Universitat Oberta de Catalunya

Nubes personales, ¿Internet 2.0?

Autor	Javier Velasco Blasco
Titulación	Ingeniería Telecomunicaciones - Telemática
Consultor	Félix Freitag

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	Nubes personales, ¿Internet 2.0?
Nombre del autor:	Javier Velasco Blasco
Nombre del consultor:	Félix Freitag
Fecha de entrega:	06/2016
Área del Trabajo Final:	Aplicaciones y sistemas distribuidos
Titulación:	*Ingeniería Telecomunicaciones - Telemática*

Resumen del Trabajo:

Este trabajo de investigación ha tenido como objetivo principal el estudio del estado del arte en relación al campo de la computación en nube aplicada específicamente al entorno del hogar. Debido al aparentemente escaso interés actual por este modelo de sistema distribuido en el ámbito personal no existe una definición clara de lo que realmente significa y debe requerir una arquitectura de este tipo. Por ello a lo largo del trabajo se ha intentado delimitar y especificar en qué debe consistir un modelo de nube para este ámbito y se han analizado las actuales posibilidades dentro de este tipo de arquitectura de sistemas que pudiesen adaptarse a esta especificación.
Se ha utilizado para ello como hilo conductor el sistema de virtualización basado en contenedores, *Docker.* Este modelo está tomando cada vez más fuerza en los entornos corporativos debido a su flexibilidad y rendimiento, y gracias a su simplicidad parece ser un candidato perfecto para la creación de la 'Nube personal', tal y como se ha optado por llamarla en este trabajo.

Debido a ámbito en el cual se desarrolla este trabajo, el hogar, se ha considerado pertinente la realización de un trabajo que pudiese llegar al mayor número de personas, y no únicamente a audiencias técnicas. Por ello como objetivo secundario este trabajo intenta manejar un lenguaje de fácil compresión realizando un breve repaso por la historia y los conceptos más importantes que han desembocado en el estado actual de esta tecnología.

Abstract:

As a main goal of this research project I have studied the state of the art in cloud computing, specifically applied to the home field. Due to the current lack of interest in cloud distributed system model applied to home users there is not any clear definition of the meaning and requirements for this kind of system architecture. Is because of this throughout this document I have tried to demarcate and define what it means home/personal cloud. To do this I have made a research of different options in the market that could be applied to the proposed definition.
I have used Docker System as a driving force that has lead us to accomplish this objective. This open virtualization software is becoming very popular inside corporative environments due to its flexibility and good performance. As well thanks to its simplicity it seems a perfect candidate to be a key part

of home/personal cloud.

Due to the sphere in which this work is involved, home, it has been considered important as a secondary goal the use of an easy language since the audience target is not limited to people with deep technical skills. So I have defined basic concepts conducting the reader through the most important milestones in the history of this technology until today.

Palabras clave:

Nube, personal, doméstica, Docker, sistema distribuido, virtualización

Índice

Índice 5
Convenciones usadas en este trabajo 7
Introducción 8
Evolución de Internet 11
Hacia la nube 14
Necesidad de la nube 22
Tipos de nube 24
Inconvenientes 26
Nube personal 28
Dockers 31
Perspectiva de usuario 35
Prueba de concepto 36
Instalación de Dockers 38
Herramientas básicas 41
Volúmenes 44
Orquestación 46
Posibles problemas 51
Análisis de las distintas opciones 53
Análisis del sistema Dockers 55
Descripción 55
Implementación de Dockers 55
Posibilidades 56
El futuro 57
Conclusión 61
Glosario 62
Bibliografía 63
Índice de figuras ..... **Error! Bookmark not defined.**

Índice de figuras

Figura 1 – *Internet.* 8
Figura 2 – *Internet 2.0* 9
Figura 3 – *Evolución de las redes* 11
Figura 4 – *SDN* 12
Figura 5 – *Evolución de la computación* 14
Figura 6 – *Sistema monolítico* 15
Figura 7 – *Virtualización clásica I* 16
Figura 8 – *Virtualización clásica II* 17
Figura 9 – *Evolución del CPD* 21
Figura 10 – Evolución de las comunicaciones y la computación 22
Figura 11 – Consumo de ancho de banda por tecnología. *Fuente: [tCaYS]* 24
Figura 12 – *Tipos de servicios de nube.* 25
Figura 13 – *Taxonomía de la nube* 26
Figura 14 – *Arquitectura Docker* 33
Figura 15 - *Evolución* 34
Figura 16 – *Conocimiento técnico de la población* 35
Figura 17 – *Dockers: Funcionamiento básico* 36
Figura 18 – *Entorno de pruebas* 37
Figura 19 – *Kubernetes: Esquema básico* 46
Figura 20 – *Kubernetes: Esquema completo para un host* 47
Figura 21 – *Objetivos de inversión* 58
Figura 22 – *Comparativa en el desarrollo de software de pago y libre* 59

Convenciones usadas en este trabajo

Las convenciones utilizadas en este documento son las siguientes:

Cursiva

- Los nombres propios en el texto referentes a software así como las referencias tecnológicas se escribirán en cursiva.

`Consolas`

- Los comandos, salidas de comandos, nombres de ficheros o contenidos de ficheros a los que se hagan referencia dentro del texto se escribirán en calabaza con el tipo de letra Consolas. Ejemplo:

  "Aunque la lista completa puede consultarse a través de `docker –help` aquí incluyo los más importantes."

Enmarcado

- Los ejemplos donde aparezcan comandos vendrán enmarcados con un fondo gris y el comando incluirá el símbolo de usuario que aparece en el *prompt*. Si existe salida cuando se ejecuta ese comando esta se escribirá en gris oscuro todo con el tipo de letra Consolas. Ejemplo:

```
# uname -a
Linux debian 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1+deb8u4 (2016-02-29) x86_64 GNU/Linux
```

Introducción

Nuestra forma de comunicación sufrió un cambio radical a raíz de la aparición de Internet. El primer aspecto evidente de este cambio se caracterizó por la capacidad de acceder a la información de manera inmediata y actualizada, en muchos casos, a información que hasta entonces no había sido fácil de conseguir.
Tecnológicamente hablando se han ido sucediendo una tras otra diferentes evoluciones que han sacudido los paradigmas iniciales de Internet (nuevos y más eficientes protocolos, virtualización de redes y servicios, etc.), lo cual nos ha conducido finalmente a un nuevo nivel de abstracción tanto a nivel de red, SDN, como a nivel de sistemas, la nube.

Ahora bien desde el punto de vista del usuario estos conceptos no tienen cabida ya que su labor se desarrolla en una capa superior totalmente disociada de estas complejidades. Atendiendo a este punto de vista el usuario ha vivido dos grandes transformaciones hasta la fecha:

- Acceso a Internet: El usuario comenzó a utilizar la red digital para acceder a contenidos estáticos. Esto expandió las posibilidades del conocimiento en la sociedad moderna pero limitaba el acceso a una información no modificable por el usuario.
- *Web 2.0*: Este nuevo concepto de internet produce una revolución en la forma en la que el usuario comienza a interactuar con Internet. La red se convierte en un lugar en el que se intercambia información, se dan opiniones, se realizan peticiones, en resumen, el comportamiento del usuario determina el contenido al que accede. Internet deja de ser monolítico e inalterable.

En cualquier caso se hace evidente que hasta el momento, independientemente de las distintas mejoras y adición de nuevas capacidades, el sentido de la relación del usuario común con Internet ha sido precisamente ese, el de un usuario accediendo a servicios de algún tipo existentes en el mundo digital más allá de las fronteras de su casa como esquematiza la figura:

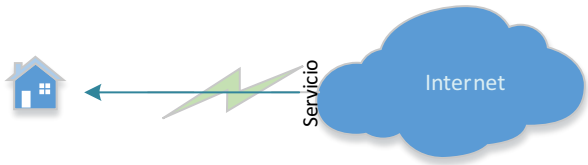

Figura 1 – *Internet.*
La figura muestra donde se encuentra el consumidor y el productor en Internet actualmente

Pero en este momento parece acercarse un nuevo cambio de paradigma en esta comunicación, la nube personal.

A día de hoy ya es posible que un individuo pueda implementar en su propia casa un sistema que le permita compartir contenidos o servicios creados por él con el resto del mundo digital y con cierto nivel de disponibilidad. El usuario ya no tiene la necesidad de ser relegado a un mero consumidor sino que se convierte en proveedor de algún servicio. A partir de este momento cambiamos el modelo de comunicación para permitir a otros usuarios de Internet acceder a servicios o contenidos creados por otros usuarios en sus propios hogares.

Es cierto que ya hace tiempo que una persona puede crear un servicio en su ordenador personal, pongamos como ejemplo una web, y ofrecérselo al resto del mundo digital. Pero la diferencia a día de hoy es el gran número de posibilidades que existen para ir más allá y facilitar la alta disponibilidad de estos servicios personales, o incluso la colaboración entre usuarios que están en sus casas para trabajar en un entorno común tal y como hasta ahora se hacía a nivel corporativo o universitario tal y como se muestra en la figura 2. Y esta perspectiva viene en gran medida de la mano de las capacidades que proporciona la nube.

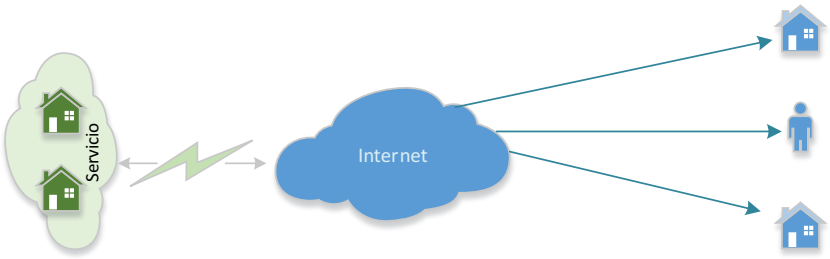

Figura 2 – *Internet 2.0*
La figura muestra la posibilidad de que en el futuro los productores, utilizando las oportunidades que ofrece la tecnología de nube, puedan ser los propios usuarios

Debido a que en gran medida este es un trabajo de investigación para llevarlo a cabo se ha consultado todo tipo de soporte de información, desde libros, productos audiovisuales, artículos o páginas *web*. En un principio se exploró la posibilidad de denominar doméstica a este tipo de nube, entendiendo esta como un sistema implementado dentro del hogar, es decir, con relación directa sobre el individuo. Pero a lo largo de la investigación llevada a cabo para recopilar toda la información necesaria para crear este documento ha surgido la necesidad de modificar el término debido a que el verdadero significado de nube doméstica hace referencia a soluciones de nube corporativas implementadas físicamente en el país de origen de los usuarios. Sólo encontramos algunas contadas referencias que asocien este término a nubes personales situadas en el hogar. Por tanto, y a pesar de existir tímidos intentos de redefinir este vocablo para referirse a esta última acepción, podemos afirmar que en general dentro del ámbito de nube debemos traducir doméstico como nacional y relacionarlas con el entorno corporativo, a pesar de que este término no esté oficialmente aceptado todavía en los estándares. Por ello parecía conveniente encontrar otro término para aludir a las nubes que puedan crear usuarios individuales en sus hogares. De ahí que se decidiera hacer referencia a este tipo de infraestructura como nube personal.

A lo largo del documento se hará una revisión de numerosos conceptos que irán enfocándose poco a poco hacia la nube personal como una evolución de la percepción de Internet muy relacionada con el movimiento *Hágalo usted mismo*[1]. Se comenzará revisando la evolución de los conceptos básicos que han permitido el surgimiento de esta arquitectura, tanto en relación a las telecomunicaciones como a los sistemas, para una vez comprendidos justificar el nacimiento de la nube personal. Posteriormente se realizarán y analizarán algunas pruebas de concepto llevadas a cabo con la intención de identificar hasta que punto estamos ante la presencia de este nuevo paradigma para, a partir de toda la información analizada, llegar a una conclusión. Por tanto se documentarán las pruebas de concepto más importantes que se han llevado a cabo y que permitan justificar las conclusiones. Por su parte las pruebas sobre productos menos decisivas se presentarán como documentos separados.

El documento tendrá cierta validez si, al menos, al finalizar su lectura se llega a una conclusión sobre el futuro del individuo en el mundo de la informática y las telecomunicaciones. Más allá de áridos conceptos y terminología, para el usuario medio el futuro debe acercar al hogar gran parte de las capacidades hasta ahora reservadas a las empresas o los entornos de investigación, facilitando el desarrollo personal en todos los ámbitos y asignándole el protagonismo que se merece y que cada uno quiera ofrecer.

La tecnología ofrece herramientas incontables que podrían ser aprovechadas por mentes inquietas para crear nuevos conceptos, y poco a poco estas herramientas se van haciendo asequibles al usuario individual. Como veremos más adelante una necesidad importante para dar este paso la gratuidad o el bajo precio. Por tanto la pregunta es, ¿se considerará en el futuro que nos encontramos actualmente ante el origen de un Internet 2.0?

[1] DIY de sus siglas en inglés (*Do It Yourself*).

Quién sabe si la evolución de los servicios personales gracias a la nube y la impresora 3D puedan dar un giro radical al desarrollo de la sociedad en una u otra dirección transformando a cada persona de consumidora a productora[2].

Esto será lo que trataré de esclarecer en este proyecto a través de la revisión teórica de las posibilidades actuales, así como los posibles caminos que estas pueden desarrollar en el futuro.

Finalmente partiendo del paradigma de la nube, que está tomando cada más fuerza, se realizará una prueba de concepto para, a partir de los resultados y del propio proceso de implementación, intentar obtener los objetivos deseados. Primero asentar los conceptos de una manera estable para permitir analizar el estado de madurez actual de lo que podría denominarse nube personal, y segundo justificar si realmente es aplicable a día de hoy la idea de un nuevo cambio de paradigma en nuestra relación con Internet.

A continuación se detallan y enmarcan las tareas e hitos del proyecto en el tiempo a través del listado de todas ellas y su encuadre en un diagrama de Gantt:

Name	Begin date	End date
Análisis teórico	3/7/16	4/7/16
Recopilación de fuentes	3/7/16	3/8/16
Recopilación de información	3/9/16	3/15/16
Análisis de la información	3/16/16	4/7/16
Análisis teórico de la situación actual y prueba de concepto - Fase I	4/8/16	4/14/16
Evolución de Internet	4/8/16	4/8/16
Hacia la Nube	4/11/16	4/11/16
Necesidad de la Nube	4/11/16	4/11/16
Tipos de Nube	4/11/16	4/11/16
Inconvenientes	4/11/16	4/11/16
Nube doméstica	4/12/16	4/13/16
Perspectiva de usuario	4/13/16	4/14/16
Análisis práctico	4/15/16	4/22/16
Implementación de Docker	4/15/16	4/18/16
Pruebas	4/18/16	4/22/16
Entregable: PEC 2	4/25/16	4/25/16
Análisis de la prueba de concepto y conclusión - Fase II	4/25/16	5/18/16
Análisis de las distintas opciones	4/25/16	4/29/16
Análisis del sistema Docker	5/2/16	5/9/16
Descripción	5/2/16	5/4/16
Implementación de Dockers	5/2/16	5/6/16
Instalación	5/2/16	5/6/16
Configuración	5/2/16	5/6/16
Ejemplo	5/2/16	5/6/16
Posibilidades	5/6/16	5/9/16
El Futuro	5/10/16	5/12/16
Conclusión - ¿Internet 2.0?	5/13/16	5/16/16
Glosario	5/17/16	5/18/16
Revisión	5/19/16	5/20/16
Entregable: PEC 3	5/23/16	5/23/16
Memoria - Entrega final	5/23/16	6/9/16
Entrega final	6/10/16	6/10/16

[2] Aunque escrito hace 4 años este artículo deja entrever una visión de la tecnología 3D cercana al individuo, más filosófica que compleja: http://tecnologia.elpais.com/tecnologia/2012/02/10/actualidad/1328877228_999295.html

Evolución de Internet

Aunque este trabajo trata de enfocar su visión únicamente en el nivel de los sistemas, entendidos estos como *software*, es importante entender que estos deben integrarse perfectamente con capas inferiores como son las capas que proveen la comunicación entre estos sistemas[3]. Existe una evolución paralela entre ambos mundos que es indisoluble desde el momento en el que las dos primeras máquinas se comunicaron entre sí a través de una red. De hecho no queda siempre claro si los avances en alguna de estas áreas vienen empujados por necesidades en la otra, o simplemente son debidos a la investigación en algún ámbito específico, pero lo que es evidente es que ambas son inseparables y por tanto en muchos casos hablar de uno implica pensar también en el otro.

Para tener una idea general en el siguiente diagrama se muestran algunos de los grandes hitos dentro de los protocolos y arquitecturas de comunicaciones a lo largo de la historia. Estos no son sin duda los únicos avances importantes, podrían incluirse tecnologías como la primera basada en conmutación de etiquetas (ATM) por poner un ejemplo, pero lo único que se pretende es mostrar un esbozo que pueda hacer de guía para dirigir esta sección hacia el surgimiento de Internet y el actual acercamiento de los sistemas de comunicación a la arquitectura de nube.

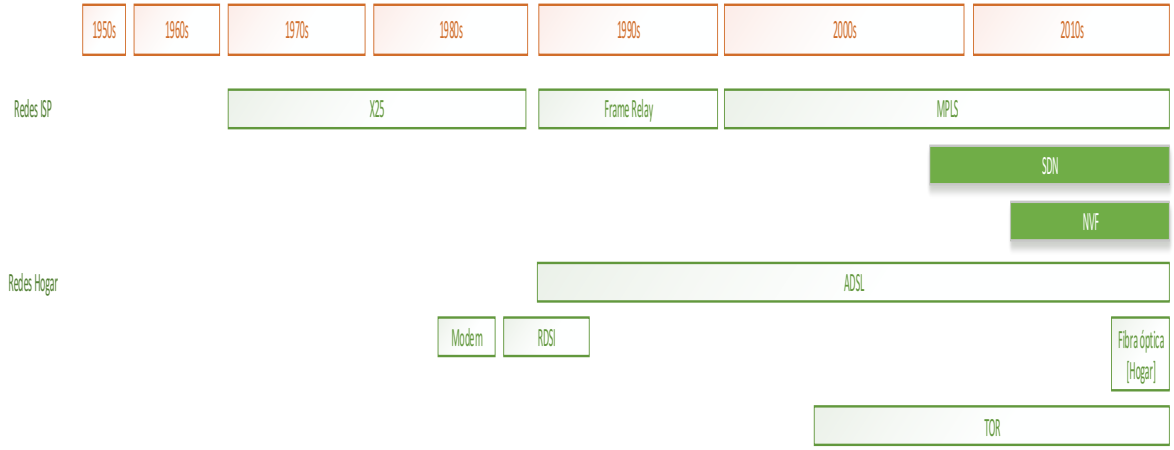

Figura 3 – *Evolución de las redes*[4]
La figura resume la evolución de las tecnologías de las telecomunicaciones a lo largo de la historia

Se puede observar viendo este esquema que la gran revolución en las comunicaciones comienza a partir de la década de los 90s con la aparición de RDSI en el hogar y sobre todo del ADSL. Esto coincide con la aparición de Internet. Es evidente que para poder conectar ordenadores entre sí y obtener algún resultado satisfactorio se requiere de una velocidad de intercambio de datos adecuada. El uso de los modem, que en España era la forma de comunicación típica en los años 90, ofrecía una velocidad de transmisión de datos muy lenta[5]. Nos encontrábamos ante un Internet básico, repleto de páginas *web* estáticas, y básicamente útil para ser explotado por las empresas mediante anuncios y propaganda de un único sentido de manera similar a la televisión. El usuario hacía uso de él para buscar información o para hablar a través de los canales de *chat*, donde la gente se reunía a cierta hora como se podría hacer en un parque. En esta época el uso de Internet en España no era masivo, esto era evidente en los primeros servicios de este tipo como 'La Plaza' de *Teleline* donde no solían reunirse más de 20 personas[6].

[3] Para obtener más información sobre las distintas capas que permiten implementar aplicaciones que funcionan en red se puede consultar el modelo OSI. Estos conceptos son algo que no se va a explicar en este documento como suele hacerse en el 90% de cualquier trabajo sobre informática. Se considera que ya existe infinidad de documentación al respecto. Una fácil referencia puede encontrarse en: https://es.wikipedia.org/wiki/Modelo_OSI

[4] Las fechas son estimativas, entre otras cosas porque el uso de distintas tecnologías en el tiempo de los países.

[5] Aunque en realidad los primeros modem se utilizaban a principios de los 60s su uso cotidiano dentro de los hogares comenzó a usarse a finales de los 80s.

[6] Existían canales de chat internacionales con más usuarios donde cada uno podía crear su propio canal como IRC Hispano.

A finales de este década RDSI constituye el salto del sistema analógico al digital tanto para transportar voz como datos lo que además de mejorar la velocidad de transmisión permite que el usuario pueda hacer uso de la línea telefónica para conectarse a Internet y para hablar por teléfono a la vez, cosa que el modem no admitía, ya que eran acciones excluyentes.

A pesar de que la tecnología ADSL fue desarrollada alrededor de 1989 la realidad es que no comenzó a hacerse uso de ella masivamente hasta bien entrado el año 2000. Esta tecnología sigue siendo la más utilizada hoy en día aunque actualmente está tendiendo a desaparecer al ser sustituida por la fibra óptica. La gran importancia del ADSL estriba en su gran velocidad que permitió el desarrollo de la llamada *Web 2.0*, y hoy en día la expansión de la nube.

Se ha señalado como un hito importante en este diagrama la aparición de TOR. Este es un servicio que permite al usuario visitar Internet sin ser rastreado, o al menos sin que sea fácil hacerlo. Básicamente para poder operar este software de enrutamiento convierte el ordenador personal del usuario en un *router*, o nodo, por el cual pueden pasar otros usuarios. De esta manera para llegar a cualquier lugar de la red el camino a seguir puede atravesar nodos en distintos países a través de otros usuarios lo que hace muy difícil hacer un seguimiento de los usuarios. Se ha incluido como un hito debido a que hoy en día precisamente la seguridad del usuario y más concretamente su privacidad es asunto de debate, como posteriormente veremos en la sección 'Nube personal', y de hecho está siendo uno de los más importantes protagonistas a la hora de dirigir el proceso evolutivo de la propia nube.

Dentro del mundo de la operadora se muestran en cajas verdes dos importantes piedras angulares que participan, desde el mundo de las comunicaciones, en las decisiones y diferentes posibles implementaciones de la nube. SDN y NFV llevan este desarrollo un paso más allá permitiendo una gestión global y mucho más simple del conjunto sistemas-comunicaciones utilizando para ello un nivel de abstracción desconocido hasta la fecha.

Tradicionalmente la gestión de los sistemas de comunicación se hace directamente sobre las máquinas que se dedican a manejar el tráfico de datos según unas reglas más o menos complejas que los administradores configuran directamente en ellas. El cambio por ejemplo de cierto nivel de calidad de servicio para un tráfico determinado de un cliente concreto podía llegar a ser complejo y requerir de cambios a su vez en muchos otros dispositivos de la red. A principios de los 00s se comienza a investigar hacia la separación del control y los datos en las redes de comunicaciones añadiendo un nuevo nivel de abstracción.

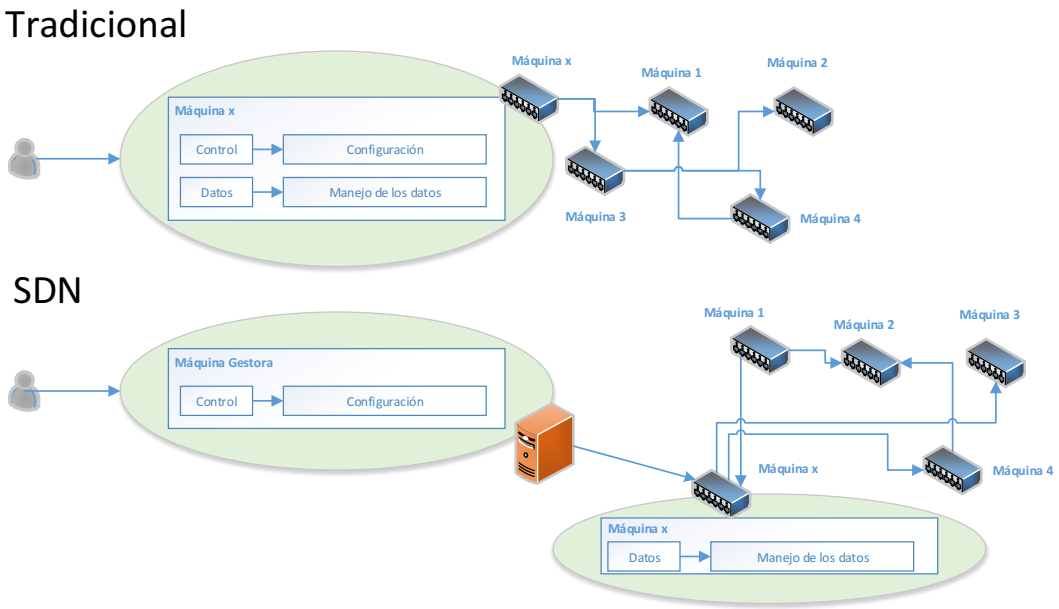

Figura 4 – *SDN*
La figura muestra en consiste el concepto de SDN

Como se muestra en la figura anterior el administrador accede a un gestor de red a través de un interfaz amigable para hacer cambios en la red de cualquier tipo de complejidad. Este gestor traduce los datos a un lenguaje menos abstracto y más cercano a la máquina y envía las órdenes a los dispositivos de red adecuados. Los dispositivos reciben la nueva configuración a través de un cliente que es capaz de entenderla y el cual la traduce a comandos específicos de la máquina. Por tanto el resultado es que la labor del administrador se reduce drásticamente, igual que probablemente el número de administradores necesarios y su conocimiento requerido, de forma similar a como ocurre en la nube a través de los orquestadores como se verá más adelante.

Aunque la idea de separar de alguna forma el control del manejo de datos puro y duro por parte la máquina surge ya a principios del milenio no es hasta finales de la primera década cuando se introduce el concepto de unir este paradigma a la definición de las redes a través de *software* (SDN). *Openflow*, que ve la luz en 2011, es el primer software que permite de realizar esta labor.

Por último el concepto NFV, surgido en 2012, hace referencia a la virtualización de los sistemas de comunicaciones (*cortafuegos, enrutadores, balanceadores, conmutadores*,…) en lugar de usar máquinas dedicadas a ello con un hardware específico para realizar estas labores. En la siguiente sección se dan más detalles de lo que implica el término de virtualización.

Como se puede apreciar y se comentará más adelante la evolución consiste en añadir capas y capas de abstracción intentando separar cada vez más la prestación del servicio del hardware que lo soporta[7].

[7] Para más información sobre SDN y NFV: http://sandbox.ietf.org/doc/draft-gu-sdnrg-problem-statement-of-sdn-nfv-in-dc/?include_text=1

Hacia la nube

Tampoco se va a introducir en este apartado una descripción detallada de la evolución de la informática desde los años 50, ya existe mucha documentación al respecto si se desea tener una visión más profunda. Se pretende centrarse sobre todo en el desarrollo de los ordenadores personales y los servidores hasta la invención de la nube. Como punto clave de este desarrollo encontramos una vez más el concepto de abstracción cuya evolución en el mundo de la informática usaremos como vehículo para analizar el surgimiento de la nube.

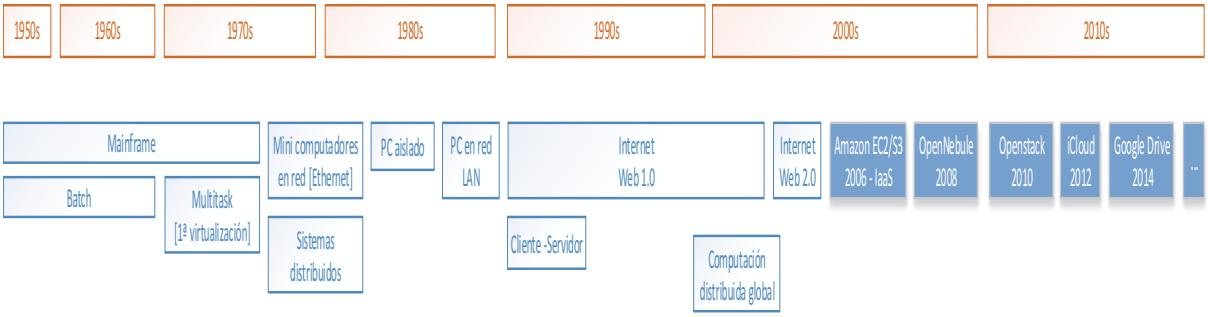

Figura 5 – *Evolución de la computación*[8]
La figura muestra la evolución de las tecnologías de telecomunicaciones a lo largo de la historia

Como vemos a través de la figura en los años 50 se desarrolla un potente tipo de computador corporativo al que se le da el nombre de *mainframe*. Estos eran grandes computadores con mucha potencia que en sus orígenes ocupaban habitaciones y eran especialmente costosos. Los trabajos que realizaban eran secuenciales, es decir, comenzaba una tarea la acababa y realizaba la siguiente, este es el significado de *batch* en la figura. En la década de los 70 las posibilidades de estas máquinas aumentan y admiten la conexión de terminales tontos a través de los cuales trabajan los usuarios compartiendo el tiempo de proceso del computador central. Pero los *mainframes* son muy caros y requieren potencia eléctrica y espacio, así que a fínales de los 70 se da un salto importante en la evolución informática y se desarrollan los minicomputadores, los cuales permiten que más empresas puedan tener acceso a la computación sin tener que gastarse ingentes sumas de dinero o alquilar el uso de *mainframes* a grandes compañías. Básicamente son 'pequeños' sistemas que se conectan en red y comienzan a realizar tareas en conjunto, sistemas distribuidos. Poco a poco se evoluciona en las arquitecturas de comunicaciones y se llega al paradigma de cliente-servidor donde cada computador lleva a cabo un rol, o es un cliente o es un prestador de un servicio o servidor. Dicha evolución llegará en los 90 a los ordenadores personales (PC) y los acontecimientos se acelerarán rápidamente para dar lugar a Internet el cual será el caldo de cultivo para el surgimiento de la computación distribuida. De repente es posible que varios o miles de sistemas trabajen a la vez para realizar una tarea común, es decir, sumen sus recursos para un único fin. Uno de los proyectos más conocidos en relación a este tipo de computación surge en 1999 y es conocido como SETI@Home, el cual a través de un pequeño cliente instalado en los ordenadores de miles de personas con acceso a internet a lo largo y ancho del globo analiza transmisiones de radio recogidas por el telescopio de radiofrecuencia de Arecibo buscando algún tipo de patrón que pueda demostrar la existencia de vida extraterrestre[9].

El gran número de usuarios que acceden a la red va transformando poco a poco el concepto de Internet para, como comenta Terry Flew en su libro *New Media*, pasar de las web personales a los blogs y de publicar a participar, básicamente comienza el mundo de la interacción en Internet. Parece que de alguna manera Internet se le escapa de las manos a las empresas y entidades que lo utilizaban para mostrar únicamente lo que querían enseñar y ahora deben mejorar para evitar que comentarios de usuarios en foros, o a día de hoy en redes sociales y blogs, no afecten a su imagen y por tanto a sus ventas. El usuario gana poder.

[8] A principios de los 70 ya existían redes internacionales interconectadas entre países con objetivos de investigación.

[9] Con SETI@Home (Search for ExtraTerrestrial Intelligence) una parte de la capacidad de proceso de cada computador personal se presta para un fin común, la unión de todos ellos compone un malla o *grid*. http://setiathome.ssl.berkeley.edu/

Finalmente la mejora en las velocidades de transmisión tanto en las redes de los operadores como en el acceso del usuario, así como las cada vez mayores exigencias de ambos y las posibilidades de apertura de nuevos firmamentos tecnológicos que abran nuevas oportunidades de negocio desembocan en un nuevo concepto de arquitectura, la nube.

Existen algunas fuentes que afirman que el origen del concepto de nube data de los años 60 [CSAthtC], pero aunque la palabra se usase por primera vez para definir una tecnología de ese tiempo, en realidad cuando se utilizó por primera vez no se podía imaginar el nivel de evolución que éste llegaría a alcanzar. Por tanto dar por hecho que hace medio siglo unos visionarios ya pensaban en lo que ahora entendemos por nube por la simple razón de que alguien usó esta palabra para referirse a la conexión de varios terminales tontos con un mainframe que tenía capacidades de multitarea parece excesivo. Siguiendo este razonamiento igualmente aplicable sería el término en décadas anteriores para definir las conexiones de llamadas telefónicas con una centralita. Por tanto cuando se haga referencia a la palabra nube en este documento siempre estará asociada a lo que actualmente es una nube, tal y como se define más adelante en este mismo apartado, y que surgió en la década de los 90.

A lo largo de la historia siempre se ha tratado de mejorar el rendimiento en todas las acciones que realizamos a diario y reducir el esfuerzo necesario para llevarlas a cabo. Para ello el ser humano ha ido inventando herramientas cada vez más complejas, desde una punta de sílex afilada para ayudar a cortar carne cómodamente hasta un súper computador para analizar los complejos patrones meteorológicos. Es evidente que el manejo de una piedra para cortar no requiere de gran formación y conocimiento, pero en cambio conocer los entresijos de un supercomputador para que pueda realizar las tareas que lleva cabo parece algo más complejo. Aun así en ambos casos hay una palabra clave que siempre parece estar en el centro de toda mejora, la abstracción.

Sin duda esta es una de las claves de la evolución tecnológica, la cual pasa por observar algo, aislar su funcionamiento y aplicar ese funcionamiento a otras necesidades. En el mundo de la informática es importante además hacer hincapié en la idea de *qué es lo que hace* restando importancia al *cómo se hace*[10]. Esta capacidad queda patente cuando nos referimos a la virtualización como se verá pero no se limita a este nivel. En los sistemas no virtualizados esta abstracción se da entre el hardware y el *software* que es manejado por el usuario, el sistema operativo (SO), y la lleva a cabo el núcleo del mismo o *kernel*. Esta parte del sistema opera en modo privilegiado ya que debe poder acceder a cada parte del *hardware* para gestionarlo y permite que el resto de programas que componen el sistema operativo no tengan que tener conocimiento del *hardware*. Cuando un programador crea una aplicación que debe acceder al *hardware,* éste le es presentado en forma de funciones a las que se les pasan ciertos parámetros, y son estas funciones las que hacen llamadas al sistema y llevan a cabo acciones más complejas y cercanas al lenguaje utilizado por el *hardware*. Aunque cambiemos el *hardware* el *software* no debe realizar cambios ya que para él es transparente. Estos sistemas no virtualizados por tanto se componen de un *hardware*, un sistema operativo con sus ficheros del sistema, y unas aplicaciones que corren sobre el mismo. En la figura 6 se muestran cada una de estas capas superpuestas.

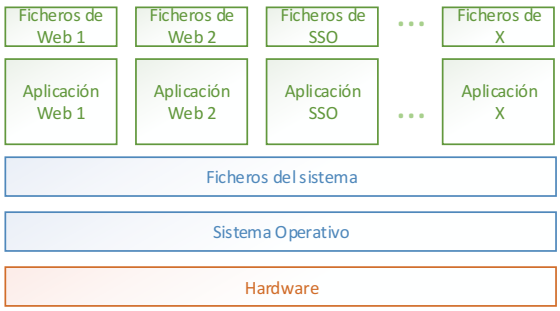

Figura 6 – *Sistema monolítico*
La figura muestra las partes básicas necesarias de las que se compone un computador simple para poder ofrecer un servicio

[10] Fuente: https://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28inform%C3%A1tica%29

Este tipo de sistema es excesivamente cerrado para permitir cambios rápidos, por ejemplo, para levantar el servicio en otra máquina si la primera se estropea. El sistema operativo está específicamente implementado para entender el *hardware* sobre el que viene instalado y si se realizan cambios en éste se necesitan modificaciones en el sistema operativo. Por tanto para llevar a cabo algún cambio se requiere dejar de dar servicio por parte de las aplicaciones que el sistema soporta, realizar el cambio y volver a levantarlo. Esto da como resultado una arquitectura monolítica por lo que en muchos casos se prestan servicios con un *hardware* infrautilizado o por el contrario con recursos insuficientes que derivan en cortes y lentitud.

La solución a esto es la virtualización, lo que implica un nivel más de abstracción en los sistemas informáticos. En la figura 7 se muestra como esta abstracción nos permite que un mismo *hardware* pueda ejecutar más de un SO sobre él, asignarle una cantidad de recursos específica a cada uno de manera dinámica y que todos ellos vean los recursos que tienen garantizados como los únicos disponibles. Por tanto no sólo hacemos una abstracción del *hardware* completo de cara al sistema operativo como hasta ahora sino que dividimos los recursos en grupos con distintas capacidades y mostramos cada conjunto de recursos como un todo de cada sistema operativo. De aquí que al conjunto que compone el sistema operativo más sus aplicaciones y *hardware* asociado se le denomine máquina virtual.

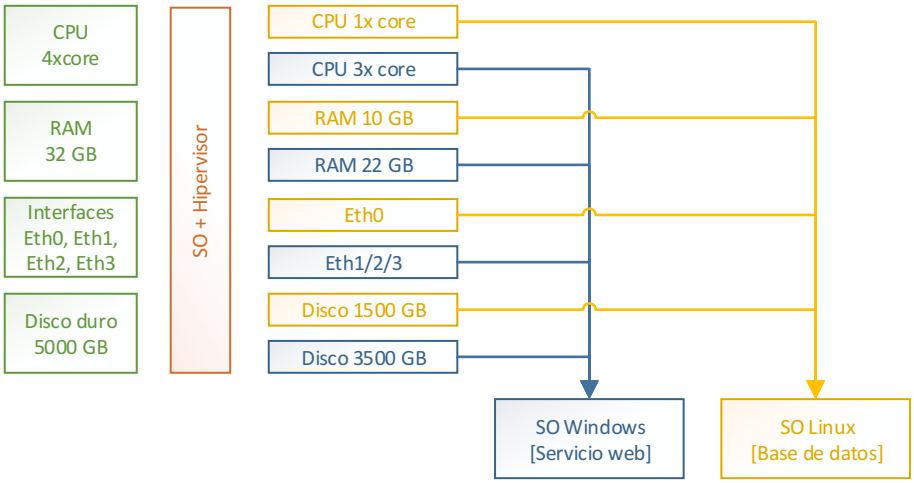

Figura 7 – *Virtualización clásica I*
La figura explica en qué consiste el concepto de virtualización hardware

Por otra parte esta misma capacidad de abstracción del *hardware* nos permite repartirlo por diferentes localizaciones y presentárselo al *software* de manera transparente, ya que éste no se preocupa de los entresijos de las capas inferiores. Por tanto si tengo varios servidores virtuales prestando un servicio en común, estos pueden encontrarse repartidos por varios centros de proceso de datos. Esto nos proporciona como veremos la capacidad de ubicuidad. Si bien es cierto que esto podía realizarse antes de la virtualización, sin ella este proceso es tosco y complejo, y las configuraciones de red requeridas son más exigentes, perdiéndose todas las facilidades que proporcionan los modernos sistemas de virtualización.

Por tanto la virtualización ofrece la gran ventaja de aislar unos sistemas de otros dentro de un mismo *hardware*, lo que proporciona un mayor grado de aprovechamiento del mismo con cierto nivel de seguridad ya que los problemas que puedan afectar a un sistema, como por ejemplo un uso excesivo de alguno de sus recursos asignados, no afecten a los demás. Además como veremos en el siguiente punto, esta es la base de la capacidad *multi tenant* característica de la nube así como uno de los puntos fuertes que otorgan su flexibilidad.

En la siguiente figura se muestran las capas clásicas de un sistema virtual atendiendo al *software* que requieren. Vemos como a partir del mismo *hardware* mostrado en la Figura 3 se incluye una nueva capa llamada *Hypervisor*[11] que es la encargada de ofrecer este nuevo nivel de abstracción del *hardware* de cara a los sistemas que se montan por encima. Estos se encuentran aislados unos de otros o comunicados entre sí a

[11] La aplicación *Hypervisor* o VMM (*Virtual Machine Monitor*) se encarga de gestionar los sistemas operativos virtuales también llamados para abreviar máquinas virtuales (VM por sus siglas en inglés).

través de redes virtuales configuradas a su vez sobre *switches* virtuales. En pocas palabras, desde la capa *Hypervisor* para arriba todo es virtual y es el *Hypervisor* el que se encarga de simular y gestionar todos los recursos *hardware*, lo que permite que las máquinas virtuales funcionen a pleno rendimiento y puedan comunicarse entre ellas y con el exterior.

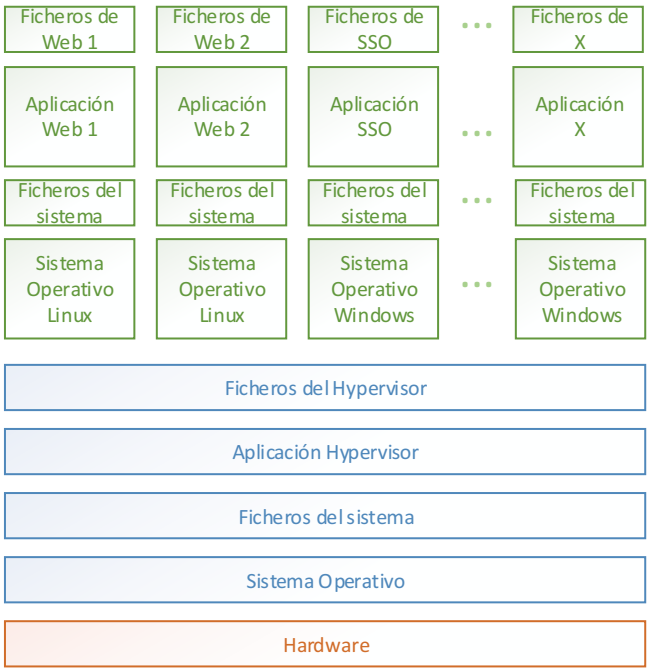

Figura 8 – *Virtualización clásica II*
La figura muestra las partes básicas necesarias de las que se compone un sistema virtual para poder ofrecer un servicio

Llegados a este punto de evolución surge el concepto de nube. A pesar de estar muy generalizada la idea de nube no siempre parece ser perfectamente entendida. Esto quizá es debido en parte a que se trata de un concepto tecnológico que está en boca de todos hoy en día, desde expertos en la materia a completamente legos, por tanto al intentar traducirlo para que sea comprendido también dentro de un mundo ajeno a las telecomunicaciones se pueden perder parte de sus acepciones. Como es lógico las empresas que ofrecen este tipo de servicio lo definen de manera sesgada intencionadamente para que el usuario que lo compra se vea impresionado y evitan ofrecer detalles farragosos. Pero incluso dentro del ámbito informático la idea de nube no está del todo bien asentada. Es habitual que se defina la nube en telecomunicaciones por alguna de sus partes olvidando otras igual de importantes por lo que nos encontramos con que en ciertas empresas, más allá del mundo de los proveedores de servicio, se implementan soluciones parciales que también son consideradas como nubes aunque no sea así, o al menos no de manera rigurosa.

Una nube como tal dentro del mundo de las telecomunicaciones se encuentra intrínsecamente relacionada con un servicio que esa nube proporcione, son indisociables, de ahí que hablemos de servicios en la nube, computación en la nube o computación bajo demanda. No se contrata una nube a un proveedor de nube sino un servicio que será prestado desde una nube, aunque la misma sea privada. Antes esto era evidente, se daba por hecho que la arquitectura de la infraestructura que soportaba el servicio era cosa del proveedor y por tanto el cliente básicamente sólo le daba importancia a los acuerdos de nivel de servicio. Pero hoy en día el propio diseño implementado para proveer los servicios ha cobrado tanta importancia como el servicio en sí mismo por lo que es primordial situar cada concepto en su lugar.

Precisamente la finalidad de utilizar la palabra nube pretende expresar cierto grado de desconocimiento sobre la infraestructura que la sostiene[12], se pueden exigir ciertos requerimientos al proveedor de servicios en

[12] La teoría más extendida sobre el origen del término hace referencia a los diagramas de red donde se dibujaba una nube para representar primero redes telefónicas y luego de datos de las cuales los clientes no tienen información sobre su arquitectura, ni les interesa.

relación a su implementación como por ejemplo que sea privada o que su infraestructura se encuentre físicamente en el país de la empresa que la contrata[13], este último punto cada vez más demandado, pero no se le exigirá nunca al proveedor un tipo de hardware, virtualización o localización específicos, ya que esto será decisión del propio proveedor que sólo se preocupará de atender a las demandas del cliente dentro de lo establecido. En primera instancia se define un servicio, más adelante detallaremos en qué puede consistir este servicio, y posteriormente se provee dicho servicio sobre una plataforma 'cambiante' tanto en su localización como en sus recursos. Estos cambios son transparentes para el usuario que únicamente se preocupa de contratar una cantidad específica de recursos.

La nube por definición debe permitir el acceso a un conjunto de recursos que son provisionados de manera rápida, dinámica y con cierto grado de simplicidad, para lo cual se utilizan sistemas virtualizados que proporcionan el alto nivel de dinamismo necesario para permitir escalar los recursos bajo demanda, de manera transparente y automática para el usuario. La nube se ofrece al cliente como un servicio en el cual se paga únicamente por lo que se consume.

Además de esto es muy importante entender que los servicios provistos en la nube se caracterizan por su ubicuidad en cuanto al acceso y al lugar donde se encuentran físicamente las máquinas que proporcionan el servicio. Este último aspecto es clave, ya que implica que la infraestructura puede y debe encontrarse replicada en distintos lugares y por tanto el usuario no tiene conciencia de a qué localización está accediendo realmente. La situación física de las infraestructuras a las que el usuario accede puede depender de varios factores como por ejemplo la localización física del propio usuario o la disponibilidad o no de un emplazamiento concreto en un momento determinado.

De acuerdo con la última definición disponible en NIST, año 2011[14], la computación en la nube debe tener las siguientes características básicas para considerarse como tal:

- Servicio bajo demanda.
- Gran ancho de banda desde múltiples tipos de plataformas cliente.
- Recursos comunes a distintos clientes[15], tanto físicos como virtuales, que son asignados bajo demanda de manera dinámica y cuya localización es desconocida por el cliente.
- Rápida elasticidad para provisionar y liberar recursos.
- Control y optimización de recursos por el proveedor y el usuario.

A día de hoy quizá esta enumeración de características que deben cumplirse para considerar la existencia de una nube parece quedarse algo corta, o simplemente quizá resulta algo inexacta atendiendo al nivel de desarrollo actual de ciertas tecnologías como por ejemplo la relacionada con la virtualización. Por ello y para intentar ofrecer una definición más clara y rigurosa se intenta redefinir este grupo de propiedades esenciales.

- Gran ancho de banda: Lo que permite que casi cualquier tipo de plataforma que tenga acceso a Internet pueda hacer uso de los servicios de la nube, así como la deslocalización de los recursos.

- Virtualización: Gran parte de los sistemas que soportan los recursos en la nube se encuentran virtualizados ya que esto permite mayor dinamismo, simplicidad y eficiencia a la hora de realizar provisiones. A día de hoy existen numerosas empresas dedicadas exclusivamente al mundo de la virtualización, y además de éstas, existen opciones de software libre que proporcionan esta capacidad.

  Hay que recordar que cuando hablamos de virtualización en este contexto nos referimos a virtualización *hardware*, es decir, se lleva a cabo una abstracción del *hardware* para permitir presentárselo al *software* que se ejecuta por encima de tal manera que desde su punto de vista está

[13] Este punto será tratado más detenidamente en la sección "Nube doméstica".

[14] National Institute of Standards and Technology: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
Otros autores también se decantan por esta definición. [tCaYS]

[15] Multi-tenant en inglés.

utilizando todo el disponible en la máquina (memoria, CPU, disco, etc.). En este sentido cuando hablamos de *software* nos referimos a un sistema operativo[16] que correrá ciertas aplicaciones.

- Orquestación: Es evidente que toda esta complejidad en las capas inferiores al *hypervisor* proporciona grandes ventajas pero exige un cierto nivel de abstracción (una vez más) de cara al usuario para que éste pueda gestionar y monitorizar de manera cómoda, rápida y dinámica los recursos, ya sean *software*, de infraestructura, o de almacenamiento. Para ello se crean unas herramientas de administración que permiten modificar la provisión de cualquier tipo de servicio que ofrezca la nube. Estás herramientas se encargarán de suavizar las complejidades de ciertos procesos de cara al usuario.

  En muchos casos muchos de los productos actuales que se integran con la nube utilizan un interfaz de aplicación común para que los propios usuarios puedan automatizar tareas e incluso integrarlos con sus propios orquestadores[17]. Para ello sólo necesitan conocer qué elementos se pueden administrar, cual es la estructura que organiza estos elementos para saber dónde encontrarlos (al igual que con una estructura de directorios y ficheros) y el formato utilizado para realizar las consultas o modificaciones.

  Gracias al orquestador no sólo el proveedor del servicio sino el propio cliente del mismo puede hasta cierto punto gestionar los recursos sin tener que conocer las complejidades de los sistemas que corren por debajo.

- Ubicuidad: Las propias mejoras del ancho de banda en las redes de comunicaciones así como la virtualización permiten que los recursos que soportan los servicios puedan encontrarse en diferentes localizaciones que replican sus datos de manera síncrona[18], y además puedan ser gestionados de modo sencillo como si se encontraran en una misma localización, gracias a la virtualización. Esto como contrapartida además añade una importante característica inherente a este tipo de sistemas, la alta disponibilidad, ya que se puede tener el *hardware* repartido por varias localizaciones y por tanto tener el *software* funcionando sobre él de manera continua aunque una de ellas se caiga.

  Esta ubicuidad, que se da también en el origen del tráfico, es decir en el usuario, provoca que el cliente de la nube no sepa a ciencia cierta donde se encuentran exactamente los recursos. De hecho en muchos casos se da la situación en la cual el usuario contrata acceso a un *software* en la nube[19] a una empresa que a su vez contrata un servicio sobre el que montar esa aplicación a un proveedor por lo que la cadena se puede alargar y hacer todavía más difícil localizar el alojamiento final del servicio.

- Flexibilidad: Un punto muy importante a tener en cuenta es la capacidad para realizar cambios de manera ágil en la provisión de los servicios. De nada nos sirve tener sistemas repartidos entre diferentes localizaciones con un gran ancho de banda si la arquitectura es inamovible y la modificación de una provisión se convierte en un trabajo complejo y lento.

- Paga según consumes: Es muy importante tener en cuenta que en este tipo de servicios la granularidad en la relación precio-consumo es mucho mayor que en sistemas anteriores. Debido a que podemos gestionar los recursos asignados de manera dinámica según las necesidades de cada momento, el cliente puede modificar sus necesidades y por tanto pagar más o menos según estas exigencias. No es necesario pagar por un servidor completo por ejemplo sino por una cantidad 'x' de memoria con 'y' capacidad de proceso. En resumen el usuario únicamente paga por lo que consume, y esto conlleva un ahorro evidente.

Algunos autores incluyen entre estas características otras dos como son el almacenamiento específico de la nube [DaMCI] y la capacidad de *multi-tenancy*. En este trabajo ambas se descartan como características básicas

[16] Dentro del apartado nube personal veremos que este modelo también está cambiando actualmente.

[17] El lenguaje más comúnmente extendido para estas API por sus siglas en inglés (Application Programming Interface) es fullREST.

[18] Dependiendo del tipo de dato no siempre es posible una replicación síncrona. En este sentido nos encontramos con dos tipos de tráfico, elástico e inelástico. [FoMN2016]

[19] Se repasarán las distintas ofertas de servicios en la nube en la sección 'Tipos de nube'.

de una nube debido a que existen servicios en la nube que no requieren de ninguna de ellas y no por ello el servicio deja de ser prestado desde una arquitectura de nube. En cualquier caso debido a la importancia de ambos conceptos quedan explicados a continuación.

- *Multi-tenancy*: Como se ha avanzado anteriormente esta característica se define como la capacidad que permite que diferentes clientes puedan utilizar los mismos recursos sin que ello signifique que unos puedan acceder a los recursos asignados a otros. La definición de esta capacidad queda bastante patente cuando nos referimos a los proveedores de servicios de Internet de toda la vida (ISP). Cuando varios clientes contratan por ejemplo una MacroLAN para uso empresarial con acceso a Internet, el tráfico se transmite a través de decenas de máquinas que no son de uso exclusivo de un único cliente sino que se comparten, pero no por ello los clientes tienen acceso a las redes de otros clientes. La finalidad de este ejemplo no es afirmar que en el caso de las redes hablemos también de *multi-tenancy* ya que de hecho este concepto surge a partir de la nube, simplemente se añade para clarificarlo.

  Pero la realidad es que existen nubes privadas contratadas por clientes que exigen que la infraestructura sea de uso exclusivo para el cliente que la contrata.

- Almacenamiento: Evidentemente el almacenamiento es necesario para prestar servicios por lo que en este sentido la necesidad del mismo parece más bien una obviedad. En cuanto al tipo de almacenamiento, hasta el momento el paradigma es la base de datos relacional donde se guardan los datos en tablas y las tablas se relacionan entre sí por claves. Esto permite que la información asociada a un dato no se repita y podamos acceder a ella a través de consultas. Por ejemplo tenemos nombres de personas en una tabla, países en otra y razas en otra, todas ellas con un identificador único. En una tabla distinta tenemos a personas con su información relativa al nombre, país de origen y raza haciendo referencia a estas otras tablas donde se guardan estos datos. Por tanto obtenemos una estructura jerárquica de tablas. En este caso la información nunca se repite, por ejemplo en la tabla de países sólo encontramos los que existen una única vez y nada más.

  Pero cuando los datos son masivos y se accede continuamente a ellos es más eficiente buscar en una estructura plana en lugar de realizando consultas que pueden llegar a ser complejas, y para ello usamos bases de datos no relacionales donde todos los datos están al mismo nivel, no hay jerarquía (*flat object storage*). Los datos se ordenan y se consultan a través de una clave y esta clave tiene unos atributos asociados. Estos atributos pueden estar repetidos ya que podemos encontrar por ejemplo los atributos 'Javier' 'España' 'Caucásico' repetido varias veces aunque con distinto identificador. El beneficio que aporta este tipo de base de datos es mayor que el perjuicio ya que aunque ocupan más espacio hoy en día el almacenamiento es muy barato. Este tipo de almacenamiento es típico en la nube, aunque no se limita sólo a este. Los proveedores de nube también proporcionan almacenamiento a través de bases de datos relacionales, por ello no se considera indispensable esta característica en este documento como para estimarla ineludible dentro de una arquitectura que siga el paradigma de la nube[20].

De la lista de características básicas se derivan todas las bondades de la nube, es evidente que no todo es perfecto y por ello nuevos conceptos surgen cada poco tiempo que tienden a mejorar los aspectos más delicados como pueden ser la seguridad, pero a grandes rasgos en este momento se disfruta de un alto grado de abstracción. Este trabajo de hecho trata de ahondar en una de estas evoluciones aunque aplicada exclusivamente al entorno de las nubes personales.

Tras un rápido escrutinio de todas las características lo primero que resulta evidente es que las tres primeras parecen ser la médula sobre la que parten el resto de las posibilidades que ofrece la nube, la virtualización, la orquestación y un gran ancho de banda para compartir la información de control de los sistemas de manera rápida entre las distintas localizaciones. Aparentemente el resto de capacidades se derivan de estas dos pero debido a que las más importantes no aseguran la existencia del resto es necesario enumerarlas todas como imprescindibles dentro de este tipo de servicio.

[20] En cuanto al tipo de almacenamiento siguen usándose DAS, NAS, SAN.

La nube ha surgido relativamente hace poco tiempo y está aquí no únicamente para quedarse sino para evolucionar rápidamente. Ha destrozado literalmente las concepciones de los centros de proceso de datos estancos convirtiéndolos en sistemas a desmantelar. Esto no significa que no existan a día de hoy implementaciones más arcaicas y que no se les saque partido ya que para cada necesidad existe siempre una distinta solución, pero la realidad es que de una forma u otra irán caducando y renovándose hacia esta nueva visión de la misma forma que en el ámbito de las comunicaciones se avanzó desde el *frame relay* a MPLS y en futuro cercano se implementa sobre estos la abstracción SDN.

En el siguiente diagrama se muestra grosso modo la evolución de los centros de proceso de datos como proveedores de algún tipo de servicio a los usuarios, hasta llegar al concepto de nube.

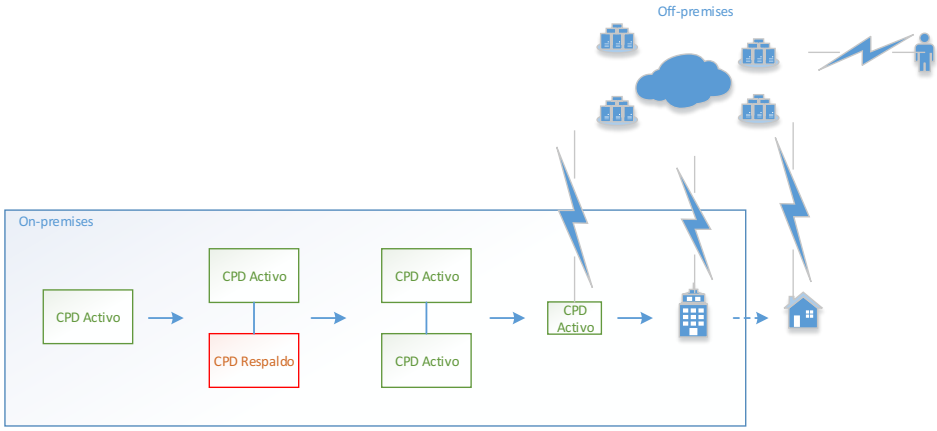

Figura 9 – *Evolución del CPD*
La figura muestra la evolución de los CPD empresariales desde su origen hasta su posible desaparición futura

Podemos observar como inicialmente para que una empresa, entendiendo como empresa una entidad pública o privada, pudiese satisfacer las necesidades inherentes al servicio que prestaba debía implementar su propio centro de proceso de datos, o CPD, donde se montaba la arquitectura *hardware* y *software* que le permitiese llevar a cabo esa labor. Estos CPD podían componerse de algunos servidores y sistemas de comunicaciones básicos guardados en un pequeño armario, u organizarse en filas de *racks*[21] repletos de equipamiento informático y de comunicaciones con una arquitectura avanzada de refrigeración de pasillos fríos y calientes y complejos y caros sistemas de aire acondicionado.

Posteriormente según el uso de Internet se hacía cada vez más masivo e indispensable se comienza a pensar en la necesidad de mantener los servicios activos de manera continua y por tanto tener preparados planes de contingencia ante posibles desastres. A partir de este momento las empresas que pueden permitírselo crean un segundo centro de proceso de datos como respaldo del primero ante la posibilidad de caída del activo. Aunque en muchos casos puede llevar tanto tiempo levantar el centro de respaldo, con todos los datos sincronizados y la misma calidad de servicio tras una caída del centro principal, que para entonces el centro primario ya se encuentra disponible.

Gracias a las nuevas capacidades en los protocolos de comunicaciones como GSLB[22] o Anycast[23], así como a la mejora de la velocidad de las líneas las empresas comienzan a optar por tener ambos centros de proceso dando servicio a la vez. Esto permite asegurar que ante la caída de uno el otro estará disponible automáticamente y lo que es casi igual de importante, ambos centros de proceso de datos se están rentabilizando de manera continua.

---

[21] Soporte metálico destinado a alojar equipamiento electrónico, informático y de comunicaciones. Las medidas para la anchura están normalizadas para que sean compatibles con equipamiento de distintos fabricantes. También son llamados bastidores, cabinas, *cabinetes* o armarios. Fuente: https://es.wikipedia.org/wiki/Rack

[22] *Global Server Load Balancing* permite que las peticiones DNS que realizan los usuarios para acceder a un servicio concreto sean balanceadas a uno u otro CPD atendiendo a distintos valores como la carga del servicio en un centro específico.

[23] *Anycast* es una metodología de direccionamiento que permite que una misma dirección IP pueda existir en diferentes localizaciones a la vez, lo cual utilizado junto con un protocolo de enrutamiento dinámico como BGP permite que las peticiones de acceso a un servicio por parte de los usuarios sean traducidas por el DNS a una IP y entonces enviadas al servicio que se encuentre más cerca de ellos.

Finalmente la nube entra en escena y las empresas comienzan a darse cuenta de que puede resultar más rentable en muchos sentidos dejar que un tercero se encargue de alojar todas las máquinas, el almacenamiento o el *software* necesarios para prestar un servicio antes que pagarlo y gestionarlo uno mismo. Por tanto éstas comienzan a reducir sus centros de proceso de datos hasta una mínima expresión y alojar sus grandes servicios en la nube. A partir de aquí ya sólo quedan aparentemente dos pasos más, la eliminación completa del CPD dentro del edificio de la empresa, y como último e hipotético caso, la desaparición del propio edificio de la empresa.

¿Por qué se va a mantener el gasto de un edificio si la velocidad de las comunicaciones permite que los empleados puedan trabajar desde casa? Efectivamente hoy en día gracias a los grandes anchos de banda que conectan a los usuarios a Internet se puede desempeñar cualquier tipo de trabajo, reuniones telefónicas, videoconferencias, acceso a servicios, etc. desde casi cualquier localización. Si además añadimos a esto que básicamente cualquier servicio puede alojarse en la nube, lo que implica la desaparición del CPD corporativo, ¿qué razón tendrán muchas empresas en el futuro para mantener el alquiler o la compra de un edificio?

Vemos por tanto que tal y como se comentaba a la hora de enumerar las características básicas que definen una nube, no por cumplir únicamente alguno de sus rasgos distintivos nos encontramos ante una arquitectura de nube. Por ejemplo dos centros de proceso de datos funcionando activamente a la vez no componen una nube. En su expresión básica no disfrutan de ubicuidad en un sentido amplio, ni se asegura un cambio de provisión automático, etc.

En la figura se muestran en conjunto las evoluciones tecnológicas que han permitido desarrollar este nuevo concepto.

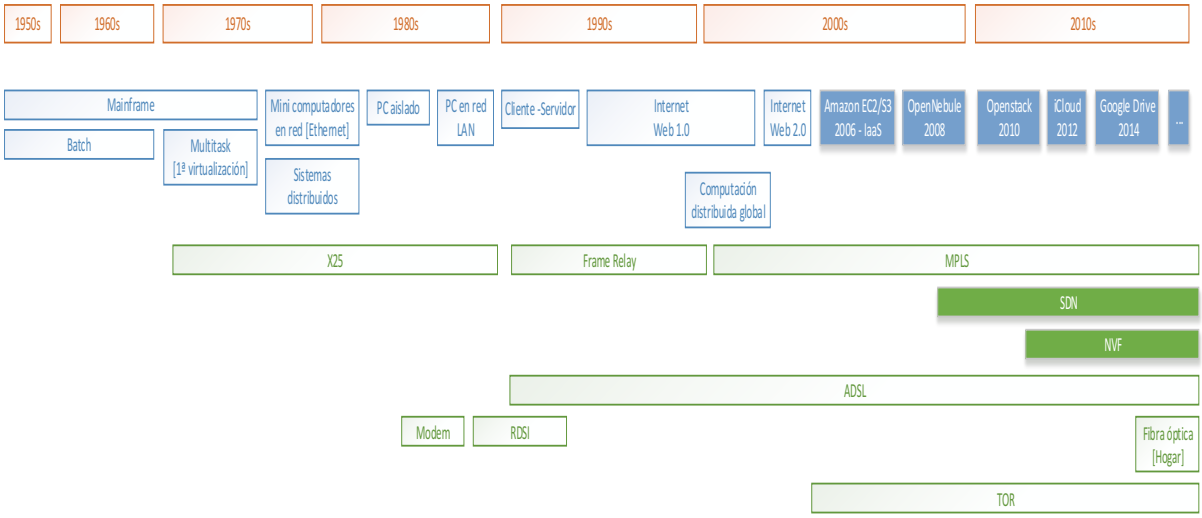

Figura 10 – Evolución de las comunicaciones y la computación
La figura muestra la evolución de las tecnologías de computación y comunicaciones a lo largo de la historia

Necesidad de la nube

La nube surge por necesidad como casi todas las evoluciones que surgen y permanecen. Como se ha comentado en el apartado anterior existen unas características que la definen y recíprocamente estas descubren a su vez la razón por la cual se llega a ellas. A continuación se van a enumerar varias de las razones, muy probablemente no todas, que han empujado y forzado esta evolución.

Según evoluciona el concepto del CPD se buscan posibles mejoras en estos entornos, existen modelos de centro de proceso de datos que incluso embeben todos sus componentes dentro de un *container* para poder transportarlos fácilmente si es demandado por alguna necesidad. Se diseñan arquitecturas específicas para

aprovechar el frío generando corrientes de aire, se mejoran los sistemas de conexión para acelerar el tiempo de activación, entra en el juego el CPD verde que limita las emisiones contaminantes, etc. Todo esto en el fondo no hace más que encarecer los precios a la hora de implementar un CPD y por otra parte requiere de personal más especializado para mantenerlos. En resumen las empresas sufren este aumento en el coste y el gasto añadido de requerir personal más especializado.

La necesidad de *multi-tenancy* en conjunción con la virtualización permite que los proveedores puedan utilizar los mismos dispositivos para atender la necesidad de varios clientes. Esto lo que significa es que los proveedores pueden comprar recursos en grandes cantidades, lo cual siempre resulta más barato que comprar lo necesario para un solo cliente. Gracias a la virtualización ningún recurso resulta en desuso y todos se pueden aprovechar repartiéndolo entre distintos clientes y servicios. De esta forma a las empresas les resulta más fácil y barato delegar la creación y gestión de sus sistemas a un tercero debido al ahorro relacionado con el gasto en *hardware*, en electricidad para mantener el CPD, personal especializado que administre los sistemas, personal especializado en seguridad y contratos de soporte con los proveedores *software* y *hardware*. Hay que tener en cuenta que delegar la gestión implica la eliminación de muchas preocupaciones ya que existen muchas capas que se dejan de atender, desde la capa física, de virtualización, servicio, recursos, acceso, entrega, control, usuario, etc.
Además estas dos capacidades unidas permiten a las empresas realizar pruebas reales en entornos de test de manera más barata, rápida y realista. Anteriormente esto se conseguía contratando una empresa especializada de consultoría para que realizara pruebas de carga específicamente definidas para nuestros servicios, o teniendo un equipo especializado en nómina y comprando licencias de *software* específico para llevar a cabo este tipo de labores. Actualmente podemos montar los servicios de test y acceder a ellos con software gratuito de nube desde distintas ubicaciones o desde la misma simulando grandes cargas y localizaciones. O también contratando servicios *SaaS* específicos para esta labor.
Esto en cuanto a la parte de simulación del cliente, por parte del servidor se puede modificar dinámicamente la cantidad de recursos asignados al servicio en nuestra propia nube contratada aprovechando su flexibilidad en cuanto a la provisión, lo que nos permite realizar estimaciones rápidas sobre nuestras necesidades.

La virtualización añade una capacidad muy interesante como es la portabilidad. Podemos transportar nuestro servicio de un lugar a otro con mayor facilidad.

La deslocalización de los centros de datos en el proveedor de servicios nos añade de manera nativa una capacidad muy importante, la alta disponibilidad. Debido a la exigencia de tener los datos replicados la empresa no sólo se ahorra el coste de un CPD sino de dos.

La propia acción de delegar la gestión de los servicios a un tercero, aunque no es exclusiva de los proveedores de nube, evita incorrectos dimensionamientos a la hora de provisionar servicios, o si esto sucede la ocurrencia por parte del proveedor en una penalización gracias a los contratos de nivel de servicio. Dentro de este ámbito se incluye no sólo el dimensionamiento en cuanto a la capacidad de los servidores sino también en relación con la potencia eléctrica, cableado,…

Gracias a la característica de pagar por lo consumido se consigue un ahorro por parte del cliente que no tiene que comprar más *hardware* del necesario por si la aplicación crece con lo cual ahorra dinero. Además como el proveedor sólo cobra por lo que consume se preocupará de dimensionar correctamente los recursos asignados y evitar el sobredimensionamiento, lo que indirectamente favorece el concepto de CPD verde.

Por último si nos fiamos de los departamentos de ventas de los proveedores podríamos incluso añadir mayor seguridad ya que esta se gestiona mejor debido a que el proveedor está preparado para ello. No sólo a nivel *software* sino de la propia infraestructura (control de acceso físico, alarmas, incendios,...)

Para acabar es interesante comprobar el éxito real de la nube. En la siguiente imagen podemos ver la evolución del consumo de ancho de banda por parte de los usuarios de los servicios de comercio electrónico de *Amazon* una vez que la empresa pasó de utilizar una inmensa y estática infraestructura de IT a una arquitectura de nube. En menos de 18 meses se puede comprobar el crecimiento exponencial gracias al uso de esta tecnología.

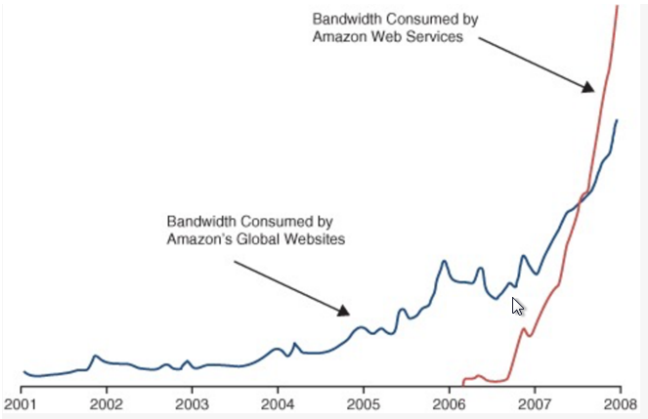

Figura 11 – Consumo de ancho de banda por tecnología. *Fuente: [tCaYS]*
La figura muestra la evolución del consumo de ancho de banda de los usuarios utilizando arquitecturas anteriores a la nube (en azul) y arquitecturas de nube (en rojo)

En resumen el negocio promueve el avance en la sociedad humana así como la necesidad es el acicate de la evolución en otros entornos.

Tipos de nube

Debido a la ingente documentación relativa a este aspecto de la nube simplemente vamos a enumerar y resumir brevemente los tipos de nube y servicios que estas prestan.

Las nubes se clasifican en cuatro tipos, aunque la documentación oficial sólo admite tres, más otros dos que están en ciernes. Se podría decir que los cuatro primeros vienen definidos por el lugar donde su infraestructura está implementada:

- Nube pública: Situada en las instalaciones del proveedor dan servicio a varios clientes a la vez, es decir, comparten la misma infraestructura entre todos ellos.
- Nube privada: Situada en las instalaciones del proveedor dan servicio a un sólo cliente por lo que ofrece mayor seguridad.
  No se debe confundir este tipo de nube con aquella cuya infraestructura reside en las instalaciones de la propia empresa o en el proveedor. Desde esta perspectiva hablamos de:

  On-premise: Residente en la propia empresa.
  Off-premise: Residente en las instalaciones del proveedor.

- Nube hibrida: Mezcla de las dos anteriores donde ciertos servicios que requieran una mayor seguridad y control se prestan en una nube privada y dedicada y el resto en la pública.
- Nube doméstica: Nube privada cuya infraestructura se encuentra situada además en el país del cliente que la contrata.

Por su parte los dos nuevos tipos que están en desarrollo y por tanto cuyos estándares están en fase de definición son[24]:

- *Inter-cloud*: Según se desarrollan y evolución las arquitecturas de nube se va necesitando cada vez más que las nubes de distintos proveedores se comuniquen entre sí y para ello se necesita una puerta

[24] Prefiero incluir los nombres en inglés ya que suenan más descriptivos. Para más información se puede consultar la IETF (The Internet Engineering Task Force (IETF), organización de estándares abiertos que se encarga de desarrollar estándares de Internet): http://sandbox.ietf.org/doc/draft-aazam-cdni-inter-cloud-architecture/?include_text=1

de enlace entre dos nubes. Este nuevo escenario se conoce con el nombre de computación inter-*cloud* de nubes federadas.

- *Media-cloud*: Teniendo en cuenta que el 62% del tráfico de Internet es generado por el video y se espera que llegue en un futuro hasta el 90%, se comienza a definir un estándar para nubes específicamente diseñadas para manejar esta situación.

Una vez se tiene claro qué tipo de nube se requiere se definen tipos de servicio que se necesita obtener de ella. Como ya hemos visto las características necesarias para que una infraestructura se considere una nube están claras por lo tanto lo que interesa saber ahora es qué ofrece dicha infraestructura.

Para definir el tipo de servicio se utiliza una nomenclatura específica para cualquier servicio que se proporcione, esta consiste en el prefijado del servicio seguido de 'aaS' (del inglés *as a service*). Si buscamos documentación encontramos por regla general únicamente tres tipos de servicio, a saber, *software*, plataforma, e infraestructura o hardware (*SaaS, PaaS, IaaS/HaaS*) [tCaYS]. La que ofrece cada servicio es lo siguiente:

- Software: El proveedor proporciona una aplicación empaquetada y el usuario lo único que puede y necesita hacer es configurarla, de manera similar a como lo haría tradicionalmente si la compra y la instala en sus servidores. Ejemplo *salesforce.com*
- Plataforma: El proveedor proporciona un *pack* compuesto por un sistema operativo, un servidor de algún tipo, típicamente un servidor web, que utiliza una serie de recursos que se escalan automáticamente según se necesita y finalmente un entorno de programación para trabajar sobre ese servidor. Básicamente proporciona todas las herramientas que permitir la creación de una aplicación. Ejemplo *Google AppEngine (GAE) o MS Azure*.
- Infraestructura: El proveedor suministra una máquina virtual con un sistema operativo específico y unos recursos asociados y el usuario monta su aplicación sobre ese sistema. Ejemplo *Amazon EC y S3*.

A día de hoy cada vez se definen más servicios para ser prestados en la nube. Debido a que algunos de ellos ya no entran dentro del paradigma típico de la aplicación como tal, ni de ninguna de las otras definiciones, sino que son nuevos servicios en sí mismos, se van añadiendo nuevos elementos a esta lista como por ejemplo *FWaaS* (servicio de cortafuegos), *LBaaS* (servicio de balanceo de carga), *SECaaS* (seguridad como servicio), *WANaaS* (WAN como servicio), *NaaS* (Red como servicio) y tantos otros ya tengan razón de ser o simplemente por puro marketing. En realidad muchos de ellos podrían recaer en *SaaS* pero cada vez más se tienden a excluir de este conjunto. A continuación se puede ver un esquema que intenta explicar desenfadadamente lo que implica cada tipo de servicio de nube a los neófitos en este tema.

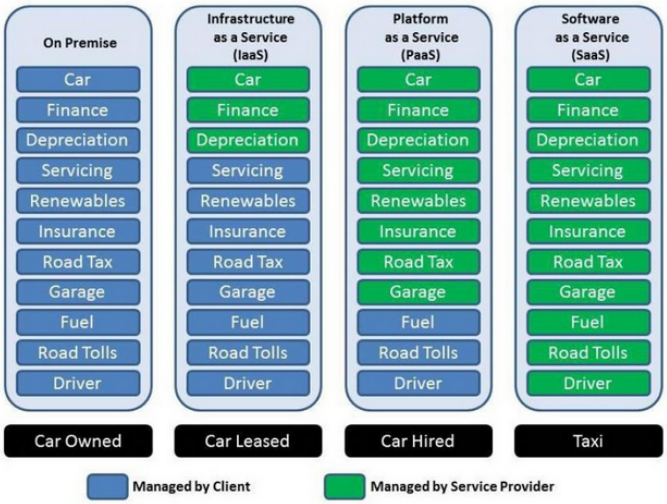

Figura 12[25] – *Tipos de servicios de nube.*

[25] Diagrama original por *Cloud53*, página oficial: https://www.cloud53.co.uk/aas/

La figura intenta mostrar desde otra perspectiva lo que implica cada uno de los más conocidos servicios de nube.

Para finalizar se expone un esquema donde se hace un resumen de cada uno de los tipos de servicio que proporciona la nube y se enumeran las empresas, o los nombres de sus productos, que prestan estos servicios. La realidad es que desde el 2008 hasta la fecha se ha evolucionado mucho y existen muchos nuevos productos, pero se ha decidido incluirla aquí por la claridad que proporciona:

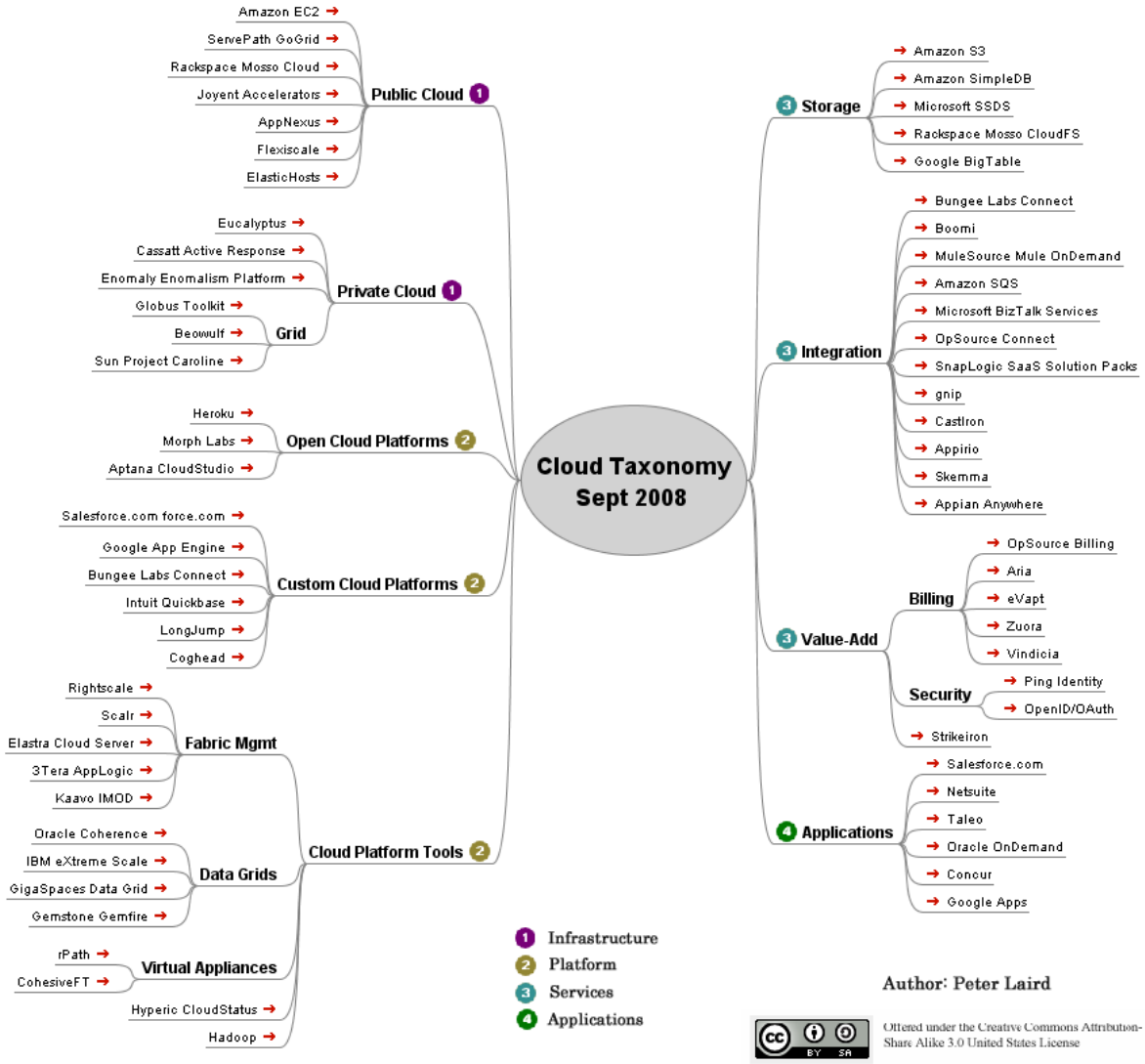

Figura 13 – *Taxonomía de la nube*[26]

La figura muestra distintas servicios ofrecidos por proveedores de nube agrupados por la tipología de los mismos

Inconvenientes

Evidentemente los inconvenientes de la nube están enfocados al cliente que la contrata, no al proveedor, y a grandes rasgos todos confluyen en el mismo punto, la seguridad en todas sus vertientes, desde la necesidad de tener los datos en infraestructuras de tercero gestionadas por estos al desconocimiento total de donde están en cada momento estos datos. De acuerdo a una lista publicada por Gartner estos inconvenientes se resumen en siete, todos ellos originados por la falta de control en parte del cliente en algún aspecto de la seguridad[27]:

[26] Fuente: https://sites.google.com/site/saaslink/Laird_CloudMap_Sept2008.png

[27] Fuente: http://www.infoworld.com/article/2652198/security/gartner--seven-cloud-computing-security-risks.html

- Control de acceso a los datos tanto físico como lógico.
- Control del cumplimiento de las regulaciones de seguridad.
- Conocimiento del lugar físico donde se encuentran los datos.
- Transporte seguro de los datos entre las diferentes localizaciones.
- Control de lo que sucederá a los datos ante un desastre.
- Dificultad para exigir una correcta investigación ante incidentes de seguridad, y probablemente falta de posible información cuando estos ocurran.
- Control de lo que pueda pasar si el proveedor es comprado, o qué pasará con los datos si quiebra.

Resumiendo, la centralización de las aplicaciones y el almacenamiento de los datos genera una interdependencia de los proveedores de servicios. Obviando la confianza que se debe depositar en ellos en todos los sentidos anteriormente enumerados, más algún otro que se comenta en la siguiente sección, el cliente además debe confiar en que la seguridad que proporciona la nube para sus servicios es la adecuada, es decir, que tienen grandes y buenos equipos de expertos en la materia.

Desde cierto punto de vista podría pensarse que la nube sigue la metáfora de poner todos los huevos en la misma cesta, ya que encontramos muchas grandes empresas utilizando estos servicios, si por alguna razón falla la seguridad en la infraestructura de ese proveedor en concreto, un atacante no sólo tendrá acceso a los datos de una empresa sino posiblemente de muchas de las que se hospeden en él.

Es posible que en algún grado esta falta de seguridad sea sólo una cuestión de imagen, de temor al cambio, pero sin duda hay un punto que es incontestable y que requiere de un tratamiento delicado por parte de los proveedores. Los sistemas están más allá de las fronteras físicas de su dueño y por tanto también sus datos, y además, salvo en el caso de que la empresa en cuestión pueda pagar una nube privada, estos sistemas se comparten entre diferentes clientes lo cual conlleva dos posibles riesgos. El primero es que se debe asumir una confianza hacia el proveedor que permita al cliente convencerse de que ningún administrador con acceso a la infraestructura realizará ninguna acción indebida, voluntariamente o no. Y en segundo lugar debemos confiar en que otros clientes que compartan la misma infraestructura, y que por tanto tengan un aparente mayor acceso que un usuario externo, no podrán acceder a la información del cliente.

Esta situación es muy parecida a la que se daba a principios de la primera década de siglo cuando los proveedores de servicios de Internet optaron por incluir cifrado *IPSec* en las comunicaciones que atravesaban sus dispositivos ya que ciertos clientes no confiaban en los dos puntos descritos anteriormente. Por ello se crearon dos mecanismos de transporte diferenciados, MPLS y *IPsec* sobre MPLS.

Nube personal

Se ha comentado en secciones anteriores que el concepto de nube engloba varios principios como son un gran ancho de banda, virtualización, orquestación, ubicuidad, provisión dinámica, sencilla y eficiente y por último pago según se consume. Casi todas estas características pueden aplicarse al ámbito del hogar pero la realidad es que el nivel de necesidad de todas ellas se reduce, perdiendo en algún caso sentido, cuando nos referimos a un ámbito personal. Se hace necesario por ello adaptar esta lista para que la aplicación de una nube en el ámbito del hogar tenga cierto nivel de realismo. En este sentido podemos redefinir este índice en una frase.

"*Una nube personal se implementa para que uno o varios servicios escogidos por un individuo puedan ser accesibles desde cualquier punto con acceso a Internet, replicados en varias localizaciones, y que una gestión sencilla le permita mantenerlo disponible de manera continua en alguna de ellas pudiendo en cualquier momento modificar la cantidad de recursos reservados al servicio*."

Es muy importante darse cuenta de que para un hogar el tipo de servicio más adecuado sería *SaaS*, de ahí que se indique '*para que uno o varios servicios escogidos por un individuo'.* La razón es evidente, dando por hecho la limitación de conocimiento técnico del usuario lo ideal es que se le ofrezcan unos servicios prefabricados y sencillos de configurar, casi del tipo *out-of-the-box*[28].

Quizá un punto que debería añadirse teniendo en cuenta el ámbito del que estamos hablando es la gratuidad, no de los recursos *hardware* pero sí de los que proporcionan las capacidades indicadas en la definición anterior. Evidentemente no se pretende que los servicios que se puedan prestar tengan un rendimiento de acceso profesional pero sí suficiente para, como se comenta en la introducción, ofrecer la posibilidad al individuo de transformarse en productor de sus propias ideas sin necesidad de tener que recurrir para ello a una empresa que le cobre por algo que ya podemos hacer nosotros, aunque sólo sea por almacenar y compartir nuestros datos entre varias localizaciones.

Para ello un elemento importante sería el uso de un servicio DNS dinámico gratuito que permita asociar una zona DNS a nuestro servicio aunque nuestra IP de acceso a Internet sea dinámica[29]. Esto puede permitir que trabajando con nombres DNS podamos crear una infraestructura de nodos deslocalizados con acceso público que se comuniquen entre sí y se sincronicen. Y como no, el uso de la virtualización gratuita, como por ejemplo la basada en contenedores a través de *Docker* que se analiza en este documento dentro de la prueba de concepto.

En cuanto al *hardware* la gran ventaja de sistemas como *Docker* es que se instalan nativamente sobre sistemas *Linux*. Esto permite que las posibilidades a la hora de escoger el computador que se encargará de ofrecer los servicios sean extremadamente variadas y más asequibles a cualquier presupuesto[30]. Como por ejemplo PC de sobremesa, Mini ITX, Mini PC, sistemas embebidos o *Raspberri Pi*. Cada soporte tendrá sus ventajas e inconvenientes. Una explicación detallada de cada uno así como la guía para escoger uno u otro la podremos encontrar en [SHAwLaRP].

Por otra parte atendiendo a la lógica y a la definición ofrecida anteriormente se puede deducir que la característica que da nombre a la nube, la ubicuidad, en el ámbito del hogar queda relegada a un segundo plano. O al menos drásticamente reducida ya que el usuario de este tipo de nube es el propio proveedor lo que descarta el concepto. Por tanto en esencia la idea queda reducida en este ámbito a la unión de distintas localizaciones. Por otra parte se debe tener en cuenta que el adjetivo personal en este tipo de nube no necesariamente se traduce como un único inquilino de una casa o piso, se puede dar la circunstancia de que

[28] Esta característica también llamada por sus siglas OOTB, permite que la funcionalidad que ofrece un producto funcione inmediatamente después de su instalación sin configuración o modificación.

[29] Un ejemplo de servicio gratuito es: https://www.zoneedit.com/free-dns/

[30] Existen sistemas hardware con sistemas operativos *Linux* instalados basados en *Raspberri Pi* que cuestan menos de 15€, aunque este evidentemente no será probablemente el soporte más elegido. Este tipo de sistemas tienen mucho éxito debido a la cantidad de utilidades que pueden instalarse en ellos por lo que por poco precio puedes obtener un servicio dedicado. Ver: http://raspberrypihelp.net/raspberrypi-tutorials/

varias personas con un nexo común decidan llevar a cabo un proyecto a través de la nube uniendo para ello sus recursos. O simplemente varios amigos compartiendo algún tipo de datos y sincronizándolos.

Finalmente un último elemento muy importante que puede llevar a un usuario a alojar sus servicios y datos localmente es la seguridad. De hecho no sólo los usuarios finales tienen en cuenta este punto sino que, tal y como se adelantaba en la introducción, en los últimos tiempos ha surgido un nuevo concepto en la nube empresarial, la nube doméstica, que responde a esta misma necesidad y percepción. En ambos casos el usuario, individuo o empresa, se plantea acercar su nube geográficamente y por tanto optar por una nube cuya infraestructura esté situada en territorio nacional o en su propia casa. En este punto se va a intentar profundizar en la razón por la que de repente surge el concepto de nube doméstica en el ámbito empresarial ya que a través del análisis de estas razones se entenderá también la razón por la que actualmente, y probablemente de manera progresiva, cada vez más usuarios comenzarán a explotar sus propios recursos en lugar de contratarlos. La pregunta clave es, ¿por qué se restringe de pronto el ámbito de la nube a un país concreto cercenando de esta manera en gran medida la idea inicial que subyace bajo el concepto de nube? A priori reducir el espacio geográfico donde puede ser desplegada parece un retroceso en la evolución de esta tecnología.

Como se ha comentado la razón más importante es la seguridad. En el apartado anterior se explicaba que la propia definición de nube trae aparejada una externalización de los datos más allá de las fronteras controladas por el cliente, pero hasta hace relativamente poco tiempo la suspicacia que esto pudiese producir en los clientes de este tipo de servicio se limitaba a los típicos aspectos de la seguridad, ¿accederá sin yo saberlo alguna otra persona a estos datos?, ¿es seguro entregárselos a un tercero para que los custodie?

Curiosamente este concepto a día de hoy se aplica crecientemente en un sentido algo distinto al que cabría esperar. Los usuarios no temen tanto el robo de datos por un tercero, entendiendo como tercero un individuo u otra corporación, sino por las agencias de seguridad de los propios gobiernos y especialmente de los extranjeros. Por tanto subimos un peldaño más para incluir dentro de la definición de atacante a los propios gobiernos soberanos que supuestamente protegen nuestra privacidad y nuestros derechos. De hecho en una gran parte de los casos el principal gobierno extranjero que despierta suspicacias en cuanto a la falta de privacidad de los datos es el estadounidense, debido en gran medida a la estrecha relación que mantiene con Europa.

En efecto, los usuarios, tanto individuos como corporaciones, están preocupados por la impunidad con que parece actuar este país en relación a la privacidad debido a la cantidad de casos que han salido a la luz en los últimos tiempos relacionados con el espionaje a ciudadanos nacionales y extranjeros a través de la información virtual[31] que estos manejan. Y la consecuencia de ello es que los propios proveedores de servicios de nube se ven afectados por la desconfianza de los usuarios a la hora de contratar sus servicios. En este sentido comenzamos a conocer casos en los que para poder comercializar con éxito el servicio el proveedor decide implementar su infraestructura en el mismo país del cliente que la contrata.

Debemos tener en cuenta que la mayor parte del tráfico de Internet pasa a través de Estados Unidos y que los más conocidos proveedores de servicios de nube están situados en este país[32]. Asimismo no ayuda a aumentar la confianza del mercado en este sentido la aparición de decenas de noticias que describen cómo el gobierno americano, a través de sus agencias de seguridad, intenta recopilar y almacenar datos de los consumidores de Internet en todo el mundo de manera masiva. Estos actos generan cierto miedo por parte de todo tipo de usuarios que suele ser rebatido a través de la constante negación de los mismos. Pero cada vez encontramos más debates, opiniones y revelaciones en Internet que corroboran esta actuación por parte del gobierno estadounidense, que evidentemente no es el único que actúa de esta manera, aunque probablemente sí el que hasta la fecha parece haber tenido un mayor éxito. Por tanto ¿no es lícito desconfiar cuando la información manejada por empresas y usuarios se encuentra almacenada o simplemente pasa a través de empresas de este país?

Existen claros ejemplos de este comportamiento invasivo a la hora de manejar datos personales. En ocasiones basados en la explotación de agujeros de seguridad en software existente, como por ejemplo el paquete de

[31] Como información virtual se pretende definir todo aquel dato que es transportado o accesible por Internet o por redes privadas.

[32] Típicos ejemplos son *Microsoft*, *Google* o *Amazon*.

cifrado de código abierto *OpenSSL* y su ya famoso fallo de programación denominado *Heartbleed* que permitía descifrar las comunicaciones que los usuarios mantenían con sus bancos, redes sociales, correos, servicios sanitarios, etc.[33] Pasando por el espionaje masivo durante más de una década por parte de la NSA (Agencia Nacional de Seguridad americana) a sus propios ciudadanos y empresas revelado por *Wikileaks* o *Edward Snowden*[34]. E incluso llegando más allá al presionar a importantes empresas privadas para que lleven a cabo la instalación de puertas traseras en sus dispositivos móviles que permitan acceder a todos los datos locales de los usuarios[35], o para que les permitan consultar información sobre la situación geográfica y hábitos de los usuarios [eLOdG].

Todo esto lo que finalmente está provocando es un nuevo replanteamiento en el concepto de nube en todo el mundo, que aunque sin duda parece ser un retroceso tecnológico obedece a una necesidad de seguridad para el usuario final de este servicio.

Por otra parte más allá de todas estas acusaciones de espionaje ilegal nos encontramos ante un problema legal. Las leyes sobre privacidad de los ciudadanos en Estados Unidos son mucho más laxas que en Europa. De hecho la conocida como *Patriot Act* o *USA freedom Act*, una ley surgida específicamente para combatir el terrorismo en Estados Unidos, autoriza a la NSA a acceder sin cortapisas a datos privados de ciudadanos locales o foráneos alegando para ello la defensa de la seguridad nacional.

La idea fundamental que se repite de manera reiterativa como justificación a este tipo de leyes refrendadas por numerosos gobiernos es la afirmación en última instancia de que a través de ellas únicamente se persigue el bien y la seguridad común por lo cual la población inocente no tiene nada que temer ni ocultar. Pero aunque este discurso cala en ciertos sectores de la sociedad estadounidense no consigue aliviar el recelo surgido en la ciudadanía europea muy celosa de su privacidad, que ve en él más una peligrosa referencia a la literatura de Orwell que una realidad.

Por otra parte, aun dando por hecho que esta aseveración fuese cierta, el gran problema que se desprende de estas acciones es sobre todo el almacenamiento por tiempo indefinido de los datos. Esto permite guardar todo tipo de información sobre una persona anónima en la actualidad que pueden ser utilizados en el futuro si ese ciudadano se convierte en alguien relevante dentro de la sociedad, o simplemente realiza alguna acción fuera de los intereses del custodio de esos datos. Esto sin duda da una vuelta de tuerca más al concepto de derecho al olvido en Internet[36]

En el caso americano, a pesar de que ha habido ciertas modificaciones en su *USA freedom Act* el año pasado cuando expiró su validez y tuvo que renovarse, la realidad es que dichas variaciones parecen más bien cosméticas y de cara a la galería debido precisamente a la creciente percepción en el extranjero de que existen problemas de privacidad en este país que pudiesen afectar a su economía[37].

Por tanto se puede concluir que a día de hoy una de las grandes ventajas que proporciona la nube, sobre todo a los propios proveedores del servicio, y que consiste en la deslocalización de la información y por ende en el

---

[33] http://www.huffingtonpost.com/2014/04/11/nsa-heartbleed_n_5134813.html
"*The National Security Agency knew of the existence of the catastrophic bug for at least two years and kept it a secret from the public and the cybersecurity community in order to exploit it*"

[34] http://www.theguardian.com/commentisfree/2013/aug/11/nsa-internet-surveillance-email
http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data
http://www.ciphercloud.com/blog/can-domestic-cloud-services-address-cloud-computing-risks:
"[…] *the growing mountain of revelations of government agencies demanding cloud service providers (CSPs) to hand over private consumer and enterprise data. For markets outside the United States, fears are particularly strong that any data stored on US soil or by US-based CSPs is vulnerable to NSA surveillance*."

[35] http://www.apple.com/customer-letter/
"*While we believe the FBI's intentions are good, it would be wrong for the government to force us to build a backdoor into our products. And ultimately, we fear that this demand would undermine the very freedoms and liberty our government is meant to protect*."

[36] "Dadme seis líneas escritas por la mano del hombre más honesto, y yo encontraré algo para hacerlo ahorcar." Armand Jean du Plessis, Cardenal y Duque de Richelieu (1585 - 1642) .

[37] http://www.usatoday.com/story/news/politics/2015/06/02/patriot-act-usa-freedom-act-senate-vote/28345747/
"*It also begins the process of rebuilding the confidence of Internet users worldwide in American providers of digital services*."
http://edition.cnn.com/2015/05/30/politics/what-happens-if-the-patriot-act-provisions-expire/index.html
"*The government has numerous other tools, including administrative and grand jury subpoenas, which would enable it to gather necessary information*."

desconocimiento de donde se encuentran los datos por parte de los usuarios, se convierte a su vez en un problema de confianza que deriva en un riesgo de negocio. Al no conocer el paradero de sus propios datos el cliente no sabe que leyes están siendo aplicadas a los mismos ya que estas serán las correspondientes a las que aplique el país donde estos residen. Tenemos cierta información de cuáles son estas leyes en Estados Unidos y evidentemente en nuestros propios países, o al menos podemos tenerla, pero es evidente que tampoco podemos confiar mucho en las leyes que se apliquen en países tan opacos como China por citar un único ejemplo. Por todo ello es probable que los usuarios cada vez exijan más a sus proveedores de servicio, que se formalice bajo contrato el lugar donde vayan a residir sus datos en cualquier momento así como el uso que se hará sobre ellos. Y quizá, dando un paso más allá, esta sea una de las razones clave para que los usuarios decidan implementar sus servicios en sus propios hogares.

En cuanto al tipo de servicios que se ofrecen hoy en día dentro del concepto de nube personal, la realidad es que la mayor parte de los ellos se basa en la posibilidad de realizar copias de respaldo y acceso a ficheros. Muchas empresas ofrecen diferentes opciones en la mayoría de los casos supeditadas a la compra de una NAS que se instala en casa del usuario y le permite guardar los datos que tengamos repartidos por nuestros ordenadores para después acceder a ellos y compartirlos por Internet a través de una gestión web sencilla[38]. Una de las mejores opciones actualmente es el software libre O*wncloud*[39], el cual permite sincronizar todos los datos que tengamos tanto en ordenadores como almacenados en *Amazon S3, Sharepoint* y otros. Además tiene herramientas colaborativas, sincronización con móviles, calendarios,... Este software es básicamente una herramienta de apoyo y de compartición de ficheros, pero a diferencia de una nube los recursos residen en casa del usuario únicamente, o en su defecto funciona como puerta de enlace para acceder al recurso si este se encuentra en otro lugar como por ejemplo en la nube de *Amazon*, en cualquier caso no proporciona en sí mismo un servicio de nube.

En las siguientes secciones y a partir del conocimiento de lo que existe en el mercado enfocado al mundo del hogar se analizará si se puede llegar un paso más allá a través de la opción *Docker,* y si no es así, si puede quizá existir algún lugar donde esta arquitectura pueda cubrir un hueco importante.

Dockers

La idea que subyace dentro del concepto de *Docker* es sencilla, pretende ser un híbrido entre los antiguos sistemas monolíticos que permitían correr varias aplicaciones concurrentemente sobre un mismo sistema operativo, y que siguen siendo los utilizados en entornos domésticos, y los modernos modelos de virtualización que sacan partido de la potencia del hardware actual para poder ejecutar sobre una misma máquina varios sistemas operativos a la vez con sus correspondientes aplicaciones.

A través de los diagramas que se han mostrado en secciones anteriores[40] podemos comprobar a primera vista que, a pesar del paso evolutivo que supone la virtualización de los sistemas, este modelo evidencia dos grandes problemas relacionados con el rendimiento, la capacidad de hardware necesario para poder soportarlo y la ingente cantidad de ficheros incluidos que no serán utilizados:

En cuanto a la capacidad que requiere una máquina para poder correr varios sistemas operativos a la vez, es cierto que cada pocos años se duplica la capacidad de los procesadores[41] y que la memoria y el almacenamiento mejoran vertiginosamente en cuanto a rendimiento y velocidad. De hecho ya hace tiempo que muchas empresas han pasado de usar discos duros de acceso secuencial basados en platos giratorios a los que se accede a través de un brazo lector, de manera similar a un tocadiscos, a los discos duros de estado

[38] La realidad es que estas soluciones no suelen cumplir con el concepto de nube y lo utilizan más como un término de marketing. No conectan diferentes PC deslocalizados con el servicio replicado sino que es un único dispositivo el que recolecta todos los datos y al que se puede acceder. Como ejemplo de sincronización de todos los ficheros elegidos de nuestros ordenadores a nuestra NAS: http://www.wdc.com/en/products/products.aspx?id=1640 .
Un paso más allá tenemos la posibilidad de crear grupos de acceso a nuestra NSA con distintos permisos y revisión de versiones de nuestros ficheros: https://www.synology.com/en-global/dsm/6.0/cloud_file_syncing

[39] Ver *Owncloud* para empresa: https://owncloud.org/nine/

[40] Figuras 3 y 5.

[41] La ley de Moore afirma que aproximadamente cada dos años se duplica la velocidad de un procesador. [Ref: http://www.mooreslaw.org/]

sólido, mucho más rápidos en relación tanto al acceso de los datos como a su lectura y de los que sólo les queda de *disco* el nombre. Pero a pesar de todo ello esta tecnología es costosa y la potencia requerida por los modelos de virtualización es muy alta, un punto que se convierte en crítico cuando hablamos del ámbito personal. Se debe tener en cuenta que este modelo requiere por seguridad que por cada aplicación que presta un servicio se ejecute un sistema operativo completo por debajo de ella, además del propio sistema que a su vez se encuentra corriendo bajo todas las virtualizaciones.

Por otra parte encontramos que la exigencia de tener tantos sistemas operativos ejecutándose a la vez genera a su vez otro inconveniente como es la cantidad de ficheros necesarios en un modelo de virtualización. En general una aplicación se ejecuta en un sistema operativo que incluye miles de ficheros que en su mayoría nunca serán utilizados por esa aplicación. Esto es así debido a que los sistemas operativos están pensados para ser ejecutados en entornos donde a priori se desconoce para qué serán utilizados. Algo similar nos ocurre por ejemplo con nuestros dispositivos móviles. Un móvil con un sistema operativo *Android* por defecto[42] tendrá un peor rendimiento que un dispositivo con un sistema operativo ad-hoc, y tendrá que suplir este problema a través de un hardware más potente o personalizando el sistema operativo para su modelo específico de móvil. Este peor rendimiento viene ocasionado por la necesidad de adaptarse a cualquier hardware que pueda ser utilizado en un teléfono lo que permite por ejemplo que pueda ser instalado en un móvil con tecnología Intel o AMD. Pero, ¿no mejoraría el rendimiento si elimino toda aquella parte del sistema que nunca vaya a ser utilizada?

Por lo tanto nada más arrancar el sistema nos encontramos con decenas de aplicaciones que jamás utilizaremos y que a su vez vienen acompañadas de sus librerías, sus herramientas asociadas, sus ficheros de log, etc. Este problema se acrecienta en los entornos virtuales actuales ya que debemos ir sumando sistemas operativos por cada servicio que desplegamos. Ciertamente podemos solventar de alguna manera este problema intentando 'higienizar' cada uno de los sistemas operativos virtuales que utilizamos, pero inevitablemente siempre será necesario duplicar una y otra vez los ficheros del sistema ya que los encontraremos tanto en el sistema base, host, como en cada sistema virtual. Si bien es cierto que este formato es útil cuando por alguna razón necesitamos correr diferentes sistemas operativos o con diferentes versiones.

Además de estos problemas de rendimiento los sistemas virtuales clásicos adolecen de ciertos problemas de seguridad sobre lo cual se ha escrito mucho y de hecho en este mismo documento se ha comentado en la sección 'Inconvenientes'. Pero además de lo ya comentado hay otro punto a tener en cuenta. Sabiendo que la base de un ataque a un sistema se basa en la vulnerabilidad de las aplicaciones que corre así como la del propio sistema operativo y por ende de todos los ficheros que ambos utilizan, tanto más vulnerables seremos cuando más elevado sea el número de ficheros que utilicemos para correr nuestras aplicaciones. Por otra parte cuantos más ficheros tengamos que controlar y actualizar más probable es que se cometa un error al hacerlo. Pongamos el caso en el que se arranquen veinticuatro máquinas virtuales que den servicio a tres aplicaciones diferentes, lo cual no es extraño. Sin necesidad de realizar cálculos nos podemos hacer una idea de lo que puede suponer asegurar, gestionar y tener actualizados ficheros de posiblemente diferentes sistemas operativos, con diferentes versiones más sus aplicaciones asociadas.

Otro dato a tener en cuenta es la importancia de la disponibilidad de los servicios. Las empresas firman exigentes contratos para establecer unas rigurosas y elevadas garantías de seguridad que vienen definidas incluso por el número de minutos anuales que un servicio puede no estar disponible durante un año[43]. Esta característica de los servicios, la disponibilidad, es de hecho tan esencial que recae dentro del ámbito de la seguridad informática de la misma forma que lo hacen la confidencialidad, autenticidad o integridad. Por ello es importante simplificar al máximo la ejecución de nuestros servicios, siempre cumpliendo altos estándares de fiabilidad, para permitir una rápida puesta en marcha de nuestros sistemas. En el caso de los modelos virtuales actuales la velocidad con la que los sistemas arrancan, y por tanto levantan los servicios que soportan, no son excesivamente elevadas ante ciertas circunstancias como por ejemplo ante la caída de un host[44]. Esto es debido a que para levantar todos los servicios se necesitan levantar muchos sistemas y aplicaciones antes como son el propio sistema base, la aplicación que soporta todas las máquinas virtuales y

[42] Recalco la expresión 'por defecto' ya que lo habitual es que las marcas de móviles personalicen el sistema operativo Android para ajustarlo al hardware que utilizan sus teléfonos.
[43] SLA: Service Level Agreement
[44] Llamamos host al sistema base que soporta todas las máquinas virtuales.

virtualiza el *hardware* (*hypervisor*) y finalmente las máquinas virtuales en sí mismas y sus aplicaciones con toda su conectividad en perfecto estado.

Por último y como punto final se debe tener en cuenta la complejidad a la hora de llevar a cabo la administración de los sistemas. Sabemos que no es trivial integrar las capacidades de cada máquina virtual ofrecidas por su *Kernel* con los recursos ofrecidos por el sistema base que a su vez arranca su propio *Kernel* y que finalmente se integra con su entorno. Un ejemplo típico podría ser el acceso a la red, lo que en un servidor normal consiste en una acción tan simple como configurar los parámetros básicos de red y conectar la máquinas a través de un cable, en un sistema de virtualización clásico requiere de una configuración doble ya que se deben configurar redes virtuales dentro de un mismo hardware que permitan comunicar las máquinas virtuales entre sí y a su vez con el exterior a través de la propia red física del sistema base que conecta con el resto del mundo. Cualquier sistema virtual deberá realizar estos pasos pero cuando se trata de máquinas virtuales con sus propios sistemas operativos las configuraciones pueden ser más afinadas y la vez más complejas. Aunque a priori esto pueda parecer un problema menor una vez que todo está configurado y funcionado, cuando surgen incidencias su resolución puede llegar a complicarse.

Una vez enumeradas estas objeciones, ¿en qué pueden favorecernos *Docker*? En gran medida el enfoque de *Docker* para afrontar la mayoría de estos inconvenientes se puede apreciar a través del siguiente diagrama donde se aprecia claramente la simplificación de capas en esta arquitectura comparado con la virtualización clásica.

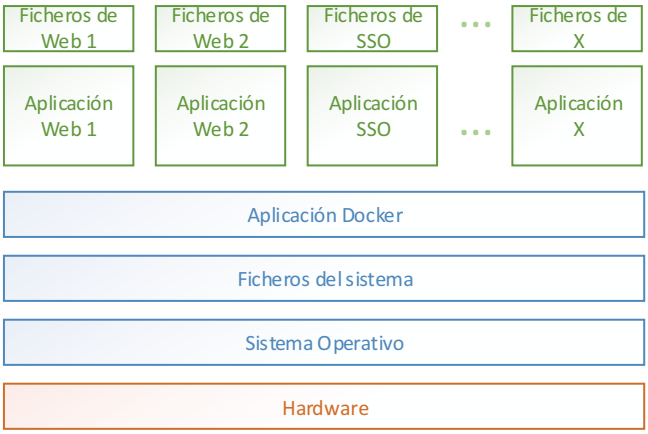

Figura 14 – *Arquitectura Docker*
La figura muestra las partes básicas necesarias de las que se compone un sistema virtual Docker para poder ofrecer un servicio

Como resumen rápido *Docker* trata de correr aplicaciones existentes o creadas por nosotros dentro de un entorno aislado que llamamos contenedor, en el diagrama en verde. Este contenedor consta de todos los ficheros que pueda necesitar la aplicación específica que queremos correr y ninguno más, es decir, no contendrá ficheros superfluos que puedan ralentizar el funcionamiento de la aplicación de alguna manera o simplemente hacer su gestión más compleja.

El concepto diferenciador de *Docker* consiste precisamente en la inclusión dentro de cada contenedor únicamente de los ficheros estrictamente necesarios para correr nuestra aplicación. En principio este concepto tiene cierto parecido al concepto de virtualización.

Claramente se distingue la menor cantidad de ficheros de sistema y aplicación necesarios para ejecutar las mismas aplicaciones en un sistema *Docker*. Esto repercute primeramente en el peso de los sistemas que deben ejecutarse y por tanto en el tiempo que se necesita para que todo el software esté disponible ante una caída. En segundo lugar como ya se ha comentado a menor número de ficheros menor posibilidad de aprovechar sus vulnerabilidades. Por otra parte ante sistemas más ligeros se requieren menos recursos para ejecutarlos. Y por último, como punto final los contenedores son tan básicos que su configuración es más sencilla y por tanto su gestión más fácil.

Es evidente que no todo es perfecto, por ejemplo, esta reducción de ficheros obliga a que las aplicaciones que corren dentro de los contenedores tengan que acceder a servicios que ofrece el sistema operativo del host, y que estos se compartan entre otros contenedores. Esto tampoco parece una fórmula ideal para mejorar la seguridad en este tipo de arquitecturas, de hecho en este sentido el nivel de seguridad es todavía menor comparado con la sistemas de máquinas virtuales.

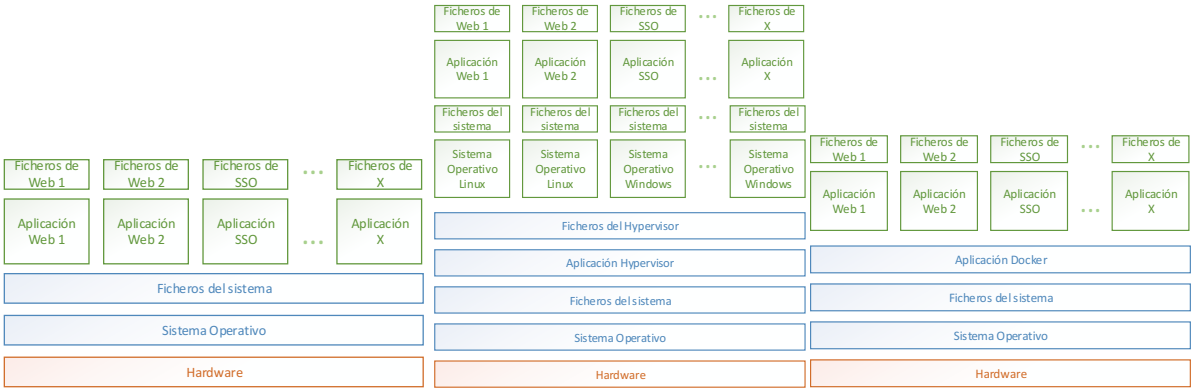

Figura 15 - *Evolución*

La figura muestra las partes básicas necesarias de las que se componen los distintos sistemas que ofrecen servicios comparando unos con otros

Las diferentes capas necesarias en cada sistema que muestra la figura anterior nos permite de hecho ver claramente como en relación a la cantidad de ficheros necesarios la opción de *Docker* se acerca más a la primera que a la segunda opción. De hecho en un estudio de IBM se muestra a través de un completo estudio como el rendimiento de *Docker* es bastante bueno si comparamos con otros sistemas de virtualización, en el caso del artículo KVM[45].

Una ventaja importante para usuarios menos especializados es la existencia de miles de aplicaciones ya montadas por otros usuarios que básicamente sólo hay que meter en un contenedor sin modificar para comenzar a funcionar. Y por otra parte se puede ejecutar sobre casi cualquier plataforma:

- Sistemas operativos de uso general como *Linux, Windows*,…
- En entornos de nube.
- En sistemas operativos dedicados y diseñados para correr estos contenedores como *Atomic*. La ventaja de estos últimos es que debido a que *Docker* necesita menos 'ficheros' para correr, la máquina virtual que los contiene puede ser más ligera y por tanto funcionar con mayor eficiencia y ocupar menos espacio.
- En la nube (*Openstack, KVM, google cloud*,…)

[45] La mejor opción de rendimiento requiere de ciertas características de configuración que en esta parte del documento todavía no se está capacitado para entender. Fuente:
http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

Perspectiva de usuario

Hasta el momento se han comentado diferentes posibilidades a la hora de ofrecer servicios de nube a los usuarios situados en su hogar, estos paradójicamente se alejan más del concepto de nube cuanto más simple es su utilización. Esto tiene sentido ya que en su gran mayoría los usuarios que están interesados por este tipo de tecnologías y su uso no tienen excesivo conocimiento al respecto y por tanto las empresas enfocan su esfuerzo en vender un dispositivo más o menos caro con el mensaje 'pínchelo y listo'. Pero esta tendencia es probable que vaya cambiando con el tiempo según estas tecnologías maduren y la población vaya renovándose a medida que aumenten las generaciones nacidas en la era digital.

Es evidente que la perspectiva del usuario con respecto a una tecnología es lo más importante para que las novedades se asienten y tengan éxito, pero también es cierto que nuestra sociedad está muy fragmentada en cuanto a su conocimiento y sus intereses. Por tanto no se debe excluir una opción y abandonar su evolución simplemente porque para una parte significativa de los usuarios actuales con desconocimiento excesivo les resulte demasiado complejo. Existen muchos nichos de mercado y a cada uno se le debe dar una opción adecuada a su situación. Nos encontramos en una época que se caracteriza por una gran diversidad en cuanto al nivel de conocimiento en relación a las nuevas tecnologías, lo realmente importante es analizar primero si existe mercado actual y segundo si este crecerá en el futuro teniendo en cuenta la evolución vertiginosa de la sociedad.

Por ello cuando se llegue a una conclusión con respecto a la nube personal en este documento se tendrá muy en cuenta este aspecto teniendo en cuenta que cada vez más gente tendrá un conocimiento medio de las nuevas tecnologías mientras que la población caracterizada por su desconocimiento casi total de las mismas se irá reduciendo. Como se aprecia en la figura es probable que la curva que muestre la cantidad de personas que tienen un alto conocimiento de las mismas se mantendrá poco variable en los próximos años.

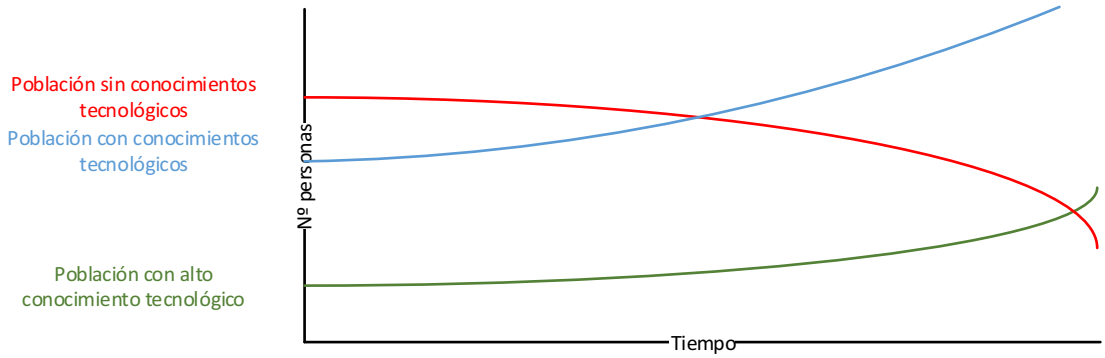

Figura 16 – *Conocimiento técnico de la población*
La figura muestra la evolución de los conocimientos tecnológicos de la población a lo largo del tiempo

Prueba de concepto[46]

A continuación se describen los pasos básicos necesarios para realizar una instalación de *Docker* en el sistema operativo *Debian 8.3.4*. El motor *Docker* puede ser instalado sobre gran variedad de sistemas operativos como hemos visto pero este trabajo no pretende ser un manual genérico de instalación por lo que este apartado se limita a describir los pasos estrictamente necesarios que permitan entender más claramente los conceptos teóricos. Para consultar otras instalaciones la mejor opción es dirigirse a la página oficial de *Docker*[47].

Además es importante destacar que muchos de los pasos seguidos en esta sección pueden ser extraídos así mismo de la página oficial, la intención al detallar algunos aquí es la de ir acompañándolos de explicaciones y de este modo asimilar los conceptos de una manera didáctica. También se añaden varias acciones específicas dependientes del sistema operativo escogido que no vienen documentadas y que han sido resueltas a lo largo de la prueba de concepto.

Una vez acabado este proceso de instalación y pruebas elementales en nuestro sistema nos encontraremos con el siguiente entorno funcionando:

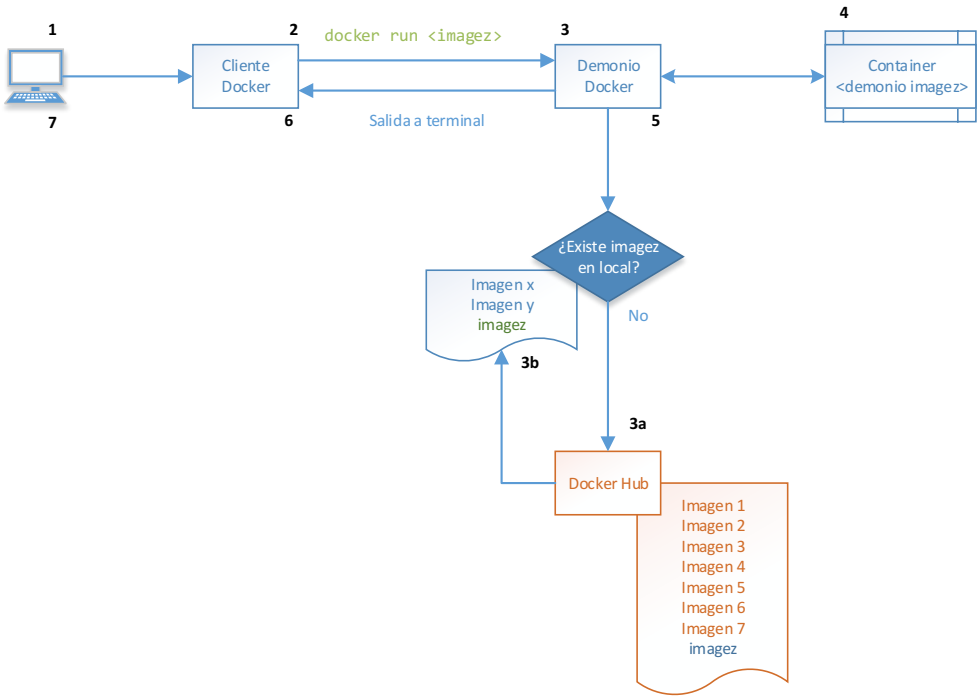

Figura 17 – *Dockers: Funcionamiento básico*
La figura muestra el funcionamiento básico de la virtualización basada en contendedores Dockers

A partir de aquí tendremos una idea básica de cómo crear un contenedor para proporcionar un servicio así como de los conceptos fundamentales a partir de los cuales podremos seguir avanzando. Es importante tener claros ciertos conceptos dentro del mundo *Docker*, para consultar más detalles lo mejor es acudir al apartado 'Glosario'. Los pasos definidos en este diagrama son los siguientes:

1. El usuario ejecuta un comando para crear un contenedor que corra una imagen específica.
2. Este comando es recibido por el cliente *Docker* que a su vez se lo reenvía al demonio *Docker*.
3. El demonio *Docker* comprueba si esa imagen existe en su registro local de imágenes:

    a. En caso de que no exista conecta con *Docker Hub* y la busca en su librería.

[46] Se han consultado varias fuentes de Internet y [DC2Ed] durante la prueba de concepto.
[47] https://docs.docker.com/linux/

b. Si la encuentra la baja al sistema local y si no indicará que no le ha sido posible encontrarla[48]:

```
# docker run test
Unable to find image 'test:latest' locally
Pulling repository docker.io/library/test
docker: Error: image library/test not found.
See 'docker run --help'.
```

4. Una vez la imagen se encuentra en el registro local de imágenes, ya sea porque estaba o porque ha sido bajado del registro *Docker*, el demonio crea un contenedor para esa imagen, la cual es ejecutada dentro.
5. Si este contenedor en concreto debe producir una salida para el usuario, el contenedor se la envía al demonio.
6. El demonio se la envía a su vez al cliente *Docker*.
7. El cliente *Docker* se la envía a la salida estándar, es decir la pantalla, y el usuario la ve.

Por tanto nos damos cuenta de varias cosas, la primera es que el concepto de imagen en el mundo *Docker* es el mismo que el de aplicación en nuestro entorno habitual, así como un contenedor es lo mismo que la instancia de un demonio, es decir, una aplicación ejecutándose. Por otra parte un contenedor no tiene por qué correr una aplicación que proporcione un servicio a usuarios externos de manera ininterrumpida, puede simplemente ejecutarse durante un mínimo periodo de tiempo para realizar una tarea y después parar. Aunque es evidente que la idea que subyace en el uso de *Dockers* es precisamente la de prestar un servicio y por tanto la mantener aplicaciones corriendo de manera continua.

Para asentar estos conceptos a continuación se describe paso a paso la instalación y uso de *Docker* sobre una plataforma *Linux Debian*. Posteriormente y como parte de una instalación más avanzada se realizará la instalación sobre este mismo sistema de una herramienta de orquestación para *Docker* llamada *Kubernetes*[49].

Para realizar esta prueba de concepto se ha utilizado un portátil con procesador *Intel® Core™ i7* con 16GB de RAM sobre el que corría un sistema operativo *Windows 7* de 64 *bits*. Así mismo se ha instalado sobre él el *software* de virtualización *VMWare Workstation 9* que ha funcionado como *Hypervisor* de una máquina virtual corriendo un *Debian 8.3.4* 64 bits básico, sin aplicaciones extra ni entorno de ventanas, *X-Windows*. Finalmente sobre este *Linux Debian* se ha instalado el entorno *Docker* con diferentes contenedores corriendo a la vez. De manera esquemática el entorno utilizado se muestra en la figura siguiente:

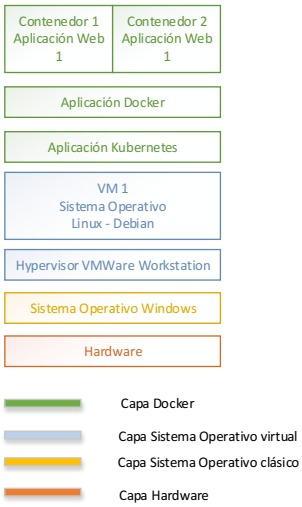

Figura 18 – *Entorno de pruebas*

La figura muestra las partes básicas necesarias de las que se compone un sistema virtual Docker para poder ofrecer un servicio

[48] La imagen *hello-world* sirve para mostrar que *Docker* ha sido correctamente instalado. Se encarga de realizar el proceso completo descrito en este apartado, conectar con *Docker Hub*, bajar la imagen y ejecutarla en el contenedor y finalmente mostrar la salida por pantalla. Independientemente de que esta imagen se encuentre en local siempre se realizará este proceso.

[49] Existen numerosas herramientas de orquestación. El uso por parte de *Docker* de un *API fullREST* permite que se puedan desarrollar e integrar fácilmente este tipo de aplicaciones.

Debido a que la aplicación *Kubernetes* consume bastantes recursos la máquina virtual utilizada para soportarla se ha configurado con 30GB de espacio y 4GB de RAM. Tras realizar pruebas con menor cantidad de recursos se ha concluido que estos son los recursos mínimos necesarios.

Nótese que *Docker* no requiere de la instalación previa de *Kubernetes* para poder funcionar. La gran ventaja de *Kubernetes* como orquestador es la capacidad de gestionar de manera automática las réplicas de los contenedores, que en términos de *Kubernetes* se agrupan en *pods*. Es la propia aplicación la que se encarga de monitorizarlos, levantar otros contenedores si alguno se cae, balancear el tráfico entre ellos, etc. Todo sin intervención manual.

Instalación de Dockers

Queda más allá de la intención de este documento detallar los pasos necesarios para instalar el sistema virtual que soportará esta prueba de concepto. Principalmente porque no es necesario ya que *Dockers* puede correr sobre nuestro sistema operativo directamente sin necesidad de virtualizarlo, de hecho en entornos reales esa será la elección más adecuada ya que es la que aprovechará completamente los beneficios de *Dockers*.

Por tanto una vez que tenemos instalado un sistema operativo *Debian* compatible seguimos el siguiente procedimiento:

1. Se recomienda tener un *Kernel* mayor de 3.10:

```
# uname -a
Linux debian 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt20-1+deb8u4 (2016-02-29) x86_64 GNU/Linux
```

2. Para instalar el paquete *Dockers* usaremos el comando `apt-get` lo que facilita el proceso ya que instala automáticamente todos los ficheros necesarios preconfigurados y deja la aplicación preparada para ser ejecutada. Debido a que usaremos un repositorio de software al que nos conectaremos de manera segura a través de https instalamos los paquetes necesarios para que la aplicación `apt` pueda bajarlos. Para ello además deberemos añadir la clave pública del servidor remoto al que nos conectaremos lo que nos permitirá autenticarnos y bajar el paquete *Docker*:

```
# apt-get install apt-transport-https ca-certificates
# apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
58118E89F3A912897C070ADBF76221572C52609D
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir
/tmp/tmp.bUz4ajCBmT --no-auto-check-trustdb --trust-model always --primary-keyring
/etc/apt/trusted.gpg --keyring /etc/apt/trusted.gpg.d/debian-archive-jessie-automatic.gpg --
keyring /etc/apt/trusted.gpg.d/debian-archive-jessie-security-automatic.gpg --keyring
/etc/apt/trusted.gpg.d/debian-archive-jessie-stable.gpg --keyring /etc/apt/trusted.gpg.d/debian-
archive-squeeze-automatic.gpg --keyring /etc/apt/trusted.gpg.d/debian-archive-squeeze-stable.gpg
--keyring /etc/apt/trusted.gpg.d/debian-archive-wheezy-automatic.gpg --keyring
/etc/apt/trusted.gpg.d/debian-archive-wheezy-stable.gpg --keyserver hkp://p80.pool.sks-
keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
gpg: requesting key 2C52609D from hkp server p80.pool.sks-keyservers.net
gpg: key 2C52609D: public key "Docker Release Tool (releasedocker) <docker@docker.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

3. Para que `apt` pueda bajar el paquete del repositorio correcto debemos añadir su dirección en el fichero de fuentes de `apt` ya que las fuentes por defecto de *Debian* no lo contienen. Así que al principio del fichero `/etc/apt/source.list` añadimos la línea `deb https://apt.dockerproject.org/repo debian-jessie main` y actualizamos[50]:

```
# apt-get update
```

[50] Actualmente existen paquetes disponibles para las arquitecturas de *Debian, Jessie, Wheezy y Stretch*.

4. Comprobamos que el repositorio que estamos usando es el correcto:

```
# apt-cache policy docker-engine
  Installed: (none)
  Candidate: 1.10.2-0~jessie
  Version table:
     1.10.2-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.10.1-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.10.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.9.1-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.9.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.8.3-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.8.2-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.8.1-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.8.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.7.1-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.7.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.6.2-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.6.1-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.6.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
     1.5.0-0~jessie 0
        500 https://apt.dockerproject.org/repo/ debian-jessie/main amd64 Packages
```

5. Instalamos el paquete usando el usuario *root*:

```
# apt-get install docker-engine
```

6. Comprobamos que el servicio está corriendo:

```
# ps aux|grep docker
root      5092  0.1  3.1 336416 31636 ?        Ssl  23:34   0:00 /usr/bin/docker daemon -H fd://
root      8491  0.0  0.2  12728  2068 pts/1    S+   23:38   0:00 grep docker
```

7. Podemos consultar todos los ficheros usados por el motor *Docker*. Veremos donde se encuentran los ficheros de configuración, así como la documentación (`man`) y sus scripts asociados:

```
# dpkg-query -L docker-engine
```

8. Ahora comprobamos que el entorno está funcionando correctamente. Para ello ejecutamos una imagen básica que será bajada del *Docker Hub* y posteriormente ejecutada, es decir convertida en *container*. Este *container* nos devolverá en pantalla la frase `Hello World` y acabará. En este punto habremos superado los siete pasos del diagrama que se muestra al inicio de esta sección.

```
# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
03f4658f8b78: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7
Status: Downloaded newer image for hello-world:latest

Hello from Docker.
This message shows that your installation appears to be working correctly
```

Podemos bajarnos no solo una imagen simple consistente en un comando cuyo resultado al correrlo como un contenedor sea una salida directa por pantalla, sino imágenes más complejas como un sistema operativo básico en cuyo caso podremos ejecutar dentro del contenedor que lo corre cualquiera de las aplicaciones que contenga. Para ello usamos el comando `docker run <contenedor> <comando>`. Hay que tener en cuenta que el contenedor sólo estará ejecutándose durante el tiempo en el que se ejecuta la imagen que hayamos escogido y posteriormente el contenedor se parará. Veamos esto a través de ejemplos.

En esta primera ejecución de una imagen dentro de un contenedor, punto 8, hemos visto una aplicación muy simple que únicamente mostraba por pantalla un mensaje indicando que nuestra instalación *Docker* funciona correctamente. Una vez mostrado este mensaje por pantalla al usuario el contenedor se para, es decir, la imagen deja de ejecutarse. Evidentemente esto no resulta muy útil más allá de la comprobación de que el sistema funciona bien.

Dando un paso más allá podemos hacer que un contenedor ejecute un sistema operativo, por ejemplo otro *Debian*. Para ello lo único que debemos hacer es seguir el mismo proceso que con la imagen anterior usando `docker run`, ya que este comando se encarga de bajar automáticamente la imagen a nuestro sistema local,ya que si no, ésta no existe, y ejecutarla. Pero si recordamos el concepto de Docker comentado en apartados anteriores, ejecutar un sistema operativo como tal no realizará ninguna tarea productiva ya que no correrá ninguna aplicación que dé ningún servicio. Los ficheros necesarios para poder existir los cogerá del propio sistema operativo que hace de *host*, nuestro *Debian 8.3.4*, y ningún demonio será ejecutado por lo que el contenedor no correrá nada. Así que lo que vamos a hacer es ejecutar una de las aplicaciones básicas que vienen integradas en cualquier sistema operativo, un listado de ficheros y directorios.

Por ejemplo podemos realizar un listado de los ficheros del directorio raíz de la imagen *Debian* a través del comando `docker run debian /bin/ls -la`. Este comando al ser ejecutado realizará las siguientes acciones:

- Bajará la imagen del sistema *Debian* a nuestro sistema local ya que no la tenemos.
- Creará un contenedor *Debian* que ejecutará ese comando.
- La salida será enviada al usuario.
- El contenedor se parará ya que el comando `ls` sólo realiza esa acción puntual y se acaba.

Hay que reseñar que pararse no es lo mismo que eliminarse (`docker rm <contenedor>`), de hecho podremos comprobar todos los contenedores existentes en un momento determinado con `docker ps -a.` Esto muestra los contenedores y su estado por lo que aparecerán tanto los que fueron ejecutados y están parados como los que están corriendo actualmente. Por otra parte si tenemos objetos asociados a ese contenedor, como por ejemplo un volumen, no podremos borrarlos hasta que el contenedor haya sido eliminado. Esto lo veremos más adelante.

Una vez más seguimos sin hacer nada realmente útil ya que la utilidad deriva de la posibilidad de crear demonios persistentes dentro de los contenedores que presten algún servicio. Por tanto como punto final de esta introducción a Dockers, vamos a comprobar la razón por la cual este sistema de virtualización tiene un alto grado de eficiencia. Para ello crearemos un contenedor *debian* que corra un terminal *bash* interactivo y comparar por ejemplo los demonios que este contenedor está corriendo para poder compararlos con los que corren nuestro host, que de hecho tiene aplicaciones limitadas corriendo en él. Para ello ejecutamos los comandos indicados abajo. Saldremos del terminal, parando el contenedor, a través del comando `exit`:

```
# docker run -ti debian /bin/bash
root@b80cb11ba702:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  7.0  0.2  20244  2992 ?        Ss   04:12   0:00 /bin/bash
root         7  0.0  0.2  17500  2096 ?        R+   04:12   0:00 ps aux
```

Como se puede comprobar el único demonio que está corriendo nuestro sistema es el propio *bash*. De ahí que el contenedor consuma tan pocos recursos. Si trasladamos esta idea a una aplicación real podemos entender la gran ventaja de los *Dockers* ya que únicamente el demonio necesario para esa aplicación de servicio será el que se esté ejecutando.

En este punto se puede considerar que se ha dado una introducción e idea general de en qué consisten los *dockers*. Salimos del contenedor y damos por finalizada esta fase inicial:

```
# exit
```

Herramientas básicas

En este apartado se van a introducir ciertas utilidades necesarias para trabajar con este sistema de virtualización de aplicaciones. Existen muchas más capacidades que no vienen descritas aquí pero la gran ventaja de *Dockers* es que es muy simple de utilizar una vez se comprende su naturaleza y todos los comandos describen perfectamente su funcionalidad. Por otra parte se han escogido expresamente las herramientas que se han considerado básicas para poder avanzar en este sistema y entender mejor porqué y cómo funciona la orquestación que veremos en el último apartado de esta sección.

Cuando creamos un *Docker* los contenedores se comunican entre sí y con el exterior a través una red interna con capacidad para miles de *hosts*[51] de manera similar a como lo hacen las máquinas virtuales en otros sistemas de virtualización. Esto nos permite crear varios contenedores que sirvan una misma aplicación y balancear el tráfico entre ellos, o también conectar contenedores que cumplan distintas funciones como por ejemplo una aplicación web y su base de datos, o un sistema de seguridad en cadena que analice y filtre el tráfico haciéndolo pasar a través de él (cortafuegos, IPS, AV,…). Podemos comprobar las direcciones IP asignadas a los contenedores de muchas formas pero la más fácil es a través de los comandos que nos proporciona la propia herramienta[52].

```
# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "72862ec83b06bb1c3e751c902ba28613cfbcfffd318d0b5d60df8e47c000fb07",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Containers": {
            "9641d148504411172f76f746a9e6852dc9c6706754d0fc4072430a77fc0a6b48": {
                "Name": "elegant_joliot",
                "EndpointID": "d8914af914d30f28db7c584f61ce02987e25f62bc5815aca1d9985a0ec155edd",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        }
    }
]
```

[51] Concretamente una red de máscara 255.255.0.0 (CIDR /16)

[52] Formas más complejas pueden ser:
`docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' clever_ride`

```
# ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:a7:1f:a9:74
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:a7ff:fe1f:a974/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4355 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:178978 (174.7 KiB)  TX bytes:19149016 (18.2 MiB)
```

*la tarjeta `docker0` representa la red bridge de *Docker* y es la red que viene por defecto en todas las instalaciones aunque podemos cambiarla.

Fuera del *Docker:*

```
# traceroute -n www.google.es
traceroute to www.google.es (173.194.116.191), 30 hops max, 60 byte packets
 1  192.168.146.2  10.373 ms  10.195 ms  10.064 ms
 2  * * *
```

Dentro del *Docker:*

```
# traceroute -n www.google.es
traceroute to www.google.es (173.194.116.183), 30 hops max, 60 byte packets
 1  172.17.0.1  0.100 ms  0.047 ms  0.055 ms
 2  192.168.146.2  0.460 ms  5.022 ms  4.873 ms
 3  * * *
```

Podemos crear una red virtual dentro de nuestro propio host para conectar nuestros contenedores así como con el mundo real, modo *bridge*. Y podemos crear una red de contenedores más allá de nuestro *host*, modo *overlay* basada en VXLAN[53].

```
# docker network create velascoNet
8c47152763dd1c2878208b919d3465ab3b9a4b8cd4c2999215dec8117ca587b0
# docker network ls
NETWORK ID          NAME                DRIVER
909f8188697d        none                null
0bb1d1d029b2        host                host
72862ec83b06        bridge              bridge
8c47152763dd        velascoNet          bridge

# ifconfig
br-8c47152763dd Link encap:Ethernet  HWaddr 02:42:b9:92:e6:ad
          inet addr:172.18.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:b9ff:fe92:e6ad/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3468 (3.3 KiB)  TX bytes:7092 (6.9 KiB)

# docker run --net=velascoNet -ti velasco/debian:v1 /bin/bash
# docker network inspect velascoNet
[
    {
        "Name": "velascoNet",
        "Id": "8c47152763dd1c2878208b919d3465ab3b9a4b8cd4c2999215dec8117ca587b0",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
```

[53] Para ello se necesitan VXLAN *gateways* en los centros donde se encuentren los contenedores. Este es el nexo de unión entre SDN *overlay* y la nube. [FoMN2016]

```
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Containers": {
            "986cd7aa46b0303c4915045a14e90f525050a8488c8547235a45689256d0217b": {
                "Name": "evil_booth",
                "EndpointID": "988a7a212abe0038c643bc5c981c53f50716faad7ecf75831b101b2ff20cea03",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {}
    }
]
```

Es importante saber que se puede conectar un container a más de una red para comunicar y aislar grupos de contenedores. De hecho estos cambios en la configuración de red se pueden realizar en caliente sobre contenedores que ya estén corriendo:

```
# docker network connect velascoNet
```

Este tipo de red nos permite conectar los contenedores entre ellos dentro de un mismo host y con el exterior pero si queremos comunicarlos con contenedores existentes en otros hosts necesito otro tipo de red, overlay. Para comunicar los host entre sí y que se puedan intercambiar información sobre sus contenedores necesitamos abrir ciertos puertos en nuestros cortafuegos, en nuestra prueba de concepto debido a que la comunicación se realiza entre máquinas virtuales dentro de un mismo *VMware worksation* no necesitamos tener en cuenta estos requisitos.

Hemos visto que se puede ejecutar una sesión interactiva dentro de un sistema operativo. La utilidad de esta acción no parece ir más allá del aprendizaje, pero en cambio sí que puede resultar útil ejecutar una sesión *bash* dentro de un contenedor que ya está corriendo una aplicación, y por tanto ofreciendo un servicio. Esto podemos hacerlo de dos formas posibles:

```
# docker exec -it contenedor1 bash
# docker attach webNodo1
```

En ambos casos si no queremos alterar el contenedor una vez acabemos de realizar las acciones necesarias salimos sin pararlo ejecutando en este orden Ctrl+p Ctrl+q

Algo que resulta básico y de uso habitual cuando se trabaja con *Dockers* es la capacidad de modificar las imágenes a nuestro gusto, o crear nuestras propias imágenes a partir de aplicaciones[54]. Para ello la ejecutamos dentro de un contenedor y hacemos los cambios necesarios. Por ejemplo instalamos unos paquetes:

```
# docker run -ti debian /bin/bash
echo deb http://ftp.es.debian.org/debian/ jessie main >> /etc/apt/sources.list
apt-get update
apt-get install traceroute
```

A partir de aquí guardo los cambios en la imagen origen y cuando la volvamos a ejecutar lo haremos con esta modificada.

```
# docker commit -m "Added source list and traceroute" -a "Javier Velasco" 8257fd45fb3b
velasco/debian:v1
sha256:a8a95174ae3eca17715051ee0a6d607af11e4394d3a04fd9f2d1c5e0bbf44473

# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
velasco/debian      v1                  a8a95174ae3e        26 seconds ago      144.8 MB
debian              latest              f50f9524513f        9 days ago          125.1 MB
```

[54] Un simple y rápido manual para ello se puede encontrar en:
https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images

```
hello-world         latest              690ed74de00f        4 months ago        960 B

# docker run -ti velasco/debian:v1 /bin/bash
```

Un punto muy importante cuando hablamos de virtualización es la capacidad de limitar el consumo de recursos a las máquinas virtuales, o en este caso a los contenedores. En el caso de los Dockers esto lo hacemos directamente cuando creamos contenedores a través de modificadores (`docker run`). Algunos modificadores útiles son:

- -m: Memoria (en MB).
- -c: Prioridad de CPU.
- -h: Asignación de un *hostnane* específica al contenedor. Si no se especifica lo asigna el sistema aleatoriamente.
- -p: Mapea un puerto externo del host a un contenedor para que se pueda acceder a él desde fuera.
- -P true: Publica estas asignaciones de puertos en el host.
- -rm: Cuando salimos del contenedor con `exit` este se elimina y no tenemos que hacerlo nosotros manualmente.

Volúmenes

Un servicio real probablemente requerirá que varias aplicaciones distintas o que una misma aplicación corriendo diferentes instancias compartan datos. Un ejemplo básico sería una página web. Si tenemos varios contenedores corriendo una misma aplicación web sería lógico pensar que las páginas que la componen sean comunes, es decir, que si cambio algo en la web todos los contenedores actualicen este mismo cambio. Para ello lo más fácil es crear un volumen persistente en el host que soporta los contenedores y que todos accedan a los mismos ficheros. Las claves más importantes son las siguientes[55]:

- Los volúmenes son inicializados cuando un contenedor es creado. Si la imagen con la que se crea un contenedor contiene datos en el punto de montaje, especificando los datos existentes, éstos son copiados en el nuevo volumen una vez que éste se inicializa. Esto no se aplica cuando montamos un directorio en el *host*.
- Los volúmenes de datos pueden ser compartidos y rehusados entre los contenedores.
- Los cambios en los volúmenes de datos son hechos directamente.
- Los cambios en los volúmenes de datos no serán incluidos cuando actualizamos una imagen.
- Los volúmenes de datos persisten incluso si el contenedor es eliminado. De hecho para borrar los volúmenes de datos deberemos hacerlo manualmente.

Lo primero que haremos será crear un nuevo volumen que contendrá la página *web* y lo montaremos en un contenedor no demonizado. Este contenedor nos servirá como referencia de montaje de este volumen para los contenedores que corran el servidor *web*. Para mayor claridad directamente montamos el directorio del host donde se guarda la web por defecto en *Apache2*:

```
# docker create -v /var/www/html:/www --name webData velasco/debian:v1
```

A partir de aquí arrancamos un contenedor con este volumen montado. Para ello lo referenciamos a través de su contenedor 'dueño', esto nos permite compartir datos entre distintos contenedores:

```
# docker run --volumes-from webData --name webNodo1 --net=velascoNet -ti velasco/debian:v1 /bin/bash
```

Arrancamos más contenedores con ese mismo volumen compartido:

[55] Fuente: https://docs.docker.com/engine/userguide/containers/dockervolumes/
Los volúmenes en *Dockers* son un tema complejo en el cual no se profundiza en este trabajo. Para más información consultar bibliografía recomendada, incluyendo la página oficial de *Dockers*.

```
# docker run --volumes-from webData --name webNodo2 --net=velascoNet -ti velasco/debian:v1 /bin/bash
```

Comprobamos que el volumen está montado en ambos contenedores:

```
# docker inspect webNodo1
        "Mounts": [
            {
                "Name": "67e828fc9d43131890fb35bb0f0b065cebea79b658ff0c1a80d18ee1db71d079",
                "Source":
"/var/lib/docker/volumes/67e828fc9d43131890fb35bb0f0b065cebea79b658ff0c1a80d18ee1db71d079/_data",
                "Destination": "/www",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],

# docker inspect webNodo2
        "Mounts": [
            {
                "Name": "67e828fc9d43131890fb35bb0f0b065cebea79b658ff0c1a80d18ee1db71d079",
                "Source":
"/var/lib/docker/volumes/67e828fc9d43131890fb35bb0f0b065cebea79b658ff0c1a80d18ee1db71d079/_data",
                "Destination": "/www",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
```

Ahora vamos a asegurarnos de que el volumen está siendo compartido realmente entre los contenedores. Para ello desde un contenedor creo un fichero en esta carpeta:

```
root@100ad81a0c3a:/www# echo generando un fichero desde contenedor webData> test1.txt
root@100ad81a0c3a:/www# ls
test.txt
```

Y en otro contenedor compruebo que el fichero está ahí. Es importante resaltar que aunque estoy usando la misma imagen para crear los contenedores éstos no están compartiendo más que esa carpeta ya que son instancias distintas de una misma imagen. Es decir, que si hago algún cambio en alguno de los contenedores fuera de este volumen el cambio no se verá reflejado en los contenedores que están corriendo paralelamente. Por otra parte puedo salir de los contenedores, pararlos o crear nuevos pero el volumen seguirá siendo compartido entre todos. Si salgo de todos los contenedores a la vez o incluso si reinicio el host, el volumen sigue existiendo con todos los datos por lo que puedo crear contenedores con este volumen montado y acceder a él.

```
# docker rm webNodo1
# docker rm webNodo2
# docker ps -a
CONTAINER ID        IMAGE               COMMAND         CREATED           STATUS     PORTS     NAMES
2f6f2b2c759e        velasco/debian:v1   "/bin/bash"     3 minutes ago     Created              webData

# docker run --volumes-from webData --name webNodo1 --net=velascoNet -ti velasco/debian:v1 /bin/bash
root@f5dae1eeb69f:/# cd /www
root@f5dae1eeb69f:/www# ls
test1.txt  test2.txt
```

Pero si elimino el contenedor inicial al que se asocia este volumen, a pesar de que el volumen no se borra no podré crear contenedores que accedan a este volumen a través de (`--volumes-from`) ya que no existe.

Los volúmenes compartidos con la opción -v son directorios que se encuentran en el host. Esto es útil para crear páginas web compartidas y persistentes entre todos los contenedores como en nuestra prueba de concepto. Por tanto además de crear ficheros desde los distintos contenedores podemos crearlos desde el propio *host* y compartirlos:

```
# docker volume ls
DRIVER              VOLUME NAME
local               720fdee917102332bdb00ad3f4e79263e52129b49e2b5c31b1bba97e3d3fce22
# ls -lah
/var/lib/docker/volumes/720fdee917102332bdb00ad3f4e79263e52129b49e2b5c31b1bba97e3d3fce22/_data
total 20K
drwxr-xr-x 2 root root 4.0K Mar 11 09:26 .
drwxr-xr-x 3 root root 4.0K Mar 11 08:48 ..
-rw-r--r-- 1 root root   47 Mar 11 08:49 test1.txt
-rw-r--r-- 1 root root   47 Mar 11 08:50 test2.txt
```

Orquestación[56]

Una vez hemos creado nuestras aplicaciones para ofrecer distintos servicios podemos encontrarnos con un entorno complejo de gestionar con decenas de imágenes, volúmenes de datos y contenedores a los cuales queremos balancear el tráfico. Incluso puede ser posible que tengamos varios *hosts* con contenedores que forman parte del mismo servicio. Por ello necesitamos poder gestionar todo esto de una manera más cómoda y hasta cierto punto automatizada que nos permita no tener que preocuparnos por ejemplo de si los contenedores necesarios para ofrecer un servicio están funcionando o alguno se encuentra parado por alguna razón. Como se ha comentado en el apartado "Hacia la nube" esto es lo que llamamos orquestación.

Existen varias herramientas para llevar a cabo esta labor dentro de la virtualización a través de *Dockers*. En nuestra prueba de concepto utilizaremos *Kubernetes* ya que es la creada por Google y la más extendida. Este orquestador permite la definición de grupos de contenedores llamados *pods* donde le indicamos la cantidad de contenedores que deben estar levantados y funcionando y es la propia herramienta la que se encarga de monitorizarlos, balancear el tráfico y levantar nuevos contenedores en caso necesario, todo ello sin intervención humana. Por tanto la medida mínima de provisión para *Kubernetes* es el *pod*.

Kubernetes trabaja de manera jerárquica como un árbol donde el origen de todas las ramas es el servicio que se presta al usuario. Cada una de las hojas se monitoriza y se actúa en consecuencia hasta llegar a las hojas finales que son los contenedores. En este sentido la figura siguiente muestra las capas existentes:

Figura 19 – *Kubernetes: Esquema básico*
La figura muestra las partes básicas en las que Kubernetes separa los servicios

El *pod* incluye los contenedores, los conjuntos de *pods* a su vez son monitorizados por el replicador y finalmente las réplicas son controladas por el servicio que se encarga de balancear el tráfico entre las mismas. Existen distintos tipos de reparto de carga entre las distintas réplicas por parte del servicio aunque el más común es el servicio balanceador de carga que envía las peticiones en modo *round robin*[57] entre ellas. De manera más estructurada por tanto nos encontramos con el esquema siguiente:

[56] Se han consultado varias fuentes de Internet además de [DCBaDwK2016] y [DOaM] durante la prueba de concepto.
[57] Una petición se envía a cada réplica y cuando no hay más se vuelve al primero y se realiza la misma acción.

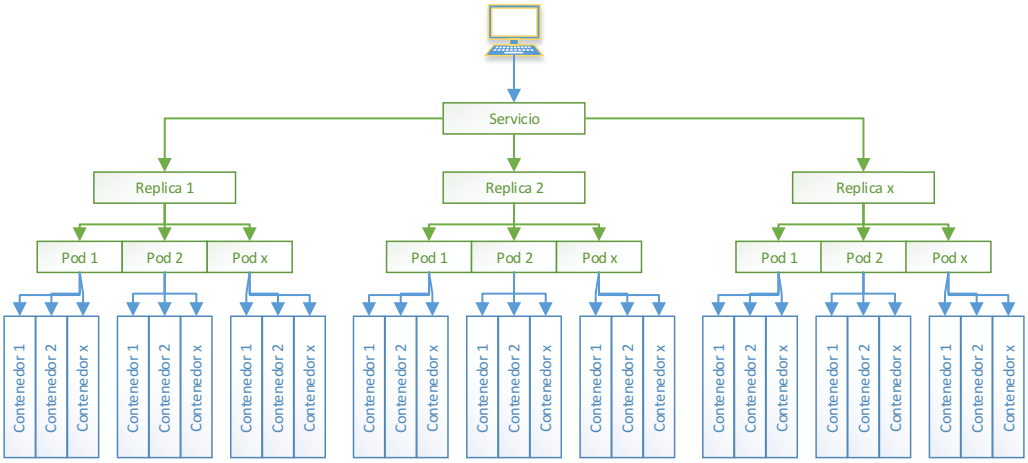

Figura 20 – *Kubernetes: Esquema completo para un host*
La figura muestra un esquema ejemplo de un servicio con diferentes réplicas

Como se puede comprobar el uso de *Kubernetes* resulta útil cuando se tiene una gran cantidad de contenedores que gestionar, es decir, un servicio que requiera de gran potencia. En principio el envío de tráfico a las distintas réplicas por parte del *servicio* es un proceso más complejo que requiere de *software* de terceros. Concretamente se requiere de un *software* que ofrezca un servicio clave-valor distribuido. Este tipo de programa ofrece servicios centralizados para sistemas distribuidos que proporcionan la capacidad de coordinar el *cluster* de réplicas y gestionar su estado. Esto lo hace almacenando y replicando la información entre elementos que componen la arquitectura de *Kubernetes*. Por ejemplo si tenemos un servicio de tipo balanceador de carga que se compone de 3 réplicas como en el esquema superior, las réplicas indicarán el número de peticiones de clientes que les han llegado y han gestionado, de esta manera el servicio puede hacer un balanceo de carga equitativo. De la misma manera permite la recuperación ante fallos.

En realidad los servidores de clave valor fuera del entorno *Docker* se usan también para otras labores como la de inventario en red, guardado de configuraciones dinámicas, coordinación de servicios,… En nuestra prueba de concepto utilizaremos *Etcd* ya que se integra de manera nativa con *Kubernetes*[58].

Instalamos los paquetes necesarios en el caso de que no estén ya instalados.

```
# apt-get update
# apt-get upgrade
# apt-get install make
# apt-get install curl
# apt-get install gcc
```

Instalamos *etcd*[59], en este caso ya no se puede hacer a través de `apt-get`.

```
# mkdir /home/<usuario>/software
# cd /home/<usuario>/software
# curl -L  https://github.com/coreos/etcd/releases/download/v2.3.0-alpha.1/etcd-v2.3.0-alpha.1-linux-amd64.tar.gz -o etcd-v2.3.0-alpha.1-linux-amd64.tar.gz
# tar xzvf etcd-v2.3.0-alpha.1-linux-amd64.tar.gz
# cd etcd-v2.3.0-alpha.1-linux-amd64
```

Probamos que funciona. Para ello lo ejecutamos.

```
#./etcd
```

[58] Otro software de este tipo bastante utilizado es Consul: https://www.consul.io/intro/getting-started/agent.html. Para una breve descripción de Etcd se puede consultar el siguiente artículo: http://thenewstack.io/about-etcd-the-distributed-key-value-store-used-for-kubernetes-googles-cluster-container-manager/

[59] Fuente: https://github.com/coreos/etcd/releases

Desde otro terminal generamos una llave y comprobamos que existe:

```
# /home/<usuario>/etcd-v2.3.0-alpha.1-linux-amd64/etcdctl set mykey "esta es mi llave"
esta es mi llave
# /home/<usuario>/etcd-v2.3.0-alpha.1-linux-amd64/etcdctl get mykey
esta es mi llave
```

Paramos el proceso *etcd* (ejecutando Ctrl+C) ya que todavía no hemos instalado *Kubernetes*, será éste quien lo arranque posteriormente como parte de su proceso de arranque. *Kubernetes* lo buscará en `/usr/local/bin` así que copiaremos el ejecutable a esta carpeta.
Posteriormente instalaremos la última versión *Kubernetes* utilizando en este caso la herramienta de actualización y revisión de versiones de software *git* en lugar de *curl*.

```
# cd /home/<usuario>/software
# git clone https://github.com/kubernetes/kubernetes.git
# cd kubernetes
# make release
```

Si todo acaba sin ningún error veremos los siguientes mensajes y podremos comenzar a usar *Kubernetes*:

```
[...]
+++ [0317 07:38:02] Docker builds done
+++ [0317 07:38:53] Building tarball: full
+++ [0317 07:38:53] Building tarball: test
```

Para poder usar *Kubernetes* deberemos tener en cuenta algunos puntos. Primero, necesitaremos tener instalado el lenguaje *Go*, ya que *Etcd* está programado con este lenguaje y *Kubernetes* lo utiliza cuando lanza ciertos scripts. De lo contrario obtendremos un error durante la instalación que nos alertará de esta circunstancia.

```
Can't find 'go' in PATH, please fix and retry.See http://golang.org/doc/install for installation instructions.
```

Por otra parte la versión de *Go* debe ser superior o igual a la 1.4, lo que nos obliga a instalar el paquete manualmente ya que esta versión no es instalada por defecto a través de `apt-get`:[60]

```
# apt-get install golang
```

Al instalar *Kubernetes* podremos ver este mensaje.

```
Detected go version: go version go1.3.3 linux/amd64.Kubernetes requires go version 1.4 or greater.Please install Go version 1.4 or later.
```

En cualquier caso la instalación es fácil, únicamente deberemos bajar la última versión, desempaquetarla en `/usr/local/bin` y finalmente añadir al PATH la localización de sus binarios:

```
# curl –OL https://storage.googleapis.com/golang/go1.6.linux-amd64.tar.gz
# export PATH=$PATH:/usr/local/go/bin
```

Además necesitaremos utilizar el comando `sudo`, a no ser que queramos modificar manualmente los *scripts*, por lo que debemos asegurarnos de que existe en nuestra distribución o instalarlo específicamente:

```
# apt-get install sudo
```

Arrancamos *Kubernetes* a través de un *script* que nos levantará todos los demonios necesarios para trabajar con *Kubernetes*:

[60] Si ya tenemos instalado esta versión a través de `apt-get` deberemos eliminar dicha instalación y eliminar todos los ficheros que queden dentro del sistema.

```
# hack/local-up-cluster.sh
```

Si algo falla más allá de estos errores básicos se puede averiguar el problema analizando la salida y los *logs* en la carpeta *tmp* o consultar los errores básicos en *Github*[61].

```
  /tmp/kube-apiserver.log
  /tmp/kube-controller-manager.log
  /tmp/kube-proxy.log
  /tmp/kube-scheduler.log
  /tmp/kubelet.log
```

A partir de aquí se ejecutan los siguientes comandos (que los pedirá de hecho la propia instalación):

```
# ./kubectl.sh config set-cluster local --server=http://127.0.0.1:8080 --insecure-skip-tls-verify=true
cluster "local" set.
# ./kubectl.sh config set-context local --cluster=local
context "local" set.
# ./kubectl.sh config use-context local
switched to context "local".
```

Y ya se puede comenzar a gestionar *Kubernetes* con el comando `kubectl.sh`. Comprobamos el estado del orquestador:

```
# ./kubectl.sh cluster-info
Kubernetes master is running at http://127.0.0.1:8080

# ./kubectl.sh config view
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: http://127.0.0.1:8080
  name: local
contexts:
- context:
    cluster: local
    user: ""
  name: local
current-context: local
kind: Config
preferences: {}
users: []
```

Igual que *Dockers*, los posibles comandos a ejecutar dentro de *Kubernetes* son muy explicativos, en cualquier caso se puede encontrar más información en la página oficial[62].

El proceso básico de creación sigue la jerarquía de los diagramas mostrados más arriba:

1. Creo un *pod*, por defecto se le asigna un único contenedor.
2. Le asigno el número mínimo de nodos/contenedores.
3. Creo un controlador de réplicas (RC: *Replication Controller*) para controlar el estado de los contenedores.
4. Lo convierto en servicio para que sea accesible y se balancee entre los contenedores del *pod*
5. Gestiono la provisión del servicio, por ejemplo añado replicas (*pods*) del mismo que el propio *Kubernetes* se ocupará de gestionar[63]

Los objetos se pueden crear de dos formas, a través de comandos, en cuyo caso se crean con los valores por defecto, o definiendo los datos de los mismos de manera más específica en un fichero (de extensión *.yam* o

[61] https://github.com/kubernetes/kubernetes/wiki/Debugging-FAQ
[62] http://kubernetes.io/
[63] *Replication-controller* se encarga de monitorizar los *pods*, es decir, si falla un *pod* arranca otros para mantener el número indicado. Se ejecuta automáticamente el usar el comando `kubectl.sh run`.

.json) y creando el objeto a partir de él. En esta prueba de concepto lo haremos de ambas formas. Para más información revise la página oficial donde se explica la sintaxis y el formato en ambos casos.

Creamos un *pod* básico:

```
# ./kubectl.sh run my-nginx --image=nginx --replicas=2 --port=80
# ./kubectl.sh get pod
NAME                        READY     STATUS    RESTARTS   AGE
my-nginx-3800858182-c4zwh   0/1       Pending   0          14s
my-nginx-3800858182-ychiy   0/1       Pending   0          14s
```

Para comprobar los pasos que ha realizado *Kubernetes* para crear este *pod* usamos `./kubectl.sh get events`
Por defecto sólo se puede acceder a los *pod* de manera interna a través de la red virtual que se configura automáticamente cuando creamos los contenedores. Es decir, usuarios externos no podrán acceder a los servicios que prestan hasta que no configuremos el *pod* como un servicio. Por ello si consultamos los servicios que en este momento está gestionando *Kubernetes* sólo aparecerá él mismo:

```
# ./kubectl.sh get services
NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   3h
```

Esto es debido a que no se ha creado ningún *rc*:

```
# ./kubectl.sh get rc
# ./kubectl.sh delete rc my-nginx
Error from server: replicationcontrollers "my-nginx" not found
```

Lo creo a través de un fichero *yaml*:

```
javExampleRC.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: javi-nginx
        image: nginx
        ports:
        - containerPort: 80
```

Creo el objeto a partir del fichero anterior:

```
root@debian:/home/javs/kubernetes/cluster# ./kubectl.sh create -f ../examples/javExampleRC.yaml
replicationcontroller "nginx" created

root@debian:/home/javs/kubernetes/cluster# ./kubectl.sh describe replicationcontrollers/nginx
Name:           nginx
Namespace:      default
Image(s):       nginx
Selector:       app=nginx
Labels:         app=nginx
Replicas:       2 current / 2 desired
Pods Status:    0 Running / 2 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen      LastSeen          Count    From                                    SubobjectPath    Type
Reason                  Message
```

```
  ---------      --------         -----    ----                                    -------------    --------
------                    -------
  1m              1m                1        {replication-controller }                          Normal
SuccessfulCreate        Created pod: nginx-dj3bi
  1m              1m                1        {replication-controller }                          Normal
SuccessfulCreate        Created pod: nginx-u2gf2
```

Creamos un *pod* básico:

```
# ./kubectl.sh run my-nginx --image=nginx --replicas=2 --port=80
# ./kubectl.sh get pod
NAME                        READY     STATUS    RESTARTS   AGE
my-nginx-3800858182-c4zwh   0/1       Pending   0          14s
my-nginx-3800858182-ychiy   0/1       Pending   0          14s
```

Posibles problemas

Error 1

Posibles problemas al crear un nodo nos muestra el *pod* como pendiente:

```
./kubectl.sh get pod
NAME                            READY     STATUS    RESTARTS   AGE
debian-kuber-1552179373-djzcy   0/1       Pending   0          50s
hello-node-544049829-sbj1j      0/1       Pending   0          55m
```

Podemos ver más detalles con `./kubectl.sh describe pod debian-kuber`

Error 2

Si obtenemos el siguiente error es que la hora está mal puesta en el *host*:

```
I0320 10:46:23.899150     13138 nodecontroller.go:472] Evicting pods on node 127.0.0.1: 2016-03-20
10:46:23.899126811 +0000 GMT is later than 2016-03-18 02:37:51.59229358 +0000 GMT + 5m0s
```

Error 3

El siguiente error se puede ignorar, en realidad el servicio corre correctamente, problema de *Kubernetes*.

```
E0320 14:21:08.193209   32831 controllermanager.go:216] Failed to start service controller:
ServiceController should not be run without a cloudprovider.
```

Error 4

Vemos la siguiente información en `/tmp/kube-controller-manager.log` cada segundo:

```
I0320 10:50:10.780468   15424 kubelet.go:2365] skipping pod synchronization - [Failed to start
ContainerManager system validation failed - Following Cgroup subsystem not mounted: [memory]]
```

Kubernetes necesita tener montado `cgroups` para poder utilizar la capacidad de limitar los recursos usados por los nodos. Podemos ver estos recursos montados uno a uno ejecutando `mount`. Si alguno no está montado `controllermanager` fallará y no podrá ser arrancado. En este caso vemos que tenemos un problema al montar uno de ellos, `memory`, ya que nos los indica el log, y además no encontramos la siguiente línea al ejecutar `mount`:

```
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
```

Si analizamos la línea concreta de código que lanza ese error vemos:

```
# less /home/javs/kubernetes/pkg/kubelet/kubelet.go
```

```
// state every sync-frequency seconds. Never returns.
func (kl *Kubelet) syncLoop(updates <-chan kubetypes.PodUpdate, handler SyncHandler) {
        glog.Info("Starting kubelet main sync loop.")
        // The resyncTicker wakes up kubelet to checks if there are any pod workers
        // that need to be sync'd. A one-second period is sufficient because the
        // sync interval is defaulted to 10s.
        syncTicker := time.NewTicker(time.Second)
        housekeepingTicker := time.NewTicker(housekeepingPeriod)
        plegCh := kl.pleg.Watch()
        for {
                if rs := kl.runtimeState.errors(); len(rs) != 0 {
                        glog.Infof("skipping pod synchronization - %v", rs)
                        time.Sleep(5 * time.Second)
                        continue
                }
                if !kl.syncLoopIteration(updates, handler, syncTicker.C, housekeepingTicker.C, plegCh)
{
                        break
                }
        }
}
```

Podemos a su vez ver el error al comprobar la descripción del nodo:

```
kubectl.sh describe no

Conditions:
KubeletNotReady                 Failed to start ContainerManager system validation failed - Following
Cgroup subsystem not mounted: [memory]

Events:
KubeletSetupFailed      Failed to start ContainerManager system validation failed - Following Cgroup
subsystem not mounted: [memory]
```

Si creamos un *pod* veremos que tenemos un error:

```
kubectl.sh describe pod
Events:
FailedScheduling        no nodes available to schedule pods
```

El problema es que estoy usando *Debian*, y esta distribución de *Linux* desactiva por defecto el subsistema de memoria en el *kernel* por lo que necesitamos activarlo. Para ello simplemente editamos `/etc/default/grub` y modificamos la siguiente línea:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet cgroup_enable=memory"
```

Ejecutamos `update-grub` y reiniciamos. Ahora ya podemos crear nuestros *pods*. Comprobamos que si los detenemos y eliminamos seguimos teniendo siempre dos nodos dentro del *pod* ya que *Kubernetes* se encargará de monitorizarlo.

Análisis de las distintas opciones

Tras realizar un repaso de todos los tipos de servicios que se prestan en la nube y la arquitectura utilizada para prestarlos se puede hacer un pequeño resumen en relación a las distintas opciones existentes a la hora de implementar una nube haciendo hincapié en el destinatario de este servicio:

- Empresa
    - Los recursos, de cualquier tipo[64], se encuentran en localizaciones externas a la empresa que contrata el servicio de nube y están totalmente deslocalizados.
    - Los recursos, de cualquier tipo, se encuentran en localizaciones externas a la empresa que contrata el servicio de nube y están localizados en el país de la empresa que los contrata.
    - Los recursos, de cualquier tipo, se encuentran en localizaciones de la propia empresa. Es decir, empresas que deciden montar su propia nube utilizando todos sus centros de datos. Un buen ejemplo de esto es *Openstack*.

- Desarrollador
    - Los recursos, de cualquier tipo, se encuentran en localizaciones ajenas al desarrollador o pequeña empresa que contrata el servicio de nube y están totalmente deslocalizados.
    - Los recursos, de cualquier tipo, se encuentran en localizaciones de los propios desarrolladores o pequeña empresa. Es decir, personas con conocimiento técnico que deciden montar su propia nube, o contratan un servicio de consultoría para que lleve el proyecto a cabo. Un perfecto ejemplo sería precisamente *Docker* y *Kubernetes*.

- Usuario
  En este caso la única posibilidad real de nube es de pago, donde la localización de los datos se sitúa en casa del proveedor. No existe opción gratuita aprovechando los propios recursos existentes en casa del usuario. De hecho se puede decir que a nivel gratuito lo más cerca de este servicio que se ha encontrado son soluciones de *software* colaborativo o de compartición donde los recursos se encuentran situados allí donde se instala la aplicación.

    - Los recursos, de cualquier tipo, se encuentran en localizaciones ajenas al usuario que contrata el servicio de nube y están totalmente deslocalizados. Un ejemplo sería el servicio prestado por la compañía *Arsys*[65].
    - Los recursos son comprados a una empresa, caso de NAS, y se instalan en casa del usuario. A cambio de una mensualidad los datos de esa NAS pueden replicarse en localizaciones de la empresa, lo que *acerca* un poco la idea de nube al usuario. Este servicio está limitado al almacenamiento y compartición de datos.
    - Los recursos pertenecen al propio usuario. El *software* ofrece gran cantidad de aplicaciones relacionadas con la colaboración (agendas, correo, *chat*, *tagging*,…) y la compartición de ficheros. Un buen ejemplo de esto es el ya comentado *Owncloud* donde la aplicación se convierte en punto de acceso para otros usuarios que quieren acceder a los distintos recursos en todos sus sabores.

En el caso de *Arsys* su nube consta de tres centros de proceso de datos en España y un nodo de nube en Estados Unidos. Es importante hacer hincapié en que no todos los servicios prestados por proveedores de servicios cumplen en realidad con los estándares de nube aunque así lo proclamen, por ello si se desea contratar es importante comprobar estos aspectos. En algunos casos se trata de compañías que se han dedicado tradicionalmente a ofrecer servicios de *hosting* las cuales con el estallido de la nube han incluido en la propaganda sobre sus servicios la palabra nube simplemente por utilizar virtualización dentro de su centro de proceso de datos para dar su servicio.
Por otro lado tenemos aplicaciones de compartición de ficheros que se venden bajo el sobrenombre de nube. En este caso nos encontramos con servicios que en realidad no son nubes ya que, o bien todos los datos se

[64] Con la expresión 'cualquier tipo' se engloban los comentados en la sección 'Tipos de Nube' como SaaS, IaaS, PaaS, etc.
[65] Se incluye esta empresa principalmente por ser española (Logroño), concretamente una empresa subsidiaria de *United Internet Group*. *Web* oficial: https://www.arsys.es/servidor-cloud

encuentran centralizados en un dispositivo como en caso de los servicios de NAS, o si están repartidos en distintas localizaciones no lo están gracias al *software* de nube sino a que el propio usuario ya tiene sus datos repartidos por varios lugares como en el caso de *Owncloud*. Por tanto en este último caso lo que se ofrece es un servicio de colaboración y compartición de ficheros que se 'desentiende' de donde puedan encontrarse estos, simplemente accede a ellos. Además los servicios están limitados a los ofertados e implementados en el propio software por tanto, almacenamiento y compartición de ficheros en el caso de NAS, y mejorando muchísimo este punto, almacenamiento, compartición y herramientas colaborativas en el caso de *Owncloud*.

Por último, habiendo nombrado el servicio ofertado a través de NAS no sería justo dejar de comentar en este punto que existen innumerables aplicaciones que permiten acceder a nuestros datos sin coste alguno. De hecho incluso se pueden compartir carpetas con ficheros de video a los que se puede acceder a través de televisores inteligentes para poder ser visionados. La única diferencia apreciable en estos casos es que es el propio usuario el que debe dar de alta su dispositivo en un servicio de nombres dinámico para poder ser accesible desde Internet, algo no excesivamente complejo por otra parte.

Análisis del sistema Dockers

A continuación se va a realizar un análisis del sistema *Docker* tras el cual se concluirá si nos es útil para la finalidad que se propone en este trabajo. Finalmente tras tener todos los datos sobre la mesa se acabará con un intento de incursión en el futuro de las nubes personales, ¿le servirá al ciudadano de a pie la nube para independizarse un poco de las grandes empresas y convertirse en productor sin tener que pagar para ello una tarifa a un tercero como hasta ahora?

Descripción

Como se ha visto *Docker* es una arquitectura de virtualización basada en contenedores (LXC[66]), lo que en pocas palabras tiene dos importantes consecuencias. La primera es que su implementación no recae sobre máquinas virtuales. Y la segunda es que al no existir éstas no se requiere para su instalación de una compleja, y en muchos casos cara, capa *hypervisor* para controlar y monitorizar su estado y disponibilidad, ya sea la implementación local o repartida entre diferentes localizaciones[67]. Muy al contrario *Docker/Kubernetes* crea una arquitectura totalmente gratuita, ligera y de rápida instalación.

Tras realizar la prueba de concepto una de las más importantes realidades que han salido a la luz ha sido su excepcional rendimiento. Esto es debido a que en realidad el concepto de contenedor es más simple que el de máquina virtual y también lo es su implementación, por tanto no requiere de tantos recursos. En el fondo este sistema lo único que hace es aprovechar la capacidad *control groups* (*cgroups*) de *Linux* para aislar, monitorizar, priorizar, controlar y limitar el uso de los recursos del sistema a los servicios. Para ello *cgroups* hace uso de diferentes controladores o subsistemas que gestionan cada tipo de recurso, así como de otras herramientas de *Linux* como *namespace* para aislar grupos de servicios los cuales son separados de otros grupos y de esta manera se impide que cada grupo vea los recursos de otros.

Por tanto de manera simplificada un contenedor no es más que un proceso aislado del resto al que se le asignan una cantidad concreta de recursos y que no puede ver o acceder a recursos de otro proceso. Siguiendo este paradigma el usuario levanta tantos procesos como quiera, uno por contenedor, y les asigna los recursos deseados. Este proceso es el *servicio* a prestar que a diferencia de una máquina virtual es único y no requiere de un sistema operativo propio con sus ficheros y demonios asociados. De ahí la gran velocidad a la hora de levantarlos, tirarlos, configurarlos e incluso monitorizarlos y repartirlos por diferentes localizaciones. Es un concepto de virtualización en su estado más básico y esencial.

Una importante limitación es la dependencia de los contenedores con el sistema operativo que corre por debajo, único a todos ellos, lo que limita la creación de contenedores a imágenes específicas compatibles con ese sistema operativo. Si se requiere levantar aplicaciones de otro sistema se deberá instalar éste y sobre él, *Docker*. Un ejemplo podría ser la instalación de *Docker* sobre una máquina que corra *Windows*, otra que corra *Debian Sid* y otra que corra *Debian Jessie* levantando en cada una contenedores con servicios específicos de cada plataforma, mientras que otra opción podría consistir en levantar máquinas virtuales con diferentes versiones de sistemas operativos y sobre ellas los contenedores. Las combinaciones dependerán de las necesidades aunque para aprovechar el gran rendimiento de *Docker* la primera opción sería la correcta.

Implementación de Dockers

Una característica importante de este sistema es la simplicidad de la instalación la cual básicamente se lleva a cabo a través de cuatro comandos. Una vez ejecutados el sistema está listo para ser utilizado a través de un conjunto completo pero sencillo de comandos con sus argumentos que por sí solos son capaces de llevar a cabo todas las acciones que permite *Docker*.

---

[66] De su nombre inglés *Linux ConTainers*.

[67] Como ejemplos de *hypervisores* de pago tenemos *VMWare ESXi* o *Microsoft Hyper-V*, en distribución gratuita (*Open Source*) tenemos *KVM* o *Xen* basados en *Linux*.

Gracias a esto la creación de gestores para la administrar *Docker* no es compleja, en su versión más simple lo único necesario que se debe implementar es un cuadro de mandos *web* de manejo intuitivo que se encargue de ejecutar por debajo los comandos necesarios siguiendo las normas establecidas, y de mostrar los ficheros de log de manera comprensible para el usuario. Un buen ejemplo en desarrollo es *Shipyard*.

No ocurre lo mismo con el orquestador *Kubernetes*. En este caso nos encontramos ante un sistema más complejo pensado para grandes despliegues donde se deben controlar múltiples localizaciones y cuya instalación no es tan simple. De hecho a lo largo de la prueba de concepto se han debido resolver algunos problemas no documentados en la versión escogida.

Posibilidades

Debido a que en gran medida se analiza en la conclusión de este trabajo la utilidad real de este sistema a la hora de ser aplicado a nubes personales, en este apartado sólo cabe comentar que teniendo en cuenta las necesidades requeridas para una nube personal y los comentarios del apartado anterior, se puede deducir que *Docker/Kubernetes* no parece constituir por sí sólo un motor de virtualización apto para los usuarios finales con bajo conocimiento técnico.

Conclusión

A lo largo de todo el trabajo se ha intentado guiar al lector paso a paso evitando profundizar en temas que requiriesen un alto nivel de conocimiento técnico. La única parte realmente técnica del mismo, la prueba de concepto, se ha planteado de una manera simple y con la vista puesta en todo momento en la explicación llana y la complementación con lo que se ha tratado en el trabajo. En la última parte del proyecto se han esbozado claramente las capacidades de todos los sistemas que se han considerado relacionados con el fin del proyecto para finalmente desembocar en una conclusión clara y concisa. Por tanto ante la pregunta, ¿existe en la actualidad algún servicio de nube real compatible con un usuario personal tal y como se definió en la sección del mismo nombre? La respuesta es claramente que no.

Los auténticos servicios de nube están diseñados para empresas o usuarios con suficiente conocimiento técnico como para poder desplegarlas, y las ofertas dirigidas al consumidor final en su hogar no se podrían considerar como nubes atendiendo a todo lo que se ha visto hasta el momento.

Pero lo que sí es cierto es que la opción *Docker* analizada en este documento cumple con algunas de las premisas más importantes, es gratuita y observa técnicamente la definición ofrecida para la nube personal. Aunque tiene dos grandes inconvenientes, el primero es que su implementación no es trivial si queremos que los recursos se encuentren repartidos en varias localizaciones, como de hecho se requiere. Y la segunda es que aunque proporciona el nivel necesario de facilidad y simplicidad a la hora de poner en funcionamiento una aplicación y existen cientos de imágenes diferentes para prestar diferentes servicios, estas están orientadas básicamente al desarrollo.

No se puede ser idealista y pensar en un servicio de nube para que un usuario común desde su casa pueda implementarlo sin conocimiento básico de lo que está haciendo, pero en este momento no existe un término medio. Por tanto podemos concluir que actualmente no nos encontramos en lo que a lo largo del trabajo se ha dado en llamar Internet 2.0. A día de hoy el rol de productor no ha llegado a casa, al menos ayudado por la arquitectura de nube.

El futuro

En cuanto al futuro, en mi modesta opinión es probable que a corto plazo, digamos tres años vista, el panorama en el ámbito de la nube personal no cambie excesivamente, al menos en cuanto al desarrollo de soluciones gratuitas. Actualmente el desarrollo de la nube, así como en general cualquier tipo de arquitectura distribuida, fluye a lo largo de dos vertientes principales, la empresa y la investigación universitaria. Desde esta perspectiva es por tanto comprensible que las nuevas arquitecturas de comunicaciones y sistemas no hayan puesto sus ojos en el mundo de los usuarios personales.

En una primera instancia habitualmente el origen de los saltos evolutivos dentro de las nuevas tecnologías se ha encontrado en organizaciones sin ánimo de lucro como IETF[68], o en la universidad, aunque en este último caso muchas veces el dinero que promovía los proyectos de investigación viniese de fondos gubernamentales. Es por ello que las universidades eran las primeras en poner en práctica los nuevos progresos y los mejorasen hasta poder ser aplicados en el mundo empresarial. Por otra parte cualquier mejora tecnológica una vez contrastada y probada tiene como destino el lucro por lo cual son adoptadas, mejoradas y explotadas por las empresas.

En la figura siguiente se muestra como a grandes rasgos las necesidades que requieren ser cubiertas en estos dos entornos a nivel de sistemas distribuidos son las aplicaciones colaborativas de investigación, documentación de cualquier tipo o seguimiento de proyectos, las aplicaciones enfocadas al desarrollo que utilizan a su vez marcos de colaboración específicos para los desarrolladores o los sistemas que dan servicio a terceros, desde una aplicación web dinámica con su base de datos a un servicio de alquiler de recursos

[68] IETF (*Internet Engineering Task Force*) es una parte activa de ISOC (*Internet Society*) organización internacional sin ánimo de lucro fundada en 1992 que tiene como objetivos el contribuir a la ingeniería de Internet asegurando el desarrollo abierto. Una de sus tareas más importantes es la creación de estándares relacionados con Internet, la educación y la creación de políticas. Página oficial: http://www.internetsociety.org/

hardware sobre el cual el cliente monta su propio producto. Por tanto al ser estos los paradigmas empresariales, son estos mismos los elementos sobre los cuales se incide a la hora de desarrollar nuevas tecnologías que permitan un abaratamiento de costes y una mejora del servicio. En conclusión, el individuo como tal queda excluido de cualquier beneficio e incluso del propio conocimiento de que existan interesantes cambios en el mundo tecnológico, es decir aislado.

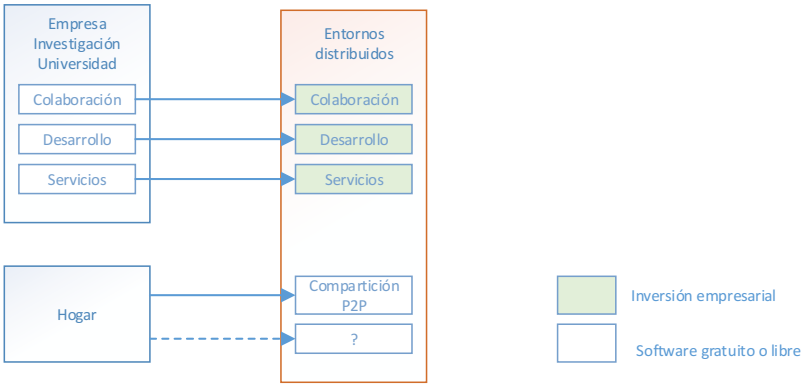

Figura 21 – *Objetivos de inversión*
La figura muestra cuales son los principales objetivos a la hora de invertir en el avance de tecnologías distribuidas. Vemos como dentro del ámbito del hogar la inversión es inexistente.

Existen tecnologías cuyo destino evidente es el uso personal como por ejemplo una consola de juegos o, hasta ahora, un ordenador, y por ello su comercio se enfoca hacia el individuo. Se requiere de un cambio de perspectiva para poder atisbar más allá de lo evidente y darse de cuenta de que muchas tecnologías son deformables en su uso y por tanto pueden aplicarse a distintas necesidades más allá de para la cual fueron creadas. Pongamos el ejemplo del computador personal cuyo origen puede considerarse el *mainframe* y posteriormente el mini computador en red. En cambio los primeros PC eran máquinas aisladas que incluían básicamente paquetes de ofimática, entornos básicos de programación y juegos, todo ello para uso personal. Incluso en el momento en el cual los PC comenzaron a conectarse a través de redes personales y posteriormente de Internet, a pesar de que se desarrollaron programas para poder compartir y colaborar entre distintos individuos, curiosamente todos estos programas colaborativos iban destinados al uso productivo, es decir, a empleados y no a individuos en sus hogares. En sus casas los PC se conectaban a Internet únicamente para recibir información de las páginas *web* o para relacionarse a través de chats, mientras que el resto de programas que tenían instalados estaban destinados a uso individual no colaborativo.

Es interesante observar como a lo largo de la historia evolutiva de los sistemas en red la inversión siempre ha relegado al usuario del hogar a un punto aislado a pesar de su capacidad de consumo en grupo, probablemente porque se le ha considerado de escaso potencial económico y cuya única utilidad apreciable en el mundo de las telecomunicaciones era la propaganda que se le podía mostrar en Internet. Esta desatención ha provocado probablemente la pérdida de millones de euros en negocios potenciales que nunca se han materializado y que han sido asumidos por desarrollos libres. Una muestra muy descriptiva de ello es el origen de las aplicaciones P2P[69], que de hecho personalmente consideraría como la primera gran revolución de Internet. En 1996 vio la luz una modesta aplicación *Mac* de compartición de archivos para empresas y universidades, *Hotline Communications*[70], que en breve evolucionó en decenas de nuevas aplicaciones gratuitas del mismo tipo y que revolucionaron el mundo de Internet golpeando duramente el modelo económico de los derechos de autor. Millones de personas comenzaron a utilizar este tipo de *software* y en cambio ninguna empresa vaticinó o siquiera pensó en el potencial de los usuarios como grupo para compartir datos, muy al contrario se les seguía, y sigue, considerando como individuos independientes con escasa capacidad colaborativa. Es por ello que actualmente los grandes cambios de paradigma a nivel de usuario vienen habitualmente promovidos por desarrolladores a través de proyectos de software libre que si tienen éxito son imitados y vendidos posteriormente por las empresas.

[69] Del inglés *Peer to Peer*, o como traducción más descriptiva, extremo a extremo.
[70] Cita: https://es.wikipedia.org/wiki/Historia_de_las_aplicaciones_P2P

En apariencia podría parecer que la inversión de las compañías en un producto concreto no ayuda a que ese producto vaya a favorecer al usuario que pretende obtenerlo gratis, pero esto no es tan cierto. En realidad depende entre otras cosas de la complejidad del producto, no es lo mismo la creación de un pequeño programa P2P que una mente inquieta puede crear sin excesiva inversión de tiempo, que el diseño de una arquitectura de nube.

La elaboración de un producto complejo consta de varias fases, dos de las más importantes (aunque no lo parezca) son la investigación y el desarrollo, I+D. Si las empresas invierten en la búsqueda de nuevos productos para el usuario del hogar, este verá aflorar ante él distintas posibilidades que irán mejorando en todos los sentidos a lo largo del tiempo ya que la propia compañía y sus competidoras se encargarán de mejorarlo. En una segunda fase la comunidad de desarrolladores de software libre prestará atención al producto y creará desarrollos gratuitos parecidos que a su vez irán mejorando y se convertirán en opciones plausibles para el usuario[71]. En cambio si no existe esta primera inversión en I+D por parte de las empresas debido a que no tienen visión suficiente como para ver en ello un nicho de mercado, la comunidad de programadores deberá realizar todo el trabajo desde cero, lo cual llevará mucho más tiempo.

Se podría alegar una objeción a esto, ¿qué interés podría tener una empresa en desarrollar un producto con todo el dinero que ello implica si luego se va a encontrar con *software* libre que realice la misma labor? La respuesta a esto es clara y tiene varias facetas. La primera es la más evidente, el primero que pisa el parqué del mercado es el que consigue los clientes los cuales posteriormente deberán ser convencidos de que otra opción es mejor y si así lo creen invertir, en tiempo o dinero, en el cambio[72,73]. Y la segunda no menos importante es el soporte que una empresa privada puede ofrecer, personalizado y con niveles establecidos de tiempos de respuesta. Mientras que una comunidad de desarrolladores que no cobra por él únicamente ofrecerá un soporte limitado, también conocido como *best effort*. Por tanto, sí, a las empresas les interesa ser las primeras en ofrecer un producto, y sí, a los usuarios les interesa que las empresas inviertan en soluciones a pesar de que muchos de ellos acaben optando por las versiones gratuitas.

Teniendo lo anterior en mente, en pocas palabras, para que la nube personal tenga un futuro se requiere una inversión. En un principio, como se ha comentado a lo largo del trabajo, desde sus inicios esta arquitectura ha sido orientada hacia las empresas, desde compañías grandes a pequeñas. Esto propicia el surgimiento de desarrollos gratuitos como *Openstack*[74], que consiste en un inmenso proyecto de colaboración que abarca todos los aspectos de la nube, básicamente desarrollado como *IaaS*, y que entre otros servicios soporta *Docker* y *Kubernetes*. Posteriormente, y el punto en el que nos encontramos en el momento de la creación de este documento, comienzan a emerger servicios de nube orientados a pequeños negocios y a desarrolladores que necesitan colaborar en proyectos. Un buen ejemplo de esto es el servicio ofrecido por *Arsys* al usuario.

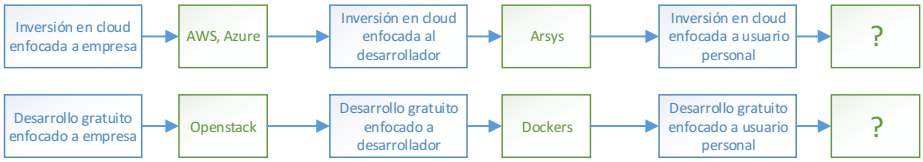

Figura 22 – *Comparativa en el desarrollo de software de pago y libre*
Esta figura muestra como hasta el momento no existen desarrollos realmente utilizables por usuarios personales ni de pago ni gratuito. Se incluyen algunos ejemplos en cada caso pero evidentemente se pueden encontrar muchos más en el mercado.

El diagrama anterior refleja otra conclusión importante y es que todos los servicios enfocados a una visión empresarial, es decir de inversión, se caracterizan porque la custodia de los datos se lleva a cabo por la

[71] Un gran ejemplo es *Open Office*.

[72] "*El porvenir pertenece a los innovadores*." André Gide

[73] De hecho empresas como Google invierten recursos en desarrollar software gratuito lo que les permite no sólo ver como es acogido en la comunidad y analizar su potencial sino también generar una necesidad y aprovechar en su beneficio las mejoras de la comunidad, el mejor ejemplo es *Dockers*.

[74] Proyecto original de *Rackspace* y NASA surgida en 2010. Por defecto utiliza como *hypervisor* KVM aunque puede utilizar cualquier otro. Existen varias distribuciones importantes como IBM, HP, Mirantis, RedHat o UbuntuStack. Página oficial: http://www.openstack.org/software/

compañía proveedora, así como la gestión de los servicios. Por otra parte desde la perspectiva del desarrollo gratuito se tiene la posibilidad de crear una nube propia con los datos y la gestión residiendo en casa del usuario, pero se requiere de un conocimiento técnico para su implementación que dista mucho de ser asumible por un usuario.

Una vez entendido esto, en mi opinión existen tres puntos que dificultan el futuro del desarrollo de la nube personal. El primero, como he comentado, tiene su origen en que no existe básicamente inversión por parte de las empresas destinadas a ella y todo lo que conlleva. Por otra parte derivado de este punto, el usuario no tiene constancia siquiera de las capacidades que le brindaría esta nueva tecnología por lo que en su mayor parte ni puede imaginar las ventajas que ésta pudiera ofrecerle, y por tanto no se crea una necesidad de producto que genere una demanda. Y finalmente el nivel de conocimiento del usuario es muy limitado actualmente y desarrollar a su medida requiere de más esfuerzo. Estas objeciones unidas conducen a un estancamiento, una vez más, en las arquitecturas de sistemas distribuidos desarrollados para el hogar. El usuario cuando se encuentra más allá de las fronteras de una empresa sigue siendo percibido como un individuo básicamente incomunicado.

Ahora bien, dicho todo esto y una vez analizadas las razones por las que la nube personal se encuentra en una realidad tan árida también hay cierto espacio para ser optimista de cara al futuro. Actualmente se aprecia como ciertas compañías están comenzando a poner la vista en la nube personal, o al menos orientada a pequeños proyectos, y ofrecen servicios idénticos a los que utilizan las grandes empresas incluyendo diferentes elecciones en cuanto a los recursos y sistemas operativos de nuestros servidores, orquestación para gestionar imágenes, balanceadores, copias de respaldo, seguridad, almacenamiento, etc. Este es un paso muy importante que puede servir de acicate para que las comunidades de desarrollo decidan invertir su tiempo en crear arquitecturas similares implementadas en el propio hogar. Hay que recordar que las necesidades dentro de este ámbito quedan muy reducidas en comparación con las arquitecturas de nube empresariales o de desarrollo y por lo tanto el esfuerzo a realizar es menor. De hecho gran parte de los requerimientos ya existen. A lo largo de este documento se ha analizado el sistema gratuito de contenedores *Docker*, esta es en definitiva toda la arquitectura que necesitaría un usuario personal para llevar a cabo una implementación de nube. En realidad la parte restante debería teóricamente ser la menos compleja y sería una orquestación pensada para un usuario con bajo nivel de conocimiento, y no complejos sistemas de orquestación como *Kubernetes*, totalmente orientado a grandes despliegues. En resumen un interfaz simple y amigable.

Desde mi punto de vista queda un pequeño empujón, no para tener grandes productos de nube personal pero sí al menos para comenzar a allanar el camino a esta tecnología dentro del hogar. Básicamente para ello restarían dos importantes acciones desde el punto de vista de *Docker*:

1. Un cuadro de mando que permita crear contenedores sincronizados rápidamente en distintas localizaciones de manera automática[75], así como una gestión sencilla de los mismos. Es decir, un orquestador a medida.
2. Un conjunto de imágenes sencillas consistentes en responder unas pocas preguntas por parte del usuario para permitir su despliegue y su posterior monitorización y control sin necesidad de atención por su parte. El surtido de imágenes debería incluir las necesidades actuales en las que más se está haciendo hincapié como el almacenamiento sincronizado entre diferentes dispositivos del cliente o la compartición de datos y video, incluyendo el video *streaming*, pero también opciones para la creación de sistemas de colaboración de todo tipo.

Aunque creo que se llegará a este punto en el futuro considero que todavía queda algún tiempo para ello y no lo veremos a corto plazo. Quedaría pendiente un estudio exhaustivo de todos los proyectos lanzados dentro de la comunidad para ver cual tiene más probabilidades de cumplir con todos los requisitos en un futuro. Entre ellos es muy interesante el trabajo que realiza *CoreOS*[76] a través de sus proyectos de código abierto para contenedores *Linux*.
Hasta que esto no ocurra no se crearán estándares que definan este tipo de nube, ni tan siquiera probablemente se hablará de la nube personal, o quizá *home cloud*. Es probable que ocurra lo mismo que con conceptos como *inter-cloud* comentados en este mismo trabajo, primero surgen necesidades que se intentan

[75] Quizá indicando a través de DNS dinámico la localización de cada PC del cliente donde se debe desplegar la aplicación.
[76] Página oficial https://coreos.com/

alcanzar como se puede ante la ausencia de productos, y a partir de la demanda un grupo adecuado de trabajo comienza su estandarización.

Sin duda creo que esta debe estar en manos de IETF que a su vez se ha encargado de definir, y básicamente *homologar*, todos los tipos de nube. Soy de la opinión de que ISOC debería liderar este tipo de acciones fuera de intervenciones gubernamentales o empresariales. Es importante hacer notar que la nube personal no debe tener requerimientos específicos de una u otra tecnología, ni siquiera de contenedores. En este trabajo se considera que la mejor opción es esta pero una estandarización de este tipo no debe entrar en el uso de una u otra opción sino únicamente realizar la definición formal, aunque siempre puede hacer recomendaciones. Por tanto, a pesar de que agencias como el NIST o grupos como *Cloud Native Computing Foundation*[77] realicen sus propios estándares e investigaciones, estas no serán definitivas en este sentido.
Por otra parte es necesario añadir que existen ya organismos que estandarizan distintas partes de las arquitecturas de nube[78] como por ejemplo:

- SNIA (*Storage Networking Association*) para el estándar de almacenamiento,
- OASIS para el estándar de empaquetamiento en la nube usando TOSCA
- DMTF (*Distributed Management Task Force*) para el estándar de empaquetamiento en la nube usando OVF (aunque OVF no se use sólo en la nube sino en general en la virtualización). Y dentro de DMTF:
    - CMWG (*Cloud Management Working Group*) para el estándar de CIMI o Interface de gestión de la infraestructura de nube
    - CADF (*Cloud Auditing Data Federation Working Group*) para el estándar de gobernanza y seguridad en la nube.

Aparte de esto es cierto que existe un vacío en cuanto a la creación de un estándar para definir el uso y capacidades de contenedores, por lo que se debe encontrar un 'responsable' para ello lo que permitiría que quien quisiera pudiese construir contenedores bajo un mismo conjunto de exigencias y por tanto de seguridad y estabilidad. Cosa que no ocurriría si esta estandarización se llevase a cabo por una empresa como Google. Probablemente uno de los mejores encaminados para esta labor es *CoreOS*.

Ahora sólo nos queda ver cómo se desarrolla el futuro.

[77] Este grupo pretende aglutinar las tecnologías de virtualización relacionada con contenedores y micro servicios. Página oficial https://cncf.io/ de forma similar a *CoreOS* pero en este caso sin entrar en proyectos de desarrollo.
[78] [CTC+] [CSAthtC]

Glosario

- Microservicio [MSFSA]: Una arquitectura de microservicio divide el software en muchos pequeños servicios que se comunican entre ellos a través de un API y que pueden ser desplegados de manera independiente. Cada equipo de trabajo se encarga de su propio microservicio y por tanto no depende de otros equipos lo que permite un escalar fácilmente cada proceso. Esto evita entre otras cosas que cada vez que se realiza un cambio en un servicio el resto se vean afectados. *Docker* es una arquitectura microservicio.

Dockers

- Contenedor: Entorno aislado y controlado donde corremos las aplicaciones. Por tanto el *contenedor* viene definido por un proceso que está siendo ejecutado, un demonio en terminología *Linux*. Esta ejecución se lleva a cabo a través del comando `docker run`.
- Docker: Marca comercial del concepto. También es el nombre del comando[79].

    - Docker Cloud: Permite desplegar y gestionar aplicaciones en modo *Docker* en una nube, de manera distribuida físicamente, o en la infraestructura existente en una empresa (*on-premise*). Básicamente es un REST API que ofrece, entre otras, opciones de provisión, despliegue, *logging* de una aplicación y añade la integración con otros componentes ofrecidos por *Docker* como el *Hub*
    - Docker Hub: Repositorio de imágenes públicas y privadas. Permite que un equipo de desarrollo trabaje sobre una imagen añadiendo capacidades de gestión de usuarios, flujos de trabajo, etc. También permite que cualquiera baje una imagen pública ya existente y la utilice en su entorno *Docker* o suba una suya y hacerla pública

- Dockeryzing: Proceso de integrar una aplicación existente en un *Docker*. Primero se construye la imagen de esa aplicación, es decir, se traduce esa aplicación al mundo *Docker* (se crea un fichero básicamente con los comandos a ejecutar y que serán llamados en bloque posteriormente). Después se corre un *contenedor* que contendrá esa aplicación.
- Imagen: Fichero que constituye la aplicación que no está corriendo. Aplicando una analogía de sistemas operativos la imagen sería la aplicación y el *contenedor* sería una instancia de esta aplicación en ejecución.
- Registro: Repositorio de imágenes. Las imágenes pueden guardarse en un repositorio público o en un repositorio privado. La ubicación de un repositorio privado evidentemente puede ser nuestro propio sistema, pero *Docker* también nos da la opción de guardar nuestras imágenes en modo privado en un registro situado en la nube, el *Docker Hub*.

Kubernetes

- Pod: Conjunto de contenedores que ejecutan un mismo servicio. *Kubernetes* crea *pods* y define la cantidad mínima de contenedores que deben estar corriendo de manera simultánea. Si por alguna razón un contenedor deja de ejecutarse el orquestador se encarga de levantar otro rápidamente que preste el servicio mínimo que se definió para eso *pod*.
- Réplica: Objeto que instancia esos *pods*. Al replicador se le indica cuantas veces se requiere que un *pod* este replicado y el mismo replicador se ocupa de levantarlos y monitorizarlos. Si una de las réplicas falla se encarga de eliminarla y levantar otra.
- Servicio: Hace accesibles por red a los *pods*. Un puerto el servicio puede ser tipo *load balancer*, y si contiene varias réplicas se ocupa de que el tráfico se balancee entre las réplicas usando *etcd*.

[79] Existen otros productos ofrecidos por *Docker* pero van más allá del ámbito de este trabajo ya que están dirigidos a las empresas.

Bibliografía

A continuación se muestra la bibliografía en orden alfabético:

- [CSAthtC] – Cloud Standards: Agreements that hold together clouds. Marvin Waschke. Ed. Apress. 2012
- [CTC+]: CompTIA Cloud+ CV0-001. Emmett Dulaney. LiveLessons.
- [DaMCI] – Deploying and managing cloud infrastructure. Salam, Gilani & Haq. Ed. Sibex. 2015
- [DC2Ed]: Audiosvisual: Docker Containers. Christopher Negus. LiveLessons. 2nd edition
- [DCBaDwK2016]: Docker Containers: Build and Deploy with Kubernetes, Flannel, Cockpit and Atomic. Christopher Negus. Ed. Prentice Hall. 2016
- [DOaM]: Audiosvisual: Docker Orchestration and MicroServices. Charles Crawford. LiveLessons.
- [eLOdG]: Audiovisual: El lado oculto de Google. Directora Sophie Roland. 2015 Upside Television
- [FoMN2016]: Foundations of Modern Networking SDN, NFV, QoE, IoT, and Cloud. William Stallings. Ed. Pearson., 2016
- [MSFSA] – Microservices Book, Flexible Software Architectures. Eberhard Wolff. GitHub Pages
- [SHAwLaRP]: Smart home automation with Linux and Rasperry Pi. Steven Goodwin. Ed. Apress.
- [tCaYS] – The cloud at your service: The when, how and why of enterprise cloud computing. Rosenberg & Mateos. Ed. Manning. 2011

Páginas *web* consultadas entre Marzo y Mayo de 2016 para realizar este trabajo:

http://www.apple.com/customer-letter/
https://www.arsys.es/servidor-cloud
https://cncf.io/
http://www.ciphercloud.com/blog/can-domestic-cloud-services-address-cloud-computing-risks
http://edition.cnn.com/2015/05/30/politics/what-happens-if-the-patriot-act-provisions-expire/index.html
https://www.consul.io/intro/getting-started/agent.html
https://coreos.com/
https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images
https://docs.docker.com/engine/userguide/containers/dockervolumes/
https://docs.docker.com/linux/
http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf
http://tecnologia.elpais.com/tecnologia/2012/02/10/actualidad/1328877228_999295.html
https://github.com/coreos/etcd/releases
https://github.com/kubernetes/kubernetes/wiki/Debugging-FAQ
https://sites.google.com/site/saaslink/Laird_CloudMap_Sept2008.png
http://www.huffingtonpost.com/2014/04/11/nsa-heartbleed_n_5134813.html
http://www.infoworld.com/article/2652198/security/gartner--seven-cloud-computing-security-risks.html
https://www.kernel.org/doc/Documentation/cgroup-v1/
http://kubernetes.io/
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
http://www.mooreslaw.org/
http://www.openstack.org/software/
https://owncloud.org/nine/
http://raspberrypihelp.net/raspberrypi-tutorials/
http://sandbox.ietf.org/doc/draft-aazam-cdni-inter-cloud-architecture/?include_text=1
http://sandbox.ietf.org/doc/draft-gu-sdnrg-problem-statement-of-sdn-nfv-in-dc/?include_text=1
http://setiathome.ssl.berkeley.edu/
https://www.synology.com/en-global/dsm/6.0/cloud_file_syncing
http://www.theguardian.com/commentisfree/2013/aug/11/nsa-internet-surveillance-email
http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data
http://thenewstack.io/about-etcd-the-distributed-key-value-store-used-for-kubernetes-googles-cluster-container-manager/
http://www.usatoday.com/story/news/politics/2015/06/02/patriot-act-usa-freedom-act-senate-vote/28345747/
https://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28inform%C3%A1tica%29
https://es.wikipedia.org/wiki/Historia_de_las_aplicaciones_P2P
https://es.wikipedia.org/wiki/Modelo_OSI
http://www.wdc.com/en/products/products.aspx?id=1640
https://es.wikipedia.org/wiki/Rack
https://www.zoneedit.com/free-dns/