativos.

La principal ventaja de la virtualización de contenedores sobre la virtualización con máquinas virtuales es que la virtualización por contenedores comparte un único sistema operativo host y un único kernel y a partir de ahí crea contenedores dentro de ese sistema operativo que incluyen todo el software, el entorno y las bibliotecas para una aplicación determinada. Esta forma de virtualización es muy diferente a la virtualización con VM ya que con las VMs se requiere que el sistema ejecute varias copias del sistema operativo invitado, lo que requiere mucha memoria y degrada drásticamente el rendimiento de todo el sistema.

En la virtualización de contenedores, el sistema operativo virtual no necesita duplicar la funcionalidad del sistema host si no que el sistema host es responsable de administrar estas llamadas al sistema y al hardware, au-

mentando el rendimiento del sistema al hospedar más máquinas virtuales basadas en contenedores [47], [48].

El principal desafío en la virtualización de contenedores es el problema del aislamiento entre aplicaciones, maquinas y contenedores. A pesar de los mecanismos implementados para conseguir el aislamiento de los contenedores, éstos siguen compartiendo el mismo sistema operativo host, lo que lo hace vulnerable a las amenazas de seguridad para todo el sistema [49]. Otro inconveniente de la virtualización de contenedores es que cada contenedor debe usar el mismo sistema operativo que el sistema operativo base, mientras que las instancias de virtualización se pueden ejecutar en un sistema operativo único. Por ejemplo, un contenedor creado en un host basado en Linux no podía ejecutar una instancia del sistema operativo Windows Server o aplicaciones diseñadas para ejecutarse en Windows Server.

Existen varias tecnologías para implementar la virtualización por contenedores, Docker y LXC son las más relevantes las cuales son descritas brevemente a continuación:

- Docker [45] es una tecnología de contenedores que tiene como objetivo proporcionar una solución para implementar una aplicación junto con sus dependencias automáticamente en un entorno autónomo denominado contenedor [50]. Es una aplicación ligera que permite la rápida implementación de contenedores los cuales están aislados entre sí y se les da una interfaz de red individual. El aislamiento proporciona una solución similar a las máquinas virtuales (VM), pero sin la sobrecarga de ejecutar el sistema operativo invitado sobre el sistema operativo host. Docker ofrece una sobrecarga mínima la cual se logra compartiendo el kernel con el sistema operativo host, a diferencia de las máquinas virtuales que ejecutan el kernel del sistema operativo invitado en un hipervisor [46].

  La gestión de los contenedores docker se realiza a través de espacios de nombres de kernel de Linux y grupos de control, esto permite aislar los procesos y limitar los recursos con una sobrecarga insignificante. Los procesos que se ejecutan en contenedores diferentes tienen la visión de creer que se ejecutan en sistemas operativos independientes con sus propios recursos asignados, como CPU y memoria tal como si se estuviesen ejecutando sobre un host tradicional. La tecnología Docker están orientada a que cada contenedor encapsule un servicio nada más, por lo que en el caso de implementar aplicaciones compuestas de varios servicios, se deberá desplegar tantos contenedores como servicios compongan la aplicación.

Docker	**LXC**
Virtualización de aplicaciones	Sistemas completamente virtualizados
Independientemente de la plataforma	Plataforma Linux
Para un propósito específico	Multi-propósito
Puede usar LXC	Puede albergar instancias docker
Distribución de software	Gestión de máquina
Encapsulación de aplicaciones	Alternativa a máquinas virtuales
PaaS	IaaS
Un contenedor para ejecución de un único proceso	Un contenedor para la ejecución de múltiples procesos

Tabla 2: La tabla muestra las principales características y diferencias entre las tecnologías de contenedores Docker y LXC.

- LXC [48] es una virtualización a nivel de sistema operativo que abstrae los recursos computacionales a través de grupos de control (cgroups) para controlar los recursos del sistema a través de espacios de nombres creando y aislando los objetos dentro del contenedor [51], [52]. Los contenedores de LXC comparten el kernel con el sistema operativo host por lo que sus procesos y sistema de archivos son accesibles desde el host. Esta manera de funcionar se denomina virtualización ligera ya que no requiere emulación de hardware físico. El propósito principal de usar contenedores de Linux es proporcionar recursos o el mismo entorno, como una máquina virtual gracias a unas características de seguridad del núcleo, que le permite crear un entorno virtual en la misma máquina sin un hipervisor [51], [47].

La tabla 2 muestran diferencias ente las tecnologías Docker y LXC. Inicialmente, Docker estaba basado en LXC y libvirt y ofrecía una interfaz de espacio de usuario que permitía un entorno de contención del kernel de Linux y así trabajar como un contenedor. Sin embargo a partir de la versión 0.9, Docker se basa en la librería Libcontainer y esto supone un cambio en su forma de trabajar ya que no le limita a trabajar con LXC y otros paquetes externos. Libcontainer permite a los contenedores trabajar con espacios de nombres de Linux, grupos de control, perfiles de seguridad de AppArmor, interfaces de red y reglas de firewall de una manera coherente y predecible. No se basa en componentes del espacio de usuario de Linux como LXC, libvirt o systemd-nspawn y esto hace que libcontainer sea más estable y eficiente.

Requisitos	Descripción	VM	Docker	LXC
R4.1	Memoria usada	Alta	Baja	Baja
R4.2	Tamaño imagen	Alto	Bajo	Bajo
R4.3	Tiempo de arranque	Min.	Seg.	Seg.
R4.4	Seguridad	Alta	Media	Media
R4.5	Eficiencia	Media	Alta	Alta
R4.6	Servicios contenidos	>=1	1	>=1

Tabla 3: Diferencias de diferentes tecnologías de virtualización.

2.2.5 *Resumen*

La capacidad de contar con mecanismos que permitan encapsular y empaquetar servicios software es un elemento clave que facilita la gestión de los servicios sobre diferentes arquitecturas de servicios. Por lo general, las tecnologías utilizadas por las infraestructuras difieren unas de otras y no son compatibles. Esto implica que un servicio software que funcione perfectamente sobre un sistema TI al llevarlo a otro sistema puede ser que no llegue a funcionar. Los mecanismos de virtualización permiten resolver este problema utilizando VMs o contenedores para albergar los servicios en un entorno de ejecución perfectamente optimizado a sus necesidades. A pesar de que el uso de la virtualización puede ser una solución, no existe una tecnología de virtualización capaz de proporcionar todas las características técnicas idóneas para gestionar los servicios de manera óptima.

La tabla 3 muestra una comparativa de las principales herramientas de virtualización y sus características respecto a los requisitos identificados en la tabla 1. Como se puede ver, cada una de las tecnologías es capaz de aportar unas capacidades específicas que las hace idóneas para unas funciones determinadas. El análisis se puede dividir en dos grandes bloques, por un lado las máquinas virtuales (VMs) y por otro lado los contenedores. La virtualización basada en contenedores es más simple y considerablemente más ligero que las VMs [53], a pesar de que las VMS son más maduras, más seguras y más eficaz. Por lo tanto, una consideración principal para la virtualización basada en hipervisor con respecto a la virtualización basada en contenedores es el rendimiento general de las aplicaciones. Estudios como [54] han intentado realizar evaluaciones de rendimiento para cate-

gorías específicas de cargas de trabajo de aplicaciones que se ejecutan en contenedores frente a máquinas virtuales.

A pesar de los beneficios de la virtualización mostrados en esta sección, los ingenieros deben conocer y tener en cuenta las particularidades de cada una de las tecnologías de virtualización disponibles para escoger la tecnología que más le convenga para encapsular los servicios que serán desplegados. La posibilidad de que los propios sistemas sean capaces de encapsular los servicios sobre la tecnología más adecuada a la plataforma de despliegue es un tema poco explorado y que será tratado en esta tesis más adelante en el capítulo 5.

2.3 SOFTWARIZACIÓN

2.3.1 *Introducción*

La Softwarization es el proceso de transformar los equipos físicos tradicionales a los nuevos conceptos tecnológicos como son los dispositivos programables y la virtualización [55] [56]. La softwarización tiene como objetivo ofrecer nuevos servicios implementando eficientemente funciones con bajos gastos de capital (CAPEX) y gastos operativos (OPEX). En el caso de la infraestructura de dispositivos de red, NFV es una de las tecnologías de virtualización, que puede virtualizar todos los recursos de red y la infraestructura [57] proporcionando flexibilidad en la administración de recursos para crear las funciones de red (Network functions, NF) dinámicamente, controlar la capacidad de las funciones y escalar según el uso y la demanda [58] [59].

La softwarización de los dispositivos es cada vez más importante ya que los SPs no sólo han migrado una gran parte de sus servicios a la nube, sino que también han comenzado a migrar aplicaciones a la nube los cuales fueron diseñadas para funcionar en dispositivos físicos inteligentes. Los dispositivos de consumo, como teléfonos inteligentes, tabletas, televisores inteligentes, etc., actualmente contienen una gran cantidad de software que se puede mover fácilmente a la nube para mejorar la calidad del servicio. Sin embargo, esto no se queda aquí, si no que se propone mover la funcionalidad de los dispositivos físicos a la nube para aprovechar las ventajas de recursos y computación de la nube. Además de como ampliar las capacidades de los dispositivos mediante la reducción del consumo energético de los costos de mantenimiento [60].

Además de la virtualización de dispositivos y gracias a la virtualización de las redes domésticas, los proveedores de servicios pueden desplegar

en la nube las redes domésticas de los usuarios de manera transparente para éstos. A este escenario multimedia doméstico digital lo llamamos Red Personal Virtual del Hogar (Virtual Home Personal Network, VHPN).

Los VHPN proporcionan mecanismos para el intercambio de información utilizando los protocolos estándar de la capa 2 del modelo OSI [61]. Las VHPNs componen un ecosistema virtual similar al mundo físico donde los usuarios interactúan con las aplicaciones de software de una manera similar a como lo hacen con los dispositivos físicos conectados con las redes domésticas tradicionalmente, mientras que los SP son capaces de reducir CAPEX y OPEX.

2.3.2 *Virtualización de las redes de datos*

Uno de los ámbitos más beneficiados de la softwarización son las redes de comunicaciones se han visto muy beneficiadas por la posibilidad de virtualizar dispositivos físicos de las redes de comunicaciones. Esto ha sido un punto de inflexión ya que ha permitido eliminar gran parte de los dispositivos físicos que forman las redes de comunicaciones (tales como proxies, routers, hubs de conexiones, firewalls, etc), mediante la emulación por software de dichos dispositivos en forma de servicios de red. Esta tecnología de servicios de red, y los enlaces que los conectan entre sí, está definido por las tecnologías de las funciones de red virtualizadas (Network Function Virtualization, NFV) [5] y las redes definidas por software (Software Defined Networking, SDN) [62].

NFV y SDN son tecnologías complementarias que buscan el funcionamiento óptimo de las redes de servicios desde el punto de vista de la administración de red [63]. NFV es definida por ETSI y su objetivo es reemplazar y complementar los dispositivos de red física con funciones de red virtual [64]. Por ejemplo, es posible virtualizar las funciones de red a partir de elementos de red que existen físicamente en redes tradicionales como nodos residenciales o elementos de conmutación [64]. Con la virtualización de dispositivos físicos, los proveedores de equipos de red pueden innovar más rápido y reducir sus costos de fabricación [65] mediante la emulación de las funcionalidades de sus dispositivos a través de software. La Figura 6 ilustra un enfoque clásico del dispositivo de red (en el lado izquierdo) y un enfoque simple de la arquitectura NFV (en el lado derecho). El enfoque clásico muestra varios dispositivos de red físicos que se pueden comprar y formar redes y el enfoque NFV muestran la conexión entre los dispositivos de red virtual y los tipos de equipos de red. El equipo de red se despliega para formar un entorno de nube para que los VNF (Virtual Network Functions) funcionen. La tecnología SDN se encargaría de proveer los enlaces

y enrutamiento de red necesarios, para que conecten cada VNF y nodo virtualizado a través de la red virtual gestionada por el controlador de la SDN.

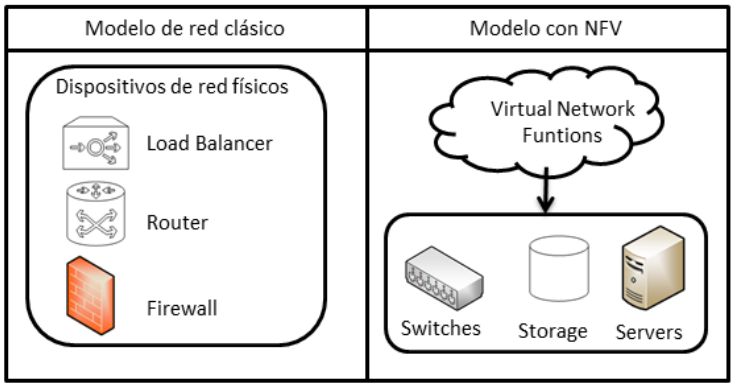

Figura 6: Comparación gráfica de los servicios de red alojados en infraestructura de red tradicional frente a otras basadas con tecnología NFV.

La gran flexibilidad de NFV y SDN permite desplegar servicios en arquitecturas dedicadas, distribuidas o incluso permite componer sistemas de servicios de comunicaciones distribuidos sobre múltiples centros de datos (Multi-access Edge Computing). Para gestionar el despliegue de servicios NFV y su conexión a través de las SDN, la ETSI ha propuesto un bloque funcional responsable para la orquestación y gestión del ciclo de vida de la infraestructura NFV llamado MANO (Management and Orchestration) [66] . MANO proporciona una arquitectura para la administración y coordinación de todos los recursos virtuales en los entornos informáticos virtualizados (almacenamiento, computación, red y diversos recursos virtualizados) y servicios de red [67] [68]. La arquitectura NFV MANO incluye tres componentes principales:

1. NFV Orchestrator (NFVO): Responsable de la gestión de nuevos paquetes de servicios de red (NS) y funciones de red virtual (Virtual network funcion, VNF); Administración de creación/operación/eliminación de NS; gestión de recursos/servicios; verificación y autenticación de operaciones de infraestructura de virtualización de funciones de red (NFVI).

2. Virtual Network Function Manager (VNFM): Supervisa la creación/operación/eliminación de instancias VNF; cooperación y el papel de ajuste entre NFVI y NFVO.

3. VIM: Regula y lleva a cabo el almacenamiento, la computación, la red y varios recursos virtualizados [69] [70] [71].

NFV y SDN fueron diseñados para añadir flexibilidad a las infraestructuras proporcionando un framework flexible y dinámico que permitiese gestionar las redes de una manera más eficaz. No sólo consiguieron su objetivo, sino que sentaron la bases para ir un paso más allá y plantear la posibilidad de llevar la virtualización de dispositivos físicos más allá de los dispositivos de red.

2.3.3 *Virtualización de dispositivos genéricos en servicios software*

Como hemos visto en el apartado anterior, la tecnología SDN-NFV ha sustituido satisfactoriamente gran parte de las arquitecturas de red de los sistemas tradicionales, todos los componentes que solían ser esenciales en la gestión de red tradicional ahora pueden verse como un servicio y se pueden suministrar a cualquier operador de red (virtual), proveedores de servicios o incluso dirigidos a los usuarios finales de la misma manera como si se tratase de un servicio común (por ejemplo: un firewall, espacio de almacenamiento, etc.). Este nuevo modelo de gestión de dispositivos de red a través de software ha ido cambiando la gestión de las arquitecturas de servicios. También ha cambiado la forma de entender los sistemas de TI y ha modelado la tendencia de usar arquitecturas de servicios software gestionados con un modelo de "Todo como servicio"(XaaS) en la nube [72].

El modelo XaaS va más allá de la tecnología NFV que se centra en el ámbito de los servicios de red. XaaS trata de emular cualquier dispositivo o servicio que sea necesario con el objetivo de proporcionar la funcionalidad deseada sobre un entorno en la nube. Esto abre un abanico de posibilidades a los proveedores de servicios debido a que es una manera de facilitar la administración de los sistemas de información, añadiéndoles flexibilidad, elasticidad y características de programación, todos ellos aprovisionados a través de software.

En este sentido, XaaS abre las puertas a un modelo de negocio nuevo donde los brokers de servicios digitales, en formato de mercado digital, mantienen una cartera de servicios de infraestructura listos para ser desplegados tras su previa compra. Así los operadores de servicios digitales acceden a un catalogo de servicios donde puedan buscar los servicios que necesiten antes de iniciar el desarrollo de aquellos servicios necesarios para su propio negocio. Un ejemplo dentro de un escenario de distribución de contenidos digitales lo podemos encontrar en la creación de instancias bajo demanda y la orquestación de servicios como: el almacenamiento en caché dinámico de datos de vídeo (CDNaaS), la descarga de tráfico específico (TOFaaS).

La viabilidad de la tecnología para modelar todo como un servicio ha venido de la mano del avance de tecnologías claves tales como la tecnología de virtualización y la tecnología de cloud computing. En lo apartados 2.2.2 e 2.4 se han presentado los aspectos más relevantes de estas tecnologías que han hecho posible esta nueva alternativa de ecosistema de servicios.

2.3.4 *Resumen*

En esta sección se ha comentado la importancia de los avances en la softwarizacion de dispositivos. De hecho, esta idea no es nueva y hay trabajos que proponen aproximaciones para llevar las funciones de los dispositivos customer-premises equipment (CPE) al núcleo de la red de la empresa de distribución de contenidos [73], [74], [75] y [76]. Esto está motivado por la tendencia de mover las funciones del dispositivo a la nube [73] y [74]. Trabajos como [73] y [74] proponen trasladar parte de las funciones del equipo de las instalaciones del cliente (CPE) al centro de datos ubicado en el núcleo de la red como un CPE virtual (vCPE). De esta manera se reduce la complejidad de los dispositivos alojados en el borde de la red, mientras que la mayor parte de la inteligencia permanece en la nube reduciendo las necesidades hardware de los dispositivos finales.

Sin embargo, la softwarizacion va más allá y trata de implementar las funciones desarrolladas tradicionalmente por los dispositivos físicos dentro de un entorno controlado completamente por software. La softwarización es un elemento clave para los sistemas de servicios futuros ya que permitirá prescindir de dispositivos físicos en ciertos escenarios de aplicación mientras que a su vez hace uso de los enormes beneficios que aporta en la gestión de los dispositivos softwarizados.

2.4 DESPLIEGUE DE SERVICIOS EN ENTORNOS CLOUD COMPUTING

2.4.1 *Introducción*

El tráfico en los sistemas de distribución de contenidos por Internet ha crecido exponencialmente en los últimos años. Esto se debe a que en los últimos años el ecosistema de los servicios digitales ha cambiado debido principalmente al aumento en el número de dispositivos móviles, a la necesidad de anchos de banda más exigentes para mejorar la calidad de los servicios y al desarrollo de nuevas aplicaciones y servicios, los cuales tienen que coexistir con las aplicaciones y servicios ya asentadas en el mercado lo que genera un tráfico de red adicional.

En el caso de los servicios dedicados a la distribución de contenidos tradicionalmente se le asocian tareas de: almacenamiento en caches distribuidas

geográficamente en la red, transcodificación de contenidos audiovisual y la gestión de distribución con calidad de dichos contenidos. Estas tareas pueden consumir grandes cantidades de recursos informáticos y de almacenamiento en intervalos de tiempo puntuales dependiendo de la demanda de cada momento [77]. Por lo tanto, estas tareas pueden ser muy dinámicas y necesitar gran cantidad de requisitos de recursos de manera no constante. Con el fin de gestionar los servicios de distribución de contenidos de manera rentable, cada vez más proveedores de servicios han adoptado la infraestructura en la nube debido a que puede satisfacer a las demandas de forma oportuna y asignar recursos informáticos, de comunicación y de almacenamiento de forma adaptable de acuerdo con los requisitos [78].

La calidad del servicio (QoS) en servicios de distribución de contenidos multimedia depende en gran medida de la arquitectura de red subyacente. De hecho, la integración de la tecnología de la nube en la distribución de contenidos es clave para lograr satisfacer la demanda. Por lo tanto, la tecnología en la nube es ideal para implementar cualquier arquitectura de servicios de las que se mostraron en el apartado 2.1.

Con el objetivo de mejorar el funcionamiento de los servicios en la nube, el modelo de aprovisionamiento servicios desde un único proveedor de la nube es un modelo de negocio que se quedó obsoleto por la llegada de otros modelos basados en la computación el borde de la red.

Los modelos basados en la computación en el borde de la red tienen el objetivo de mover centros de datos a ubicaciones geográficas cerca de los usuarios finales para reducir la latencia de red y mejorar la experiencia del usuario final [79]. Sin embargo, esta tendencia requiere de un alto gasto de capital (CAPEX) y altos gastos operativos (OPEX). Asumir estos gastos no es realista y la mayoría de los proveedores de servicios de distribución de contenidos no construyen los sistemas por sí solos si no que se apoyan en nubes de otros proveedores. Esto reduce sus gastos y les permiten ofertar sus servicios creando una gran nube conocida como nube híbrida. Por lo tanto, la nube híbrida (ver Figura 7) es una alternativa rentable para los proveedores de servicios [80] la cual les permite extender rápidamente sus servicios a múltiples ubicaciones geográficas con garantías de QoS.

Sin embargo, para gestionar bien la nube híbrida es necesario abordar correctamente varios problemas. En primer lugar, se necesita administrar los servidores en la nube privada y en la nube pública de forma coordinada. En segundo lugar, dado que la nube híbrida está distribuida geográficamente, es necesario realizar la administración de la nube híbrida desde un enfoque con mecanismos que tomen en cuenta la distribución de los servicios para una mejor escalabilidad. Para hacer frente a estas soluciones es necesario disponer de mecanismos que facilita la interoperabilidad a través de APIS estandarizadas.

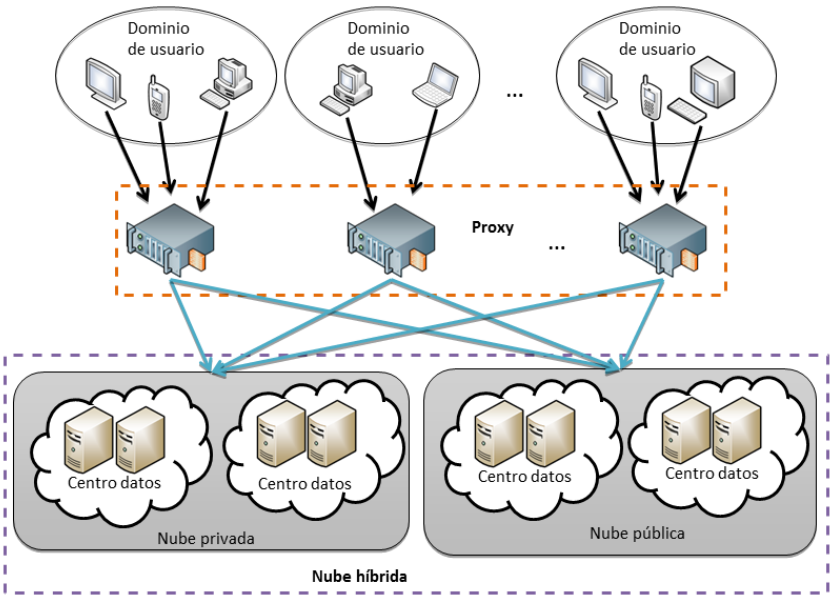

Figura 7: Diferencias entre los modelo de arquitecturas de la nube privada, la nube pública y la nube híbrida.

En los subapartados siguientes se van a describir algunos aspectos a tener en cuenta y necesarios para facilitar la gestión remota e interoperabilidad de sistemas en la nube. El objetivo es poder trabajar de manera coordinada y de manera óptima respecto a capas de más alto nivel en la que se alojan los servicios.

2.4.2 *Capas software en la nube*

Todo sistema informático basado en cloud computing está formado por un conjunto de capas software con objetivos y características técnicas especificas. Cada una de ellas provee funcionalidades particulares a otros componentes de la misma capa, o incluso a componentes que pertenecen a otra capa [81]. Estas capas se clasifican en: capa de aplicaciones, capa de plataforma de ejecución (integra: servidores, servicios comunes de aplicaciones, frameworks, etc.) y capa de infraestructura. La Figura 8 muestra la disposición de las 3 capas.

En la capa de aplicaciones se pueden encontrar las aplicaciones y servicios que proporcionan funcionalidades de interés para las organizaciones o usuarios finales. En la capa de Plataformas se encuentran aquellos servicios o aplicaciones orientadas fundamentalmente a proporcionar servicios y capacidades de interés a los componentes software instalados en la capa de aplicación. Finalmente, en la capa de infraestructura se encuentran los

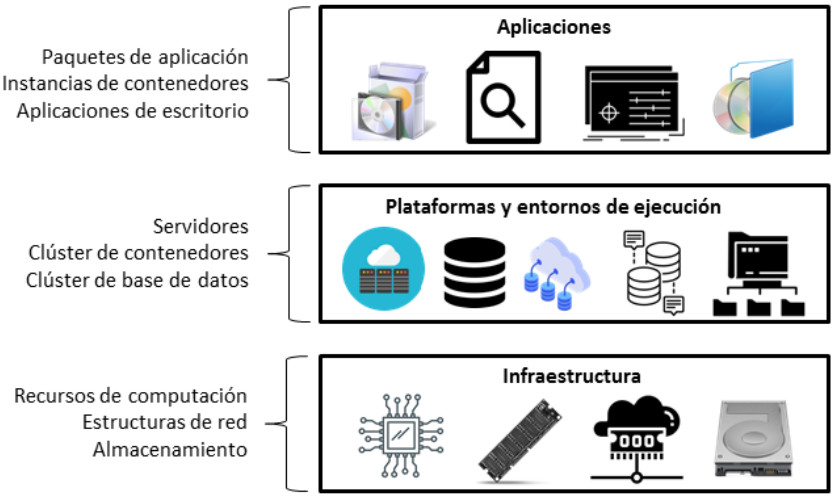

Figura 8: Capas software que componen una arquitectura software basado en infraestructura sobre la nube.

recursos de infraestructura y las herramientas y servicios que administran estos recursos.

2.4.3 *Infraestructura Cloud Computing y recursos disponibles*

En la computación en la nube, existen tres recursos esenciales que se suelen tener en cuenta a la hora de proporcionar el aprovisionamiento de los servicios: computación, almacenamiento y redes. A partir de estos recursos, los proveedores de servicios son capaces de combinarlos y empaquetarlos de diferentes maneras, dando lugar a máquinas virtuales y contenedores capaces de aprovisionar servicios. Incluso también podrían formar otros recursos complejos los cuales están compuestos de recursos básicos y que podrían llegar a ser considerados como recursos compuestos o de alto nivel. Un ejemplo de estos recursos podrían ser: Base de datos como servicio (DBaaS), balanceadores de carga, servidores de nombres de red (DNS), gestión de identidades, etc. Con el objetivo de sentar una base de conocimiento de cara a secciones posteriores a continuación se describen brevemente los recursos básicos que componen cualquier sistema de las tecnologías de la información:

- **Recursos computacionales** Los recursos de computación básicamente ejecutan código. El recurso de computación se puede ofrecer a través de: máquinas virtuales (VM), servidores físicos, clústeres de servidores, clústeres de hospedaje de aplicaciones y tiempos de ejecución de código sin servidor (FaaS).

- **Recursos de almacenamiento** Los recursos de almacenamiento son los encargados de proveer mecanismos de almacenamiento de información a través de: volúmenes de disco, bases de datos o repositorios centrales para archivos.
- **Recursos de red** Los recursos de red permiten aprovisionar y cambiar las redes a petición desde el código fuente que define la estructura de la red. Esto ofrece muchos beneficios, por ejemplo: las redes definidas por software (Software Defined Networking, SDN) [62] las cuales no sólo permiten crear y desplegar estructuras de redes sino que también permite aplicar seguridad en las redes desplegadas.

2.4.4 *Lenguajes de especificación de infraestructura*

Antes de la computación de la nube, los administradores del sistema han estado utilizando durante décadas los scripts para automatizar las tareas de administración de infraestructura. Los lenguajes de scripting de uso general como Bash, Perl, PowerShell, Ruby y Python siguen siendo una parte esencial del kit de herramientas de un equipo de infraestructura. Sin embargo, con el tiempo aparecieron nuevas necesidades y se ha ido extendiendo el uso de lenguajes declarativos específicos de dominio para la administración de infraestructura.

Los lenguajes específicos de dominios (Domain Specific Language, DSL) [82] son usados para describir las infraestructuras en cualquier entorno en la nube. Es decir, es un modelo destinado a describir una infraestructura en un formato declarativo utilizando una abstracción de alto nivel de componentes de infraestructura.

La gran aportación de los lenguajes declarativos fue simplificar el código de infraestructura, ya que separaba la definición de la infraestructura deseada de las tareas que debían realizarse para conseguir la infraestructura deseada. Es decir, los desarrolladores especifican la infraestructura final que desean y la herramienta se encarga de ejecutar las tareas necesarias para conseguir aquella infraestructura.

Los lenguajes declarativos no es la única alternativa para definir código de infraestructura. Existen herramientas para gestionar infraestructuras que utilizan lenguajes de programación de propósito general (lenguajes imperativos). Los lenguajes imperativos destinado a la gestión de infraestructuras surgen de la necesidad de cubrir limitaciones de los lenguajes declarativos a la hora de desplegar infraestructuras. Como se ha comentado anteriormente, los lenguajes declarativos basan su funcionamiento en la especificación del estado final de la infraestructura y con esta información la herramienta correspondiente es capaz de realizar el despliegue de

dicha infraestructura. Sin embargo, en el caso de los lenguajes imperativos, los desarrolladores deben implementar un conjunto de instrucciones para especificar cómo debe desplegarse la infraestructura deseada.

Por lo general, el código que define grandes infraestructuras esta compuesto por una mezcla de código declarativo e imperativo. Esto no es un error en la gestión de la administración del código, sino que es necesario ya que cada uno de los dos paradigmas tiene características que le hace idóneo para gestionar ciertas particularidades de las infraestructuras. La definición de un código de infraestructura implica muchas tareas diferentes como la definición de recursos de infraestructura, configuración de diferentes instancias de recursos y organización del aprovisionamiento de múltiples piezas interdependientes de un sistema. Algunas de estas necesidades pueden expresarse simplemente utilizando un lenguaje declarativo. Sin embargo, en ocasiones las tareas son más complejas y se manejan mejor con un lenguaje imperativo. Saber en qué momento se debe utilizar un paradigma u otro es una labor que requiere experiencia en el campo y requiere un amplío conocimiento para saber donde trazar los límites.

Para esta tesis, los DSLs son importantes porque definen infraestructuras de sistemas facilitando la escritura de código, su comprensión permitiendo realizar adaptaciones y composición de sintaxis compatibles con varias herramientas de la industria actual. Un ejemplo de pseudocódigo que configura un servidor a través de DSL se muestra a continuación:

Cuadro 1: Ejemplo de lenguaje DSL en el que se especifican los paquetes a instalar así como el despliegue que debe hacerse de la aplicación

```
package: jdk
package: tomcat

service: tomcat
port: 8443
user: tomcat
group: tomcat

file: /var/lib/tomcat/server.conf
owner: tomcat
group: tomcat
mode: 0644
contents: $TEMPLATE(/src/appserver/tomcat/server.conf.template)
```

Este código garantiza que se instalen dos paquetes de software, Java Development Kit (JDK) y servidor de aplicaciones Tomcat. El código mostrado define un servicio que se debe instalar y ejecutar, incluido el puerto que escucha además del usuario y el grupo de usuarios con permisos para ejecutarse. Por último, el código define que se debe crear un archivo de

configuración de servidor a partir de un archivo de plantilla. Dependiendo de la herramienta de gestión de la configuración que se quiera utilizar se deberá adaptar el formato a dicha herramienta. Por ejemplo, Ansible [32], Chef [83] y Puppet [31] son herramientas de gestión de configuración donde cada uno tiene su propio DSL para configurar máquinas. En todos ellos, los lenguajes proporcionan mecanismos para gestionar paquetes, archivos, servicios, cuentas de usuario, etc.

2.4.5 *Herramientas para la administración de recursos y componentes SW entre proveedores de entornos en la nube*

Existen estándares de administración de recursos en la nube que proporcionan modelos de descripción para los recursos en la nube. Estos modelos describen los recursos de la nube y, por lo tanto, permiten la interoperabilidad entre ellos. Entre los estándares más conocidos se encuentran CIMI [84] (Cloud Infrastructure Management Interface), OCCI [85] (Open Cloud Computing Interface) y TOSCA [86] (Topology and Orchestration Specification for cloud Applications) entre otros. A continuación se describe brevemente los estándares de interoperabilidad más relevantes para esta tesis:

Cloud Infrastructure Management Interface (CIMI)

CIMI es un estándar de Distributed Management Task Force (DMTF) que describe formas estandarizadas de configurar, describir y administrar recursos de infraestructura en la nube [84] y algunos casos de uso en [87]. CIMI define los siguientes tipos de recursos básicos en el nivel de infraestructura:

I Máquina, que representa un recurso informático que encapsula CPU y memoria,

II Red, que representa un servicio lógico interconectado para reenviar el tráfico de datos entre puntos finales, y

III Volumen: representando un recurso de almacenamiento en el nivel de bloque o en el nivel del sistema de archivos.

CIMI también define asociaciones para conectar estos tipos de recursos: MachineNetworklnterface entre máquina y red y MachineVolume entre máquina y volumen.

Topology and Orchestration Specification for Cloud Applications (TOSCA)

TOSCA es una especificación propuesta por Organization for the Advancement of Structured Information Standards (OASIS) que permite definir la topología requerida de una aplicación de software en la nube además

de los recursos de infraestructura en la nube. Se centra en definir la topología de la aplicación para que sea procesada por el orquestador de nube. No determina cómo se implementa la topología deseada en la nube si no que se centra en la creación y gestión semiautomática de servicios complejos proporcionando un lenguaje para describir las relaciones entre los componentes (mediante una topología de servicio) y sus comportamientos operativos (como proceso de orquestación) [88].

TOSCA proporciona un formato de intercambio estandarizado, bien definido, portátil y modular para la estructura de los componentes de la aplicación, las relaciones entre ellos y sus correspondientes funcionalidades de administración. TOSCA permite la implementación automatizada completa y otras funcionalidades de administración, como escalar o realizar copias de seguridad de las aplicaciones a través de la combinación de los dos conceptos principales de TOSCA: (1) topologías de aplicaciones y (2) planes de administración. Las topologías de aplicación proporcionan una descripción estructural de la aplicación, los componentes en los que consta y las relaciones entre ellos. Cada nodo va acompañado de una lista de operaciones que ofrece para gestionarse a sí mismo. Por lo tanto, la topología no es sólo una descripción de los componentes de la aplicación y sus relaciones, sino también una declaración explícita de sus capacidades de administración.
Los planes de administración combinan estas capacidades de administración para crear tareas de administración de nivel superior, es decir tareas complejas formadas por tareas más simples. Esto permite ejecutar tareas totalmente automatizadas que permitirán realizar operaciones avanzadas sobre las aplicaciones, por ejemplo tareas de implementación, configuración y administración. La Figura 9 presenta una descripción abstracta de una aplicación basada en TOSCA donde se muestra los dos conceptos principales de TOSCA y su relación. Es decir, aparece la topología de la aplicación contiene nodos, que están conectados por relaciones y además aparece el plan de administración el cual se inicia mediante una operación de administración de mensajes y llamadas externa de los nodos de la topología.

TOSCA se ha utilizado ampliamente para la orquestación basada en modelos en la informática en la nube. Cloudify y OpenStack Heat han sido dos de los primeros en adoptar la especificación TOSCA para permitir implementaciones automáticas de servicios complejos en múltiples proveedores de nube. La versión inicial (versión 1.0) de TOSCA, se basa en Extensible Markup Language (XML) Schema 1.0 el cual se publicó en noviembre de 2013 [88]. Más recientemente, en diciembre de 2016, OASIS publicó una versión más reciente que proporcionaba la especificación TOSCA en una representación de Yet Another Markup Language (YAML) con el objetivo de simplificar la creación de plantillas de servicio TOSCA [90].

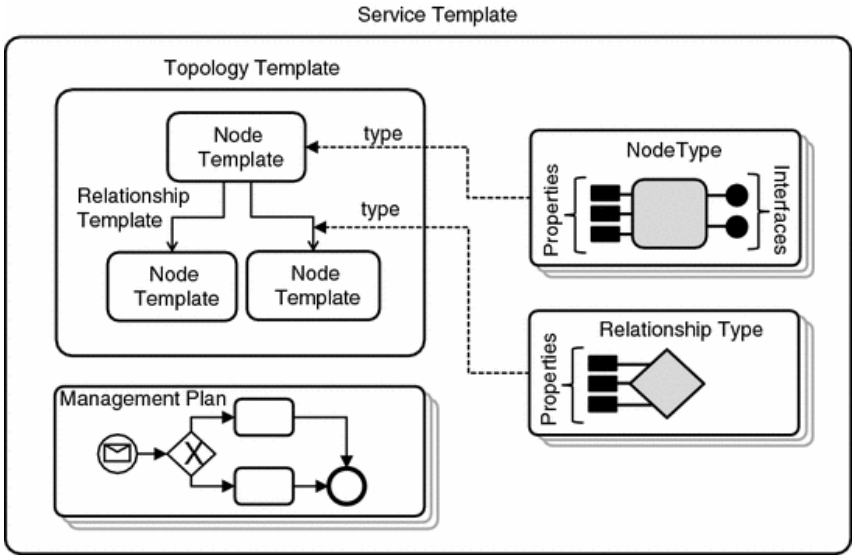

Figura 9: Definición de la plantilla de servicios en el estándar TOSCA adaptado de [89]. En la figura se puede ver que la plantilla de servicio se compone de una plantilla de topología compuesta por nodos relacionados entre sí. Los nodos y las relaciones entre ellos están definidos por su tipo. La plantilla de servicio también incluye el plan de administración.

TOSCA también define una solución de empaquetado para simplificar la portabilidad de varios archivos entre diferentes motores de orquestación compatibles con TOSCA llamado ToSCA Cloud Service Archive (CSAR) [91]. CSAR comprende la plantilla de servicio TOSCA, así como archivos de configuración, código de software e imágenes necesarias para la creación de instancias de un servicio en particular.

Open Cloud Computing Interface (OCCI)

OCCI [92] comprende un conjunto de especificaciones inicialmente publicadas alrededor de 2009 a través de la organización Open Grid Forum (OGF). El objetivo principal de la especificación OCCI es proporcionar un conjunto de APIs para administrar de forma remota los recursos en la nube proporcionados por diferentes proveedores. La versión inicial de la especificación OCCI se refería principalmente a la capa de IaaS que proponía una API basada en REST. En estos modelos, los recursos se identifican mediante un localizador uniforme de recursos (URL) y los usuarios pueden interactuar con esos recursos mediante métodos estándar del Protocolo de transferencia de hipertexto (HTTP). Los recursos también se pueden vincular en función de las necesidades del usuario.

La arquitectura OCCI es bastante modular y extensible y consta de tres módulos principales:

1. OCCI Core representa el módulo central del estándar, proporciona mecanismos y semántica a través de una ontología que proporciona

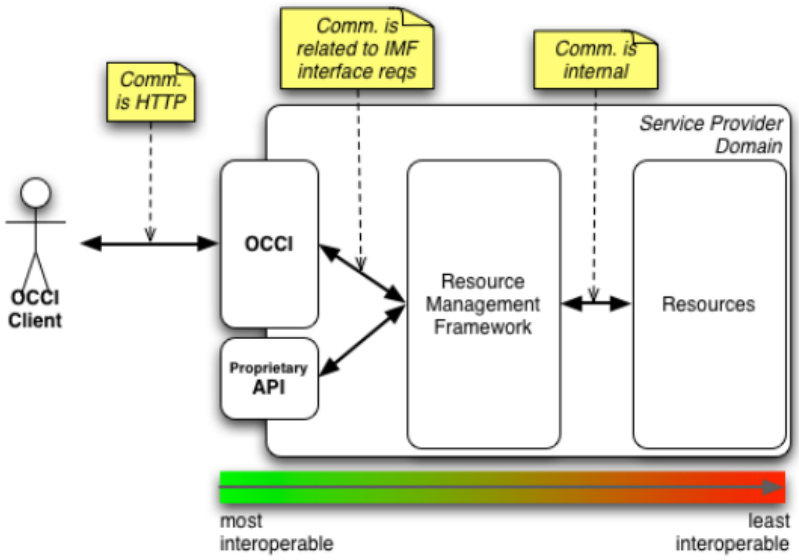

Figura 10: Modelo de arquitectura OCCI desde [93]. La figura muestra la arquitectura simplificada de un proveedor de infraestructura compuesto por un conjunto de recursos, un framework encargado de gestionar los recursos de la infraestructura y dos APIs de conexión que conectan a dicho framework. Estas APIs proporcionan conexión externa al framework a través del estándar OCCI o a través de una API privada.

definiciones y clases para describir y administrar recursos en la nube. Este modelo es independiente de los demás y podría utilizarse como un componente independiente en diferentes contextos. Proporciona una abstracción de los recursos reales mediante la clase Resource proporcionando un conjunto de atributos comunes entre diferentes tipos de recursos, interconectados a través de la clase Link. La clase abstracta Entity se utiliza junto con las clases Kind y Category para clasificar e identificar recursos en varios proveedores de nube.

2. OCCI Infrastructure representa una extensión del módulo Core que proporciona una abstracción específica para la capa IaaS. Los tres tipos de infraestructura definidos por este módulo son los recursos de proceso, red y almacenamiento que están conectados a través de NetworkInterface o StorageLink.

3. OCCI RESTFul HTTP Rendering describe el formato de serialización utilizado para la comunicación entre el cliente y el servidor. Se basa en el concepto de arquitectura orientada a recursos mediante el protocolo HTTP para identificar y controlar los recursos proporcionados por un proveedor de nube.

Las principales motivaciones detrás de la especificación OCCI son: la interoperabilidad, la portabilidad, la integración, la innovación y la reutilización.

Cloud Data Management Interface (CDMI)

CDMI especifica una interfaz de administración de datos en la nube a través de la definición de una interfaz funcional que las aplicaciones usarán para crear, recuperar, actualizar y eliminar elementos de datos de la nube. CDMI utiliza un api REST para ofrecer la implementación del estándar [94]. El estándar define un contenedor y un objeto de datos como artefactos principales con los cuales pueden implementarse diferentes funcionalidades. A continuación se enumeran algunas de las operaciones que se pueden llevar a cabo con los artefactos antes mencionados:

- descubrir las capacidades de un proveedor de almacenamiento en la nube
- creación de un nuevo contenedor
- creación de un nuevo objeto de datos
- enumerar el contenido de un contenedor
- leer el contenido de un objeto de datos
- leer sólo el valor de un objeto de datos
- eliminar un objeto de datos

Open Virtualization Format (OVF)

OVF [95] describe un formato abierto, seguro, portátil, eficiente y extensible para el empaquetado y distribución de software que se ejecutará en máquinas virtuales. OVF es mantenido por el Grupo de Trabajo de Administración de Distribución (DMTF) [96]. Un descriptor OVF no solo puede contener una máquina virtual, sino varias máquinas virtuales, denominadas sistemas de máquinas. De acuerdo con la especificación, un paquete OVF puede contener los siguientes artefactos:

- un descriptor OVF con extensión .ovf
- puede contener un manifiesto ovf con extensión .mf
- puede contener un certificado OVF con extensión .cert
- puede contener archivos de imagen de disco
- puede contener archivos de recursos adicionales, como imágenes ISO

Además, OVF proporciona la capacidad de definir una cantidad de máquinas virtuales de memoria y cpu, así como la configuración de red. El archivo de manifiesto contiene resúmenes SHA-1 de todos los archivos para garantizar la integridad a lo largo de varias transferencias. Además, para garantizar la autenticidad, el archivo de manifiesto se puede firmar. Todos los artefactos se pueden comprimir en un dispositivo (.ova como final de archivo) para facilitar la distribución. Todos los principales productos de virtualización admiten la importación y exportación de archivos OVF.

Cloud Application Management for Platforms (CAMP)

CAMP [97] intenta estandarizar la interacción con los proveedores de PaaS. El estándar propuesto por Organization for the Advancement of Structured Information Standards (OASIS) y utiliza interfaces REST para proporcionar su funcionalidad.

Los recursos administrados por el sistema se dividen en plataforma, componentes de plataforma, componentes de aplicaciones y conectores. Una plataforma describe diferentes aspectos relacionados con la nube. Un componente de plataforma es un servicio distinguido ofrecido por la plataforma. Los componentes de la aplicación configuran partes de una aplicación. Todos los componentes (y recursos) de la aplicación juntos forman un paquete. Además de estos recursos CAMP también cubre todo el ciclo de vida de una aplicación.

Al igual que OVF, CAMP define un paquete de implementación de plataforma (Platform Deployment Package, PDP) para entregar los datos de una aplicación de forma estructurada. Este paquete normalmente contiene los siguientes elementos:

- un archivo plan de implementación con extensión .dp.
- puede contener un archivo de manifiesto con la extensión .mf.
- puede contener un certificado con la extensión .cert.
- puede contener paquetes específicos de idioma.
- puede contener de recursos adicionales.

El archivo más importante es el plan de implementación el cual contiene toda la información sobre cómo configurar el entorno necesario para la aplicación. Proporciona plantillas y configuraciones para los componentes de la plataforma y define las dependencias necesarias.

Deltacloud

Deltacloud [98] es una interfaz de programación de aplicaciones (API) desarrollada para abstraer las diferencias entre las implementaciones de computación en la nube. Por lo general, cada proveedor de infraestructura

en la nube proporciona una API adaptada para su gestión, el propósito de Deltacloud es proporcionar una API unificada basada en REST que se pueda usar para administrar servicios en cualquier nube. DetlaCloud también es compatible con CIMI (Cloud Infrastructure Management Interface) 2.4.5.

Apache Libcloud

Apache Libcloud [99] ofrece a los administradores y programadores una interfaz para acceder a diferentes proveedores de IaaS. El uso de LibCloud permite no tener que preocuparse por las diferencias de API entre los proveedores permitiendo cambiar el proveedor sea más fácil y elimina la necesidad de tener que sufrir la curva de aprendizaje de APIs externas antes de poder cambiar a un nuevo proveedor. La desventaja es que no todas las características especializadas están disponibles a través de la interfaz y en el caso de incorporar nuevas funcionalidades la comunidad de LibCloud tarda un tiempo en incorporar los cambios necesarios.

2.4.6 *Herramientas que proporcionan gestión y control de la infraestructura de recursos en la nube*

En la sección 2.4.5 se han analizado diferentes soluciones en forma de librerías y APIs REST diseñadas para la gestión y control de software y recursos de infraestructura. El objetivo de esas soluciones es proveer mecanismos que faciliten el control y gestión de esos elementos de una manera común y estandarizada para facilitar la integración con otros sistemas. Estas soluciones además de conseguir de proporcionar un marco de referencia para facilitar la gestión de estos elementos también han servido como base para la construcción de herramientas más avanzadas. Estas herramientas consisten en sistemas complejos que además de implementar las funcionalidades de las soluciones mostradas en 2.4.5 también implementan mecanismos más complejos capaces de proporcionar otras funcionalidades más avanzadas tales como tareas de orquestación, sistemas de escalado automático, etc. Además, algunos de estos sistemas completos integran herramientas de las soluciones de la sección anterior y así pueden proporcionar funcionalidades de gestión relacionadas desde los recursos de más bajo nivel al software situado a nivel de aplicación. A continuación se describen las herramientas más relevantes:

OpenNebula

OpenNebula [100] es una solución sencilla pero muy completa y flexible para crear y administrar nubes privadas. OpenNebula combina las tecnologías de virtualización existentes con características avanzadas para multitenan, aprovisionamiento automático y elasticidad. OpenNebula es uno de los más populares de código abierto para la gestión de la nube y ha sido utilizado activamente por muchas instituciones, académicos, investigadores y

empresas [101] [102] [103] [104]. OpenNebula asume que la infraestructura física adopta una arquitectura clásica similar a un clúster con un front-end y un conjunto de hosts donde se ejecutarán máquinas virtuales y dispone de una red física que une todos los hosts con el front-end. En OpenNebula los componentes básicos son:

- Front-end. Ejecuta los servicios OpenNebula.
- Hosts habilitados para hipervisor. Proporcionan los recursos necesarios para las máquinas virtuales.
- Almacenes de datos. Contienen las imágenes base de las máquinas virtuales.
- Redes físicas. utilizadas para admitir servicios básicos como la interconexión de los servidores de almacenamiento y las operaciones de control de OpenNebula, y VLAN para las máquinas virtuales.

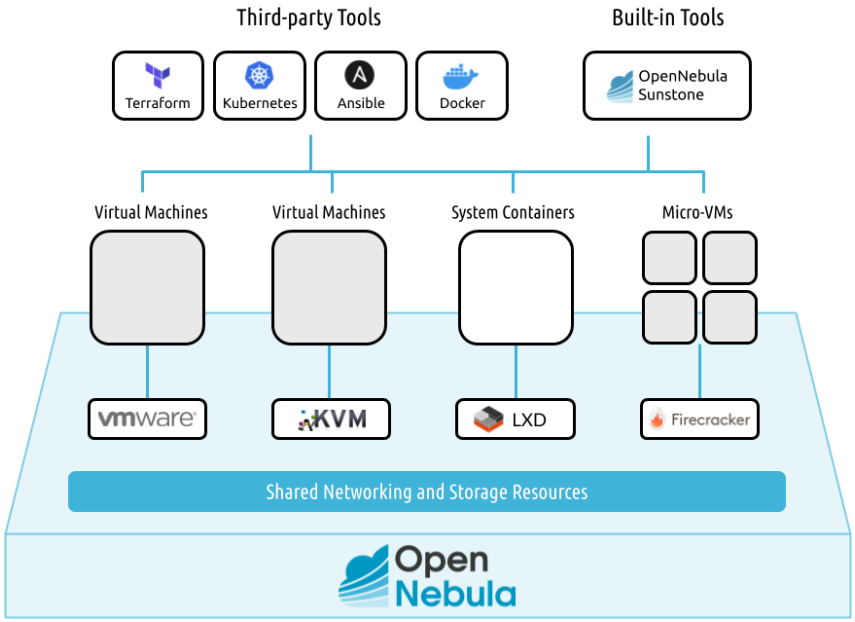

Figura 11: Modelo de arquitectura de OpenNebula [105]. La figura muestra la distribución de los componentes que la forman y las tecnologías compatibles.

La arquitectura de OpenNebula es modular y ofrece soporte para hipervisores, monitorización, almacenamiento, redes y servicios de gestión de usuarios. Además, es capaz de administrar infraestructura virtual con la capacidad de implementar servicios virtualmente ya sea en una nube externa o a nivel local dentro de un grupo de recursos [106]. OpenNebula

también ofrece interfaces que permiten acceder a todas sus funcionalidades, tanto a nivel de control del sistema como del software desplegado. Estas interfaces se denominan: interfaz de nube e interfaz del sistema [100] las cuales se describen a continuación:

- La interfaz en la nube permite al usuario final administrar la máquina virtual, la imagen y la red de una manera sencilla.
- La interfaz del sistema describe como toda la funcionalidad de OpenNebula que gestiona todos los componentes y elementos dentro y fuera de ella. Por ejemplo, la interfaz XML-RPC que administra el recurso de OpenNebula, incluida la red, los clústeres, las imágenes, la máquina virtual, los hosts y los usuarios [100].

OpenStack

OpenStack [107] es un conjunto de herramientas de software para crear y administrar plataformas de computación en la nube públicas y privada. OpenStack es además una colección de proyectos de software de código abierto que proporciona una solución de infraestructura como servicio (IaaS) a través de un conjunto de servicios interrelacionados. OpenStack ofrece facilidades para implementar máquinas virtuales (VM) y otras instancias que controlan diferentes tareas para administrar un entorno en la nube. Facilita los procesos de escalado horizontal facilitando el funcionamiento de las aplicaciones que requieren adaptarse dinámicamente la carga de trabajo. OpenStack consta de una serie de proyectos interrelacionados escritos en Python que ofrecen todos los servicios principales que un software de IaaS debe ofrecer. Los principales proyectos pertenecientes a OpenStack son [109]: servicio de computación (Nova), servicio de almacén de objetos (Swift), servicio de almacenamiento en bloques (Cinder), servicio de red (Neutron), panel de control (Horizon), servicio de identidad (Keystone), servicio de imágenes (Glance) y servicio de métricas (Ceilómetro). La Figura 12 muestra la arquitectura de OpenStack. A continuación se describen cada uno de los componentes:

- Compute (Nova): OpenStack Compute (Nova) es un controlador de computación en la nube el cual se utiliza para implementar y administrar un gran número de máquinas virtuales y otras instancias para gestionar tareas informáticas.
- Almacenamiento de objetos (Swift): OpenStack Object Storage (Swift) es un sistema de almacenamiento redundante y escalable para objetos y archivos. Los objetos, así como los archivos, se escriben en varias unidades de disco repartidas por servidores del centro de datos, el software OpenStack solo es responsable de garantizar la replicación e integridad de los datos sobre el clúster.

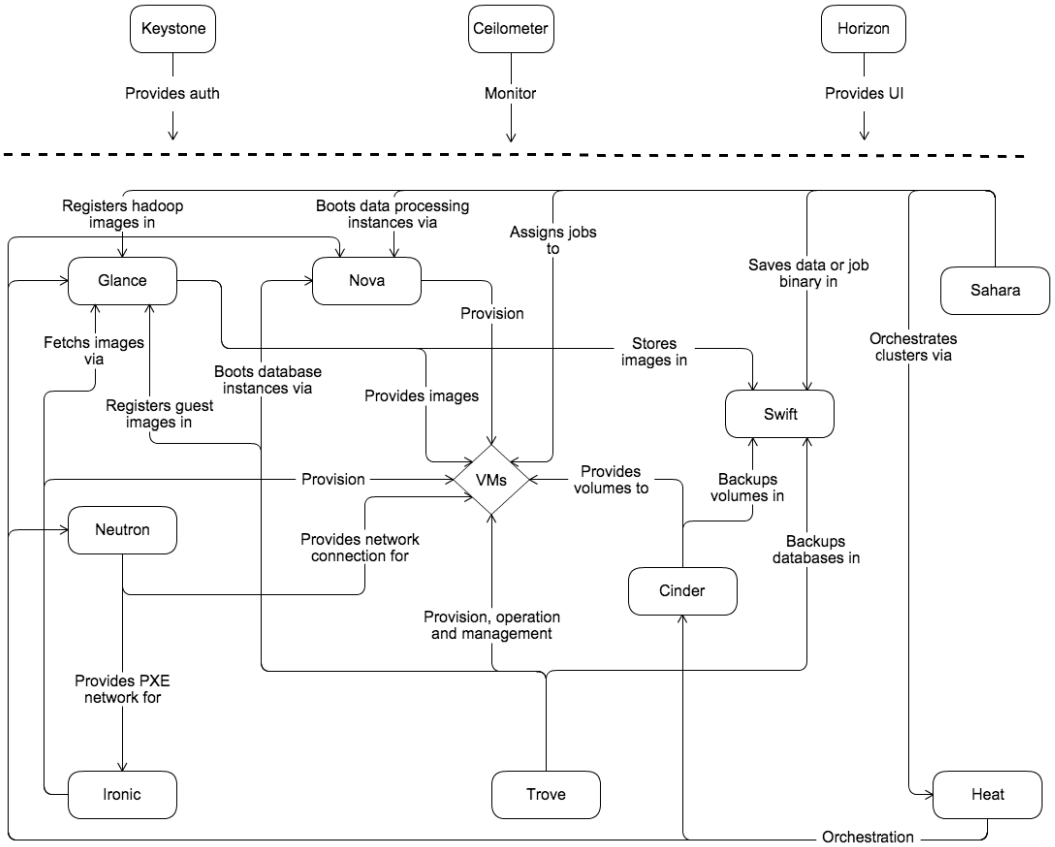

Figura 12: Modelo de arquitectura conceptual de OpenStack [108]. La figura muestra la distribución de los componentes que la forman y como se relacionan unos con otros.

- Almacenamiento en bloques (Cinder): OpenStack Block Storage (Cinder) es un componente de almacenamiento en bloque, que es más análogo a la noción tradicional de que un equipo puede acceder a ubicaciones específicas en una unidad de disco, también proporciona dispositivos de almacenamiento persistentes a nivel de bloque para su uso con instancias de proceso de OpenStack. En OpenStack, el sistema de almacenamiento en bloque administra la creación, conexión y desconexión de los dispositivos de bloques a los servidores.

- Redes (Neutrones): OpenStack Networking (Neutron) es la parte encargada de proporcionar la capacidad de comunicación del sistema. además permite administrar las redes y direcciones IP de forma fácil, rápida y eficiente.

- Dashboard (Horizon): OpenStack Dashboard (Horizon) es el panel para la gestión de OpenStack que proporciona a los administradores y

usuarios una interfaz gráfica para acceder, aprovisionar y automatizar los recursos basados en la nube.

- Servicio de identidad (Keystone): OpenStack Identity (Keystone) proporciona servicios de identidad para OpenStack a través de un directorio central de usuarios los cuales están asignados a los servicios de OpenStack a los que pueden acceder. Proporciona varios medios de acceso y actúa como un sistema de autenticación común en todo el sistema operativo en la nube y puede integrarse con otros servicios de directorio de terceros.
- Servicio de imágenes (Glance): OpenStack Image Service (Glance) proporciona servicios para gestionar imágenes a OpenStack, es decir, servicios de detección, registro y entrega para imágenes de disco y servidor. Además también permite que estas imágenes se usen como plantillas al implementar nuevas instancias de máquina virtual.
- Telemetría (Ceilómetro): OpenStack Telemetry Service (Ceilometer) proporciona servicios de telemetría que permiten a la nube proporcionar servicios de facturación a usuarios individuales de la nube, monitoriza el uso del sistema de cada usuario y también de cada uno de los diversos componentes de una nube Openstack.
- Orquestación (Heat): OpenStack Orchestration (Heat) es un servicio que permite a los desarrolladores almacenar los requisitos de una aplicación en la nube en un archivo que define qué recursos son necesarios para esa aplicación.

OpenVim

OpenVIM [110] es una arquitectura sencilla, ya que solo hay un servicio que admite las operaciones fundamentales. OpenVIM proporciona una herramienta CLI (Command Line Interface), denominada openvim, que se puede ejecutar en la misma máquina virtual o en una máquina virtual diferente.

También dispone de una API RESTful, lo que permite que los servicios que la utilizan mejoren la interoperabilidad, escalabilidad, manipulación de recursos, portabilidad y confiabilidad. Esta API permite la administración del ciclo de vida de varios elementos, como: redes, instancias e imágenes. OpenVIM también incorpora una base de datos MySQL [110].

El servicio OpenVIM, también denominado openvimd, es el elemento clave de la arquitectura lógica. Este servicio, permite crear, eliminar y almacenar máquinas virtuales, imágenes y redes. Cada tipo de operación se delega en un subproceso específico: servidor HTTP, controlador DHCP, controlador OpenFlow (OFC), administración de bases de datos y host [110]. OpenVIM se puede ejecutar en cinco modos de operación diferentes:

- De forma predeterminada, se ejecuta en modo de "test", donde los nodos de proceso y el controlador OpenFlow (OFC) no son necesarios. Este modo no implica una implementación real y se utiliza solo para realizar pruebas de la API y la base de datos.
- El modo “normal” implica un comportamiento normal, lo que significa que se requiere un OFC y uno o más hosts reales.
- El modo “host only” se puede utilizar cuando no se proporciona el controlador OpenFlow (OFC). Puede funcionar utilizando un solo nodo de proceso y conexiones al plano de datos creadas manualmente.
- El modo de “develop” utiliza una implementación basada en la nube con un tamaño de memoria mediano y una red de puente como la OFC no es necesaria.
- El modo “OF only” garantiza la prueba de la controlador OpenFlow (OFC) sin necesidad de hosts informáticos [110].

2.4.7 *Resumen*

Como se ha comentado anteriormente, la infraestructura de TI es una combinación de hardware, software, recursos de red y servicios necesarios para la implementación, el mantenimiento y la administración del entorno de aplicaciones. La implementación de infraestructuras complejas con muchos requisitos de hardware y software se convierte en un problema y causa errores si se utilizan herramientas de implementación tradicionales (tales como scripts de configuración), ya que estas herramientas requieren tomar en cuenta características específicas [111] para poder realizar correctamente el despliegue del software.

Esta situación ha sido tenido en cuenta por los proveedores de nube y han desarrollado diferentes recursos y servicios, disponibles en la propia nube, para facilitar a sus clientes la implementación y despliegue de la infraestructura. Sin embargo, a pesar de la existencia de estas herramientas desarrolladas por los proveedores de la nube, los administradores de los sistemas en la nube deben afrontar tareas complejas y además deben hacer frente a la gestión de sistemas que incrementan la dificultad de gestionarlos correctamente debido a la creciente complejidad de las aplicaciones y de los requisitos de las mismas. Esta situación genera problemas de:

- heterogeneidad de los recursos en la nube.
- falta de compatibilidad con el modelo de recursos de los diferentes proveedores de nubes.

- diversidad de APIs de conexión a los proveedores de infraestructura en la nube que dificulta la gestión de diferentes infraestructuras en la nube.
- falta de capacidad de integración entre diferentes entornos de nube.
- transferencia de aplicaciones entre nubes causa complejidad y errores irrazonables.
- despliegue sobre múltiples infraestructuras los cuales exigen demasiados recursos humanos para el despliegue y la preparación.

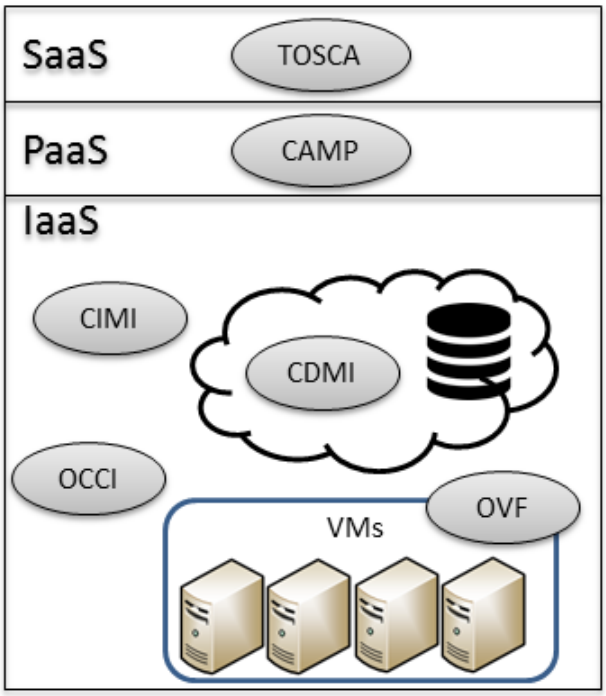

Figura 13: La figura muestra una clasificación de los estándares cloud analizados en este capítulo. La figura clasifica los estándares dependiendo de su ámbito de funcionamiento (IaaS, PaaS o SaaS).

Estos problemas pueden ser abordados desde diferentes perspectivas a través de los estándares analizados en este capítulo (ver Figura 13). Cada estándar tiene como objetivo conseguir algún aspecto clave en la gestión de la infraestructura en la nube. En función del análisis realizado se han identificado tres aspectos de interés por los organismos de estandarización (Figura 14):

1. Automatización en la gestión y despliegue de aplicaciones. El objetivo es proporcionar un lenguaje para expresar cómo implementar y

administrar automáticamente aplicaciones complejas en la nube. Esto se logra a través de la definición de una topología abstracta de aplicaciones complejas donde se describan su implementación y administración.

2. Portabilidad y gestión de aplicaciones a través de la IaC. Este objetivo trata de proporcionar una forma estandarizada de describir la topología de las aplicaciones. También aborda la portabilidad de la administración confiando en la portabilidad de los lenguajes de flujo de trabajo utilizados para describir los planes de implementación y administración.

3. Interoperabilidad y reutilización de componentes. Donde el objetivo es describir los componentes de aplicaciones complejas en la nube de una manera que permita la interoperabilidad y la reutilización de los componentes.

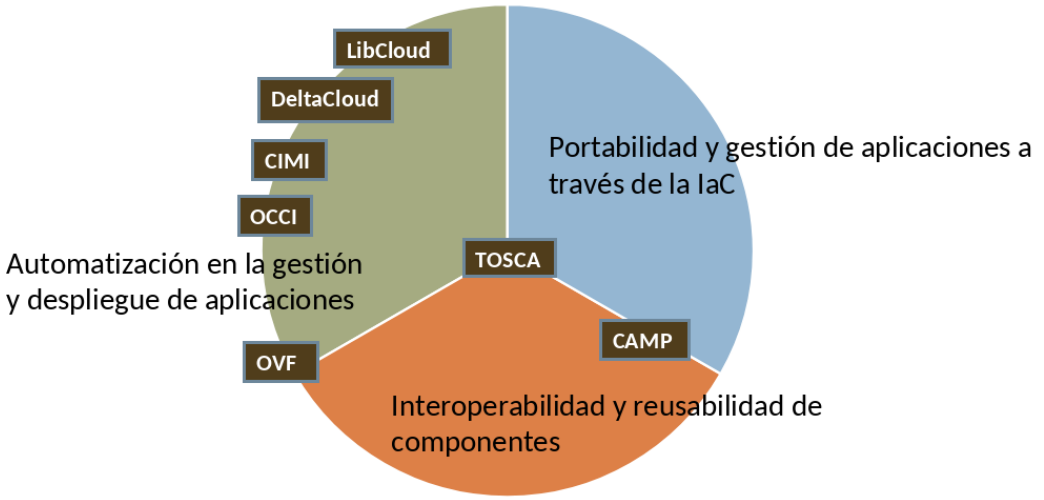

Figura 14: Comparación del estándar TOSCA frente a otros estándares. La figura muestra como existen diferentes soluciones para proporcionar funcionalidades de gestión en la nube y como TOSCA integra todas estas funcionalidades en un mismo estándar.

TOSCA tiene un enfoque de muy alto nivel para la nube. Debido a que TOSCA no incluye artefactos de implementación predefinidos para recursos de infraestructura, los proveedores de productos deben proporcionarlos ellos mismos o hacer uso de artefactos provenientes de terceras partes. Para resolver este problema se puede hacer uso de OCCI o CIMI para gestionar los recursos de infraestructura e incluso CAMP para realizar tareas avanzadas de configuraciones relacionadas con la capa de plataforma. APIs como

	OpenNebula	**OpenVIM**	**OpenStack**
CIMI			✓
OCCI	✓		✓
TOSCA	✓	✓	✓
CAMP			
CDMI	✓		
OVF	✓	✓	✓
LibCloud	✓		✓
DeltaCloud	✓		✓

Tabla 4: Estándares cloud implementados sobre proyectos de IaaS.

libcloud o Deltacloud podrían aportar valor para el caso de implementaciones portátiles y de conectores de gestión de recursos en la nube con bajos requisitos de funcionamiento.

Los estándares analizados no sólo son estándares que cualquier desarrollador puede utilizar para hacer sus aplicaciones más accesibles, sino que estos estándares han sido integrados en proyectos que proporcionan herramientas completas para la gestión de la IaaC. Algunos de los proyectos más importantes con licencia abierta son los que se han comentando en la sección 2.4.5. En la tabla 4 se muestran la relación de estos proyectos con cada uno de los estándares que integran para facilitar la interoperabilidad con otros sistemas.

Todos estos estándares ofrecen funcionalidades y capacidades a las infraestructuras en la nube. Sin embargo, todas estas capacidades no son suficientes para cubrir las necesidades de las arquitecturas de servicios de la siguiente generación. En la tabla 5 se muestran aquellos aspectos cubiertos por estos estándares.

La tabla 6 muestra una comparación que contempla las diferentes tecnologías de interoperabilidad descritas en esta sección. Una de las características más importantes al realizar el estudio de las tecnologías es que cada una de ellas utiliza su propio DSL y además no todos los DSLs tienen el mismo grado de desarrollo y alcance. Por lo tanto, los recursos en la nube descritos con estos estándares podrían no ser interoperables debido a su esquema y vocabulario heterogéneos. En un entorno colaborativo donde las organizaciones están dispuestas a compartir sus recursos en la nube, traducir descripciones de recursos en la nube es, por lo tanto, una tarea manual y tediosa. Esto a su vez crea silos de nube y recursos en la nube no reutilizables.

	CIMI	**OCCI**	**TOSCA**	**CAMP**	**CDMI**	**OVF**	**LibCloud**	**DeltaCloud**
R1			✓			✓		
R2	✓	✓	✓	✓		✓	✓	✓
R3								
R4						✓		
R5			✓			✓		
R6			✓			✓		
R7						✓		
R8			✓		✓	✓		
R9	✓	✓					✓	✓
R10			✓					
R11			✓					
R12			✓	✓				
R13			✓					
R14								
R15								
R16								
R17			✓					
R18						✓		
R19								

Tabla 5: Requisitos de las futuras arquitecturas de servicios satisfechos por las infraestructuras en la nube.

	CIMI	**OCCI**	**TOSCA**	**CAMP**	**CDMI**	**OVF**	**LibCloud**	**DeltaCloud**
DSL	Básico	Medio	Avanzado	Básico	No	Básico	No	No
API	Sí	Sí	No	Sí	Sí	No	Sí	Sí
Actualizada en:	2016	2016	2020	2012	2020	2015	2021	2015
Gestión recursos de infra-estructura	Sí	Sí	Sí	No	No	Sí	Sí	Sí
Gestión a nivel plataforma	No	Sí	Sí	Sí	No	No	No	Sí
Gestión aplicaciones	No	No	Sí	Sí	No	No	No	No
Orquestacion de servicios	No	No	Sí	No	No	No	No	No
Intercambio Información	No	No	No	No	Sí	No	No	No

Tabla 6: Comparativa entre las tecnologías para la gestión de la nube.

Como se puede observar en la tabla 6, TOSCA es la especificación que ofrece más funcionalidades además de ser la única que esta siendo mantenida en los últimos años. Durante la investigación se descubrió que era imposible llevar a cabo una configuración completa entre los recursos de diferentes proveedores de nube, ya que incluso los parámetros básicos presentados por el estándar TOSCA no son totalmente compatibles con los proveedores de nube más avanzados. Sin embargo, se supone que mientras la idea de unificar los recursos de la nube está evolucionando, los proveedores de nube pueden agregar los recursos que faltan. Además, el estándar TOSCA requiere clarificación y desarrollo. Sin embargo, TOSCA permite que todas las actualizaciones se agreguen rápidamente. Además, como solución alternativa se está planeando agregar compatibilidad con configuraciones de software dentro de máquinas virtuales para implementar el servicio mediante TOSCA y para la administración de estado de las infraestructuras existentes.
Para resolver estos problemas, esta tesis proporciona mecanismos en la que, partiendo de la información de la infraestructura, los mecanismos propuestos pueden adaptar esta información de manera sencilla, a las configuraciones requeridas basadas en alguno de los estándares descritos en el apartado 2.4.5. Además de las configuraciones adicionales necesarias a través del uso de las herramientas que se van a analizar en la secciones posteriores.

3

IAC COMO SOPORTE AL DESPLIEGUE DE SOFTWARE EN LA NUBE

En el capítulo 2 se ha presentado el estado del arte de relacionado con las arquitecturas de servicios, las tecnologías de virtualización y mecanismos para gestionar recursos y despliegues de servicios sobre entornos basados en la nube. En este capítulo se analiza la tecnología y herramientas de la Infraestructura como Código (IaC) como una complemento a las soluciones mostradas en 2.4 cuyo objetivo es proporcionar mecanismos de gestión para entornos en la nube. Este capítulo comienza introduciendo a la IaC a través de la sección 3.0.1, en 3.1 se presenta una clasificación de los diferentes tipos de herramientas que se pueden encontrar en el ámbito de la IaC. A continuación en 3.2 se describe las herramientas destinadas para gestionar la configuración, instalación y eliminación de software de manera automática. En 3.3 se muestra herramientas que van más allá del alcance de las herramientas mostradas en 3.2 cuyo objetivo no sólo es configurar software sobre una máquina sino que permiten desplegar componentes y servicios software sobre infraestructuras en la nube y configurar las posibles interacciones requeridas entre sí. Finalmente, en 3.4 se presentan los proyectos y soluciones más relevantes que integran varias herramientas y mecanismos de los mostrados en 3.2 y 3.3.

3.0.1 *Introducción*

Dentro del ámbito de la industria de servicios de distribución de contenidos multimedia, los proveedores de servicios deben ser capaces de adaptar sus servicios y sistemas respecto a las necesidades de cada momento. Esto no sólo se refiere a que debe adaptarse a la oferta del mercado de servicios si no que debe ser capaz de disponer de los mecanismos adecuados que le permita mantener y desplegar software rápidamente incorporando nuevos cambios. En esta línea, el mercado actual de las tecnologías de la información (TI) investiga continuamente en la "necesidad de velocidad", es decir: velocidad en el despliegue, ciclos de lanzamiento más rápidos, velocidad en la recuperación y más. Esta necesidad se refleja en el desarrollo de la metodología DevOps [112].

DevOps es una familia de técnicas que acortan el ciclo de desarrollo de software y también combinan las actividades de desarrollo de software con operaciones de TI [113], [114].

Como parte de la familia de prácticas de DevOps, el concepto de infraestructura como código (IaC) [115] promueve la administración de los conocimientos y de la experiencia adquirida previamente a través de scripts reutilizables, compuestos de código de infraestructura en lugar de realizar tareas manualmente. Estas tareas manuales realizadas por los administradores de sistemas suelen ser lentas, laboriosas y, a menudo, incluso propensa a errores. La aplicación de la tecnología IaC en la administración de sistemas soluciona estos problemas mejorando el mantenimiento y los tiempos de despliegue de los servicios sobre diversas infraestructuras incluso entornos de computación en la nube de múltiples proveedores.

En los últimos años, la evolución y uso de la IaC ha impulsado el desarrollo de muchos tipos de lenguajes y frameworks orientados al despliegue de software, donde cada uno de ellos se ocupa de un aspecto específico de la gestión de infraestructura. Como por ejemplo: herramientas capaces de aprovisionar y orquestar máquinas virtuales (Cloudify [116], Terraform [117], etc.), herramientas que aprovisionan y orquestan contenedores (tales como contenedores desarrollados para: Docker Swarm [118], Kubernetes [119]), herramientas de gestión de imágenes de máquina (Packer [120]) o incluso herramientas de gestión de la configuración software(Chef [83], Ansible [32], Puppet [31], etc.).

Esta situación hace que el panorama de los lenguajes y herramientas de la IaC sea complejo por la heterogeneidad tecnológica y por el gran número de soluciones disponibles. Esto refleja el gran interés que ha suscitado la IaC pero complica la comprensión y adopción de esta nueva tecnología. En el contexto actual, arrojar luz sobre la adopción, las cuestiones y los desafíos actuales de la IaC es fundamental para llevarla a la madurez y facilitar su desarrollo.

A pesar de estar en una fase de desarrollo temprana de la IaC, hay dos vertientes bien diferenciadas: la gestión de la configuración y la orquestación. Cada una de estás vertientes involucra aspectos diferentes de la IaC pero a la vez se complementan. Se describen estas dos vertientes en los apartados 3.2 y 3.3.

3.1 ECOSISTEMA DE TECNOLOGÍAS DE LA IAC

La infraestructura como código es compleja, aborda muchos ámbitos de configuración debido a que los sistemas de TI son complejos y están compuestos por multitud de tecnologías software y recursos hardware que se conectan y distribuyen sobre diferentes niveles tecnológicos de los sistemas (más información en el apartado 2.4.2). Estos niveles están relacionados unos con otros proporcionando herramientas y soporte al nivel superior, el cual no podría funcionar sin el nivel inferior.

A pesar de la dependencia entre estos niveles existen herramientas especificas para cada nivel que facilitan la configuración por código del software de cada uno de los niveles. Sin embargo, parte de estas herramientas están diseñadas para trabajar específicamente sobre niveles concretos debido a que están diseñadas para trabajar con los aspectos técnicos y únicos de cada nivel. Para este trabajo hemos analizado las diferencias y aspectos técnicos de cada nivel y se han identificado 4 categorías en las que se pueden clasificar las diferentes herramientas de IaC en función de sus capacidades técnicas. A continuación se describe cada uno de los tipos de herramientas:

1. Herramientas para la Gestión del despliegue de configuración y cambios post instalaciones. Estás herramientas están diseñadas principalmente para gestionar cambios de configuración sobre aplicaciones o servicios ya instalados y funcionando sobre un entorno de ejecución.

2. Herramientas para la Gestión del despliegue de aplicaciones y configuraciones. Estas herramientas se centran en la instalación y despliegue de software sobre máquinas ya aprovisionadas con una imagen del sistema operativo a falta de configurar los entornos de ejecución y los servicios o aplicaciones que se ejecutarán sobre ellos. Estas herramientas están preparadas para configurar plataformas, frameworks, servidores, entornos de ejecución, etc.

3. Herramientas de aprovisionamiento de infraestructura. Este tipo de herramientas se encargan de aprovisionar y configurar los recursos software para definir instancias de máquinas y las redes de datos que las provee de conexión. Una vez configuradas y ejecutadas las instancias correspondientes, las aplicaciones y servicios correspondientes pueden ser instalados sobre ellas.

4. Herramientas para la Plantillas infraestructuras. Estas herramientas se encargan de definir la estructura de los sistemas y las imágenes de las instancias de las máquinas que serán instaladas sobre un infraestructura. Con un cierto grado de configuración. Estas herramientas son capaces de configurar sistemas a partir de dichas plantillas para que posteriormente sean desplegadas sobre el proveedor de infraestructura.

La Figura 15 muestra las categorías comentadas por niveles además de las herramientas más populares que podemos encontrar en la industria actual. Como se puede observar en dicha figura, algunas herramientas proporcionan mecanismos de configuración pertenecientes a otras categorías. Es importante indicar que algunas herramientas están especialmente diseñadas para operar en alguna de las categorías y no por proporcionar

mecanismos de varias categorías va a ser mejor que otra herramienta que sólo proporcione mecanismos en una categoría. Si no que es posible que la herramienta diseñada para trabajar sobre una única categoría nada más disponga de un soporte mayor y más especifico que otras herramientas con un alcance más amplio.

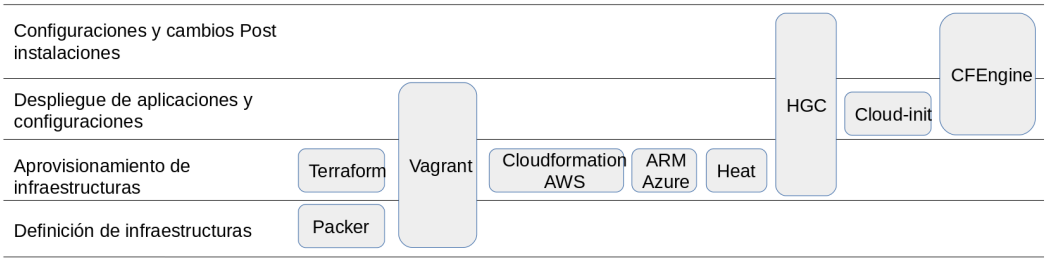

Figura 15: Ecosistema de herramientas de configuración de infraestructuras. El bloque Herramientas de Gestión de Configuración (HGC) incluye las herramientas: Ansible, Chef, SaltStack y Puppet.

3.2 GESTIÓN DE LA CONFIGURACIÓN

Como se ha visto en la sección 3.1, existen varios tipos de herramientas para gestionar diferentes aspectos software de los diferentes niveles de infraestructura de los sistemas. Los diferentes componentes software distribuidos sobre diferentes niveles forman un ecosistema digital el cual puede ser fácilmente gestionado si se cuenta con las herramientas adecuadas. La IaC se apoya en la gestión de la configuración para gestionar el despliegue de software sobre sistemas TI. La gestión de la configuración tiene como objetivo automatizar, supervisar, diseñar y administrar los procesos de configuración manuales. Su uso permite a las empresas escalar más fácilmente sin contratar personal adicional para la administración de sistemas. El alcance del proceso de la gestión de configuración implica las siguientes actividades:

- Identificación de elementos de configuración.
- Controlar los cambios en los elementos de configuración.
- Creación o especificación para crear productos de trabajo desde el sistema de gestión de configuración.
- Mantener la integridad de la línea de base.
- Proporcionar datos precisos de estado y configuración a los desarrollos, usuarios finales y clientes.

3.2.1 *Herramientas de la gestión de la configuración*

Las herramientas de IaC son de vital importancia en aquellas compañías que tienen que gestionar grandes sistemas de TI debido a que configurar un único servidor con el software necesario es una tarea razonablemente sencilla. Sin embargo, en un escenario en el que numerosos servidores necesitan tener el mismo (o similar) software instalado y requieran algo de configuración podría disparar la complejidad de gestionarlos debido a que el proceso, requeriría muchas horas de trabajo por parte de los administradores de sistemas en el caso de que no usasen algún tipo de automatización.

Con el objetivo de resolver este problema se desarrollaron nuevas herramientas de gestión de configuración para abordar la necesidad de gestionar nuevos servidores con configuraciones y actualizaciones predefinidas, permitiendo procesos de automatización más fluidos y manejables. Estas herramientas de gestión permiten mantener los equipos sincronizados sobre infraestructuras independientemente que sean centro de datos o sistemas de computación en la nube.

Las herramientas de gestión de configuración funcionan especificando la configuración de un sistema a través de estados. Por ejemplo, se especificarían estados para configurar la infraestructura básica del sistema (almacenamiento, redes). Cada paso de la configuración a menudo depende de otros, por ejemplo, para crear una aplicación se debe instalar un compilador y luego compilar el código fuente antes de realizar su instalación; o para ejecutar un servicio orientado a la Web, es posible que se requiera un servidor web y un servidor de bases de datos. Para controlar estas dependencias, normalmente se expresa cada regla de la configuración junto con sus condiciones previas y posteriores. Por ejemplo, la siguiente regla de Puppet expresa que para ejecutar el servidor de correo Postfix como un servicio, el sistema debe haber instalado el paquete y el archivo de configuración correspondiente. Al final, el servicio estará en estado de ejecución:

Cuadro 2: Ejemplo de despliegue con Puppet que expresa las dependencias y configuración de un servidor de correo Postfix

```
service {'postfix':
require => [
Package['postfix'],
File['/etc/postfix/main.cf],
]
enable =>'true',
ensure => 'running',
}
```

Las especificaciones declarativas de alto nivel, como las anteriores, permiten utilizar técnicas de ingeniería de software para organizar grandes configuraciones complejas de diversos hosts. Actualmente existe un gran número de herramientas de gestión de la configuración. De hecho, en la literatura existen más herramientas para la gestión de la configuración (HGC), pero no se han tomado en cuenta en este trabajo debido a que el objetivo es conseguir tener configuraciones disponibles para trabajar con las herramientas más populares del mercado. Esto es importante ya que permite suavizar la curva de aprendizaje correspondiente a la gestión de los sistemas a la hora de incorporar personal nuevo en la administración de los sistemas. Para este trabajo se ha tomado en consideración aquellas herramientas más relevantes y que proporcionan un mayor número de funcionalidades permitiendo abarcar múltiples escenarios de aplicación. En la Figura 16 se puede ver un gráfico de la popularidad de las principales herramientas que hay disponibles en la actualidad, a continuación se muestran las principales herramientas de configuración:

- **Ansible** [32]: Ansible es una popular plataforma de infraestructura como código que se utiliza para automatizar la implementación y configuración de infraestructuras. Aunque está pensado principalmente como una herramienta para configurar rápidamente un grupo de máquinas remotas, también se puede utilizar para configurar una máquina local.
- **Puppet** [31]: Puppet es otra herramienta de gestión de configuración orientada a servidores la cual ofrece una forma automatizada de analizar, instalar y gestionar software.
- **Chef** [83]: Chef es una herramienta de gestión de la configuración, que se utiliza para la configuración de máquinas virtuales en la nube y servidores. Además, es capaz de configurar los archivos y el software en una máquina.
- **Salt**[121]: SaltStack es una herramienta de gestión de configuración de código abierto basada en Python que ayuda a la ejecución remota de tareas, la automatización de eventos y gestión de la configuración.
- **CFEngine**[122]: Es una solución que ayuda a los usuarios a garantizar el cumplimiento a través de la configuración de directivas, automatizar la instalación actualizaciones sobre infraestructuras.
- **Cloud-init**[1]: Cloud-init es una herramienta para personalizar instancias en la nube. La herramienta suele ser utilizada para personalizar imágenes las cuales comienzan siendo clones de otras imágenes

1 https://cloud-init.io/

y necesitan personalizaron o ser inicializadas con una configuración concreta. Cloud-init es la herramienta que aplica esta configuración a las instancias de forma automática. Cloud-init fue desarrollado inicialmente por Canonical para las imágenes cloud de Ubuntu usadas por AWS. Desde entonces, la aplicación ha evolucionado y puede ser usada en otras muchas distribuciones y en otros entornos cloud (y no cloud).

3.2.2 *Características de las herramientas de gestión de la configuración*

Las HGC se caracterizan por propiedades y características que las diferencia de otras herramientas satisfaciendo necesidades tecnológicas que no fueron tenidas en cuenta por otras herramientas. Esto hace que unas HGC sean más adecuadas para unos propósitos que otros. A continuación se describen algunas de las propiedades más relevantes para describir las herramientas de configuración de software. Además más adelante estas características servirán para evaluar y comparar las diferentes herramientas analizadas en este trabajo.

- **Infraestructura**: Esta característica indica si una infraestructura es mutable o inmutable. Las infraestructuras mutables hacen referencia a aquellos entornos que cambian la configuración después de que ha sido previamente instalada debido a que los administradores siempre están haciendo ajustes o añadiendo código. Las herramientas evolucionaron para administrar esta complejidad y poner orden en la configuración y actualización de decenas a miles de servidores. Una infraestructura inmutable es aquella en la que los servidores nunca se modifican después de instanciarse. Si es necesario actualizar o cambiar algo, los servidores nuevos se destruyen y se crean de nuevo a partir de una plantilla común con los cambios deseados.

  Como se puede ver en la tabla 7 el valor de esta propiedad es siempre mutable ya que es una propiedad intrínsecamente ligada a las herramientas de gestión de la configuración, sin embargo esta propiedad es totalmente diferente en la tabla 9 ya que con las herramientas de orquestación, la inmutabilidad se aplica fácilmente a los servidores, ya que normalmente tienen compatibilidad integrada para administrar el ciclo de vida de un recurso, desde la creación hasta la finalización del uso y borrado de los servicios implicados en una cadena de servicios.

- **Tipo de Arquitectura**: Hay dos tipos de arquitecturas en las herramientas analizadas, 'Cliente-servidor' y 'Solo cliente'. La arquitectura Cliente-servidor implica extraer la configuración de un servidor

central a los nodos esclavos sin ningún comando. Mientras que, en una arquitectura "Solo cliente", las configuraciones en el servidor se envían a los nodos con comandos específicos.

Chef, Puppet y SaltStack necesitan que en los nodos exista un agente software para instalar el software indicado por el servidor central. Éste agente recibe los comandos correspondientes desde el servidor central y el agente aplica los cambios indicados sobre la máquina local.

El agente normalmente se ejecuta en segundo plano en cada máquina y es responsable de la instalación de las últimas actualizaciones de gestión de la configuración. En el caso de Ansible, no se requiere de la instalación de ningún agente adicional, de hecho, las máquinas son gestionadas remotamente a través de una conexión SSH que da acceso a un terminal donde se ejecutan los comandos necesarios para materializar la configuración elegida. Y todo ello desde otra terminal remota tal como si lo hiciese un administrador.

De forma predeterminada, Chef, Puppet y SaltStack requieren que ejecute un servidor maestro para almacenar el estado de la infraestructura y distribuir actualizaciones. Cada vez que se desea actualizar algo en la infraestructura, se utiliza un cliente para enviar nuevos comandos al servidor maestro y el servidor maestro envía las actualizaciones a todos los demás servidores, o esos servidores extraen las actualizaciones más recientes del servidor maestro de forma regular.

Utilizar servidores maestro ofrece algunas ventajas. En primer lugar, es un único lugar central donde puede ver y administrar el estado de la infraestructura. En segundo lugar, algunos servidores maestros pueden ejecutarse continuamente en segundo plano y aplicar la configuración. De esta manera, si alguien realiza un cambio manual en un servidor, el servidor maestro puede revertir ese cambio para evitar la deriva de configuración. Sin embargo, usar un servidor maestro tiene algunos inconvenientes:

- Necesidad de infraestructura adicional. Es necesario implementar un servidor adicional, o incluso un clúster de servidores adicionales (para alta disponibilidad y escalabilidad), solo para ejecutar el servidor maestro.
- Gestión de tareas de mantenimiento. Se debe mantener, actualizar, realizar una copia de seguridad, supervisar y escalar los servidores maestros.
- Seguridad más compleja y crítica. Se debe proporcionar una manera para que el cliente se comunique con los servidores maes-

tros y una manera para que los servidores maestros se comuniquen con todos los demás servidores, lo que normalmente significa abrir puertos adicionales y configurar sistemas de autenticación adicionales, aumentando la probabilidad de exponer vulnerabilidades a los atacantes.

Ansible no tiene servidor maestro de forma predeterminada si no que funciona conectándose directamente a cada servidor a través de SSH, por lo que de nuevo, no tiene que ejecutar ninguna infraestructura adicional ni administrar mecanismos de autenticación adicionales (es decir, simplemente use las claves SSH).

- **Gestión ciclo de vida, Aprovisionamiento VM, Networking, Gestión almacenamiento, Gestión backups y Orquestación de servicios** son características más orientadas a herramientas de orquestación ya que algunas de ellas permiten gestionar el ciclo de vida de componentes software como VNFs, gestionar VMs y además realizar las conexiones de red virtuales requeridas, gestionar almacenamientos externos (tales como copias de seguridad) e incluso realizar la orquestación de servicios creando complejas aplicaciones.

Como se puede ver en la tabla 7, existen varias herramientas para dar soporte a la gestión de configuración, sin embargo, elegir las herramientas de administración adecuadas para gestionar una infraestructura es una decisión crítica. Estas herramientas son capaces de realizar las funciones necesarias para realizar la mayor parte de las tareas de gestión requeridas. Sin embargo, cada una ofrece diferentes metodologías, roles de usuarios y formas de trabajar. Por ejemplo, Ansible tiene sus ventajas, ya que se dirige más hacia un rol orientado a administradores de sistemas con respecto a su estructura y paradigma. Además, es una de las opciones más fáciles de esta lista para configurar y empezar a usar de inmediato. Salt es similar en este aspecto, pero es diferente de Puppet y Chef, ya que están más orientadas a los desarrolladores. Además, Ansible y Salt están orientadas a trabajar con infraestructuras más uniformes. En el caso de Salt, es una de las herramientas más fiables y robustas, sin embargo presenta una cierta dificultad para los usuarios principiantes A pesar de esto, Salt proporciona opciones avanzadas de escalabilidad y un entorno de trabajo accesible desde la perspectiva del administrador. También hay varios complementos disponibles para mejorar y ampliar las capacidades de la interfaz de usuario haciéndolo más fácil de manejar.

Puppet y Chef son más fáciles de utilizar. En el caso de Puppet es necesario disponer de conocimiento en Ruby para poder aprovechar al máximo su amplia gama de características, estructura y escalabilidad. La configuración de Puppet puede llegar a ser muy granular y a veces complicada donde se

deberá aprender nuevos procedimientos de codificación para suavizar la curva de aprendizaje cuando se utiliza su programación DSL (Lenguaje Específico de Dominio). Pero es la apuesta más segura si está buscando un entorno de software no heterogéneo. Chef es una herramienta muy fiable y sencilla de manejar si el usuario dispone de experiencia con el desarrollo ya que requiere conocimientos de lenguajes de programación. De hecho está enfocada principalmente para facilitar el acceso a usuarios con habilidades de programador.

Todas estas herramientas desempeñan un papel específico al configurar la infraestructura y los estados deseados. Su utilización dependerá enteramente de sus necesidades de configuración, soporte personal y los conocimientos necesarios para implementarlas.

3.2.3 *Discusión*

En este apartado se va a analizar el estado de las diferentes herramientas analizadas en el apartado 3.2 para ver el grado de adopción, de uso e implantación en la industria. Esto es importante ya que ver cuáles son las herramientas más utilizadas frente a otras y su popularidad ayudan a entender el estado del mercado en función de las necesidades y preferencias de los usuarios. La Figura 16 muestra el gráfico con el análisis de las

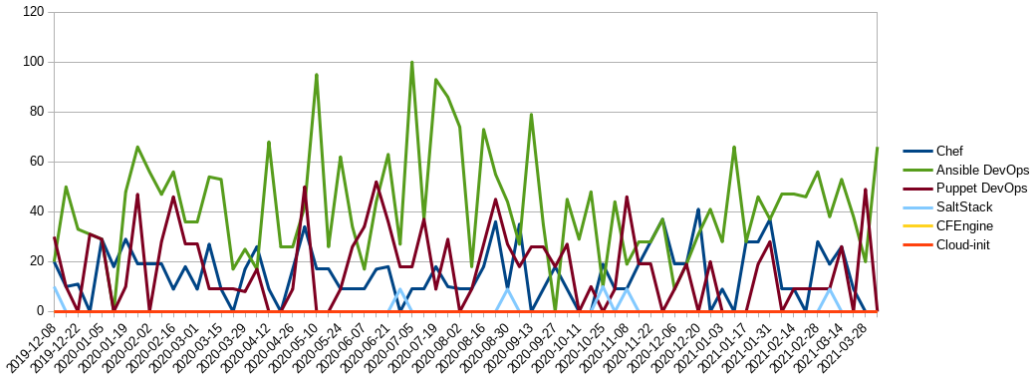

Figura 16: Evolución del uso de las herramientas de gestión de la configuración en los últimos años. El gráfico muestra claramente con Ansible ha sido la opción preferida por los administradores de sistemas durante los últimos años.

herramientas orientadas a la gestión de la configuración. La gráfica ha sido elaborada a partir de los datos obtenidos con la herramienta Google Trends en la que se ha introducido una consulta por cada herramienta analizada (Ansible, Puppet, Chef, CFEngine, Cloud-init y SaltStack) en el marco de tiempo comprendido entre octubre de 2018 hasta Marzo de 2021. Google

	Chef	**Puppet**	**Ansible**	**SaltStack**
Infraestructura	Mutable	Mutable	Mutable	Mutable
Lenguaje	Procedimiento	Declarativo	Procedimiento	Declarativo
Arquitectura:	Cliente-servidor	Cliente-servidor	Sólo cliente	Sólo cliente
Gestión ciclo de vida	No	No	No	No
Aprovisionamiento VM	No	No	No	No
Networking	No	No	No	No
Gestión almacenamiento	No	No	No	No
Gestión Empaquetado	Sí	Sí	Sí	Sí
Templating	Sí	Sí	Sí	Sí
Provisionamiento de servicios	Sí	Sí	Sí	Sí
Gestión backups	No	No	No	No
Orquestación de servicios	No	No	No	No
Orientado a:	Múltiples máquinas	Múltiples máquinas	Múltiples máquinas	Múltiples máquinas

Tabla 7: Comparación de herramientas de gestión de configuración software. La tabla muestra algunas de las características más relevantes de cada una de las tablas.

Trends[2] muestra la popularidad de una herramienta en función de las búsquedas realizadas a través del buscador Google. Como se puede observar, Ansible es mayoritariamente la principal herramienta utilizada en la escala de tiempo indicado en el gráfico.

Le sigue a cierta distancia las herramientas Chef y Puppet. Finalmente, la herramienta SaltStack, CFEngine y Cloud-init presentan una popularidad muy baja en comparación a la demás herramientas. Tomando en cuenta los perfiles de usuarios de cada herramienta, se puede ver que la mayor parte de los usuarios de las herramientas de configuración está formado por usuarios con perfiles profesionales orientados a administradores de sistemas debido a que Ansible está orientada a usuarios con perfil de admi-

2 https://trends.google.es/trends/

Requisito	Chef	Puppet	Ansible	Salt
R1				
R2				
R3	✓	✓	✓	✓
R4				
R5				
R6				
R7				
R8	✓	✓	✓	✓
R9				
R10	✓	✓	✓	✓
R11				
R12				
R13				
R14	✓	✓	✓	✓
R15	✓	✓	✓	
R16	✓	✓	✓	✓
R17				
R18	✓	✓	✓	✓
R19	✓	✓	✓	✓

Tabla 8: Requisitos cumplidos por cada una de las herramientas de gestión de la configuración analizadas en el trabajo.

nistrador. A través de la tabla 7 se puede realizar un análisis comparativos de cada una de las herramientas presentadas. Como se puede observar, Chef y Ansible usan un lenguaje de estilo de procedimiento donde se escribe código que especifica, paso a paso, cómo lograr el estado final deseado. La responsabilidad para determinar el proceso de implementación óptimo está en el usuario. SaltStack y Puppet utilizan un estilo declarativo donde se escribe código que especifica el estado final deseado. Luego la propia herramienta IaC determina cómo lograr ese estado de la manera más eficiente posible. Los lenguajes de procedimiento son más familiares para los administradores del sistema ya que suelen estar familiarizados con el scripting. Sin embargo, las herramientas declarativas están más familiarizadas con los usuarios desarrolladores de código. Como se puede ver en la tabla 7 cada una de las herramientas son capaces de proveer un conjunto de funcionalidades que las hace idóneas para distintas soluciones dependien-

do de las necesidades a cubrir. Para evaluar el grado de cumplimiento de cada una de estas herramientas con los requisitos mostrados en la tabla 1 se ha desarrollado la tabla mostrada en 8 en la que se verifica el grado de cumplimiento de cada una de las herramientas analizadas en este capítulo. Como se puede ver en la tabla 1, la mayoría de las herramientas proporcionan, como es lógico, aquellas funcionalidades propias de las herramientas de gestión de la configuración tales como la gestión eficiente del software (R10, R14, R16 y R19) sobre múltiples plataformas (R3 y R8). Además permiten proporcionar gestión de la seguridad e interoperabilidad con otros sistemas (R8 y R18) y todo ello a través de mecanismos de configuración accesibles al personal técnico mediante el uso de lenguajes declarativos que facilitan la curva de aprendizaje junto a una gran documentación desarrollada durante años (R15).
Sin embargo, estas herramientas no satisfacen otros requisitos tales como La softwarización de cualquier funcionalidad (R7), la gestión de recursos de infraestructura (R9), mecanismos para la gestión del ciclo de vida de los servicios (R11), entre otros. Por lo que el uso de este tipo de herramientas requiere el estudio de otras herramientas o mecanismos que sean capaces de integrarse o usar las herramientas analizadas para ser capaz de satisfacer cada uno de los requisitos identificados en 7.

3.3 APROVISIONAMIENTO AUTOMATIZADO

Chef, Puppet, Ansible y SaltStack son herramientas de gestión de configuración (HGC), lo que significa que están diseñadas para instalar y administrar software en máquinas instanciadas. Las herramientas que se van a presentar en esta sección son herramientas de aprovisionamiento, lo que significa que están diseñadas para aprovisionar servidores así como el resto de su infraestructura (como equilibradores de carga, bases de datos, configuración de redes, etc.), dejando las tareas de configuración de más alto nivel a otras herramientas.

Estos dos tipos de herramientas no son mutuamente excluyentes, ya que la mayoría de las HGC pueden hacer cierto grado de aprovisionamiento y la mayoría de las herramientas de aprovisionamiento pueden hacer cierto grado de administración de la configuración. Sin embargo es preferible usar HGC cuando sea necesario gestión de la configuración y usar herramientas de aprovisionamiento cuando haya que hacer tareas de aprovisionamiento de recursos debido a que cada tipo de herramienta esta diseñada principalmente para trabajar en un ámbito concreto.

3.3.1 *Herramientas de aprovisionamiento*

En este apartado se recogen soluciones que permiten realizar tareas de aprovisionamiento y de orquestación capaces de realizar la gestión de máquinas virtuales, servicios y recursos sobre múltiples proveedores de infraestructuras. A continuación se describen las herramientas más relevantes:

Kubernetes

Kubernetes es una plataforma de orquestación de contenedores de código abierto diseñada para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores. De hecho, Kubernetes se ha establecido como el estándar de facto para la orquestación de contenedores y es el proyecto insignia de la Cloud Native Computing Foundation (CNCF), respaldado por grandes empresas. Kubernetes facilita la implementación y el funcionamiento de aplicaciones en una arquitectura de microservicios a través de una capa de abstracción que aplica sobre un grupo de hosts. Esta capa de abstracción permite tareas de administración tales como:

- Control del consumo de recursos por aplicación
- Distribución uniforme de la carga de las aplicaciones sobre una infraestructura de hosts.
- Balanceador de la carga de manera automática sobre las diferentes instancias de una aplicación.
- Monitorización del consumo de recursos para evitar que las aplicaciones consuman demasiados recursos.
- Mover una instancia de aplicación de un host a otro si se produce escasez de recursos en un host o si se cae.
- Gestión automática de recursos en caliente cuando se agrega o elimina un host al clúster.

El componente central de Kubernetes es el clúster. Un clúster se compone de muchas máquinas virtuales o físicas que tienen un rol especializado como un maestro o como un nodo. Cada nodo hospeda grupos de uno o más contenedores (que contienen las aplicaciones) y el nodo principal se comunica con los nodos sobre cuándo crear o destruir contenedores. Al mismo tiempo, indica a los nodos cómo volver a enrutar el tráfico en función de las nuevas configuraciones de contenedores.

El componente Kubernetes Master es el punto de acceso (o el plano de control) desde el que los administradores y otros usuarios interactúan con

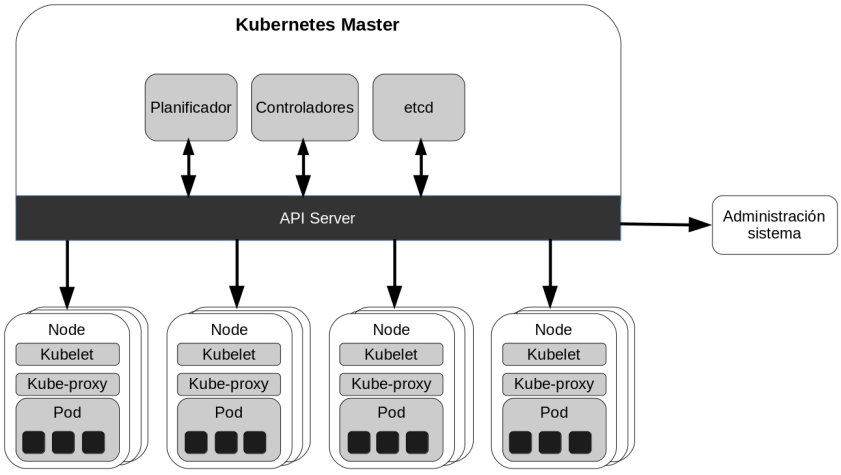

Figura 17: La figura muestra la arquitectura principal de Kubernetes donde se puede apreciar los principales componentes de la arquitectura y la relación que existen entre unos y otros.

el clúster para administrar la implementación de contenedores. Un clúster siempre tendrá al menos un maestro, pero podría llegar a tener más componentes Master en función del patrón de replicación del clúster. El componente Kubernetes Master almacena los datos de estado y configuración de todo el clúster en etcd el cual es un almacén de datos clave-valor persistente y distribuido. Cada nodo tiene acceso a ectd, y a través de él, los nodos mantienen las configuraciones de los contenedores que están ejecutando. Todos los nodos de un clúster de Kubernetes deben configurarse con un entorno de ejecución para contenedores el cual normalmente es Docker. El entorno de ejecución del contenedor se inicia y administra los contenedores a medida que Kubernetes los incluye en los nodos del clúster. Luego los contenedores arrancar las aplicaciones desplegadas en su interior (servidores web, bases de datos, servidores de API, etc.). Cada nodo de Kubernetes ejecuta un proceso de agente denominado kubelet que es responsable de administrar el estado del nodo. Esta administración consiste en iniciar, detener y actualizar los contenedores según las instrucciones del plano de control. El kubelet recopila información tal como el rendimiento, el estado del nodo, los pods y los contenedores que ejecuta. Toda esta información se comunica al plano de control para que pueda tomar decisiones. El proxy kube es un proxy de red que se ejecuta en los nodos del clúster. También puede funcionar como un equilibrador de carga para los servicios que se ejecutan en un nodo. Dentro del Kubernetes, los pods son la unidades básicas y están formados de uno o más contenedores los cuales se ubicarán en el equipo host y pueden compartir recursos. A cada pod se le asigna una dirección IP única dentro del clúster, lo que permite a la aplicación usar puertos sin conflictos.

Docker Swarm

Docker Swarm es un grupo de máquinas físicas o virtuales que ejecutan el entorno de ejecución de Docker y que se han configurado para unirse en un clúster. De esta manera Docker Swarm proporciona un alto nivel de disponibilidad a las aplicaciones basadas en docker. Docker Swarm es una herramienta de orquestación de contenedores, lo que significa que permite al usuario administrar varios contenedores implementados sobre varios host. Cada una de las máquinas que forman parte del clúster son llamadas nodos y están controladas por un servicio de administración el cual controla las actividades del clúster. Normalmente, dentro de Docker Swarm hay varios nodos de trabajo y al menos hay un nodo de administrador que es responsable de gestionar eficientemente los recursos de los nodos de trabajo y garantizar que el clúster funcione correctamente.

Docker Swarm permite que al crear un servicio para ser ejecutado en Swarm se pueda definir el estado óptimo del servicio (número de réplicas, puertos del servicio, recursos de red y almacenamiento, etc.). Durante la ejecución del servicio, Docker Swarm intentará mantener este estado deseado reiniciando o reasignando las tareas no disponibles y equilibrando la carga entre diferentes nodos.

CloudFormation

CloudFormation [123] es específico de Amazon Web Services (AWS) [124] y no puede utilizarse para aprovisionar infraestructura en ningún otro proveedor de nube. Sin embargo, la cobertura de AWS es extensa. Casi todos los servicios y recursos que puede crear en AWS se pueden modelar en CloudFormation.

AWS CloudFormation es un servicio que permite a los desarrolladores crear recursos desde AWS de forma controlada. Para esto, los recursos se escriben en archivos de texto utilizando el formato de notación de objetos JavaScript (JSON) o otro formato de lenguaje de marcado (YAML). Las plantillas requieren una sintaxis y estructura específicas que depende de los tipos de recursos que se van a crear y administrar. El uso de JSON en las plantillas es útil para el control de versiones, pero pueden llegar a ser rápidamente difíciles de leer y mantener, especialmente a medida que los entornos se hacen más grandes y complejos.

Azure Resource Manager

Azure Resource Manager (ARM) [125] es el servicio de implementación y administración para Azure. Proporciona una capa de administración que le permite crear, actualizar y eliminar recursos de la cuenta de Azure. ARM usa las características de administración, como el control de acceso, la auditoría y las etiquetas, para proteger y organizar los recursos después de la implementación.

Cuando un usuario envía una solicitud de cualquiera de las herramientas, las API o los SDK de Azure, Resource Manager recibe la solicitud, autentica y autoriza la solicitud. Resource Manager envía la solicitud al servicio de Azure, que lleva a cabo la acción solicitada. Debido a que todas las solicitudes se controlan mediante la misma API, los resultados y funcionalidades son coherentes en todas las distintas herramientas.

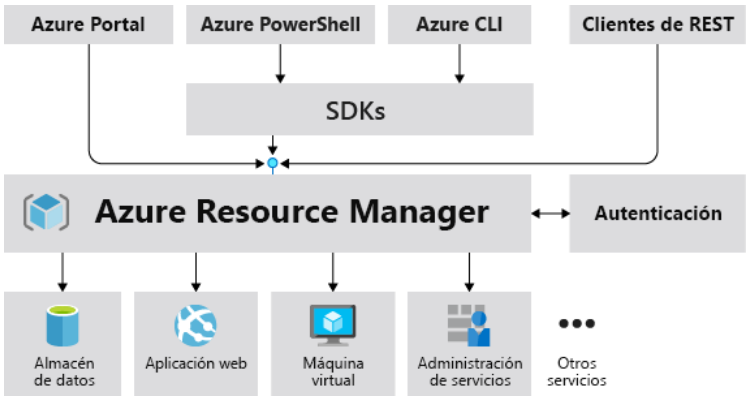

Figura 18: La figura muestra la arquitectura del ARM desde [125] y las relaciones con otros componente de la arquitectura de Azure.

En la imagen 18 se muestra el rol que Azure Resource Manager desempeña en el control de solicitudes de Azure. Azure Resource Manager permite:

- Administrar la infraestructura mediante plantillas declarativas en lugar de scripts.
- Implementar, administrar y supervisar todos los recursos de una solución grupalmente, en lugar de controlarlos individualmente.
- Volver a implementar la solución repetidamente a lo largo del ciclo de vida del desarrollo y tener la seguridad de que los recursos se implementan de forma coherente.
- Definir las dependencias entre recursos de modo que se implementen en el orden correcto.
- Aplicar control de acceso a todos los servicios.
- Aplicar etiquetas a los recursos para organizar de manera lógica todos los recursos de la suscripción.

OpenStack Heat

Heat [126] es un proyecto de OpenStack el cual proporciona orquestación permitiendo describir la infraestructura en la nube o componer una aplicación a partir de un archivo de texto escrito en un lenguaje propio llamado

Heat Orchestration Template (HOT) [109]. HOT es un archivo de plantilla, por lo general proporcionado por el administrador con la configuración deseada la cual procesa Heat y permite realizar la asignación de recursos e interactuar con otros componentes de OpenStack como Neutron [109] para gestionar los recursos de red; Nova para la creación de máquinas virtuales y Cinder para el almacenamiento en bloques [109].

Heat se compone de tres servicios principales en su infraestructura: HeatEngine, Heat API y CloudWatch. HeatEngine es el servicio principal que analiza los modelos y proporciona características que son los elementos básicos de una pila de capas software. HeatEngine, después de analizar un modelo, implementará y configurará recursos en la nube. Heat expone una API REST nativa y una API compatible con Amazon AWS Query a través del servicio Heat API. La Figura 19 presenta la arquitectura y las integraciones de los servicios internos de Heat [109].

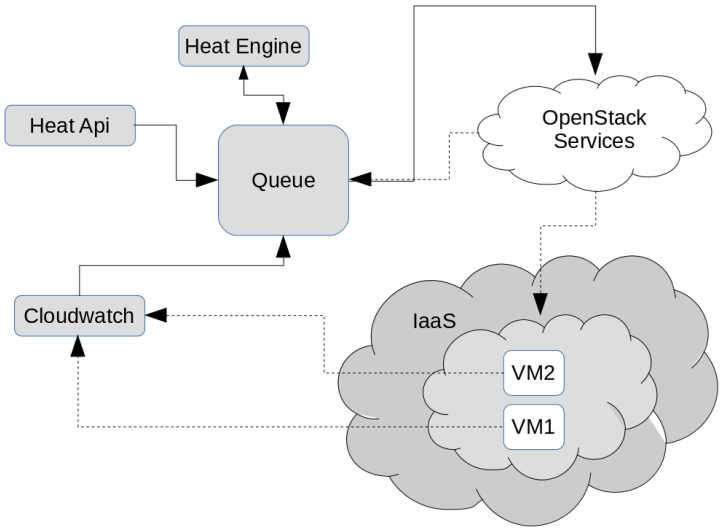

Figura 19: La figura muestra la arquitectura de OpenStack Heat y los servicios principales que implementa para proveer orquestación sobre una infraestructura IaaS.

Las plantillas HOT permiten trabajar con parámetros de entradas a través de una sección dedicada a ello, de esta manera, HOT permite a los usuarios personalizar una plantilla durante la implementación. Por ejemplo, esto permite proporcionar nombres de pares de claves personalizados o identificadores de imagen para su uso en una implementación.

Terraform

Terraform [127] es una herramienta de código abierto que permite la definición de infraestructura como código utilizando un lenguaje de programación declarativo simple [127]. Permite la implementación y administración de esta infraestructura en varios proveedores de nube pública (por ejem-

plo, AWS, Azure, Google Cloud y DigitalOcean), así como en plataformas de virtualización privadas (por ejemplo, OpenStack y VMWare) mediante algunos comandos [127]. Cuando se trata de servidores, Terraform ofrece varias maneras de configurarlos y conectarlos a las herramientas de administración de configuración existentes [117].

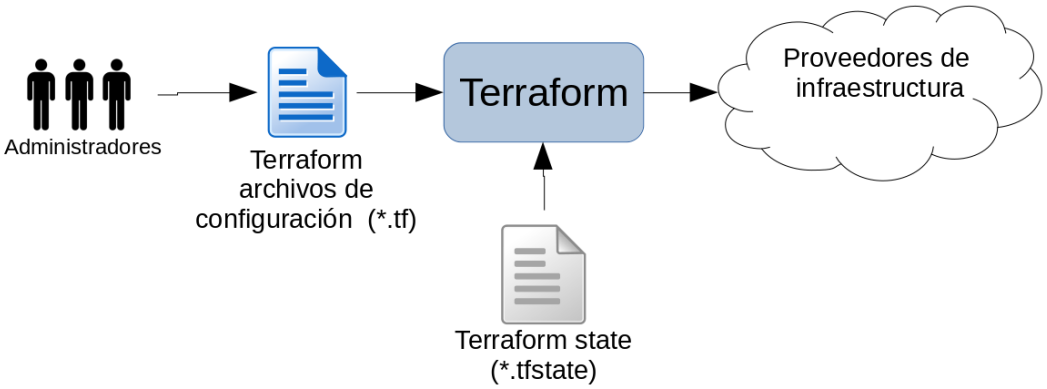

Figura 20: La figura resume los aspectos más importantes a tener en cuenta durante la gestión y despliegue de software sobre proveedores de infraestructura a través de la herramienta de gestión de configuración Terraform.

En la Figura 20 se muestra el papel de Terraform en un proceso de orquestación de nube, donde la herramienta lee un archivo de definición y aprovisiona los distintos recursos en el proveedor. En este flujo de trabajo es posible ver que desde un archivo de definición, Terraform interactúa con un proveedor de nube y realiza el aprovisionamiento de servicios de acuerdo con la configuración solicitada. El servicio puede ser una máquina virtual, una base de datos, un servicio de almacenamiento o lo que admita el proveedor. Terraform realiza un seguimiento del estado de la infraestructura creando y aplicando cambios cuando alguna característica necesita ser actualizada, agregada o eliminada [117]. También proporciona una manera de importar recursos existentes y dirigirse solo a recursos específicos. También es fácilmente extensible con plugins [117] lo que permite extender sus capacidades.

Juju

Juju [128] es una herramienta de código abierto para la automatización y la orquestación de servicios la cual permite modelar, configurar, administrar, mantener, implementar y escalar servicios en la nube de forma rápida y eficiente en nubes públicas, así como en contenedores MaaS (Metal as a Service) [3]), OpenStack y LXC.

3 Metal as a Service es una tecnología que permite que los servidores físicos se comporten como máquinas virtuales en la nube. En lugar de tener que administrar cada servidor

El modelado de servicios permite una implementación y administración de servicios rápidos y fáciles, conectando esos servicios y escalarlos rápidamente hacia arriba o hacia abajo, todo ello sin interrupciones en el entorno de nube.

Juju proporciona una configuración dinámica, que permite volver a configurar los servicios sobre la marcha, agregar, quitar o cambiar las relaciones entre servicios y escalar. Una vez instalado un cliente Juju, los entornos Juju se pueden arrancar en diferentes proveedores, que pueden ser servicios en la nube de varios proveedores. Juju es responsable de orquestar los servicios que se instalarán de acuerdo con una descripción de alto nivel o paquetes. Como se puede ver en la tabla 9, existen muchas funcionalidades comunes a cada una de las soluciones recogidas en la tabla pero a simple vista se puede observar que Juju y Heat son herramientas muy completas. Sin embargo, Heat está orientada principalmente a trabajar dentro del proyecto OpenStack y Juju es una herramienta más abierta que permite ser utilizada en diversos entornos con multitud de tecnologías además de realizar algunas tareas propias de herramientas de gestión de la configuración.

3.3.2 *Herramientas orientadas a orquestación VNFs*

El proceso de interconectar servicios unos con otros es el proceso de la 'Orquestación'. La orquestación nace a partir de los equipos de TI modernos ahora responsables de administrar de cientos a miles de aplicaciones y servidores, debido a que la administración manual simplemente no puede escalar las necesidades actuales. La orquestación es esencial para poder gestionar aplicaciones de alto rendimiento, escalado dinámico y sistemas en la nube, lo que alivia a los administradores de una carga muy laboriosa.

Por lo tanto, mientras que la automatización hace referencia a una sola tarea, la orquestación organiza las tareas para optimizar un flujo de trabajo. Por ejemplo, orquestar una aplicación significa no solo implementar una aplicación, sino también conectarla a la red para que pueda comunicarse con los usuarios y otras aplicaciones. En la nube, la orquestación suele ser clave para garantizar que las actividades de inicio automático para que el escalado automático se lleven a cabo en el orden correcto, con las reglas de seguridad y los permisos adecuados.

Un factor a tener en cuenta es que a medida que los entornos de nube siguen creciendo en complejidad, aumenta la necesidad de orquestación. El requisito más difícil de satisfacer es la coordinación de los componentes y la administración de las dependencias entre la configuración de un servicio y otro y esto se consigue con el proceso de orquestación.

---

individualmente, MAAS convierte la arquitectura física en un recurso elástico similar a la nube.

	Juju	**Terraform**	**Heat**	**CloudFormation**	**Azure Resource Manager**	**Kubernetes**	**Docker Swarm**
Tareas	Gest. Conf y Orquestación	Orquestación	Orquestación	Orquestación	Orquestación	Orquestación	Orquestación
Infraestructura	Inmutable	Inmutable	Inmutable	Inmutable	Inmutable	Inmutable	Inmutable
Lenguaje	Procedimiento	Declarativo	Declarativo	Declarativo	Declarativo	Declarativo	Declarativo
Gestión ciclo de vida	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Aprovisionamiento VMs	Sí	Sí	Sí	Sí	Sí	Contenedores	Contenedores
Networking	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Gestión Empaquetado	Sí	No	No	Sí	No	No	Sí
Templating	Sí	No	Sí	Sí	Sí	Sí	Sí
Aprovisionamiento de servicios	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Gestión backups	Sí	Sí	Sí	Sí	Sí	No	No
Gst relaciones entre servicios	Sí	Sí	Sí	No	No	Sí	Sí
Orientado a:	Múltiples máquinas	Múltiples máquinas	OpenStack	AWS	Azure	Múltiples máquinas	Múltiples máquinas

Tabla 9: Comparativa de herramientas orientadas a la orquestación.

La ETSI en cooperación con otros organismos de estandarización y grandes empresas de comunicaciones han definido un framework genérico y de referencia con el objetivo de servir de referencia a las empresas del sector. Este framework define una arquitectura genérica para permitir la orquestación de servicios de red. El framework se denomina MANO y es una especificación desarrollada por el grupo de trabajo MANO que pertenece a la ETSI [129]. Este grupo de trabajo se ha ido haciendo más conocido por su orientación a la administración y orquestación de proyectos NFV.

MANO NFV se encarga de la administración y orquestación de todos los recursos en un centro de datos virtualizado, incluidos los recursos de proceso, redes, almacenamiento y VMs. El enfoque principal de NFV MANO es permitir la incorporación flexible de los componentes de la red de manera rápida y libre de errores. La arquitectura del framework NFV se muestra en la Figura 21 se compone de los siguientes bloques funcionales [130]:

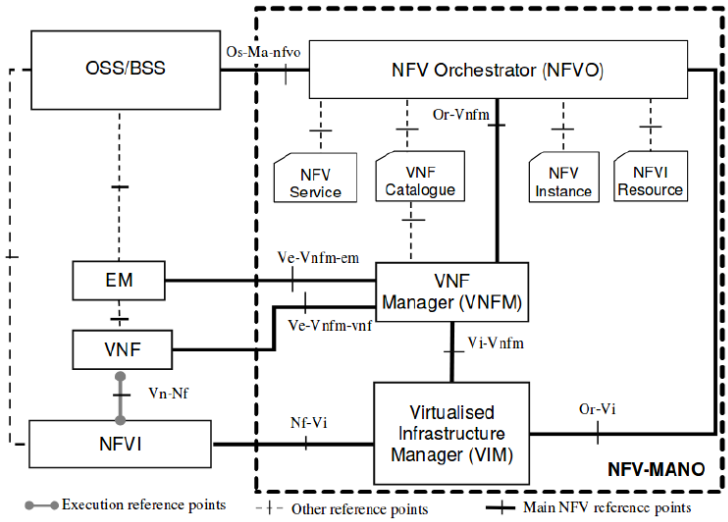

Figura 21: Arquitectura del framework MANO propuesto por la ETSI para la gestión de sistemas NFVs [131]. La figura muestra los bloques funcionales y los principales componentes software que componen cada uno de los bloques.

- **Operations/Business Support System (OSS/BSS)**: para garantizar la compatibilidad con los sistemas basados en esta arquitectura, la arquitectura ETSI MANO permite integrarse con API abiertas en sistemas OSS/BSS. El bloque funcional OSS/BSS es responsable de las funciones operativas y empresariales como el inventario de red, el aprovisionamiento de servicios, la configuración de red, la gestión de fallos, la captura de pedidos y la facturación. Sin embargo, para la interoperabilidad con el entorno virtualizado el módulo OSS/BSS no está preparado;

- **Sistema de Gestión de Elementos (EMS)**: este bloque funcional es responsable de la gestión de fallos, configuración, rendimiento y seguridad para una función de red virtual (VNF);
- **Función de red virtual (VNF)**: representa las funciones de red virtual (como enrutadores virtualizados, conmutadores y firewalls) implementados sobre hardware básico;
- **Infraestructura NFV (NFVI)**: representa la infraestructura física (computación, almacenamiento y redes) y el software donde se implementan, administran y ejecutan VNF;
- **Virtual Infrastructure Manager (VIM)**: responsable del control y gestión de los recursos NFVI (como computación, almacenamiento y redes). Las soluciones VIM disponibles incluyen OpenStack [107] y Amazon Web Services (AWS) [132] como sistemas operativos en la nube y OpenDayLight [133] y ONOS [134] como controladores SDN;
- **VNF Manager (VNFM)**: supervisa la configuración y la gestión del ciclo de vida (creación, actualización, escalado y finalización) de instancias VNF que se ejecutan sobre máquinas virtuales o contenedores;
- **NFV Orchestrator (NFVO)**: también conocido como orquestador de servicios de red NFV, es responsable de la orquestación de recursos NFVI a través de múltiples VIMs. También es el encargado de la gestión del ciclo de vida (incorporación, creación de instancias, escalado, actualización y finalización) de servicios de red para ofrecer conectividad de extremo a extremo (cadenas de funciones de servicio) y sectores de red.

Además de los bloques funcionales antes mencionados, ETSI NFV MANO propone cuatro repositorios utilizados para almacenar información de gestión y orquestación [130]:

- Servicio NFV: se trata de un conjunto de plantillas predefinidas que especifican los procedimientos seguidos para el servicio de incorporación, creación y finalización;
- Catálogo VNF: conjunto de plantillas que describe los atributos de los VNF;
- Instancia NFV: se utiliza para almacenar toda la información relativa a las funciones y servicios de red virtual;
- Recurso NFVI: se utiliza para almacenar información relacionada con NFVI.

El framework ETSI NFV MANO es un framework de referencia para la implementación de orquestadores debido a la participación de importantes organismos de estandarización y empresas de comunicaciones. A continuación se describen los proyectos más relevantes de código abierto relacionados con la orquestación y que en algunos casos están basados en las especificaciones recogidas en el framework ETSI NFV MANO. En la tabla 12 se resumen las características proporcionadas por cada uno de estos proyectos. La "compatibilidad con varios dominios" (consulte la Tabla 11) expresa la capacidad de orquestar recursos en varios dominios administrativos pertenecientes a diferentes proveedores de infraestructura.

XOS

XOS [135] es un servicio de orquestación desarrollado para trabajar bajo la arquitectura de CORD [136] (Central Office Re-architected as a Datacenter). CORD es una de las principales soluciones abierta de referencia orientada a redes 5G pero con capacidad de trabajar con las tecnologías SDN y NFV. CORD se caracteriza por ser un framework compuesto de módulos que se ejecutan en centros de datos como infraestructura en la nube[2]. Uno de esos módulos que lo componen es el orquestador XOS que se encarga del aprovisionamiento de los servicios que corren sobre CORD. CORD utiliza OpenStack para el aprovisionamiento de infraestructura como servicio (IaaS) y Open Networking Operating System (ONOS) para la gestión de la red separando los planos de control y datos. Como orquestador, XOS controla OpenStack, ONOS y M-CORD que es el framework correspondiente para la red móvil. XOS proporciona abstracciones que unifican los accesos desde diferentes proveedores de infraestructura de nube tanto pública como privada. Además, XOS es capaz de utilizar los propios sistemas de administración existentes en los centros de datos que proporcionan la infraestructura en la nube, para implementar mecanismos de control y asignación de recursos de bajo nivel.

XOS incluye las siguientes capas: (i) un modelo de datos que registra la información de estado del sistema, (ii) un conjunto de vistas personalizables que se ejecutan sobre del modelo de datos permitiendo el acceso a los servicios de orquestación y (iii) un controlador para la administración y distribución de información de estado en diferentes dominios. CORD se puede utilizar en múltiples ámbitos: redes residenciales, móviles y empresariales.

Cloudify para NFV MANO

Cloudify [116] es una plataforma NFV alojada en la nube y código abierto creada originalmente por la compañía GigaSpace para optimizar la orquestación y administración de NFV. Consiste en un NFVO para la gestión de servicios de ciclo de vida y un gestor VNF genérico. Es compatible tanto con dispositivos virtualizados (contenedores y máquinas virtuales) como

con dispositivos no virtualizados. Aunque Cloudify proporciona una implementación de la mayoría de los bloques funcionales ETSI NFV MANO, no es totalmente compatible. Para proporcionar una administración completa del servicio de ciclo de vida, Cloudify utiliza un blueprint basado en TOSCA (Topología y especificación de orquestación para aplicaciones en la nube y es capaz de soportar operaciones avanzadas de orquestación.

ONAP

ONAP [137] es un proyecto desarrollado por la Fundación Linux que permite el diseño, la creación de VNFs y la orquestación de extremo a extremo de servicios compuestos con VNFs. ONAP fue el resultado de la unión de OPEN-O [138] y ECOMP de AT&T [139]. Tanto OPEN-O como ECOMP fueron desarrollados para la orquestación de SDN y NFV. Onap incluye módulos de big data e inteligencia artificial (IA) los cuales permiten la optimización de directivas y además permiten automatizar la implementación y administración de servicios de red. Ha sido adoptado por proveedores de servicios globales como: AT&T, China Mobile, Ericsson, Nokia, Cisco y Huawei.

OSM NFVO

OSM [140] es una plataforma de gestión y orquestación NFV de código abierto alojada por ETSI. Esta plataforma fue desarrollada para alinearse con los modelos de información ETSI NFV y cumplir con los requisitos de producción.

OSM consta de dos orquestadores desacoplados, es decir, orquestador de recursos y orquestador de servicios. El orquestador de recursos proporciona orquestación en los dominios de tecnología SDN y en la nube. El orquestador de servicios controla la administración del ciclo de vida de los servicios de red y los VNF y a partir del modelos de datos YANG [4] [141] y [142].

OpenBaton NFVO

OpenBaton [143] es una implementación de código abierto de la NFVO basada en el estándar de referencia ETSI NFV MANO y la especificación OASIS TOSCA [144]. OpenBaton ofrece muchas características para el cumplimiento de la especificación ETSI NFV MANO. Algunas de las características más importantes incluyen, un NFVO para la orquestación de servicios de extremo a extremo, un VNFM genérico para la administración de infraestructura de múltiples proveedores, autoescalado y motores de administración de eventos. OpenBaton puede orquestar en varios dominios administrativos. Se puede ejecutar encima de diferentes NFVI que incluyen contenedores de AWS, Openstack, Docker y LXC. La desventaja

4 YANG es un lenguaje de modelado que facilita la definición de la semántica de datos operacionales, datos de configuración, notificaciones y operaciones.

de este proyecto es su pequeña comunidad lo que amenaza la capacidad de mantenimiento y el tiempo de vida del proyecto.

X-MANO

X-MANO [145] es una plataforma de orquestación NFV entre dominios. Esta plataforma incluye varias interfaces y módulos para garantizar la confidencialidad de la información y la programación del ciclo de vida del servicio entre dominios.

Gohan

Gohan [146] es un motor de orquestación SDN y NFV de código abierto liderado por NTT Innovation Institute [5]. Se basa en microservicios los cuales forman un único proceso desde donde se realiza el proceso de orquestación. El motor de orquestación de Gohan es accesible a través de varios componentes software independientes: una API REST, una interfaz a través de línea de comandos y una interfaz de usuario vía web. Además, Gohan soporta un modo de trabajo basado controlado por esquemas en el cual se utilizan esquemas basados en el formato JSON para definir servicios y directivas para poder implementar servicios. Gohan puede ser utilizado como una capa de orquestación de servicios de red en la parte superior de los servicios en la nube y también puede realizar tareas propias de MANO NFV para administrar VIMs(Virtual Infraestructura Manager) y elementos de red.

Tacker

Tacker [147] es un proyecto de código abierto liderado por OpenStack centrado en la creación de un orquestador de servicios de red compatible con ETSI NFV MANO (NFVO) y un VNFM genérico para la implementación y operación VNF. La plataforma VIM utilizada por Tacker es OpenStack. La NFVO proporciona orquestación de servicio completa en varias VIMs, optimización en la colocación de VNFs (mediante descriptores de servicio) y asignación de recursos. En paralelo, VNFM administra el ciclo de vida de la infraestructura NFV, incluida la creación/finalización, supervisión, migración, configuración y reparación automática de VNFs.

Tenor

TeNor [148] es un orquestador multiusuario/multi NFVI-PoP (NFVO) desarrollado en el proyecto FP7 T-NOVA. Al igual que Gohan, la arquitectura TeNor se basa en microservicios para un funcionamiento más modular del sistema. Uno de los módulos clave de TeNor es el microservidor de asignación de servicios, utilizado para asignar los VNF que componen un servicio de red solicitado y cumple la mejor ubicación disponible en la infraestructura. El módulo de asignación de servicios se implementa mediante el servicio de red y los descriptores VNF. TeNor adopta tanto el

5 https://gohan.cloudwan.io/

concepto de VNFM genérico como los VNFMs específicos propuestos por la especificación ETSI NFV MANO [149].

3.3.3 *Discusión*

En esta sección se han analizado diferentes herramientas diseñadas para permitir el aprovisionamiento automatizado de recursos y servicios sobre infraestructuras. La sección 3.3.1 ha presentado herramientas para realizar tareas de aprovisionamiento y de orquestación capaces de gestionar máquinas virtuales, servicios y recursos sobre múltiples proveedores de infraestructuras. En la tabla 3.3.1 se puede ver una comparación que permite comparar algunos de los aspectos más importantes relacionados con las herramientas analizadas en la sección 3.3.1.
Uno de los requisitos más importantes propuestos en la sección 2.1.6 es la compatibilidad con múltiples infraestructuras y como se puede ver en la tabla 3.3.1 las herramientas Heat, CloudFormation y ARM están orientadas a la infraestructura que proporciona su propia compañía por lo que no son de interés para implementar las soluciones propuestas en este trabajo. Las herramientas Kubernetes y Docker Swarm proporcionan un gran número de funcionalidades pero están orientadas a trabajar con contenedores, principalmente contenedores Dockers, por lo que estos orquestadores no servirían para gestionar VMs ni recursos de infraestructura que permitan instanciar completamente el entorno operativo de una máquina física. Finalmente, las herramientas Juju y Terraform proporcionan un gran número de funcionalidades y capacidades que les hacen ser adecuadas para trabajar con las soluciones propuestas de la secciones 5 y 6. La tabla 10 muestra los requisitos satisfechos por cada una de estas herramientas según los requisitos identificados en la sección 2.1.6.

La sección 3.3.2 analiza las principales herramientas de orquestación diseñadas para trabajar en entornos NFV y SDN sobre infraestructuras basadas en la nube. En la tabla 11 se presenta el alcance de cada una de estas soluciones, por ejemplo, la tabla muestra que todas las herramientas están diseñadas para trabajar con NFV además de permitir la gestión de infraestructuras en la nube, excepto X-MANO que es un orquestador diseñado para gestionar sistemas NFV pertenecientes a dominios distintos. Otra característica importante de estas herramientas es que no todas incorporan gestión de redes a través de mecanismos SDN.

Las herramientas analizadas en la sección 3.3.2 se caracterizan por estar basadas en la mayor parte en el framework MANO de la ETSI [129]. En

Requisito	Juju	Terraform	Heat	CloudFormation	ARM
R1	✓	✓	✓	✓	✓
R2				✓	✓
R3	✓	✓			
R4	✓	✓	✓	✓	✓
R5	✓			✓	
R6	✓	✓	✓	✓	✓
R7	✓	✓	✓	✓	✓
R8	✓	✓	✓		
R9	✓	✓	✓	✓	✓
R10	✓	✓	✓	✓	
R11	✓	✓	✓	✓	✓
R12	✓	✓			
R13					
R14	✓	✓	✓		
R15					
R16	✓	✓	✓		
R17					
R18					
R19	✓	✓			

Tabla 10: Requisitos cumplidos por cada una de las herramientas de aprovisionamiento analizadas en el trabajo.

Soluciones de Orquestación	Cloud	NFV	SDN
CORD/XOS [136]	✓	✓	✓
ONAP [137]	✓	✓	✓
OSM NFVO[140]	✓	✓	✓
Cloudify [116]	✓	✓	
OpenBaton [143]	✓	✓	
X-MANO [145]		✓	
Gohan [146]	✓	✓	✓
Tacker [147]	✓	✓	
TeNor [148]	✓	✓	✓

Tabla 11: La tabla muestra las soluciones de orquestación analizadas en el capítulo e indica para cada una de ellas si proporcionan funcionalidades de para gestionar recursos en la nube, para trabar con sistemas NFV y si incorporar mecanismos de SDNs.

Soluciones de Orquestación	VNFM	VIM	NFVO	OSS/BSS	Multi-site
CORD/XOS [136]	✓		✓		✓
ONAP [137]	✓	✓	✓	✓	✓
OSM NFVO [140]	✓	✓	✓		
Cloudify [116]	✓		✓		
OpenBaton [143]	✓	✓	✓		✓
X-MANO [145]			✓		✓
Gohan [146]	✓		✓	✓	
Tacker [147]			✓		
TeNor [148]			✓		

Tabla 12: La tabla muestra como las herramientas analizadas en la sección cumplen con las especificaciones definidas por el framework de referencia MANO propuesto por la ETSI.

la tabla 12 se muestra para cada una de las herramientas mostradas en la sección, que componentes del framework MANO de la ETSI son implementados por cada una de las soluciones. Como se puede ver en la tabla, todas las herramientas implementa un orquestador para NFV y la mayor parte también implementan un gestor de VNF que les permite gestionar las instancias VNF desplegadas sobre VMs o contenedores.

Acorde con los requisitos mostrados en la sección 2.1.6, estas herramientas pueden satisfacer algunos requisitos ya que las herramientas proporcionan catálogos de servicios (R2), soporte multiplataforma (R3), gestionan el encadenamiento de servicios (R13), entre otros... Sin embargo, hay otros muchos requisitos que no pueden ser proporcionados por estas herramientas ya que están orientadas a gestionar principalmente servicios NFV y este trabajo trata de gestionar cualquier tipos de servicios.

3.4 PROYECTOS Y SOLUCIONES COMPLETAS

En esta sección se recogen los proyectos con herramientas más avanzadas capaces de realizar una gestión avanzada de las tareas de orquestación de servicios. Estos proyectos ya componen sistemas completos que permiten realizar tareas de gestión de instancias de máquinas virtuales, configuraciones de servicio, orquestación y composición de servicios virtuales.

Las soluciones presentadas en esta sección se diferencia de las herramientas presentadas en la sección 3.2.1 y 3.3.1 en que no son sólo herramientas para gestionar software. Si no que definen sistemas complejos de servicios, completamente funcionales y que pueden integrar algunas de las herramientas vistas en 3.2.1 y 3.3.1. Cada una de estas soluciones puede alojar los servicios de múltiples proveedores de servicios. Además, son capaces de proporcionar mecanismos para la gestión completa de funciones de red virtualizadas. A continuación se describen los proyectos más relevantes:

T-NOVA

T-NOVA es un proyecto FP7 [6] que tiene el objetivo principal de implementar un framework totalmente compatible con la arquitectura NFV de ETSI permitiendo la gestión y orquestación de funciones de red sobre el NFVI. Además, el objetivo de T-NOVA es proporcionar un mercado abierto que pueda ser utilizado por los operadores como un catálogo público promoviendo la oferta de servicios VNF y facilitando la actividad comercial y la interacción fluida entre las diversas partes interesadas del negocio que interactúan con el sistema T-NOVA.

T-NOVA Marketplace proporciona una interfaz intuitiva al framework MANO y permite a los desarrolladores de servicios y VNFs publicar y

6 https://ec.europa.eu/eurostat/cros/content/fp7-projects

describir ofertas de servicios, así como permitir a los clientes navegar y seleccionar servicios, implementarlos y administrarlos. Una descripción más detallada del T-NOVA Marketplace se puede encontrar en [150].

El componente central del framework T-NOVA es el T-NOVA Orchestrator ("TeNOR"). Su objetivo clave es abordar las operaciones de administración del ciclo de vida de los NS y los VNF a través de infraestructuras de red distribuidas y virtualizadas.

Open Source MANO

Open Source MANO (OSM)[140] es una plataforma de código abierto que permite la implementación del componente MANO desde el framework de NFV. El proyecto OSM esta desarrollado por la ETSI. Su objetivo es crear un orquestador de servicios de red de extremo a extremo con la capacidad de diseñar y automatizar servicios de red reales teniendo en cuenta todas las características complejas del entorno de implementación. OSM ofrece soporte para agilizar el desarrollo de la tecnología NFV garantizando la comunicación entre el orquestador y otros componentes de la arquitectura ETSI NFV, como el Network Function Virtualization Infrastructure (NFVI) o las funciones de red [151]. En la Figura 22 se muestra cómo OSM interactúa con los componentes VIM (Virtual Infraestructura Manager) y VNF en la arquitectura NFV. El VIM se utiliza para crear VNF y establecer enlaces virtuales entre ellos. Para ello, OSM requiere que VIM disponga de una API de conexión que le permita acceder a los VNF desplegados para ejecutar las configuraciones [151].

OSM está diseñado para trabajar con plataformas de infraestructura virtual, que están controladas por un administrador de infraestructura virtual (VIM) y plataformas SDN. Estas últimas son controladas por un administrador de infraestructura que cuente con un controlador SDN [151]. Para ello, OSM también incluye OpenVIM como VIM de referencia que respeta las directrices de NFV, con interfaces que permiten la conexión con los nodos de computación y almacenamiento desde el NFVI y utilizando un controlador OpenFlow (OFC) para implementar la infraestructura de red [152].

OpenBaton

OpenBaton [153] es una implementación de código abierto de la NFVO basada en el estándar de referencia ETSI NFV MANO y la especificación OASIS TOSCA. OpenBaton ofrece muchas características para el cumplimiento de la especificación ETSI NFV MANO. Algunas de las características más importantes incluyen, un NFVO para la orquestación de servicio de extremo a extremo, un VNFM para la administración de infraestructura de múltiples proveedores, FCAPs, escalado automático y motores de administración de eventos. OpenBaton puede organizarse en varios dominios

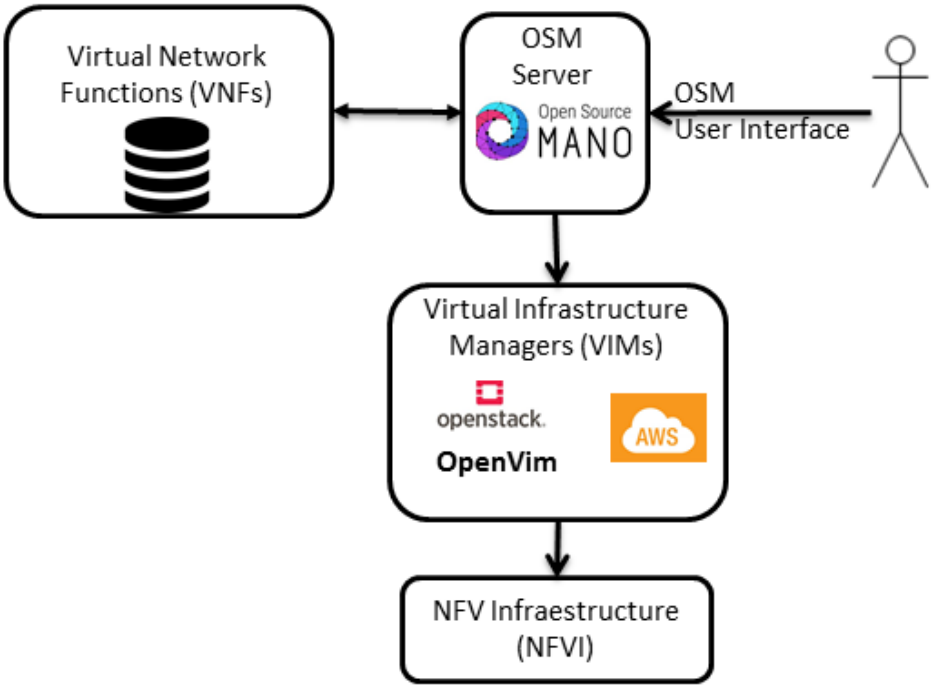

Figura 22: Arquitectura básica de la plataforma OSM.

administrativos. Se puede ejecutar encima de diferentes NFVI que incluyen AWS, Openstack, contenedores Docker y contenedores LXC.

Open Baton también proporciona un kit de desarrollo de software (SDK) para implementar nuevos VNFM específicos. Además, proporciona un mecanismo de plugin que permite la integración sencilla de componentes externos, como el sistema de monitorización y el Administrador de Infraestructura Virtualizada (VIM). Por último, también dispone de un motor de eventos permite la integración de componentes de terceros que mejoran el conjunto de funcionalidades que proporciona el framework. Puede comunicarse con uno o más administradores de VNF a través de una API REST o un sistema de mensajería tal como el protocolo AMQP de RabbitMQ.

OpenStack+Tacker

OpenStack+Tacker se compone de la plataforma OpenStack que proporciona los servicios principales de cloud computing: computación, redes, almacenamiento, identidad e imagen. Y por otro lado esta el proyecto Tacker (analizado en la sección 3.3.2) que su objetivo es realizar las tareas de gestor genérico de VNFs y orquestador de NFV (NFVO) para trabajar con servicios de red y funciones de red virtual (VNF) sobre una infraestructura basada en OpenStack. Tacker se basa en el framework ETSI MANO para definir su arquitectura. Tacker permite orquestar servicios de red extremo a extremo utilizando VNFs [155].

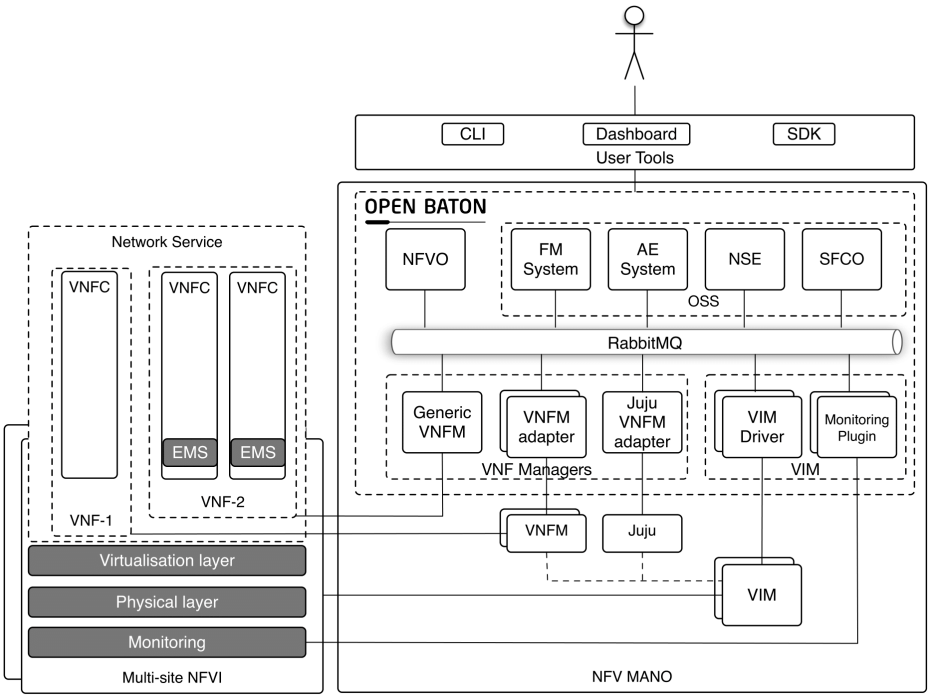

Figura 23: Arquitectura de OpenBaton desde [154] en la que se muestran los diferentes bloques que agrupan diferentes funcionalidades. La figura muestra los tres bloques principales: bloque User Tools que contiene las principales interfaces de usuario; bloque NFV MANO que contiene los componentes software para realizar las tareas de gestión y orquestación de servicios NFV; y finalmente el bloque Multi-site NFVI que agrupa los bloques encargados de gestionar los recursos de infraestructura.

Tacker proporciona recursos y funciones de red virtual extremo a extremo tales como:

- VNF Catalog es un repositorio de VNFDs que incluyen definiciones VNF mediante plantillas TOSCA. Las plantillas TOSCA son un mecanismo que se utilizan para gestionar la información de las VNFs. Cuando se desarrolla una nueva VNF, los desarrolladores y los proveedores de VNFs crean manualmente la definición de la VNF (VNFD) y la cargan en el catálogo VNF.
- VNFM. Es el gestor de las VNFs y se encarga de gestionar el ciclo de vida de la cada VNF (acciones de crear actualizar y eliminar). También se encarga de tareas de aturorecuperación y autoescalado de VNFs a partir de políticas predefinidas y establece la configuración inicial de cada VNF.

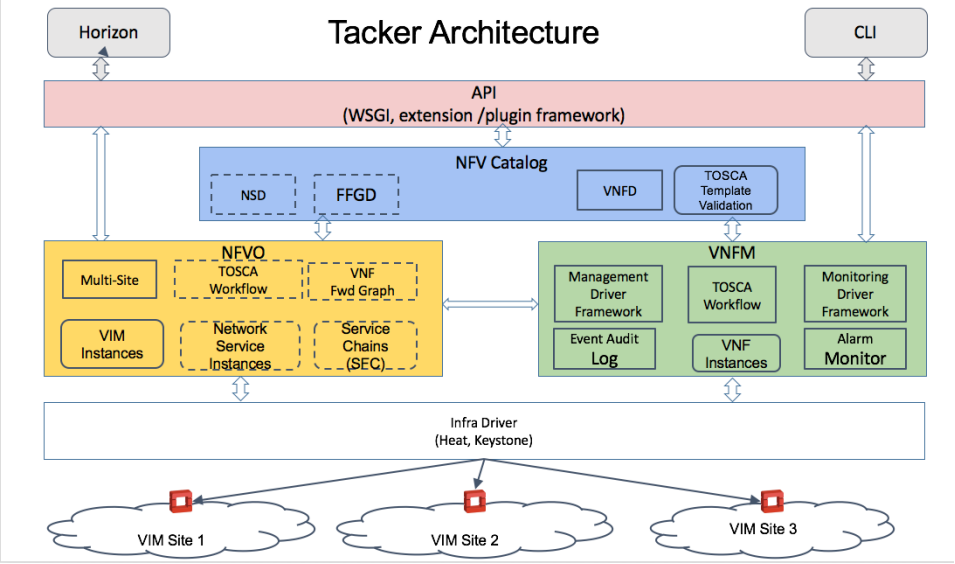

Figura 24: Arquitectura de OpenStack+Tacker desde [155]. La figura muestra la arquitectura del orquestador tacker donde la funcionalidad está agrupada por bloques funcionales representados con diferentes colores.

- NFVO. Es el encargado de la implementación de servicios de red extremo a extremo a través del uso de plantillas VNFs. Para ello aplica las políticas de despliegue de VNF para ajustarse a los parámetros de funcionamiento preestablecido.

Cloudify

Cloudify [116] es una plataforma NFV de código abierto creada originalmente por GigaSpace para optimizar la orquestación y administración de sistemas NFV. Cloudify consiste en un NFVO para la gestión del servicio del ciclo de vida y un administrador de VNF genérico. Cloudify es compatible tanto con dispositivos virtualizados (contenedores y máquinas virtuales) como con dispositivos no virtualizados. Aunque Cloudify proporciona una implementación de la mayoría de los bloques funcionales ETSI NFV MANO pero no es totalmente compatible.

Las principales características de Cloudify son [156]:

- Modelado de aplicaciones: describe una aplicación con todas sus características (infraestructura, middleware, código de aplicación, scripts, configuración de herramientas, métricas y registros), de forma genérica y descriptiva. El lenguaje Cloudify se basa en TOSCA

- Orquestación: mantiene y ejecuta aplicaciones. Además de la creación de instancias, es posible realizar acciones como modificar el tamaño de los recursos, operación de recuperación y de mantenimiento;

- Extensibilidad: proporciona abstracción de componentes reutilizables para el sistema. Dispone de la capacidad para modelar cualquier cosa en un lenguaje descriptivo, por ejemplo, IaaS, entornos de computación en la nube, herramientas de administración de configuración, etc;
- Seguridad: proporciona control sobre quién puede acceder y realizar operaciones sobre los componentes del sistema.

3.4.1 *Discusión*

En la sección 3.4 se han analizado los proyectos que integran herramientas que les permiten configurar soluciones para realizar una gestión avanzada de las tareas de orquestación de servicios. Estos proyectos ya componen sistemas completos que permiten realizar tareas de gestión de instancias de máquinas virtuales, configuraciones de servicio, orquestación y composición de servicios virtuales. La tabla 13 presenta un resumen de las funcionalidades que proporcionan cada una de los proyectos presentados en la sección 3.4. Al analizar dicha tabla se puede observar las necesidades actuales del mercado, por ejemplo, todos los proyectos proporcionan soporte para OpenStack como solución para gestionar infraestructuras. Además, la tabla también refleja como las VMs son elementos claves en los nuevos sistemas y continúan siendo más relevantes que los contenedores ya que estos últimos no son implementados por OSM y T-NOVA. La tabla también muestra la importancia de los lenguajes descriptivos ya que es el preferido para realizar el empaquetado de servicios a través de lenguajes YAML como TOSCA o HOT. Para validar lo adecuados que son las soluciones presentadas por cada uno de estos proyectos se ha desarrollado la tabla 14 en la que se muestran los requisitos satisfechos por cada uno de ellos. Como se puede ver en dicha tabla, la mayoría de los requisitos son satisfechos por la mayoría de los proyectos pero algunos requisitos continúan sin ser cubiertos. Por ejemplo, de los requisitos mostrados en la sección 2.1.6 hay algunos requisitos que no son contemplados por estos proyectos tales como R14, R15 y R19 los cuales reflejan la necesidad de optimizar el despliegue de servicios y componentes software tomando en cuenta el software ya instalado sobre el sistema (R14), la necesidad de simplificar las soluciones al menor número de componentes con el objetivo de facilitar la curva de aprendizaje de los nuevos técnicos (R15) y tratar de no ajustarse a arquitecturas en particular sino todo lo contrario, ajustar la arquitectura base a los servicios con las funcionalidades finales que se desean proporcionar los clientes (R19).

	Open Baton	**OSM**	**OpenStack+Tacker**	**Cloudify**	**T-NOVA**
Adaptador de infraestructuras VIM	OpenStack Amazon Docker Test	Openstack VMWare AWS OpenView	VIM: OpenStack Kubernetes	VIM: AWS Azure Openstack Vsphere	VIM: Openstack Open Daylight
Entorno de Virtualización	VM Contenedores	VM	VM Contenedores	VMs Contenedores	VMs
Operaciones y Ciclo de Vida VNFs	1.Instanciación 2.Configuración 3.Arranque 4.Parado 5.Finalización 6.Escalado	1.Modelado 2.On-boarding 3.Creación NS 4.NS funcionamiento 5.Finalización NS	N/A	1.Procesamiento de cadenas de eventos 2.Escaladado de métricas 3.Agregación 4.Análisis...	1.Inciación 2.Parada 3.Reinicio 4.Escalado 5.Desescalado
Empaquetado	TAR CSAR (TOSCA)	Documentos basados en YAML	TOSCA	TOSCA	HOT
Interfaces	CLI Dashboard (GUI)	Dashboard (GUI) Interface Web CLI	CLI Horizon	CLI Web	N/A
Soporte Multidominio	No	No	No	No	Sí

Tabla 13: Tabla comparativa de proyectos descritos en este trabajo los cuales integran la tecnología NFV-SDN.

Requisito	MANO ETSI	T-NOVA	OSM	OpenBaton	OpenStack+Tacker	Cloudify
R1	✓	✓	✓	✓	✓	✓
R2	✓	✓	✓	✓	✓	✓
R3	✓	✓	✓	✓	✓	✓
R4	✓	✓	✓	✓	✓	✓
R5	✓	✓	✓	✓	✓	✓
R6	✓	✓	✓	✓	✓	✓
R7	✓	✓	✓	✓	✓	✓
R8	✓	✓	✓	✓	✓	✓
R9	✓	✓	✓	✓	✓	✓
R10	✓	✓	✓	✓	✓	✓
R11	✓	✓	✓	✓	✓	✓
R12	✓	✓	✓	✓	✓	✓
R13	✓	✓	✓	✓	✓	✓
R14						
R15						
R16	✓	✓	✓	✓	✓	
R17	✓	✓	✓	✓	✓	✓
R18		✓	✓	✓	✓	✓
R19						

Tabla 14: Requisitos cumplidos por cada una de las soluciones completas analizadas en el trabajo.

3.5 PRINCIPIOS DE IMPLEMENTACIÓN PARA IOC

Para implementar IaC correctamente existen varios principios para diseñar e implementar infraestructura en plataformas en la nube. Estos principios articulan el razonamiento para usar las tres prácticas básicas (definir todo como código, probar y entregar continuamente y construir piezas pequeñas). A continuación se presentan' los principios:

- **Asumir que los sistemas no son fiables.** Para la administración de todo sistema cloud se debe asumir que el sistema se ejecuta en hardware poco fiable. Esto significa que los administradores deben estar preparados y tener planes de contingencia para desconectar partes del sistema debido a errores no planificados. Esto puede llegar a implicar a desconectar el negocio mientras se parchean o actualizan los sistemas.

- **Hacer que todo sea reproducible.** Una forma de hacer que un sistema sea recuperable es asegurarse de que puede reconstruir sus piezas sin esfuerzo y de forma fiable. Para ello se debe prestar especial atención aspectos como la configuración, las versiones de software y las dependencias como código. La reproducibilidad no solo facilita la recuperación de un sistema fallido, sino que también le ayuda a:
  - Hacer que los entornos de prueba sean coherentes con la producción.
  - Replicar sistemas en todas las regiones para la disponibilidad.
  - Agregar instancias bajo demanda para hacer frente a la alta carga.
  - Replicar sistemas para dar a cada cliente una instancia dedicada.

  La capacidad de construir y reconstruir sin esfuerzo cualquier parte de la infraestructura es importante debido a que elimina el riesgo y el miedo a realizar cambios, y puede manejar los errores con confianza. También permite aprovisionar rápidamente nuevos servicios y entornos.

- **Minimizar la variación.** A medida que un sistema crece, se vuelve más difícil de entender, más difícil de cambiar y de reparar. Los sistemas crecen con el número de piezas, y también con los tipos de piezas diferentes. Por lo tanto, una forma útil de mantener un sistema manejable es tener menos tipos de piezas, para mantener la variación baja. Es más fácil administrar cien servidores idénticos que cinco servidores completamente diferentes.

- **Asegúrese de que puede repetir cualquier proceso.** Los administradores deben ser capaz de duplicar cualquier cambio realizado sobre su infraestructura. Es más fácil repetir acciones utilizando scripts y herramientas de administración de configuración que hacerlo a mano. Pero la automatización puede ser mucho trabajo, especialmente si no estás acostumbrado.

- **Desarrollar software desechable.** Desarrollar software desechable significa crear componentes software con características especificas para algo en concreto, software con un fin específico que tiene un objetivo concreto y que una vez que ha cumplido su misión puede ser desechado sin afectar a otras partes del sistema. Esto ayuda a reducir la complejidad de los sistemas y a tener sistemas más limpios, creando flexibilidad operativa, disponibilidad y escalabilidad. Aplicar estos principios en el despliegue de servicios definidos por código da como resultado sistemas capaces de hacer frente a la demanda requerida por los consumidores de servicios, mediante la aplicación de múltiples mecanismos de replicación, adaptación de las necesidades de recursos y distribución geográfica de los servicios necesarios para cumplir con dicha demanda.

3.6 RESUMEN DEL CAPÍTULO

En los últimos años, con el auge de la computación en la nube y el escalado de aplicaciones distribuidas de gran tamaño, la demanda de aprovisionamiento automático de infraestructuras de TI ha crecido de manera importante [157] [158]. Esta tendencia va acompañada del auge del movimiento DevOps [159], que hace hincapié en la integración del desarrollo y las operaciones, para cerrar la separación técnica y psicológica entre estos dos roles, tradicionalmente separados. Para habilitar esta transición, los procesos de implementación de software robustos y repetibles son de suma importancia. Este estado de cosas ha llevado a la adopción de herramientas de gestión de la configuración [160] [161], que despliegan y configuran automáticamente los sistemas de software.

Algunos de los avances más recientes en esta dirección durante los últimos años se pueden ver en herramientas que soportan el proceso de desarrollo de aplicaciones SDN. Hay una variedad de este tipo de herramientas, como VeriFlow [162], y lenguajes de alto nivel, como Frenetic [163] o el proyecto NetIDE [164] el cual tiene como objetivo proporcionar un entorno de desarrollo totalmente integrado que respalde todo el ciclo de vida de desarrollo de las aplicaciones de controladores SDN. Sin embargo, todos ellos se centran en el desarrollo de aplicaciones de control SDN y no tienen en

cuenta las funciones de red genéricas. El proyecto UNIFY [165], en cambio, ofrece una primera idea de cómo aplicar el modelo DevOps a NFV. Proporcionan, por ejemplo, una herramienta de depuración multi-componente llamada Epoxide [166]. En comparación con estas herramientas, el enfoque de SONATA es un paso adelante y combina un potente kit de desarrollo de software (SDK) con una plataforma de servicio flexible para proporcionar soporte extremo a extremo para desarrolladores de servicios.

En la categoría de orquestación y gestión, las soluciones provienen principalmente de la evolución de la plataforma Cloud Orchestration como OpenStack [107] o OpenNebula [100], que proporcionan la funcionalidad básica para implementar y administrar máquinas virtuales predefinidas únicas, pero no pueden manejar servicios compuestos. Otras soluciones, como OpenStack Heat [109], Terraform [127] y ADT [167], son capaces de implementar servicios completos compuestos por varias máquinas virtuales, pero no se centran en las necesidades específicas de la función de red.

Los enfoques y proyectos más notables que se centran directamente en la orquestación del servicio NFV se pueden dividir en dos categorías. El primero consiste en proyectos de investigación, como T-NOVA [168] y UNIFY [165], [169], y la segunda categoría consiste en herramientas de código abierto, como OpenMANO [170], Open-Baton [153] y OpenStack Tacker [155].

Las plataformas de servicio de T-NOVA y UNIFY proporcionan funcionalidades básicas de orquestación para servicios encadenados descritos por un gráfico de servicios. La arquitectura de UNIFY tiene como objetivo la creación de servicios automatizados y dinámicos y la orquestación recursiva de recursos. Su orquestador incluye algoritmos de optimización para la colocación de componentes de servicio. Las acciones específicas del servicio relacionadas con la colocación y el escalado se implementan como un componente de servicio. T-NOVA a través de TeNOR [171] es capaz de orquestar servicios de red distribuidos en varios centros de datos (NFVI-PoPs). T-NOVA admite el encadenamiento de varios VNF en cada servicio de red. Además, T-NOVA proporciona un mercado VNF para VNF de terceros.

OpenMANO y OpenStack Tacker pretenden ser implementaciones de referencia de la capa MANO definidas en la arquitectura ETSI NFV ISG [172] pero ambas están al comienzo de su desarrollo. Otras soluciones de orquestación académica son Cloud4NFV [173] y vConductor [174].

Todas las herramientas de orquestación presentadas, en gran medida, siguen el mismo principio e intentan crear una única solución de orquestación para diferentes tipos de servicios. Esto crea varias restricciones para los desarrolladores de servicios, ya que no pueden influir en el proceso de

orquestación como tal. Por ejemplo, la colocación y el escalado de servicios y sus componentes.

Además, estas soluciones están diseñadas principalmente para trabajar con componentes de red definidos en las especificaciones NFV-SDN de la ETSI. Lo que hace difícil implementar nuevos modelos de arquitectas que difieran de las especificaciones preestablecidas. Por esta razón se plantea aprovechar parte de los trabajos propuestos por la ETSI y adaptarlos según es estado actual de la industria. En esta tesis se ha analizado y puesto énfasis en los mecanismos de IaC como solución a la definición y despliegue de nuevas infraestructuras. La capacidad de adaptación a distintos proveedores de infraestructuras y el uso de herramientas y soluciones basadas en lenguajes de definición de infraestructuras son los pilares más importantes sobre los que se sostiene el trabajo de esta tesis.

Como se ha visto a lo largo de este capítulo, existen varias herramientas que hacen uso de lenguajes específicos para definir la infraestructura de los sistemas. Por lo general, estas herramientas se basan en lenguajes específicos de dominio (DSL) para definir la infraestructura de este modo, ecosistemas, redes, dispositivos y servicios que podrían recrearse en varios entornos de nube. Algunos de estos lenguajes son compatibles con varias plataformas en la nube. Sin embargo, la variedad de iniciativas de IaC hace que algunas herramientas introduzcan modificaciones en los lenguajes por lo que requiere que los desarrolladores tengan conocimiento de cada uno de las infraestructuras de nube disponibles y que podrían ser el destino de aquel ecosistema.

Como resultado del análisis hecho en los apartados anteriores, hay evidencia de una proliferación de herramientas, técnicas, lenguajes y enfoques de automatización de la Infraestructura como código, pero, por otro lado, el estándar desarrollado como clave llamado TOSCA, es adoptado por apenas un 20 % del total de practicantes [175]. Esto puede interpretarse como una deficiencia del estándar en términos de su difusión y explotación o tal vez, una divergencia de su viabilidad de los requisitos reales que los profesionales plantean.

Además, se observa una gran diversidad de tipos de herramientas, que van desde herramientas de orquestación (por ejemplo, Terraform, Juju) hasta la gestión de configuraciones (por ejemplo, Puppet), la gestión de topologías (por ejemplo, Kubernetes), la contenedorización (por ejemplo, Docker) y mucho más.

De estos datos, se desprende que el código de infraestructura es políglota por diseño y, por lo tanto, los usuarios de estas herramientas deben esforzarse por encontrar las mezclas, patrones y cualquier anti-patrones que coincidan o existan en su práctica y perspectiva teórica.

En [175] se presenta una encuesta en la que indica que el 30 % de los encuestados, conocen, Kubernetes, Vagrant [176], Chef, Terraform, Ansible y Docker. De hecho, cada una de estas herramientas se ocupa de un aspecto diferente del desarrollo de IaC: Kubernetes permite la orquestación de contenedores de cualquier tipo; Vagrant permite definir y administrar máquinas virtuales; Chef y Ansible se ocupan de la gestión de la configuración de los servicios; Terraform tiene como objetivo organizar los servicios implementados en diferentes infraestructuras (VM, contenedores, nubes públicas y privadas); Docker es la principal elección al construir contenedores. Otra observación derivada de los resultados sobre la adopción de herramientas de la IaC se refiere a la estandarización del lenguaje. A pesar del esfuerzo de estandarización detrás de TOSCA, los orquestadores habilitados para TOSCA todavía están lejos de ser la solución estándar para el desarrollo de la IaC.

4

FEDERACIÓN DE REDES DE DISPOSITIVOS MULTIMEDIA DOMÉSTICOS

Para comprender el problema de la compartición de contenidos multimedia, hay que tener en cuenta que el crecimiento de los contenidos multimedia en los últimos años es un hecho contrastado, las nuevas posibilidades que nos ofrecen los dispositivos móviles facilitan la creación y la calidad de los contenidos multimedia generados. Por lo general estos contenidos generados tienen un fin y en muchos casos su finalidad es mostrar algo a otras personas, lo que significa que un gran porcentaje de esos contenidos va a ser compartido de alguna manera. Existen varias formas de compartir los contenidos, a través de plataformas de distribución de contenidos tales como lo hacen grandes compañías tales como Netflix[1], Amazon[2], etc. O a través de redes sociales como TikTok[3], Instagram[4] o Facebook[5] donde los usuarios suben el contenido que quieren compartir con otros usuarios. Otras aplicaciones tales como Whatsapp[6], Telegram[7], etc permiten compartir contenidos con usuarios con los que se tiene un contacto más estrecho tienen características muy similares a las redes sociales tradicionales.

Finalmente, puede compartirse contenido a contactos más personales a través de mecanismos de conexión en los que los contenidos se transfieren entre dispositivos directamente sin utilizar infraestructura de terceros. Esta opción de compartir contenidos no sólo tiene la ventaja de no necesitar estar conectados a Internet y de estar dado de alta en sistemas de terceros, si no que permite a los usuarios implicados conservar la propiedad de los contenidos y no pierden el control ni los derechos de usuario sobre contenidos personales.

Este último caso, objeto de esta contribución permite crear ecosistemas digitales formados por redes de dispositivos heterogéneos interconectados que necesitan autoconfiguracion de red como DHCP [177], DNS [178],SLP [179], etc. y pueden recurrir a protocolos de descubrimiento de servicios como Universal Plug and Play (UPnP) [180], ZeroConf [181], Jini [182], Digital Living Network Alliance (DLNA) [183].

1 https://www.netflix.com
2 https://www.amazon.com
3 https://www.tiktok.com
4 https://www.instagram.com
5 https://www.facebook.com
6 https://www.Whatsapp.com
7 https://web.telegram.org

En la actualidad existen multitud de dispositivos que implementan una o más de estas tecnologías proporcionando abstracciones para la interacción entre dispositivos en el hogar. Sin embargo, se limitan a operar dentro de la red domésticas y no son capaces de operar entre dispositivos localizados en redes situadas fuera del dominio doméstico. Esta contribución permite extender el alcance de los dispositivos basados en el protocolo UPnP más allá de las redes locales y les proporciona funcionalidades adicionales permitiendo crear grupos y comunidades cerradas de dispositivos. Estos grupos de dispositivos pueden compartir sus contenidos de la misma manera que lo harían si se encontrasen en la misma red local. El trabajo presentado en este capítulo fue desarrollado a partir de los trabajos realizados en los proyectos de I+D: Raudos2[8] y HAUS[9]. Los artículos [7], [8] y [6] también han sido relevantes para el desarrollo de este capítulo.

4.1 MOTIVACIÓN

La proliferación de la banda ancha, el incremento de la capacidad de los dispositivos móviles que permite almacenar gran cantidad de contenidos multimedia y el deseo de disponer de acceso a los contenidos desde cualquier dispositivo digital del usuario, ha crecido la importancia de disponer de tecnologías para facilitar el intercambio de recursos y contenidos a través de redes doméstica remotas.

Estas tecnologías disponen de la capacidad de ofrecer una amplia gama de servicios deseables a los consumidores, incluyendo el intercambio de contenido, la formación de comunidades de redes domésticas y la composición de servicios entre múltiples redes domésticas.

Sin embargo, hay una serie de desafíos significativos en la ampliación de las tecnologías centradas en las tecnologías que permiten conectar, interactuar y compartir contenidos a través de esas tecnologías. Los principales desafíos podrían ser:

1. Rendimiento: los protocolos y arquitecturas que se han diseñado e implementado para interconectar dispositivos de consumo los cuales están optimizados para funcionar dentro del hogar por lo que al extender su alcance a redes inter-dominio deben asegurar que no hay una degradación significativa en la calidad de la experiencia.

2. Seguridad: los protocolos que proporcionan mecanismos de compartición de contenidos en las redes domésticas no están preparados para

8 RAUDOS2 Red Interactiva Multiplataforma de Distribución de Contenidos Audiovisuales (TSI-020302-2010-67) (Proy. Nacional)

9 HAUS-Hogar digital y contenidos Audiovisuales adaptados a los Usuarios (IPT-2011-1049-430000) (Proy. Nacional)

contemplar amenazas de seguridad cuando estos protocolos trabajan con entornos multi-dominio. Esto implica la consideración sobre este tipo de escenario.

Existe por otro lado, un problema de gestión de derechos digitales, dado que tanto los contenidos comerciales como los generados por el usuario, podrían tener restricciones de acceso que debe respetarse tanto en la red local como en interacciones entre dispositivos pertenecientes a dominios administrativos (o redes) diferentes [184].

3. Capacidad de gestión: las actuales tecnologías para compartir contenidos digitales sobre una red doméstica deben ser capaces de gestionar redes formadas por un número de dispositivos relativamente pequeño.

   A la hora de conectar diferentes redes, pese a que dichas redes tengan un pequeño número de dispositivos conectados individualmente, el número global de dispositivos a gestionar, considerando todas las redes simultáneamente, crece y con ello su complejidad. Se necesitan mecanismos adicionales para permitir este incremento de dispositivos para que el funcionamiento de los sistemas sea eficiente.

4. Dinamismo: el estado de los dispositivos de las redes domésticas son altamente dinámicos, los dispositivos de consumo a menudo están ocupados, apagados o no disponibles. Además, estos dispositivos se conectan y desconectan con frecuencia de la red (por ejemplo, dispositivos móviles que se conecta o desconectan cuando los usuarios acceden al hogar o cuando los usuarios adquieren nuevos dispositivos).

4.2 APROXIMACIÓN

Para presentar el contexto del problema, a través de la Figura 25 se va a introducir un escenario de aplicación.

En ese escenario, los usuarios se encuentran en diferentes ubicaciones físicas y comparten contenido multimedia con otros usuarios a través de sus dispositivos de la misma manera que si estuviesen todos conectados a la misma red local. Los diferentes usuarios (azul, verde y amarillo) disponen de dispositivos con capacidades de conexión basados en UPnP y se encuentra conectados a redes locales que conectan a Internet (excepto el usuario azul que está conectado a la red del operador móvil contratado).

Los dispositivos del usuario se unen a una Red Virtual UPnP (VUN), que será explicada en la sección 4.3, de forma que cada usuario pueda acceder

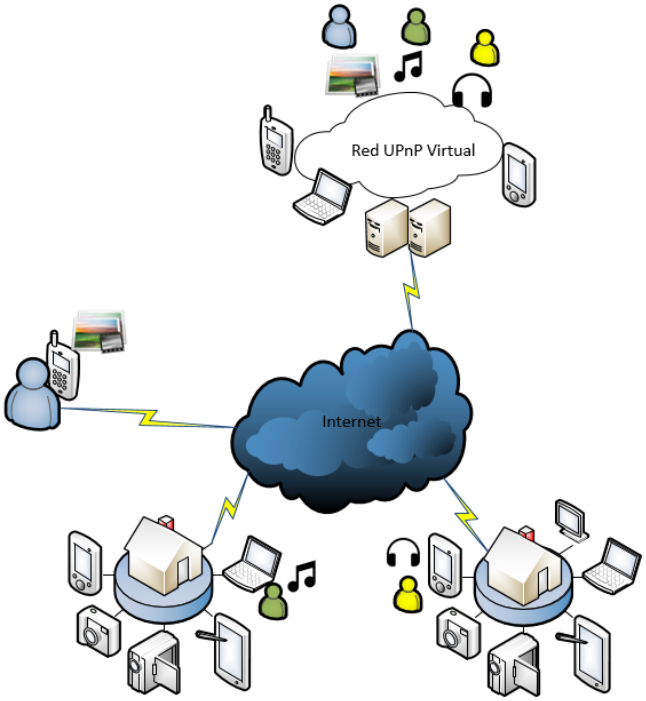

Figura 25: Escenario de aplicación que muestra las redes de virtuales remotas y los dispositivos conectados a ellas. La figura muestra como diferentes usuarios conectados desde redes locales situadas en localizaciones geográficas remotas son capaces de compartir contenidos entre ellos a través de una red overlay propuesta como solución en este capítulo.

a los contenidos almacenados en los dispositivos de otros usuarios siempre que la política de privacidad de cada usuario lo permita.

La VUN a la que se conectan los usuarios permite establecer una política de compartición de contenidos gestionada por las aplicaciones que los ofrecen a modo de overlay sobre internet a nivel UPnP. Los diferentes usuarios pueden acceder a la VUN usando diferentes dispositivos en diferentes localizaciones como se explica en la siguiente sección.

4.3 ARQUITECTURA DE REDES FEDERADAS COMPUESTAS DE DISPOSITIVOS DOMÉSTICOS

La arquitectura propuesta en la Figura 25 permite añadir dispositivos UPnP a una red virtual compuesta por dispositivos UPnP en lugar de un conjunto de redes conectadas como en [183], donde se presenta una solución basada en conexiones VPNs para compartir contenidos entre dispositivos como si todos los dispositivos se encontrasen en la misma red. Sin embargo, las soluciones basadas en redes VPNs presentan una carga adicional en la transferencia de la información debido a necesidad de cifrar y

decodificar la información intercambiada en los dispositivos. Además, no se permite un control granular a nivel de aplicación limitando las opciones de configuración del sistema. Esta solución permite crear la red virtual con solo los dispositivos remotos deseados y no todos los dispositivos que pertenecen a una red local. De esta manera, la privacidad es fácil de manejar por parte de los usuarios, ya que esto abre la opción a nuevas posibilidades tales como poder crear redes dedicadas a temas específicos o grupos de usuarios. Esta red virtual se llama VUN (Red virtual UPnP) y los dispositivos UPnP conectados a esta red funcionan como si estuvieran en una red local interactuando y comunicándose con otros dispositivos que soporten el protocolo UPnP.

Para poder crear esta red es necesario que los dispositivos incorporen una solución cuya arquitectura interna contiene unos pocos servicios adicionales para gestionar las conexiones necesarias. Estos servicios son: el servicio gestión de la conexión, el servicio de gestión de intercambio de contenidos, y el servicio de dispositivo multimedia virtual.

El **Servicio de Gestión de la Conexión** (SC) es responsable de establecer y mantener la comunicación entre los dispositivos de usuarios y la red virtual UPnP o VUN. El **Servicio de Gestión de Intercambio de Contenidos** (SGIC) es responsable de la transferencia de contenido entre dispositivos. Por último, el **Servicio de Dispositivo Multimedia Virtual** controla las comunicaciones UPnP entre dispositivos dentro de la red remota. La Figura 26 muestra la arquitectura de estos servicios.

Como se ha comentado anteriormente, el protocolo UPnP sólo permite la gestión den contenidos entre dispositivos conectados a una misma red local por lo que es necesario aplicar algún tipo de mecanismos para que pueda trabajar con redes remotas.

Esto se consigue a través del Servicio de Gestión de la Conexión que hace uso del protocolo eXtensible Messaging and Prsence Protocol (XMPP) [185] como una capa de comunicación intermedia entre los dispositivos UpnP y la VUN.

XMPP es un protocolo abierto basado en XML y diseñado principalmente para el intercambio de mensajes e información de presencia entre usuarios en tiempo real. XMPP es ideal para este trabajo porque ofrece comunicación y gestión de presencia entre cada dispositivo. Además, debido a que utiliza para el transporte de la información protocolos basados en XML y el protocolo está estandarizado a través de la RFC3920 [186] , dispone de un alto grado de interoperabilidad.

El protocolo XMPP ofrece varias funcionalidades que son aprovechadas para añadir funcionalidades adicionales a la red de dispositivos. XMPP permite crear dominios privados compuesto por ciertos contactos lo que permite crear comunidades privadas. Un dispositivo podría formar parte

de varias comunidades lo que permitiría que un mismo usuario pudiese compartir contenidos en aquellas comunidades de interés, por ejemplo comunidades para amigos, para familiares, etc.

XMPP dispone de mecanismos para poder establecer streaming de video entre dos usuarios. A pesar de que XMPP en sí no está preparado para transmitir vídeo, XMPP permite realizar la negociación para compartir vídeo entre dos usuarios finales. El intercambio de contenidos de vídeo puede ser realizado mediante el uso de arquitecturas: una arquitectura *cliente-servidor-cliente* o *Peer to Peer (P2P)*.

La arquitectura *cliente-servidor-cliente* se caracteriza por el uso de un servidor central en algún de la red de Internet que actúa como caché o repetidor del contenido que se comparte entre dos clientes. En el caso de usar la arquitectura P2P, los clientes utilizan XMPP para intercambiar la información de señalización y control para crear una conexión y posteriormente establecer el streaming del vídeo entre los clientes. Una vez establecida la conexión, el contenido es transferido entre los clientes de manera directa. A continuación se describen más detalladamente el funcionamiento de cada uno de los servicios.

4.3.1 *Servicio de Gestión de la Conexión*

El Servicio de Gestión de la Conexión se conecta a los diferentes dispositivos UPnP vinculados a la VUN a través del servidor XMPP.

El SC es un cliente XMPP y su función principal es superar las limitaciones de los dispositivos UPnP cuando trabaja con dispositivos ubicados en redes remotas. En la solución propuesta, los dispositivos UPnP unidos al VUN son capaces de intercambiar mensajes a través del protocolo XMPP. Los mensajes UPnP están incrustados dentro de los mensajes generados por el protocolo XMPP, por lo tanto, cuando un dispositivo UPnP quiere enviar un mensaje a otro dispositivo, si se encuentra en la misma red local, simplemente envía el mensaje, a través de la interfaz de red correspondiente. Sin embargo, si el dispositivo está en una red remota y ambos dispositivos están vinculados a la VUN, el primer dispositivo entrega el mensaje a su SC, y posteriormente el SC envía ese mensaje UPnP incrustado dentro de un mensaje XMPP hacía el SC del segundo dispositivo. Este último CS recupera el mensaje y lo entrega a la pila UPnP del segundo dispositivo.

XMPP administra la incorporación de nuevos dispositivos. El sistema propuesto permite que cada usuario pueda asignar un alias a cada dispositivo conectado a la red para facilitar su identificación respecto a otros dispositivos conectados a la VUN. Esto no sólo facilita la gestión de los dispositivos por parte de los usuarios sino que también actúa como identificador global (y contextualizado) fuera de la red local para cada VUN. Este

alias, permite a los dispositivos conectarse a otros a través de un servidor XMPP y usar UPnP de forma trasparente.

Cada alias identifica un dispositivo dentro de un dominio XMPP. Por lo tanto, desde un dispositivo se puede buscar y posteriormente conectarse con otro dispositivo tan sólo conociendo el alias del dispositivo deseado. De esta manera, el dispositivo no necesitar saber la dirección IP del otro dispositivo, con tan sólo el alias es posible identificar el dispositivo evitando tener que recordar la dirección IP del dispositivo desde el cual se conecta el contacto. Los mecanismos del protocolo XMPP son los encargados de gestionar los alias de los dispositivos y de establecer las conexiones y la creación de una red de alto nivel formada por dispositivos UPnP.

Los dispositivos de esta red de alto nivel se comunican con otros dispositivos UPnP pertenecientes a esa red de alto nivel de forma transparente. Gracias al mecanismo de gestión de contactos de XMPP, el usuario puede gestionar la lista de dispositivos UPnP y conectarse a ellos independientemente de la topología de red. Esta tarea es equivalente a la tarea de administrar los contactos del usuario en una aplicación de mensajería instantánea (AMI) las cuales permiten crear, eliminar y actualizar contactos. El alias de cada dispositivo podrían ser similar a los nombres de usuario en una AMI facilitando la gestión de los contactos. El alias podría coincidir con el propietario del nombre de usuario del dispositivo para facilitar la gestión.

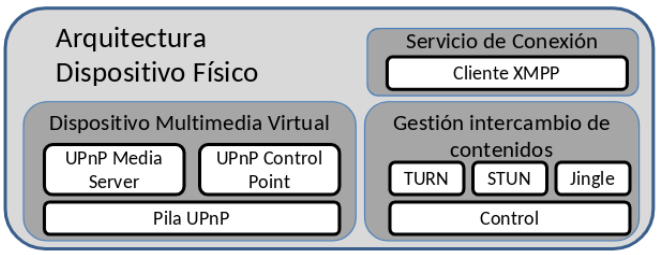

Figura 26: Esquema de bloques donde se muestran los servicios necesarios para que un dispositivo UPnP pueda conectar a la red virtual remota. Estos servicios permiten, a un dispositivo genérico que implemente la pila UPnP, conectar con los servicios UPnP de otros dispositivos conectados también a la red virtual remota.

4.3.2 *Servicio de Gestión de Intercambio de Contenidos*

El Servicio de Gestión de Intercambio de Contenidos (SGIC) es responsable de gestionar la transferencia de contenidos multimedia entre dispositivos. Una vez que el usuario decide visualizar o reproducir un contenido, es necesario transferir el contenido deseado entre los dispositivos implica-

dos. El sistema detecta la red donde se encuentra el contenido. Cuando el contenido se transfiere entre dos dispositivos UPnP en la misma red no hay ningún problema porque no hay elementos de red entre ellos que puedan impedir el intercambio tales como los Network Address Translation [10] (NAT) [187] o firewalls[11] . Sin embargo, si el contenido que se debe transferir entre los dispositivos esta detrás de un NAT, requerirá el uso de mecanismos para permitir la conexión entre ellos ya que los dispositivos dispondrán de una dirección privada que podría afectar a su conexión. Especialmente cuando la conexión se inicia desde Internet a los dispositivos ubicados en una red privada.

SGIC tiene en cuenta estas situaciones y trata de resolverlo resolviéndolo a través de los protocolos STUN (Simple Transversal de UDP sobre NATs) [188] y TURN (Traversal Using Relay NAT)[189]. STUN informa a las aplicaciones sobre los puertos y direcciones públicas cuando alguna aplicación desde internet quiere conectar a la aplicación conectada a una red privada situada detrás de un NAT. Además, el uso de NAT permite a un dispositivo transferir contenidos a otro dispositivo incluso si ambos están dentro de redes locales conectadas a internet a través de NATs.

Esta solución es válida mientras que el NAT no tenga en cuenta ni la dirección IP de origen del paquete ni tampoco el puerto. Sin embargo, en ocasiones se tiene en cuenta la dirección IP de origen, el puerto de origen o ambas cosas, y eso impide establecer el mapeo a través de STUN. Como solución se requiere el uso de un servidor adicional que implemente TURN con direcciones IP públicas resolviendo el problema de no conocer direcciones de origen y puertos de origen conocidos.

El protocolo proporciona un conjunto de direcciones y puertos accesibles desde otras direcciones publicas, que las aplicaciones deben usar para poder conectar garantizando así, junto con la identificación independiente de XMPP, la transparencia localización [190] deseada. Así, cada dispositivo establece una conexión al servidor proxy y se establece el bypass entre los dos dispositivos. De esta manera la comunicación puede ser establecidas entre los dos dispositivos incluso existiendo un NAT entre ambos. La información obtenida por estos protocolo es transferida a los dispositivos a través del protocolo Jingle [191].

Jingle es una extensión del protocolo XMPP que se emplea para inicializar y gestionar sesiones multimedia entre dos dispositivos finales. Jingle usa la infraestructura de XMPP como un canal para la señalización de la

10 Los dispositivos NAT son usados para permitir la conexión entre dispositivos conectados en una red privada con dispositivos conectados a una red pública a través de la traducción de las direcciones IP asignadas en cada segmento de la red.

11 Un firewall es una aplicación o un hardware que controla el acceso a la red y supervisa el flujo de tráfico de red.

gestión de las sesiones multimedia entre dos entidades. Llegados a este punto, una vez conocido el uso que se le da a XMPP en la solución, el lector puede intuir que la combinación de XMPP y Jingle, podría ser similar al uso para el que fue concebido el protocolo Session Initiation Protocol (SIP) [192]. Sin embargo, existe una diferencia radical entre ambos protocolos, que los distingue para el uso que nos ocupa. Mientras SIP proporciona gestión de sesiones multimedia, XMPP proporciona un intercambio de información estructurada, que permite además, utilizando Jingle, establecer sesiones multimedia. Pese a que el uso de XMPP con SIP pueda proporcionar una funcionalidad equivalente [193], la integración entre Jingle y XMPP facilitan su uso y la funcionalidad proporcionada es adecuada para el propósito perseguido.

Una vez que la sesión multimedia ha sido negociada, la transferencia de los contenidos es llevada a cabo a través de otros protocolos más apropiados para este tipo de contenidos tales como RTP (Real-Time Transport Protocol) [194]. Para este trabajo se ha usado las especificaciones XEP-166 [195] la cual pertenece al conjunto de protocolos de la serie XEPs (XMPP Extension Protocolos series) que extienden la funcionalidades de XMPP.

4.3.3 *Servicio de dispositivo multimedia virtual*

El Servicio de Dispositivo Multimedia Virtual (SDMV) se encarga de gestionar las comunicaciones con dispositivos UPnP en la red local y remota. Consta de tres servicios internos donde cada uno de ellos tiene un objetivo específico. Uno es responsable de implementar la pila UPnP (utilizada por otros servicios). Los otros dos bloques implementan la funcionalidad Media Server UPnP (MSU) y el otro la funcionalidad Control Point UPnP (CPUP).

La pila UPnP conecta con la red local y con el VUN a través del SC procesando los mensajes según el estándar UPnP y entrega la información recopilada a las capas superiores (CPUP y MSU). El bloque MSU permite a los usuarios compartir contenido almacenado en el dispositivo local con los dispositivos ubicados en la VUN y en la red local. Estos contenidos pueden tener preferencias asociadas al usuario y se encuentran almacenadas en el MSU. Cuando el dispositivo está activo, la MSU debe obtener primero las preferencias del usuario que indicarán qué contenido se comparte o se oculta. A continuación, el dispositivo se conecta al VUN para anunciar su presencia y el contenido disponible. En ese momento, todos los demás dispositivos conocerían la presencia del nuevo dispositivo UPnP en la red y se indexará en la lista de dispositivos en cada dispositivo UPnP.

El SDMV también implementa la funcionalidad de un CPUP extendido ya que dispone de la capacidad de intercambiar mensajes con dispositivos

UPnP conectados al VUN. Cada vez que un dispositivo es conectado o desconectado a la red, tiene que anunciar su presencia o desconexión. Estos mensajes se propagan a cada dispositivo usando XMPP y el CPUP los recibe a través del SC. De esta manera es consciente de qué dispositivos están conectados al VUN. Además, a través del CP, los usuarios pueden acceder al contenido disponible en los dispositivos conectados a la VUN.

Gracias a la arquitectura de UPnP, el SDMV podría integrar también un reproductor de forma que podrá reproducir contenidos de la VUN y de la red local y redirigirlos a cualquier otro dispositivo UPnP compatible forme parte éste o no de la VUN.

4.4 OPTIMIZACIÓN EN DISPOSITIVOS LIMITADOS

En el apartado anterior se ha mostrado un sistema basado en servicios que hace más accesibles y más seguro el consumo de contenidos digitales personales. La arquitectura esta formada por un conjunto de servicios que conviven en un dispositivos ofreciendo las funcionalidades necesarias para que puede producirse ese intercambio. Open Services Gateway initiative (OSGi) es una de las tecnologías orientadas a servicios más relevantes en el desarrollo de software basado en arquitecturas de servicios. De hecho, uno de los pilares fundamentales del diseño de la tecnología OSGI fue su compatibilidad con Universal Plug and Play (UPnP).

La especificación OSGi [196] define un framework inspirado en arquitecturas orientadas a servicios (SOA) el cual permite gestionar servicios con el objetivo de crear aplicaciones basadas en servicios. OSGi se basa normalmente en Java y cada elemento del framework OSGi se denomina bundle. Un bundle es un paquete de software que puede proporcionar servicios a otros bundles instalados en el framework OSGi. Cada bundle también puede exportar a otras bibliotecas o paquetes de paquetes. Por lo tanto, las aplicaciones OSGi están formadas por recursos coordinados ofrecidos desde diferentes bundles (véase Figura 27). OSGi es una tecnología clave para los proveedores de servicios, ya que les permite supervisar y mantener de forma remota los servicios implementados en los Gateways.

Debido a que la solución propuesta debe ser desplegada sobre dispositivos electrónicos con pocas capacidades de gestión, y que la tecnología OSGi es una tecnología ideal para desplegar software sobre estos dispositivos, en este apartado se propone una solución para optimizar de manera automática los bundles que componen una aplicación.

La solución no sólo es viable para mantener un bajo footprint del software instalado, sino que debido a que estos dispositivos normalmente se encuentran conectados a Internet y que están sujetos a posibles actualizaciones software, el uso de esta solución, permite aplicar mecanismos de

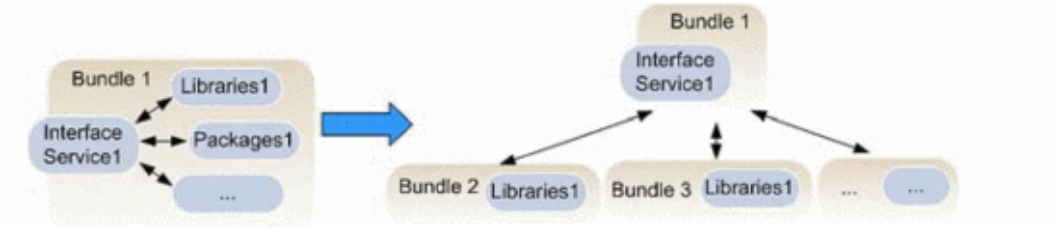

Figura 27: La figura muestra dos posibles opciones de empaquetar las librerías que requiere un servicio OSGi. Las librerías requeridas pueden incluirse dentro de un bundle OSGi o el bundle de servicio 1 puede importar todas las librerías desde otros bundles OSGi configurados para exportar dichas librerías.

optimización de manera automática y autónoma acompañando al proceso de actualización. Ésto, es una propiedad muy valiosa para este tipo de escenarios de aplicación.

Los mayores problemas de las aplicaciones en dispositivos de consumo e IoT son problemas de seguridad en diversos planos y la gestión de una creciente población de dispositivos que generará tráfico sin precedentes, como ya se mencionó. Pese a que muchas de las soluciones de IoT son maduras y la mayoría de las plataformas de dispositivos de consumo están muy probadas, existe un gran problema de seguridad al no existir un consenso técnico [197] ni regulatorio [198] en lo concerniente a la seguridad. El informe de la FTC (Federal Trade Commission) de enero de 2015 [199] respecto a la seguridad en IoT, señalaba la seguridad de los dispositivos, los APIs a los que estos acceden, la autenticación y las actualizaciones de sistema como los mayores problemas a los que se enfrentaba el mercado.

Como se discute en el informe, los dispositivos en IoT pueden tener tamaños, formas y funcionalidades muy diferentes, pero comparten conjunto de atributos diferenciadores de otras tecnologías que requiere especial atención desde el punto de vista de la seguridad. Todos ellos están dotados de capacidad de cómputo más o menos limitada y, por economía de escala, suelen disponer de un sistema operativo y aplicaciones con características similares. Entre estas aplicaciones, multitud de ellas utilizan OSGi para incluir nuevas funcionalidades y servicios. Esto convierte dichos sistemas en susceptibles de ser reprogramados para exceder el propósito inicial del dispositivo. Además, esto puede suceder sin el conocimiento del usuario dado que rara vez disponen de un sistema de monitorización que permita advertir cambios en el dispositivo.

La reutilización de las plataformas de hardware, integrados, drivers e incluso entornos de desarrollo sobre las que se construyan los dispositivos

pueden ocasionar que una vulnerabilidad encontrada o introducida en un dispositivo pueda ser explotada sobre un gran conjunto de dispositivos. Por esta razón, entre otras, el procedimiento de actualización y gestión de las dependencias es clave para evitar problemas de seguridad así como para optimizar el rendimiento de los dispositivos.

Una vez que sobre un dispositivo los bundles han sido instalados, los mecanismos de optimización propuestos procesan los datos recopilados decidiendo cómo reconfigurar el framework para optimizar sus recursos. Para que esto sea posible, la solución propone un modelos de programación el que que el desarrollador debe distribuir las bibliotecas, servicios y paquetes en varios en distintos bundles en su lugar para agrupar estos elementos en el mismo bundle. La figura 1 muestra el modelo presentado.

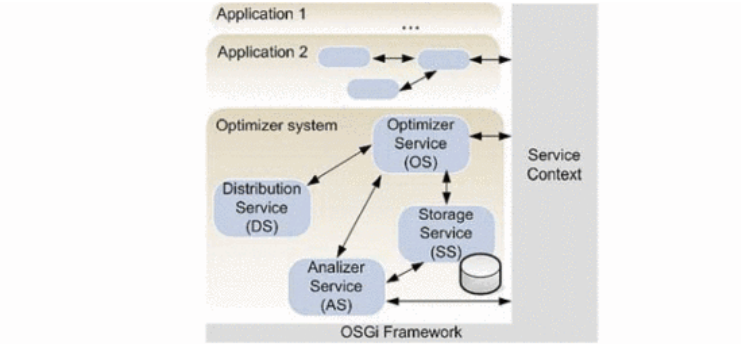

Figura 28: La figura muestra los servicios OSGi y sus relaciones dentro del framework OSGi para trabajar conjuntamente con el objetivo de realizar el proceso de optimización propuesto en este capítulo.

La arquitectura de optimización se compone de cuatro servicios con funciones específicas (véase la Figura 28). El primer servicio es el "Servicio analizador"(AS) que tiene como objetivo detectar los paquetes instalados en el marco OSGI y analizar cada uno de ellos con el fin de averiguar las dependencias, los paquetes y bibliotecas software exportados por ellos y cualquier otro parámetro que se pueda compartir. El AS solicita al framework OSGI la lista de servicios que se ejecutan en ese momento que se obtiene del archivo `manifest.mf` de cada fichero “bundles” que contiene información sobre los bundles individuales.

Este archivo existe en cualquier bundle, ya que se utiliza para registrar los recursos utilizados por el paquete como clases, paquetes, servicios, etc. El AS utiliza el "Servicio de almacenamiento"(SS) para almacenar la información recopilada del sistema. El SS administra esta información que ofrece servicios a otros paquetes como la búsqueda, eliminación y almace-

namiento; su comportamiento es similar a una base de datos.

El "Servicio de Distribución"(DS) se encarga de recuperar la información de sistemas remotos. La información de los sistemas remotos se organiza en una tabla compartida pública (SB) basada en una tabla hash distribuida (DHT) [200] la cual se comparte a través de una red P2P.

Por último, el Servicio de Optimizador (OS) es responsable de tomar la decisión de modificar o mantener los componentes del sistema. Tiene tres modos de operación, en el primer modo intenta encontrar la mejor configuración utilizando sólo la información almacenada por el SS (información local). Este modo optimiza los componentes utilizando componentes que existen en el sistema local (optimización local). En el segundo modo de funcionamiento, el sistema operativo puede seleccionar el DS para localizar los componentes más adecuados en los sistemas remotos. En el tercer modo, el sistema operativo utiliza una combinación de ambos sistemas (AS y DS) para aumentar la calidad de la optimización (optimización global).

Para realizar la optimización, el sistema define una estructura de información para cada componente que se divide en tres secciones: identificación de componentes (hash, nombre, fabricante, firma, versión...), datos de evaluación e identificación del evaluador. Los datos de evaluación contienen una puntuación (S_i) y se comparan con una puntuación de umbral (TS). Hay un TS para cada componente. El TS señala la puntuación mínima que una nueva versión del componente debe lograr para sustituir el componente actual. Cuando el proceso de evaluación implica componentes remotos, se comporta de forma diferente por motivos de seguridad. Para evitar la instalación de componentes peligrosos desde sistemas remotos maliciosos, nuestra solución utiliza la sección de identificación del evaluador, que identifica al autor del bundle publicado mediante mecanismos criptográficos como pueden ser los certificados de clave publica de desarrollador [201]. Esta información se utiliza para obtener una puntuación de reputación (RS). La puntuación de reputación se calcula de la siguiente manera a partir de la información proporcionada por el sistema operativo y el RS:

$$S_{RP} = \sum S_i D(mR, RS)$$

La ecuación obtiene una puntuación (S_{RP}) sumando el valor de las puntuaciones de las diferentes entradas que coinciden con el propósito del bundle buscado. Esas puntuaciones se filtran con la expresión $D(mR, RS)$ las cuales utilizan el valor mínimo de reputación para aceptar una recomendación (mR) y la reputación del recomendador (RS).

Esta función devuelve 1 cuando mR es mayor que RS y 0 de lo con obtiene una puntuación (SRP) sumando el valor de las puntuaciones de las diferentes entradas que coinciden con el propósito del bundle buscado. Esas puntuaciones se filtran con la expresión $D(mR, RS)$ las cuales utilizan el valor mínimo de reputación para aceptar una recomendación (mR) y la reputación del recomendador (RS). Esta función devuelve 1 cuando mR es mayor que RS y 0 de lo contrario. Si SRP es mayor que los umbrales registrados, el optimizador actualizaría o sustituiría el componente.

4.5 CONCLUSIONES

En este capítulo se ha definido una arquitectura que permite extender las redes locales compuestas de dispositivos domésticos del hogar a redes superpuestas 4.3 y publicada en [202]. Además, también se ha propuesto una solución 4.4 y publicada en [203] la cual optimiza el uso de librerías software sobre frameworks OSGi. OSGi es un entorno de ejecución muy extendido en dispositivos con recursos limitados y la solución propuesta para optimizar el software ayuda a mejorar el funcionamiento general de los dispositivos.

Ambas soluciones contribuyen a que dispositivos domésticos con pocos recursos puedan ampliar las capacidades de los protocolos de compartición de contenidos multimedia ampliamente extendidos en la industria tales como UPnP AV y DLNA [204]. La solución propuesta extiende estos protocolos haciendo uso del protocolo XMPP para llevar los mensajes UPnP intercambiados a lugares remotos de internet.

El protocolo XMPP está orientado a servicios de mensajería instantánea entre usuarios. Esta solución utiliza XMPP para crear redes de dispositivos en lugar de usuarios permitiendo que los dispositivos se conecten y desconecten de la red XMPP y permitiéndoles intercambiar contenidos de forma transparente a la localización.

XMPP es ideal para este cometido ya que gracias a las funcionalidades de indicador de presencia, gestión de grupos o la multitud de extensiones disponibles no sólo permite un funcionamiento e integración óptima, si no que también permite añadir nuevas funcionalidades a las ya existentes propias ofrecidas por las redes UPnP.

Debido a que los protocolos de UPnP y DLNA suelen ser implementados sobre dispositivos con bajas capacidades de cómputo, la optimización de paquetes en OSGi propuesta proporciona una solución para reducir los recursos de memoria de estos dispositivos de manera automática en base a unas puntuaciones que permiten minimizar el consumo de memoria y

facilitan las actualizaciones proporcionando un mayor control en términos de seguridad.

5

ARQUITECTURA DE REDES DE DISPOSITIVOS SOBRE REDES VIRTUALIZADAS

Este capitulo presenta una arquitectura más avanzada para facilitar el despliegue de aplicaciones complejas basadas en múltiples servicios que la mostrada en el capítulo 4. Esta arquitectura es el fruto resultante de años de diseño, análisis de necesidades y seguimiento de la evolución de tecnologías relacionadas con la gestión de servicio y sistemas de tecnologías de la información.

Este capítulo, introduce en la sección 5.1 los retos de diseño de la solución, detallándose la arquitectura propuesta en este trabajo en la sección 5.2, cuya validación se realiza en el capítulo 9. Más adelante, en el capítulo 6 se presentan mecanismos para desplegar servicios de manera sencilla para facilitar la configuración para el despliegue de aplicaciones compuestas de múltiples servicios. El capítulo 7 valida la propuesta realizada para las configuraciones de despliegue. El capítulo 8 describe los mecanismos desarrollados en el ámbito de esta tesis para optimizar el despliegue de servicios y componentes software sobre diferentes entornos. Finalmente, el capítulo 9 proporciona una validación general mediante un prototipo.

La arquitectura descrita en este capítulo fue publicada a través del artículo [9] formando parte del proyecto MAGOS [1] y el proyecto CYNAMON [2]. En el capítulo 9 se realiza la validación de la arquitectura propuesta en este capítulo.

5.1 PRINCIPIOS DE DISEÑO

En el capítulo 2 se ha realizado análisis del estado del arte en el que se extrajeron los requisitos necesarios para una arquitectura de despliegue multiproveedor. Considerando esos requisitos que se exponen de nuevo en la tabla 15, este capítulo describe los principios de diseño, la arquitectura propuesta y las limitaciones de este trabajo.

1 Referencia al proyecto MAGOS: TEC2017-84197-C4-1-R
2 Referencia al proyecto CYNAMON: P2018/TCS-4566

Requisitos	Descripción
R1	Mecanismos para gestión de servicios.
R2	Catalogo de servicios.
R3	Soporte Multiplataforma.
R4	Encapsulado de servicios.
R5	Empaquetado de servicios.
R6	Soporte para la configuración de enlaces virtuales.
R7	Softwarización de cualquier tipo de funcionalidad.
R8	Mecanismos de interoperabilidad con otros sistemas.
R9	Gestión de recursos de infraestructura.
R10	Mecanismos para desplegar y configurar software de manera desatendida.
R11	Administración del ciclo de vida de los servicios
R12	Adaptar/configura infraestructura en función de los requisitos de funcionamiento de los servicios.
R13	Gestionar encadenamiento de servicios.
R14	Mecanismos para optimizar el despliegue de servicios.
R15	Curva de aprendizaje moderada
R16	Requisitos de funcionamiento mínimos
R17	Autoconfiguración
R18	Aislamiento
R19	Integración personalizable

Tabla 15: Requisitos para el desarrollo de las futuras infraestructuras de servicios comentados en el capítulo 2

En la Figura 29, se presentan los elementos claves para hacer frente a los desafíos presentados. La identificación de estos elementos claves ha sido importante ya que ha permitido definir los principios de diseño que han servido como guía para elaborar la solución propuesta. A continuación se enumeran los principios de diseño:

1. **Dispositivos virtualizados** La virtualización de dispositivos permite agregar capacidades adicionales a dispositivos simples que reducen los gastos de capital (CAPEX). Hay dos casos a considerar, dispositivos puramente virtualizados, que no tienen, por tanto, requisitos de hardware y dispositivos cuyo hardware es limitado que pueden incorporar nuevas funcionalidades utilizando la cloud para instanciar los servicios de los que la nueva funcionalidad dependa sin exigir un hardware más complejo en el dispositivo original.

2. Hardware y software no propietarios. La solución propuesta se puede implementar en cualquier entorno virtualizado evitando:
   a) problemas de lock-in de proveedor: al permitir el uso de diferentes proveedores, se mitiga el problema de dependencia del proveedor.
   b) uso de firmware completamente propietario en dispositivos: recurriendo al uso de servicios instanciados en la cloud, es posible minimizar la particularización del firmware de dispositivo, requiriendo únicamente un firmware básico que permita, entre otros, el acceso a la cloud donde se encontraría la complejidad del servicio.
   c) actualizaciones limitadas: al reducir la complejidad del dispositivo en varias dimensiones (tanto hardware como software), se reduce el software susceptible de ser actualizado, reduciendo el número y complejidad de las actualizaciones.
   d) seguridad incrementada: como consecuencia de la simplificación del dispositivo y de la reducción del software susceptible de ser actualizado, se consigue reducir la superficie de ataque del dispositivo y se incrementa la capacidad de corrección de vulnerabilidades al ser el software objetivo más sencillo.

3. **Mecanismos de conexión dinámicos**. Utiliza mecanismos de red definidos por software y discutidos en la sección 2.3.2 que permiten implementar, configurar y controlar redes dinámicamente. Esto permite adaptar la red según el tráfico y la carga de los nodos.

4. **Infraestructura definida por código**. La especificación de la infraestructura necesaria para el despliegue de servicios a través de un len-

guaje de infraestructura, discutida en la sección 3, permite implementar piezas de software de forma independiente en la plataforma y garantizar el funcionamiento tras su despliegue, independientemente del entorno donde se ejecute.

5. **Orquestación para múltiples sistemas**. La aplicación desarrollada a través de microservicios, módulos y componentes de software permite componer una multitud de servicios a través de la orquestación de las mismas. Dichas herramientas de orquestación se discuten en la sección 3.3.1.

6. **Uso de APIs estándar**. El uso de interfaces bien definidas facilita la interconexión con sistemas de terceros. Por lo tanto, también reduce el tiempo de comercialización, mejorando la interoperabilidad y facilitando la comprensión de los sistemas.

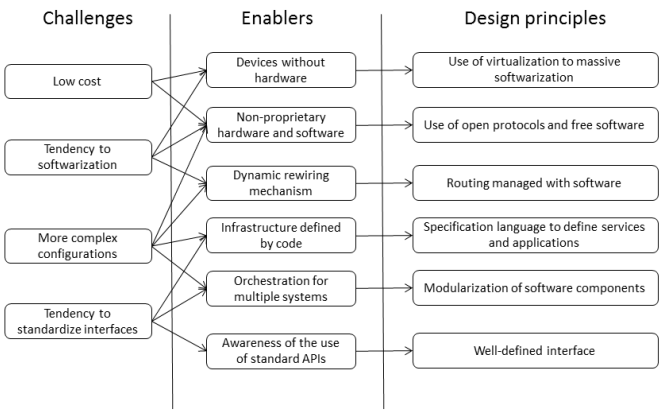

Figura 29: Desafíos y principios de diseño. Adaptado de [205].

5.2 DISEÑO DE LA ARQUITECTURA

La Figura 32 muestra la arquitectura principal propuesta en esta tesis. La arquitectura permite que la solución diseñada sea compatible con la arquitectura NFV [206] y con los componentes de NFV Management and Orchestration (NFV MANO) [66]. La solución se divide en dos bloques principales:

- **Infrastructura de Orquestación e Implementación** (INFrastructure Orchestration and Deployment system, INFORD): propuesta de un entorno virtualizado y flexible con el objetivo de implementar un ecosistema de servicios orientado a satisfacer los requisitos anteriormente mencionados, que se discute en la sección 5.2.1

- Sistema de Supervisión y Gestión de Infraestructura (Infrastructure Monitoring and Management system, IMOM): framework que ofrece las herramientas necesarias para definir, construir e implementar infraestructuras de servicio basadas en la arquitectura INFORD, que se discute en la sección 5.2.2

5.2.1 *Infrastructure Orchestration and Deployment System (INFORD)*

La mayoría de las plataformas multimedia se caracterizan por proporcionar contenidos y servicios digitales a sus clientes. Por lo general estas plataformas dan acceso a sus suscriptores a un conjunto de servicios básicos que pueden ser extendidos, es decir, a través de un modelo de subscripción compuesto por un paquete de servicios básico y otros paquetes, los suscriptores pueden configurar un sistema a la carta.

Los clientes tan sólo tienen que contactar con el proveedor de servicios para seleccionar el paquete de servicios deseado y contratarlo para poder disfrutar de los servicios de su compra. Esta manera de funcionar se corresponde a un modelo tradicional ampliamente extendido y que actualmente adoptan muchos proveedores de servicios. Sin embargo, en los últimos años se ha introducido el concepto Marketplace el cual permite a los clientes acceder a un catalogo de servicios disponibles en la plataforma listos para ser activados y posteriormente consumidos por los clientes. La integración de los marketplaces en las nuevas plataformas de servicios facilita la gestión de los servicios contratados por los clientes ya que hacerlo de la manera tradicional, a través de paquetes de servicios, estaría expuesto a tener un gran número de paquetes de servicios complicando su gestión y haciendo más difícil decidir el paquete a contratar por parte de los clientes. La integración de los marketplaces dentro de las nuevas plataformas de servicios no es el único cambio que las diferencian de los modelos tradicionales, en los últimos años la distribución de los centros de datos en el borde de la red ha supuesto un avance en la optimización de los sistemas software en muchos aspectos, algunos relacionados con la adaptación de los servicios a las circunstancias de la red y otros relacionados al contexto de la localización geográfica de los consumidores.

Para abordar todos estos aspectos relevantes, en esta tesis proponemos la infraestructura INFORD la cual define un entorno virtualizado y flexible cuyo objetivo es implementar un ecosistema de servicios orientado a satisfacer los requisitos de los SPs enumerados en la tabla 1.

INFORD tiene dos capas. La capa inferior, denominada Virtualized Network Manager (VNM), es responsable de proporcionar los recursos necesarios para crear un ecosistema virtual en la nube compuesto por dispositivos virtuales de consumo y servicios. Este ecosistema virtual formado por dispositivos virtuales conectados a una red virtual se denomina Red de usuario virtual (Virtual User Network, VUN). Los dispositivos virtuales son utilizados por los consumidores domésticos para acceder a los servicios disponibles de la red VUN a través de sencillos dispositivos físicos. Estos dispositivos físicos conectan a los dispositivos virtuales extendiendo las capacidades funcionales de los dispositivos físicos. Los servicios de la VUN también están accesibles a los dispositivos que quieran conectarse directamente a la red VUN.

Como se puede ver en la Figura 30 que muestra un ejemplo representativo de VUN, el componente principal de la VUN es el gestor de red local (Local network manager, LNM). Su función es gestionar las conexiones entre dispositivos conectados al VUN. LNM proporciona un servicio de enrutamiento de red con un plano de control encargado de gestionar un componente de red definida por software (SDN). Cuando se añade un nuevo dispositivo o servicio a la VUN tan sólo hay que modificar la configuración del LNM para que el nuevo dispositivo o servicio tenga conectividad dentro de la red. Además, LNM podría ser utilizado para implementar un control de acceso sencillo limitando el acceso a un servicio o dispositivo a través de políticas de red, es decir, un dispositivo o aplicación podría ser accedido solamente desde otro dispositivo o incluso todo lo contrario. Un dispositivo o servicio podría ser accedido solamente desde una ubicación de red externa a la VUN.

La VUN permite además crear instancias de varios dispositivos virtuales como cortafuegos (Firewalls), puertas de enlace residenciales (Gateways), un dashboard web para la gestión de la VUN por parte los usuarios domésticos y almacenamiento conectado a la red (NAS) entre otros componentes. Algunos de estos dispositivos se muestran en el ejemplo de Figura 30 junto a otros dispositivos domésticos tales como STB (que representaría a un Set Top Box), Dev1 y Dev2 (que corresponden a dispositivos genéricos). La VUN permite que todos los dispositivos de consumo, como tabletas, teléfonos inteligentes, Smart TVs. . . sean capaces de interactuar con servicios y dispositivos virtuales alojados en la VUN y por lo tanto sin importar su localización o naturaleza física o virtual.

La VUN está conectada a la red física a través de un componente de canal de acceso privado (Private Access Channel, PAC). El PAC es canal virtual conformado por un conjunto de reglas SDN cuyo porpósito es tunelizar el tráfico de la red a la que tienen acceso los dispositivos de usuario al VNM.

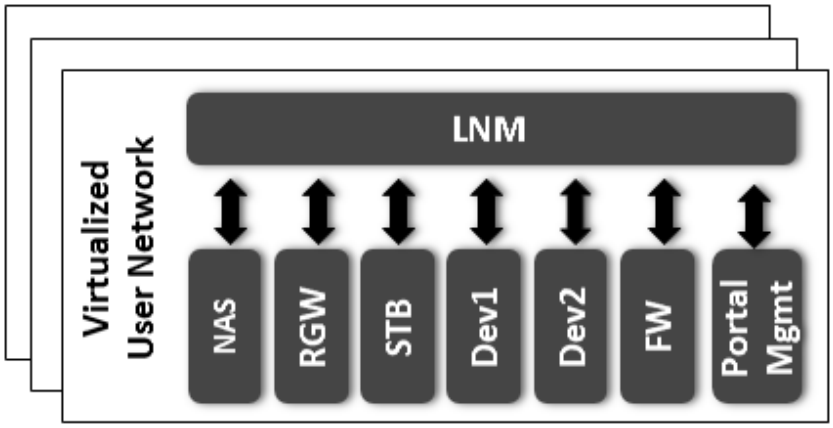

Figura 30: La figura muestra la arquitectura de la VUN compuesta por varios componentes software. El componente LNM corresponde al gestor de red local (LNM), los componentes Dev1 y Dev2 representa dispositivos smartphones virtualizados, y los bloques: NAS, RGW, STB representa dispositivos multimedia desplegados en la VUN a través de su virtualización. Portal Mgmt representa una interfaz web a la que los usuarios conectan para aplicar cambios de configuración.

Si un dispositivo de usuario cambia su red o se agrega un nuevo dispositivo de consumidor al VUN, se crea un nuevo PAC.

La capa superior de INFORD es la capa Administración de aplicaciones (Application Management, AP). Esta capa proporciona componentes de software al VUN (como dispositivos virtuales o servicios) para que se cree una instancia en la red y puedan ser utilizados por los clientes. AP puede estar instalada sobre la misma infraestructura INFORD o sobre otra infraestructura remota conectada a la capa VNM tal como se muestra en la Figura 31.

El Servicio de Aprovisionamiento de Aplicaciones (Application Provisioning Service, APS) es responsable de proporcionar un catálogo de servicios y un motor de búsqueda de aplicaciones, basado en un motor de búsqueda semántico adaptado de [207], el cual permite realizar búsquedas a los usuarios consumidores con el objetivo de seleccionar servicios que puedan ser instalados sobre la VUN. El APS no sólo proporciona el software que se instalará en los dispositivos de consumo, sino también los dispositivos y servicios virtuales. Para gestionar el despliegue y mantenimiento de servicios y dispositivos virtuales el APS conecta con los servicios Device Application Catalogue Service (DACS) y el Cloud Application Catalogue Service (CACS). A continuación se describen estos servicios:

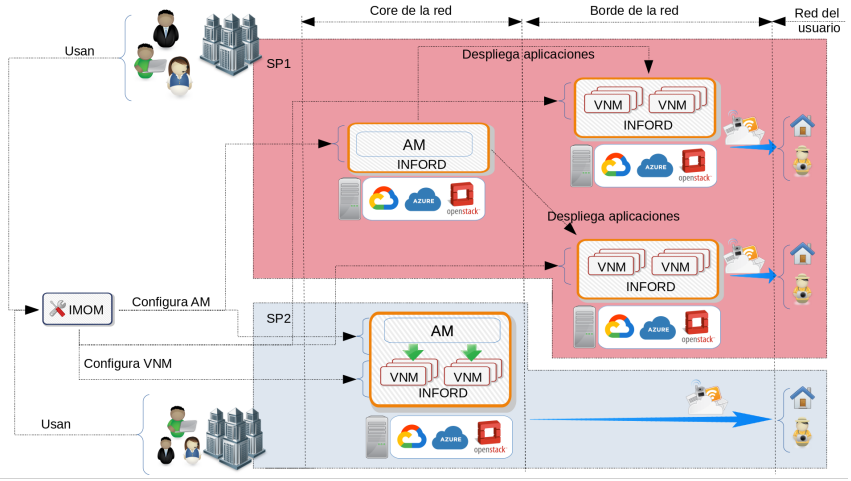

Figura 31: La figura muestra la relación de IMOM e INFORD en un contexto real. Además muestra las interacciones entre los diferentes componentes software que componen cada sistema y los diferentes roles de usuarios implicados en su funcionamiento.

- El DACS gestiona los dispositivos virtuales que complementarán o sustituirán a los dispositivos físicos. Estos dispositivos son instalados en la VUN y son el medio de acceso de los usuarios domésticos para que puedan acceder y consumir servicios desplegados del sistema.
- El CACS se encarga de gestionar servicios que proporcionan funcionalidades como almacenamiento, transcodificación, servicios DLNA, etc. Estos servicios normalmente serán gestionados por los proveedores de servicios. Por ejemplo, en el ámbito de los servicios de distribución de contenidos multimedia, los SPs podrán ofrecer sus servicios a través del CACS y ofrecer paquetes de servicios relacionados con ofertas de contratación. Estos servicios serán consumidos por los dispositivos virtuales desplegados en la VUN y por los dispositivos físicos conectados a la VUN.

Para entender mejor la relación de todos estos componentes pongo un ejemplo de un escenario básico donde un cliente contrata con un proveedor de servicios (SP) un paquete de servicios basado en la VUN. El SP aprovisiona los recursos necesarios sobre INFORD para posteriormente desplegar los servicios y software correspondiente a la VUN. El paquete de servicios basado en la VUN contiene inicialmente un número determinado de servicios y funcionalidades común a todos clientes. Una vez terminado el aprovisionamiento, el cliente puede acceder a los servicios proporcionados por el SP. Entre los servicios más destacados, el cliente tiene acceso al Servicio de Aprovisionamiento de Aplicaciones (Application Provisioning Service, APS) el cual, de manera similar a un Marketplace, le permite reconfigurar

su VUN añadiendo nuevos servicios o dispositivos e incluso eliminarlos de la red. El cliente puede acceder al APS a través del componente Portal Mgmt mostrado en la Figura 30 el cual proporciona una interfaz web que muestra las diferentes opciones de configuración disponibles. El Portal Mgmt está compuesto por un servidor de aplicaciones [3] que permite ejecutar servicios de aplicación totalmente independientes entre ellos. El servidor de aplicaciones conecta con el APS y éste se encarga de desarrollar la lista de servicios y dispositivos virtuales disponibles para ser instalados. Para obtener la lista del software disponible el APS conecta con el DACS y el CACS para que cada uno de ellos consulten sus repositorios y elaboren dicha lista. La ventaja de utilizar un servidor de aplicaciones en el componente Portal Mgmt son los beneficios en su gestión, al estar compuesto por servicios independientes permite añadir o actualizar servicios de manera sencilla. Por ejemplo, permite gestionar (añadir, actualizar, eliminar, etc.) un servicio de búsqueda de servicios, o un servicio dispositivos virtuales a instalar, de manera independiente a los otros que ya se encuentran desplegados en el componente Portal Mgmt.

En el caso de que un cliente desee instalar un servicio no instalado en la VUN, el cliente accede a la VUN y conecta con el Portal Mgmt para buscar y seleccionar el servicio deseado, entonces el servicio encargado de gestionar el servicio de instalaciones (instalado en el servidor de aplicaciones dentro del componente Portal Mgmt) conecta con el APS a través de una interfaz REST indicando la acción de instalar el software indicado por el cliente. El APS conecta con el DACS y este obtiene el paquete de software y la configuración necesaria para poder instalar este servicio sobre la VUN. Dependiendo de las implementaciones registrada en el repositorio del DACS, el servicio deseado podría ser instanciado a partir de archivos de configuración basados en el paradigma de la IaC. Esto permitiría hacer uso de las HGC vistas en el capítulo 3.

Finalmente, el Servicio de Control y Monitorización (Control and Monitoring Service, CMS) recopila información referente a las instalaciones registrando errores en la instalación, número de instalaciones realizadas por servicio y estadísticas de uso. La información recopilada es utilizada para elaborar reportes de uso y análisis estadísticos que ayudan a tener una mejor visión global sobre el funcionamiento del sistema.

3 Un servidor de aplicaciones es llamado a aquellos servidores software capaces de gestionar funciones de lógica de negociación y de acceso a los datos de las aplicaciones facilitando la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

5.2.2 *Infrastructure Monitoring and Management system (IMOM)*

IMOM es un framework que ofrece a los SPs las herramientas necesarias para definir, construir e implementar infraestructuras de servicio basadas en la arquitectura INFORD. La arquitectura de INFORD y su relación con IMOM se muestran en la Figura 32 donde se muestran las capas que componen IMOM.

Estas capas se organizan en componentes y servicios software que permiten a los SPs materializar sus modelos de negocio en forma de servicios interconectados que se implementan mediante INFORD. La capa superior de IMOM, Business Support Services (BSS), discutida en la sección 3.3.2 como parte del estándar MANO [129], persigue la monetización de la plataforma a través de la información almacenada acerca de los servicios disponibles en el sistema, además de el catálogo de servicios disponibles en la plataforma y la información relacionada con las transacciones realizadas.

IMOM puede ser utilizado por los SPs para configurar su propia infraestructura y, a continuación, desplegar servicios. Además de los proveedores de servicios, los fabricantes de software y los administradores también pueden acceder a IMOM para implementar y configurar software a través de la capa BSS. Cada uno de los perfiles de usuario tiene acceso limitado a la plataforma en función del rol asignado al usuario.

La capa BSS está formada por varios servicios con funciones concretas a las que sólo los usuarios con el rol adecuado pueden acceder. Los servicios que componen BSS son:

- El **Servicio de Administración de Usuarios**. Este servicio está ubicado en el bloque gestión de clientes (Customer Management, CM). Administra el acceso de los usuarios a los servicios IMOM. El servicio CM proporciona un panel de acceso web común a todos los usuarios. Cuando el sistema autentica a un usuario, se muestra un panel con las acciones disponibles que puede realizar el usuario en función de su rol.

- El **Componente Gestión de Pedidos** (Order Management, OM). El servicio es responsable de gestionar pedidos realizados por los clientes. Un pedido está compuesto por un conjunto de servicios (puede que algunos de ellos interconectados añadiendo nuevas funcionalidades al sistema), los pedidos se materializan mediante una secuencia de descomposición y orquestación de tareas que dan lugar a la creación de instancias y configuración de servicios y componentes software sobre uno o varios sistemas INFORD.

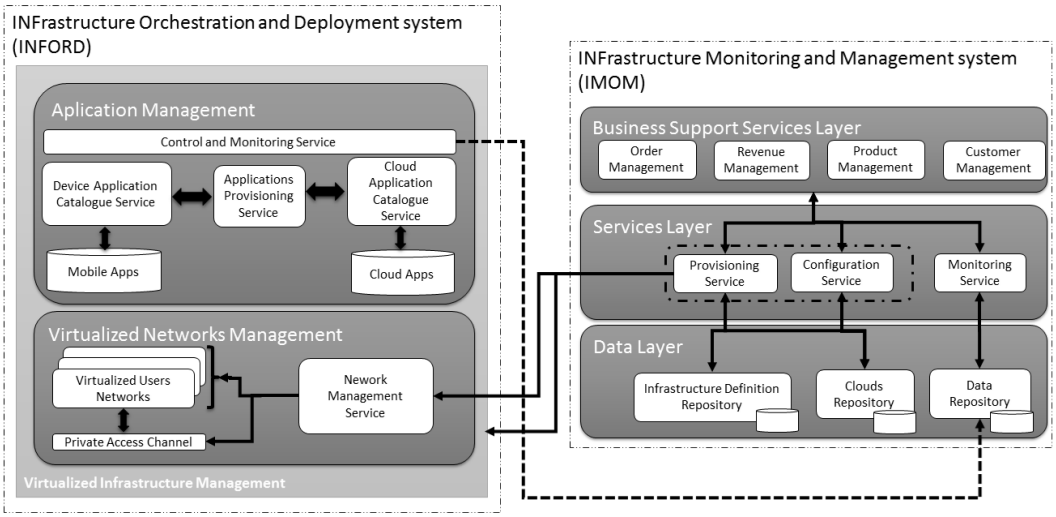

Figura 32: Esta figura muestra los dos bloques principales de arquitectura presentados en este artículo y la relación entre ellos. IMOM proporciona mecanismos para especificar e implementar servicios sobre INFORD. INFORD proporciona un entorno de software multiplataforma donde implementar servicios definidos en IMOM.

- El **Componente Gestor de Productos** (Product Manager, PM). Es el servicio responsable de la gestión de productos desde un punto de vista comercial.
- El **Componente Gestión de Ingresos** (Revenue Management, RM). Este servicio se encarga del cobro y facturación.

Una vez que los SPs han definido las configuraciones deseadas en la capa BSS, el siguiente paso es desplegar e instanciar dichas configuraciones en una infraestructura INFORD. El proceso de para poder desplegar dichas configuraciones es llevado a cabo a través de la **Capa de Servicios** (SL) la cual recibe las especificaciones de alto nivel desde la capa BSS. SL interpreta las especificaciones de alto nivel y las traduce en archivos de configuración y de despliegue que pueden ser interpretados por los componentes de INFORD. SL se compone de los siguientes servicios:

- **Servicio de aprovisionamiento** (Provision Service, PS). Este servicio se encarga de desplegar los servicios sobre la infraestructura INFORD una vez que se ha generado la configuración de cada uno de los servicios.
- **Servicio de configuración** (Configuration Service, CS). Este servicio se encarga de generar la configuración correspondiente de cada servicio a ser desplegado en función de los parámetros introducidos por el SP.

- **Servicio de monitorización** (Monitoring Service, MS). El MS mide la calidad de los servicios ofrecidos por el CS. El MS recibe información del repositorio de datos y genera estadísticas e informes.

PS y CS trabajan **juntos** realizando tareas de orquestación realizando tareas de despliegue de componentes software, configuraciones y adaptaciones de entornos de ejecución, conexión, encadenamiento de cadenas de servicios además de composición de servicios básicos a partir de plantillas de configuración. Además, son capaces de aplicar dichos cambios en las infraestructuras INFORD a través del componente Network Manager Service[4] integrado en la infraestructura INFORD el cual permite la interacción a través de mecanismos basados en API REST y a través de terminales remotos. El trabajo de PS y CS es fundamental porque es responsable del correcto funcionamiento de los servicios desplegados. El orquestador tiene en cuenta factores tales como: el ciclo de vida de cada servicio, las características de cada componente software y la configuración adecuada para múltiples plataformas en las que se puede implementar el servicio. A partir de esta información, el orquestador genera el código fuente con un formato entendible por las Herramientas de Gestión de la Configuración (HGC) y de esta manera toda la configuración de instalación, despliegue y configuración queda empaquetada en archivos de texto, con un formato entendible por los técnicos, que pueden ser interpretados por las HGC encargadas de aplicar los cambios indicados en dichos archivos.

Por último, la capa inferior de IMOM es la **Capa de Datos** (Data Layer, DL) la cual proporciona tres repositorios independientes en los que se pueden almacenar datos de diferente naturaleza.

- Repositorio de infraestructura de definición (Repository Definition Infraestructura, RDI). Es el encargado de almacenar la información relativa a los elementos y servicios de software.
- Repositorio de clouds (Cloud repository, CR). Se encarga de almacenar el inventario de las infraestructuras INFORD registradas.
- Repositorio de datos (Data Repository, DR). Su función es almacenar información histórica sobre el funcionamiento de cada servicio instanciado.

5.2.3 *Interacción entre IMOM e INFORD*

La solución propuesta es el punto de unión que integra a diferentes participantes en el ecosistema de servicios digitales que existen actualmente.

4 El NMS se encarga de gestionar la infraestructura software desplegada sobre INFORD y esta basada en una HGC. Se define en la sección 7.1

Cada uno de los participantes tiene un rol específico e interactúa con la plataforma de una manera diferente, véase la Figura 33. La capa BSS es el punto de entrada común para diferentes usuarios IMOM. La capa BSS consta de un conjunto de servicios segmentados por perfiles de usuario. Cada perfil de usuario se define a partir de las operaciones que pueden realizar. Los usuarios consumidores son los usuarios que conectan a la plataforma para consultar el catálogo de servicios disponibles y contratar aquellos que sean de su interés.

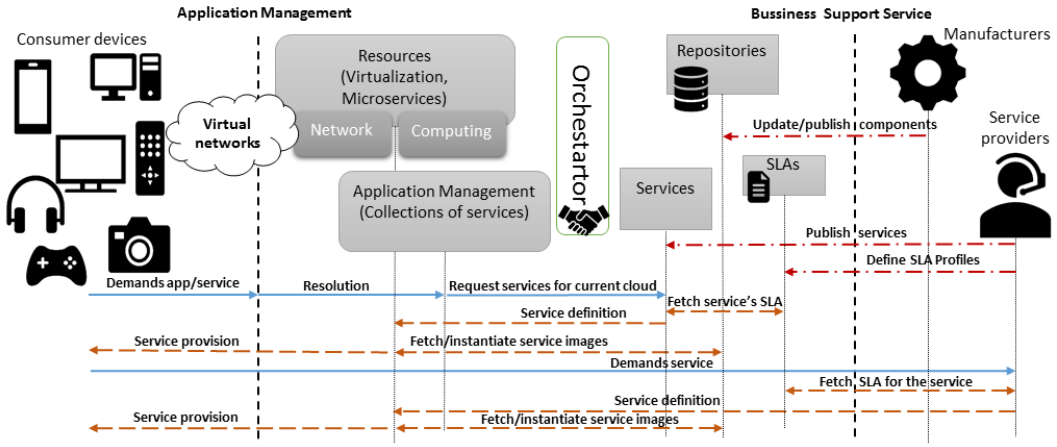

Figura 33: Rol de los diferentes componentes de IMOM e INFORD durante la instanciación y petición de una aplicación o servicio. La figura muestra la secuencia de los mensajes intercambiados entre los principales componentes de la arquitectura propuesta.

Una vez que se ha especificado la estructura y servicios que se van a desplegar, por cada usuario consumidor de servicios que vaya ha contratar los servicios del SP, INFORD creará una instancia del VUN para cada usuario. Un usuario puede añadir dispositivos/servicios a su VUN gracias al Servicio de Aprovisionamiento de Aplicaciones (APS) la cual creará automáticamente las instancias y se configurarán.

Además, APS permite la resolución automática de dependencias un alto nivel. En el caso en que un dispositivo necesite acceder a una funcionalidad proporcionada por otro dispositivo que no se encuentre instalado en la VUN, el dispositivo puede pedir el despliegue desatendido de aquel dispositivo para cumplir su propósito a través del APS. Para esto, el dispositivo envía una solicitud a APS, a través de una interfaz de administración de aplicaciones donde la solicitud contiene una descripción de las necesidades y APS instalará el dispositivo solicitado. Después de eso, el dispositivo podrá acceder a la funcionalidad contenida desde el dispositivo instalado

recientemente en la VUN.

Un caso similar podría ocurrir en el caso de que un dispositivo tratase de acceder a un servicio anunciado por un servicio contenido en el catalogo de un paquete de servicios desde un SP. En el caso de que aquel servicio no se encuentre disponible en la VUN, APS tiene la capacidad de recuperar aquel servicio desde el SP e instalarlo en la VUN. APS también podría ajustar parámetros de configuración en la VUN para que las peticiones relacionadas con el servicio deseado sean redireccionadas a una infraestructura exterior que provea de la funcionalidad solicitada.

Las operaciones relacionadas con la publicación y el mantenimiento de repositorios de componentes de software son responsabilidad de los desarrolladores. Los desarrolladores se conectarán a la plataforma a través de la capa BSS y solo podrá interactuar con los servicios utilizados para publicar y actualizar componentes. Del mismo modo, los SP solo podrán acceder a los componentes de software y las acciones permitidas (publicar y desplegar servicios sobre infraestructuras en INFORD).

La publicación o actualización de componentes software en la plataforma requiere que los desarrolladores proporcionen información complementaria para especificar y definir las características de cada componente. Este proceso es necesario para las tareas de orquestación y para administrar el catálogo de servicios. En el capítulo 6 se presenta los modelos de datos y aspectos a considerar a la hora de proporcionar la información necesaria a tener en cuenta para cada servicio y componente software a registrar en el sistema.

En el caso de los SPs, el proceso es similar para la publicación y configuración de servicios. Cuando un SP desea implementar un servicio, realiza una solicitud al orquestador indicando los servicios que se van a implementar. El orquestador solicita información de los repositorios sobre la definición de cada servicio y las características técnicas de la plataforma INFORD donde se implementarán. A continuación, genera las configuraciones necesarias para llevarlas a cabo. El orquestador propuesto en este trabajo trabaja muy diferente a los orquestadores analizados en la sección 3 los cuales basan su funcionamiento en imágenes preconfiguradas de dispositivos o sistemas operativos que se despliegan sobre infraestructuras. De hecho, el modelo de referencia para la gestión y orquestación de la ETSI MANO-NFV propone el uso de repositorios específicos para almacenar aquellas imágenes software ya preparadas para ser desplegadas en alguna infraestructura compatible.

En este trabajo se ha tratado de ir más allá y se ha propuesto un modelo de orquestador capaz de generar dichas imágenes a partir de la información proporcionada por el SP desde la capa BSS.

El modelo de orquestador propuesto requiere disponer de la información necesaria para poder desplegar dicho software, esta información se organiza según el modelo de datos mostrado en el capítulo 7. A partir de esta información el orquestador y haciendo uso de las Herramientas de Gestión de Configuración (HGC), el orquestador es capaz de generar y desplegar software sobre múltiples infraestructuras. El modelo de orquestador propuesto se describe en profundidad a través de la implementación presentada en 7. El modelo de orquestador propuesto hace uso de un modelo de datos para recopilar la información referente a los servicios y componentes software de manera estructurada, en 7 se describe en profundidad su implementación.

Las capacidades del orquestador junto con la arquitectura propuesta permite cubrir los requisitos presentados en el capítulo 2.1.6 debido a que solamente el orquestador es capaz de proveer servicios y software empaquetado y encapsulado a medida, facilita la softwarización de nuevas funcionalidades, proporciona soporte multiplataforma, etc. En la tabla 16 se muestra los requisitos satisfechos por la solución propuesta.

5.3 CONCLUSIONES

En este capítulo se ha definido una arquitectura de servicios más avanzada que la mostrada en el capítulo 4 la cual presenta un diseño de red local del hogar compuesta por dispositivos domésticos a los que los usuarios pueden acceder independientemente de la localización geográfica. Además, dicha red puede ser configurada sobre múltiples infraestructuras independientemente del proveedor y la tecnología utilizada debido a que para el diseño se ha tomado muy en cuenta el uso de las tecnologías de virtualización que facilitan la interoperabilidad e importación y exportación de los servicios virtualizados alojados en dichas redes.

Sin embargo, la arquitectura mostrada en este capítulo es más avanzada y compleja la cual toma en cuenta más aspectos de relevancia los cuales están indicados en los requisitos mostrados en 2.1.6. Para hacer frente al desafío de satisfacer todos estos requisitos ha sido necesario proporcionar una arquitectura dividida en dos bloques principales IMOM e INFORD entre los que se distribuyen las funcionalidades. INFORD representa el bloque con la arquitectura de infraestructura que permite desplegar y funcionar los

Requisitos	**Descripción**	
R1	Mecanismos para gestión de servicios.	✓
R2	Catalogo de servicios.	✓
R3	Soporte Multiplataforma.	✓
R4	Encapsulado de servicios.	✓
R5	Empaquetado de servicios.	✓
R6	Soporte para la configuración de enlaces virtuales.	✓
R7	Softwarización de cualquier tipo de funcionalidad.	✓
R8	Mecanismos de interoperabilidad con otros sistemas.	✓
R9	Gestión de recursos de infraestructura.	✓
R10	Mecanismos para desplegar y configurar software de manera desatendida.	✓
R11	Administración del ciclo de vida de los servicios.	✓
R12	Adaptar/configura infraestructura en función de los requisitos de funcionamiento de los servicios.	✓
R13	Gestionar encadenamiento de servicios.	✓
R14	Mecanismos para optimizar el despliegue de servicios.	✓
R15	Curva de aprendizaje moderada.	✓
R16	Requisitos de funcionamiento mínimos.	✓
R17	Autoconfiguración.	✓
R18	Aislamiento.	✓
R19	Integración personalizable.	✓

Tabla 16: Requisitos satisfechos por la arquitectura IMOM e INFORD.

servicios multimedia. IMOM contiene las herramientas necesarias para definir el funcionamiento y servicios que se van a desplegar sobre INFORD. IMOM permite definir la configuración deseada y más adecuada para el despliegue sobre INFORD.

En conclusión, la arquitectura propuesta satisface los principios de diseño enumerados al inicio del capítulo y todo ello a través de una arquitectura modular basada en servicios software capaz de adaptarse a nuevas necesidades en caso de ser necesario.

6

MECANISMOS AVANZADOS DE DESPLIEGUE DE SOFTWARE MULTICAPA

En el capítulo 5 se ha presentado una arquitectura de servicios que da soporte y facilidades a los proveedores de servicios cuando estos ofrecen sus servicios. Dicha arquitectura propone un conjunto de servicios y mecanismos que facilita a los proveedores de servicios la gestión y despliegue de sus servicios. Esta arquitectura ha sido dividido en dos grandes bloques: el bloque INFORD el cual se compone de componentes que son desplegados en infraestructuras donde se instalarán los servicios de lo SPs y el bloque IMOM que proporciona el conjunto de servicios y mecanismos para facilitar la gestión de las infraestructuras remotas además de los servicios que se desplegarán sobre ellos.

En este capítulo se va a presentar los mecanismos que facilitan el despliegue de software y servicios haciendo uso de herramientas de Gestión de la Configuración (HGC) que fuera presentadas en el capítulo 3. Los mecanismos mostrados en este capítulo fueron presentados como parte de un artículo [9]. Parte del desarrollo de las ideas mostradas en este capítulo son una evolución de los trabajos realizado en el proyecto OSAMI[1].

6.1 CONFIGURACIÓN DE INFRAESTRUCTURAS POR CAPAS DE SOFTWARE

La arquitectura propuesta en el capítulo 5 facilita la gestión de servicios a los SPs. Dicha arquitectura se divide en dos grandes bloques: INFORD 5.2.1 e IMOM 5.2.2. El bloque IMOM es el encargado de gestionar los servicios que van a ser desplegados en infraestructuras remotas haciendo uso de herramientas diseñadas para la gestión del software. Algunas de estas herramientas fueron analizadas en el capítulo 3. Gestionar todo el software de una infraestructura de servicios es complicado debido a que implica gestionar software diseñado para objetivos muy diferentes y que trabajan a niveles software muy distintos. Es decir, algunas herramientas trabajan en configuraciones a nivel de aplicación otras herramientas están diseñadas para trabajar a nivel de sistema operativo y otras se centran en la gestión de infraestructura. Para hacer posible la gestión completa de todos los niveles de configuración software, en esta tesis se propone un modelo segmentado

1 OSAMI-Commons Open Source Infrastructure Ambient Inteligence Commons (TSI-020400-2009-92) (Proy. Internacional)

por capas de software para facilitar su gestión y se han identificado cuatro capas (figura 34): Capa de red de dominio, Capa de clúster, Capa de nodo y Capa de host. Cada capa contiene software especifico para la capa sobre la que se va a desplegar. Dicho software es independiente de otros componentes que puedan estar situados en otras capas. A continuación se presenta una breve descripción de cada una de las capas:

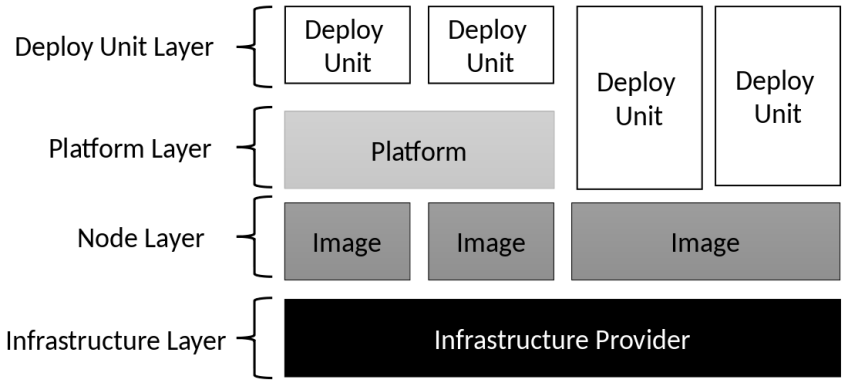

Figura 34: Estructura de las capas Domain Network, Cluster, Node y Host en la pila de capas. La figura muestra como las capas superiores se apoyan en las capas inferiores las cuales les proporcionan las funcionalidades necesarias para que las capas superiores puedan funcionar.

- **Capa de Unidad de Despliegue** (Deploy Unit layer, DUL): contiene los elementos de más alto nivel mas alto denominados unidades de despliegue (Deploy unit, DU). Las DUs son ecosistemas de servicios donde se implementan servicios o dispositivos inteligentes virtuales (vSD), las aplicaciones o servicios locales en la VUN.

  Cada DU puede contener un conjunto de servicios configurados para trabajar conjuntamente proporcionando funcionalidades a los elementos externos a través de su interfaz. Un ejemplo de orquestación de servicios se puede encontrar en los Sistemas de Contenedor de Servicios (MicroServices container Service, MCS), los cuales proporcionan herramientas capaces de administrar varios contenedores para formar una aplicación. DUL puede proporcionar un framework para la ejecución de los contenedores. De hecho, las Redes de usuario virtual (VUN), descritas en la sección 5.2.1, pueden ser desplegadas perfectamente sobre una DUL. Estas redes estaban formadas por ecosistemas digitales que contenían servicios y dispositivos virtualizados capaces de interactuar de manera transparente con dispositivos de usuario físicos conectados a la red doméstica.

- **Capa Plataforma** (Platform layer, PL): proporciona al DUL una base o entorno de ejecución para hacer funcionar determinado software

con el que es compatible independiente de las capas inferiores. La PL puede estar desplegada sobre un nodo de computación por una o varias Capas de Nodo (Node Layer, NL) pero a su vez proporcionará un middleware fiable independientemente del número de nodos alojados en la NL. PL proporciona un entorno operativo de ejecución, completamente configurado y preparado para dar soporte a los servicios y aplicaciones instaladas sobre el mismo.

- **Capa Nodo** (Node layer, NL) está formada por imágenes con sistemas operativos que proporciona un entorno de ejecución estándar en el que se ejecuta CL o DUL.
- **Capa Host** (Host layer, HL) tiene la capacidad de dar proveer soporte a una o más NLs. La HL hace las funciones de un hipervisor o gestión de recursos de infraestructura. Esta capa proporciona acciones para la gestión de recursos de una infraestructura y proporciona información sobre la misma. A partir de esta información, es posible gestionar los elementos desplegados sobre la Capa Nodo.

La descripción formal de los componentes y configuraciones en cada una de estas capas puede realizarse a través de diferentes formatos como XML, JSON, YAML, etc. Sin embargo, para este trabajo se ha preferido hacerse a través de lenguajes declarativos, como los discutidos en 2.4.4, los cuales permiten especificar los componentes software que se deseen utilizar. Además, en caso de que algo no pueda ser representado a través de los lenguajes declarativos puede realizarse a través de código imperativo en forma de scripts los cuales especifican los pasos a seguir para llegar a la configuración deseada.

El código declarativo es útil para definir el estado deseado de un sistema, especialmente cuando no hay mucha variación en los resultados que desea y es necesario definir la forma de alguna infraestructura que se va a necesitar replicar con un alto nivel de coherencia. Por lo tanto, el código declarativo es bueno para definir entornos reutilizables o incluso algunas partes de los entornos.

Además, a veces es necesario escribir código reutilizable pero que pueda producir resultados diferentes dependiendo de la situación. Para estas situaciones los lenguajes declarativos permiten introducir ciertas variaciones a través de la definición de parámetros de configuración definidos en el código descriptivo.

Sin embargo, introducir esta complejidad en lenguajes declarativos como YAML, JSON, XML puede no ser la opción más correcta ya que se pierde

claridad en el código y añade complejidad. En estas situaciones, podría ser más adecuando utilizar lenguajes programables o imperativos debido a que son más apropiados para crear bibliotecas y capas de abstracción.

Teniendo en cuenta estas reflexiones, para el desarrollo de esta tesis se ha optado por la opción de usar una solución basada en lenguaje declarativo ya que simplifica y permite una mejor comprensión de las infraestructuras a desplegar. Además, permite modelar la infraestructura a través de ficheros de texto estructurados en formatos XML, JSON o YAML que facilitan su lectura por herramientas de desarrollo ampliamente extendidas.

Para solucionar el problema de introducir variabilidad en el resultado final se propone el uso de código imperativo en forma de script al finalizar el despliegue de componentes definidos en el lenguaje declarativo. En esta tesis tan sólo se propone usar esta opción para hacer ajustes de configuración muy concretos una vez que se ha desplegado/instalado el software.

Actualmente existen mecanismos para definir y configurar software a través de código los cuales se describieron en la sección 3.2. Entre las iniciativas de este tipo consideradas para su adopción en esta tesis encontramos TOSCA [144] o NetConf/Yang [208] y [209], donde el operador definiría un conjunto de archivos de configuración para especificar la arquitectura e implementarla.

Sin embargo, esos mecanismos son rígidos porque utilizan plantillas que tienen contenido estático y se escriben para una configuración con un objetivo específico a la vez que apenas permiten introducir flexibilidad en los modelos definidos en los archivos de especificación de código. En el caso de actualizar esa configuración, es necesario introducir manualmente los cambios en la plantilla.

La gran ventaja de utilizar un modelo basado en capas software es que permite descomponer y subdividir un sistema complejo en dichas capas facilitando la comprensión y funcionamiento de los orquestadores. Además permite simplificar las opciones y alternativas de configuración proporcionando un grado de abstracción de configuraciones adicionales pertenecientes a otras capas.

Cada capa está compuesta por su propio modelo de datos y las reglas de funcionamiento predefinidas para esa capa. Esto condiciona en gran medida la manera de trabajar del orquestador limitando su alcance a la capa que debe orquestar y a su vez reduce el tiempo de respuesta a la hora de

converger en una solución final. Además, facilita la depuración de las soluciones obtenidas. De hecho, este es un modelo utilizado ocasionalmente en soluciones basadas en tecnologías semánticas ya que son capaces de inferir acciones y generar un flujo de trabajo basado en tareas de acuerdo con las reglas y objetivos definidos [210].

Uno de los aspectos importantes que se ha tenido en cuenta en los trabajos de esta tesis ha sido mantener la compatibilidad de las soluciones obtenidas con el software y herramientas más populares en el mundo de las empresas de TI y comunicaciones. Como se ha visto en la sección 3, existen muchas herramientas de configuración destinadas a fines concretos que permiten configurar diferentes aspectos de lo sistemas de TI tales como: herramientas para describir infraestructuras de sistemas, herramientas de aprovisionamiento de recursos, herramientas para la gestión y despliegue de software y herramientas para la gestión de cambios en post instalaciones (ver Figura 15).

El modelo de capas mostrado en la Figura 34 y la lista de herramientas comentadas tienen una correspondencia directa ya que cada una de las herramientas encajan perfectamente en alguna de las capas propuestas en este trabajo. De hecho, es posible que alguna de las herramienta tenga capacidades para trabajar en varias capas, sin embargo en estos casos la herramienta no proporciona capacidades completas para realizar una gestión adecuada de cada capa. A continuación se detallan los procesos que desarrollan configuraciones para cada una de las capas dentro del orquestador.

6.1.1 *Capa Infraestructura*

Cuando a través de IMOM se desea iniciar el despliegue de servicios sobre una infraestructura de servicios, es necesario que previamente el proveedor de servicios haya accedido al catalogo de servicios disponibles con el objetivo de seleccionar los servicios que le interesan. Una vez seleccionados los servicios para ser desplegados, el orquestador de IMOM (compuesto por PS y CS[2], descritos en la sección 5.2.2, se encargan de identificar la configuración necesaria para desplegar esos servicios. Para poder identifi-

2 PS corresponde al Servicio de Aprovisionamiento o Provision Service, definido en la sección 5.2.2, cuyo propósito es desplegar los servicios sobre la infraestructura INFORD una vez que se ha generado la configuración de cada uno de los servicios. Por otro lado, CS corresponde al Servicio de configuración o Configuration Service, definido en la misma sección, que se encarga de generar la configuración correspondiente de cada servicio a ser desplegado en función de los parámetros introducidos por el Proveedor de servicios (SP).

car las la configuración adecuada es necesario que PS tenga en cuenta las características técnicas de la infraestructura INFORD donde se desplegarán lo servicios seleccionados.

Un proveedor de servicios que desea desplegar una arquitectura de servicios para proveer su propia oferta de servicios deberá configurar la infraestructura adaptada a sus necesidades. La arquitectura INFORD 5.2.1 va a permitir desplegar y configurar las configuraciones requeridas de manera rápida y sencillas a través de IMOM 5.2.2. Una vez introducidos los servicios en IMOM, el orquestador de IMOM comenzará con el proceso de despliegue y configuración software sobre INFORD. El orquestador comenzará dicho proceso configurando las capas situadas en las partes más bajas de la Figura 34 y según vaya terminando de aplicar los cambios de las capas más bajas comenzará con la configuración de las capas que se soportan en la capa ya configurada. Como se puede ver en la Figura 34, la capa situada en la parte de más abajo es la Capa de Infraestructura.

La Capa de Infraestructura es la capa compuesta por proveedores de infraestructura donde se configuran los recursos hardware virtualizados con el objetivo de crear subsistemas donde se desplegará el software necesario para crear múltiples instancias de sistemas independientes. Con el objetivo de gestionar eficazmente la información relevante a las capas se ha organizado a través del modelo de datos mostrado en la Figura 35. La estructura de datos esta formado por las siguientes tablas:

- ConectorProveedor. Esta tabla contiene la información de conexión para que el PS pueda conectarse a la infraestructura cloud que tiene desplegada la arquitectura INFORD. La tabla dispone de los campos básicos que facilitan lo proveedores de infraestructura cloud tales como la dirección IP, el nombre de usuario y claves para identificarse en el sistema, la región de interés, etc.

- ProveedorInfraEstructura. La tabla contiene información general sobre el proveedor cloud que proporciona los recursos de infraestructura.

- Infraestructura. Esta tabla contiene información acerca de la infraestructura cloud en la que se configurará y gestionará los recursos para poder desplegar máquinas virtuales que servirán de base para configurar los servicios y los componentes software.

- Imagen. Los registros contenidos en la tabla contienen información referente al sistema operativo que tiene instalado cada uno de los registros almacenados en la tabla Nodo.

- Usuario. Esta tabla contiene información sobre la configuración de los usuarios que debe tener aplicada cada elemento de la tabla Nodo.
- Nodo. Los registros almacenados sobre esta tabla hacen referencia a los entornos operativos en forma de máquinas virtuales o contenedores que encapsulan software en su interior. Los entornos operativos representados por esta entidad se corresponden con los elementos contenidos en la Capa Nodo.
- ConfiguraciónNodo. Esta tabla contiene información de configuración correspondiente a cada registro de la tabla Nodo.
- Red. La tabla contiene información relacionada con las configuraciones de red asociadas a la tabla Nodo.

En el caso de que sea necesario los campos de cada una de las tablas pueden ser modificados en caso de ser necesario.

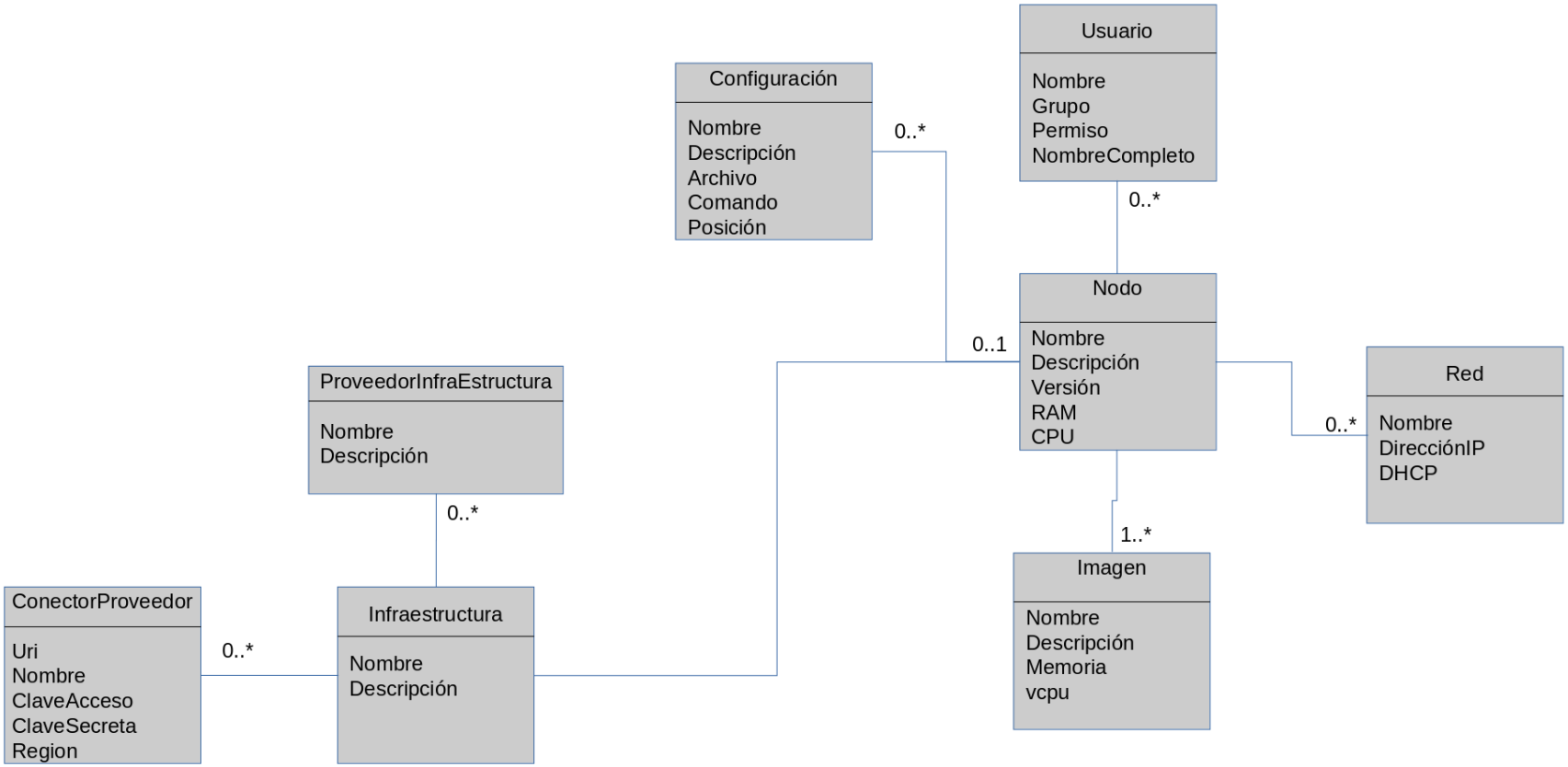

Figura 35: Modelo de datos de la capa de Infraestructura. La figura muestra las tablas, los campos que contienen y la relación entre ellas modelando en su conjunto la estructura de la información para la capa de Infraestructura.

El orquestador de IMOM posee la capacidad con múltiples proveedores de infraestructura, ya que el mismo modelo de datos es capaz de almacenar los datos de conexión requeridos para cada proveedor de infraestructura. Además, este modelo permite almacenar la topología de sistemas complejos los cuales pueden estar compuestos de varias instancias de máquinas junto con las redes que las conectan. Esta capa esta diseñada de forma que puede hacer uso de cualquier herramienta de aprovisionamiento de infraestructura tales como: Terraform, Vagrant, Cloudformation, Azure Resource Manager, etc... que fueron discutidas en el capítulo 3.

6.1.2 *Capa de Nodo*

Una vez que el orquestador ha sido capaz de conectar con la Capa de Infraestructura y configurar los elementos que pudieran ser necesarios en función de las necesidades particulares del proveedor cloud y de la configuración proporcionada por IMOM, el orquestador de IMOM procede a configurar la Capa de Nodo. La Capa de Nodo (NL) proporciona entornos de ejecución en forma de máquinas virtuales o reales, sobre las que correrá un sistema operativo donde posteriormente se desplegarán los servicios del SP.

Durante el proceso de configuración de los nodos de NL, el orquestador de IMOM crea los entornos de ejecución sobre cada uno de los nodos desplegados en la Capa de Nodo. La información que define la configuración de cada Nodo se encuentra registrada en el modelo de datos mostrado en la Figura 35. A partir de esa información, el orquestador genera archivos de configuración para que posteriormente sean procesados por una HGC encargada de configurar cada uno de los Nodos registrados en el modelo de datos. La HGC es capaz de gestionar imágenes de sistemas operativos que serán desplegados sobre un proveedor de infraestructura y además sera capaz de aplicar aspectos de configuración avanzados tales como usuarios del sistema, redes de conexión o incluso algún servicio de red tal como DHCP, cortafuegos además de algunas reglas del propio cortafuegos.

6.1.3 *Capa de Plataforma*

La Capa de Plataforma (PL) se corresponde con el software encargado de proveer los entornos de ejecución sobre los que funcionarán los servicios y aplicaciones definidos en la Capa de Despliegue. La configuración y despliegue de los elementos contenidos en esta capa se realizan después de que el orquestador de IMOM ha finalizado el despliegue de la Capa de Nodo. El proceso de configuración de la PL requiere información registrada

en el modelo de datos 36 el cual tiene como objetivo gestionar eficazmente la información correspondiente de cada middleware que es desplegado en la Capa de Plataforma. El modelo de datos esta formado por las siguientes tablas:

- Plataforma. Esta tabla contiene información del middleware, entorno de ejecución, framework que permitirá desplegar los servicios y aplicaciones de los SPs.
- Dependencia. Esta tabla permite registrar librerías y componentes software necesarios para ser instalados antes de que se instale los middleware registrados en la tabla Plataforma.
- Atributo. Esta tabla contiene valores de parámetros de configuración que están relacionados con cada registro de la tabla Plataforma. El objetivo de esta tabla es registrar parámetros que podrían ser necesario aplicar a la hora de configurar el software.
- Usuario. Esta tabla contiene información sobre la configuración de los usuarios que debe tener aplicada cada elemento de la tabla Plataforma.
- Configuración. Esta tabla contiene información de configuración correspondiente a cada registro de la tabla Plataforma.

En esta capa se prepara el software que proveerá un middleware, entorno de ejecución o frameworks necesarios donde se ejecutarán las aplicaciones y servicios finales del sistema. Por ejemplo, un servicio web basado en java que requiere contenedores EE, EJB, etc. para facilitar el acceso de bases de datos. Debido a lo extendido que está el uso de middlewares, entornos de ejecución o frameworks en sistemas informáticos, las HGC analizadas en la sección 3.2.1 disponen mecanismos para desplegar fácilmente este tipo de software.

6.1.4 *Capa Unidad de Despliegue*

Las Unidades de Despliegue (DUL) representan el software de las aplicaciones y servicios que serán instalados en la parte más alta de la pila de capas mostradas en la Figura 34, por encima de esta capa no hay más software. Para el despliegue de los elementos contenidos en la Unidad de Despliegue, el orquestador de IMOM recopila la información que especifica el software que debe ser instalado en esta capa. La estructura de esta información se ajusta al modelo de datos presentado en la Figura 37. El modelo de datos esta formado por las siguientes tablas:

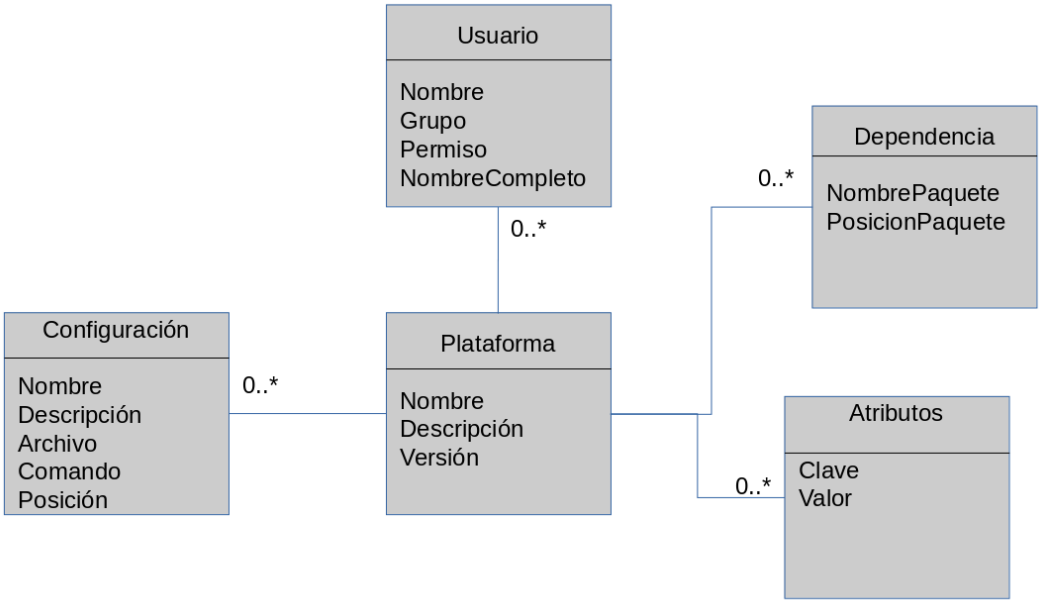

Figura 36: Modelo de datos de la Capa de Plataforma. La figura muestra las tablas, los campos que contienen dichas tablas y la relación entre ellas modelando en su conjunto la estructura de la información para la capa de Plataforma.

- UnidadDespliegue. Esta tabla contiene información correspondiente al software que va ser instalado dentro de DUL. Este software puede estar formado por servicios, aplicaciones o componentes software necesarios para el SP.
- Dependencia. Esta tabla permite registrar librerías y componentes software necesarios para ser instalados antes de que se instale el software registrados en la tabla UnidadDespliegue.
- Atributo. Esta tabla contiene valores de parámetros de configuración que están relacionados con cada registro de la tabla UnidadDespliegue. El objetivo de esta tabla es registrar parámetros que podrían ser necesario aplicar a la hora de configurar el software.
- Usuario. Esta tabla contiene información sobre la configuración de los usuarios que debe tener aplicada para cada elemento de la tabla UnidadDespliegue.
- Configuración. Esta tabla contiene información de configuración correspondiente a cada registro de la tabla UnidadDespliegue.

El proceso de despliegue de los elementos de la capa DUL requiere que el orquestador de IMOM, a partir de la información almacenada en el modelo de datos, genere los archivos de configuración que indiquen el software

y los cambios que deben realizarse para desplegar los servicios correspondientes. Estos archivos de configuración tienen un formato comprensible para la HGC encargada de aplicar dichos cambios. En IMOM, los técnicos deberán indicar previamente qué HGC será utilizada para aplicar estos cambios, ya que es necesario conocerla debido a que el orquestador generará los archivos de configuración acorde a la HGC seleccionada porque el orquestador generará los archivos de configuración con un formato comprensible para la herramienta.

La capa DUL provee de los servicios o aplicaciones requeridos por los SPs para ofrecer su paquete de servicios a los usuarios finales. Por ejemplo, en esta capa se instalarán servicios para gestión de usuarios, servicios de gestión de contenidos digitales o de distribución de contenidos, etc.

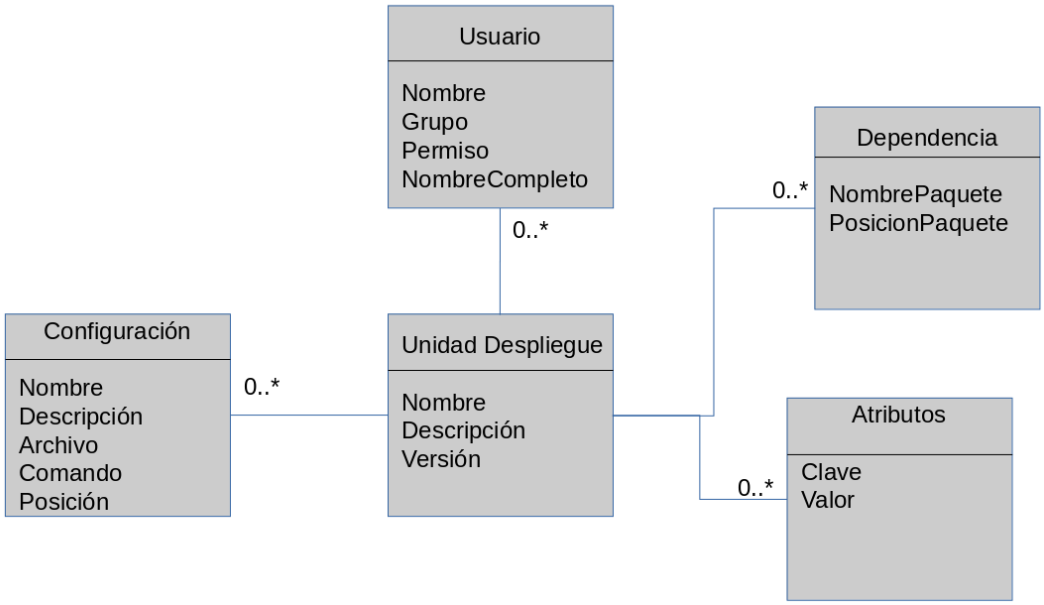

Figura 37: Modelo de datos de la Capa de Unidad de despliegue. La figura muestra las tablas, los campos que contienen dichas tablas y la relación entre ellas modelando en su conjunto la estructura de la información de la capa de Unidad de despliegue.

Delegar la gestión del software en Herramientas de Gestión del a Configuración (HGC) no sólo facilita y asegura la aplicación de los cambios deseados si no que también permite la actualización de software que ya ha sido desplegado. La actualización de software puede ser debido a la necesidad de personalizar algún software ya instalado o tan sólo se deba a la necesidad de aplicar alguna actualización de software. Cuando es necesario aplicar cambios en el software instalado, el orquestador a través de la HGC comprueba si el servicio (o el componente software deseado) se encuentra instalado, en caso afirmativo se comprueba si el servicio está actualizado y

si no es así la HGC inicia el proceso donde se comprueba la actualización de las dependencias y servicio a actualizar junto a sus configuraciones y usuarios.

6.2 PROCESO DE CONFIGURACIÓN DE INFRAESTRUCTURA MULTICAPA

Una vez presentadas la diferentes capas software en las que pueden descomponerse una infraestructura de un sistema de TI, en este apartado se presentan los componentes independientes de sistema que permitirán el despliegue de una aplicación de SP sobre un proveedor de infraestructura dado.

En el capítulo 3 se ha realizado un análisis pormenorizado de las principales herramientas de uso libre donde se describieron y se clasificaron dependiendo si estaban orientadas a configurar algún tipo de software o a la gestión o definición de infraestructuras. En la Figura 38 se muestra una clasificación de las herramientas vistas en los capítulos 2 y 3 distribuidas en el modelo de capas software propuesto en la sección 6.1. En la Figura 38 cada fila corresponde a una de las capas software mostradas en la sección 6.1 y a la derecha de esta columna se muestran las herramientas analizadas en los capítulos 2 y 3. La figura no sólo muestra las HGCs sino también las APIs y conectores capaces de gestionar los recursos de cada capa. Esta clasificación es útil ya que de un vistazo puede ayudar a los técnicos saber qué herramienta podría utilizar para configurar software o gestionar recursos de algún proveedor de infraestructura.

La tabla 17 complementa la información de la Figura 38 con el objetivo de ayudar a relacionar cada capa de infraestructura con las tareas que se realizan en cada una de ellas junto a las herramientas y estándares que facilitan su configuración.

El análisis de las herramientas disponibles realizado en el capitulo 2 evidencia que no hay herramientas que permitan abarcar toda la pila software de una infraestructura (cuestión que se puede observar en la Figura 38 y tabla 17).

Por lo tanto, cumplir con los objetivos marcados requiere de la combinación de varias herramientas para abarcar aquellos aspectos de la configuración que no pueden ser cubiertos por una única herramienta. Para dar una visión de esta problemática se describen a continuación tres escenarios de aplicación que combinan el uso de diferentes HGCs para configurar soft-

Capa	Procesos	Aplicación práctica	Mecanismos de configuración
Deploy Unit	Actualizaciones de SW Aplicar configuraciones post-instalaciones Instalación de SW	Carga de datos en BBDD Restauración de Backups Replicación de servicios Configuraciones adicionales	TOSCA, Ansible, SaltStack, Puppet, Chef
Platform	Instalación de frameworks Instalación de entornos de configuración Instalación de Servidores de aplicación	Java, Docker, BBDD, Tomcat	TOSCA, CAMP
Node	Definición de infraestructura por plantillas	LXD, Imágenes SO.	Packer, TOSCA, OVF
Infrastructure	Aprovisionamiento de infraestructura	Crear VMs. Crear redes virtuales.	OCCI, CIMI, Libcloud, DeltaCloud, Terraform, Cloudformation, OpenStack Heat, TOSCA

Tabla 17: Distribución de tareas y herramientas por capa de infraestructura.

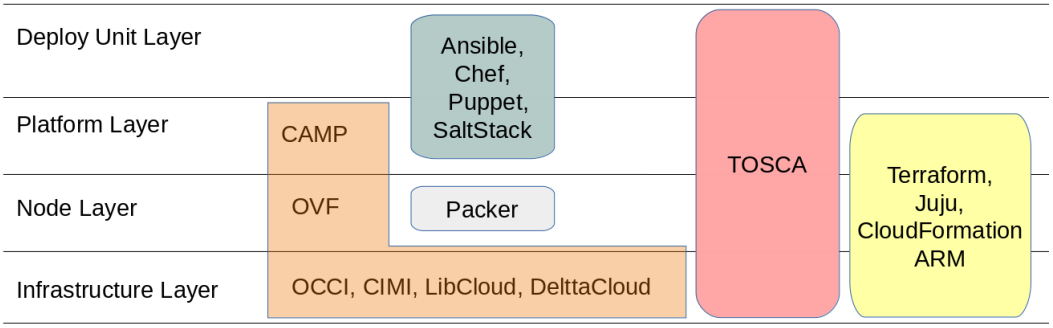

Figura 38: La figura muestra las capas software identificadas en este trabajo y también las herramientas de gestión de configuración sobre las capas en las que son capaces de trabajar. Como se puede ver, TOSCA es la herramienta con más versátil capaz de aportar funcionalidades en cada una de las capas.

ware situado en los diferentes niveles de las infraestructuras.

Cada herramienta una de las herramientas analizadas tiene sus propias ventajas y desventajas y los administradores deben ser capaz de evaluar su caso particular y seleccionar aquellas herramientas que más se adapten a sus necesidades. A continuación se presentan tres combinaciones comunes que funcionan bien en una serie de situaciones:

1. Aprovisionamiento de infraestructura (Terraform) junto con software de gestión de la configuración (Ansible): En este caso Terraform implementaría toda la infraestructura subyacente, incluida la topología de red (es decir, VPC, subredes, tablas de rutas), almacenes de datos (por ejemplo, MySQL, Redis), equilibradores de carga y servidores.

   Ansible implementaría las aplicaciones en la parte superior de esos servidores. Este es un enfoque fácil, ya que no hay ninguna infraestructura adicional para ejecutar (Terraform y Ansible son aplicaciones solo para clientes) y hay muchas maneras de conseguir que Ansible y Terraform trabajen juntos (por ejemplo, Terraform agrega etiquetas especiales a sus servidores y Ansible utiliza esas etiquetas para encontrar el servidor y configurarlas).

   Sin embargo esta configuración tiene un inconveniente que se debe a que la HGC Ansible trabaja principalmente con código imperativo y está orientado principalmente a servidores mutables, por lo que a medida que se vayan introduciendo cambios, la infraestructura crezca

y las configuraciones se amplíen, el código crecerá y el mantenimiento puede volverse más difícil.

2. Aprovisionamiento más definición por plantillas (templating) del servidor. Ejemplo: Terraform y Packer. Se usaría Packer para empaquetar las aplicaciones como imágenes de máquina virtual y Terraform para implementar: (a) servidores con estas imágenes de máquina virtual y (b) el resto de la infraestructura, incluida la topología de red (es decir, VPC, subredes, tablas de rutas), almacenes de datos (por ejemplo, MySQL, Redis) y equilibradores de carga.

   Este enfoque parte de que no existe una infraestructura inicial por lo que se evitan problemas derivados de configuraciones iniciales. Además, se trata de un enfoque de infraestructura inmutable, que facilitará el mantenimiento ya que cualquier cambio o modificación requiere la destrucción eliminación del software correspondiente y su nuevo despliegue implementado aquellos cambios, eliminando posibles incoherencias o incompatibilidades con el software previamente instalado. Sin embargo, hay dos inconvenientes importantes.

   En primer lugar, las máquinas virtuales pueden tardar mucho tiempo en compilarse e implementarse si es necesario realizar grandes cambios o instalaciones complejas del software ralentizando la velocidad de despliegue. En segundo lugar, las estrategias de despliegue que se puede implementar con Terraform son limitadas y estas limitaciones pueden ser evitadas a través del uso de scripts de configuración. Sin embargo, los scripts de configuración suelen ser complejos de desarrollar por lo que se podría recurrir al uso de Herramientas de Gestión de la Configuración.

3. Aprovisionamiento más definición por plantillas (templating) del servidor y orquestación. Para este escenario se podrían usar las herramientas: Terraform, Packer, Docker y Kubernetes donde en primer lugar s haría uso de Packer para crear una imagen de máquina virtual que tenga instalado Docker y Kubernetes.

   A continuación se usaría Terraform para desplegar: (a) un clúster de servidores donde cada servidor ejecuta esta imagen de máquina virtual y (b) el resto de la infraestructura que compuesta por la topología de red (es decir, VPC, subredes, tablas de rutas), almacenes de datos (por ejemplo, MySQL, Redis) y balanceadores de carga. La configuración final se completa al inicio del clúster de servidores donde se

forma un clúster de Kubernetes para ejecutar y administrar las aplicaciones dockerizadas. Este enfoque tiene la ventaja de que las imágenes de Docker son rápidas de crear permitiendo ser probadas y ejecutadas en un equipo local antes de desplegarlo en la infraestructura final.

El inconveniente de esta solución es la complejidad debido a que los clústeres de Kubernetes son difíciles y costosos de implementar y de gestionar, aunque la mayoría de los proveedores de infraestructura proporcionan servicios administrados de Kubernetes que facilitan parte de este trabajo. Además, el uso de varias capas de abstracción (Kubernetes, Docker, Packer) añade complejidad al sistema final.

6.3 CONCLUSIONES

En este capítulo se ha presentado un modelo jerárquico de capas software sobre el cual se distribuyen los diferentes elementos software necesarios para la ejecución de servicios de propósito general. En este capítulo se propone dividir las infraestructuras de los sistemas TI en 4 capas independientes pero relacionadas unas con otras en donde el software se gestiona específicamente en función de cada capa.

La Figura 34 muestra la pila de capas propuesta en este capítulo, como puede ser visto en la figura, la capa más baja es la encargada de la gestión de los recursos hardware que definen la infraestructura del sistema junto con las redes y conexiones, la siguiente capa que se apoya sobre esta primera capa contiene instancias de imágenes de sistemas operativos sobre las que se configurará el software de las capas superiores. La capa tercera es la encargada de proporcionar los entornos de ejecución, plataformas y frameworks que soportarán los servicios de la capa superior. Finalmente la capa superior contiene los servicios y aplicaciones de más alto nivel y que suelen proporcionar la funcionalidad final a lo clientes y consumidores.

La configuración de cada una de las capas requiere modelar la información de manera particular debido a que no todas las capas se comportan igual, ni tienen los mismos objetivos y ni proporcionan las mismas funcionalidades por lo que ha sido necesario analizarla una a una para identificar el conjunto de datos relevantes para cada una de ellas.

Como conclusión, la segmentación de los sistemas en capas permiten un mayor control y focalización en las configuraciones perimetrales dentro de cada una de las capas. Esto permite reducir la probabilidad de errores de configuración, ayuda a la selección de las herramientas adecuadas para

configurar la capa correspondiente y reduce la complejidad de las configuraciones minimizando el alcance a la capa a configurar.

7

VALIDACIÓN DE LOS MECANISMO DE ORQUESTACIÓN Y DESPLIEGUE DE IMOM

Este capítulo está dedicado a la validación de las ideas presentadas en el Capítulo 6. En primer lugar la sección 7.1 presenta la arquitectura del prototipo desarrollado, sus componentes y las diferentes capas software que lo componen. Para la validación se ha realizado una especificación de la infraestructura sobre el modelo de datos y a partir del cual, el PS[1] ha generado el código de especificación de infraestructura en un formato adaptado al proveedor de infraestructura seleccionado para el escenario de pruebas. A continuación en la sección 7.3 se presentan las medidas obtenidas en la ejecución del prototipo y posteriormente se muestra un análisis de dichos resultados. Por último, la sección 7.4 concluye señalando los principales resultados derivados de las pruebas de realizadas.

7.1 DETALLES DEL PROTOTIPO

Con el objetivo de probar los mecanismos de configuración implementados en el PS, se ha desarrollado un prototipo que permite generar configuraciones para realizar, posteriormente, el despliegue de software automáticamente a través de la instanciación de dicha configuración. Estos mecanismos permiten generar configuraciones software para cualquiera de las capas que forman un sistema de TI. Es decir, desde las capas más bajas cercanas al hardware hasta las capas superiores destinadas a albergar a aplicaciones y servicios. La arquitectura del prototipo desarrollado se muestra en la Figura 39, donde se puede ver que está compuesta por una arquitectura distribuida sobre dos nodos físicos. A continuación se describen cada uno de ellos:

- **Nodo del servicio de aprovisionamiento**. Representa un equipo desde donde los ingenieros de sistemas pueden diseñar y definir la arquitectura INFORD. Para este cometido utilizan el PS, el cual genera configuraciones que serán aplicadas sobre el Nodo INFORD. Estas configuraciones son desplegadas en la infraestructura INFORD a través del Network Manager Service, que se describe más adelante.

1 PS corresponde al Servicio de Aprovisionamiento o Provision Service, definido en la sección 5.2.2, cuyo propósito es desplegar los servicios sobre la infraestructura INFORD una vez que se ha generado la configuración de cada uno de los servicios.

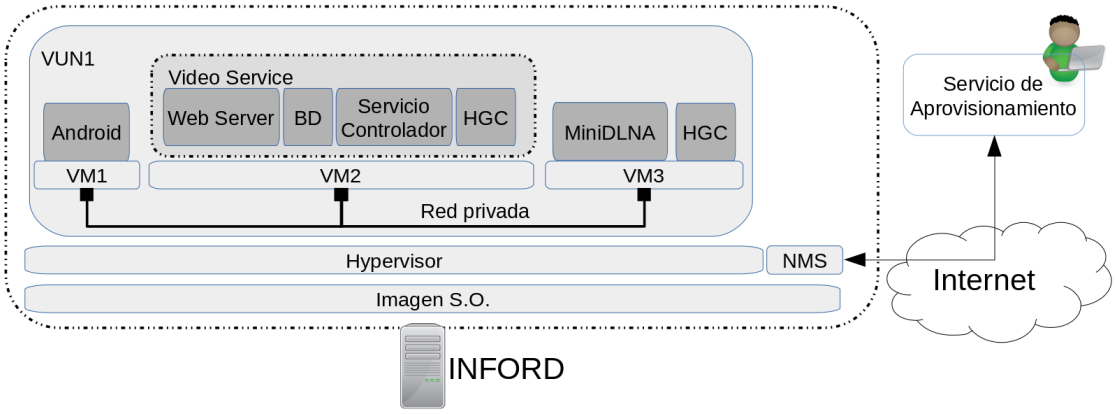

Figura 39: La figura muestra la arquitectura del prototipo desarrollado para validar las capacidades de orquestación a través de las herramientas de configuración. La figura muestra el software utilizado en el prototipo y la jerarquía de las capas software que lo componen.

- **Nodo INFORD**. Representa una máquina física la cual presenta un modelo de infraestructura por capas de software acorde al modelo mostrado en la sección 6.1. Para el prototipo propuesto, el nodo INFORD está compuesto por una máquina que contiene instalada una imagen del sistema operativo Ubuntu Server LTS 20.04. Esta imagen es instanciada en la máquina junto a dos componentes software:
  - **NMS (Network Manager Service)**. El NMS se encarga de gestionar la infraestructura software desplegada sobre INFORD y esta basada en una HGC. Para el prototipo desarrollado se ha optado por utilizar Vagrant [176] como HGC debido a que proporciona funcionalidades necesarias a la hora de desplegar instalar y configurar software. Una de las características más importantes es que Vagrant permite trabajar con diferentes proveedores de infraestructura lo que permite extender el alcance del prototipo a múltiples proveedores de infraestructura.
  - **Hypervisor**. Este componente representa un hipervisor genérico, sin embargo, el prototipo desarrollado utiliza un hipervisor basado en VirtualBox debido a su versatilidad, fiabilidad y licencia de uso. La arquitectura modular del prototipo podría soportar otros tipos de hipervisores.

A través de estos componentes y gracias a la organización de la arquitectura del prototipo en capas, componentes software y servicios, el PS[2] es capaz de transferir las especificaciones del modelo de datos contenido en IMOM

2 Servicio de aprovisionamiento (Provision Service, PS) que se encarga de desplegar los servicios sobre la infraestructura INFORD. Está definido en la sección 5.2.2

a una infraestructura real. El PS traduce la información de especificación de infraestructura definida en IMOM al formato del código de especificación de infraestructura comprendido por la herramienta Vagrant. Una vez hecho esto, Vagrant interpreta dicho código e instancia la infraestructura indicada además de las posibles configuraciones de red.

Para poder validar mejor la solución propuesta se ha planteado modelar un escenario de aplicación donde se instancia un ecosistema virtual formado por una red privada VUN y varios dispositivos domésticos virtuales los cuales son conectados a la red privada. Entre los dispositivos virtuales y servicios se ha instanciado un dispositivo virtual basado en el sistema operativo Android, un servidor de contenidos digitales a través del protocolo DLNA y un servicio de vídeo inteligente (Smart Video Service, SVS). El SVS está compuesto por un repositorio de vídeos, un servidor web, una base de datos y un controlador encargado del funcionamiento general del SVS.

El SVS ofrece una interfaz web donde los usuarios pueden acceder a las funcionalidades, entonces el servicio controlador gestiona las peticiones consultando la información almacenada en el repositorio sobre cada usuario además de los vídeos disponibles en el sistema, y como resultado, puede ofrecer un catalogo de vídeos personalizado o incluso iniciar la visualización de algún vídeo. El despliegue, instalación y configuración del SVS es gestionado a través de una HGC una vez que se ha creado la máquina virtual la cual alberga dicho servicio (VM2). El proceso de despliegue de este servicio se compone de dos subprocesos: el primero incluye las acciones ejecutadas por Vagrant para configurar VM2 y la configuración de red, y el segundo subproceso incluye el despliegue de SVS a través de una HGC. Para el desarrollo del prototipo se ha seleccionado utilizar Puppet como HGC debido a que se puede integrar fácilmente con Vagrant y de esta manera aunque el proceso de despliegue se compone de dos subprocesos, el proceso de despliegue queda completamente integrado en una única ejecución que incluye la ejecución secuencial de los dos subprocesos que componen el proceso de despliegue.

7.2 DETALLE DEL FUNCIONAMIENTO

Para poder validar el prototipo desarrollado y crear la infraestructura mostrada en la Figura 39 es necesario disponer de la información que modela dicha infraestructura. Esta información esta almacenada y estructurada según el modelo de datos mostrado en la Figura 35.

Nombre	vCPU	RAM	Dirección IP	Archivo configuración de la HGC
VM1	1	2048	192.168.100.11	-
VM2	1	2048	192.168.100.12	installSWVM2.pp
VM3	1	1024	192.168.100.13	installSWVM3.pp

Tabla 18: Información almacenada en el modelo de datos para describir el prototipo desarrollado.

Como se puede ver en la Figura 39, el prototipo se compone de varias máquinas con las características que se muestran en la tabla 18. La tabla es una representación simplificada de la información contenida en el modelo de datos de la Figura 35. Se ha resumido esta información para ayudar al lector a la hora de comprender la idea general de lo que se pretende explicar en este capítulo, sin profundizar en la información contenida en el modelo de datos y sus relaciones, tan sólo mostrando la información relevante desde una perspectiva sencilla y simple.

Una vez que el PS dispone de la información que especifica la infraestructura del prototipo, el PS se encarga de generar el código de configuración adecuado para que pueda ser procesado por las herramientas de aprovisionamiento disponibles. Para esto, el servicio de aprovisionamiento (Provision Service, PS) integra un plugin de Vagrant que le permite generar un código de despliegue de infraestructura el cual, está compuesto por dos bloques de código bien diferenciados. Por un lado está el bloque que contiene la información en la que se define los nodos, redes, conexiones y las características que definen dicha infraestructura. Dicha especificación se muestra en el formato mostrado en el cuadro siguiente:

Cuadro 3: Especificación de los nodos, redes, conexiones y características del prototipo en formato JSON

```
servers=[
{
  :hostname => "VM1",
  :ip => "192.168.100.11",
  :box=> "dictcp/android-x86",
  :ram => 2048,
  :cpu => 1,
},
{
  :hostname => "VM2",
  :ip => "192.168.100.12",
  :box=> "ubuntu/trusty64",
```

```
    :ram => 2048,
    :cpu => 1,
    :pathFiles => "puppet/files/",
    :configuracionHGC => "installSW.pp"
  },
  {
    :hostname => "VM3",
    :ip => "192.168.100.13",
    :box=> "ubuntu/trusty64",
    :ram => 2048,
    :cpu => 1,
    :pathFiles => "puppet/files/",
    :configuracionHGC => "installSWVM3.pp"
  }
]
```

Y por otro lado se encuentra el bloque con el código correspondiente de aplicar la configuración sobre la infraestructura (para este prototipo la infraestructura esta formada tan sólo por una máquina física como base).

Este último bloque de código implementa un algoritmo que lee la información de la variable "servers" (cuadro 5) la cual contiene la arquitectura a desplegar en formato JSON. La información en la variable "servers.[es] generada por el PS a partir de los parámetros de configuración registrados en el modelo de datos de IMOM.

A continuación, el algoritmo procesa la información de cada uno de los elementos a instanciar y aplica uno por uno la configuración correspondiente sobre la infraestructura. El algoritmo diseñado se caracteriza por ser sencillo y genérico lo que facilita su mantenimiento y adaptación a nuevas necesidades. El algoritmo permite trabajar tanto en arquitecturas simples compuestas de pocas máquinas, como arquitecturas con un gran número de máquinas conectadas a través de redes independientes. El algoritmo se muestra en el cuadro 4:

Cuadro 4: Algoritmo del prototipo para despliegue de infraestructuras a través de Vagrant

```
Vagrant.configure(2) do |config|
    servers.each do |machine|
        config.vm.define machine[:hostname] do |node|
            node.vm.box = machine[:box]
            node.vm.hostname = machine[:hostname]
            node.vm.network "private_network", ip: machine[:ip]
            node.vm.provider "virtualbox" do |vb|
```

```
                #vb.customize ["modifyvm", :id, "--memory", machine[:
                    ram]]
                # Nombre de la VM
                vb.name = machine[:hostname]
                # Memoria
                vb.memory = machine[:ram]
                # Num. procesadores
                vb.cpus = machine[:cpu]
            end
        #-------------------------------------
        # Bloque aprovisionamiento con Puppet
                ...
        #-------------------------------------
        end
    end
end
```

El código mostrado en el cuadro 4 se encarga del aprovisionamiento de las capas inferiores de las infraestructuras sobre la que se desplegará el resto del software. Como se puede apreciar en dicho código, el algoritmo procesa una por una las especificaciones de la variable "servers" e instancia los elementos correspondientes.

El código mostrado en el cuadro 4 se ha simplificado eliminando la parte del código que se encarga del aprovisionamiento software de las capas superiores el cual está asignado normalmente a las HGCs. Este código se integraría dentro de la sección "Bloque aprovisionamiento con Puppet" y en el cuadro 5 se muestra dicho código.

Cuadro 5: Bloque de código utilizado en el prototipo para el aprovisionamiento a través de la HGC Puppet.

```
node.vm.provision :puppet do |puppet|
  # Path a los ficheros para configurar manifiestos
  puppet.manifests_path = machine[:pathFiles]
  # Nombre del manifiesto que se va a ejecutar inicialmente
  puppet.manifest_file = machine[:configuracionHGC]
  # Carpeta ficheros auxiliares
  puppet.module_path = machine[:pathFiles]
  # Opciones de Puppet. Se activa el modo debug y verbose
  puppet.options = [
    '--verbose',
    '--debug',
  ]
end
```

Para proporcionar las funciones de las HGCs en este prototipo se ha optado por el uso de la HGC puppet por su claridad a la hora de definir el software a través del código declarativo y por tener una curva de aprendizaje moderada. El prototipo podría soportar alguna otra HGC, de hecho Vagrant es compatible con la mayoría de las HGC analizadas en este trabajo. Vagrant viene preparado para integrar HGCs en el proceso de instanciación de infraestructura, pero en el caso que esto no fuese posible, en esta tesis se ha tenido en cuenta esta posibilidad y para solucionarlo se había pensando en utilizar un sencillo script capaz de instalar la HGC sobre la máquina instanciada para que posteriormente se pudieran aplicar las configuraciones adaptadas a la HGC instalada. La idea es que dicho script de configuración inicial se ejecutase al arrancar por primera vez la nueva máquina instanciada creándose así una nueva configuración para poder desplegar software a través de la HGC instalada.

Durante el desarrollo de esta tesis se ha comprobado que la mayoría de las herramientas de gestión de infraestructura dispone de algún mecanismo que permite ejecutar comandos o scripts de configuración. Esto es importante ya que permite explotar la posibilidad de integrar una HGC en el proceso de iniciación de la instancia de la infraestructura final.

Como se ha comentado a lo largo del trabajo, uno de los objetivos es conseguir despliegues de infraestructuras automáticas a través de infraestructuras mutables. Esto implica disponer de imágenes genéricas listas para ser instanciadas sobre un infraestructura y posteriormente hacer los cambios necesarios para adaptar la configuración a su uso final. Estos cambios se realizan a través de las HGCs encargadas de instalar los servicios y componentes software de más alto nivel.

Hasta ahora se ha descrito como se crea la infraestructura a partir de la información proporcionada en la variable "servers"(cuadro de código 4), pero no se ha comentado cómo es el proceso de despliegue del software referente a los servicios y aplicaciones. Por lo tanto, se va a proceder a describir los componentes software que componen la VUN para posteriormente comentar el proceso y configuraciones para el despliegue del software de los servicios contenidos en la VUN.

Como se ha mostrado en la Figura 39, la VUN se compone de tres VMs que se describen a continuación. VM1 la cual está formada por una imagen del SO Android que puede ser considerada como dispositivo inteligente al cual se conectan los usuarios desde dispositivos más modestos gracias a una interfaz VNC, Web, etc. Esta manera de funcionar es similar a un espe-

jo ya que este mecanismo permite a los usuarios que su dispositivo físico refleje las acciones del dispositivo virtual. El dispositivo virtual está capacitado para tener acceso a las funciones avanzadas y a los servicios de la VUN y todas las acciones realizadas son mostradas en el dispositivo físico a pesar de que pueda tener capacidades inferiores al dispositivo virtual.

VM2 esta compuesta principalmente por un conjunto de servicios que forman el SVS. El SVS está compuesto principalmente por un servidor web con motor PHP, una base de datos y un servicio controlador desarrollado en PHP que se encarga de gestionar las peticiones de los usuarios. En función de la petición enviada por los usuarios, el servicio controlador consultará la información almacenada en la base de datos y en el repositorio de contenidos multimedia y podrá proporcionar el catalogo de contenidos disponibles o acceso para la reproducción a los mismos. La HGC instalada sobre la VM2 es la encargada de configurar estos servicios, para ello el PS genera código de configuración adaptado a la sintaxis de Puppet para que los servicios puedan ser desplegados sin problemas.

Traducir la configuración almacenada en el modelo de datos al formato de Puppet no es una tarea sencilla, por esta razón se han creado múltiples plantillas adaptadas para trabajar con Puppet y se han clasificado en función de su objetivo, es decir, se han creado plantillas básicas dependiendo de si se pretende instalar paquetes software, ejecutar servicios o incluso aplicar configuraciones optimizadas. Esto se hace a través de fragmentos de código bien definidos en los que tan sólo se requiere sustituir los nombres de los servicios o paquetes requeridos, salvo en ciertos casos en los que se requiere añadir alguna pieza de código adicional que podría realizarse a cargo del administrador o a través de reglas que gestionen patrones de comportamiento previamente configurados. En el cuadro 6 se muestra el código de configuración destinado a desplegar el software en la VM2 el cual está contenido en el fichero "installSWVM2.pp":

Cuadro 6: Código del prototipo para realizar la instalación de los servicios y software necesario para VM2.

```
    # Actualizar los repositorios de paquetes
exec { "apt-get update":
    command => "/usr/bin/apt-get update"
}
# Lista de paquetes de PHP para instalar
$packages = [
    "mysql-server",
    "apache2",
    "php5",
```

```
        "php5-cli",
        "php5-mysql",
        "php5-dev",
        "php5-curl",
        "php-apc",
        "libapache2-mod-php5"
    ]
    # Instalar los paquetes
    package { $packages:
        ensure => present,
        require => Exec["apt-get update"]
    }
    # Arrancar el servicio de Apache
    service { "apache2":
        ensure => running,
        require => Package["apache2"]
    }
    # Arrancar el servicio de MySQL
    service { "mysql":
        ensure => running,
        require => Package["mysql-server"]
    }
    # Copia de los archivos del controlador
    file { "/var/www/html/controlService.php":
              ensure => present,
              source => "puppet:///modules/controlService.php",
    }
    # Configurar base de datos con el fichero para configurarlo
     file { 'my.cnf':
        ensure  => present,
        path    => '/etc/mysql/my.cnf',
        source  => 'puppet:///modules/mysql/my.cnf',
    }
```

En el cuadro 6 se muestra el archivo generado por el PS a partir de la información del modelo de datos, el archivo es sencillo y es sencillo observar acciones llevadas a cabo por la HGC. Por ejemplo, la pieza de código `exec{...}indica`que se debe actualizar los repositorios de software desde donde se instalará el software. El fragmento `package{...}indica`los paquetes que deben ser instalados, `service{...}indica`que el servicio debe ser arrancado (la opción: `ensure=>running indica`que debe ejecutarse al iniciar la máquina) y `file{...}copia`ficheros de configuración a las localizaciones adecuadas, en este caso se usa para determinar el comportamiento de la base de datos.

Esta configuración es almacenada en el archivo "installSWVM2.pp" el cual es referenciado en la configuración del nodo a configurar a través del parámetro "configuracionHGC" (parámetro indicado para cada VM a configurar y especificado en el cuadro 5). Este parámetro hace referencia al archivo con la configuración que debe ser aplicada sobre la máquina que va a ser desplegada.

El proceso de configuración para VM3 es similar a VM2 pero mucho más sencillo ya que está compuesto básicamente por el servicio MiniDLNA. MiniDLNA es un servicio DLNA que permite distribuir contenidos multimedia mediante el protocolo DLNA a los dispositivos que se encuentran en la misma red local. Los servicios contenidos en VM3 son desplegados una vez que la VM ha sido creada a través de la configuración contenida en el fichero de configuración "installSWVM3.pp". La configuración contenida en dicho fichero se muestra a continuación:

Cuadro 7: Código del prototipo para realizar la instalación de los servicios y software necesario para VM3.

```
# Actualizar los repositorios de paquetes
exec { "apt-get update":
    command => "/usr/bin/apt-get update"
}
 $packages = [
    "mysql-server",
    "minidlna",
]
# Instalar los paquetes
package { $packages:
    ensure => present,
    require => Exec["apt-get update"]
}
# Instalar de  minidlna
package { "minidlna":
    ensure => present,
    require => Exec["apt-get update"]
}
# Arrancar el servicio de  minidlna
service { "minidlna":
    ensure => running,
    require => Package["minidlna"]
}
# Configurar servidor minidlna
 file { 'minidlna.conf':
    ensure  => present,
```

```
    path    => '/etc/minidlna.conf',
28  source  => 'puppet:///modules/filesConf/minidlna.conf',
}
```

Como se puede ver en el cuadro 6, el código generado para VM3 es mucho más sencillo que el generado para VM2 ya que tan sólo se requiere configurar un único servicio.

7.3 RESULTADOS

En el apartado anterior se ha presentado la arquitectura de un prototipo y además se han mostrado los conceptos básicos de funcionamiento. En este apartado se muestran las medidas realizadas durante su funcionamiento.

Durante la ejecución del prototipo, el PS generó código que pudo ser interpretado por la HGC Puppet dando como resultado un pequeño ecosistema software. Aprovechando las ventajas que el prototipo permite de destruir y volver a crear la infraestructura al completo desde cero, se ha realizado una investigación sobre un caso práctico muy habitual en los proveedores de servicios que ocurre cuando tienen que incluir cambios en servicios que ya están desplegados sobre un servidor o infraestructura. Para este proceso existen dos posibilidades:

1. Se aplican las modificaciones sobre las máquinas que están en uso.
2. Se aprovisionan nuevas máquinas que incorporen los cambios para reemplazar las máquinas no actualizadas una vez que han sido validadas.

Estos dos paradigmas para aplicar los cambios se corresponden con el concepto de infraestructura mutable (1) e infraestructura inmutable (2). Sin embargo, existe la posibilidad de poder optar a un modelo híbrido entre ambas opciones, es decir, un modelo intermedio que permite hacer modificaciones de tamaño mediano. El modelo intermedio no está orientado a realizar una modificación pequeña como podría ser la modificación de un parámetro de configuración (modelo 1 sería muy eficiente) ni modificaciones complejas que requiera la modificación de 1 o más servicios (modelo 2 sería lo más conveniente).

Un modelo híbrido podría cubrir situaciones en las que un proveedor de servicios proporcione servicios personalizados a los clientes. Por ejemplo en el caso de la VUN propuesta en este trabajo, un proveedor de servicios podría aprovisionar VUN genéricas para los nuevos usuarios que contraten el servicio. En este caso se tomaría en cuenta que las VUN estarían

formadas por VM ligeras o incluso contenedores de microservicios, entonces cuando un SP va proceder a instanciar la VUN contratada al cliente puede pasar dos cosas:

1. Las VUN sean idénticas y requieran poca personalización por lo que el SP podría optar por abordar el problema como si se tratase de una infraestructura inmutable. Se aprovisionaría una nueva VUN desde desde el repositorio de imágenes.

2. Los SPs tienen un modelo de negocio en el que los clientes pueden personalizar la VUN desde un catalogo de servicios dando lugar a VUNs personalizadas. El aprovisionamiento a través del proceso de configuración correspondiente a infraestructuras inmutables sería inviable porque habría que tener tantas imágenes de las VMs como opciones existentes en el catalogo de servicios. Y el aprovisionamiento a través del paradigma de infraestructuras mutables sería muy costoso. Sin embargo, el paradigma híbrido sería ideal para este escenario ya que permitiría partir de imágenes con configuraciones básicas (red local, dispositivos y servicios básicos) y luego aplicar los cambios necesarios para adaptar la VUN a la configuración especificada por cada cliente según las opciones indicadas por el usuario.

El caso práctico implementado a través del prototipo desarrollado permite dar una visión en profundidad del problema porque permite desplegar un escenario de aplicación para ser ser evaluado. El prototipo implementado contiene todos los componentes mostrados en la arquitectura de la Figura 39 y se han medido los tiempos que se tarda en desplegar las VMs que forman las VUN. Los tiempos se muestran en el gráfico presentado en la Figura 40 en la que se muestra el tiempo que tarda cada VM en crearse, aplicar todos los cambios e instalaciones de software requeridas, desplegarse y ponerse operativa.

Como se puede ver en el gráfico de la figura 40, VM1 es la VM que más tarda en ser desplegada debido a que contiene una imagen del SO Android y requiere de tiempo de procesamiento para cargar todos los servicios necesarios. Seguido de VM1 está VM2 que requiere de algo menos tiempo que VM1 en ser desplegada, pero más tiempo que VM3 la cual es la más rápida en ser desplegada.

La cantidad de software y servicios instalados en cada VM repercuten directamente en el tiempo que tarda cada VM en terminar el proceso de configuración de la misma. Como resultado, en el gráfico 40 se aprecia que VM1 requiere más tiempo en estar lista debido a que al integrar un SO Android en su interior, existe un gran número de servicios y configuraciones que tienen que ser procesados. Para este prototipo, VM1 es una imagen

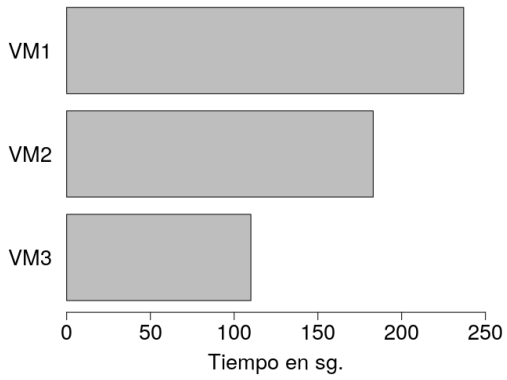

Figura 40: Gráfico de barras para comparar los tiempos consumidos por el despliegue de cada una de las máquinas virtuales (VM) configuradas para el prototipo desarrollado.

configurada previamente con el software ya instalado. Sin embargo, VM2 y VM3 parte de una imagen genérica basada en el SO Ubuntu sobre la cual se instala el software necesario. A pesar de esto, el número de servicios en VM1 es elevado requiriendo más tiempo en estar disponible que VM2 y VM3.

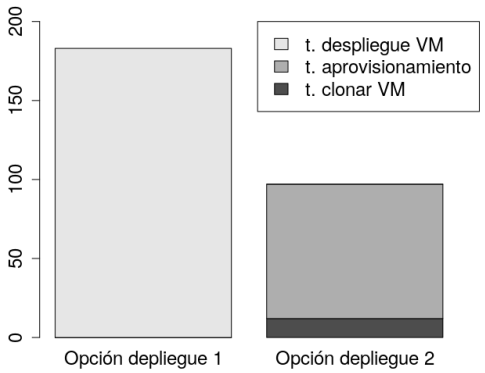

Figura 41: La figura muestra el tiempo empleado al aprovisionar la máquina VM2 al completo (Opción despliegue 1). Y también se muestra la distribución de los tiempos empleados al aprovisionar la máquina VM2 partiendo de una instancia básica preconfigurada para posteriormente instalar el SW necesario.

Durante las pruebas con el prototipo, se ha planteado la idea de, en lugar de generar desde cero las VMS para ser aprovisionadas y desplegadas, tener las imágenes con parte de la configuración básica de la máquina y a

partir de ahí clonarla y completar, posteriormente, su configuración a partir de esa configuración básica. El objetivo es medir el tiempo consumido en estas operaciones y comparar si es más rentable que tener que realizar el proceso de aprovisionar la máquina al completo desde el inicio. La Figura 41 muestra el gráfico de los tiempos consumidos en cada una de las tareas involucradas en el proceso para la VM2 donde "t.despliegue VM" representa el tiempo que se tarda en instanciar y desplegar VM2 con el software ya instalado, "t. clonar VM" es el tiempo que se tarda en clonar una VM preconfigurada con el software básico al que se le va a instalar software adicional. Finalmente, "t. aprovisionamiento" es el tiempo tomado en llevar a abo la instalación del software adicional sobre una máquina VM con configuración básica. La Figura 42 muestra el gráfico las medidas tomadas para VM3.

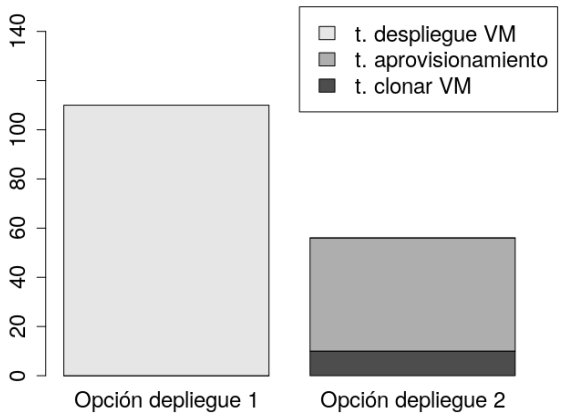

Figura 42: La figura muestra el tiempo empleado al aprovisionar la máquina VM3 al completo (Opción despliegue 1). Y también se muestra la distribución de los tiempos empleados al aprovisionar la máquina VM3 partiendo de una instancia básica preconfigurada para posteriormente instalar el SW necesario.

Al analizar las Figura 41 y 42 se puede ver que la opción de despliegue 1 requiere más tiempo frente a la opción de despliegue 2. Es decir, lleva más tiempo aprovisionar una máquina al completo que hacer un clonado de una máquina preconfigurada con las opciones básicas para después realizar cambios menores que dejen la máquina con la configuración deseada. Sin embargo, a pesar de que la opción 2 es rentable en cuestiones de tiempo, es más compleja computacionalmente y requiere tareas de mantenimiento más complicadas ya que la opción 2 será la opción preferida cuando sea necesario gestionar despliegues con cierta personalización.

7.4 CONCLUSIONES

En este capítulo se ha presentado el trabajo realizado para validar las ideas relacionadas con la generación de código y sus ventajas. La validación se ha realizado a través de la implementación y pruebas realizadas sobre un prototipo completamente funcional. El capítulo muestra los resultados obtenidos y se ha podido demostrar que las ideas presentadas en la tesis son acertadas y proporcionan beneficios en el despliegue y mantenimiento de infraestructuras de servicios.

A lo largo del capítulo, además de demostrar las capacidades de despliegue de la solución propuesta, se ha analizado diferentes planes de aprovisionamiento de máquinas que podrían mejorar los tiempos de despliegue en determinados circunstancias.

A pesar de estos resultados, se han identificado algunos aspectos que podrían ser analizados y desarrollados en trabajos futuros tales como:

- despliegue y uso de la solución propuesta sobre un entorno real para evaluar los beneficios reales sobre infraestructuras de gran tamaño. No sólo se trata de beneficios económicos procedentes de la reducción en los tiempos de despliegue, sino también beneficios provenientes de la mejora en la calidad del servicio.
- un estudio de patrones comunes del código orientado al despliegue, instalación y configuración del software más en profundidad que permita identificar patrones de configuraciones comunes. El objetivo es llevar a otras HGC la aplicación práctica de las ideas mostradas en este capítulo.

8

MECANISMOS AVANZADOS PARA OPTIMIZAR EL DESPLIEGUE DE SERVICIOS

Esta sección describe los mecanismos desarrollados en el ámbito de esta tesis para optimizar el despliegue de servicios y componentes software sobre diferentes entornos. El despliegue de servicios software no es una tarea sencilla y un despliegue correcto que tenga en cuenta los factores apropiados es importante ya que determina el rendimiento general del sistema y el adecuado consumo de recursos.

En este capítulo se presentan aspectos fundamentales que deben ser tomados en cuenta a la hora de desplegar un servicio, y también los requisitos para el tipo de servicios que va a ser desplegado, en función todo estos parámetros se deciden las condiciones relacionadas con la configuración del entorno de ejecución, la infraestructura del sistema donde va a ser desplegado, su localización geográfica y las tecnologías más adecuadas en el caso de que se decida realizar algún tipo de encapsulamiento del mismo. Este capítulo tiene su origen en algunas de las ideas generadas de la necesidad de gestionar servicios sobre múltiples entornos tecnológicos durante el desarrollo de los proyectos I+D: IRENE[1], COMMIN[2] y REMEDISS[3].

8.1 OPTIMIZACIÓN - ORQUESTACIÓN DE TECNOLOGÍAS

Esta sección describe una herramienta llamada SYStem for Managing the DEployment of Virtualized Services (SYSMDEVS) cuyo objetivo es recopilar información relevante sobre cada servicio que se desea desplegar y, en función de la información facilitada sobre la infraestructura INFORD donde se van a desplegar aquellos servicios, emplear mecanismos capaces de proponer la configuración del software de la capa inferior con el objetivo de configurarla óptimamente en función de las necesidades de los servicios que se ejecutarán sobre ella.

1 IRENE - Incentivación del reciclaje de envases con NFC en España (PT-2012-1036-370000) (Proy. Nacional)

2 COMINN - Centro comercial interactivo con interacción natural (IPT-2012-0883-430000) (Proy. Nacional)

3 REMEDISS - Red médica social sensorizada (IPT-2012-0882-430000)

De esta manera, cuando INFORD va a proceder a desplegar los servicios que componen una aplicación de usuario, SYSMDEVS evaluará y propondrá la configuración más adecuada del software ubicado por debajo de los servicios de la aplicación.

SYSMDEVS se divide en dos capas como puede verse en la Figura 43. La capa superior llamada Interface Layer (IL) es la capa que proporciona las interfaces y mecanismos necesarios para permitir a los usuarios introducir la información necesaria en el sistema. La capa IL es utilizada por dos perfiles de usuarios bien diferenciados, por un lado registra la información proveniente de los usuarios administradores de sistemas los cuales configuran parámetros correspondientes al funcionamiento interno de SYMDEVS. Es decir, los administradores del sistema ajustan parámetros, introducen reglas de acción, actualizan plantillas y configuran procesos que intervienen en la forma en que funciona la herramienta. Esta información se registra en el repositorio de configuración. Por otro lado, los desarrolladores de servicios e ingenieros de software introducen servicios software e información relacionada con ellos.

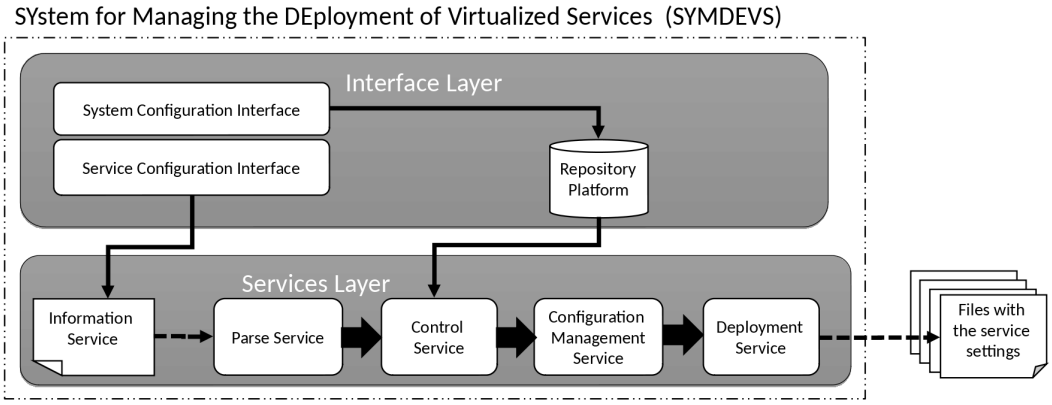

Figura 43: Arquitectura del sistema SYMDEVS y los principales componentes que forman cada uno de dichos bloques. La figura también muestra el orden secuencial de las tareas.

Para facilitar la gestión y desarrollo de esta capa, la funcionalidad de la capa se reparte en dos módulos bien definidos: el **módulo de configuración del sistema** (System Configuration Interface, SyCI) y el **módulo de configuración del servicio** (Service Configuration Interface, SeCI).

El SeCI proporciona una interfaz de acceso a los proveedores de servicios y desarrolladores para que puedan registrar sus servicios software y además añadir información relevante sobre cada uno de los servicios. La información registrada para cada uno de los servicios permitirá a SYMSDEVS determinar la configuración más óptima para su despliegue. La interfaz

SeCI facilitará a los usuarios la información a introducir a través de una interfaz guiada que presentará los valores de los parámetros que deben ser configurados para cada servicio. La interfaz esta provista principalmente de tres formularios donde cada uno está orientado a configurar los aspectos de configuración relacionados con la configuración del entorno tecnológico, selección del tipo de infraestructura y la configuración del entorno tecnológico relacionado con la virtualización. El formulario sobre el entorno de configuración muestra cada uno de los parámetros de configuración presentados en la tabla 19. De forma similar, El formulario encargada de la selección del tipo de infraestructura se compone de los parámetros mostrados en la tabla 20 y finalmente el formulario encargado de recoger los datos para los parámetros requeridos para la elección del entorno de virtualización se muestran en la tabla 21. Cada uno de los campos contenidos en cada uno los formularios podrá tener cualquiera de estos valores: "Alto", "Medio" o "Bajo" y a partir de esta información SYMSDEVS podrá inferir las configuraciones más adecuadas para cada uno de los servicios.

El módulo SyCI proporciona la capacidad a los administradores del sistema de poder añadir, eliminar o modificar los parámetros utilizados para inferir la configuración apropiada de cada servicio. Es decir, permite gestionar los parámetros que aparecen en cada uno de los formularios del módulo SeCI y de las tablas 19, 20 y 21. De esta manera se consigue que SYMDEVS pueda modificar su funcionamiento y así estar actualizado en todo momento. Una vez que la capa IL ha registrado la información ne-

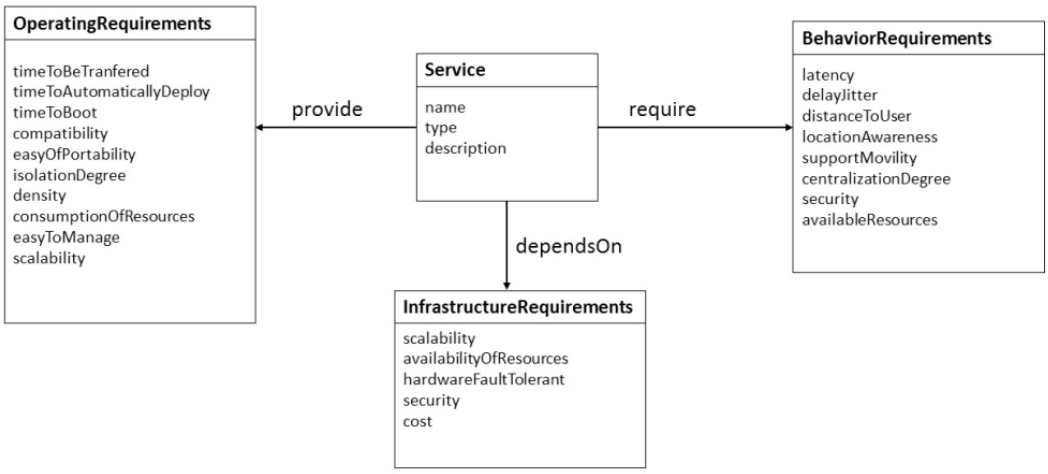

Figura 44: Modelo de datos de SYMDEVS que define la estructura de la información en el repositorio de servicios. El elemento principal del modelo es la tabla Service donde se registran los servicios disponibles del sistema. Las demás tablas relacionadas con la tabla Service son utilizadas para complementar la información de cada servicio.

cesaria para el funcionamiento de SYMDEVS, IL indica a SL que procese la configuración de los servicios deseados. IL proporciona información de cada uno de los servicios a procesar a través de una especificación y definición por servicio.

SL está compuesto por un conjunto de servicios donde cada uno se hace cargo de tareas especificas las cuales son ejecutadas de manera secuencial. El proceso comienza cuando la IL transfiere la especificación de un servicio a SL a través de la **Interfaz del Servicio de Información** (Information Service, IS) la cual la entrega al **servicio de interpretación** (Parser Service, PS), que procesa esta información y crea un modelo a través de la estructura de datos mostrado en la Figura 44. La estructura de datos contiene la tabla principal llamada Service la cual contiene información básica sobre cada servicio registrado. Por cada servicio se registra información adicional relacionando la tabla Service con las siguientes tablas :

- BehaviourRequirements. Esta tabla registra la información correspondiente con los requisitos de de funcionamiento que deben ser satisfechos por la infraestructura donde se va a desplegar el servicio para poder funcionar correctamente.
- OperatingRequirements. La tabla registra las características del servicio que estarán disponibles cuando el servicio haya sido desplegado. Estas características serán utilizadas por los orquestadores para poder comparar servicios similares registrados en el repositorio de servicios.
- InfrastructureRequirements. Esta tabla sirve para registrar requisitos de infraestructura principalmente relacionados con servicios de alta disponibilidad. Cuando un servicio requiere una infraestructura flexible y dinámica puede registrar las necesidades sobre esta tabla en la que podrá indicar si requiere mecanismos de tolerantes a fallos del hardware, posibilidad de escalado, coste máximo por uso, etc.

La información contenida en cada una de las tablas representa los campos identificados para el trabajo de esta tesis. En caso necesario se podría ampliar cada tabla con los campos que fuesen necesarios.

Una vez que la información ha sido persistida conforme al modelo de datos, ésta se envía al **Servicio de Control** (Control Service, CS) para que sea procesada. El CS recibe la información a procesar desde el **Servicio de Aprovisionamiento** (Provision Service, PS) y además esta conectado con el **Repositorio de Datos** (Repository Platform, RP) para obtener preferencias de configuración que ayudarán a definir los parámetros finales para la propuesta de la configuración final del servicio.

Con la información recibida desde PS, CS puede generar la configuración necesaria para crear la configuración de despliegue del servicio, pero antes de realizar esto CS recupera la información que hay en RP (información introducida por los administradores) y la utiliza para crear una configuración acorde a los criterios y reglas establecidas por los administradores previamente almacenadas en el repositorio de configuración. Para realizar este proceso, el CS está compuesto de varios servicios en su interior (figura 45) que evalúan estrategias antes de proporcionar un resultado final.

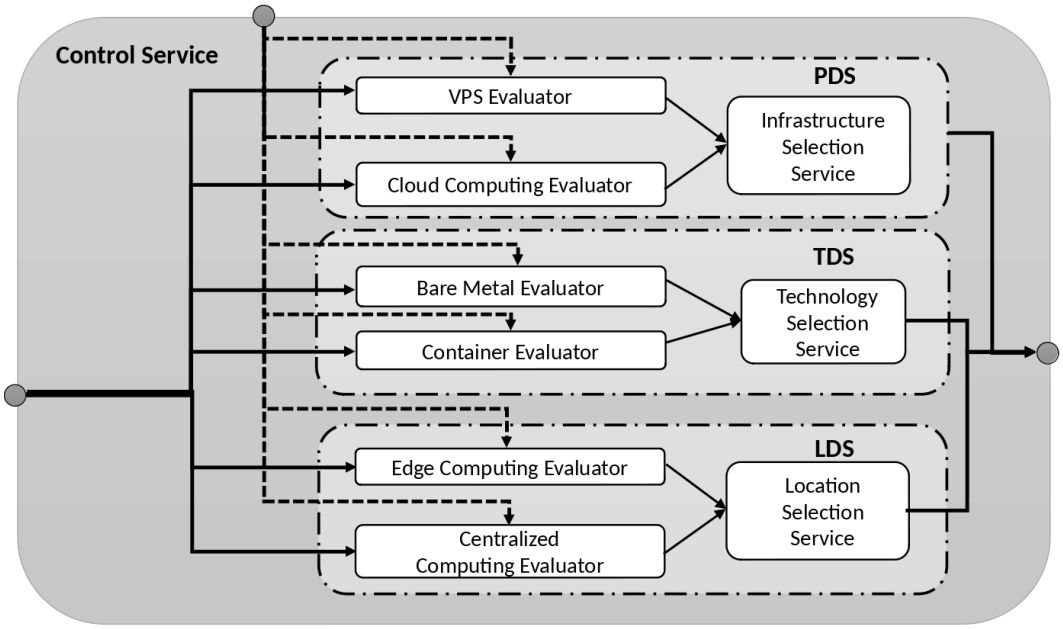

Figura 45: Diagrama de bloques que componen el servicio Control Service. Como se puede ver en la figura, el Control Service se compone de los servicios: Servicio Discriminador de Plataforma, Servicio Discriminador de Tecnologías y el Servicio Discriminador de Localización.

CS está dividido en tres bloques: el **Servicio Discriminador de Plataforma** (Platform Discriminator Service, PDS), el **Servicio Discriminador de Tecnologías** (Technologies Discriminator Service, TDS) y el **Servicio Discriminador de Localización** (Location Discriminator Service, LDS).

El PDS tiene la capacidad de determinar si un servicio debería ser desplegado en un centro de datos en la nube situado en el core de la red o debería ser desplegado en un centro de datos situado en el borde de la red. Para tomar esta decisión, el PDS cuenta con 3 componentes: el VPS Evaluator (VE), el Cloud Computing Evaluator (CCE) y el Infrastructure Selection Service (ISS). A continuación se describe cada uno:

- VPS Evaluator (VE): Este componente contiene mecanismos para evaluar lo apropiado que es un servicio para ser desplegado sobre una infraestructura basada en un modelo VPS (Private Virtual Server).
- Cloud Computing Evaluator (CCE). Este componente contiene mecanismos para evaluar la idoneidad de desplegar el servicio en una infraestructura basada en un modelo Cloud Computing. El componente se divide en dos bloque encargados de evaluar el servicio sobre dos infraestructuras en la nube bien diferenciadas: 1) nube centralizada sobre un centro de datos, gestionado por el bloque Core Computing; 2) nube descentralizada sobre infraestructuras remotas en el borde de la red, gestionado por el bloque Edge Computing.
- Infrastructure Selection Service (ISS). Este componente es el encargado de recopilar los resultados obtenidos por VE y CCE, hacer una comparación de los resultados obtenidos y en función de ello proponer la configuración más adecuada para el servicio evaluado.

El funcionamiento de PDS es sencillo, tomando en cuenta las directrices y los parámetros de configuración registrados en PR, el VE y CCE evalúan lo adecuado que sería un servicio para cada una de las infraestructuras disponibles y asignan un resultado numérico para cada una de estas opciones.

Posteriormente, ISS analiza cada puntuación y selecciona la estrategia con el valor más alto. La sección 8.3 detalla cómo se calcula la puntuación.

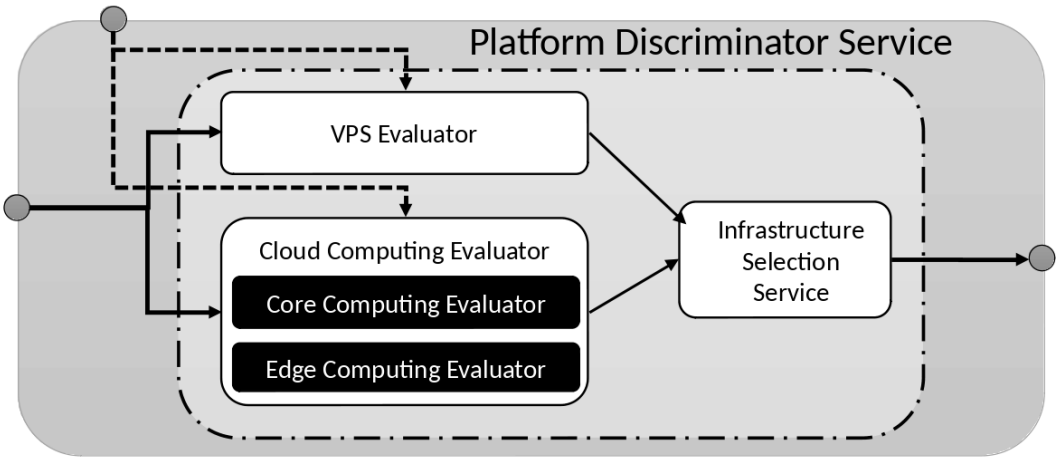

Figura 46: La figura muestra el diagrama de bloques del PDS. Además, la figura muestra los principales componentes que lo componen y las relaciones entre ellos.

El TDS es el componente encargado de determinar si un servicio debería ser desplegado sobre un entorno de virtualización o no. Para tomar

esta decisión, el TDS cuenta con 3 componentes: el Bare Metal Evaluator (BME), el Container Evaluator (CE) y el Tecnology Selection Service (TSS). A continuación se describe cada uno:

- Bare Metal Evaluator (BME). Este componente dispone de mecanismos pra evaluar lo apropiado que es un servicio para ser desplegado sobre una infraestructura basada en un modelo BareMetal.
- Container Evaluator (CE). Este componente contiene mecanismos para evaluar lo apropiado que es un servicio para ser desplegado sobre un entorno virtualizado. El componente se divide en dos bloque encargados de evaluar el servicio sobre dos tipos de virtualización bien diferenciadas: 1) la virtualización a través de VMs, gestionado por el bloque Virtualization; 2) la virtualization ligera basada en contenedores, gestionado por el bloque Light Virtualization.
- Tecnology Selection Service (TSS). Este componente es el encargado de recopilar los resultados obtenidos por BME y CE, además realiza una comparación de los resultados obtenidos y en función de ello proponer la configuración más adecuada para el servicio evaluado.

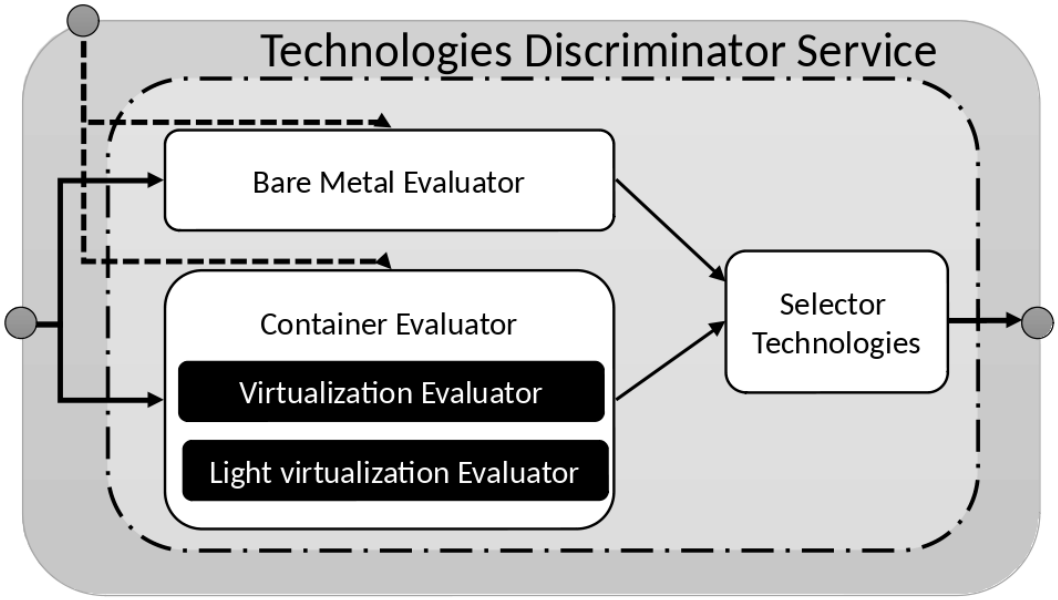

Figura 47: La figura muestra el diagrama de bloques del TDS. Además, la figura muestra los principales componentes que lo componen y las relaciones entre ellos.

El funcionamiento de TDS es muy similar al PDS, TDS utiliza la información almacenada en PR y el BME y CE evalúan lo adecuado que sería

un servicio para cada una de los entornos disponibles (virtualizados o no), como resultado de la evaluación se asigna un resultado numérico para cada una de estas opciones. Posteriormente, TSS analiza cada puntuación y selecciona la estrategia con el valor más alto. La sección 8.2 detalla cómo se calcula la puntuación.

Finalmente, el LDS es el componente que tiene el objetivo de determinar si un servicio debería ser desplegado sobre el centro de datos situado en el core de la red o en el borde de la red atendiendo a las características del servicio. Esta decisión se toma a partir de las salidas de los componentes que componen LDS: Distributed Computing Evaluator (ECE) y el Centralized Computing Evaluator (CeCE). A continuación se describe cada uno de ellos:

- Distributed Computing Evaluator (DCE). Este componente contiene mecanismos para evaluar lo apropiado que es un servicio para ser desplegado sobre una infraestructura basada en un modelo Edge Computing.
- Centralized Computing Evaluator (CeCE). Este componente contiene mecanismos para evaluar lo apropiado que es un servicio para ser desplegado sobre una infraestructura basada en un modelo Cloud Computing.
- Location Selection Service (LSE). Este componente es el encargado de recopilar los resultados obtenidos por BME y CE, , además realiza una comparación de los resultados obtenidos y en función de ello proponer la configuración más adecuada para el servicio evaluado.

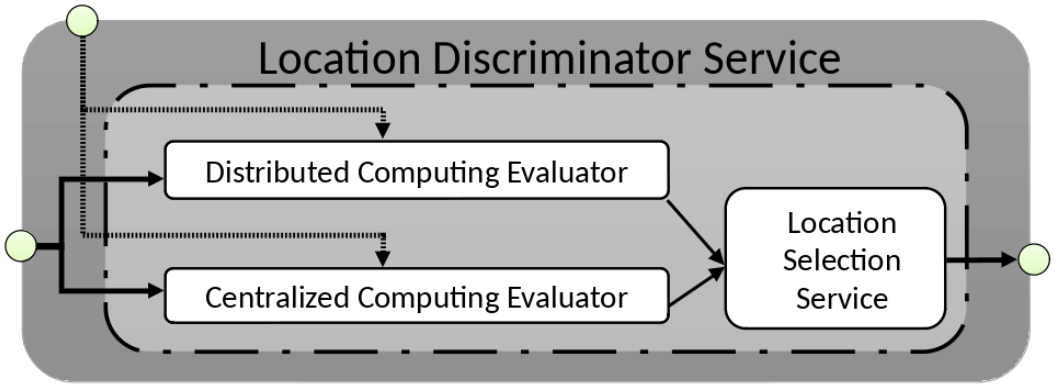

Figura 48: La figura muestra el diagrama de bloques del LDS. Además, la figura muestra los principales componentes que lo componen y las relaciones entre ellos.

LDS requiere de la información almacenada en PR para que DCE y CeCE determinen lo adecuado que sería desplegar un mismo servicio sobre un centro de datos localizado en el core de la red o en un centro de datos localizado en alguna localización cercana al usuario final. Esta decisión se toma a partir del resultado del procesamiento de la información relacionada dentro de los módulos DCE y CeCE. Posteriormente, LSS analiza cada puntuación y selecciona la estrategia con el valor más alto. La sección 8.2 detalla cómo se calcula la puntuación.

Tener en cuenta la selección de infraestructura y entorno tecnológico sobre el cual desplegar un servicio es importante ya que permite optimizar el rendimiento general del sistema y por lo tanto mejorar la calidad de la experiencia del usuario (QoE). Tener en cuenta estos aspectos facilita la gestión de los servicios permitiéndoles escalar más rápido, hacer su mantenimiento más sencillo y facilita su migración entre diferentes centros de datos, etc.

Una vez que CS ha terminado las tareas correspondientes, los resultados del proceso son enviados al **Configuration Management Service** (CMS) el cual se encarga de generar la configuración del servicio a través de un modelo de software.

La configuración del servicio puede estar basada en una implementación directa del servicio (Bare Metal) o el servicio puede ser encapsulado con tecnologías de virtualización dentro de máquinas virtuales o contenedores ligeros (depende de los resultados del TDS).

CMS contiene un conjunto de plugins los cuales le permite generar configuraciones diferentes en función de las opciones soportadas por el PDS y el TDS. Es decir, cada plugin solo es responsable de transformar las especificaciones recibidas del CS en la configuración final del software. Por ejemplo, en el caso del TDS para generar la configuración de un servicio basado en un entorno basado en VMs, el PM usará el plugin encargado de generar las configuraciones correspondientes al despliegue basado en VMs.

Como resultado final, el CMS genera un paquete completo con los archivos de configuración listos para que los procese el orquestador correspondiente. Una posible mejora del CMS podría ser extender la salida del resultado, o conectar a otro servicio complementario, de manera que pueda conectar con los interfaces de las plataformas remotas que permitan implementar la configuración final directamente sobre la infraestructura deseada.

8.2 SELECCIÓN DEL ENTORNO TECNOLÓGICO EN LA QUE DESPLEGAR EL SERVICIO

Como se ha visto en la sección anterior, el TDS es capaz de analizar las características del servicio y decidir qué entorno tecnológico de ejecución es el más adecuado para los servicios a desplegar. Esta sección muestra los mecanismos utilizados por el TDS para seleccionar dicho entorno.

Durante el análisis realizado en este trabajo se han identificado tres entornos tecnológicos bien diferenciados cada uno de ellos con características bien diferenciadas. Esto permite poder escoger el entorno más adecuado a las necesidades de cada escenario. Los posibles entornos tecnológicos sobre los que desplegar los servicios en el ámbito de este trabajo son:

- **Bare Metal**: Este entorno se caracteriza por la instalación del servicio directamente sobre el sistema operativo. Este modelo se caracteriza por un modelo de instalación de software tradicional.
- **Virtualización basada en VMs**: Este entorno se caracteriza por la instalación del servicio/aplicación en un sistema operativo encapsulado en una máquina virtual.
- **Virtualización ligera**: Al igual que el entorno de virtualización, el entorno de virtualización ligera encapsula el servicio/aplicación en un entorno virtualizado, pero en este caso la virtualización se limita al sistema operativo sin necesidad de virtualizar hardware (estos modelos se discuten en la sección 2.2).

CS recopila las necesidades y características registradas en el PR correspondientes al servicio a evaluar y aplica un algoritmo para obtener un resultado numérico por cada una de los entornos tecnológicos recogidos en las columnas de la tabla 19. El algoritmo compara una a una las características recogidas en la tabla 19 con los requisitos del servicios a evaluar. Como resultado el algoritmo asigna un valor numérico a cada entorno tecnológico. El entorno tecnológico que tenga el valor más alto es la opción recomendada por SYSDEMVS.

SYSDEMVS modela las necesidades del servicio en función de las características requeridas por el servicio a evaluar. Esta lista de características se modelan en forma de vector como (1) donde c_i representa cada una de las características que definen las necesidades del servicio.

$$C = c_1, c_2, ..., c_n \tag{1}$$

	Bare metal	**VM**	**Virtualización ligera**
T. para ser transferido	Alto	Medio	Bajo
T. para despliegue automático	Alto	Medio	Bajo
T. de arranque	Alto	Medio	Bajo
Compatibilidad	Alto	Medio	Bajo
Fácilmente portable	Bajo	Medio	Alto
Grado de aislamiento	Alto	Medio	Bajo
Densidad	Bajo	Medio	Bajo

Tabla 19: Comparación de las diferentes características que tienen cada uno de los entorno tecnológico tomados en cuenta en este trabajo. "T." indica tiempo

El vector (2) recoge las características que definen el entorno tecnológico Bare metal que podría utilizarse para desplegar dicho servicio. La lista de característica que hacen referencia al entorno tecnológico Bare metal se muestra en la tabla 19. En el vector (2), cada b_i representa una de las características del entorno Bare metal.

$$B = b_1, b_2, ..., b_n \tag{2}$$

A partir de los vectores de características C y B, el componente CS aplica la expresión matemática (3) para obtener un resultado numérico que indica el número de características que son cumplidas para el servicio que está siendo evaluado. Este resultado queda almacenado en Op_1. El parámetro w_i permite realizar ajustes en los cálculos de la ecuación permitiendo asignar pesos para cada c_i evaluada.

$$Op_1 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = b_i \\ 0, & \text{otherwise} \end{cases} | \tag{3}$$

Una vez que se ha calculado el valor de la opción 1 (Op1), CS procede a calcular el valor de la opción 2 (Op_2) que se corresponde al uso de entorno tecnológicos basados en VMs. Para este objetivo, la lista de características que modelan la virtualización por VMs viene definido por el vector (4), donde v_i representa cada una de las características de un entorno tecnológico de despliegue basado en un modelo de VMs

$$V = v_1, v_2, ..., v_n \tag{4}$$

Al igual que en el caso anterior, a partir de los vectores de características C y V, el componente CS aplica la expresión (5) para obtener un resultado numérico donde se indica el número de características que son cumplidas para el servicio que está siendo evaluado. Este resultado queda almacenado en la variable Op2.

$$Op_2 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = v_i \\ 0, & \text{otherwise} \end{cases} | \tag{5}$$

Finalmente, aplicaríamos el mismo proceso para calcular el valor numérico correspondiente a la opción 3 (Op3) la cual se corresponde con el uso de entornos tecnológicos en virtualización ligera, es decir basada en contenedores. En este caso el vector de características (6) donde l_i representa cada una de las características de un entorno tecnológico de despliegue basado en un modelo de virtualización ligera a través de contenedores.

$$L = l_1, l_2, ..., l_n \tag{6}$$

CS comparará los vectores de características de C y L con el objetivo de obtener una evaluación con formato numérico que indique el grado de satisfacción de características para el entorno tecnológico de virtualización ligera. El resultado será registrado en Op3 después de aplicar la expresión (7).

$$Op_3 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = l_i \\ 0, & \text{otherwise} \end{cases} | \tag{7}$$

Una vez que CS ha evaluado todas los entornos tecnológicos disponibles tan sólo debe comparar los resultados de las diferentes opciones disponibles y notificar el resultado final al siguiente servicio que sería el PMS.

8.3 SELECCIÓN DE LA INFRAESTRUCTURA DE DESPLIEGUE

Esta sección muestra los mecanismos del PDS para obtener la infraestructura óptima entre varias , de cara al despliegue de un servicio determinado. En este trabajo se han identificado dos posibles infraestructuras a tener en cuenta:

	VPS	**Cloud Computing**
Escalabilidad y flexibilidad	Bajo	Alto
Disponibilidad de recursos	Alto	Bajo
Hardware tolerante a fallos	Bajo	Alto
Seguridad	Alto	Bajo
Coste	Bajo	Alto

Tabla 20: Comparación entre las diferentes características que tiene cada una de las infraestructuras contempladas en este trabajo, la tabla muestra las fortalezas y debilidades de cada una de ellas.

- **Infraestructura basada en servidores privado virtuales** (VPS): Esta infraestructura se caracteriza por tener todos los recursos físicos disponibles para los servicios. Esta infraestructura es conocida también como infraestructura Bare Metal.
- **Infraestructura basada en la nube**: Esta infraestructura se caracterizada por una capa que gestiona los recursos de infraestructura y los asigna a instancias de ejecución donde los servicios y aplicaciones son ejecutados y gestionados.

CS recopila las necesidades y características registradas en PR correspondientes al servicio a evaluar y aplica un algoritmo para obtener un resultado numérico por cada una de las infraestructuras recogidos en las columnas de la tabla 20. El algoritmo compara una a una las características recogidas en la tabla 20 con los requisitos del servicios a evaluar. Como resultado el algoritmo asigna un valor numérico a cada infraestructura. La infraestructura que tenga el valor más alto es la opción recomendada por CS.

SYSDEMVS modela las necesidades del servicio en una lista de características que modelan las preferencias de funcionamiento requeridas por el servicio a evaluar. Esta lista de características se modelan en forma de vector definido por (8) donde c_i representa cada una de las características que recoge las necesidades del servicio.

$$C = c_1, c_2, ..., c_n \tag{8}$$

Posteriormente se crea el vector que define las características que definen la opción de infraestructura basada en VPSs que podría utilizarse para desplegar dicho servicio. La lista de características que hacen referencia a la infraestructura VPS viene modelada por (9) donde pi representa cada una de las características de una infraestructura de despliegue basado en

VPS.

$$P = p_1, p_2, ..., p_n \tag{9}$$

A partir de los vectores de características C y P, el componente CS aplica la expresión (10) siguiente para obtener un resultado numérico que indica el número de características que son satisfechas para el servicio que está siendo evaluado. Este resultado queda almacenado en la variable Op4.

$$Op_4 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = p_i \\ 0, & \text{otherwise} \end{cases} | \tag{10}$$

Una vez que se ha calculado el valor de la opción 4 (Op4), CS procede a calcular el valor de Op5 que se corresponde al uso de infraestructuras basadas en la computación en la nube. Para este objetivo la lista de características que modelan la infraestructura en la nube viene definido por (11) donde z_i representa cada una de las características de una infraestructura de despliegue basado en la nube.

$$Z = z_1, z_2, ..., z_n \tag{11}$$

Al igual que en el caso anterior, a partir de los vectores de características C y Z, el componente CS aplica la expresión (12) para obtener un resultado numérico donde se indica el número de características que son satisfechas para el servicio que está siendo evaluado. Este resultado queda almacenado en la variable Op5.

$$Op_5 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = z_i \\ 0, & \text{otherwise} \end{cases} | \tag{12}$$

Una vez que CS ha evaluado todas los modelos de infraestructura disponibles tan sólo debe comparar los resultados de las diferentes opciones y notificar el resultado final al siguiente servicio que sería el PMS.

8.4 SELECCIÓN DEL TIPO DE COMPUTACIÓN CENTRALIZADA VS COMPUTACIÓN EN EL BORDE

En esta sección se va abordar la toma de decisión referente a desplegar un servicio sobre un modelo centralizado o en un modelo descentralizado

	Computación centralizada	**Computación en el borde**
Latencia	Alto	Bajo
Retardo jitter	Alto	Bajo
Distancia al usuario	Alto	Bajo
Consciente con la localización	Bajo	Alto
Soporte a la movilidad	Bajo	Alto
Grado de centralización	Alto	Bajo
Seguridad	Bajo	Alto

Tabla 21: Comparación entre las diferentes características que tiene cada una de las localizaciones contempladas en este trabajo, la tabla también muestra las fortalezas y debilidades de cada una de ellas.

en el borde de la red, esto es un aspecto importante a la hora de desplegar un servicio ya que puede influir gran medida en la calidad del servicio prestado.

Este trabajo ha considerado relevante tomar en cuenta los requisitos y necesidades que forman parte del proceso de selección a la hora de escoger la localización de la infraestructura para desplegar un servicio. En este trabajo se han identificado dos localizaciones que vienen recogidas en la tabla 21 junto a las principales características que las definen.

CS recopila las necesidades y características registradas en PR correspondientes al servicio a evaluar y aplica un algoritmo para obtener un resultado numérico por cada modelo de infraestructura disponible en las columnas de la tabla 21.

El algoritmo compara una a una las características recogidas en la tabla 21 con los requisitos del servicios a evaluar. Como resultado el algoritmo asigna un valor numérico a cada modelo de infraestructura. El modelo de infraestructura que tenga el valor más alto es la opción recomendada por CS.

SYSDEMVS modela las necesidades del servicio en una lista de características las cuales reflejan las preferencias de funcionamiento requeridas por el servicio a evaluar. Esta lista de características se modelan en forma de vector definido en (13) donde c_i representa cada una de las características que recoge las necesidades del servicio.

$$C = c_1, c_2, ..., c_n \tag{13}$$

Además, se crea el vector que define las características que definen la opción de infraestructura de computación centralizada que podría utilizarse para desplegar dicho servicio. La lista de característica que hacen referencia a esta infraestructura vienen modelada por (14) donde g_i representa cada una de las características del modelo de infraestructura centralizada.

$$G = g_1, g_2, ..., g_n \tag{14}$$

A partir de los vectores de características C y G, el componente CS aplica la expresión (15) para obtener un resultado numérico que indica el número de características que son cumplidas para el servicio que está siendo evaluado. Este resultado queda almacenado Op6.

$$Op_6 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = g_i \\ 0, & \text{otherwise} \end{cases}| \tag{15}$$

Una vez que se ha calculado el valor de Op6, CS procede a calcular el valor Op7 que se corresponde al uso de infraestructuras descentralizadas en el borde de la red. Para este objetivo la lista de características que modelan la infraestructura en la nube viene definido por (16), donde m_i representa cada una de las características de una infraestructura de despliegue basado en la nube.

$$G = g_1, g_2, ..., g_n \tag{16}$$

Al igual que en el caso anterior, a partir de los vectores de características C y M, el componente CS aplica la siguiente expresión matemática para obtener un resultado numérico donde se indica el número de características que son cumplidas para el servicio que está siendo evaluado. Este resultado queda almacenado en la variable Op7.

$$Op_7 = \sum_{i=1}^{n} |w_i \begin{cases} 1, & \text{if } c_i = m_i \\ 0, & \text{otherwise} \end{cases}| \tag{17}$$

Una vez que CS ha evaluado todas los modelos de infraestructura disponibles tan sólo debe comparar los resultados de las diferentes opciones y notificar el resultado final al siguiente servicio que sería el PMS.

8.5 CONCLUSIONES

En este capítulo se ha presentado unos mecanismos para optimizar el despliegue de componentes software sobre sistemas TI. El capítulo ha mostrado como desplegar un servicio no es una tarea sencilla y se deben considerar diversos factores antes de realizar el proceso. En el capítulo se ha presentado una arquitectura y el modelo de datos que recoge la información necesaria a la hora de realizar el proceso de evaluación.

La solución propuesta considera tres aspectos fundamentales a evaluar: 1) la plataforma sobre la cual se va a desplegar el servicio, 2) el tipo de tecnología de virtualización con la que se encapsulara el servicio en cuestión (o tal vez no sea necesario utilizar algún mecanismos de virtualización), y 3) la localización idónea del servicio, es decir, si sería conveniente desplegarlo en el core de la red o en algún centro de datos situado en el borde de la red.

En el capítulo se han mostrado los principales factores a ser considerados y además se ha presentado una arquitectura y el proceso de evaluación a seguir para indicar si un servicio es apropiado para un tipo de arquitectura o no.

9

VALIDACIÓN MEDIANTE PROTOTIPO

En este capítulo se presenta un prototipo para evaluar parte del trabajo propuesto en esta tesis. El prototipo mostrado en este capítulo implementa la arquitectura presentada en el capítulo 5 y verifica los planteamientos realizados en el capítulo 8 para optimizar tecnologías empleadas sobre un mismo caso práctico. En la sección 9.3 se evalúa la arquitectura propuesta en 5 a través de los requisitos identificados en la sección 2.1.6 para comprobar si el diseño de la arquitectura cumple con las necesidades identificadas. La sección 9.1 presenta un prototipo en un escenario de prueba donde se describe la implementación de la arquitectura y los principales componentes desarrollados. Las medidas recogidas por las pruebas realizadas del prototipo fueron analizadas y discutidas en la publicación [9].

9.1 PROTOTIPO DE ESCENARIO DE PRUEBA

Para el desarrollo del prototipo de evaluación se ha desarrollado parte de la arquitectura de INFORD sobre la cual se desplegarán algunos servicios diseñados para la distribución de contenidos multimedia. La Figura 49 muestra un diagrama de la ubicación y la conexión de los componentes. El prototipo desarrollado consiste en un **Servicio de Vídeo Inteligente** (Smart Video Service, SVS) que gestiona a los usuarios, proporciona el catálogo multimedia y transmite vídeo. Los usuarios interactúan con el SVS a través de una interfaz gráfica (GUI) basada en plantillas web responsivas.

Cuando un usuario solicita un vídeo a través de la GUI, el **Componente Controller** (CC) recibe una solicitud HTTP+XML/JSON con datos en el cuerpo con formato JSON, esos datos tienen una estructura JSON que contiene metadatos con información sobre el reproductor tales como la resolución, formatos soportados, etc. El CC se conecta al **Componente Base de Datos** (CBD) para obtener patrones de preferencias de uso, es decir, el CBD registra visualizaciones junto a las características del vídeo a reproducir tales como el bitrate, resolución, etc. y además las características del reproductor que los ha reproducido para poder inferir automáticamente cúal sería la configuración de la reproducción más apropiada en función de la información registrada previamente. A continuación, CC recupera el vídeo tomando en cuenta la información proporcionada por el CBD, es decir, el vídeo con las características más apropiadas en función del reproductor

que se va a utilizar para reproducirlo y, por último, el vídeo se entrega al usuario final a través del **Componente Web Server** (CWB) el cual es un servidor web que es responsable de entregar los vídeos y de proporcionar una interfaz web a los usuarios.

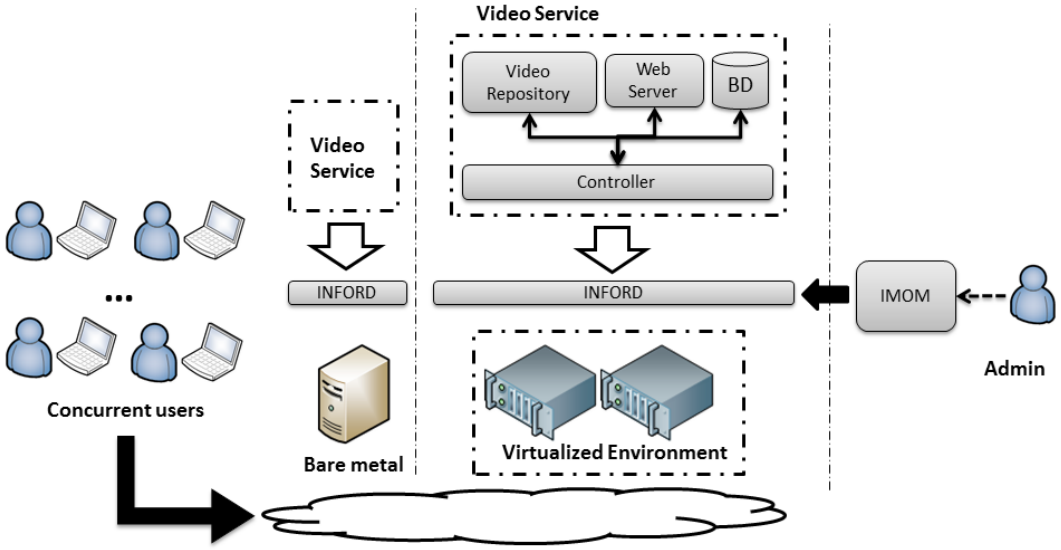

Figura 49: Arquitectura para probar la solución propuesta. El servicio de vídeo se implementa a través de IMOM. IMOM se puede implementar sobre la infraestructura Bare metal o virtualizada. Los usuarios consumen los servicios simultáneamente. El administrador planifica la implementación de SVS a través de IMOM y finalmente la implementa a través de INFORD.

El SVS se ha definido a través de las plantillas de configuración en la capa de datos IMOM. La configuración de la plantilla permite implementar el SVS en diferentes entornos de ejecución.

9.2 FUNCIONAMIENTO DEL PROTOTIPO

A diferencia de otros trabajos, la solución propuesta no mejora el rendimiento de los servicios o aplicaciones que se despliegan en la arquitectura propuesta. El objetivo es mejorar la compatibilidad, la gestión y el despliegue de aplicaciones y servicios. Las aplicaciones y servicios a desplegar deberían ser optimizadas individualmente por el desarrollador, siendo el objetivo de esta solución optimizar el despliegue sin modificar la programación original de la aplicación a desplegar . Por este motivo, esta sección presenta una comparación del funcionamiento de un servicio en diferentes entornos de prueba. Para ello se han definido tres tipos de entornos de prueba. El primer entorno de prueba (B) consta de un servidor para el despliegue en Bare metal, el segundo entorno de prueba (L) consta de

una capa de virtualización ligera (como el contenedor de Linux [44]) y, por último, el tercer entorno de prueba (D) también consta de una capa de virtualización ligera orientada a microservicios (por ejemplo, Docker [44]).

Para analizar cómo funciona el prototipo, hemos utilizado la herramienta de benchmarking de código abierto JMeter [211] para medir los tiempos de respuesta y el tiempo de conexión de cada solicitud variando el número de usuarios simultáneos. La prueba realizada implica la conexión y visualización de contenido multimedia por parte de cada usuario simulado. La prueba simula 125 usuarios simultáneos que estaban realizando solicitudes progresivamente alcanzando su carga máxima en 20 segundos. Hemos medido la carga del sistema de acuerdo con el tiempo necesario para establecer cada conexión. El objetivo es analizar cómo varían los tiempos de conexión aumentando los usuarios que acceden al servicio.

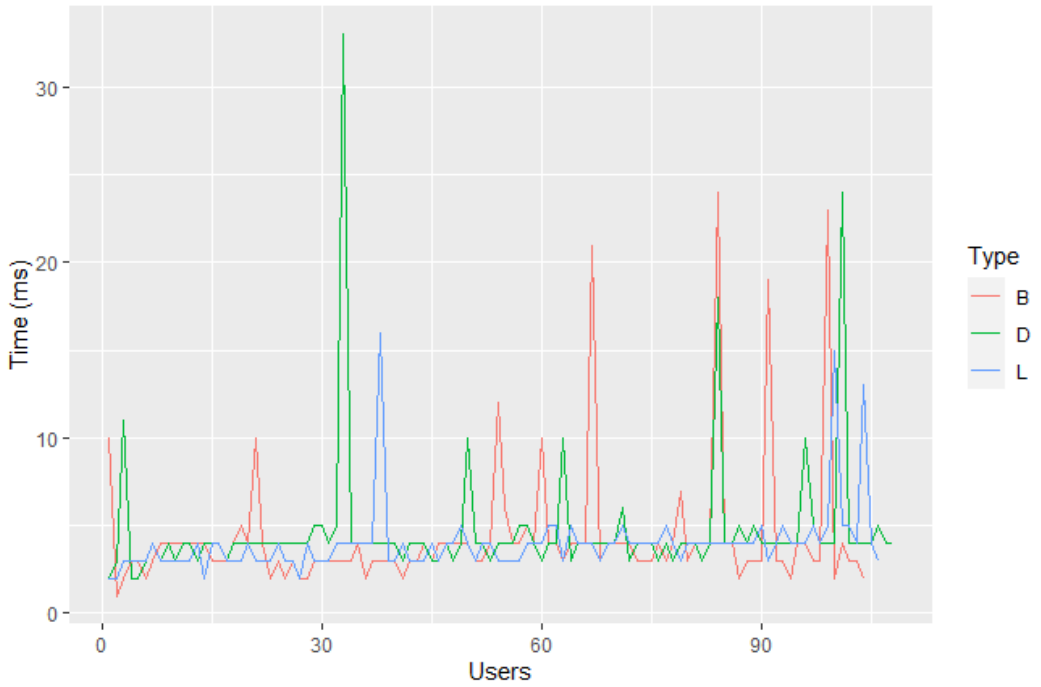

Figura 50: Comparación de los tiempos de conexión obtenidos en las pruebas para cada infraestructura en función del número de usuarios. Nomenclatura: B es infraestructura Bare metal. L es la infraestructura de contenedores de Linux y D es una infraestructura de contenedores de microservicios.

La Figura 50 muestra los tiempos de conexión de cada uno de los entornos para la comparación del rendimiento. Como se puede ver en la figura de la 50, las medidas obtenidas para cada uno de los entornos son muy similares. Para analizar los resultados desde una perspectiva diferente, se

ha creado el histograma de la Figura 51 donde se muestra el histograma de los tiempos de conexión obtenidos para cada entorno de prueba. El lector puede ver cómo se resuelven la mayoría de las solicitudes antes de 10 msg y también las medidas obtenidas de los diferentes entornos de prueba no son muy diferentes.

El entorno B presenta los mejores resultados con tiempos ligeramente más cortos que los entornos virtualizados D y L. Este resultado se esperaba porque, a pesar de la optimización de los entornos de virtualización, la capa de virtualización que encapsula el software dentro introduce un pequeño retardo en las tareas de procesamiento. El entorno L es ligeramente más rápido que el entorno D. Creemos que puede deberse a las características del software utilizado para la virtualización. Somos conscientes de que hay muchas opciones de virtualización, pero el objetivo de este trabajo no es medir el rendimiento de cada una de las opciones posibles. Nuestro objetivo es verificar la flexibilidad para poder implementar servicios en diferentes entornos. Y también se ha podido comprobar el comportamiento del mismo servicio en diferentes entornos.

Las conclusiones de rendimiento obtenidas por estas pruebas pueden ser utilizadas por los arquitectos de software para decidir qué entorno es el más adecuado para implementar servicios. Para esta decisión, la evaluación del rendimiento es un factor importante, pero también habrá que tener en cuenta las características y propiedades de cada entorno de tiempo de ejecución. Debido a que cada tecnología de virtualización proporciona propiedades especiales y, por lo tanto, cada una de ellas es ideal para situaciones y requisitos específicos.

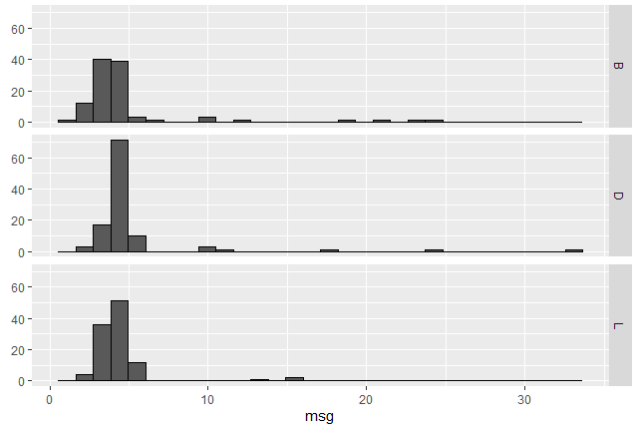

Figura 51: Histogramas de los tiempos de conexión para cada infraestructura probada. Nomenclatura: B es infraestructura Bare metal. L es la infraestructura de contenedores de Linux y D es una infraestructura de contenedores de microservicios.

	Descripción	**Solución propuesta**
R1	Mecanismos para gestión de servicios.	Sí
R2	Catalogo de servicios.	Sí
R3	Soporte Multiplataforma.	Sí
R4	Encapsulado de servicios.	Sí
R5	Empaquetado de servicios.	Sí
R6	Soporte para la configuración de enlaces virtuales.	Sí
R7	Softwarización de cualquier tipo de funcionalidad.	Sí
R8	Mecanismos de interoperabilidad con otros sistemas.	Sí
R9	Gestión de recursos de infraestructura.	Sí
R10	Soporte/ Herramientas para desplegar y configurar software de manera desatendida.	Sí
R11	Administración del ciclo de vida de los servicios	Sí
R12	Adaptar/configura infraestructura en función de los requisitos de funcionamiento de los servicios.	Sí
R13	Gestionar encadenamiento de servicios.	Sí
R14	Mecanismos para optimizar el despliegue de servicios.	Sí
R15	Curva de aprendizaje moderada	Sí
R16	Requisitos de funcionamiento mínimos	Sí

Tabla 22: Requisitos cumplidos por la solución propuesta en este trabajo de tesis.

9.3 ADECUACIÓN DE LA SOLUCIÓN PROPUESTA A LOS REQUISITOS

En este apartado se va a proceder a comentar la medida en la que la solución propuesta en esta tesis cumple con los requisitos establecidos en la tabla 1 y las soluciones que aporta a los problemas expuestos en los diferentes apartados del capítulo 2 y 3. En el capítulo 5, se ha presentado un modelo de arquitectura y varios mecanismos que ayudan al funcionamiento y ciclo de vida de los servicios que corren sobre la arquitectura propuesta.

Para comprobar el grado de cumplimiento de la solución propuesta se ha tomado la tabla 1 para verificar los requisitos que se cumplen con la solución propuesta. En la tabla 22 se muestra los requisitos que son satisfechos y a continuación se describe el grado de cumplimiento de cada uno de los requisitos.

1. R1. Este requisito recoge la necesidad de gestionar servicios de una plataforma de servicios permitiendo que cada servicio se pueda añadir, borrar, modificar, etc. A través del modelo de arquitectura mostrado en el capítulo 5 este requisito se cumple ya que IMOM integra un orquestador de servicios compuesto por los servicios Provision Service (PS) y el Configuration Service (CS) encargados de desplegar, registrar, modificar o eliminar los servicios.

2. R2. El requisito especifica la necesidad de disponer de un catalogo de servicios que permita gestionar los servicios de forma similar a un repositorio de servicios. Al igual que en R1, este requisito es satisfecho a través de la arquitectura mostrada en el capítulo 5 la cual dispone de dos tipos de catálogos de servicio: 1) el catalogo contenido en IMOM sobre la capa BSS el cual es accedido principalmente por los SPs y 2) los catálogos dentro de INFORD alojados en el componente Application Provisioning Service (APS). Debido a la existencia de estos repositorios de servicios se da por satisfecho este requisito.

3. R3. El requisito número 3 define la necesidad de que los servicios sean capaces de funcionar sobre diferentes infraestructuras independientemente de la tecnología base que se utilice. Este requisito es satisfecho en este trabajo ya que el diseño de la arquitectura presentada en el capítulo 5 no impone requisitos que no puedan cumplir otras infraestructuras actuales y/o futuras limitando así el despliegue sobre ellas. Además, este trabajo propone el uso de tecnologías de virtualización que permiten encapsular los servicios dentro de máquinas virtuales y contenedores lo que permitiría desplegar estos contenedores o VMs sobre múltiples infraestructuras sin problemas de incompatibilidad de los servicios encapsulados. Finalmente, el uso del paradigma de la Infraestructura como Código también proporciona mecanismos para ofrecer soporte multiplataforma ya que permite desplegar un mismo servicio sobre diferentes infraestructuras instalando los servicios y los componentes software que requieran. Por lo tanto, se considera que este requisito es satisfecho ya que el trabajo propuesto en esta tesis es capaz de proporcionar independencia a los servicios frente al software de la infraestructura.

4. R4. Este requisito indica la necesidad de encapsular los servicios para facilitar el despliegue y la portabilidad de servicios entre diferentes proveedores de infraestructuras y también asegurar la interoperabilidad. La solución propuesta en el capítulo 6 propone mecanismos para gestionar la encapsulación a través de máquinas virtuales (sección 2.2) y con herramientas de gestión de la configuración analizadas en el estado del arte en la sección 3 del capítulo 3.2.

5. R5. Este requisito describe la necesidad de empaquetar o encapsular las librería o software de terceros necesarios para que un servicio pueda funcionar correctamente. El objetivo es asegurar que el servicio disponga de las dependencias software necesarias para asegurar el correcto funcionamiento de los servicios. En este trabajo, la manera para asegurar las dependencias de los servicios es través de la especificación de las mismas en el registro de los servicios cuando son dados de alta en IMOM 5.2.2. Durante el despliegue, el orquestador de IMOM formado por Provision Service (PS) y el Configuration Service (CS) gestionan resolución de dependencias antes de proceder a la instalación de los servicios asegurando la instalación de todo el software complementario. El proceso completo de despliegue e instalación de los servicios junto a sus dependencias se realiza a través de herramientas de gestión de la configuración por lo que se asegura el cumplimiento de este requisito.

6. R6. Este requisito insta a trabajar preferiblemente con conexiones y enlaces virtuales para conectar los diferentes servicios y componentes software sobre las infraestructuras. Conforme a las tecnologías mostradas en el capítulo 2, concretamente descritas en la sección 2.3.2 en las que se ponía de manifiesto que las tecnologías basadas en redes definida por software (SDN) para proporcionar conectividad a través de software permitía crear sistemas flexibles, adaptables y robustos a través de mecanismos software. En el trabajo propuesto en esta tesis se ha tenido en cuenta este requisito durante la fase de diseño y tal como se describe en el capítulo 6. La configuración de las redes necesarias se realiza a nivel de infraestructura, concretamente en la capa de infraestructura descrita en la sección 6.1.1 donde se interconectan cada uno los nodos al ser desplegados sobre la infraestructura correspondiente.

7. R7. El requisito propone en digitalizar cualquier dispositivo que provea de algún servicio. Este paradigma de virtualizar cualquier dispositivo físico ha sido analizado en la sección 2.3 y se ha tomado en cuenta en este trabajo a través de los mecanismos diseñados para virtualizar y desplegar servicios software tomando en cuenta las diferentes capas software 34 que pueden estar implicadas en las labores realizadas por un dispositivo.

8. R8. Este requisito describe la necesidad de disponer de mecanismos que permitan la comunicación con otros sistemas para el intercambio de información para múltiples propósitos. De las principales tecnologías analizadas en el capítulo 2, la tecnología basada en la nube es

una de las tecnologías más afectadas por este requisito y por esta razón se analizaron aquellas soluciones en la sección 2.4. Debido a la importancia de este requisito, el trabajo realizado en esta tesis recoge la posibilidad de poder conectar con diferentes proveedores de la nube registrando los conectores que permiten acceder a sus recursos. Esta información se recogen el modelo de datos 35 utilizado para gestionar las configuraciones de la capa de Infraestructura de la sección 6.1.1. De esta manera este requisito es contemplado en las soluciones propuestas en esta tesis.

9. R9. Este requisito indica que hay que proporcionar la capacidad de gestión de recursos de infraestructura para permitir asignar los recursos necesarios a la hora de instanciar la infraestructura necesaria. Este requisito se ha tenido en cuenta durante el diseño de la arquitectura propuesta en 5 y en la sección 6.1.1 se describe la capa infraestructura que proporciona los mecanismos necesarios para satisfacer este requisito.

10. R10. El requisito número 10 recoge la necesidad de automatizar al máximo el despliegue software de manera automática. Las tecnologías que permiten realizar esto han sido analizadas en la sección 3.2 y son utilizadas por el orquestador de IMOM 5.2.2 para realizar el despliegue del software deseado. En el capítulo 7 puede validarse este requisito a través del prototipo implementado demostrando que el requisito ha sido satisfecho.

11. R11. El requisito propone el uso de mecanismos que permitan gestionar el ciclo de vida de los servicios desplegados en una arquitectura de servicios. La solución para la configuración de infraestructuras por capas de software presentado en 6.1 contempla el uso de plataformas o entornos de ejecución las cuales permiten la gestión de software en el entorno de ejecución que proporcionan. La capa de plataforma descrita en 6.1.3 es la encargada de satisfacer este requisito y en el prototipo desarrollado en el capítulo 7 se muestra como es satisfecho.

12. R12. Este requisito especifica la necesidad de poder adaptar aspectos de configuración del entorno de ejecución de un servicio antes de que este sea desplegado sobre dicho entorno. Esta situación puede darse a la hora de mover servicios entre diferentes infraestructuras las cuales tienen algunas diferencias técnicas entre ellas. En el trabajo desarrollado en esta tesis se propone utilizar la gestión del software a través de las diferentes capas software presentadas en el capítulo 6. Concretamente la Capa de Plataforma 6.1.3 ofrece mecanismos y el modelo de datos necesario, mostrado en la Figura 36, para poder

adaptar las configuraciones a los servicios que se desplieguen sobre el entorno de ejecución. Debido a esto R12 sería satisfecho por la solución propuesta en esta tesis.

13. R13. El requisito propone la capacidad de orquestar y componer servicios complejos a través de servicios más simples para poder automatizar su configuración. En la sección 2.1.4 se analizó la importancia de las arquitecturas de encadenamiento de servicios y esto se ha tratado de reflejar en la arquitectura propuesta en esta tesis a través de la arquitectura presentada en el capítulo 5. El proceso de encadenar servicios se gestiona desde el orquestador de IMOM 5.2.2 el cual es capaz de componer la cadena de servicios desde las especificaciones registradas por el proveedor de servicios en el momento que introduce la información de los servicios que van a componer su arquitectura de servicios.

14. R14. Este requisito propone el uso de utilizar mecanismos que permitan optimizar el funcionamiento general de las arquitecturas de servicios a través de técnicas de optimización. A través de este requisito se consigue que con pequeñas optimizaciones sobre los servicios desplegados se obtiene un ahorro considerable a nivel de rendimiento sobre la infraestructura del proveedor de nube. Esto se ha tomado en cuenta en el desarrollo de la tesis y en el capítulo 8 se ha propuesto una solución que permite introducir mejoras de funcionamiento global de los servicios a desplegar a través de varias estrategias de optimización sobre el bloque INFORD de la arquitectura presentada en la Figura 32. Estas estrategias se describen en las secciones 8.3, 8.2 y 8.4.

15. R15. Este requisito propone usar tecnologías fáciles de usar que no requieran un proceso de aprendizaje largo. Para abordar este problema, en la sección 3.2 se analizaron diferentes herramientas orientadas a la gestión de la configuración y en la sección 3.2.3 se mostró un análisis que mostraba qué herramientas eran las más usadas. La solución propuesta en 4 permite que el orquestador de IMOM pueda trabajar con varios tipos de herramientas de gestión de la configuración, permitiendo que los administradores escojan las opciones con las que más cómodos se encuentren.

16. R16. Este requisito insta a que las configuraciones de las arquitecturas de servicios estén correctamente configurados y optimizados para minimizar el consumo computacional y por lo tanto energético. El trabajo propuesto en esta tesis aborda el problema de las configuraciones óptimas en la sección 8 y la reducción y duplicación de librerías

software en la sección 4.4. Además, el hecho de usar la arquitectura presentada en el capítulo 5 ya contribuye a usar infraestructuras configuradas a medida en lugar de usar infraestructuras de propósito general.

17. R17. El requisito expone la necesidad de usar mecanismos que permita a los proveedores de servicios ofrecer a sus clientes la posibilidad de realizar configuraciones y ajustes por ellos mismos aportándoles tener un cierto grado de control sobre el paquete de servicios que contratan. En la arquitectura presentada en el capítulo 5, el bloque INFORD 5.2.1 integra el Servicio de Aprovisionamiento de Aplicaciones (APS) que actúa como marketplace de servicios y dispositivos virtuales que los clientes pueden consultar e instalar los servicios que requieran en sus redes virtuales privadas. De esta manera se consigue que este requisito quede cubierto por el trabajo propuesto.

18. R18. Este requisito insta a proporcionar seguridad en los espacios de usuarios con ánimo a restringir y asegurar el correcto acceso a la información. Este requisito depende bastante de las tecnologías utilizadas para instanciar las infraestructuras. Debido a al diseño propuesto recoge las necesidades de los proveedores de servicios a través de la capa BSS de IMOM (sección 5.2.2) y luego traduce esas necesidades en archivos de configuración a través del orquestador para que sean aplicadas por las herramientas de gestión de la configuración, se considera que este requisito se cumple. De hecho, herramientas como las analizadas en la sección 3.2.1, por ejemplo Puppet [31], ofrecen opciones de configuraciones de cuentas y grupos de usuarios.

19. R19. Este requisito propone no usar modelos de arquitecturas específicos que limiten el despliegue de servicios. Es decir, cada servicio tiene requisitos únicos que deben ser satisfechos para permitir que los servicios se conecten entre sí en el "espacio de usuario" de manera automática. El modelo de capas software presentado en 7 permite definir la infraestructura y software que existe por debajo de los servicios, por lo que el software que soporta a aquellos servicios podría desplegarse en función de sus necesidades. Para realizar este proceso se introducirían las especificaciones de las capas software a desplegar y el orquestador generaría los archivos de configuración correspondientes para aplicar aquellas configuraciones, de esta manera se cumple con el requisito.

9.4 CONCLUSIONES

En este capítulo se ha presentado un prototipo para validar la arquitectura mostrada en el capítulo 5, la validación se ha realizado a través del despliegue de servicios sobre una infraestructura INFORD para la cual se han desarrollado unos servicios diseñados para la distribución de contenidos multimedia. Además, se ha comparado el uso funcionamiento del prototipo desplegado sobre infraestructuras Bare metal, Contenedores Linux y contenedores basados en microservicios. El capítulo también muestra los resultados obtenidos de ejecutar el prototipo sobre las diferentes infraestructuras.

El capítulo también presenta un análisis del grado de cumplimiento de los requisitos propuestos en la sección 2.1.6. En la sección 9.3 muestra uno a uno la razón por la que el trabajo propuesto en esta tesis es capaz de satisfacer cada uno de los requisitos.

10 CONCLUSIONES

Finalmente, en este capítulo se presentan las conclusiones generales sobre el trabajo desarrollado y se realiza un resumen de los beneficios de la propuesta presentada. También se proponen líneas de investigación futuras.

10.1 PRINCIPALES CONTRIBUCIONES

Para alcanzar los objetivos desarrollados durante el trabajo de esta tesis se definieron unos micro objetivos que ayudaron a seguir la línea de desarrollo establecida inicialmente. La consecución progresiva de estos micro objetivos han permitido poco a poco ir evaluando el progreso y el correcto avance de las tareas desarrolladas acorde a la línea de progreso planificada inicialmente. Como resultado final, se ha desarrollado un trabajo donde se presenta una arquitectura de servicios flexible y adaptable mediante mecanismos programáticos capaz de ajustarse a múltiples condiciones. Además, también se propone mecanismos que hacen que la arquitectura automatice procesos de despliegue proporcionándole capacidades de autoconfiguración y adaptación para ser desplegada sobre múltiples infraestructuras en la nube. A continuación se muestran las contribuciones más relevantes.

Diseño de un ecosistema digital doméstico e integrado

Como contribución se ha presentado el diseño de un ecosistema digital doméstico capaz de conectar dispositivos multimedia domésticos, dispositivos móviles y servicios de consumo sobre una misma red virtual independientemente donde se encuentren físicamente cada uno de estos elementos. El objetivo es que todos estos elementos se puedan conectar a la red virtual de manera sencilla y trabajen como si se encontrasen sobre la red del hogar permitiendo a los consumidores gestionar su ecosistema digital de manera simple y sencilla. Conseguir la integración de todos estos elementos no es sencilla y se ha requerido el uso de diferentes tecnologías como la virtualización para implementar mecanismos que proporcionen entornos de ejecución adaptables a diferentes necesidades. La virtualización es una tecnología clave para poder adaptar software a múltiples entornos de ejecución, esto se muestra en la sección 2.2 del capítulo 2 del estado del arte

donde se presenta un análisis de las diferentes tecnologías de virtualización.
La necesidad de utilizar diferentes opciones de configuración sobre entornos de configuración requiere de mecanismos que faciliten estas tareas para reducir la probabilidad de cometer errores y poder agilizar los procesos. El uso de herramientas de Infraestructuras como Código (IaC) permiten realizar estas tareas. En el capítulo 3, se realiza una clasificación de las herramientas IaC que sirve como base para determinar qué herramienta es la más adecuada para las diferentes tareas de despliegue.

Para poder administrar y gestionar el ecosistema digital primero se desarrolló una arquitectura inicial presentada en el capítulo 4 y publicada en el artículo [202]. Posteriormente, esta arquitectura se mejoró dando lugar a una nueva arquitectura, mostrada en el capítulo 5, la cual está compuesta por varios niveles de servicios y se esta dividida en dos grandes bloques. Cada uno de estos bloques esta compuesto por diferentes componentes independientes que interactúan unos con otros para poder realizar sus funciones. Esta arquitectura se ha publicado en una revista [9] y se los resultados de validación se presentan en el capítulo 7. Concretamente la parte relacionada con el despliegue de diversos dispositivos sobre redes virtuales alojadas sobre una infraestructura virtualizada.

Modelo de arquitectura flexible y adaptable

Una de las contribuciones más relevantes de esta tesis es el diseño de una arquitectura de servicios capaz de adaptarse a las necesidades del entorno de la infraestructura y al conjunto de servicios que van a ser desplegados sobre la misma.

Este objetivo ha sido cubierto satisfactoriamente a través del uso de diferentes tecnologías tales como la virtualización o la Infraestructura como Código (IaC). Escoger como base una arquitectura de servicios en lugar de otro tipo de arquitectura, como por ejemplo una arquitectura monolítica, no es aconsejable debido a las múltiples ventajas que proporcionan las arquitecturas de servicios respecto a las arquitecturas monolíticas. Principalmente, las ventajas de las arquitecturas de servicios son la capacidad de organizar y gestionar eficientemente los sistemas a través de servicios y esto es un aspecto necesario para el trabajo desarrollado en esta tesis debido a que facilita aplicar mecanismos de gestión automatizados.

Debido a la necesidad de mejorar el desempeño sobre entornos de aplicación específicos se han desarrollado alternativas que se basan en la ar-

quitectura de servicios tradicional para satisfacer necesidades en esas situaciones. La sección 2.1 presenta arquitecturas de servicios basadas en el modelo tradicional pero que incorporan modificaciones que las hacen más idóneas para determinadas situaciones. En esta línea, en esta tesis se han identificado varios tipos de arquitecturas de servicios y se han clasificado en: arquitectura de servicios distribuidos geográficamente, arquitectura de Brokers de servicios, arquitectura de servicios encadenados y arquitectura basadas en microservicios (todas estas arquitecturas son analizadas en la sección 2.1). Sin embargo, a pesar de que todas estas arquitecturas con capaces de ajustarse a multitud de situaciones existen necesidades que no han sido cubiertas y además, con la evolución de las tecnologías han aparecido nuevas retos por abordar. Por ejemplo, mecanismos para facilitar le mover servicios entre infraestructuras, despliegue de servicios automatizados que integren capacidades de adaptación los entornos de ejecución a los servicios a desplegar y optimización de los propios servicios que van a ser desplegado, facilidades para virtualizar cualquier dispositivo digital, soporte para la configuración de enlaces virtuales, la reducción de la latencia de los servicios de entrega de contenidos digitales, incremento y simplificación de tareas de gestión de las redes, optimización en la colocación de servicios de red, gestión de servicios de manera desatendida, el desarrollo de herramientas avanzadas, etc.

Para hacer frente al desafío de satisfacer todos estas necesidades se ha desarrollado la arquitectura mostrada en el capítulo 5, la cual se divide en dos bloques principales IMOM e INFORD entre los que se distribuyen las funcionalidades. INFORD representa el bloque con la arquitectura de infraestructura que permite desplegar y funcionar los servicios multimedia. IMOM contiene las herramientas necesarias para definir el funcionamiento y servicios que se van a desplegar sobre INFORD. IMOM permite definir la configuración deseada y más adecuada para el despliegue sobre INFORD. IMOM permite registrar los componentes software y servicios que posteriormente podrán desplegarse sobre INFORD, a través de un orquestador de software encargado de desplegar de manera automática los servicios.

Esta arquitectura ha sido publicada en el artículo [9] y se ha validado en los capítulos 7 y 9. En el capítulo 7 se valida la arquitectura y el funcionamiento del orquestador (detallado en el capítulo 6) y en el capítulo 9 se presentan los resultados de ejecutar un prototipo desplegado sobre la infraestructura INFORD utilizando como base diferentes tipos de infraestructuras.

Automatización en la configuración de infraestructuras y despliegue de servicios

Otra de las contribuciones de esta tesis es el diseño de mecanismos capaces de realizar despliegues de servicios y su posterior configuración de manera automática. La idea no es realizar sólo estas tareas a nivel de aplicación sino que también se realizan tareas que incluyen niveles más abajo, incluso los niveles relacionados con la infraestructura. El objetivo ha sido cubierto a través del uso de APIs estandarizados de conexión, herramientas de orquestación y herramientas de gestión de la configuración.

Como se puede ver en la sección 2.4 y el capítulo 3 donde se presenta un análisis de todas estas soluciones, no existe una herramienta o solución que sea capaz de proporcionar los mecanismos necesarios para configurar todos los aspectos que hay que tomar en cuenta para configurar una infraestructura desde las capas de servicios situada en los niveles más altos de una infraestructura de servicios, hasta los niveles más bajos compuesto de los recursos de computación. Para poder abordar el problema, en el capítulo 6 se propone una distribución de capas software la cual divide en cuatro niveles lo que sería una infraestructura software.

Cada una de las capas tiene unas características y aspectos de configuración particulares que la hace diferentes entre sí debido a que agrupa componentes software semejantes pero que a su vez son diferentes de los componentes ubicados en las otras capas. En el capítulo 6 se analiza cada una de estas capas y en función de ello se presenta las herramientas más adecuadas (de las ya analizadas anteriormente en el estado del arte) para gestionar los componentes de cada una de las capas. Dicha configuración se realiza a través de un orquestador alojado en la arquitectura propuesta en el capítulo 5. El orquestador esta preparado para integrarse con las herramientas diseñadas para gestionar el software(analizadas en el estado del arte) y las encargadas de gestionar recursos de infraestructura en la nube. El orquestador propuesto se beneficia de esta segmentación de las infraestructuras en capas permitiéndole un mayor control y focalización en las configuraciones contenidas en cada una de las capas.

Esto permite reducir la probabilidad de errores de configuración, ayuda a la selección de las herramientas adecuadas para configurar la capa correspondiente y reduce la complejidad de las configuraciones reduciendo las opciones a tener en cuenta en la configuración de la capa a configurar. Parte de la solución propuesta para abordar este objetivo ha sido publicado anteriormente en la revista [9] la cual incluyó parte de la validación presen-

tada en el capítulo 7.

Optimización de dependencias y del despliegue de servicios software

Esta tesis también propone mecanismos de optimización como contribución necesaria para complementar las anteriores contribuciones. Concretamente se propone optimizar el despliegue de servicios y componentes software sobre diferentes entornos. El despliegue de servicios software no es una tarea sencilla y un despliegue correcto que tenga en cuenta los factores apropiados es importante ya que determina el rendimiento general del sistema y el adecuado consumo de recursos, sobre todo cuanto más crece el sistema.

A lo largo del desarrollo de los diferentes capítulos se ha hecho patente la necesidad de profundizar en aspectos de optimización no sólo ligados a la gestión del software configurado sino también tener en cuenta aspectos de la tecnología utilizada, la localización en donde desplegar los servicios o el tipo de infraestructura más adecuada.

En el capítulo 8 se proponen unos mecanismos automatizados para optimizar el despliegue de componentes software sobre sistemas TI. Durante el desarrollo del trabajo de la tesis se han identificado tres aspectos fundamentales a evaluar: 1) la plataforma sobre la cual se va a desplegar el servicio, 2) el tipo de tecnología de virtualización con la que se encapsula el servicio en cuestión (o tal vez no sea necesario utilizar algún mecanismos de virtualización), y 3) la localización idónea del servicio, es decir, si sería conveniente desplegarlo en el core de la red o en algún centro de datos situado en el borde de la red. El capítulo 9 presenta un prototipo que se ha utilizado como validación de los mecanismos propuestos como optimización y muestra las medidas tomadas en cada uno de los casos.

Modelado de la información del software de infraestructura

Otra contribución de esta tesis es proporcionar un modelo de información estructurado que represente los diferentes elementos software distribuidos desde el nivel más bajo de una infraestructura, es decir desde el nivel de recursos de infraestructura, hasta el nivel de aplicación sobre el cual se desplieguen los servicios con las funcionalidades de más alto nivel. Durante la sección 2.4 del capítulo 2 del estado del arte se analizaron algunos estándares orientados a proporcionar mecanismos de gestión de recursos de infraestructura. Algunos de estos estándares propone modelos de datos para modelar los recursos de infraestructuras. Sin embargo, estos

modelos están diseñados específicamente para cubrir el ámbito de aplicación del estándar en cuestión por lo que son modelos muy limitados y no resultan válidos para las funcionalidades requeridas en este trabajo.

El diseño de un modelo de información es importante ya que permite registrar y, por lo tanto, conocer los recursos que hay disponibles en una infraestructura. Además, desde el punto de vista software, no sólo permite saber qué servicios se encuentran en el sistema para disponer de un catalogo de servicios que podrían ser desplegados en cualquier momento, sino que también permiten conocer los requisitos de funcionamiento del software en forma de dependencias para ser satisfechas antes de desplegar un software. Las dependencias no sólo pueden ser librerías o componentes software sino que también pueden ser conexiones de red con características especiales de latencia, servicios complementarios, etc.

A lo largo del desarrollo de la tesis, según se avanzaba con la investigación ha surgido la necesidad de registrar la información para los diferentes propósitos abordados en la investigación. En el capítulo 6 se presentan modelos de información para las capas software comentadas anteriormente con el objetivo de ser registrada en repositorios y posteriormente recuperada por el orquestador de IMOM para generar configuraciones. En los capítulos 7 y 9 durante la validación de los prototipos desarrollados se utilizaron estos modelos de información.

Orquestación del software basado en herramientas de la IaC

La orquestación de software a través de herramientas de la IaC es otra contribución fruto del trabajo de esta tesis. La orquestación del software se realiza a través de dos servicios contenidos en el bloque IMOM de la arquitectura propuesta en el capítulo 5.

Los procesos de orquestación software en el trabajo desarrollado en esta tesis no solo facilitan la gestión de sistemas basados en servicios sino que además permiten conectar esos servicios a la red para que puedan comunicarse con los usuarios y otras aplicaciones. Esto garantiza la ejecución de actividades en el orden correcto y escalado de recursos acorde a las reglas de seguridad preestablecidos y con los permisos adecuados. En el capítulo 3 se analizan varias herramientas de orquestación que permiten realizar tareas de aprovisionamiento y orquestación, algunas de estas tareas incluyen gestionar máquinas virtuales, servicios y recursos. Cada una de estas herramientas posee características específicas que otras herramientas no poseen, de hecho la tabla 11 y 12 presenta una comparación de las características

ofrecidas por cada una de las herramientas.

En este trabajo se presenta un orquestador que se apoya en diferentes herramientas para realizar las tareas de orquestación, el orquestador propuesto recopila la información que especifica el software y servicios a ser desplegados y genera el código con el formato correspondiente para que posteriormente pueda ser procesado por una de las herramienta orquestación. El orquestador toma en cuenta el modelo de capas software y dependiendo qué capa vaya a configurar, genera un formato de código correspondiente a la herramienta más adecuada a la capa a configurar. En el capítulo 3 se muestran varias clases de herramientas en función del ámbito de funcionamiento, por ejemplo las herramientas de gestión de configuración serán utilizadas por el orquestador en el caso de necesite configurar servicios o plataformas software. Sin embargo, en el caso de tener que configurar recursos de infraestructura, las herramientas de aprovisionamiento serían las más adecuadas para este caso y el orquestador proporcionaría la configuración en el formato adecuado para que la herramienta pudiese funcionar. En el capítulo 7 se presenta un prototipo que muestra su funcionamiento a través del despliegue de una infraestructura virtualizada junto con servicios de distribución de contenidos multimedia.

10.2 CONCLUSIONES

En esta tesis se ha desarrollado una arquitectura programable capaz de integrar servicios y dispositivos domésticos sobre diferentes proveedores de infraestructura en la nube. Sin embargo, esto presenta varios problemas debido a la heterogeneidad de las infraestructuras en la nube relacionados con la interoperabilidad y Data Lock-In. Además, la capacidad de adaptarse a múltiples infraestructuras es compleja, y los mecanismos tradicionales utilizados para implementar software tienen limitaciones que reducen su alcance a ciertas plataformas.

Para abordar estas limitaciones, uno de los primeros trabajos ha sido estudiar varias alternativas de modelos de arquitecturas de servicios orientadas a satisfacer las necesidades sobre diferentes escenarios de aplicación. Las arquitecturas de servicios son una forma eficaz de gestionar sistemas complejos de TI, pero dependiendo del tamaño, el tipo de aplicación y los requisitos del sistema a implementar, los diseñadores podrían optar por alguno de los modelos de arquitecturas recopilados durante este trabajo.

La mayor parte de los modelos de arquitecturas de servicios son una evolución del modelo de arquitectura de servicios tradicional, las diferentes

alternativas han surgido con el objetivo de solucionar necesidades encontradas sobre los sistemas reales. Esta situación ha servido como punto de partida de la idea principal de la tesis, la cual ha promovido la inquietud de pensar cuáles serán las futuras necesidades y cambios que deberán afrontar las futuras arquitecturas de servicios para hacer frente a las necesidades de las aplicaciones actuales. Se prevé que algunas de las posibles necesidades requerirán la reducción de la latencia en las comunicaciones, el incremento y simplificación de tareas de gestión de las redes, la optimización en la colocación de servicios de red, la gestión de servicios de manera desatendida y el desarrollo de herramientas avanzadas. Cubrir todas estas necesidades facilitará la gestión de los procesos encargados de las tareas de mantenimiento y gestión de grandes y complejas arquitecturas de servicios. Esto requiere rediseñar nuevos mecanismos y herramientas para abordar los futuros escenarios de sistemas orientados a servicios. De hecho, la personalización en las futuras arquitecturas de servicios será uno de los grandes desafíos a conseguir ya que implicará que los recursos deben configurarse dinámicamente y agruparse para ser capaces de proporcionar diferentes perfiles de servicio a demanda.

Algunas de los principales tecnologías clave para hacer satisfacer las nuevas necesidades se describen a continuación:

1. La tecnología de la virtualización la cual proporciona mecanismos que permiten encapsular y empaquetar servicios software para facilitar la gestión de los servicios sobre diferentes arquitecturas de servicios. Por lo general, las tecnologías utilizadas por las infraestructuras difieren unas de otras y no son compatibles. Esto implica que un servicio software que funcione perfectamente sobre un sistema TI al llevarlo a otro sistema puede ser que no llegue a funcionar. Los mecanismos de virtualización permiten resolver este problema utilizando máquinas virtuales o contenedores para albergar los servicios en un entorno de ejecución perfectamente optimizado a sus necesidades. A pesar de que el uso de la virtualización puede ser una solución, no existe una tecnología de virtualización capaz de proporcionar todas las características técnicas idóneas para gestionar los servicios de manera óptima. Por lo que en este trabajo se han analizado las diferentes tecnologías de virtualización para conocer las fortalezas y debilidades de cada una de ellas, ya que, dependiendo de las capacidades específicas de cada herramienta se podrá saber cual es la más adecuada para cumplir el objetivo deseado. En este trabajo se han clasificado las tecnologías de virtualización en dos grandes bloques: por un lado las máquinas virtuales (VMs) y por otro lado los contenedores. La virtualización basada en contenedores es más simple y considerablemente

más ligera que las VMs, pero las VMs son más maduras, más seguras y más eficaz. Por lo tanto, una consideración principal para la virtualización basada en hipervisor con respecto a la virtualización basada en contenedores es el rendimiento general de las aplicaciones. Como se ha comentado, todas estas características que hacen unas tecnologías diferentes de otras requiere que los ingenieros deban conocer y tener en cuenta las particularidades de cada una de ellas para poder seleccionar la tecnología que más le convenga para encapsular los servicios que van a ser desplegados sobre una arquitectura de servicios. El análisis y estudio para determinar qué tecnología es la más adecuada para encapsular diferentes tipos de servicios es un tema poco explorado y que ha sido abordado en esta tesis.

2. La softwarización de dispositivos físicos es relevante para el crecimiento de las arquitecturas de virtualización de dispositivos. La softwarizacion no es un tema nuevo y hay trabajos que proponen aproximaciones para llevar las funciones de los dispositivos customer-premises equipment (CPE) al núcleo de la red de la empresa de distribución de contenidos. Esto está motivado por la tendencia de mover las funciones del dispositivo a la nube donde normalmente se propone trasladar parte de las funciones del equipo de las instalaciones del cliente (CPE) al centro de datos ubicado en el núcleo de la red como un CPE virtual (vCPE). Una de las motivaciones principales es reducir la complejidad de los dispositivos físicos alojados en el borde de la red, permitiendo acceder a las funciones más complejas en la nube a través de dispositivos que no requieren de altas prestaciones para acceder a estas funciones. De esta manera, la parte de computación más exigente se encuentra en la nube y los usuarios pueden hacer uso de aplicaciones de alto rendimiento las cuales son ejecutadas en la nube. Es decir, la mayor parte de la inteligencia permanece en la nube reduciendo las necesidades hardware. La softwarización es un elemento clave para los sistemas de servicios futuros ya que permitirá prescindir de dispositivos físicos en ciertos escenarios de aplicación mientras que a su vez hace uso de los enormes beneficios que aporta en la gestión de los dispositivos softwarizados.

3. La tecnología de computación en la nube también es clave para este trabajo, debido a que soluciona el problema de la implementación de infraestructuras complejas con muchos requisitos de hardware y software. La complejidad de los sistemas pueden dar problemas y causar errores si se utilizan herramientas de implementación tradicionales (tales como scripts de configuración), ya que estas herramientas requieren tomar en cuenta características específicas de la infra-

estructura que se está utilizando, para poder realizar correctamente el despliegue del software. Esta situación ha sido tenido en cuenta por los proveedores de nube y han desarrollado diferentes recursos y servicios, disponibles en la propia nube, para facilitar a sus clientes la implementación y despliegue de la infraestructura. Sin embargo, a pesar de la existencia de estas herramientas desarrolladas por los proveedores de la nube, los administradores de los sistemas en la nube deben afrontar tareas complejas y, además, deben hacer frente a la gestión de sistemas que incrementan la dificultad de gestionarlos correctamente debido a la creciente complejidad de las aplicaciones y de los requisitos de las mismas. Esta situación genera problemas de: a) heterogeneidad de los recursos en la nube; b) falta de compatibilidad con el modelo de recursos de los diferentes proveedores de nubes; c) diversidad de APIs de conexión a los proveedores de infraestructura en la nube que dificulta la gestión de diferentes infraestructuras en la nube; d) falta de capacidad de integración entre diferentes entornos de nube; y e) la transferencia de aplicaciones entre nubes causa complejidad y errores irrazonables y despliegue sobre múltiples infraestructuras los cuales exigen demasiados recursos humanos para el despliegue y la preparación. Estos problemas pueden ser abordados desde diferentes perspectivas a través de los estándares donde cada estándar tiene como objetivo conseguir algún aspecto clave en la gestión de la infraestructura en la nube. En función del análisis realizado se han identificado tres aspectos de interés por los organismos de estandarización:

- Automatización en la gestión y despliegue de aplicaciones. El objetivo es proporcionar un lenguaje para expresar cómo implementar y administrar automáticamente aplicaciones complejas en la nube. Esto se logra a través de la definición de una topología abstracta de aplicaciones complejas donde se describan su implementación y administración.
- Portabilidad y gestión de aplicaciones a través de la IaC. Este objetivo trata de proporcionar una forma estandarizada de describir la topología de las aplicaciones. También aborda la portabilidad de la administración confiando en la portabilidad de los lenguajes de flujo de trabajo utilizados para describir los planes de implementación y administración.
- Interoperabilidad y reutilización de componentes. Donde el objetivo es describir los componentes de aplicaciones complejas en la nube de una manera que permita la interoperabilidad y la reutilización de los componentes.

4. Esta tesis analiza los estándares diseñados para gestionar recursos y software en la nube. Además, muestra una comparación que contempla las diferentes tecnologías de interoperabilidad. Una de las características más importantes al realizar el estudio de los estándares es que cada una de ellas utiliza su propio DSL y además no todos los DSLs tienen el mismo grado de desarrollo y alcance. Por lo tanto, los recursos en la nube descritos con estos estándares podrían no ser interoperables unos con otros debido a su esquema y vocabulario heterogéneos. En un entorno colaborativo donde las organizaciones están dispuestas a compartir sus recursos en la nube, traducir descripciones de recursos en la nube es, por lo tanto, una tarea manual y tediosa. Esto a su vez crea silos en la nube y recursos en la nube no reutilizables. Como resultado de esta situación y con la aparición de nuevas necesidades durante los últimos años, OASIS ha desarrollado la especificación TOSCA. TOSCA es la especificación que ofrece más funcionalidades además de ser la única que esta siendo mantenida en los últimos años. Sin embargo, durante la investigación se descubrió que era imposible llevar a cabo una configuración completa entre los recursos de diferentes proveedores de nube, ya que incluso los parámetros básicos presentados por el estándar TOSCA no son totalmente compatibles con los proveedores de nube más avanzados. Además, el estándar TOSCA requiere clarificación y desarrollo. A pesar de esto, TOSCA permite que todas las actualizaciones se agreguen rápidamente. Durante el desarrollo de la tesis se ha sido consciente de estos problemas y con el objetivo de resolverlos, esta tesis propone mecanismos que, partiendo de la información de la infraestructura, pueden adaptar esta información de manera sencilla, a las configuraciones requeridas basadas en alguno de los estándares descritos en este trabajo.

5. Disponer de las herramientas de gestión del software clave es necesario para la gestión de los nuevos sistemas. De hecho, actualmente existen varios tipos de herramientas para gestionar diferentes aspectos software de los diferentes niveles de infraestructura de los sistemas. Los diferentes componentes software distribuidos sobre diferentes niveles forman un ecosistema digital el cual puede ser fácilmente gestionado si se cuenta con las herramientas adecuadas. La IaC se apoya en la gestión de la configuración para gestionar el despliegue de software sobre sistemas TI. La gestión de la configuración tiene como objetivo automatizar, supervisar, diseñar y administrar los procesos de configuración manuales. Las herramientas de gestión de configuración funcionan especificando la configuración de un sistema a través de estados. Por ejemplo, se especificarían estados para confi-

gurar la infraestructura básica del sistema (almacenamiento, redes). Cada paso de la configuración a menudo depende de otros, por ejemplo, para crear una aplicación se debe instalar un compilador y luego compilar el código fuente antes de realizar su instalación; o para ejecutar un servicio orientado a la Web, es posible que se requiera un servidor web y un servidor de bases de datos. Durante el desarrollo de la tesis se ha realizado un estudio que recoge las principales herramientas de gestión de la configuración. En el estudio se puede observar como cada una de las herramientas son capaces de proveer un conjunto de funcionalidades que las hace idóneas para distintas soluciones dependiendo de las necesidades a cubrir.

6. Las herramientas de aprovisionamiento también tienen un papel importante en las futuras arquitecturas de servicios. Estas herramientas están diseñadas para aprovisionar servidores así como el resto de su infraestructura (como equilibradores de carga, bases de datos, configuración de redes, etc.). En ocasiones puede haber un cierto solapamiento entre las herramientas de aprovisionamiento y las herramientas de gestión de la configuración (HGC). Esto es debido a que que la mayoría de las HGCs pueden hacer cierto grado de aprovisionamiento y la mayoría de las herramientas de aprovisionamiento pueden hacer cierto grado de administración de la configuración. Sin embargo, es preferible usar HGC cuando sea necesario gestión de la configuración y usar herramientas de aprovisionamiento cuando haya que hacer tareas de aprovisionamiento de recursos debido a que cada tipo de herramienta esta diseñada principalmente para trabajar en un ámbito concreto.

Como resultado del análisis de las diferentes tecnologías claves se procedió a desarrollar un primer concepto de arquitectura con capacidad de crear redes virtuales para compartir contenidos a través de dispositivos físicos situados en diferentes dominios de red. Dicha arquitectura permite extender las redes locales compuestas de dispositivos domésticos del hogar a redes superpuestas. Además, también se ha propuesto una solución la cual optimiza el uso de librerías software sobre frameworks OSGi. OSGi es un entorno de ejecución muy extendido en dispositivos con recursos limitados y la solución propuesta para optimizar el software ayuda a mejorar el funcionamiento general de los dispositivos. Ambas soluciones contribuyen a que dispositivos domésticos con pocos recursos puedan ampliar las capacidades de los protocolos de compartición de contenidos multimedia ampliamente extendidos en la industria tales como UPnP AV y DLNA. La solución propuesta extiende estos protocolos haciendo uso del protocolo XMPP para llevar los mensajes UPnP intercambiados a lugares remotos

de internet. El protocolo XMPP está orientado a servicios de mensajería instantánea entre usuarios. Esta solución utiliza XMPP para crear redes de dispositivos en lugar de usuarios permitiendo que los dispositivos se conecten y desconecten de la red XMPP y permitiéndoles intercambiar contenidos de forma transparente a la localización. XMPP es ideal para este cometido ya que gracias a las funcionalidades de indicador de presencia, gestión de grupos o la multitud de extensiones disponibles no sólo permite un funcionamiento óptimo, si no que también permite añadir nuevas funcionalidades a las ya existentes ofrecidas por las redes UPnP. Debido a que los protocolos de UPnP y DLNA suelen ser implementados sobre dispositivos con bajas capacidades de cómputo, la optimización de paquetes en OSGi propuesta proporciona una solución para reducir los recursos de memoria de estos dispositivos de manera automática en base a unas puntuaciones que permiten minimizar el consumo de memoria y facilitan las actualizaciones proporcionando un mayor control en términos de seguridad.

La solución comentada anteriormente no satisfacía parte de las necesidades comentadas en el punto 1 descrito en esta sección, y por esta razón se comenzó con el diseño de otro modelo de arquitectura más complejo pero que satisficiera las necesidades identificadas en el punto 1. Dicha arquitectura presenta un diseño de red local del hogar compuesta por dispositivos domésticos a los que los usuarios pueden acceder independientemente de la localización geográfica. Además, dicha red puede ser configurada sobre múltiples infraestructuras independientemente del proveedor y la tecnología utilizada debido a que para el diseño se ha tomado muy en cuenta el uso de las tecnologías de virtualización que facilitan la interoperabilidad e importación y exportación de los servicios virtualizados alojados en dichas redes.

Para proporcionar las funcionalidades adicionales necesarias por la arquitectura propuesta se proponen las siguientes contribuciones adicionales:

- Se propone un un modelo jerárquico de capas software sobre el cual se distribuyen los diferentes elementos software necesarios para la ejecución de servicios de propósito general. Este modelo propone dividir las infraestructuras de los sistemas TI en 4 capas independientes pero relacionadas unas con otras en donde el software se gestiona específicamente en función de cada capa. La capa más baja es la encargada de la gestión de los recursos hardware que definen la infraestructura del sistema junto con las redes y conexiones. La siguiente capa, que se apoya sobre esta primera capa, contiene instancias de imágenes de sistemas operativos sobre las que se configurará el software de las capas superiores. La capa tercera es la encargada de proporcionar los entornos de ejecución, plataformas y frameworks que soportarán los

servicios de la capa superior. Finalmente, la capa superior contiene los servicios y aplicaciones de más alto nivel y que suelen proporcionar la funcionalidad final a lo clientes y consumidores. La configuración de cada una de las capas requiere modelar la información de manera particular debido a que no todas las capas se comportan igual, ni tienen los mismos objetivos, y ni proporcionan las mismas funcionalidades, por lo que ha sido necesario analizarlas una a una para identificar el conjunto de datos relevantes para cada una de ellas. Como conclusión, la segmentación de los sistemas en capas permiten un mayor control y focalización en las configuraciones perimetrales dentro de cada una de las capas. Esto permite reducir la probabilidad de errores de configuración, ayuda a la selección de las herramientas adecuadas para configurar la capa correspondiente, y reduce la complejidad de las configuraciones minimizando el alcance a la capa a configurar.

- Como contribución también se han presentado unos mecanismos para optimizar el despliegue de componentes software sobre sistemas TI. Desplegar un servicio no es una tarea sencilla y se deben considerar diversos factores antes de realizar el proceso. Con este objetivo se ha desarrollado una arquitectura y el modelo de datos que recoge la información necesaria a la hora de realizar el proceso de optimización. La solución propuesta considera tres aspectos fundamentales a evaluar: 1) la plataforma sobre la cual se va a desplegar el servicio, 2) el tipo de tecnología de virtualización con la que se encapsulara el servicio en cuestión (o tal vez no sea necesario utilizar algún mecanismos de virtualización), y 3) la localización idónea del servicio, es decir, si sería conveniente desplegarlo en el core de la red o en algún centro de datos situado en el borde de la red. Durante este trabajo se han mostrado los principales factores a ser considerados, también se ha presentado una arquitectura y el proceso de evaluación a seguir para indicar si un servicio es apropiado para un tipo de arquitectura o no.

En conclusión, se ha desarrollado una arquitectura programable capaz de crear instancias de servicios y dispositivos domésticos multimedia en la nube. Debido a la heterogeneidad de las infraestructuras en la nube, la arquitectura propuesta utiliza mecanismos de virtualización para resolver problemas de interoperabilidad y Data Lock-In. Además, estos mecanismos permiten adaptar la arquitectura a múltiples entornos de nube, incluso a entornos de computación localizados en el borde de la red.

10.3 LÍNEAS DE INVESTIGACIÓN FUTURAS

A pesar de que en el trabajo desarrollado durante la tesis se han realizado diversas pruebas de validación y se han presentado ideas y conceptos relevantes, hay algunos aspectos que pueden ser tenidos en cuenta para ampliar el conocimiento sobre el trabajo realizado. A continuación se proponen algunos trabajos e iniciativas que pueden ser tomadas en cuenta como punto de partida para continuar con la línea de investigación iniciada en esta tesis.

1. En relación con las pruebas de validación realizadas para comprobar las ideas de despliegue automático del software, durante el desarrollo del trabajo de esta tesis se desarrolló un proceso que permitía la gestión de los recursos de infraestructura de manera simple. El proceso se dividía en dos partes: por un lado se separaba toda la información relacionada con la infraestructura a desplegar y por otro lado se encontraba el algoritmo (desarrollado durante esta investigación) encargado de procesar toda esta información sin ninguna otra acción requerida por este. Esto da sensación de una caja negra, donde por un lado se obtiene la información de la infraestructura a generar, esta información se introduce en la caja negra formado por el algoritmo y éste instancia las configuraciones definidas sobre una infraestructura. Este modelo de funcionamiento ha funcionado muy bien al implementarlo a con la herramienta de gestión de código Vagrant, pero no todas las herramientas orientadas a la orquestación de recursos funciona de la misma forma y tal vez esta forma de trabajar podría no ser compatible con otras herramientas. Por lo tanto, se propone la necesidad de un análisis de otras herramientas de orquestación de recursos para evaluar la posibilidad de extender el funcionamiento del modelo de trabajo aplicado en Vagrant a otras herramientas.

2. La generación de código automática en un formato comprensible por las herramientas de configuración no es una tarea sencilla, pero en este trabajo se ha generado código para la herramienta de gestión de la configuración Puppet. Puppet esta basado en el paradigma de los lenguajes declarativos que facilitan la tarea de transformación desde un modelo de información estructurado (como el de la solución desarrollada en esta tesis) al código con el formato soportado por Puppet. Como trabajo futuro se propone el estudio y análisis de otras herramientas de configuración software con el objetivo de ampliar los mecanismos de transformación de código a otras herramientas de gestión de la configuración, tanto a herramientas basadas en lenguajes declarativos como en las que están basadas en lenguajes imperativos.

3. Respecto a las pruebas realizadas para validar el comportamiento del prototipo sobre diferentes entornos de ejecución, los resultados mostrados en el trabajo presentan diferencias de comportamiento. Hay diferencias considerables cuando el prototipo es desplegado sobre tecnologías de contenedores de Linux frente a tecnología Docker o directamente sobre el sistema operativo. A pesar de que la elaboración, diseño y configuración de cada uno de estos entornos es laboriosa, aumentar el número de entornos de ejecución para tomar más medidas es necesario para incrementar el conocimiento de los trabajos desarrollados. Además de adquirir nuevo conocimiento sobre cómo se comportan diferentes combinaciones de tecnologías y configuraciones respecto a unos mismos servicios desplegados en cada una de las combinaciones de infraestructuras generadas.

4. En relación a la posibilidad incrementar la digitalización de dispositivos físicos no hay demasiada literatura que aborde un análisis sobre este punto. Existen soluciones que permiten llevar las funcionalidades de dispositivos físicos a un entorno completamente software a través de mecanismos de virtualización. Por ejemplo, hay dispositivos routers que su software puede ser cargado completamente sobre un entorno de ejecución que simula la parte física permitiendo que las funcionalidades proveídas por el dispositivo sean completamente operativas. Sin embargo, otros dispositivos más complejos como son los smarthphones es más complicado encontrar soluciones maduras que trabajen adecuadamente sobre estos entornos. Esto dificulta conseguir entornos completamente virtualizados en el que puedan convivir dispositivos de red y dispositivos de consumo virtualizados. Como línea de investigación futura podría proponerse profundizar en el proceso de softwarizacion o también llamado Anything as a Service (XaaS).

5. El trabajo ha propuesto la gestión de las infraestructuras a través de un modelo de capas software para poder gestionar los componentes software distribuidos por los diferentes niveles de una infraestructura. En el documento de la tesis se ha modelado los diferentes elementos contenidos en cada una de las capas a través de parámetros relevantes sobre cada uno de ellos. Como trabajo futuro se propone realizar un análisis más profundo de otros posibles parámetros relevantes que puedan ser incorporados a estos modelos para cubrir o incorporar nuevas funcionalidades que puedan ser descubiertos en el proceso de mejora.

BIBLIOGRAFÍA

[1] Cisco and/or its affiliates. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021*. Technical Report. Cisco, 2017.

[2] Arslan Ahmad, Alessandro Floris y Luigi Atzori. «QoE-aware service delivery: A joint-venture approach for content and network providers». En: *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE. 2016, págs. 1-6.

[3] J. Opara-Martins, R. Sahandi y F. Tian. «A Business Analysis of Cloud Computing: Data Security and Contract Lock-In Issues». En: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. IEEE, nov. de 2015, págs. 665-670. ISBN: 978-1-4673-9473-4. DOI: 10.1109/3PGCIC.2015.62.

[4] Nathan F Saraiva De Sousa, Danny A Lachos Perez, Raphael V Rosa, Mateus AS Santos y Christian Esteve Rothenberg. «Network service orchestration: A survey». En: *Computer Communications* 142 (2019), págs. 69-94.

[5] Margaret Chiosi y col. *Network Functions Virtualisation – Introductory White Paper*. Technical Report. ETSI, 2012. URL: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[6] Iván Bernabé, Daniel Díaz-Sánchez y Mario Organero. «Using XMPP to Federate Multimedia UPnP Device». En: *Jornadas JARCA 2011 , 27, 28 y 29 de Junio de 2011 , Huelva* (ene. de 2011).

[7] Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz-Organero. «Optimizing OSGi Services on Gateways». En: *Ambient Intelligence - Software and Applications*. Ed. por Ad van Berlo, Kasper Hallenborg, Juan M. Corchado Rodríguez, Dante I. Tapia y Paulo Novais. Heidelberg: Springer International Publishing, 2013, págs. 155-162. ISBN: 978-3-319-00566-9.

[8] Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero. «Optimizing resources on gateways using OSGi». En: *2012 IEEE International Conference on Consumer Electronics (ICCE)*. 2012, págs. 526-527. DOI: 10.1109/ICCE.2012.6161957.

[9] Iván Bernabé-Sánchez, Daniel Dıaz-Sánchez y Mario Muñoz-Organero. «Specification and Unattended Deployment of Home Networks at the Edge of the Network». En: *IEEE Trans. on Consum. Electron.* 66.4 (nov. de 2020), págs. 279-288. ISSN: 0098-3063. DOI: 10.1109/TCE.2020.3018543. URL: https://doi.org/10.1109/TCE.2020.3018543.

[10] Iván Bernabé Sánchez, Daniel Díaz-Sánchez y Mario Muñoz Organero. «Flex-box: A flexible software architecture for IPTV set-top boxes». En: *2012 IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 2012, págs. 121-125. DOI: 10.1109/ICCE-Berlin.2012.6336480.

[11] Iván Bernabé y Mario Organero. «La Nube y su Papel como Plataforma de Distribución de Contenidos de TV». En: *JARCA Conference, Rota, Spain* (ene. de 2014).

[12] Iván Bernabé, Daniel Díaz-Sánchez y Mario Organero. «Arquitectura IPTV para STB Basada en Capas de Control Jerárquicas.» En: *JARCA 2013* (ene. de 2013).

[13] Iván Bernabé y Mario Organero. «An OSGi platform for Eco-Driving systems». En: *Actas JARCA 2012. Salou.* (ene. de 2012), págs. 11-16.

[14] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher y Valerie Young. «Mobile edge computing—A key technology towards 5G». En: *ETSI white paper* 11.11 (2015), págs. 1-16.

[15] I Morris. «ETSI drops "mobile" from MEC». En: *Light Reading: New York, NY, USA* (2016).

[16] Cisco Fog Computing Solutions. «Unleash the power of the Internet of Things». En: *Cisco Systems Inc* (2015).

[17] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres y Nigel Davies. «The case for vm-based cloudlets in mobile computing». En: *IEEE pervasive Computing* 8.4 (2009), págs. 14-23.

[18] Przemyslaw Pawluk, Bradley Simmons, Michael Smit, Marin Litoiu y Serge Mankovski. «Introducing STRATOS: A cloud broker service». En: *2012 IEEE fifth international conference on cloud computing*. IEEE. 2012, págs. 891-898.

[19] K Hemant Kumar Reddy, Geetika Mudali y Diptendu Sinha Roy. «A novel coordinated resource provisioning approach for cooperative cloud market». En: *Journal of Cloud Computing* 6.1 (2017), págs. 1-17.

[20] He Li, Mianxiong Dong, Kaoru Ota y Minyi Guo. «Pricing and repurchasing for big data processing in multi-clouds». En: *IEEE Transactions on Emerging Topics in Computing* 4.2 (2016), págs. 266-277.

[21] Dan Lin, Anna Cinzia Squicciarini, Venkata Nagarjuna Dondapati y Smitha Sundareswaran. «A cloud brokerage architecture for efficient cloud service selection». En: *IEEE Transactions on Services Computing* 12.1 (2016), págs. 144-157.

[22] Sergio Nesmachnow, Santiago Iturriaga y Bernabe Dorronsoro. «Efficient heuristics for profit optimization of virtual cloud brokers». En: *IEEE Computational Intelligence Magazine* 10.1 (2015), págs. 33-43.

[23] Muhammad Kazim, Lu Liu y Shao Ying Zhu. «A framework for orchestrating secure and dynamic access of IoT services in multi-cloud environments». En: *IEEE Access* 6 (2018), págs. 58619-58633.

[24] Robert B Bohn, John Messina, Fang Liu, Jin Tong y Jian Mao. «NIST cloud computing reference architecture». En: *2011 IEEE World Congress on Services*. IEEE. 2011, págs. 594-596.

[25] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne y Xiaofei Xu. «Web services composition: A decade's overview». En: *Information Sciences* 280 (2014), págs. 218-238.

[26] Jeffrey C Broberg. «Glossary for the OASIS Web Service Interactive Applications (WSIA/WSRP)». En: *Retrieved February* 15 (2002), pág. 2009.

[27] Junchao Li, Ruifeng Guo y Xiuwu Zhang. «Study on service-oriented Cloud conferencing». En: *2010 3rd International Conference on Computer Science and Information Technology*. Vol. 6. IEEE. 2010, págs. 21-25.

[28] Tao Yu y Kwei-Jay Lin. «Service selection algorithms for composing complex services with multiple QoS constraints». En: *International Conference on Service-Oriented Computing*. Springer. 2005, págs. 130-143.

[29] Wenying Zeng, Yuelong Zhao y Junwei Zeng. «Cloud service and service selection algorithm research». En: *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. 2009, págs. 1045-1048.

[30] Ian Foster, Yong Zhao, Ioan Raicu y Shiyong Lu. «Cloud computing and grid computing 360-degree compared». En: *2008 grid computing environments workshop*. Ieee. 2008, págs. 1-10.

[31] Thomas Uphill, John Arundel, Neependra Khare, Hideto Saito, Hui-Chuan Chloe Lee y Ke-Jou Carol Hsu. *DevOps: Puppet, Docker, and Kubernetes*. Packt Publishing Ltd, 2017.

[32] Jeff Geerling. *Ansible for DevOps: Server and configuration management for humans*. Leanpub, 2015.

[33] Charles David Graziano. «A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project». En: (2011).

[34] Susanta Nanda Tzi-cker Chiueh y Stony Brook. «A survey on virtualization technologies». En: *Rpe Report* 142 (2005).

[35] Jorge Carapinha y Javier Jiménez. «Network virtualization: a view from the bottom». En: *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. 2009, págs. 73-80.

[36] Andreas Berl, Andreas Fischer y Hermann de Meer. «Using system virtualization to create virtualized networks». En: *Electronic Communications of the EASST* 17 (2009).

[37] Jeff Daniels. «Server virtualization architecture and implementation». En: *XRDS: Crossroads, The ACM Magazine for Students* 16.1 (2009), págs. 8-12.

[38] Jeanna N Matthews, Eli M Dow, Todd Deshane, Wenjin Hu, Jeremy Bongio, Patrick F Wilbur y Brendan Johnson. *Running Xen: a hands-on guide to the art of virtualization*. Prentice Hall PTR, 2008.

[39] *KVM*. Jun. de 2020. URL: https://www.linux-kvm.org/page/Main_Page.

[40] Valentin Hamburger. *Building VMware Software-Defined Data Centers*. Packt Publishing Ltd, 2016.

[41] Pradyumna Dash. *Getting started with oracle vm virtualbox*. Packt Publishing Ltd, 2013.

[42] Fabrice Bellard. «QEMU, a fast and portable dynamic translator.» En: *USENIX annual technical conference, FREENIX Track*. Vol. 41. California, USA. 2005, pág. 46.

[43] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi, Tiago C Ferreto, Timoteo Lange y Cesar AF De Rose. «Performance evaluation of container-based virtualization for high performance computing environments». En: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2013, págs. 233-240.

[44] D. Bernstein. «Containers and Cloud: From LXC to Docker to Kubernetes». En: *IEEE Cloud Computing* 1.3 (2014), págs. 81-84.

[45] *Docker: Enterprise Application Container Platform*. Jun. de 2020. URL: https://www.docker.com/.

[46] Roberto Morabito, Jimmy Kjällman y Miika Komu. «Hypervisors vs. lightweight virtualization: a performance comparison». En: *2015 IEEE International Conference on Cloud Engineering*. IEEE. 2015, págs. 386-393.

[47] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier y Larry Peterson. «Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors». En: *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*. 2007, págs. 275-287.

[48] Mathijs Jeroen Scheepers. «Virtualization and containerization of application infrastructure: A comparison». En: *21st twente student conference on IT*. Vol. 21. 2014.

[49] Sari Sultan, Imtiaz Ahmad y Tassos Dimitriou. «Container security: Issues, challenges, and the road ahead». En: *IEEE Access* 7 (2019), págs. 52976-52996.

[50] Carl Boettiger. «An introduction to Docker for reproducible research». En: *ACM SIGOPS Operating Systems Review* 49.1 (2015), págs. 71-79.

[51] Wes Felter, Alexandre Ferreira, Ram Rajamony y Juan Rubio. «An updated performance comparison of virtual machines and linux containers». En: *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2015, págs. 171-172.

[52] Rajdeep Dua, A Reddy Raja y Dharmesh Kakadia. «Virtualization vs containerization to support paas». En: *2014 IEEE International Conference on Cloud Engineering*. IEEE. 2014, págs. 610-614.

[53] David Bernstein. «Containers and cloud: From lxc to docker to kubernetes». En: *IEEE Cloud Computing* 1.3 (2014), págs. 81-84.

[54] Sogand Shirinbab, Lars Lundberg y Emiliano Casalicchio. «Performance evaluation of container and virtual machine running cassandra workload». En: oct. de 2017, págs. 1-8. DOI: 10.1109/CloudTech.2017.8284700.

[55] Bo Yi, Xingwei Wang, Keqin Li, Min Huang y col. «A comprehensive survey of network function virtualization». En: *Computer Networks* 133 (2018), págs. 212-262.

[56] Francesco Marino, Luca Maggiani, Laura Nao, Paolo Pagano y Matteo Petracca. «Towards softwarization in the IoT: Integration and evaluation of t-res in the oneM2M architecture». En: *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2017, págs. 1-5.

[57] Sang Il Kim y Hwa Sung Kim. «Semantic ontology-based NFV service modeling». En: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. 2018, págs. 674-678.

[58] Yordanos Tibebu Woldeyohannes, Ali Mohammadkhan, KK Ramakrishnan y Yuming Jiang. «ClusPR: Balancing multiple objectives at scale for NFV resource allocation». En: *IEEE Transactions on Network and Service Management* 15.4 (2018), págs. 1307-1321.

[59] Bin Zhang, Pengfei Zhang, Yusu Zhao, Yongkun Wang, Xuan Luo y Yaohui Jin. «Co-Scaler: Cooperative scaling of software-defined NFV service function chain». En: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2016, págs. 33-38.

[60] Sajad Shirali-Shahreza y Yashar Ganjali. «Protecting Home User Devices with an SDN-Based Firewall». En: 64.1 (feb. de 2018), págs. 92-100. ISSN: 0098-3063. DOI: 10.1109/TCE.2018.2811261.

[61] Hubert Zimmermann. «OSI reference model-the ISO model of architecture for open systems interconnection». En: *IEEE Transactions on communications* 28.4 (1980), págs. 425-432.

[62] Yuan Zhang, Lin Cui, Wei Wang y Yuxiang Zhang. «A survey on software defined networking with multiple controllers». En: *Journal of Network and Computer Applications* 103 (2018), págs. 101-118.

[63] Anton A. Huurdeman. *The Worldwide History of Telecommunications.* Wiley-IEEE Press, 2003.

[64] *NFV White Paper*. Oct. de 2012. URL: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[65] *NFV White Paper 2*. Oct. de 2013. URL: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[66] Jürgen Quittek y col. *Network Functions Virtualisation (NFV); Management and Orchestration*. Technical Report DGS/NFV-MAN001. ETSI, 2014. URL: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

[67] Qiang Duan, Nirwan Ansari y Mehmet Toy. «Software-defined network virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks». En: *IEEE Network* 30.5 (2016), págs. 10-16.

[68] Sourav Kanti Addya, Ashok Kumar Turuk, Bibhudatta Sahoo y Mahasweta Sarkar. «A hybrid queuing model for virtual machine placement in cloud data center». En: *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*. IEEE. 2015, págs. 1-3.

[69] Xuebiao Yuchi y Sachin Shetty. «Towards Network-Topology Aware Virtual Machine Placement in Cloud Datacenters». En: *2016 IEEE World Congress on Services (SERVICES)*. IEEE. 2016, págs. 95-96.

[70] Taekhee Kim, Taehwan Koo y Eunkyoung Paik. «SDN and NFV benchmarking for performance and reliability». En: *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2015, págs. 600-603.

[71] Xin Li y Chen Qian. «An nfv orchestration framework for interference-free policy enforcement». En: *2016 IEEE 36th international conference on distributed computing systems (ICDCS)*. IEEE. 2016, págs. 649-658.

[72] Tarik Taleb y Adlen Ksentini. «Follow me cloud: interworking federated clouds and distributed mobile networks». En: *IEEE Network* 27.5 (2013), págs. 12-19.

[73] Z. Bronstein y E. Shraga. «NFV virtualisation of the home environment». En: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. IEEE, ene. de 2014, págs. 899-904. DOI: 10.1109/CCNC.2014.6940493.

[74] T. Cruz, P. Simões, N. Reis, E. Monteiro y F. Bastos. «An architecture for virtualized home gateways». En: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. Mayo de 2013, págs. 520-526.

[75] N. Herbaut, D. Negru, G. Xilouris e Y. Chen. «Migrating to a NFV-based Home Gateway: Introducing a Surrogate vNF approach». En: *2015 6th International Conference on the Network of the Future (NOF)*. Sep. de 2015, págs. 1-7. DOI: 10.1109/NOF.2015.7333284.

[76] ETSI. *Networks Funtions Virtualisation (NFV); Use Cases*. Technical Report RGR/NFV-001ed121. ETSI, 2017. URL: https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf.

[77] Wenwu Zhu, Chong Luo, Jianfeng Wang y Shipeng Li. «Multimedia cloud computing». En: *IEEE Signal Processing Magazine* 28.3 (2011), págs. 59-69.

[78] Yonggang Wen, Xiaoqing Zhu, Joel JPC Rodrigues y Chang Wen Chen. «Cloud mobile media: Reflections and outlook». En: *IEEE transactions on multimedia* 16.4 (2014), págs. 885-902.

[79] Albert Greenberg, James Hamilton, David A Maltz y Parveen Patel. *The cost of a cloud: research problems in data center networks*. 2008.

[80] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai y Mohit Tawarmalani. «Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud». En: *ACM SIGCOMM Computer Communication Review* 40.4 (2010), págs. 243-254.

[81] Kief Morris. *Infrastructure as Code*. O'Reilly Media, Inc., 2020.

[82] Marjan Mernik, Jan Heering y Anthony M Sloane. «When and how to develop domain-specific languages». En: *ACM computing surveys (CSUR)* 37.4 (2005), págs. 316-344.

[83] Sricharan Vadapalli. *DevOps: continuous delivery, integration, and deployment with DevOps: dive into the core DevOps strategies*. Packt Publishing Ltd, 2018.

[84] Doug Davis y Gilbert Pilz. «Cloud infrastructure management interface (CIMI) model and RESTful HTTP-based protocol». En: *Specification DSP0263* 1 (2012).

[85] Mª González-Madroño Abad, Diana López Collado y Raúl Viudez Corroto. «An Open Cloud Computing Interface (OCCI) Management Console for the OpenNebula Toolkit». En: (2010).

[86] Ankita Atrey, Hendrik Moens, GV Seghbroeck, Bruno Volckaert y FD Turck. «An overview of the OASIS TOSCA standard: Topology and orchestration specification for cloud applications». En: *IBCN-iMinds, Department of Information Technology, Tech. Rep* (2015).

[87] *Cloud infrastructure management interface (cimi) use cases*. Dic. de 2014. URL: https://www.dmtf.org/standards/cloud.

[88] Open OASIS. «Topology and orchestration specification for cloud applications version 1.0». En: *Retrieved October* 10 (2013), pág. 2017.

[89] *OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 Committee Specification Draft 03 (2012)*. Jun. de 2021. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd03/TOSCA-v1.0-csd03.html.

[90] *TOSCA Simple Profile in YAML Version 1.0*. 2016. URL: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/os/TOSCA-Simple-Profile-YAML-v1.0-os.pdf.

[91] Thomas Spatzier Gerd Breiter Frank Leymann. *Topology and Orchestration Specification for Cloud Applications (TOSCA): Cloud Service Archive (CSAR)*. Technical Report. IBM, 2012.

[92] A. Edmonds, T. Metsch, A. Papaspyrou y A. Richardson. «Toward an Open Cloud Standard». En: *IEEE Internet Computing* 16.4 (2012), págs. 15-25. DOI: 10.1109/MIC.2012.65.

[93] *Open Cloud Computing Interface (OCCI)*. Jun. de 2021. URL: https://occi-wg.org/about/index.html.

[94] Storage Networking Industry Association y col. *Information Technology.—Cloud Data Management Interface (CDMI)*.

[95] S Crosby, R Doyle, M Gering, M Gionfriddo, S Grarup, S Hand, M Hapner, D Hiltgen y col. «Open virtualization format specification». En: *vol. DSP0243* 1.0 (2010).

[96] Eliza Varney. «Distributed Management Task Force, Inc». En: *Handbook of Transnational Economic Governance Regimes*. Brill Nijhoff, 2010, págs. 545-551.

[97] Mark Carlson, Martin Chapman, Alex Heneveld, Scott Hinkelman, Duncan Johnston-Watt, Anish Karmarkar, Tobias Kunze, Ashok Malhotra, Jeff Mischkinsky, Adrian Otto y col. «Cloud application management for platforms». En: *OASIS, http://cloudspecs. org/camp/CAMP-v1. 0. pdf, Tech. Rep* (2012).

[98] Delta Cloud. «Delta cloud-many clouds. one api. no problem». En: *accessed Feb* 28 (2017).

[99] *Apache libcloud*. Mayo de 2021. URL: http://libcloud.apache.org/.

[100] Daniel Molina, Carlos Martın Sánchez, Jaime Melis, Javier Fontán, Constantino Vázquez, Ruben S Montero e Ignacio M Llorente. «The OpenNebula Cloud Toolkit». En: *Open Source Cloud Computing Systems: Practices and Paradigms*. IGI Global, 2012, págs. 19-43.

[101] Xiaolong Wen, Genqiang Gu, Qingchun Li, Yun Gao y Xuejie Zhang. «Comparison of open-source cloud management platforms: OpenStack and OpenNebula». En: *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE. 2012, págs. 2457-2461.

[102] Srivatsan Jagannathan. *Comparison and Evaluation of Open-Source Cloud Management Software*. 2012.

[103] Thiago Damasceno Cordeiro, Douglas Brito Damalio, Nadilma Cintra Valenca Nunes Pereira, Patricia Takako Endo, André de Almeida Palhares, Glauco Estacio Gonçalves, Djamel Fawzi Hadj Sadok, Judith Kelner, Bob Melander, Victor Souza y col. «Open source cloud computing platforms». En: *2010 Ninth International Conference on Grid and Cloud Computing*. IEEE. 2010, págs. 366-371.

[104] Yacine Kessaci, Nouredine Melab y El-Ghazali Talbi. «A multi-start local search heuristic for an energy efficient VMs assignment on top of the OpenNebula cloud manager». En: *Future Generation Computer Systems* 36 (2014), págs. 237-256.

[105] *Opennebula*. 2021. URL: https://github.com/OpenNebula/one.

[106] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente y Ian Foster. «Virtual infrastructure management in private and hybrid clouds». En: *IEEE Internet computing* 13.5 (2009), págs. 14-22.

[107] *What is openstack?* 2021. URL: https://www.openstack.org/software/.

[108] *Open Source Cloud Computing Platform Software - OpenStack*. 2021. URL: https://docs.openstack.org/newton/install-guide-rdo/common/get-started-conceptual-architecture.html.

[109] D Michelino, JC Leon y LF Alvarez. *Implementation and testing of openstack heat*. 2013.

[110] Teodora Sechkova, Michele Paolino y Daniel Raho. «Virtualized infrastructure managers for edge computing: Openvim and openstack comparison». En: *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE. 2018, págs. 1-6.

[111] Doug Johnson. «Computing in the Clouds». En: *Learning & Leading with Technology* 37.4 (2010), págs. 16-20. DOI: ISSN-1082-5754.

[112] Christof Ebert, Gorka Gallardo, Josune Hernantes y Nicolas Serrano. «DevOps». En: *Ieee Software* 33.3 (2016), págs. 94-100.

[113] Michael Brenner y Markus Gillmeister. «Designing CMDB data models with good utility and limited complexity». En: *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE. 2014, págs. 1-15.

[114] Ian Horrocks. «What are ontologies good for?» En: *Evolution of Semantic Systems*. Springer, 2013, págs. 175-188.

[115] Cabinet Office. *ITIL Continual Service Improvement 2011 Edition*. The Stationery Office, 2011.

[116] *Cloudify*. 2021. URL: http://cloudify.co/.

[117] Kirill Shirinkin. *Getting Started with Terraform*. Packt Publishing Ltd, 2017.

[118] Fabrizio Soppelsa y Chanwit Kaewkasi. *Native Docker Clustering with Swarm*. Packt Publishing Ltd, 2016.

[119] Brendan Burns, Joe Beda y Kelsey Hightower. *Kubernetes*. Dpunkt, 2018.

[120] *Packer by HashiCorp*. Mayo de 2021. URL: https://www.packer.io/.

[121] Moshe Zadka. «Salt Stack». En: *DevOps in Python*. Springer, 2019, págs. 121-137.

[122] Mark Burgess. «A tiny overview of cfengine: Convergent maintenance agent». En: *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO*. Vol. 2005. 2005.

[123] *AWS CloudFormation - Infraestructura como código y aprovisionamiento de recirsos de AWS*. Mayo de 2021. URL: https://aws.amazon.com/es/cloudformation/.

[124] *AWS Cloud compunting - Servicios de informática en la nube*. Mayo de 2021. URL: https://aws.amazon.com/es/.

[125] *Información general del Administrador de recursos de Azure - Azure Resource Manager*. Mayo de 2021. URL: https://docs.microsoft.com/es-es/azure/azure-resource-manager/management/overview.

[126] YVRK Prasad, KP Rao y S Sasidhar. *Hot working guide: a compendium of processing maps*. ASM international, 2015.

[127] Yevgeniy Brikman. *Terraform: Up & Running: Writing Infrastructure as Code*. O'Reilly Media, 2019.

[128] Charles Butler. «Automating Orchestration in the Cloud with Ubuntu Juju». En: (2014).

[129] *Network Functions Virtualisation (NFV)—Architectural Framework*. 2021. URL: Https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[130] Network Functions Virtualisation. «Management and Orchestration». En: *ETSI GS NFV-MAN* 1 (2014), pág. V1.

[131] Network Functions Virtualisation. *ETSI Industry Specification Group (ISG),"ETSI GS NFV-MAN 001 V1. 1.1: Network Functions Virtualisation (NFV); Management and Orchestration,"* 2014.

[132] AWS Amazon. *Amazon Web Services (AWS)-Cloud Computing Services*. 2020.

[133] *Opendaylight:Home*. Jun. de 2020. URL: https://www.opendaylight.org/.

[134] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow y col. «ONOS: towards an open, distributed SDN OS». En: *Proceedings of the third workshop on Hot topics in software defined networking*. 2014, págs. 1-6.

[135] Larry Peterson, Scott Baker, Marc De Leenheer, Andy Bavier, Sapan Bhatia, Mike Wawrzoniak, Jude Nelson y John Hartman. «XoS: An Extensible Cloud Operating System». En: *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*. BigSystem '15. Portland, Oregon, USA: Association for Computing Machinery, 2015, págs. 23-30. ISBN: 9781450335683. DOI: 10.1145/2756594.2756598. URL: https://doi.org/10.1145/2756594.2756598.

[136] *CORD: Re-inventing Central Offices for Efficiency and Agility*. Abr. de 2021. URL: https://opencord.org/.

[137] *Onap Home. Open Network Automation Platform*. Jun. de 2020. URL: https://www.onap.org/.

[138] *Open-O - Open Orchestrator Project*. Abr. de 2021. URL: https://www.open-o.org.

[139] *Enhanced Control Orchestration Management and Policy) Architecture White Paper*. Technical Report. AT&T, 2021.

[140] *OSM. Open Source Mano*. Abr. de 2021. URL: https://osm.etsi.org/.

[141] Martin Bjorklund. *The YANG 1.1 data modeling language*. Inf. téc. RFC 7950, August 2016, 2016.

[142] Jürgen Schönwälder. *Common yang data types*. Inf. téc. RFC 6021, October, 2010.

[143] *Open Baton: An open source reference implementation of the ETSI Network Function Virtualization MANO specification*. Abr. de 2021. URL: https://openbaton.github.io/.

[144] *Topology and Orchestration Specification for Cloud Applications Version 1.0*. Technical Report TOSCA-v1.0. OASIS Open, nov. de 2013. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[145] Antonio Francescon, Giovanni Baggio, Riccardo Fedrizzi, Ramon Ferrusy, Imen Grida Ben Yahiaz y Roberto Riggio. «X–MANO: Cross–domain management and orchestration of network services». En: *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2017, págs. 1-5.

[146] *Gohan-REST-based api server to evolve your cloud service very rapidly*. Abr. de 2021. URL: http://gohan.cloudwan.io/.

[147] J. Chen, Y. Chen, S. Tsai e Y. Lin. «Implementing NFV system with OpenStack». En: *2017 IEEE Conference on Dependable and Secure Computing*. 2017, págs. 188-194.

[148] Jordi Ferrer Riera, Josep Batallé, José Bonnet, Miguel Dıas, Michael McGrath, Giuseppe Petralia, Francesco Liberati, Alessandro Giuseppi, Antonio Pietrabissa, Alberto Ceselli y col. «TeNOR: Steps towards an orchestration platform for multi-PoP NFV deployment». En: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE. 2016, págs. 243-250.

[149] ETSI. *GS NFV-MANoo1-v1.1.1: Network Functions Virtualisation (NFV); management and orchestration*. Technical Report. ETSI, 2021. URL: http://www.etsi.org/deliver/etsigs/NFV-MAN/001_099/001/01.01.0160/gsNFVMAN001v010101p.pdf.

[150] Georgios Xilouris, Eleni Trouva, Felicia Lobillo, João M Soares, Jorge Carapinha, Michael J McGrath, George Gardikis, Pietro Paglierani, Evangelos Pallis, Letterio Zuccaro y col. «T-NOVA: A marketplace for virtualized network functions». En: *2014 European Conference on Networks and Communications (EuCNC)*. IEEE. 2014, págs. 1-5.

[151] Michail-Alexandros Kourtis, Michael J McGrath, Georgios Gardikis, Georgios Xilouris, Vincenzo Riccobene, Panagiotis Papadimitriou, Eleni Trouva, Francesco Liberati, Marco Trubian, Josep Batallé y col. «T-nova: An open-source mano stack for nfv infrastructures». En: *IEEE Transactions on Network and Service Management* 14.3 (2017), págs. 586-602.

[152] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck y Raouf Boutaba. «Network function virtualization: State-of-the-art and research challenges». En: *IEEE Communications surveys & tutorials* 18.1 (2015), págs. 236-262.

[153] Fraunhofer Fokus y B Tu. *Open baton: an open source reference implementation of the etsi network function virtualization mano specification*. 2018.

[154] *Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification*. Jun. de 2021. URL: https://openbaton.github.io/.

[155] *Tacker - OpenStack*. 2021. URL: https://wiki.openstack.org/wiki/Tacker.

[156] *Getting started Cloudify*. 2019. URL: https://cloudify.co/getting-started/.

[157] Johannes Wettinger, Uwe Breitenbücher y Frank Leymann. «Compensation-based vs. convergent deployment automation for services operated in the cloud». En: *International Conference on Service-Oriented Computing*. Springer. 2014, págs. 336-350.

[158] Alessio Gambi, Waldemar Hummer, Hong-Linh Truong y Schahram Dustdar. «Testing elastic computing systems». En: *IEEE Internet Computing* 17.6 (2013), págs. 76-82.

[159] Michael Hüttermann. *DevOps for developers*. Apress, 2012.

[160] Mark Burgess y Alva L Couch. «Modeling Next Generation Configuration Management Tools.» En: *LISA*. 2006, págs. 131-147.

[161] Diomidis Spinellis. «Don't install software by hand». En: *IEEE software* 29.4 (2012), págs. 86-87.

[162] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar y P Brighten Godfrey. «Veriflow: Verifying network-wide invariants in real time». En: *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 2013, págs. 15-27.

[163] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story y David Walker. «Frenetic: A network programming language». En: *ACM Sigplan Notices* 46.9 (2011), págs. 279-291.

[164] Federico M Facca, Elio Salvadori, Holger Karl, Diego R López, Pedro Andrés Aranda Gutiérrez, Dejan Kostic y Roberto Riggio. «NetIDE: First steps towards an integrated development environment for portable network apps». En: *2013 Second European Workshop on Software Defined Networks*. IEEE. 2013, págs. 105-110.

[165] András Császár, Wolfgang John, Mario Kind, Catalin Meirosu, Gergely Pongrácz, Dimitri Staessens, Attila Takács y Fritz-Joachim Westphal. «Unifying cloud and carrier network: Eu fp7 project unify». En: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE. 2013, págs. 452-457.

[166] Tamás Lévai, István Pelle, Felicián Németh y András Gulyás. «EPOXIDE: a modular prototype for SDN troubleshooting». En: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, págs. 359-360.

[167] Matthias Keller, Manuel Peuster, Christoph Robbert y Holger Karl. «A topology-aware adaptive deployment framework for elastic applications». En: *2013 17th International Conference on Intelligence in Next Generation Networks (ICIN)*. IEEE. 2013, págs. 61-69.

[168] M. A. Kourtis y col. «T-NOVA: An Open-Source MANO Stack for NFV Infrastructures». En: *IEEE Trans. Netw. Service Manag.* 14.3 (sep. de 2017), págs. 586-602. ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2733620.

[169] Pontus Sköldström, Balázs Sonkoly, András Gulyás, Felicián Németh, Mario Kind, Fritz-Joachim Westphal, Wolfgang John, Jokin Garay, Eduardo Jacob, Dávid Jocha y col. «Towards unified programmability of cloud and carrier infrastructure». En: *2014 Third European Workshop on Software Defined Networks*. IEEE. 2014, págs. 55-60.

[170] DM Baek, SH Yoon y BC Lee. «Analysis of NFV Orchestrator and Openmano». En: *Electronics and Telecommunications Trends* 30.6 (2015), págs. 58-67.

[171] *TeNOR:T-NOVA Orchestrator*. 2021. URL: https://github.com/T-NOVA/TeNOR.

[172] NFVISG ETSI. *GS NFV-MAN 001 V1. 1.1 Network Function Virtualisation (NFV); Management and Orchestration*. 2014.

[173] Joao Soares, Miguel Dias, Jorge Carapinha, Bruno Parreira y Susana Sargento. «Cloud4nfv: A platform for virtual network functions». En: *2014 IEEE 3Rd international conference on cloud networking (cloudnet)*. IEEE. 2014, págs. 288-293.

[174] Wenyu Shen, Masahiro Yoshida, Kenji Minato y Wataru Imajuku. «vConductor: An enabler for achieving virtual network integration as a service». En: *IEEE Communications Magazine* 53.2 (2015), págs. 116-124.

[175] M. Guerriero, M. Garriga, D. A. Tamburri y F. Palomba. «Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry». En: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019, págs. 580-589. DOI: 10.1109/ICSME.2019.00092.

[176] Khaleel Ahmad y Masroor Ansari. «Hands-On Vagrant». En: *Design and Use of Virtualization Technology in Cloud Computing*. IGI Global, 2018, págs. 208-220.

[177] Ralph Droms y col. *Dynamic host configuration protocol*. 1997.

[178] Paul Mockapetris. «RFC 1035—Domain names—implementation and specification, November 1987». En: *URL http://www. ietf. org/rfc/rfc1035. txt* (2004).

[179] Erik Guttman, Charles Perkins, John Veizades y Michael Day. *Service location protocol, version 2*. 1999.

[180] Dong-Sung Kim, Jae-Min Lee, Wook Hyun Kwon e In Kwan Yuh. «Design and implementation of home network systems using UPnP middleware for networked appliances». En: *IEEE Transactions on Consumer Electronics* 48.4 (2002), págs. 963-972.

[181] Biniam Gebremichael, Frits Vaandrager y Miaomiao Zhang. «Analysis of the Zeroconf protocol using Uppaal». En: *Proceedings of the 6th ACM & IEEE International conference on Embedded software*. 2006, págs. 242-251.

[182] Jim Waldo. «The Jini architecture for network-centric computing». En: *Communications of the ACM* 42.7 (1999), págs. 76-82.

[183] Jung-Tae Kim, Yeon-Joo Oh, Hoon-Ki Lee, Eui-Hyun Paik y Kwang-Roh Park. «Implementation of the DLNA proxy system for sharing home media contents». En: *IEEE Transactions on Consumer Electronics* 53.1 (2007), págs. 139-144.

[184] Daniel Diaz-Sanchez, Fabio Sanvido, Davide Proserpio y Andres Marin. «DLNA, DVB-CA and DVB-CPCM integration for commercial content management». En: *IEEE Transactions on Consumer Electronics* 56.1 (2010), págs. 79-87. DOI: 10.1109/TCE.2010.5439129.

[185] Peter Saint-Andre, Kevin Smith, Remko Tronçon y Remko Troncon. *XMPP: the definitive guide*. .ºReilly Media, Inc.", 2009.

[186] *Extensible Messaging and Presence Protocol (XMPP): Core*. Abr. de 2021. URL: https://tools.ietf.org/html/rfc3920.

[187] Pyda Srisuresh y Matt Holdrege. *RFC2663: IP Network Address Translator (NAT) Terminology and Considerations*. 1999.

[188] Dan Wing, Philip Matthews, Rohan Mahy y Jonathan Rosenberg. «Session traversal utilities for NAT (STUN)». En: *RFC5389, October* (2008).

[189] *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. Abr. de 2021. URL: https://tools.ietf.org/html/rfc5766.

[190] Dan C. Marinescu. «Chapter 2 - Parallel and Distributed Systems». En: *Cloud Computing*. Ed. por Dan C. Marinescu. Boston: Morgan Kaufmann, 2013, págs. 21-65. ISBN: 978-0-12-404627-6. DOI: https://doi.org/10.1016/B978-0-12-404627-6.00002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780124046276000026.

[191] Peter Saint-Andre. «Jingle: Jabber does multimedia». En: *IEEE MultiMedia* 14.1 (2007), págs. 90-94.

[192] Eve Schooler, Jonathan Rosenberg, Henning Schulzrinne, Alan Johnston, Gonzalo Camarillo, Jon Peterson, Robert Sparks y Mark J. Handley. *SIP: Session Initiation Protocol*. RFC 3261. Jul. de 2002. DOI: 10.17487/RFC3261. URL: https://rfc-editor.org/rfc/rfc3261.txt.

[193] Peter Saint-Andre, Avshalom Houri y Joe Hildebrand. *Interworking between the Session Initiation Protocol (SIP) and the Extensible Messaging and Presence Protocol (XMPP): Architecture, Addresses, and Error Handling*. RFC 7247. Mayo de 2014. DOI: 10.17487/RFC7247. URL: https://rfc-editor.org/rfc/rfc7247.txt.

[194] Henning Schulzrinne, Stephen L. Casner, Ron Frederick y Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. Jul. de 2003. DOI: 10.17487/RFC3550. URL: https://rfc-editor.org/rfc/rfc3550.txt.

[195] *XEP-0166: Jingle*. Abr. de 2021. URL: https://xmpp.org/extensions/xep-0166.html.

[196] Andre LC Tavares y Marco Tulio Valente. «A gentle introduction to OSGi». En: *ACM SIGSOFT Software Engineering Notes* 33.5 (2008), págs. 1-5.

[197] Carlo Maria Medaglia y Alexandru Serbanati. *An Overview of Privacy and Security Issues in the Internet of Things*. Undetermined. DOI: 10.1007/978-1-4419-1674-7_38.

[198] Rolf H. Weber. «Internet of Things – New security and privacy challenges». En: *Computer Law & Security Review* 26.1 (2010), págs. 23-30. ISSN: 0267-3649. DOI: http://doi.org/10.1016/j.clsr.2009.11.008. URL: http://www.sciencedirect.com/science/article/pii/S0267364909001939.

[199] Federal Trade Commission y col. «Internet of Things: Privacy & security in a connected world». En: *Washington, DC: Federal Trade Commission* (2015).

[200] Siamak Sarmady. *A Survey on Peer-to-Peer and DHT*. 2010. arXiv: 1006.4708 [cs.DC].

[201] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley y W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). Internet Engineering Task Force, mayo de 2008. URL: http://www.ietf.org/rfc/rfc5280.txt.

[202] Mario Muñoz Organero Iván Bernabé Sánchez Daniel Díaz Sánchez. «Using XMPP to Federate Multimedia UPnP Device». En: *Jornadas JARCA* (2011).

[203] Iván Bernabé Sánchez, Daniel Dıaz-Sánchez y Mario Muñoz Organero. «Optimizing resources on gateways using OSGi». En: *2012 IEEE international conference on consumer electronics (ICCE)*. IEEE. 2012, págs. 526-527.

[204] Edwin A Heredia. *An introduction to the DLNA architecture: network technologies for media devices*. John Wiley & Sons, 2011.

[205] C. K. Dominicini, G. L. Vassoler, M. R. N. Ribeiro y M. Martinello. «VirtPhy: A fully programmable infrastructure for efficient NFV in small data centers». En: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, nov. de 2016, págs. 81-86. DOI: 10.1109/NFV-SDN.2016.7919480.

[206] Sophia Antipolis. *NFV 002 V1.2.1 Network Functions Virtualisation (NFV); Architectural Framework, ETSI ISG*. Technical Report. ETSI, 2004.

[207] H. Hong y D. Lee. «A personalized refinement technique for semantic multimedia content search in smart TV». En: 61.4 (nov. de 2015), págs. 581-587. ISSN: 0098-3063. DOI: 10.1109/TCE.2015.7389815.

[208] Rob Enns, Martin Bjorklund, Juergen Schoenwaelder y Andy Bierman. *Network configuration protocol (NETCONF)*. Inf. téc. Internet Engineering Task Force (IETF), 2011.

[209] Martin Bjorklund. *YANG-a data modeling language for the network configuration protocol (NETCONF)*. Inf. téc. Internet Engineering Task Force (IETF), 2010.

[210] S. I. Kim y H. S. Kim. «Semantic Ontology-Based NFV Service Modeling». En: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, jul. de 2018, págs. 674-678. DOI: 10.1109/ICUFN.2018.8436738.

[211] R. Abbas, Z. Sultan y S. N. Bhatti. «Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege». En: *2017 International Conference on Communication Technologies (ComTech)*. 2017, págs. 39-44.

Fe de erratas

Tesis doctoral:
“Diseño de una Arquitectura Dinámica para Set-Top Box Multi Proveedor de Servicios”

En la página iii, donde dice “y fue el contribuidor principal” debe decir “ y fue el contribuyente principal”. Lo mismo ocurre en la página iv, v y vi

En la página 13, donde dice “los diferentes tipos de de” debe decir “los diferentes tipos de”.

En la página 15, donde dice “de microservicios en el dearrollo de aplicaicones ” debe decir “de microservicios en el desarrollo de aplicaciones ”.

En la página 18, donde dice “encarga de proporcionar el servicios más adecuando” debe decir “encarga de proporcionar el servicio más adecuado”.

En la página 28, donde dice “sistemas operativos instanciados instanciados” debe decir “sistemas operativos instanciados”.

En la página 29, donde dice “a penas” debe decir “apenas”.

En la página 31, donde dice “encapsulación de los servicio” debe decir “encapsulación de los servicios”.

En la página 64, donde dice:
“Estos niveles están relacionados unos con otros proporcionando herramientas y soporte al nivel superior, el cual no podría funcionar sin el nivel inferior.” debe decir:
“Estos niveles están relacionados unos con otros, donde cada nivel proporciona funcionalidades y capacidades al nivel inmediatamente superior, de esta manera el software desplegado en la capa superior dispone de las funciones, servicios y configuraciones necesarias para el correcto funcionamiento del software instalado en dicha capa.”.

En la página 72, donde dice “entre octubre de 2018 hasta” debe decir “entre diciembre de 2019 hasta”.

En la página 84, donde dice “El framework se denomina MANO” debe decir “El framework se denomina NFV MANO (management and network orchestration)”.

En la página 100, donde dice “PRINCIPOS DE IMPLEMENTACIÓN PARA IOC” debe decir “PRINCIPOS DE IMPLEMENTACIÓN PARA IAC”.

En la página 100, donde dice “presentan‘ los” debe decir “presentan los”.

En la página 118, donde dice:
“La ecuación obtiene una puntuación (SRP) sumando el valor de las puntuaciones de las diferentes entradas que coinciden con el propósito del bundle buscado. Esas puntuaciones se filtran con la expresión D(mR, RS) las cuales utilizan el valor mínimo de reputación para aceptar una recomendación (mR) y la reputación del recomendador (RS). Esta función devuelve 1 cuando mR es mayor que RS y 0 de lo con obtiene una puntuación (SRP) sumando el valor de las puntuaciones de las diferentes entradas que coinciden con el propósito del bundle buscado. Esas puntuaciones se filtran con la expresión D(mR, RS) las cuales utilizan el valor mínimo de reputación para aceptar una recomendación (mR) y la reputación del recomendador (RS). Esta función devuelve 1 cuando

mR es mayor que RS y 0 de lo contrario. Si SRP es mayor que los umbrales registrados, el optimizador actualizaría o sustituiría el componente.” debe decir:

“La ecuación obtiene una puntuación (SRP) sumando el valor de las puntuaciones de las diferentes entradas que coinciden con el objetivo del bundle buscado. Esas puntuaciones se filtran con la expresión D(mR, RS) en donde se utiliza el valor mínimo de reputación para aceptar una recomendación (mR) y la reputación del recomendador (RS). Esta función devuelve 1 cuando mR es mayor que RS y devuelve 0 en caso contrario. Si SRP es mayor que los umbrales registrados, el optimizador actualizaría o sustituiría el componente.”.

En la página 123, donde dice:
“d) seguridad incrementada: como consecuencia de la simplificación del dispositivo y de la reducción del software susceptible de ser actualizado, se consigue reducir la superficie de ataque del dispositivo y se incrementa la capacidad de corrección de vulnerabilidades al ser el software objetivo más sencillo.” debe decir:
“d) mayor seguridad: este punto es discutible desde diferentes puntos de vistas pero en este trabajo tan sólo queremos resaltar la posibilidad de que la seguridad de un dispositivo virtualizado tiene dos puntos fuertes en este sentido. Por un lado, el dispositivo físico de capacidades reducidas no puede soportar muchas aplicaciones y servicios software ejecutándose sobre el mismo, por lo que se reduce el número de vulnerabilidades debido a que cuanto menos aplicaciones o servicios se tenga instalado menor es el riesgo de que puedan contener alguna vulnerabilidad. Por otro lado, un dispositivo virtualizado en la nube puede extender sus capacidades a través de la nube, por lo que es más fácil que a través de la instalación de software adicional, se pueda incorporar nuevos mecanismos de seguridad que disminuyan las vulnerabilidades del software instalado sobre el dispositivo físico.”.

En la página 124, donde dice “Infrastructura de Orquestación e Implementación (INFrastructure Orchestration and Deployment system, INFORD)” debe decir “Infrastructura de Orquestación y despliegue (INFrastructure Orchestration and Deployment system, INFORD)”.

En la página 153, donde dice “Cada herramienta una de las herramientas” debe decir “Cada una de las herramientas”.

En la página 205, donde dice “para facilitar le mover servicios” debe decir “para facilitar el mover servicios”.

En la página 205, donde dice “servicios que van a ser desplegado” debe decir “servicios que van a ser desplegados”.