Fundamentos y plataformas de *cloud computing*

PID_00241979

Remo Suppi Boldrito

Tiempo mínimo de dedicación recomendado: 6 horas

Universitat Oberta de Catalunya

Remo Suppi Boldrito

Ingeniero de Telecomunicaciones. Doctor en Informática por la UAB. Profesor del Departamento de Arquitectura de Computadores y Sistemas Operativos en la Universidad Autónoma de Barcelona.

Índice

Introducción

El *cloud computing* (o en sus acepciones en castellano, computación en la nube, servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos) es una propuesta tecnológica adoptada, hoy en día, por la sociedad en general como forma de interacción entre proveedores de servicios, gestores, empresas/Administración y usuarios finales para la prestación de servicios y utilización de recursos en el ámbito de las TIC (tecnologías de la información y la comunicación) y sustentado por un modelo de negocio viable económicamente.

Una definición interesante que aporta más definición sobre el *cloud computing* es la que realiza el National Institute of Standards & Technology (NIST), que expresa lo siguiente:

> «El *cloud computing* es un modelo que permite acceso ubicuo, a conveniencia, bajo demanda y por red a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con el mínimo esfuerzo de administración o sin interacción del proveedor de servicios» [Dcc11].

Este paradigma, incuestionable en cuanto a su aceptación actual, vincula proveedores y usuarios y toda su cadena intermedia de transformación y procesamiento de la información, a través de una red que en la mayoría de los casos es internet. Es habitual que los usuarios utilicen servicios en el *cloud* (ya sean servicios empresariales o personales), por ejemplo, como correo electrónico, redes sociales, almacenamiento de archivos (fotos, vídeos, ofimática) y que los utilicen desde varios dispositivos y con distintas interfaces. Tan usual es este paradigma que los usuarios digitales (jóvenes generaciones) no conocen otra forma de hacerlo ya que, por un lado, han nacido con esta tecnología que, junto con la móvil, ha transformado la sociedad actual, y por otro, la interconectividad o usabilidad de diferentes interfaces/medios de acceso son esenciales para su hacer cotidiano (compras, ocio, servicios administrativos, negocio, y una larga lista más).

1. Breve historia

Si bien tenemos presente como precursores las grandes compañías que ofrecieron este tipo de servicios (Yahoo! en 1994 y Google en 1998 –motor de búsqueda–, Hotmail en 1997 –correo– y Amazon en 2002 y AWS en 2006), es un concepto que tiene sus raíces en los años sesenta; está basado en las ideas de John McCarthy (profesor emérito de la Universidad de Stanford y cofundador del Laboratorio de Inteligencia artificial del MIT), que predijo, en 1961, durante un discurso para el centenario del MIT, que la tecnología de tiempo compartido (que estaba en auge en aquel momento) de las computadoras podría conducir a un futuro en el que el poder del cómputo e incluso aplicaciones específicas podrían ser vendidas como un servicio.

Esta idea, aunque desde otro punto de vista, también fue expresada por otros investigadores, especialmente Joseph Carl Robnett Licklider (conocido como J.C.R. o simplemente Lick), que ha sido una figura muy importante en el ámbito de la ciencia computacional y recordado, particularmente, por ser uno de los primeros que imaginó la computación interactiva moderna y su aplicación a diferentes actividades con una visión de la interconexión global de ordenadores (mucho antes que fuera construida). J.C.R. trabajó y aportó financiación a proyectos como Arpanet (predecesor de internet) o la interfaz gráfica de usuario que permitieron el acceso a servicios y recursos a personas no especialistas; ideas que, en las opiniones de expertos actuales, coinciden en que se asemejan mucho a la idea del *cloud computing* tal y como lo conocemos hoy en día.

Pero la historia del *cloud computing* también se sustenta en visionarios como John Burdette Gage, que, cuando trabajaba en Sun Microsystems, vaticinó «the network is the computer», o más recientemente, George Gilder (cofundador del Discovery Institute), que más allá de sus controvertidas opiniones en otros ámbitos, en 2006, en el artículo «The Information Factories» [Tif06], afirma que el PC ya no es el centro de atención y almacenamiento/procesamiento de la información, sino que lo es el *cloud*, y que este será el responsable de almacenar los datos de cada individuo en el planeta y que accederá a ellos por lo menos una vez en la vida. En 2013, Gilder publicó el libro *Knowledge and Power: The Information Theory of Capitalism and How It is Revolutionizing Our World*, en el cual relaciona la economía, el capitalismo y la teoría de la información de Alan Turing y Claude Shannon y cómo afectan a la sociedad actual.

Las evoluciones desde los años sesenta han sido notables e intrépidas, como se puede ver en el *Timeline of Computer History* [Tch15], pero probablemente, en cuanto a los desarrollos más vinculados al *cloud* actual, cabe destacar el desarrollo de la Web (Tim Berners-Lee propone las ideas en 1989, pero no es hasta 1990 cuando hay un prototipo del WorldWideWeb) y su evolución más

reciente, Web 2.0, y el desarrollo de internet (que en España no tiene impacto hasta principios de los años noventa con la transformación de RedIris y la conexión plena a internet).

A partir de entonces comienzan a surgir una serie de servicios, básicamente aplicaciones centradas en búsqueda, correo y páginas como Wandex, en 1993, que pretendía ser un programa para medir el tamaño de internet y que, desarrollado en el MIT, acabó siendo el primer buscador, y luego Yahoo! en 1994, Altavista en 1995 (que ha ido cambiando de propietarios desde Overture, Yahoo y ahora Microsoft) o Hotmail en 1996 (posteriormente comprado por Microsoft en 1997 cuando ya contaba con seis millones de usuarios de correo), o más recientemente también en las búsquedas de Google en 1998.

Para algunos autores, uno de los grandes hitos vinculados al *cloud* es la llegada, en 1999, de Salesforce.com (empresa que ha sido considerada en los últimos cuatro años «Forbes' Most Innovative Company»), que fue una de las primeras en ofrecer a sus clientes lo que hoy se llama SaaS (*software as a service*) y que era un software para la automatización de ventas a través de una simple página web. Algunos expertos consideran que este servicio actualmente puede ser un PaaS (*platform as a service*) y otros lo consideran una plataforma híbrida ya que ofrece las dos variantes. Esto permitió mostrar una «nueva» forma de negocio a través de internet y generar lo que hoy es una realidad a diferentes niveles de aplicaciones, servicios y recursos en la red y que se llama *cloud computing*.

En 2002, Amazon anuncia su infraestructura en la red y su ampliación en 2006 con el lanzamiento de Amazon Web Service, AWS (siendo *Elastic Compute/Storage Cloud* –EC2/S3– los servicios más importantes de AWS para proveer instancias de un servidor bajo demanda para cómputo y almacenamiento respectivamente) como un servicio orientado a negocio que permitió a las pequeñas empresas y particulares usar equipos en los cuales se ejecutan sus propias aplicaciones y con un modelo de monetización basado en el «pago por uso». En 2009 Google y otros grandes proveedores empezaron a ofrecer aplicaciones basadas en navegador popularizándose y ganando en seguridad y servicio. También otros grandes proveedores implementan sus propias infraestructuras *cloud*; por ejemplo, Microsoft Azure fue anunciado en octubre de 2008 (pero no es hasta 2010 que comienza a prestar servicios), IBM en 2011 (lanza *SmartCloud Framework* para dar soporte a su proyecto *Smarter Planet*) u Oracle en 2012 (Oracle Cloud).

En cuanto a las plataformas más representativas (no las únicas) para construir *clouds*, en 2005 se comienza a trabajar en un proyecto de investigación orientado a este ámbito basado en software *open source* y nace, en 2008, OpenNebula. En este año se forma el consorcio OpenNebula.org y es financiado por un proyecto de la UE (7th FP) llegando a tener, en 2010, 16.000 MV configuradas. En 2010, Rackspace y la NASA lanza un iniciativa *open source cloud-software* y la llaman OpenStack con el objetivo de ofrecer a las organizaciones la posibilidad de montar sus servicios de *cloud* sobre hardware estándar (en 2012 se crea

la OpenStack Foundation para promover este software en la comunidad con doscientos socios, gran parte de todos los actores importantes del mundo IT). Openstack se ha popularizado y hoy se puede encontrar en las distribuciones más conocidas (Redhat, Ubuntu, SuSE...) o también empresas que ofrecen sus integraciones como Mirantis.

También en 2010, una empresa llamada Cloud.com libera su producto orientado a la creación de *clouds*, llamado CloudStack (sobre el que había estado trabajando en secreto durante los años anteriores) bajo GPLv3. En 2011, Cloud.com es adquirida por Citrix Systems y CloudStack pasa a ser un proyecto de la Fundación Apache y distribuirse bajo la licencia Apache *Software License*. El proyecto *Virtual Grid Application Development Software* (liderado por la Rice University entre 2003-08) es el precursor de la plataforma para generar y configurar *clouds* llamada Eucalyptus. Eucalyptus es incluido en Ubuntu 9.04 (2009) y se caracteriza por integrarse con AWS. Desde 2014 la compañía del mismo nombre, y que desarrolla y mantiene el paquete, fue adquirida por Hewlett Packard y el paquete renombrando como HP Helion Eucalyptus.

Como resumen, se puede argumentar que en la actualidad es poco frecuente que una institución/empresa mediana o grande no tenga externalizados algunos servicios en el *cloud* para sus trabajadores/usuarios finales o no utilice el *cloud* para algunos de los aspectos vinculados al negocio de la misma (correo, web, intranet, etc.) excepto en aquellas que el *core business* dependa de la privacidad de los datos o estén especialmente protegidos.

2. Características, ventajas y desventajas del CC

Teniendo en cuenta las opiniones de los expertos (por ejemplo, Jamie Turner), además de la Web 2.0, las grandes revoluciones que ha facilitado la evolución del *cloud computing* es el avance de las tecnologías de virtualización, la proliferación a costos aceptables del ancho de banda y los estándares de interoperabilidad de software facilitando que, en opinión de estos expertos, hoy en día cualquier recurso y/o servicio pueda tener como base el *cloud*.

No obstante, estas opiniones tan favorables hacia el *cloud* deben considerarse dentro del marco adecuado y no caer en creencias de que el *cloud* lo puede contener y proveer todo ya que, como todo paradigma, tiene sus elementos a favor (básicamente acceso a recursos/servicios por parte de los usuarios sin ser expertos en su instalación/administración, flexibilidad de uso –y adaptativa– y precios aceptables), pero también sus puntos débiles que deben ser considerados especialmente su talón de Aquiles, que es la disponibilidad 24/7 del acceso a la red y su ancho de banda: sin red o con una red deficiente, el *cloud* no existe.

Es interesante el artículo 15 «Ways to Tell Its Not Cloud Computing» [Wtn08], que expresa claramente que no es *cloud computing* o los tres criterios para definirlo expresado por George Reese [Cca09] sobre cuando un servicio es *cloud*: accesibilidad vía *web browser* (no-propietario) o una API de *web services*, inicio sin aportar capital y solo pagar por lo que se usa cuando se use.

De forma descriptiva podemos enunciar las siguientes **características** clave para la computación en nube (orden alfabético) [Dcc11]:

1) Agilidad y autoservicio: rapidez y capacidad de proveer recursos (de forma casi inmediata) sin grandes intervenciones ni acciones por parte de las dos partes (esto se logra teniendo un flujo de trabajo predefinido muy bien ajustado y automatizando la provisión de recursos). Es decir, el usuario puede, en forma no asistida, aprovisionar capacidades de cómputo/almacenamiento/redes según le sea necesario de forma automática sin necesidad de interacción humana con el proveedor de servicios. Estos recursos estarán disponibles, en un intervalo corto de tiempo (segundos o a lo sumo unos pocos minutos), a través de la red.

2) Costo: dada la eficiencia y automatización en la gestión y administración de recursos, los precios resultantes para los proveedores de *cloud* son reducidos y pueden trasladar estos al usuario final; el usuario final no debe hacer frente a los gastos derivados de la implantación de infraestructura civil/servicios y a la inversión en máquinas, software y recursos humanos, por lo cual contabilizando todo el modelo sale favorable en relación con el *cloud*. Además, como el

control y la monitorización son automáticos, se pueden realizar medidas con un cierto nivel de abstracción apropiado para el tipo de servicio (por ejemplo, cuentas/cuentas activas, almacenamiento, procesamiento o ancho de banda); es posible ajustar el costo a modelos de, por ejemplo, pago-por-uso, pago-por-peticiones, etc. proporcionando transparencia, tanto para el proveedor como para el consumidor del servicio utilizado.

3) Escalabilidad y elasticidad: estos conceptos se refieren al aprovisionamiento de recursos en (casi) tiempo real y adaptabilidad a las necesidades de carga/uso de los mismos. Es decir, si se puede prever la carga/utilización no es necesario aprovisionar todos los recursos desde el inicio como si de una inversión local se tratara, sino planificarlos para cuando la demanda lo requiera con la consiguiente reducción del costo.

4) Independencia entre el dispositivo y la ubicación: independizar el recurso del acceso y facilitar que estos puedan ser desde una red local, corporativa u otro tipo y desde diferentes dispositivos (capacidad/tipología).

5) Mantenimiento y licencias: el modelo de actualizaciones y licencias se simplifica ya que el software se encontrará en el o los servidores del *cloud* y no existirá nada instalado en el dispositivo del usuario final.

6) Rendimiento y gestión de recursos: control exhaustivo y monitorización eficiente de los servicios para lograr una alta disponibilidad y utilización óptima de los recursos de forma automática. Estas características permiten reducir al máximo las ineficiencias aportando control y notificación inmediata, así como transparencia y seguimiento tanto al proveedor como al usuario final. Además, como los recursos se pueden configurar para servir a múltiples usuarios en un modelo de tenencia múltiple, los recursos físicos y virtuales pueden ser asignados dinámicamente y reasignados de acuerdo con la demanda de los consumidores. Esto proporciona un sentido de independencia de la ubicación y el cliente no tendrá ningún control o conocimiento sobre la ubicación exacta de los recursos proporcionados (solo a un nivel de abstracción muy alto, por ejemplo, país o centro de datos).

7) Seguridad: puede incrementarse dada la particularidad de tener los datos centralizados. Corresponderá al responsable de la aplicación la seguridad de la misma, pero al proveedor la responsabilidad de la seguridad física. Con ello, la seguridad será al menos igual que en los sistemas tradicionales, pero podrá ser mejorada (y deberá estipularse qué nivel se desea en la SLA –*Service Level Agreement*–) ya que el proveedor tiene interés en que su infraestructura sea segura para sus clientes como base de la calidad del negocio y podrá invertir en ella globalmente, mientras que esta inversión puede ser inaceptable para un cliente si debe hacer lo mismo sobre la infraestructura local.

8) Virtualización como base para el aprovisionamiento: esto permite compartir y optimizar el uso del hardware reduciendo los costos, la energía consumida/servidor y el espacio facilitando el despliegue rápido de servicios y garantizando la alta disponibilidad (servicios en *stand by*) y movilidad por carga (movimiento de máquinas virtuales a servidores más ociosos cuando la carga aumenta).

Para convencer a los más reticentes, se pueden enumerar un conjunto de **ventajas** que aporta el *cloud* como paradigma, que son (orden alfabético):

9) Fácil integración y aceptación: teniendo en cuenta su desarrollo y base en estándares, se puede integrar con mucha mayor facilidad y rapidez con el resto de las aplicaciones empresariales; el usuario final queda totalmente convencido cuando se le permite acceder desde cualquier dispositivo sin requerimientos especiales en cuanto a las necesidades del mismo.

10) Servicio global: ubicuidad y acceso a los servicios desde cualquier lugar con tiempo de inactividad reducidos al mínimo y con alta disponibilidad de los recursos.

11) Simplicidad: roles y responsabilidades muy bien definidos, separación de las actividades bien estipuladas (por ejemplo, proveedor de contenidos, proveedor de aplicación, proveedor de infraestructura y usuario final), lo cual se traduce en una forma óptima de trabajar y menor inversión por todas partes.

12) Reducción de riesgos y rapidez: no es necesario una gran inversión ni adecuación de sitios/entornos, siendo posible acelerar al máximo la implantación de nuevos servicios en las diferentes etapas de desarrollo hasta producción.

13) Reducción del uso de energía (consumo eficiente): dada la tecnología utilizada y la utilización eficiente de los recursos (favorecido por la virtualización), solo se consume lo necesario, a diferencia de los centros de datos tradicionales en los que existe un consumo fijo no dependiente de la carga/utilización.

Si bien las ventajas son evidentes y muchos actores de esta tecnología la basan principalmente en el abaratamiento de los costes de servicio, comienzan a existir opiniones en contra que consideran que un *cloud* público puede no ser la opción más adecuada para determinado tipo de servicios/infraestructura. Entre las principales causas de **desventajas** que esgrimen algunos expertos, están (orden alfabético):

1) Centralización: tanto de los datos como de las aplicaciones; genera una dependencia en relación con el proveedor; si no dispone de la tecnología adecuada (monitorización y detección) y recursos apropiados (alta disponibili-

dad), puede generar cortes o inestabilidades en el servicio. En estos casos toma especial relevancia la SLA (*service level agreement*), que especificará a qué está obligado el proveedor y las indemnizaciones que deberá hacer frente por ello.

2) Confiabilidad: la «salud» tecnológica y financiera del proveedor será un elemento clave en la continuad de su negocio y el del cliente, por lo cual las decisiones que tome el proveedor afectarán directamente al negocio cliente, y si estas no son las adecuadas, afectarán directamente a la empresa. También la empresa quedaría a merced de un mercado muy dinámico en cuanto a fusiones y monopolio (o pseudomonopolios) con el impacto que podría tener en los costos de los servicios. Un problema importante es que el proveedor por diversos motivos (legales, financieros, económicos) cierre su actividad de forma abrupta y se produzca el *data lock-in* de la empresa en los servidores del proveedor sin posibilidad de poder recuperarlos (situaciones que han pasado con el bloqueo judicial de un proveedor por diversas razones con el bloqueo de los datos de los clientes hasta que se resuelva el conflicto judicial).

3) Escalabilidad: a medida que el proveedor dispongan de más clientes, más usuarios habrá sobre el hardware, la sobrecarga en estos aumentará, y si el proveedor no dispone de un plan de escalabilidad a medio y largo plazo, para asegurar un crecimiento sostenible desde el punto de vista de las necesidades de sus clientes, se puede llegar a la saturación de los servicios, con la consiguiente degradación y pérdida de prestaciones.

4) Especialización o cualificación: la necesidad de servicios «especiales» o cualificados podrían tener una prioridad muy baja (y su consiguiente retardo en su despliegue) para el proveedor si estos no son requeridos por otros clientes, afectando al negocio de uno en particular que sí los necesita para el suyo.

5) Disponibilidad: el principal punto débil de una infraestructura en *cloud* es el acceso a internet. Si no se dispone de él y es confiable y de un ancho de banda aceptable, el *cloud* deja de tener efectividad.

6) Riesgo y privacidad: los datos «sensibles» de negocio no residen en las instalaciones de las empresas y la seguridad no depende de los recursos humanos de esta sino del proveedor del servicio. Si se asume que estos datos son de alto valor en un contexto vulnerable, el riesgo puede ser muy alto por su posible robo (copia), acceso a la información (lectura) o destrucción (borrado).

7) Seguridad: dado que la información deberá atravesar diferentes canales y servicios, puede resultar que cada uno de ellos sea un foco de inseguridad. Si bien esto puede ser resuelto mediante canales y servicios seguros, la posibilidad de un fallo en la cadena de cifrado de la información puede ser posible con los consiguientes problemas que representa. Además, en este caso el propietario de la información puede desconocer totalmente qué ha pasado y dónde ha estado el fallo.

8) ***Vendor lock-in***: es un gran problema (que en la actualidad se ha demostrado como uno de los más frecuentes) que hace que un cliente dependa de un proveedor de productos y servicios, incapaz de usar otro proveedor sin costos de cambio sustanciales, aunque el cambio signifique una reducción de costos o mejores prestaciones. Muchos desarrolladores o usuarios finales son reticentes al *cloud* por este motivo, ya que significa un compromiso adquirido; después, si se desea cambiar el costo, será muy elevado.

En [Vcc10] amplía estas «desventajas» y demuestra con números sobre casos de uso en un proveedor real algunas de estas cuestiones, apuntando cuáles son los riegos que se deben considerar antes de tomar las decisiones o de firmar un SLA.

Como se puede ver entre características, ventajas y desventajas, pueden existir elementos que entran en contradicción o que desde otro punto de vista son opuestos (por ejemplo, el tema de la seguridad); el equipo de toma de decisiones de la empresa deberá analizar cuidadosamente las posibilidades y qué ventajas y riesgos asume. Desde un punto de vista global, es una tecnología que aporta beneficios y que, con la adecuada planificación y toma de decisiones, valorando todos los factores que influyen en el negocio y escapando a conceptos superficiales (todos lo tienen, todos lo utilizan, bajo costo, expansión ilimitada, etc.), puede ser una elección adecuada para los objetivos de la empresa, su negocio y la prestación de servicios en IT que necesita o que forma parte de sus fines empresariales.

La figura siguiente muestra una visión general de los **actores y elementos** en juego dentro del *cloud computing* donde, de forma abstracta, podemos identificar:

- **Infraestructura**: recursos hardware y software que gestiona el proveedor y que serán objeto de utilización por las empresas y clientes de este.

- **Catálogo de servicios**: elementos ofrecidos por el *cloud* y que serán seleccionables por categorías o prestaciones por los clientes del proveedor de servicios.

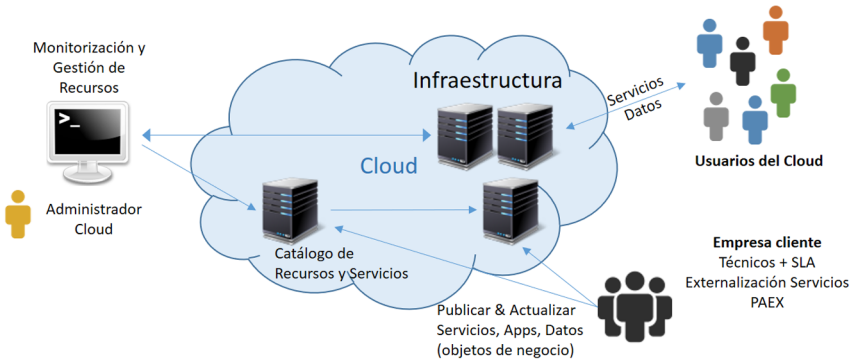
- **Administrador del *cloud***: cuerpo técnico de profesionales que garantizarán las prestaciones, seguridad, estabilidad, disponibilidad, etc. de los servicios comercializados sobre la base de una SLA firmada con cada cliente.

- **Empresa cliente**: usuarios que realizan una externalización de sus servicios de IT y publican y actualizan sus aplicaciones/datos en el *cloud*. Se basan en una garantía del servicio que ofrece el proveedor y forman parte de un acuerdo de prestación de servicios (SLA); generalmente, el costo estará basado en políticas similares a «pago por uso».

- **Usuarios**: clientes de la empresa que han puesto sus servicios en el *cloud* y que pueden conocer o no que los mismos están siendo provistos desde el *cloud*. Solo acceden a un servicio, aplicación o datos como si se entrara en los servidores de la empresa que provee el servicio.

Es importante hacer notar que los usuarios (a veces indicados como usuarios finales) generalmente no son clientes del proveedor de servicios del *cloud* sino de la empresa que provee los servicios en el *cloud*.

Por ejemplo, si consideramos los principales clientes de Amazon AWS [Acs16] (es decir, aquellas empresas que tienen la etiqueta «millones de dólares» asociada a alguna característica: ingresos, activos totales, financiación, valoración o beneficios), encontramos Adobe Systems, Airbnb, Alcatel-Lucent, Aon, Autodesk, BMW, Bristol-Myers Canon, Capital One, Comcast, Docker entre muchas. Si tomamos por ejemplo Adobe Systems, podremos ver el rol de cada uno de ellos, donde el **proveedor del *cloud*** y la infraestructura la provee **AWS**, la **empresa** que lo contrata es **Adobe** y esta comercializa los productos a través de **Creative Cloud**, que permite que **usuarios** (finales) tengan acceso a diferente software de diseño gráfico, edición de video, diseño web y servicios en el *cloud* por una cuota económica mensual.



3. Clasificación

Existen diferentes taxonomías para describir los servicios y la forma de despliegue del *cloud*. En su documento *The NIST Definition of Cloud Computing del NIST* [Dcc11], se definen cuatro modelos de despliegue y tres modelos de servicio. Como modelos de despliegue, podemos enumerar:

a) *Cloud* público: en este caso, el sistema es abierto para su uso general bajo diferentes modelos de negocio (desde pago por recursos, por uso, cuota o libre). Este tipo de recurso es mantenido y gestionado por un tercero y los datos y aplicaciones de los diferentes clientes comparten los servidores, sistemas de almacenamiento, redes y otras infraestructuras comunes; normalmente los recursos se utilizan y gestionan por medio de internet. Generalmente, un cliente no tiene conocimiento sobre con qué otros usuarios comparte la infraestructura y su utilización se contrata a través de un catálogo de recursos disponibles; su aprovisionamiento es automático (o semiautomático) después de haber cumplido con una serie de trámites en cuanto al modelo de pago y SLA. El propietario del *cloud* (proveedor de recursos y servicios) puede ser una empresa privada, una institución académica o una organización gubernamental, o una combinación de ellas, generalmente en función del tipo de clientes y recursos que se deberán proveer. Técnicamente puede haber pocas diferencias (o ninguna) con alguno de los otros modelos (especialmente con la privada); sin embargo, puede haber diferencias sustanciales en relación con la seguridad, que en el *cloud* público puede estar disponible en una red abierta como internet y sin mecanismos de cifrado.

> Como ejemplo de estos servicios podemos mencionar AWS (Amazon Web Services), Microsoft Azure, Google Compute, Rackspace o IBM-SoftLayer (hasta 2014 IBMSmartCloud) que operan su propia infraestructura y sus recursos son accesibles en internet o el CSUC (Consorci de Serveis Universitaris de Catalunya), que ofrece, bajo diferentes modelos de explotación, recursos de *cloud* para instituciones académicas y de investigación de Cataluña (http://www.csuc.cat/es/investigacion/infraestructura-en-la-nube).

Es interesante conocer que algunos proveedores intentan reducir la probable inseguridad a través de servicios de conexión directa (por ejemplo, AWS *Direct Connect* o Azure *ExpressRoute*), que permiten que los clientes puedan comprar o alquilar una conexión privada para conectarla a un punto privado del proveedor, con el consiguiente incremento de la privacidad que significa un recurso de este tipo. Para los clientes todos sus costes son operativos (OPEX). Una visión de conjunto de proveedores y servicios se puede analizar en [Cis16] (donde se analizan los puntos fuertes y débiles de cada operador), realizado en agosto de 2016, y en [Cmh15] (este informe está orientado a *Cloud-Enabled Managed Hosting*, que indica la calidad de las empresas que ofrecen servicios de migración y adaptación de otros proveedores).

b) *Cloud* privado: en este caso, la infraestructura funciona exclusivamente para una organización/empresa y es utilizada por sus unidades de negocio o departamentos; esta puede ser la propietaria, la administradora y la operadora, o algunos de estos apartados puede ser subcontratado a un tercero. Es la opción más favorable para las compañías que necesitan alta protección de datos y un alto nivel de servicio, ya que los recursos y su gestión se encuentran bajo el control y cuidado de la propia empresa/organización.

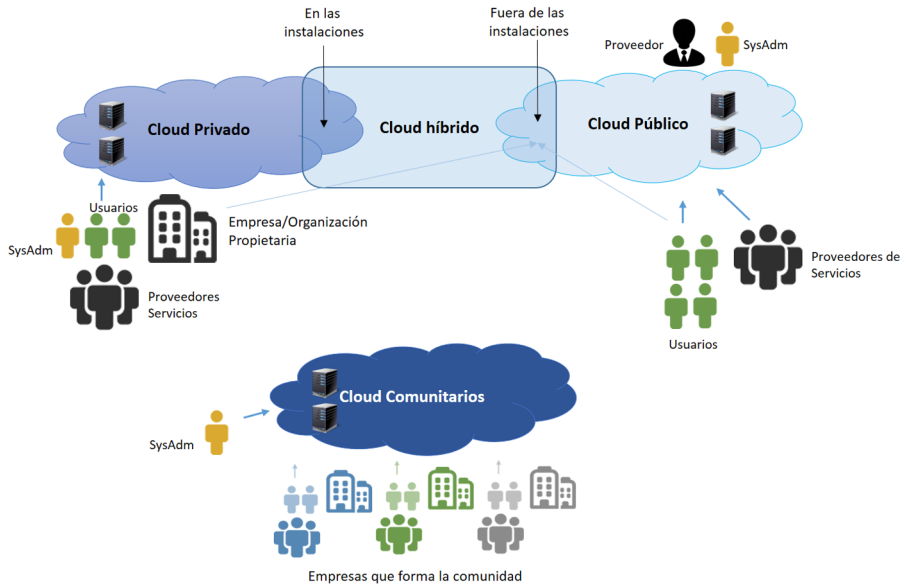
Si bien resuelve algunos problemas de algún tipo de empresa (por ejemplo, aquellas con legislación especial como las empresas públicas), esta debe asumir el costo de la inversión (CAPEX), pero como contrapartida dispone de una infraestructura bajo demanda, gestionada por personal propio (o externo, pero bajo sus criterios) y que controla qué y dónde deben ejecutarse las aplicaciones y manteniendo la privacidad de su información, permitiendo definir las políticas de acceso y evitando el *lock-in* de los datos, así como el *vendor lock-in* mencionado anteriormente.

c) *Cloud* híbrido: aquí se combinan modelos públicos y privados; el propietario dispone de una parte privada (con acceso controlado y bajo su control) y comparte otras, aunque de una manera controlada. Las nubes híbridas ofrecen la posibilidad de escalar muy rápidamente con aprovisionamiento externo bajo demanda, pero implican una gran complejidad a la hora de decidir cómo se distribuyen los datos y las aplicaciones en una banda u otra. Si bien es una propuesta atractiva para las empresas, los casos habituales de uso no pasan de aplicaciones simples sin restricciones de sincronización con bases de datos sofisticadas.

> Un ejemplo de ello son algunas implementaciones de correo empresarial o aplicaciones de ofimática.

d) *Cloud* comunitario: los servicios están diseñados para que puedan ser utilizados por una comunidad, organizaciones o empresas bajo el mismo alineamiento de objetivos o negocio (por ejemplo, bancos, distribuidores, arquitectos...) o que requieran unas características específicas (por ejemplo, seguridad y privacidad). Las empresas que forman la comunidad pueden ser los propietarios de infraestructura, así como los gestores u operadores; algunos de estos roles pueden ser subcontratados a terceros, pero bajo las indicaciones y reglas que aplica a la comunidad.

La figura siguiente muestra un esquema de estas cuatro formas de implantación y desarrollo de los *clouds*.



Otra de las clasificaciones habituales es **por el nivel de servicio** que prestan; si bien los tradicionales son tres: **IaaS** (*infraestructure as a sevice*), **SaaS** (*software as a service*), **PaaS** (*platform as a service*), hoy podemos encontrar otras extensiones o derivadas de los habituales como pueden ser **BaaS** (*business as a service*), **StaaS** (*storage as a service*), **DaaS** (*desktop as a service*), **DRaaS** (*disaster recovery as a service*), **MaaS** (*marketing as a service*); algunos autores le dan un nombre global con **XaaS** (*everything as a service*) para referirse a la creciente diversidad de servicios disponibles en el *cloud* a través de internet (consultar [Idd09]). Un caso que afirma estas consideraciones es **aPaaS** (*application platform as a service*), que permiten un rápido desarrollo y aprovisionamiento de aplicaciones (*apps*) como plataforma específica para la codificación, aprovisionamiento y despliegue de *apps* y que soporta el ciclo de vida completo, proporcionando una forma más rápida de crear aplicaciones.

Ejemplo de aPaaS

Ejemplos de su aceptación es la amplia propuesta del mercado: AWS con Elastic Beanstalk, CenturyLink con AppFog, Google con App Engine, IBM con BlueMix, entre otros).

En la **infraestructura como servicio** (IaaS) se encuentra la capa de recursos básica (generalmente máquinas virtuales, redes y almacenamiento) donde el cliente podrá poner sus SO y sus aplicaciones e implementar un servicio o utilizarlas para ejecutar sus aplicaciones. El cliente no podrá manejar o controlar el hardware subyacente, pero sí a partir del SO, almacenamiento y aplicaciones/servicios implementados sobre esas máquinas, así como algunos aspectos de la conectividad (subredes, *firewalls*, dominios...).

> Un ejemplo de gran proveedor IaaS (decenas a miles de MV) podría ser Amazon EC2, Google Compute Engine o Digital Ocean como servicio representativo para pymes (en unidades de MV).

La reflexión del usuario en esta modalidad sería «¿por qué comprar, instalar y probar infraestructura cada N años (obsolescencia) y no alquilarla y pagar por uso?». Los recursos son obtenidos sin hacer obra civil (centro de datos), con mínimos recursos humanos (no especialistas en IT), sin gran inversión inicial, escalable y eficiente y a costos aceptables.

La **plataforma como servicio** (PaaS) es la encapsulación de un ambiente de desarrollo y la provisión de una serie de módulos que proporcionan una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.). De esta forma, una estructura básica de este tipo podría consistir en un entorno que contenga servicios/aplicaciones, librerías y API para un fin específico.

Por ejemplo, para una tecnología de desarrollo en particular sobre Linux y un servidor web + una base de datos y un ambiente de programación como Perl o Ruby.

Es decir, el cliente no gestiona ni controla la infraestructura (ni servidores, ni sistemas operativos, ni almacenamiento, ni ningún tipo de elementos de red o seguridad, etc.), pero tiene el control de las aplicaciones desplegadas y de la configuración del entorno de ejecución de las mismas (bases de datos y *middelwares*). Es habitual que una PaaS pueda prestar servicios en todos los aspectos del ciclo de desarrollo y pruebas de software o en un desarrollo, gestión y publicación de contenidos.

Ejemplo de ello podrían ser Cloud9, Google App Engine, Heroku, u OpenShift.

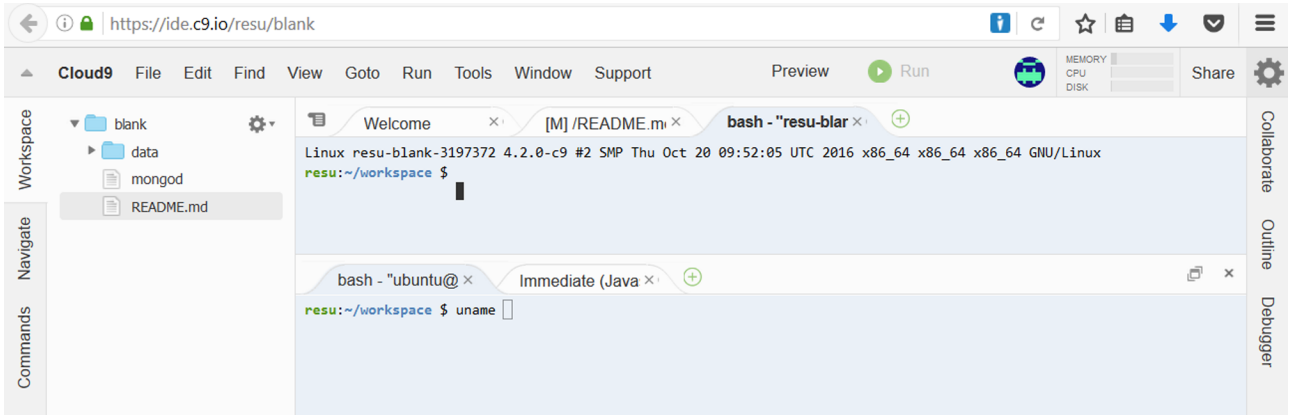
Como resumen en esta modalidad el deseo del usuario sería «Necesito este software instalado sobre este operativo y esta base de datos y con esta API», y recibiría todo el conjunto (hw, SO, base de datos, librerías, API, seguridad, GUI y otras herramientas) listo para usar en pocos segundos y desarrollar aplicaciones empresariales/móviles, páginas web, contenidos, etc.

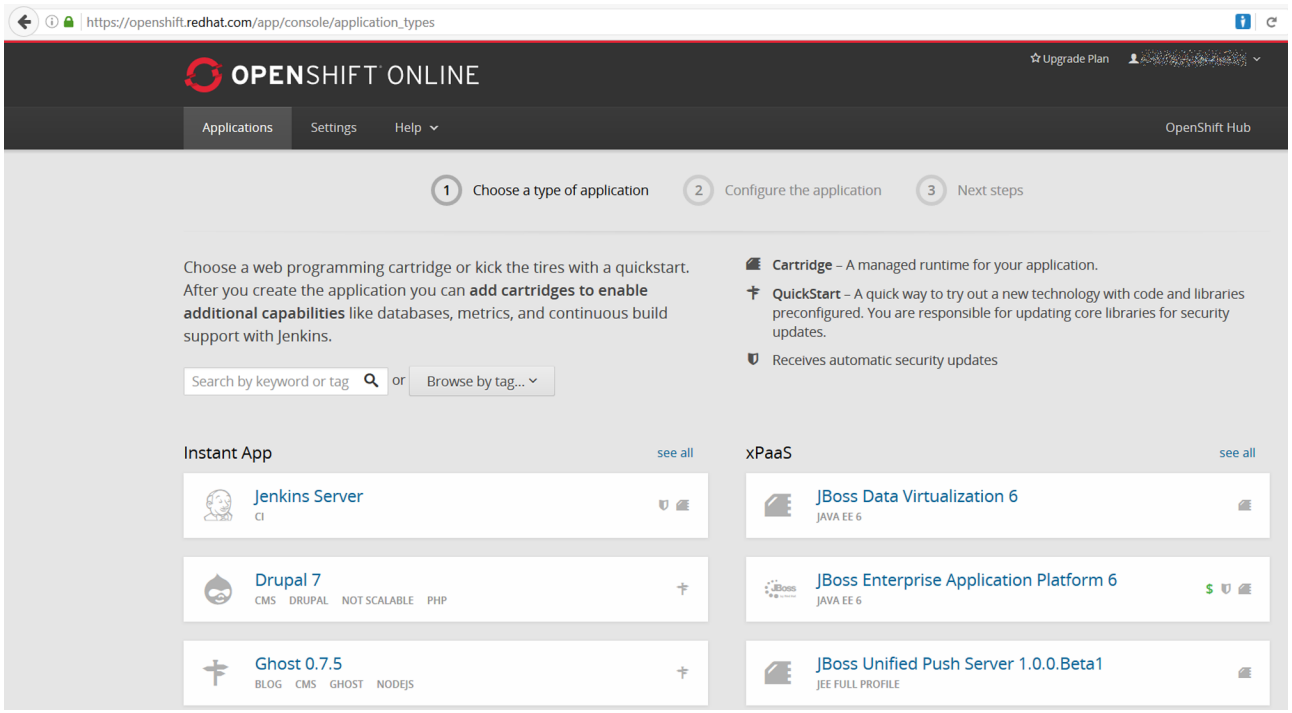
Finalmente, **software como servicio** (SaaS) se encuentra en la capa abstracta (si bien existe una tendencia bastante extendida a considerar, que el SaaS es la capa más simple del PaaS, o la más baja) y caracteriza una aplicación completa ofrecida como un servicio, bajo demanda generalmente con tenencia múltiple (arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes u organizaciones). En general, las aplicaciones bajo este modelo de servicio son accesibles a través de un navegador web y el usuario no tiene control sobre ellas, aunque en algunos casos se le permite realizar algunas configuraciones. Esto le elimina la necesidad al cliente de instalar la aplicación en sus propios ordenadores, eliminando el soporte y mantenimiento del hardware y software y mejorando el control y gestión de licencias si son necesarias.

Como ejemplos representativos de SaaS podría enumerarse WebMail (Gmail, Outlook, Yahoo, etc.), SalesForce, Google Docs, Office 365, o SiteBuilder.

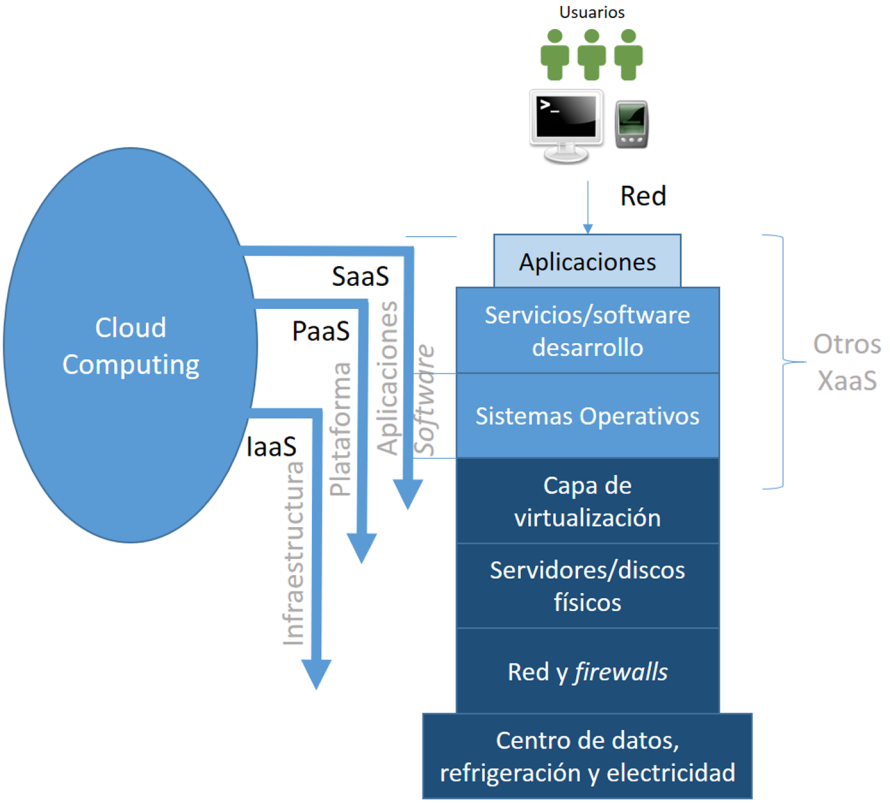
En esta modalidad se podría resumir que el pensamiento del usuario sería «Ejecute esto por mí» sin responsabilidades en la gestión de hardware o software, utilizando un navegador web o evitando la instalación de software cliente, basado en un servicio bajo demanda, con escalabilidad casi inmediata, acceso desde cualquier sitio y con cualquier dispositivo, bajo un modelo de soporte de 24/7 y un modelo de pago como *pay-as-they-go* y *pay-as-they-grow*.

Las dos figuras siguientes muestran las pantallas de Cloud9 y de Openshift (donde se pueden obtener cuentas gratuitas para pruebas de concepto con diferentes modalidades de funcionamiento y desarrollo), como ejemplo de la facilidad y simplicidad de poner en marcha un entorno complejo solo en unos segundos y preparados para desarrollar aplicaciones/servicios sobre ellos.





La figura siguiente resume en un diagrama las diferentes modalidades de servicio del *cloud* y qué implica cada una de ellas.



Si se parte de la base de que una infraestructura de *cloud computing* es un sistema distribuido en el sentido general del término, y entendiendo este como una colección de recursos conectados entre sí por una red de comunicaciones donde cada máquina posee sus componentes de hardware y software que el usuario percibe como un único sistema, es necesario **compararlo con otras tecnologías** dentro de este apartado. Entre ellas podemos mencionar:

a) Clústeres: se aplica a agrupaciones de ordenadores unidos entre sí (generalmente) por una red de alta velocidad y que se comportan como si fuesen un único ordenador. Son utilizados como entornos de procesamiento de altas prestaciones (*high performance computing*) y servicios para aplicaciones críticas o de alto rendimiento entre otros usos. Su utilización se ha popularizado para aplicaciones que necesitan un elevado tiempo de cómputo (por ejemplo, científicas) y están basados en infraestructura comercial (*blades* o *slices*) junto con un red de altas prestaciones (por ejemplo, Infiniband), que utilizando en su gran mayoría software *open source* (Linux, NFS –aunque cada vez más está siendo reemplazado por otros sistemas de archivos distribuidos como Ceph, Lustre, GlusterFS– y sistemas de gestión de colas como SGE y Slurm y librerías como por ejemplo OpenMPI para ejecutar tareas distribuidas sobre la arquitectura y OpenMP para aprovechar la potencialidad de los sistemas *multicore*) permite desarrollar y ejecutar aplicaciones concurrentes/distribuidas de altas prestaciones. Esto permite disponer de una infraestructura que provee de alto rendimiento, alta disponibilidad, con balanceo de carga, eficiente, escalable y a costos razonables. Existe una clasificación de los clústeres en función de

qué característica predomina siendo: alto rendimiento (HPC-*high performance computing*), alta disponibilidad (HA/HAC-*high availability computing*) y alta eficiencia (HT/HTC-*high throughput computing*).

b) Superordenador: es una evolución funcional a gran escala de un clúster (si bien hay diferentes arquitecturas) que poseen una capacidad de cómputo mucho más elevada que estos (y por lo tanto que un ordenador de propósito general). Su rendimiento se mide en TeraFlops (10^{12} operaciones de punto flotante por segundo) y el más potente publicado en la lista del Top500 (los quinientos superordenadores más potentes del mundo), en junio del año 2016, que es el Sunway TaihuLight con 93.014 TeraFlops, que está compuesto de 10.649.600 *cores* de procesamiento, dispone de 1,3 PetaBytes de memoria y consume 15.371 kW.

Su historia se inicia a finales de los años cincuenta, siendo IBM, Sperry Rand y Cray los grandes actores de aquella época con arquitecturas específicas (inicialmente con Univac como el primer ordenador comercial, y posteriormente IBM7030 o Cray-1), siguiendo por las arquitecturas vectoriales (década de los setenta) con un mercado dominado por Control Data Corporation (CDC) y Cray Research. Con el auge de los microprocesadores (Intel) en la década de los ochenta, comienzan a emerger multiprocesadores escalares con compartición de memoria (SMP-*symmetric multiprocessor*) y luego sin compartición de memoria (MPP-*massively parallel processor*), que luego derivan en clústeres de miles de procesadores en los noventa y a finales de la década y de forma masiva, superordenadores con decenas de miles de procesadores.

Es importante mencionar que un superordenador puede tener dos enfoques diferentes:

- la evolución del clúster (como por ejemplo en MareNostrum en Cataluña, el Finisterrae en Galicia o el Juqueen en Alemania), donde se utilizan de forma dedicada al cómputo de altas prestaciones, comparten un espacio físico y disponen de una serie de recursos comunes como redes de comunicación de alta velocidad o espacio de almacenamiento compartido y distribuido, o

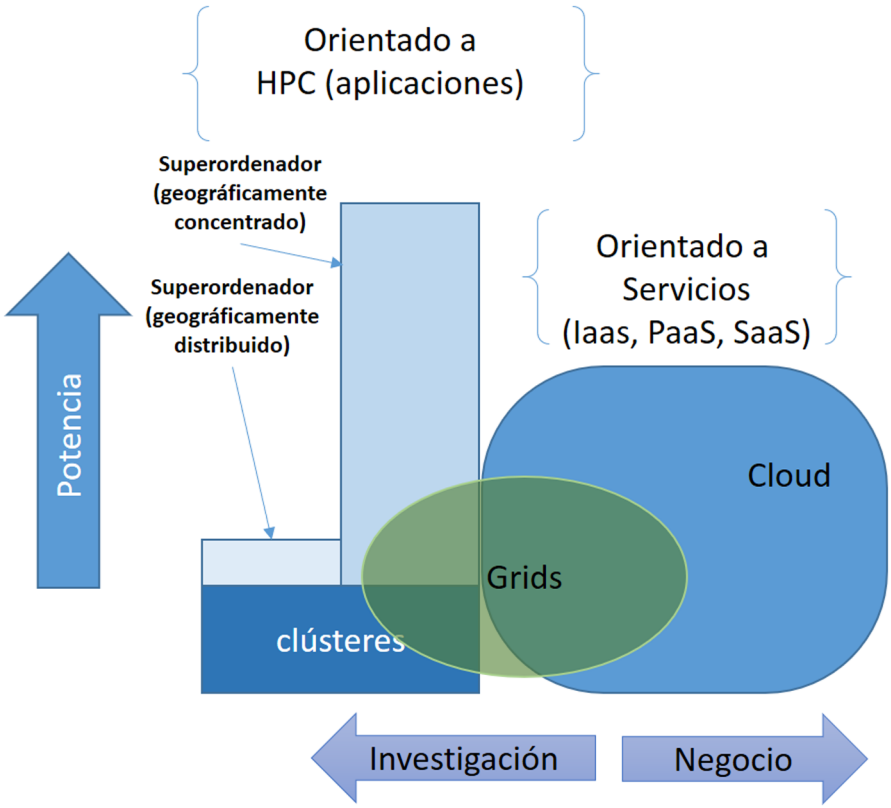
- superordenador distribuido a través de una red formada por cientos o miles de ordenadores discretos distribuidos a través de una red (internet) que se dedican de forma no exclusiva a la solución de un problema común (generalmente cada equipo recibe y procesa un conjunto de tareas pequeñas trasladando los resultados a un servidor central que integra los resultados en la solución global).

Ejemplos representativos de estos últimos son proyectos como Seti@home, Folding@home, World Community Grid o ClimatePrediction.net donde muchos de ellos están basados en Boinc (una arquitectura software *open source* que permite que voluntarios aporten sus recursos a un problema común –paradigma conocido como *volunteer computing*) donde se puede observar en la página de sus desarrolladores que la potencia de cómputo generada por esta arquitectura en todos los proyectos que la utilizan es de

12,2 PetaFlops conseguido a través de 278.493 voluntarios y 944.194 ordenadores (octubre de 2016).

c) *Grid*: se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos propiedad de diferentes instituciones, pero unidos por una capa de software (*middleware*) común que permite utilizarlos como si fuera un único superordenador. Para ello, y con el fin de colaborar aportando los recursos de cómputo gestionados por cada institución, las universidades, los laboratorios de investigación o las empresas se asocian para formar *grids* y así ceder sus recursos temporalmente, pero también para disponer del cómputo equivalente a la unión de todas estas infraestructuras. Las ideas de *grid* fueron establecidas por Ian Foster, Carl Kesselman, que fueron los precursores en la creación de Globus Toolkit considerado como la primera herramienta para construir *grids*. Entre los grandes proyectos *grid* se pueden encontrar CrossGrid o EU-DataGrid, o más recientemente, el proyecto EGI-InSPIRE, todos ellos financiados por la Comunidad Europea. Es importante destacar que el *grid* es un tipo de infraestructura de cómputo cuyo ámbito de utilización y propósito natural es el científico (*e-science*), como universidades y laboratorios de investigación, mientras que el *cloud* nace desde la prestación de servicios y los negocios a las empresas y usuarios (*e-business*) desde otras empresas, pero ambos son similares en paradigmas y estructuras para manejar grandes cantidades de recursos distribuidos, siendo considerados por algunos expertos como las dos vías del uso masivo de recursos (bajo diferentes modelos): uno para la e-ciencia exclusivamente (*grid*) y otro para las empresas (*cloud*). Es importante destacar que comienzan a ofrecerse servicios cruzados, es decir, altas prestaciones (*high performance computing*) en el *cloud* (por ejemplo, instancias con base en hardware de GPU, SSD y aceleradores como Xeon Phi).

Con esta caracterización de los recursos y su utilización podemos ubicar gráficamente cada una de estas arquitecturas en función de la potencia de cómputo involucrada y su tipo de dedicación (ver figura siguiente).



Desde el punto de vista de los **paradigmas precursores** que son esenciales en el *cloud computing*, podemos contar:

a) Cliente-servidor: el modelo cliente-servidor se refiere de forma general a los servicios implementados sobre un sistema de cómputo distribuido (en sus orígenes implementados por Sockets y RPC; ver ejemplos de programas en la asignatura Administración avanzada). Este tipo de paradigma será utilizado en todas las aplicaciones basadas en servicios, así como en una gran parte del código que se ejecute dentro del *cloud* como aplicaciones de alto rendimiento.

b) WS & SOA: no son conceptos nuevos (definidos en la década de los noventa); SOA (*service-oriented architecture*) es definido como una arquitectura para diseñar y desarrollar sistemas distribuidos y que se utiliza para el descubrimiento dinámico y el uso de servicios en una red y WS (*web services*) como servicios prestados a través de internet usando tecnologías como XML, *Web Services Description Language* (WSDL), *Simple Object Access Protocol* (SOAP), y *Universal Description, Discovery, and Integration* (UDDI). Los servicios de *cloud computing* son prestados generalmente a través de estas tecnologías, las cuales también pueden ser utilizadas internamente para su organización y gestión interna.

c) Web 2.0: indudablemente es la tecnología que facilita la compartición de información, la interoperabilidad, el diseño centrado en el usuario y la colaboración en la WWW. Evidentemente, esta tecnología aporta mucho al *cloud computing* ya que la interacción y prestación de servicios con el usuario y los

clientes (y entre aplicaciones) será a través de las posibilidades que brinda mediante estilos (CSS), lenguajes (HTML5, XML, XHTML, JavaScript, RoR, etc.), aplicaciones RIA (*Rich Internet Applications*, Ajax), agregación (RSS, ATOM), *JavaScript Client Communication*, interoperabilidad (*Representational State Transfer*, REST), APIS (WebGL, DOM, etc.), formatos de datos (JSON, XML) o *Mashup* (aplicación web híbrida).

Dentro de esta comparación, se puede argumentar que la tecnología principal que soporta al *cloud computing* es la **virtualización**. La capa de virtualización (hipervisor, por ejemplo, Xen, VirtualBox, OracleVM, KVM, VMware ESX/ESXi, Hyper-V, entre otros) separa un dispositivo físico en uno o más dispositivos «virtuales» (llamadas máquinas virtuales –MV–), donde cada uno de ellos pueden ser asignados, utilizados y gestionados por el cliente de forma muy simple, como si de máquinas físicas se tratara, pero con las consiguientes ventajas (aislamiento, fácil mantenimiento, puesta en marcha, etc.). Hoy en día todos los sistemas utilizan técnicas de virtualización asistidas por hardware (extensiones del procesador VT-x o AMD-V) de forma tal que es posible tener máquinas virtualizadas muy eficientes, disponibles y configuradas (mediante técnicas de clonado o *pool of VM in stand-by*) para ser asignadas a un usuario bajo demanda y en pocos segundos.

Un paso adicional que ha permitido un incremento notable de la eficiencia, disponibilidad y una revolución en los entornos de desarrollo ha sido la **virtualización en el sistema operativo** para la creación de un sistema escalable de múltiples entornos independientes con un mínimo impacto en la utilización de los recursos. La virtualización del SO permite que el núcleo de un SO (*kernel*) pueda albergar múltiples instancias de usuario aisladas donde cada una de ellas actúa como un «contenedor» con sus propias librerías y servicios, pero utilizando el SO subyacente. Los nombres dados a estas instancias son el de *containers*, *virtualization engines* (VE) o *jails* (por ejemplo, *FreeBSD jail* o *chroot jail*). A ojos de un usuario este «contenedor» es similar a un servidor real y tendrá todos los elementos necesarios, como si del servidor real/virtualizado se tratara, pero con solo un pequeño gasto de memoria, CPU y disco. Sobre sistemas Linux (o *nix) es una evolución de mecanismo de *chroot* más un sistema de control/limitación de la utilización de recursos de un contenedor respecto a otro. Una de las desventajas de los contenedores es que, al compartir el SO, no se pueden tener contenedores con sistemas operativos que no compartan el mismo núcleo (por ejemplo, si el *host* es Linux no podremos poner un contenedor con Windows, cosa que sí es posible si fuera una máquina virtual). Entre el software de virtualización del SO *open source* podemos encontrar Docker, LXC/LXD, OpenVZ, FreeBSD Jails, o en software propietario Virtuozzo (basado en OpenVZ).

El *cloud computing* también comparte características con otros modelos/paradigmas de cómputo distribuido (algunos de futuro) como:

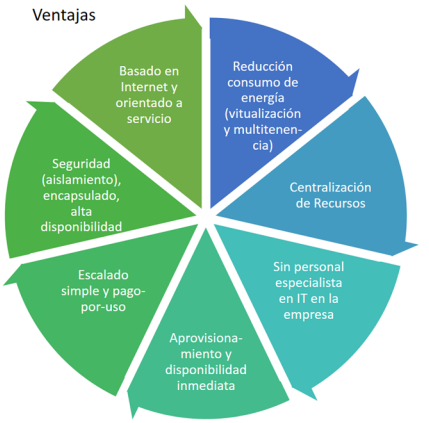
d) ***Dew computing***: nuevo paradigma de cómputo distribuido que considera que los equipos locales (ordenadores de sobremesa, portátiles y dispositivos móviles) proporcionan un entorno propicio para microservicios independientes de los servicios *cloud* y estos pueden colaborar con ellos. Es decir, este paradigma plantea la distribución de las cargas de trabajo entre servidores del *cloud* y los dispositivos discretos/locales para utilizar plenamente el potencial de ambos. Algunos autores hablan más de *mobile cloud computing* [Mcc13], donde se integran y distribuyen los servicios y el almacenamiento de los datos en los dispositivos móviles y los proveedores de servicios *cloud* (algunas experiencias acercan más a este paradigma al *volunteer computing*).

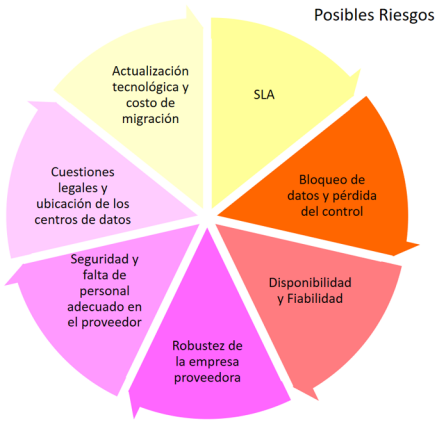
e) ***Edge computing***: es un paradigma de computación distribuida que proporciona servicios de datos, computación, almacenamiento y aplicaciones más cerca de dispositivos cliente o en los dispositivos cercanos al usuario. Es decir, procesa y almacena los datos sobre los dispositivos en el borde de la red (por ejemplo, dispositivos móviles) en lugar de enviar los datos a un lugar en la nube para ello. Es considerado una forma de computación distribuida de proximidad, donde cada uno de los dispositivos conectados a la red puede procesar los datos y solo transmitir un conjunto de ellos que sean de interés o hacerlo solo en situaciones excepcionales (alarmas o notificaciones) teniendo una capacidad autónoma y sin dependencias del servidor en la nube. Es un modelo que, con el crecimiento que se espera de la IoT (*internet of things*), permitirá que los sistemas *cloud* y las redes no se colapsen ante el incremento exponencial de datos a transmitir, almacenar y procesar. Algunos autores diferencian esta del ***fog computing*** donde se utiliza agrupaciones/multitudes de usuarios finales (o dispositivos cercanos al usuario) para: almacenar datos (en lugar de hacerlo sobre el *cloud*), comunicación (en lugar hacerlo a través de internet), el control, configuración, medición y gestión (en lugar de ser controlados principalmente *gateways* de red como ocurre en la red LTE, por ejemplo). Este paradigma ha generado un movimiento importante de las grandes empresas de tecnología (OpenFog)

f) ***Peer-to-peer computing***: este paradigma plantea el uso de una arquitectura distribuida sin la necesidad de una coordinación central para realizar cómputo y almacenamiento de datos. Los participantes son los proveedores y consumidores de recursos (en contraste con el modelo tradicional de cliente-servidor) y todos se ayudan para procesar/almacenar y comunicar los datos en forma igualitaria. Los *peer* ponen una parte de sus recursos (potencia de procesamiento, almacenamiento o ancho de banda de red) directamente a disposición de otros *peers* de la red, sin la necesidad de una coordinación central por los servidores. Este tipo de paradigma ha tenido mucha aceptación en servicios de gestión de contenidos (Bittorrent, Spotify, Skype), compartición de archivos (Gnutella, Edonkey), dinero digital (Bitcoin, Peercoin), Anonimización (I2P), entre otros.

g) ***Volunteer computing***: como se mencionó anteriormente en el apartado de Superordenadores, este tipo de procesamiento distribuido se basa en los usuarios aportan sus recursos (básicamente procesamiento y almacenamiento) para un determinado proyecto del cual forman parte. El primer registro que se tiene de este tipo de cómputo distribuido es en 1996, con Great Internet Mersenne Prime Search (búsqueda de número primos que cumplen con $2^n - 1$, por ejemplo, 3, 7, 31...) seguido por distributed.net (en 1997) y a continuación muchos otros, entre ellos Bayanihan cuyos desarrolladores propusieron el nombre de *volunteer computing*. En la actualidad muchos de ellos están basados en la arquitectura Boinc (desarrollada por la Universidad de California, Berkeley) con proyectos como los que se han mencionado anteriormente (Seti@home), o aquellos que han desarrollado su propia arquitectura software como Folding@home (Universidad de Stanford) que en mayo de 2016 anunciaron que había llegado a la marca de 100 PetaFlops (x86).

En las figuras siguientes se muestran, en forma resumida, tanto las características a favor del *cloud computing* como aquellas que no son tan favorables y que pueden representar un riesgo y que se deberán analizar con cuidado.





4. Plataformas más representativas

Más allá del extenso catálogo en las diferentes opciones de plataformas y opciones propietarias, existe un gran conjunto de plataformas basadas en software libre que permiten desarrollar y poner en marcha infraestructura de *cloud computing* sobre hardware comercial, pero sin grandes requerimientos. Estos aspectos se deben valorar muy bien, ya que son opciones totalmente válidas para sistemas en producción y algunas de ellas permiten vincularse a los operadores *cloud* públicos para extender la potencialidad de los privados. Algunas de estas plataformas cuentan con un gran número de instalaciones operativas que, dependiendo del tamaño de la empresa, las necesidades y el control de la información que se desee tener, es totalmente viable y muchas de ellas son utilizadas por incluso los operadores propietarios. Muchas de las opciones que se mostrarán a continuación son productos con una gran comunidad de desarrolladores y usuarios y algunas se han transformado en empresas (o empresas subsidiarias) o han nacido como proyectos *open source* en las empresas y que trabajan con una versión *community* y otra versión empresarial con más soporte o adaptaciones a las necesidades específicas del cliente o con SLA o QoS diferenciados, pero utilizando el mismo software que la versión *open source*. Generalmente se detecta en las versiones *open source* un factor de innovación y crecimiento constante que presenta nuevas adaptaciones, servicios y mejoras en un ciclo de evolución más rápido y constante que sus correspondientes propietarios, que portan a la generación de nuevos estándares, nuevas API públicas favoreciendo la creación de conocimiento y posibilidades adaptadas a todas las necesidades [Osc14].

No obstante, esta proliferación de plataformas y servicios puede desconcertar y generar cierta confusión y complicar la elección, pero se debe tomar desde un punto de vista pragmático y «mirar» cuál de ellas se acerca más a las necesidades y se adecúa a la infraestructura hardware de que se dispone. Es importante hacer pruebas de implantación y desarrollar sistemas de preproducción para ver cómo se comporta la plataforma escogida y si es adecuada a las necesidades reales de la organización.

El mercado, por otro lado, ha evolucionado en todos los aspectos y hoy es posible encontrar gran cantidad de proveedores (algunos de ellos ya se han mencionado), como se puede observar en las recomendaciones y la lista *The Best Cloud Computing Companies And CEOs To Work For In 2016* de Forbes, o comparación de *Cloud Computing Providers* de SoftwareInsider o la *The 100 Coolest Cloud Computing Vendors Of 2016*.

4.1. Hipervisores

El hipervisor (o monitor de máquina virtual, VMM, *virtual machine monitor*) es la primera capa que se debe poner o bien sobre el sistema operativo *host* o integrado en este y funcionando como un conjunto indivisible. Esta capa, que en el sentido más amplio puede ser software, *firmware* o hardware, es el responsable de crear y ejecutar máquinas virtuales (llamadas *guest*) sobre el ordenador base (llamado *host*). El hipervisor permite (y gestiona) que las máquinas virtuales se puedan ejecutar simultáneamente, cada uno con su sistema operativo, y compartir los recursos hardware base presentando a cada uno de ellos una plataforma virtual.

Los operativos en cada máquina virtual se ejecutan como si estuvieran sobre un hardware determinado en un entorno cerrado y solo con algunos recursos compartidos (aquellos que el hipervisor habilite) con el sistema operativo *host* (por ejemplo, directorios compartidos entre el disco del SO *host* y el SO *guest*).

Es importante destacar que los SO *guest* pueden ser de diferentes tipos (Linux, Unix, Windows, etc.) en contraposición con la virtualización del nivel del sistema operativo, donde todas las instancias (contenedores) deben compartir un único núcleo (*kernel*), aunque los SO *guest* pueden diferir en el espacio de usuario, como diferentes distribuciones de Linux, pero con el mismo *kernel*.

Más allá de la evolución e historia de los hipervisores desde la década de los sesenta, el punto que marca una diferencia es cuando los dos fabricantes de procesadores de arquitectura x86/x86-64 introducen extensiones hardware para dar soporte a la virtualización (virtualización asistida por hardware), mejorando sus prestaciones y permitiendo una evolución notable de los hipervisores sobre procesadores Intel con las extensiones Intel VT-x (Vanderpool) y de AMD con AMD-V (Pacifica) (se debe destacar que a finales del 2015 Intel representa el 87,7 % del mercado de microprocesadores, y AMD el 12,1 %).

Otra técnica alternativa para la virtualización es la llamada **paravirtualización**, la cual presenta una interfaz software de las máquinas virtuales similar (pero no idéntica) al hardware subyacente. La paravirtualización proporciona *hooks* especialmente definidos para permitir que el SO *guest* y el *host* reconozcan tareas privilegiadas que, si se ejecutan en el dominio virtual, degradarán las prestaciones (básicamente E/S). Esto permite que en una plataforma paravirtualizada, el monitor de máquina virtual (VMM) es más simple y puede reubicar la ejecución de tareas críticas desde el dominio virtual al dominio de *host* y así reducir la degradación del rendimiento del SO *guest*. El único problema de esta opción es que el SO *guest* deberá ser extendido con esta para-API para que disponga de las extensiones necesarias para comunicarse con el hipervisor. Xen, la plataforma más conocida y utilizada por los grandes provee-

dores, implementa esta técnica y soporta ejecución con dos diferentes tipos de *guest*: sistemas paravirtualizados y virtualización completa con asistencia de hardware. Ambos tipos pueden ser utilizados simultáneamente en un mismo sistema Xen y también se pueden usar técnicas de paravirtualización en un *guest* con asistencia hardware.

Los diferentes tipos de plataformas de virtualización utilizadas en la actualidad se pueden clasificar en dos tipos: aquellos en que el hipervisor gestiona el hardware directamente, es decir, hipervisor y SO *host* forman un conjunto indivisible (*bare-metal hypervisor* o *type 1*) o aquellos donde el hipervisor se instala sobre un SO *host* y se ejecuta como una aplicación más de este (*hosted hypervisor* o *type 2*). En orden alfabético:

Tipo 1	Tipo 2
Citrix XenServer	Parallels Desktop for Mac
Linux + extensiones Xen	QEMU
Microsoft Hyper-V	VirtualBox
Oracle VM Server	VMware Workstation Player
VMware vSphere Hypervisor (ESXi)	
Linux+KVM, FreBSD+bhyve	

Como se puede apreciar, hay algunas opciones como Linux's *Kernel-based Virtual Machine* (KVM) y FreeBSD's Bhyve, que son módulos del *kernel*; su instalación sobre Linux o sobre BSD convierten al SO en hipervisor, por lo tanto, de la primera categoría, pero algunos autores, al ser módulos, lo clasifican como de la segunda.

A continuación, extenderemos algunas consideraciones de los más representativos:

1) Xen Project Code: este hipervisor es utilizado en los grandes proveedores como Amazon Web Services (desde 2006), Rackspace Hosting, Verizon Cloud, entre otros, y puede ser instalado desde las distribuciones de Linux que contienen los paquetes, instalar las extensiones o desde una imagen ISO para instalarlo directamente (e incluso algunas distribuciones incluyen extensiones LiveCD para probarlo sin necesidad de instalarlo y también existen versiones comerciales del hipervisor, por ejemplo, Citrix).

Desde los orígenes en la Universidad de Cambridge, la empresa XenSource, su compra por Citrix y su retorno a la Linux Foundation como proyecto colaborativo, este ha pasado por diferentes fases (hoy en https://www.xenproject.org). En relación al *cloud*, en 2009, nace XCP (Xen Cloud Platform), una distribución binaria de Xen Project Management API (o XAPI) y surgen diferentes integraciones del proyecto Xen utilizando XAPI como Eucalyptus, Apache Clou-

dStack, OpenNebula y OpenStack y aparecen los primeros *clouds* públicos con XAPI + Openstack. En 2012, los paquetes XAPI se incluyen en Debian y Ubuntu Server permitiendo hacer los primeros *clouds* utilizando distribuciones Linux. En 2013, XenServer (un *superset* de XCP) es liberado como *open source* por Citrix y disponible en la actualidad en XenServer.org (haciendo obsoleto el proyecto XCP). Hoy en día, el proyecto Xen está centrado en el área de los sistemas operativos de *cloud*. Estos sistemas operativos ligeros y especiales (también conocidos como *unikernels*) no están diseñados para funcionar sobre hardware sino para producir pequeñas máquinas virtuales que pueden generar *clouds* masivos con hardware mínimo. El proyecto Mirage OS es uno de los primeros sistemas operativos de *cloud* en producción pero LING (Erlang-on-Xen) y OSv también conocidos como Xen Project-powered.

2) KVM: es el acrónimo de *Kernel-based Virtual Machine* y es una opción de virtualización total (*full virtualization*) para Linux sobre arquitecturas x86 que dispongan de extensiones Intel VT o AMD-V. Su instalación es por medio de módulos que se insertan en el núcleo del SO *host* y que proveen el entorno de virtualización (*kvm-intel.ko* o *kvm-amd.ko*). Con este se pueden ejecutar múltiples máquinas virtuales (Linux o Windows) sin modificación alguna sobre ellas y donde cada una podrá acceder al hardware virtualizado (red, disco, tarjeta gráfica, etc.). Existe una cierta discusión sobre la vinculación de KVM y QEMU (*Quick Emulator*) y los roles que juegan cada uno de ellos. QEMU es un paquete *open source* que permite la emulación y virtualización de un determinado hardware. Con emulación permite que un programa o SO para una arquitectura, por ejemplo, ARM, pueda ejecutarse en otra, por ejemplo, x86, mientras que en virtualización permite la ejecución del código nativo directamente sobre la CPU *host*. QEMU soporta virtualización cuando se ejecuta bajo un hipervisor Xen o cuando utiliza el módulo de KVM en el *kernel* (utilizándolos como aceleradores y no haciendo servir la emulación).

La razón de esto se explica en cómo se virtualiza la CPU: con las extensiones hardware de los procesadores se crea una *physicalCPU slice* que puede ser mapeada directamente a la CPU Virtual permitiendo una considerable mejora; esto es lo que ocurre, por ejemplo, con el módulo de KVM y es utilizado por QEMU cuando se escoge que el tipo de virtualización es KVM. Si no se utiliza esta «aceleración» (por ejemplo, porque el procesador no dispone de las extensiones hardware), QEMU utiliza TCG (*Tiny Code Generator*) para trasladar y ejecutar instrucciones de la CPU virtual a instrucciones de la CPU física (emulación) con la consiguiente reducción de prestaciones. En cuanto a KVM, utiliza QEMU como programa al crear una instancia de la máquina virtual, y una vez en ejecución, esta será como un proceso regular del SO al cual se le puede aplicar el conjunto de comandos habituales como `top`, `kill`, `taskset` y otras herramientas habituales para gestionar máquinas virtuales.

3) VirtualBox: es un hipervisor *open source*, de tipo 2, muy flexible para arquitecturas x86 y AMD64/Intel64 tanto para uso empresarial como doméstico y se instala sobre Linux, Windows, Macintosh y Solaris; soporta un gran núme-

ro de SO *guest* (Windows, DOS, Linux, OpenSolaris, OS/2, OpenBSD, Android, ChromiunOS y otros más. VirtualBox puede ser complementado con HyperBox (permite gestionar diferentes hipervisores y representa una alternativa libre a productos comerciales como VMware vCenter/ESXi y Citrix XenCenter) o phpVIrtualBox para gestionar las instancias de VirtualBox remotamente. Es una opción muy aceptable para pruebas o pequeñas/medianas instalaciones y permite que las máquinas virtuales puedan ser convertidas a otros formatos (por ejemplo, vmdk de VmWare) y ser reutilizadas en otras infraestructuras. La opción de VirtualBox está tan extendida que existen diversas páginas mantenidas por la comunidad que permiten bajarse las imágenes de diferentes SO/distribuciones ya instaladas y listas para ser cargadas y ejecutadas; entre ellas, se puede mencionar: OSBoxes, Virtualboxes, Oracle.

4) VMware Workstation Player y ESXi: VMware Workstation Pro y VMware Workstation Player (versión gratuita para uso no comercial o doméstico) se han transformado en un estándar en las empresas para ejecutar múltiples SO como máquinas virtuales sobre un PC. Estas pueden ser útiles en diferentes áreas como desarrollo de aplicaciones o replicación de entornos, o también para homogenización de recursos o acceso controlado a ellos para tener un entorno homogéneo corporativo y que puede ser controlado y gestionado desde el departamento de IT de la empresa por herramientas como VMware VSphere/Horizon FLEX. VMware vSphere Hypervisor (ESXi) es un hipervisor *bare-metal* gratuito que permite virtualizar servidores y consolidar aplicaciones con todas las ventajas de la virtualización (reducción de consumo/espacio, inversión de hardware, optimización, etc.). ESXi dispone de una consola para su gestión y control, pero no dispone de herramientas para la gestión y control del entorno virtual. Para ellos se debe acceder a través de vSphere Client (VMware Infrastructure Client), que se descarga de la página de VMware (en las últimas versiones dispone de una servicio web que reemplaza al cliente y que se verá en el siguiente módulo). Para la gestión y control de diferentes servidores ESXi, gestión en caliente de las máquinas virtuales, el almacenamiento y demás opciones de monitorización y control se debe recurrir a las herramientas propietarias de VMware que están disponibles bajo licencia.

5) Microsoft Hiper-V: es un hipervisor para sistemas de 64 bits con procesadores que disponen de extensiones hardware: VT-x y AMD-V (no obstante, los programas de gestión se pueden instalar sobre sistemas x86). Desde la versión incluida en Windows Server 2008 R2, el hipervisor incorpora funcionalidades extendidas como migración en caliente de las máquinas virtuales (*live migration*), almacenamiento en máquinas virtuales dinámicas, compatibilidad mejorada con procesadores y redes, soporte para Linux (Debian, Ubuntu, Centos/RH, Oracle, SuSE), FreeBSD, y obviamente Windows (desde Vista hasta W10 incluyendo *servers* desde 2008). Existen algunas diferencias entre Hyper-V ejecutándose por ejemplo, en Windows 10 y sobre Windows Server 2012, que se centran en la gestión de la memoria, la virtualización de GPU, la migración en caliente, réplicas, o compartición de VHDX (discos virtuales), pero por el resto son iguales (sobre Windows 10 estas extensiones reemplazan

a un producto anterior de virtualización llamado Virtual PC). Su instalación es simple tanto en W10 como en WS2012 y permite que las máquinas creadas puedan ser exportadas a otros entornos Hyper-V o directamente a Azure (plataforma de *cloud* público de Microsoft).

Microsoft también dispone de una versión llamada Hyper-V Server gratuita que incorpora un Windows Server con funcionalidad limitada + los componentes de Hyper-V (solo este rol se encuentra activo). Hyper-V Server solo dispone de interacción a través de la línea de comandos para configurar el *host*, el hardware y el software y soporta el acceso remoto a través de Remote Desktop Connection. La configuración del *host* y las máquinas virtuales puede ser realizada a través de la red utilizando programas tales como *System Center Virtual Machine Manager*, que permiten de forma fácil configurar y monitorizar el servidor. Hyper-V Server necesita menos requerimientos de memoria y disco que un Windows Server, pero está limitado a 64 máquinas virtuales cuando en un Windows Server es de 1.024.

6) Proxmox: con las facilidades de KVM y otras tecnologías, han salido diferentes soluciones como el Proxmox VE, que permite tener un entorno virtualizado con KVM y Linux Containers (LXC). Esta solución completa de virtualización *open source* es adecuada para muchas empresas ya que permite gestionar y configurar máquinas virtuales sobre la base de KVM, contenedores Linux (LXC), almacenamiento y redes virtualizadas y clústeres de alta disponibilidad. Su administración y monitorización se realiza a través de una interfaz web y soporta tanto SO *guest* como Linux y Windows en máquinas virtuales o contenedores como virtualización de sistema Linux (a nivel del SO). Esta distribución es un hipervisor tipo 1 (*bare-metal*); está basado en la combinación de Debian+KVM+LXC, soporta migración en caliente (*live migration*), permite configurar clústeres de alta disponibilidad y puede interactuar con diferentes sistemas de archivos locales o remotos (NFS, iSCSI, Ceph, GlusterFS, etc.).

La siguiente tabla muestra la distribución de hipervisores (por defecto, ya que se pueden negociar otros servicios, como por ejemplo *bare-metal servers ad-hoc*) sobre las diez empresas más significativas del informe de Gartner del 2016 [Cis16].

Proveedor	Hipervisor/es
AWS	Xen (EC2 Container Service: Docker)
Century Link	VMware
Fujitsu-Global Fujitsu-K5	Xen KVM (Openstack)
Google Compute	KVM
IBM-Softlayer	Citrix-XenServer
Microsoft Azure	Hyper-V

Proveedor	Hipervisor/es
NTT Com NTT Com Enterprise	KVM (CloudStack) VMware
Rackspace	Citrix-XenServer (default), VMware, KVM(Openstack), Hyper-V
Virtustream	VMware y KVM
VMware	VMware

Un punto importante sobre los hipervisores, como ya se mencionó anteriormente, son la evolución de estos hacia la virtualización del sistema operativo y los *containers*. Desde hace un par de años existe una disputa sobre los hipervisores y contenedores en relación con las prestaciones y otros parámetros de eficiencia, sobre todo en el *cloud*. [Cvh14]

Además, en este sentido, Ubuntu ha hecho una apuesta por un contenedor (LXD) aplicando la idea del «contenedor» al *cloud* y promocionando este como *The LXD container hypervisor*, con 10x la densidad de ESX, 25 % más rápido y 0 latencia, e indicando que se pueden mover las MV a los contenedores, fácilmente y sin modificar las aplicaciones u operaciones y donde LXD es un hipervisor de contenedor puro que ejecuta sistemas operativos y aplicaciones Linux no modificados con operaciones de estilo VM a una velocidad y densidad increíbles.

Ejemplo de esta tendencia (pero no es el único) es Google, que ha invertido en contenedores desde el principio; cualquier cosa que se haga en su plataforma (búsqueda, Gmail, Google Docs) es un contendor por cada servicio. Se podría pensar que hay muchos tipos diferentes de contenedores donde probar, pero, según los expertos, en su gran mayoría todos tienen el mismo código en la parte inferior (desde Google/LXD/Docker con *cgroups/namespaces* o *bean-counters* en OpenVZ) y además se han ido convergiendo y en la actualidad no hay prácticamente diferencias entre ellos.

Dos de los entornos de contenedores más representativos son Docker, LXC/LXD; cuentan con ecosistemas que permiten obtener los *containers* como VA pero sin la carga que representa la MV y todo el sistema operativo *guest* y sus dependencias. Según Ubuntu, LXD y Docker son complementarios ya que Docker está centrado en aplicaciones y LXD en *host machine containers* (este actúa como una máquina virtual) permitiendo ejecutar y gestionar contenedores Docker dentro de contenedores LXD haciendo a Docker aún mejor ya que LXD (en relación con KVM obtiene ×10 en prestaciones, ×14.5 en relación con la densidad, 57 % menos de latencia y 94 % más rápido en la carga). Es habitual que los desarrolladores de aplicaciones provean el contendor para su aplicación y las instrucciones para su funcionamiento.

De esta forma, con Docker en la parte superior de LXC/LXD y dadas las ventajas de su *open source*, se puede empaquetar, enviar y ejecutar cualquier aplicación como un contenedor LXC/LXD ligero, portátil y autosuficiente que se ejecuta prácticamente en cualquier lugar. Con ello, los desarrolladores tienen una herramienta potente para distribuir su código, el cual podrá ser ejecutado por el usuario sin intervención/configuración prácticamente y desplegado rápidamente en el *cloud* con una portabilidad total y sin invertir el tiempo que cuesta tener una VM correctamente configurada y enviarla o desplegarla. Según los expertos, esta nueva tendencia inicia un nuevo paradigma de virtualización (que algunos, como Ubuntu, denominan *container hypervisors*) que aporta una nueva forma de empaquetar aplicaciones teniendo servicios en el *cloud* que pueden entrar en cualquier dispositivo, permitiendo en un futuro cercano mover aplicaciones desde una plataforma a otra haciendo realidad el objetivo –pocas veces real– de la «interoperabilidad».

Una muestra de ello es el crecimiento de los ecosistemas creados alrededor de los contenedores; por ejemplo, Docker cuenta con un ecosistema oficial, DockerHub, que actúa como repositorio de las distribuciones oficiales o contenedores de la propia comunidad que pueden ser descargados y puestos en marcha en cuestión de minutos. Lo mismo podemos encontrar para LCC/LXD o también contenedores creados por particulares y puestos a disposición de la comunidad, por ejemplo, Flockport, donde se demuestra que LXD también puede centrarse en aplicaciones y no es terreno totalmente propio de Docker (que no obstante continúa teniendo un papel protagonista).

4.2. Imágenes virtuales (*virtual software appliances/cloud appliances*)

Una vez instalado y configurado el hipervisor, es necesario comenzar a desplegar (instalar y configurar y poner en marcha) las máquinas virtuales (MV) que cubran las necesidades de los usuarios.

El primer paso es escoger la distribución de sistema operativo (por ejemplo, desde Distrowatch) que mejor se adecúe y sobre este instalar y configurar todos los servicios necesarios. Esta tarea implica gran cantidad de tiempo y conocimientos ya que algunas aplicaciones, servidores o entornos requieren una gran cantidad de pasos previos de prerrequisitos (SO, librerías, aplicaciones adicionales, etc.) que se deben cumplir para que luego el software deseado funcione correctamente.

Consideremos el caso de querer probar una versión de Liferay para evaluar si se adapta a las necesidades del cliente en la versión CE basada sobre el servidor de aplicaciones Tomcat. Liferay es un gestor de alto desempeño de portales corporativos que permite la gestión de contenidos y documentos, colaboración entre usuarios y vinculación a redes sociales con incorporación de aplicaciones y totalmente adaptable y configurable a las necesidades de los clientes.

Entre los pasos de preinstalación se necesita el *framework* de Java, un servidor de bases de datos, un servidor de aplicaciones (Tomcat o WildFly), además de un servidor de correo y la configuración de ElasticSearch que se deberá instalar por separado. Además, Tomcat necesita Ant y CVS y su instalación también consume tiempo.

Después de instalados estos paquetes se puede seguir el proceso de instalación de Liferay y su configuración entre los diferentes paquetes y pruebas para verificar que todo funciona. Obviamente, más allá del tiempo consumido, no es una tarea fácil para un no experto en Liferay-Tomcat-DB-ElasticSearch y llevará un cierto tiempo y dedicación tener todo a punto para realizar una prueba, incluido a un experto en IT (que no tenga experiencia específica en estos paquetes).

Una opción que ha cobrado fuerza para acercar el software a empresas u organizaciones que no deseen invertir todo este tiempo en hacer una prueba es la de ofrecer a los usuarios un paquete preconfigurado del software (normalmente llamado *bundle*) con todas las aplicaciones y que solo se debe reconectar y adecuar, por ejemplo, en el caso de Liferay, a la base de datos, al servidor de aplicaciones y al servidor de búsquedas (ver *Deploying Liferay Portal*).

Más allá de las ventajas de la encapsulación de paquetes (*bundles*) y con las facilidades provistas por la virtualización y el *cloud*, ha irrumpido con fuerza una nueva tendencia, que es la de proveer imágenes con el software configurado y listo para usar. Estas imágenes, llamadas ***appliances***, permiten de forma fácil y simple disponer de un entorno cerrado (*sandbox*) para que rápidamente (en cuestión de minutos) se pueda probar y evaluar el software deseado sin todo el proceso de instalación y configuración del software y con solo unos pequeños pasos de adecuación al entorno si es necesario.

> Por ejemplo, para el caso de Liferay, podemos acceder a Bitnami o SystemSector, que ofrecen una gran cantidad de *appliances* de diferentes software *open source* (o propietarios vinculados a una licencia o un versión de pruebas) realizadas por expertos y para diferentes hipervisores (por ejemplo, en el caso de Liferay, sobre Bitnami o SystemSector para VMware y Virtualbox).

Este tipo de software *appliances* está en un momento de auge, ya que los responsables de IT no desean gastar gran cantidad de tiempo y esfuerzo en enfrentarse a cuestiones específicas de la instalación del software (*kernel* adecuado, librerías, dependencias, etc.) que pueden hacer que un servicio quede inestable o inseguro por omisión o no configuración de una parte crítica ya que no se es experto en todo lo que significa las particularidades del mismo (y que en algunos casos son muy complejas).

Todo esto significa que se desista de este paquete (por las dificultades añadidas que representa), lo cual representa un problema para los desarrolladores del paquete ya que no tiene aceptación por las dificultades que conlleva tener una

instalación de pruebas simple, fácil y sin riesgos. Esto genera frustración para los posibles usuarios del paquete (incluso personal de IT experimentado) y un fracaso para sus desarrolladores.

Todo esto se soluciona con una *virtual appliance* (VA), que se puede desplegar en una infraestructura con un hipervisor o sobre un *cloud* público y que contiene el sistema operativo, el software de aplicación y las dependencias necesarias, la configuración y los archivos de datos necesarios para el funcionamiento inicial y todo listo para funcionar, y además en una gran variedad de formatos para diferentes hipervisores (KVM, Xen, VMware, Virtualbox, etc.), o bien una imagen ISO para instalar o para un USB o para un *cloud* (por ejemplo, una *Amazon Machine Image* –AMI– para que pueda ser desplegada sobre EC2, *Elastic Compute Cloud*).

En el caso de las *virtual appliances* (VA), es el propio desarrollador o expertos que empaquetan, configuran y cuidan los detalles del software con el que desean que sus clientes hagan sus pruebas de concepto; el valor que conlleva es evidente: cualquier usuario, sin grandes conocimientos de IT, puede probar un determinado paquete y tomar decisiones si se adapta a sus necesidades o no.

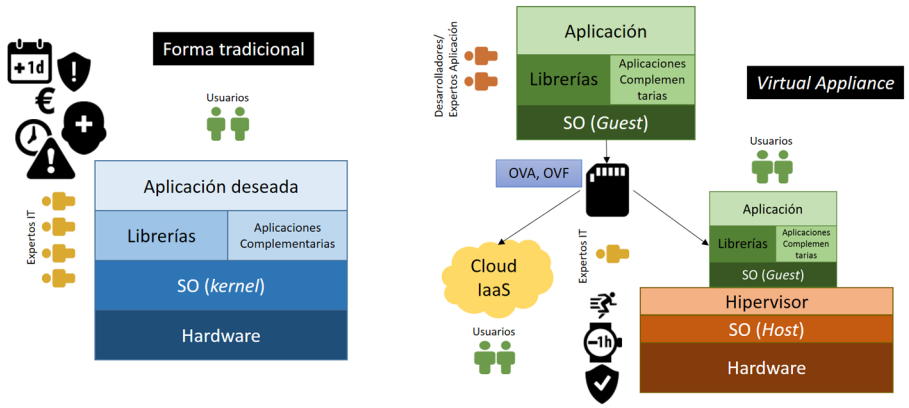
Esto representa, para los desarrolladores, un notable incremento de responsables/usuarios que prueban y evalúan este software, que si no fuera así probablemente hubieran desistido si no contaran con un grupo de IT experto y tiempo/recursos para desplegarlo, aunque solo fuera como prueba de concepto.

> Algunos autores [Cca11] utilizan una analogía equivalente al mundo del automotor: un usuario que desee un coche va a un concesionario y compra el coche de acuerdo a sus necesidades y posibilidades y sale conduciendo teniendo la seguridad de que todo funcionará y no deberá preocuparse por la seguridad si, por ejemplo, el ABS (sistema de ayuda a la frenada) está bien configurado. Seguramente, un usuario que quisiera comprar todas las piezas y construirlo en función de sus conocimientos también podría hacerlo, pero es indudable que el tiempo invertido sería mucho mayor y muchos aspectos dependerían de la habilidad/conocimientos de este. La virtualización y el *cloud* han permitido que la gran complejidad de la infraestructura quede oculta y sea fácil «probar y conducir» nuevas opciones que le pueden ser útiles dejando que la complejidad de la instalación y configuración quede en manos del desarrollador/experto.

Esta idea, como veremos más adelante, es la que da soporte a las propuestas de SaaS, que se han transformado en una opción muy viable para los desarrolladores de software dentro del mundo del *cloud*.

Si tenemos en cuenta estas ideas, se puede concluir que no es más que una evolución a la idea del SO, ya que de este no es necesario obtener las fuentes y compilarlo/configurarlo, sino que ya viene en una imagen (o directamente en un disco virtual para ejecutarlo desde un hipervisor) con todas las utilidades y software básico necesario para el funcionamiento, y además, el software extra o específico necesario se puede obtener desde un repositorio, creado por usuarios expertos en esta distribución y en la aplicación, a través de un comando simple; por ejemplo, para instalar un servidor de base de datos en Debian, solo debemos ejecutar `apt-get install mysql-server`, responder a las preguntas que nos realizará el instalador y tendremos el servicio funcio-

nando sobre esta distribución (solo se deberán hacer unos pequeños ajustes para acabar de sintonizar la aplicación a las necesidades de los usuarios). La figura siguiente resume las ideas explicadas en este apartado:



El formato más habitual para las *virtual* (*software*) *appliances* es el OVF (*Open Virtualization Format*). Este formato es un *open standard* para empaquetar y distribuir *virtual appliances* en forma abierta, segura, eficiente y portable sin dependencias especiales de un hipervisor o arquitectura del procesador. Inicialmente (2007) fue una propuesta de VMware, Dell, HP, IBM, Microsoft y XenSource enviado a la DMTF (*Distributed Management Task Force*), la cual ha normalizado el formato y liberado diferentes versiones (la última en 2015 V2.1.1) incluyendo mejoras (por ejemplo, en la 2.0, pensada especialmente para el *cloud*, con la especificación de la red y encriptación para mejorar la seguridad en el envío de *appliances*) y posteriormente ha sido adoptada como ANSI *standard* (2010) e ISO (*International Organization for Standardization*) en IEC (*International Electrotechnical Commission*) en 2011.

Un paquete OVF consiste de varios archivos en un directorio que incluye un archivo XML con extensión **.ovf**, el cual incluye metadatos tales como nombre, hardware requerido, referencias a otros archivos y una (o más) imagen de disco (en formato *raw*, que es una copia binaria sector-por-sector del medio fuente). Este directorio se puede empaquetar como un único archivo con extensión **.ova** (*open virtual appliance or application*), el cual es un archivo `tar` del directorio OVF. Hoy en día, es soportado por Amazon Elastic Compute Cloud, IBM SmartCloud, Microsoft System Center Virtual Machine Manager, OpenNode, Oracle VM, rPath, Red Hat Virtualization, SUSE Studio, VMware, VirtualBox y XenServer Citrix, entre otros [Ovf15].

Otro formato aceptado por gran parte de los proveedores es el **VMDK** (*Virtual Machine Disk*), desarrollado por VMware; actualmente es un formato abierto (VMware provee un paquete VDDK –librerías, documentación y ejemplos– para crear o acceder a archivos vmdk y una herramienta –`ovftool`– para convertir vmdk a ovf). Inicialmente soportaba imágenes de 2TB, pero en la última

actualización ha sido extendido hasta 62TB y existen herramientas *open source* como `qemu-img` que permiten hacer la conversión de formatos para adecuarlos a los diferentes hipervisores.

El formato VHD (*Virtual Hard Disk*) también utilizado por algunos proveedores de *virtual appliances* orientado a sistemas Windows® y representa un virtual *hard disk drive* (HDD). Fue creado para Virtual PC por Connectix y pasó a ser propiedad de Microsoft en 2003 y desde 2005, Microsoft tiene disponible a terceros la especificación de VHD con la promesa de hacerlo en un futuro parte de Microsoft Open Specification.

Finalmente, otro formato utilizado para algunas VA es el **qcow/qcow2** (QEMU *Copy On Write*) utilizado por QEMU/KVM, que utiliza una estrategia de optimización que demora el almacenamiento sobre el disco hasta que este es necesario, por lo cual los formatos son más pequeños que un *raw* ya que no se almacena espacio vacío. Este formato permite agregar nuevos datos a un archivo ya existente, soporta copias o instantáneas (*snapshots*) parciales múltiples, comprensión basada en `zlib` y cifrado mediante AES. Uno de los problemas de este formato es que no puede ser montado directamente como los discos *raw* y es necesario una utilidad que primero lea el archivo antes de que pueda ser montado.

En el caso de los sistemas operativos, las imágenes de la distribución se proveen generalmente como ISO (formato basado en el estándar ISO 9660 o el protocolo *Universal Disk Format* (UDF) creado como un formato solo de lectura y que mantiene la estructura original y del cual existen diversas variantes) y es el utilizado para crear CD/DVD, pero también se puede crear desde un CD/DVD con `dd if=/dev/cdrom of=/tmp/cd.iso` o con `genisoimage` si los archivos están en un directorio o montar como si fuera un directorio con `mount -o loop archivo.iso /mnt`.

Esta imagen nos será útil para arrancar sistema operativo desde un CD/DVD o USB o desde una MV y cualquier modificación posterior que se haga sobre este SO no quedará reflejada en la ISO (son utilizadas para instalar una distribución generalmente).

No obstante, existen gran cantidad de repositorios que publican SO ya instalados como VA listos para funcionar; entre ellos, podemos enumerar OSBoxes (para VirtualBox y VMware), virtualboxes.org, virtualboximages, VMA además de que la mayoría de las distribuciones ya ofrecen sus imágenes en diferentes formatos preparados para el *cloud* (por ejemplo, Ubuntu y Centos).

Con el auge de la VA han surgido una serie de sitios que proveen VA con diferentes configuraciones, o incluso alguna de ellas que permite la creación y adaptación de las VA de acuerdo a unas necesidades específicas o con una configuración predeterminada para diferentes hipervisores.

A continuación, mostraremos algunas de ellas.

a) Bitnami: es una plataforma que permite disponer de VA en diferentes formatos para ser ejecutadas tanto el local (como VM) o *containers* o directamente en el *cloud* (AWS, GoogleCloud, Azure, OracleCP, 1&1, CenturyLink o GoDaddy). Esta plataforma incluye más de ciento cincuenta aplicaciones o *stacks* (diferentes paquetes de software agrupados como LAMP, WAMP, WAPP, MAMP, LAPP o MAPP) actualizadas y seguras, optimizadas, consistentes (significa que implementar una aplicación como una máquina virtual o iniciarla en la nube es prácticamente lo mismo, por lo que es sumamente fácil pasar del desarrollo a producción en el *cloud*).

Bitnami ofrece *containers* que contienen las últimas versiones de la aplicación (aplicación + librerías + dependencias) que pueden ser accedidos de diferentes modos o incluso ser construidos de acuerdo a las necesidades del usuario con Stacksmith, También puede desplegar las VA en el *cloud* ajustando con las particularidades de cada proveedor dando la mejor opción entre arquitecturas sofisticadas y escalabilidad.

Pero, además, si se desea ejecutar en MV o instalar en un SO local, se pueden utilizar las imágenes o los binarios que utilizan los mismos componentes que las versiones de *cloud* y los contenedores permitiendo una coherencia entre todas las aplicaciones.

b) SuSEstudio: es una plataforma en línea desarrollada por SuSE para la creación de VA basada en esta distribución del SO (se puede escoger ente openSuSE y SuSEEnterprise, con licencia). Los usuarios pueden crear su propio SO, su *softwareappliance* o *virtual appliance*, seleccionado qué aplicaciones o paquetes desean tener en su Linux «tuneado».

La plataforma también ofrece una serie de imágenes preconfiguradas (jeOS, *minimal server*, escritorios GNOME y KDE, entre otros) y diferentes formatos (Live CD/DVD/ISO, VMDK, VirtualBox, VHD/Azure, USB, Xen, KVM, OVF, Amazon EC2 (AMI), PXEBoot (*onsite version only*) para ser descargados o enviados directamente al *cloud* (Azure, Amazon EC2 SUSE Cloud).

Un aspecto interesante es que SuSESTudio también permite clonar y probar la imagen generada sobre el mismo sitio, validar lo que desea el usuario, y si contiene todos los elementos necesarios antes de generar la imagen definitiva o bajar simplemente la VA creada por otros usuarios en el formato deseado.

c) Turnkey: la Turnkey *Linux Virtual Appliance Library* es un proyecto *open source* que provee de VA (100+) basadas en Debian y que pueden ser ejecutadas como VM, contenedores o desplegadas en el *cloud* (disponibles en un amplio rango de formatos, por ejemplo, ISO, ova, vmdk, Openstack, Promox, Xen LXC, Docker).

Turnkey también está conectada a Amazon y permite a través del TurnkeyHub desplegar las máquinas con un mínimo costo y con VA totalmente adaptadas a esta infraestructura. La base (TurkeyCore) para todas las VA incluyen Debian 8, con actualizaciones diarias, *1-click backup and restore*, Dynamic DNS, NTP, LVM, AJAX web Shell –provee acceso a una terminal desde un navegador– *Web Management Interface* (*Webmin*).

d) OZ: si bien existen importantes herramientas de despliegue (Ansible, Chef, Puppet), esta aplicación permite la creación automática de VA de sistemas operativos con una mínima intervención de usuario y adaptables a las necesidades que este indique en un archivo de configuración en formato xml.

Oz puede instalar el SO, customizarlo, o también generar el *package manifests*. OZ da soporte para generar VA de RHEL, CentOS, Scientific Linux, OEL, Fedora, OpenSUSE, Debian, Ubuntu, Mandrake, Mandriva, Mageia, FreeBSD y Windows en diferentes versiones. Un aspecto muy importante que presenta OZ, a diferencia de otras aplicaciones similares, es su facilidad de instalación y utilización y que la instalación realizada será idéntica a que si esta se hubiera realizado desde la ISO de la distribución sobre una máquina física (*bare-metal*).

e) BoxGrinder: es un proyecto para crear VA desde un archivo de texto, pero su última versión estable es de 2012. El usuario debe crear *appliance definition file* a partir un esquema (archivo escrito en YAML) y luego ejecutar *boxgrinder* que bajará todos los elementos necesarios, construirá la instancia en el formato seleccionado y descargará este en la plataforma indicada. BoxGrinder soporta plataformas *cloud* (EC2, Xen, KVM, VMware) y puede utilizar como SO base Fedora, RHE o CentOS o también se puede escribir el *plugin* para dar soporte a otra plataforma de virtualización o SO. Si bien es una herramienta interesante, su falta de información y detalle sobre la actividad del proyecto hace necesario tener cierta precaución con las VA generadas, ya que pueden estar desactualizadas o ser no adecuadas.

Como ya hemos mencionado anteriormente, la virtualización del sistema operativo ha llevado a crear una evolución de las *virtual appliances*, ya que en este caso se utilizan los llamados contenedores que solo incluyen el software de aplicación y las dependencias utilizando el SO *host* como SO del contenedor. Dada su importancia en el *cloud* y en los entornos DevOps, estos temas serán tratados en futuros módulos.

También en el mismo sentido de la VA es importante mencionar Vagrant, que es una herramienta orientada a desarrolladores que proporciona ambientes de trabajo (basado en MV) fáciles de configurar, reproducibles y portátiles construidos sobre la tecnología estándar y controlados por un solo flujo de trabajo ayudando a maximizar la productividad y flexibilidad.

Vagrant, que puede ser considerado como administrador de MV, utiliza *provisioners* (aprovisionadores) y *providers* (proveedores) como actores básicos para administrar los entornos de desarrollo. Los *provisioners* son herramientas que permiten a los usuarios personalizar la configuración de las MV tales como Puppet, Ansible o Chef, que son las más utilizadas en el ecosistema Vagrant. Los *providers* son los servicios que Vagrant utiliza para crear e inicializar los entornos virtuales (MV) y algunos de los proveedores más utilizados son VirtualBox, Amazon AWS, VMWare y Docker. Es decir, Vagrant está por encima de software de virtualización como un *wrapper* y ayuda al desarrollador a interactuar fácilmente con los *providers*; los requisitos de MV y software se escriben en un archivo denominado *Vagrantfile* que tendrá los pasos a llevar a cabo para crear un sistema integrado de desarrollo listo para usar en una MV (*development ready box*).

Box es el formato (extensión **.box**) para los entornos Vagrant y puede ser utilizado por cualquiera sobre cualquier plataforma que esté soportada por Vagrant garantizando que el entorno será exactamente igual en uno u otro sitio (es decir, copiando el *box* tendré el mismo entorno en una máquina que en la original). Los *boxes* pueden ser compartidos en un entorno oficial de Vagrant llamado Atlas que permite que los *boxes* (una especie de VA) puedan ser clasificados por *providers*, arquitectura o SO para acceder rápidamente a ellos (consultar los *Bentos boxes*).

Otros repositorios de VA

Además de estos repositorios (generalistas), existen otros repositorios de VA orientados hacia un hipervisor determinado: VMware Solution Exchange, Xenserver para estas plataformas, o de los propios desarrolladores, OpenVAS, Oracle, entre otros.

4.3. IaaS (*infrastructure as a service*)

Dentro de las plataformas *open source* que podemos encontrar con licencia Apache, GPL o similares tenemos (orden alfabético):

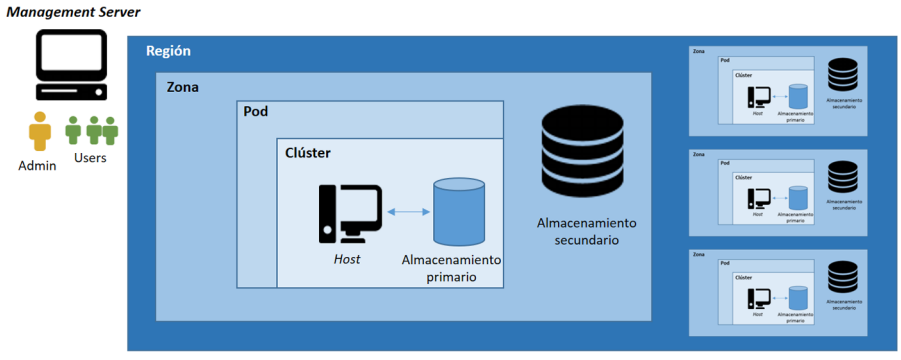
a) Apache Cloud Stack: es una plataforma IaaS *open source* (desde 2010 parcialmente bajo GPL y desde 2011 bajo ASL, ver su historia al inicio) orientada a crear, manejar y aprovisionar infraestructura *cloud* (recursos de cómputo, *pools* de almacenamiento y redes) bajo diferentes supervisores (KVM, LXC, vSphere –via vCenter–, Xenserver, Xen Project, Hyper-V). Con ello permite inicializar un servicio elástico de *cloud clomputing* y permitir a los usuarios finales aprovisionar los recursos siendo capaz de manejar centenares de servidores físicos geográficamente distribuidos en centros de datos y con muy buen escalado (casi linealmente) de servidor de administración, evitando así la necesidad de servidores de gestión a nivel de clústeres. Cloudstatck configura automáticamente las redes y el almacenamiento para cada despliegue de MV, las cuales son provistas de un *pool* de VA dentro del mismo servidor. Estas VA disponen de servicios de *firewalling*, *routing*, *DHCP*, *VPN*, *console proxy*, *storage access*, y

storage replication, ofreciendo además una interfaz web (que puede ser tuneada) para el aprovisionamiento y administración del *cloud*, así como un interfaz de usuario (también web) utilizada para ejecutar y gestionar las MV.

Esta plataforma provee una *REST-like API* para operación y administración del *cloud* y da soporte a la *EC2 API* permitiendo que herramientas de EC2 puedan ser utilizadas en ella. La instalación mínima consiste de una máquina ejecutando *CloudStackManagement Server* y otra máquina que actúa como infraestructura *cloud* (que no es más que una máquina ejecutando el hipervisor), aunque para pruebas de concepto se puede montar todo en una única máquina (*ManagementServer* y *KVM hypervisor host*).

El *management server* es el (único) punto de configuración y puede ejecutarse en una máquina dedicada o en una MV y controla el despliegue de MV en los *hosts* (asigna IP y almacenamiento a cada instancia), se ejecuta sobre Apache Tomcat y requiere MySQL. La arquitectura de Apache CloudStack diferencia entre: regiones (colección de zonas geográficamente cercanas manejadas por uno o más *management servers*), zonas (equivalente a un único *datacenter* con uno o más clústeres –*pods*– y almacenamiento secundario), *pods* (*racks* con *switchs layer-2* y uno o más clústeres), clústeres (uno o más *hosts* homogéneos y almacenamiento primario), *host* (único ordenador en un *cluster* –hipervisor–), almacenamiento primario (el que provee un clúster para la ejecución de las imágenes) y almacenamiento secundario (recurso masivo de almacenamiento). En cuanto a la red, se ofrecen diferentes tipos, pero se pueden clasificar en dos escenarios concretos: básico (similar a la clásica de AWS donde se provee una red *layer-2* y se aísla el *guest* en *layer-3* con el *bridge* sobre los hipervisores), avanzado (utiliza aislación en *layer-2*: VLANs).

Como requerimientos esta plataforma puede instalarse (v4.9) sobre CentOS/RHEL 6.3+ o Ubuntu 14.04 de 64b, 4 GB RAM, 250 GB disco (500 GB recomendado), 1 NIC + FQDN y para el *host*/hipervisor debe tener extensiones hardware (Intel-VT, AMD-V), CPU X86 de 64 bits, 4 GB RAM, 36 GB de disco y un NIC. La figura siguiente muestra la arquitectura de esta plataforma:



b) Eucaliptus: esta plataforma *open source* (GPL/ASL) permite construir *clouds* privados e híbridos compatibles con AWS (desde 2012). Eucalyptus es el acrónimo de *Elastic Utility Computing Architecture for Linking Your Programs To Useful*

Systems y permite que los recursos de cómputo, almacenamiento y redes puedan ser asignados y escalados en función de las necesidades de carga. Esta plataforma comienza a tomar forma dentro del proyecto *Virtual Grid Application Development Software* [2003-2008] en la Universidad de Rice. Un hito importante dentro del desarrollo ocurrió cuando, en el año 2009, fue incluida dentro de repositorio de Ubuntu 9.04, formando parte de la estrategia de Ubuntu Enterprise Cloud (UEC), donde esta plataforma fue la protagonista hasta la versión 11.10. En 2011 UEC es reemplazado por Ubuntu Cloud Infrastructure y adopta como plataforma OpenStack.

Eucalyptus Systems (empresa con capital-riesgo) se crea en 2009 para potenciar su desarrollo y, en septiembre de 2014, esta pasa a ser propiedad de Hewlett Packard; el software parte de la estrategia HP Helion como portfolio de *cloud computing* empresariales basados en *open source* (junto con HPE Helion OpenStack y HPE Helion Stackato como PaaS basado en *Cloud Foundry*) y el producto renombrado HPE Helion Eucalyptus. Como requerimientos necesita CentOS 7 o RHEL 7 de 64 bits y todos los servicios deben ser instalados sobre servidores físicos (no sobre MV).

Eucalyptus está basado en los siguientes componentes:

- *Cloud Controller* (CLC) es el punto de entrada en la nube para administradores, desarrolladores, gestores de proyectos y usuarios finales. El CLC maneja la persistencia y es el *backend* para el UFS (servicios de usuario, UFS, que pueden encontrarse en la misma máquina). Un *cloud* de Eucalyptus puede tener el mismo CLC. Es un programa Java que ofrece interfaces compatibles con EC2 o web y representa la interfaz administrativa permitiendo la gestión de recursos y el sistema de *accounting*, así como procesar las peticiones a la API por CLI (por ejemplo, de `euca2ools`) o GUI (como Eucalyptus *User Console*).

- *User-Facing Services* (UFS) sirven como puntos finales para los servicios compatibles con AWS ofrecidos por Eucalyptus: EC2 (computación), AS (*AutoScaling*), CW (*CloudWatch*), ELB (*LoadBalancing*), IAM (Euare) y STS.

- *Object Storage Provider* (OSP) es el proveedor de almacenamiento y puede ser Eucalyptus Walrus, Riak CS o Ceph-RGW. Walrus es el equivalente al AWS-S3 y ofrece el almacenamiento de las MV y puede ser utilizado como servicio de almacenamiento simple mediante *HTTP put/get*.

- *Management Console* es una interfaz web fácil de usar que le permite administrar el *cloud* de Eucaliptus (1 o más), y generalmente se encuentra en la misma máquina de UFS.

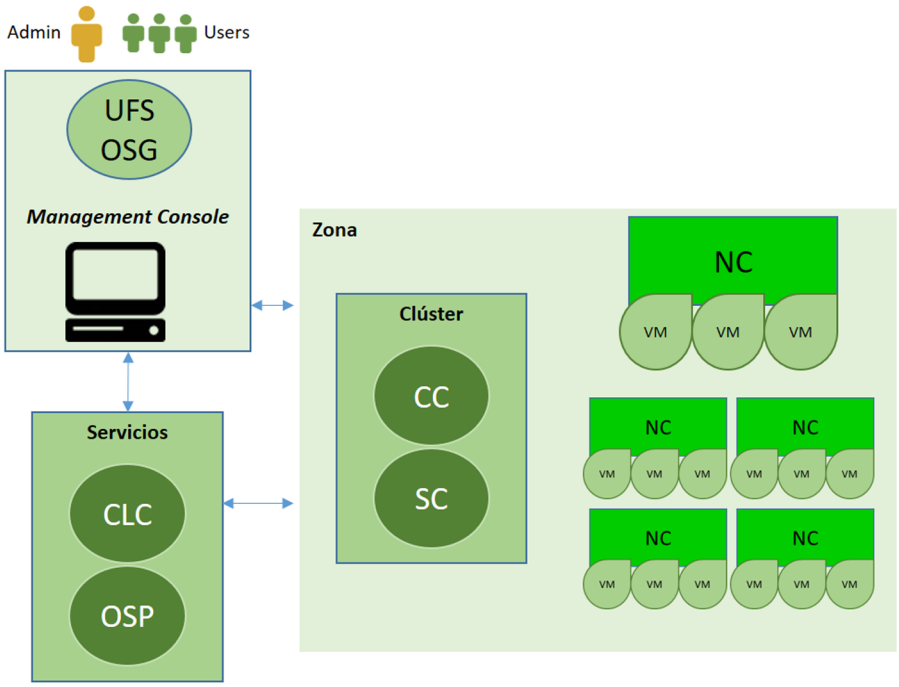
- *Cluster Controler* (CC) debe ejecutarse en una máquina *host* que tenga conectividad de red con las máquinas que ejecutan los controladores de nodo (NC) ya que recopilan información sobre estos y planifican la ejecución

de las máquinas virtuales (MV) en NC específicos. Todas las NC asociadas con un solo CC deben estar en la misma subred.

- *Storage Controller* (SC) es equivalente a Amazon *Elastic Block Store* (EBS) y puede interactuar con varios sistemas de almacenamiento exportando los volúmenes de almacenamiento y que podrán ser conectados a las máquinas virtuales y montados o accedidos como un dispositivo *raw*.

- *Node Controller* (NC) es el que se ejecuta en cualquier máquina que aloja instancias de VM. El NC controla las actividades de VM, incluyendo la ejecución, monitorización y terminación de instancias de VM.

La plataforma incluye también Euca2ools, que son herramientas CLI para interactuar con los WS que exportan una REST/*Query-based* API compatible con los servicies EC2/S3 y que también funcionan con Eucalyptus (son equivalentes a las de Amazon *api-tools* y *ami-tools*) y aceptan las mismas variables y parámetros. La figura siguiente muestra la arquitectura de Eucalyptus.

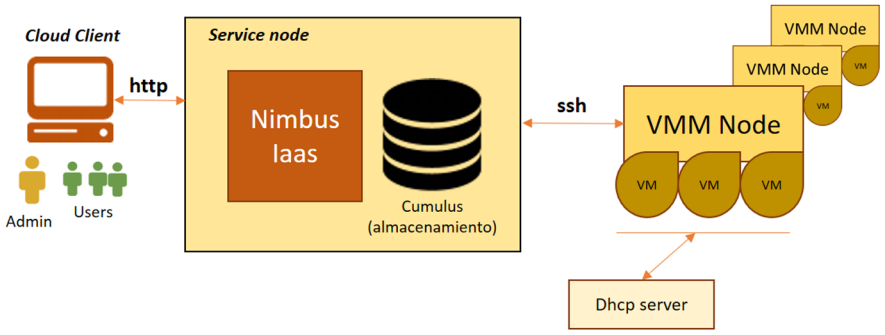


c) Nimbus: son un conjunto de herramientas *open source* (ASL) enfocado en proveer capacidades de IaaS como *clouds* privados o comunitarios incorporando recursos externos (MV) mediante Nimbus *Workspace Service* y gestionando, utilizando Cumulus, un almacenamiento basado en cuotas sobre el *cloud*. Además, utilizando Nimbus *Context Broker*, se pueden crear configuraciones seguras basadas en contextos o utilizar herramientas de escalado para adecuar la infraestructura a la carga dinámicamente a través de los proveedores, trabajar con diferentes hipervisores (Xen o KVM), incluir gestión de recursos locales (por ejemplo, sistemas como PBS) y API (incluye compatibilidad con EC2). Los

requerimientos para su instalación son los habituales y solo es necesario cumplir con las versiones de determinados paquetes y que los servicios de Nimbus estén sobre el mismo *host* que Cumulus (a partir de v2.6).

La plataforma se complementa con otras herramientas como Phantom (monitoriza los recursos y automáticamente configura y despliega nuevos para mantener diferentes demandas o fallos), cloudinit.d (herramienta para lanzar, controlar y monitorizar entornos complejos en la nube facilitando la gestión y despliegue de MV basadas sobre «planes» previamente configurados), y el *Context Broker* (un servicio que permite a los usuarios coordinar y desplegar grandes clústeres virtuales automáticamente y en forma repetitiva).

La figura siguiente muestra la arquitectura de Nimbus:



d) OpenNebula: plataforma *open source* (ASL) para construir *clouds* IaaS tanto privados como públicos/ híbridos y que permite gestionar infraestructuras de centros de datos heterogéneas. La plataforma combina las tecnologías de gestión de almacenamiento, red, virtualización, monitorización y seguridad para desplegar servicios en múltiples niveles (por ejemplo, sobre clústeres) como máquinas virtuales. Además, incluye funciones de integración, gestión, escalabilidad, seguridad y contabilidad, se basa en los estándares actuales en cuanto a interoperabilidad y portabilidad y permite que los usuarios puedan realizar interconexión con diferentes interfaces (EC2 *Query*, OGF *Open Cloud Computing Interface* y vCloud) e interactuar con diferentes hipervisores (Xen, KVM y VMware).

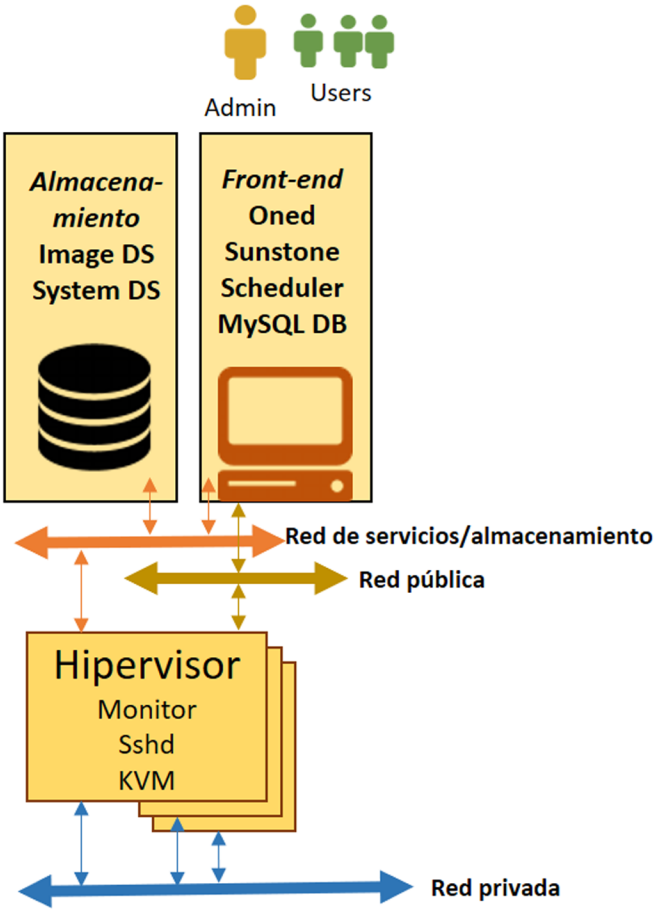
OpenNebula Systems (anteriormente C12G Labs) es la responsable del desarrollo/mantenimiento y la plataforma cuenta con un ecosistema llamado AppMakarket como un catálogo donde los usuarios/organizaciones pueden rápidamente distribuir y desplegar *appliances ready-to-run* sobre *clouds* OpenNebula.

La estabilidad de la plataforma y la calidad los servicios que provee la han consolidado como una de las opciones IaaS actuales tanto para grandes empresas como para pymes, pero también para proveedores de *hosting*, operadores de

telecomunicaciones, proveedores de servicios de TI, centros de supercomputación, laboratorios de investigación y proyectos de investigación internacionales.

El esquema de interacción es a través de un portal basado en roles y de autoservicio intuitivo, con un catálogo de servicios automatizados, interfaces de administración y el *appliance* AppMarket que permiten gestionar y administrar todos los recursos para los diferentes niveles/roles desde un único punto. Los requisitos para instalar la plataforma se diferencian en instalaciones básicas/medias (decenas de hipervisores) o avanzadas (centenas de hipervisores). En relación con las redes, la de servicios entre el *frontend* y el almacenamiento pueden ser compartidas (dado el bajo volumen de comunicaciones de los servicios) y las MV necesitarán una red pública y otra privada (para implementar *Vlans* aisladas).

En cuanto a los sistemas operativos puede ser Debian/Ubuntu o CentOS/RHEL (en todas las máquinas) con paquetes específicos para cada distribución, con KVM para los hipervisores y VLAN 802.1Q para las instalaciones básicas y VXLAN para las avanzadas. Para el almacenamiento se puede utilizar NFS/GlusterFS para las primeras (con imágenes en formato qcow2) y Ceph para las segundas, y en cuanto a la autentificación se puede configurar el sistema nativo de autentificación que dispone OpenNebula o Active Directory. Los requerimientos de hardware del *frontend* en instalaciones básicas son: 1 CPU (2 *cores*), 2GB RAM, 100GB disco y 2 NIC, mientras que para las avanzadas: 2 CPU (4 *cores*), 4Gb RAM, 500GB disco y 2 NIC. La figura siguiente muestra la arquitectura de OpenNebula [Ona15].



OpenNebula utiliza KVM como hipervisor, pero sobre la base de su flexibilidad e interoperabilidad algunas organizaciones/empresas lo utilizan como *cloud management* en VMware vCenter, llamado vOneCloud, para proporcionar una capa de aprovisionamiento *multi-tenancy* por encima de VMware vCenter. Estos despliegues buscan características de provisión, elasticidad y *multi-tenancy* como aprovisionamiento de centros de datos virtuales y federación de centros de datos, mientras que la infraestructura es administrada por herramientas ya conocidas como vSphere y vCenter Operations Manager.

e) OpenQRM (*community edition*): es una plataforma *open source* (GPL) de *cloud computing* para manejar infraestructura de centros de datos permitiendo construir sobre ellos *clouds* privados, públicos e híbridos gestionando la virtualización (a través de KVM, Linux-VServer, OpenVZ, VMware ESX o Xen), el almacenamiento, la red, la monitorización y la seguridad. Con ello, permite desplegar servicios *multi-tier* sobre clústeres de cómputo como máquinas virtuales combinando recursos locales como remotos y gestionando su asignación mediante políticas preestablecidas.

La plataforma openQRM establece un aislamiento perfecto entre el hardware (servidores físicos o virtuales) y el software (imágenes de los SO y servicios) de forma que el hardware puede ser sustituido sin necesidad de reconfigurar

el software y permite diferentes modelos de migración de MV (P2V –*physical to virtual*–, V2P –*virtual to physical*–, y V2V –*virtual to virtual*–), y también es posible hacer una transición de un hipervisor a otros con la misma MV.

La arquitectura de esta plataforma es extensible mediante conectores (*pluglins*) que permiten insertar API o conectores hacia el *cloud* público (por ejemplo, AWS) u otros *clouds* (Openstack, Eucaliptus, etc.). Los requerimientos para hacer una prueba funcional son un servidor (para el servidor openQRM, rd virtualización y almacenamiento) con Intel/AMD de 64bit, *dual/quad core* y extensiones VT-x/AMD-V, 2 Gb RAM, por lo menos 20 GB disco y un NIC (recomendado 1 Gbit/s) y acceso a internet.

La última versión (5.3) se debe descargar desde el desarrollador (que tiene publicados una serie de How-To), pero las versiones anteriores de deben descargar directamente desde SourceForge donde también se puede acceder a la documentación de la comunidad (si bien está poco actualizada).

g) OpenStack: es una plataforma *open source* (ASL) de *cloud computing* y que permite desplegar IaaS mediante un conjunto de componentes interrelacionados que controlan recursos de hardware (de diferentes vendedores) de cómputo, red y almacenamiento de un centro (o más) de datos. Los usuarios pueden gestionar y administrar el *cloud* utilizado un panel de control (*dashboard*) a través de web o a través de CLI o utilizando una API RESTful. OpenStack se inicia en 2010 como un proyecto conjunto de Rackspace Hosting y la NASA pasando a ser gestionado en 2014 por la OpenStack Foundation con el objetivo de promover esta plataforma y su comunidad y de la cual hoy forman parte más de quinientas empresas/organizaciones/instituciones.

La comunidad está organizada en ciclos de desarrollo y actualizaciones semestrales que se reúnen en sesiones de trabajo (multitudinarias) en la OpenStack Summit para facilitar el desarrollo y generar planes de futuro (en la sesión de abril de 2016 se reunieron 7.500 personas vinculadas a la plataforma y la última tuvo lugar en Barcelona en octubre de 2016; se puede acceder a todos los vídeos de las sesiones). Hoy se considera que la comunidad OpenStack está formada por 62.000+ desarrolladores/usuarios, 640+ compañías de 187 países e incluye más de 20 M+ líneas de código.

El primer código de OpenStack (2010) proviene de la plataforma Nebula de la NASA (no confundir con OpenNebula) y de la plataforma Rackspace Cloud; en 2011, Ubuntu adopta OpenStack para su estrategia de *cloud* (reemplazando Eucaliptus) y da soporte para *clouds* basados en Ubuntu 11.04 y la versión OpenStack Cactus. Igualmente pasa con Debian 7.0, que incluye la versión Essex, lo mismo para SuSE, y a partir del 2012, todas las grandes compañías de tecnología comienzan a incluirlo en sus estrategias y hacerlo disponible para sus clientes integrado o adaptado junto con sus productos (RHEL, Oracle, HP, etc.), llegando hoy en día a tener instalaciones muy particulares como la del Cern (la más grande de 190.000+ *cores*), SnapDeal (India, Markeplace

100.000+ *cores* y 16 PBytes de almacenamiento), China Mobile (10.000 nodos con 100.000 *cores*), Walmart (*e-commerce* en 30 regiones con 170k+ *cores*) entre otros.

La arquitectura de OpenStack es modular con diversos componentes, entre los que se pueden enumerar [Osd16]:

- ***Compute* (Nova)**: es la parte principal del IaaS y actúa como controlador del *cloud*. Este gestiona los conjuntos de recursos (*pools*) y puede trabajar con diferentes tecnologías de virtualización, hardware-base (*bare metal*) y configuraciones de *high-performance computing* (HPC) con KVM, VMware, Xen, Hyper-V como posibles hipervisores y con LXC como soporte para contenedores. Este módulo no necesita ningún hardware propietario (está escrito en Python), escala horizontalmente y utiliza diferentes librerías externas como Eventlet, Kombu o SQLAlchemy.

- ***Networking* (Neutron)**: gestiona las redes y sus parámetros para que no representen cuello de botella en una instalación a través de autoservicio. OpenStack proporciona diferentes estrategias de interconexión (como redes planas o VLAN, IP estáticas o DHCP reservados, IP flotantes, etc.), lo cual permite a los usuarios crear sus propias redes, controlar el tráfico y conectar los servidores y los dispositivos a una o más redes. Los administradores pueden aprovechar las redes definidas por software de tecnología (SDN) como OpenFlow y permite anexar servicios de red adicionales, como los sistemas de detección de intrusos (IDS), balanceo de carga, cortafuegos o redes privadas virtuales (VPN).

- ***Block Storage* (Cinder)**: este módulo proporciona dispositivos de almacenamiento a nivel de bloque persistentes que pueden ser utilizados por las instancias de Nova y gestiona la creación, montado y desmontado de los dispositivos de bloque a los servidores y los cuales se integran plenamente en Nova y el panel de control (*dashboard*) para que los usuarios puedan gestionar sus propias necesidades de almacenamiento. Además del almacenamiento local de Linux, puede utilizar las plataformas de almacenamiento incluyendo Ceph, CloudByte, Coraid, EMC GlusterFS, Hitachi, IBM, LIO, NetApp, Nexenta, Scality, SolidFire y HP, entre otros. El almacenamiento de bloques es apropiado para escenarios donde el rendimiento es sensible, tales como base de datos, sistemas de archivos de crecimiento dinámico o para permitir al servidor acceder a nivel de bloque «en bruto» (*raw*).

- ***Identity* (Keystone)**: implementa un directorio de identidades gestionando los servicios y los permisos que disponen los usuarios y actúa, además, como un sistema de autenticación común en todo el *cloud* integrando servicios de directorio *backend* existentes como LDAP. Es compatible con múltiples formas de autenticación, incluyendo usuario y contraseñas, sistemas basados en *tokens* e inicios de sesión (estilo AWS). También incluye una lista de todos los servicios existentes en el *cloud* OpenStack en un solo re-

gistro para que los usuarios y servicios de terceros puedan consultar qué recursos y con qué permisos pueden acceder.

- ***Image* (Glance)**: proporciona gestión sobre las imágenes de los discos y servidores (las cuales se pueden utilizar como una plantilla para nuevos despliegues). Entre sus funciones más importantes están las de almacenar y gestionar imágenes de disco y de servidores en una variedad de *back-ends* (incluyendo OpenStack *Object Storage* Swift) y dispone de una API REST para gestionar las imágenes (por ejemplo, con Heat para tener metadatos sobre las imágenes o con Nova para configurar una variación de una imagen y generar una nueva instancia). Este módulo es el único que puede agregar, borrar, duplicar o compartir una imagen y atiende las peticiones de los otros módulos según se necesite.

- ***Object Storage* (Swift)**: es un sistema de almacenamiento redundante y escalable donde los objetos/archivos se gestionan en diversas unidades de disco repartidos por los servidores del centro de datos teniendo en cuenta su replicación e integridad. Este servicio puede escalar horizontalmente añadiendo nuevos servidores y gestiona la redundancia desde la capa software, por lo cual pueden utilizarse discos y servidores de bajo costo.

- ***Dashboard* (Horizon)**: interfaz del panel de control que permite a administradores/usuarios acceder y gestionar la provisión y automatización de recursos basados en el *cloud*. Su diseño es abierto y permite que nuevas funcionalidades puedan ser incluidas (facturación, monitorización, etc.) transformándose en punto de acceso a la infraestructura *cloud* (además de la API nativa de OpenStack o la API de compatibilidad EC2).

- ***Orchestration* (Heat)**: permite, mediante plantillas, gestionar configuraciones y despliegues complejos de *cloud* a través de OpenStack-*native* REST API o *CloudFormation-compatible Query* API.

- ***Workflow* (Mistral)**: es un servicio que gestiona los flujos de tareas (*workflows*) generados por los usuarios (escritos en YAML). Se puede interactuar a través de la REST API pudiendo iniciar la ejecución del *workflow* a través de la misma API o programando su inicio a través de un evento (*trigger*).

- ***Telemetry* (Ceilometer)**: provee un único punto de contacto para la facturación unificando los contadores necesarios a través de todos los componentes de la arquitectura y permitiendo que se puedan auditar y hacer un seguimiento de los mismos con el objetivo de la transparencia.

- ***Database* (Trove)**: es un servicio de *database-as-a-service* aprovisionando un motor de bases de datos relacional y no-relacional.

- ***Elastic Map Reduce* (Sahara)**: componente que permite la provisión rápida de clústeres Hadoop mediante la especificación simplificada por parte del usuario.

- ***Bare Metal* (Ironic)**: aprovisiona máquinas *bare-metal* en lugar de instancias de MV utilizando PXE e IPMI para gestionar las máquinas hardware.

- ***Messaging* (Zaqar)**: es un servicio de mensajes *multitenancy* disponible para desarrolladores web que funciona a través de un RESTful API y que puede enviar mensajes entre los diversos componentes de un SaaS y apps móviles.

- ***Shared File System* (Manila)**: provee una API para gestionar comparticiones de objetos (independientes del proveedor) permitiendo crear, borrar, dar acceso o denegar a los objetos sobre diferentes entornos de almacenamiento como EMC, NetApp, HP, IBM, Oracle, Quobyte e Hitachi o sobre sistemas de archivos como RHEL GlusterFS.

- **DNS (Designate)**: provee una REST API para manejar *DNS as a service* siendo compatibles con otras tecnologías como PowerDNS y BIND. Este no provee el servicio como tal, sino que actúa como interfaz a DNS existentes para manejar zonas de DNS.

- ***Search* (Searchlight)**: provee capacidad de búsqueda a través de los servicios de *cloud* OpenStack a través de las API provistas por ellos e indexando los datos mediante ElasticSearch.

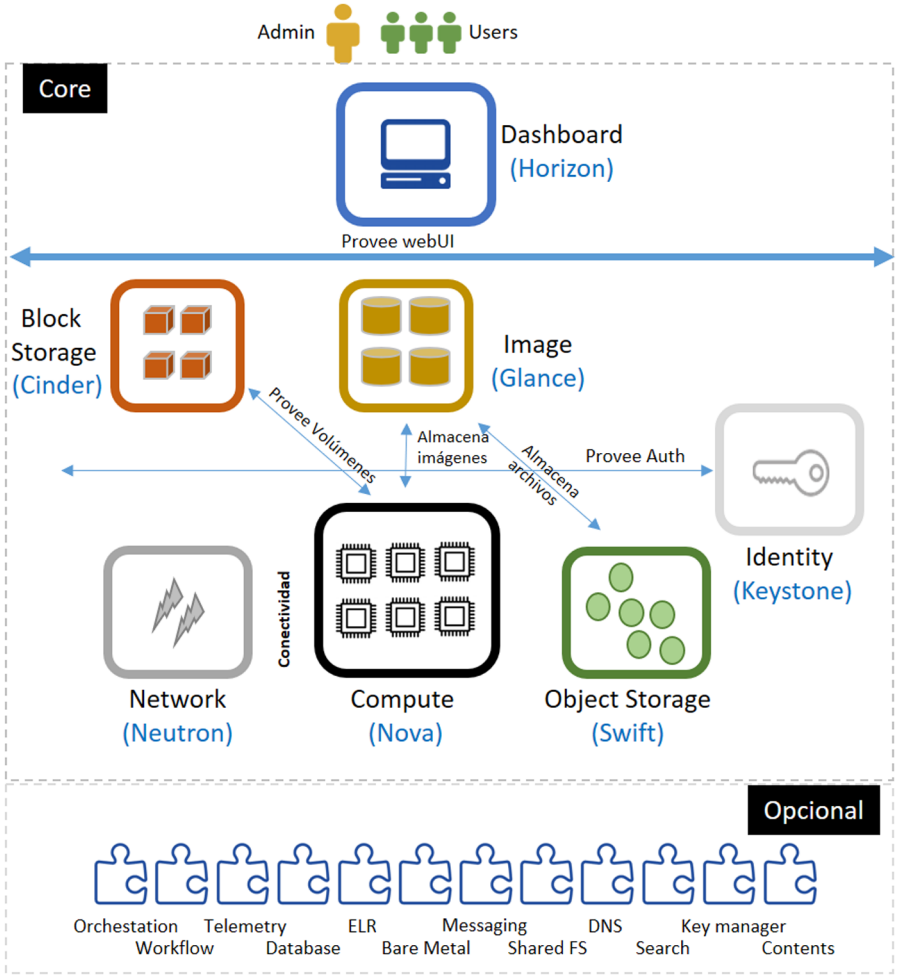
- ***Key Manager* (Barbican)**: es una REST API diseñada para el almacenamiento seguro, aprovisionamiento y gestión de llaves (secretas).

- ***Contents* (Magnum)**: provee un catálogo de aplicaciones de OpenStack permitiendo que los desarrolladores y administradores puedan publicar aplicaciones *cloud-ready* y que estas estén disponibles para los usuarios y así puedan componer entornos de aplicación con una simple selección.

La última versión, de noviembre del 2016, es Newton e incorpora Nova, Glance, Swift, Horizon, Keystone, Neutron, Cinder, Heat, Ceilometer, Trove, Sahara, Ironic, Zaqar, Manila, Designate, Barbican, Searchlight, Magnum, considerándose el núcleo (*core services*) de Openstack formado por **Nova**, **Neutron**, **Swift**, **Cinder**, **Keystone**, **Glance** (+Horizon) y el resto de servicios opcionales. Consultando el MarketPlace se podrán ver las distribuciones o *appliances* que lo soportan, los *clouds* públicos/privados que lo implementan o proveedores de formación o consultoras relacionadas con OpenStack o controladores para dispositivos específicos.

Como se puede observar en el *marketplace*, existen una gran cantidad de *appliances* de OpenStack con diferentes objetivos entre las que se puede nombrar Ubuntu Autopilot, Oracle, Mirantis, entre otros (se debe tener en cuenta que algunas *appliances* de algunos proveedores son versiones de prueba por tiempo limitado).

La figura siguiente muestra los componentes de OpenStack Newton (versión de octubre de 2016) y sus interrelaciones.



h) Ovirt: es una plataforma *open source* (ASL), creada por RH como proyecto de la comunidad y con base a su distribución y tecnología, de gestión/provisión de entornos virtualizados (con KVM) y formada por dos componentes: **oVirt engine** y **oVirt node**. Esta permite, a través de una interfaz web muy sencilla, gestionar de forma centralizada máquinas virtuales y recursos de cómputo, red y almacenamiento de forma remota y sin acceder a los recursos físicos (es el equivalente *open source* a VMware™ vSphere™).

Sus requerimientos son muy básicos, por ejemplo, para la oVirt Engine *dual-core*, 4GB RAM, 25 GB disco y NIC 1 GB, Fedora 19+, CentOS 6.5+, y para los *hosts* que desplegarán las MV (que pueden ser oVirt Node, Fedora Host, CentOS Host) *dual-core* server, 10 GB RAM (considerar que cada VM se llevará

1 GB aproximadamente), 10 GB disco, NIC 1 GB con extensiones AMD-V, VT-x y almacenamiento como NFS, iSCSI, FCP, Local, POSIX FS, o GlusterFS. Además, se necesitarán las imágenes de los discos de instalación (Windows, RHEL, Ubuntu, Fedora, OpenSuSE, CentOS) de las cuales luego se crearán las MV.

El núcleo de oVirt engine está escrito en Java, la interfaz en GWT *webtoolkit* y se ejecuta sobre el servidor de aplicaciones WildFly (antes JBoss). Los usuarios pueden ser gestionados internamente o conectados a un LDAP o AD. Para el almacenamiento oVirt utiliza PostgreSQL y provee RESTful API como interfaz a las funcionalidades de núcleo; oVirt node es un servidor ejecutando RHEL, CentOS, o Fedora (experimentalmente Debian) con hipervisor KVM y un daemon VDSM (*Virtual Desktop and Server Manager*). Toda la gestión y administración se realiza a través del portal web en oVirt engine, que se conecta con VDSM para realizar las tareas indicadas. oVirt engine puede ser instalada en un nodo separado o puede instalada sobre un nodo del clúster como un MV (*self-hosted engine*), la cual se podrá instalar manualmente o utilizar una *appliance* lista para ejecutar.

4.4. PaaS (*platform as a service*)

Las PaaS proporcionan a los usuarios herramientas para desarrollar, ejecutar y administrar aplicaciones (generalmente web), y dado que estas se ofrecen como un servicio a través de internet, los desarrolladores pueden trabajar dentro de ellas sin tener que gestionar la infraestructura subyacente, permitiéndoles concentrarse únicamente en desarrollar su aplicación. Las ofertas de plataformas incluyen componentes de muchos servicios utilizados en el desarrollo de software, incluyendo bases de datos, servidores web, sistemas operativos y almacenamiento, permitiendo que en algunas de ellas trabajen múltiples usuarios, facilitando la colaboración y compartición de código y recursos.

Entre las más destacadas (o que mejor salen en algunos indicadores como satisfacción, cuota de mercado, soporte y servicio, por ejemplo, en G2 *Crowd categories/cloud-platform-as-a-service-paas*) son: **Heroku**, **Amazon EC2**, **Microsoft Azure**, **Force.com y OpenShift** (Heroku y Force.com sobre SalesForce).

Si se consideran plataformas con un alto grado de satisfacción por sus usuarios pero que todavía no han alcanzado una alta cuota de mercado, se pueden mencionar **Morpheus** y **Dokku**. También en productos que tienen presencia y recursos, aunque con valoraciones por debajo del primer grupo, están **Google App Engine**, **AWS Elastic Beanstalk** y **AWS Lambda**.

Entre plataformas de «nicho» o muy especializadas encontramos **Cloud Foundry** y **Deis**. Existen diversas listas de las PaaS más habituales y otras valoraciones como la de IDG, que incluyen además de las mencionadas **Long-**

Jump, **IBMSmartCloud**, **CloudBees**, **EngineYard**, **HPE Helion Stackato** u otras no menos interesantes: **Platform-sh**, **Zoho Creator**, **Mendix**, **Morpheus**.

Existen diversas razones para seleccionar las plataformas PaaS, entre las que se pueden mencionar:

- Lenguajes de programación: dado que será proyecto software, las plataformas escogidas serán aquellas que den soporte al lenguaje bajo el cual se desarrollará el proyecto y estén dentro de los requerimientos del mismo.

- Público frente a privado: la PaaS pública representa beneficios de disponibilidad e inmediatez, ya que después del registro se comienza a trabajar. La PaaS privada requiere adecuación y despliegue de infraestructura y provisión del servicio requiriendo recursos adicionales y un departamento de IT adecuado a los objetivos, teniendo como contrapartida control, seguridad y privacidad.

- Tiempo de funcionamiento del servicio: dependencia total en el caso de público y controlado en el caso de privado que pueden afectar seriamente a la planificación del proyecto.

- Estructura de precios: selección del modelo adecuado a las necesidades del proyecto y con equilibrio entre servicios, soporte y recursos.

- Recursos: si la propia plataforma aporta sus propios recursos o si estos deben ser gestionados externamente (por ejemplo, sobre EC2/S3).

- Integración: el proyecto después deberá integrarse con otros servicios, por lo cual se debe analizar el tiempo necesario para realizarlas y bajo qué política para evitar inconvenientes después de iniciado el trabajo.

Sobre plataformas *open source* que pueden ser instaladas sobre infraestructura propia (o *cloud*) podemos mencionar (tener en cuenta que, si bien todas comparten objetivos similares, la visión y misión de cada una de ellas es diferente y escapa al detalle de este apartado):

- **OpenShift origin**: Origin es el proyecto comunitario que impulsa OpenShift y construido sobre la base de contenedores Docker y gestión de clústeres de contenedores Kubernetes. Origin es una plataforma activa que está redefiniendo el ciclo de vida de la aplicación y herramientas de DevOps proporcionado por una plataforma completa de aplicación de contenedores *open source*.

- **Apache Stratos** (antes WS02): es un entorno PaaS extensible que permite ejecutar aplicaciones Apache Tomcat, PHP y MySQL y con posibilidad de extenderse a otros servicios sobre las infraestructuras *cloud* más importan-

tes. Para los desarrolladores, Stratos proporciona un entorno basado en el *cloud* para desarrollar, probar y ejecutar aplicaciones escalables y para los proveedores de TI obtienen beneficio de altas tasas de utilización, gestión automatizada de recursos y la visión general de la plataforma, incluida la supervisión y facturación.

- **Cloudify**: es un entorno *open source* de orquestación que permite modelar aplicaciones y servicios y automatizar todo su ciclo de vida, incluyendo la implementación en cualquier entorno de *cloud* o centro de datos, monitorizando todos los aspectos de la aplicación desplegada, detectando problemas y fallos y permitiendo que se desplieguen soluciones (manual o automáticamente) y manejando las tareas de mantenimiento.

- **Tsuru**: PaaS *open source* desarrollada por Globo.com y disponible en GitHub. Los desarrolladores pueden utilizar `git, tsuru cli` o el propio panel de control (*dashboard*) para desarrollar aplicaciones web en diferentes lenguajes.

- **Cloud Foundry**: plataforma PaaS *open source* desarrollada originalmente por VMware y ahora por Pivotal Software (*join venture* entre EMC, VMware y General Electric). Esta plataforma soporta todo el ciclo de vida de las *apps* desde el desarrollo inicial, etapas de pruebas, hasta la publicación ajustándose a una estrategia de entrega continua. Los usuarios tienen acceso a uno o más espacios que corresponden a una etapa del ciclo de vida, y cada usuario adopta un rol diferente en función del espacio que tenga permisos para acceder.

- **Dokku**: para algunos expertos, la PaaS más pequeña y eficiente para gestionar el ciclo de vida de las *apps*.

- **Deis Workflow**: PaaS que agrega una capa *developer-friendly* a cualquier clúster Kubernetes haciendo fácil el desarrollo y despliegue de aplicaciones.

- **AppScale**: plataforma *open source* que automáticamente despliega y escala sobre *clouds* privados/públicos aplicaciones Google App Engine sin modificación.

Si bien algunas de ellas **no** se pueden calificar como plataformas PaaS, es útil mencionar plataformas orientadas a desarrolladores (*cloud IDE*) como (entre otras):

- **Cloud9.io** que ofrece cuarenta lenguajes, control sobre la aplicación de espacio de trabajo (*workspaces*) público ilimitado gratuito, depuración de aplicaciones en diferentes entornos (Django, Rails, WordPress...) en unas 300+ combinaciones, entre otras características.

- **Codenvy**: que ofrece *cloud workspaces* para grupos de desarrolladores basado contenedores Docker permitiendo disponer de una plataforma para DevOps.

- **Koding**: plataforma de desarrollo que orquesta todo el entorno y de donde los desarrolladores obtienen todo lo que necesitan para crear entornos completos y específicos de proyectos en pocos segundos utilizando una interfaz sencilla para compartir, actualizar y gestionar la infraestructura. Esta plataforma puede probarse desde koding.com. Proyecto similar Codebox.

- **Codiad**: es una plataforma IDE basada en web con unos requerimientos muy reducidos, permitiendo disponer de un entorno de desarrollo en el *cloud* sin caer en potentes recursos que necesitan los entornos IDE habituales. Es por ello que los usuarios de herramientas como Eclipse, NetBeans y Aptana encuentran en esta plataforma simplicidad y prestaciones.

- **Codeanywhere**: plataforma *cloud* que permite disponer de un entorno de desarrollo con el soporte de más de setenta y dos lenguajes y entornos preconfigurados.

- **Ideone**: compilador y depurador en línea en un modelo de *write-&-run* con soporte para más de sesenta lenguajes basados en tecnología Sphere Engine.

También podemos encontrar entornos basados en el *cloud* de objetivos generalistas, por ejemplo, **Coding Ground**, que presenta 17 terminales en línea y 96+ IDE interactivos para editar, compilar, ejecutar y compartir programas en línea en una plataforma *cloud*.

4.5. SaaS (*software as a service*)

Las plataformas SaaS son un modelo de distribución de software donde el soporte lógico y los datos que maneja se alojan en servidores de una compañía de IT (no necesariamente el proveedor del servicio) y el cliente accede a ellos a través de un software específico (software cliente, aunque en desuso) o a través de un servicio web.

La empresa proveedora se ocupa del mantenimiento, operación diaria y soporte del software usado por el cliente y en la mayoría de los casos este puede acceder desde cualquier sitio y desde cualquier dispositivo.

Dentro de este modelo de plataforma, la ejecución y el almacenamiento de los datos del cliente siempre se encuentran sobre los servidores del proveedor; el cliente no se debe preocupar por su disponibilidad, acceso, mantenimiento y seguridad, ya que todo ello recae sobre el proveedor en los términos de la SLA firmada con el cliente.

Este tipo de modelo presenta **ventajas** como:

- el cliente no necesita ni inversiones en infraestructura ni personal técnico especializado en el servicio, reduciendo los costes y riesgos de inversión,
- la operación recae en la empresa proveedora (disponibilidad, funcionalidad, seguridad, etc.) de acuerdo a una SLA,
- no hay abandono del servicio ya que el proveedor funciona mediante una cuota de pago-por-uso, por lo cual es necesario atenderlo para que continúe pagando,
- no es necesario preocupaciones por licencias por máquinas (aquellas que se pagan se utilicen o no) ya que solo se paga por el uso,
- no es necesario modificar la máquina del cliente para que acceda al servicio ni adecuar la potencia/prestaciones, ya que todo se ejecutará en la máquina del proveedor.

Asimismo, se pueden enumerar un conjunto de **desventajas** como:

- no se tiene acceso a los datos (excepto que el servicio prevea un método de descarga),
- si no se cuenta con mecanismos de cifrado y control disminuye el índice de privacidad, control y seguridad,
- el usuario accede a un servicio y no puede modificarlo o configurarlo más allá de las opciones que le permite el proveedor,
- se puede producir el *data lock-in* (en caso de cierre de actividades de la empresa proveedora) o *vendor lock-in* (costes más altos que la competencia ya que aún son mayores los de migración),
- pérdida del servicio o reducción de la calidad si la conexión es deficiente.

Mediante cuatro preguntas podemos determinar si el servicio al cual estamos accediendo es SaaS:

a) ¿Se puede acceder al servicio a través de un navegador o de protocolos normalizados como REST?

b) ¿Se puede acceder a un modelo de pago como *pay-as-you-go*?

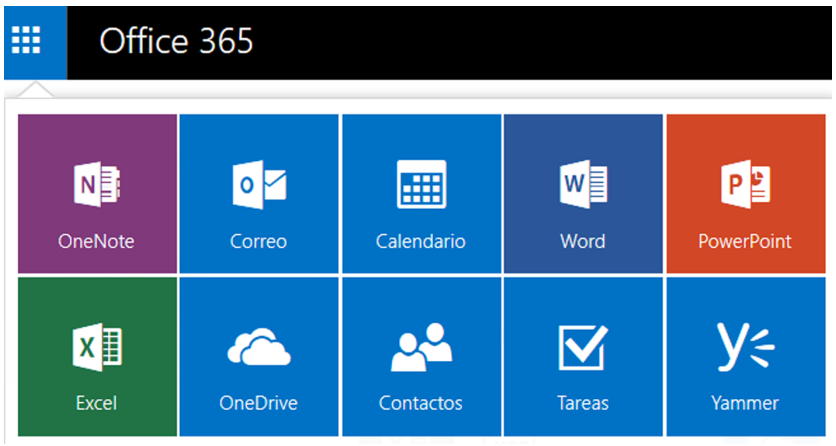
c) ¿El software escala bajo demanda?

d) ¿El proveedor asume la responsabilidad de la infraestructura, la gestión, el mantenimiento, la mejora y la supervisión del software, dejando al cliente solo utilizar el software?

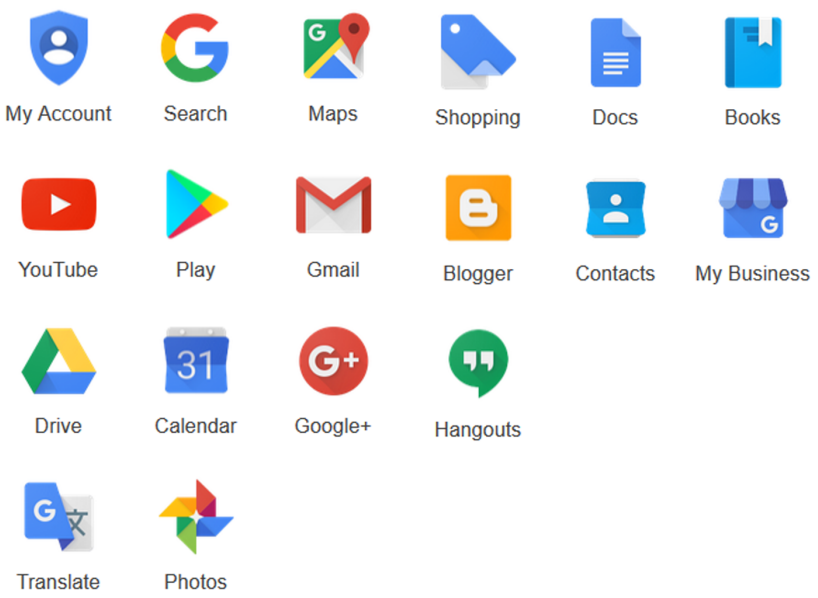
Si la respuesta es un rotundo sí, podemos asegurar casi con total seguridad que estamos ante un SaaS.

Google y Microsoft

Dos de los ejemplos que muestran habitualmente sobre SaaS son dos de las grandes compañías de IT: Google y Microsoft. Las imágenes a continuación muestran los servicios ofrecidos dentro de las plataformas correspondientes:



En Microsoft no son los únicos servicios que se ofertan dentro de la licencia SaaS (tanto para ordenador de escritorio como para dispositivo móvil) que incluye herramientas de edición de documentos (Office365), colaboración (SharePoint, Yamer), mensajes instantáneos (Lync), correo (Exchange), almacenamiento (OneDrive) pero también CRM, comunicación (Skype) y BMI, entre los más destacados. Se debe tener en cuenta que algunos de ellos son gratuitos (no *open source*) como por ejemplo el correo, almacenamiento (hasta un determinado tamaño) y otros son con licencia con diferentes modelos (por ejemplo, para centros educativos Office365 es gratuita).



Google oferta también sus recursos, aunque bajo otro modelo de negocio (gratuito+):

Además de los mostrados de búsqueda, navegadores, mapas, correo, documentos, vídeos y fotos, almacenamiento, calendarios, comunicación y traducción, ofrece redes sociales/compras y servicios para las empresas. En caso de Google, la gran mayoría de herramientas son de uso gratuito y el modelo de negocio (aunque no se conoce totalmente) está en herramientas como Adwords y Adsense, que funcionan sobre los datos que se recogen desde las diferentes herramientas (concepto llamado datos masivos o *big data*).

Entre otros ejemplos (solo alguno de ellos) de plataformas SaaS de éxito podemos mencionar (orden alfabético):

Categoría	Sitio
Almacenamiento	Dropbox MediaFire Mega
Colaboración	Asana Todoist Trello Evernote
Documentos	Pixlr, Polar (edición fotos) SiteBuilder (creación web) SlideRocket, Prezzi (creación presentaciones) Slideshare (presentaciones/documentos) StackEdit (edición en la web)
Media	NetFlix, Flickr (fotos) OutBrain (content) Spotify (música), Wikipedia (enciclopedia)
Servicios	bit.ly (URL) Flood (test) Ionic (apps) Mailchimp (mail+) Panda (antivirus)

Categoría	Sitio
Social	BuzzStream Facebook Instagram Linkedin LuckyOrange Twitter

Algunos autores consideran que el SaaS es una evolución de un modelo de negocio ampliamente conocido llamado **ASP** (*application service provider*) que proporciona servicios a los clientes a través de una red, permitiendo el acceso a una aplicación de software en particular. Este modelo ha evolucionado a partir de los crecientes costos de software especializado y de la complejidad en cuanto a la instalación y mantenimiento que se reducen y lo hacen accesible cuando funcionan en modo ASP.

En principio, podemos definir que son conceptos similares que el SaaS, pero existen **algunas diferencias**:

- El ASP es un modelo de negocio especializado (a veces con software de terceros adaptado a medida para un único cliente), se basa en una relación de tú-a-tú, mientras que SaaS es un modelo global, flexible y que en principio se basa en ubicuidad y de amplia distribución (aunque la propia empresa o terceros puedan hacer adaptaciones y configuraciones especializadas).

- Generalmente el ASP se aloja en sitios independientes del desarrollador, mientras que en SaaS los desarrolladores son los que ofrecen el software + *hosting* en un solo paquete (aunque comienza a haber desarrolladores que permiten utilizar las IaaS que disponga el cliente).

- Las ASP (la mayoría de ellas) se ejecutan mediante modelos cliente-servidor, por lo cual se requiere software específico sobre un SO determinado; se pierde la ubicuidad e independencia de SO (en SaaS solo se necesita un navegador) y, como es tecnología de un tercero, las actualizaciones y versiones dependen de la instalación realizada. Esto no pasa con el SaaS, ya que todos los clientes tienen siempre la última versión y el soporte es unificado con un servicio uniforme, con la consiguiente reducción de los costes.

- El ASP es unitario, es decir, instancias independientes y muy adaptadas/personalizadas para empresas diferentes, mientras que en SaaS, generalmente, son múltiples clientes con una sola instancia (*multitenancy*).

- El ASP puede integrar diferentes servicios como ofimática + directorio + CRM en un solo servicio en función de los acuerdos con los diferentes

proveedores, mientras que en un SaaS está orientado a un único producto (aunque algunos proveedores ofrecen SaaS multiproductos).

- Finalmente, en cuanto a la relación con el desarrollador/proveedor en SaaS, la relación es más directa ya que se habla directamente con quien ha desarrollado el software y no con un tercero que actúa de intermediario. Se podría pensar, con lo antes descrito, que un software complejo, como un ERP, solo puede funcionar en modo ASP por la gran adaptación y personalización que se necesita, pero existen casos como, por ejemplo, OpenBravo (ERP, TPV, POS Retail) que prestan este servicio como SaaS o sobre un IaaS externo (AWS) para pymes.

En cuanto a los modelos de **monetización del SaaS**, son de diferentes tipos y bastante más difusos que en los otros segmentos del *cloud*.

Los habituales son ***pay-per-user*** (modelo similar al de licencias en instalaciones locales) o ***pay-as-you-go*** (por cantidad de recursos –almacenamiento, CPU, etc.– que han sido consumidos en un periodo, por ejemplo, por hora) o su equivalente ***pay-what-you-want***.

Modelos ***freemium*** y ***premium***, donde el *freemium* representa un porcentaje muy alto de usuarios en relación con el *premium*, pero tienen limitaciones en las prestaciones/opciones y es la fuente que alimenta al *premium*. Los usuarios que realmente prueben y «usen» la aplicación se acabarán convirtiendo en *premium*; pagarán por ello y esto sustentará el modelo de negocio; el proveedor se esmerará en cultivarlo con el sentido de pertenencia a un grupo con privilegios. Los *freemium* tendrán incentivos para pasarse a *premium* y, en determinados momentos, reducciones dinámicas de las prestaciones para favorecer este paso.

Otros modelos de monetización son:

- *Long tail* (catálogo de muchos productos que se venden en pocas unidades; modelo Amazon),

- *bait & hook*, cebo y anzuelo (similar al *freemium*, acceso a unos recursos, pero limitados para incentivar el pasarse a pago, por ejemplo, *pay-as-you-go*),

- suscripción/*paywall* (distribución de los costos en una gran cantidad de usuarios y distribución de impacto inicial por una larga suscripción),

- anuncios (método indirecto que la publicidad propia o de terceros monetiza el servicio),

- micro-transacciones (aplicado al ocio generalmente, donde se adquieren recursos o servicios con inversiones de muy bajo valor pero que en su conjunto representa un gran volumen) o

- *tiered service* (variante del *pay-as-you-go*, incrementar el precio en función progresiva a medida que se necesitan más recursos).

En relación con la **estructura del SaaS**, la mayoría se basa en arquitecturas *multitenant* en las cuales una única versión de la aplicación, con una única configuración (hardware, red, sistema operativo), se utiliza para todos los clientes (*tenants*) y para soportar la escalabilidad, la aplicación se instala en varias máquinas permitiendo el escalado horizontal.

No obstante, algunas SaaS no utilizan *multitenancy* sino modelos más aislados como los contenedores (modelo de Google) o máquinas virtuales, ya que permiten una mejor privacidad, funcionamiento y bajo uso de recursos. No obstante, es un tema de gran discusión en foros especializados y cada proveedor tiene su visión particular actualmente entre *multitenancy* o *containers*.

En cuanto a plataformas para construir infraestructuras SaaS, podemos mencionar:

- Innomatic: entorno *open source* para construir plataformas *SaaS multitenant*, productos y aplicaciones empresariales en PHP. Con ella, ISV y empresas pueden construir aplicaciones *multitenancy* preparadas para ser ejecutadas en el *cloud* basadas en los entornos Composer y Symfony2.

- Appserver.io: *multithreaded application server for PHP* para crear aplicaciones *multitency* basadas en PHP.

- Drupal: CMS SaaS (openSaaS).

En modelo bajo licencia Multihoster, SurPaaS /SaaS OpenBox, SaaS Maker (gratuita para desarrolladores), y Lithium entre otros.

4.6. Otros servicios *cloud*

Ya se mencionó anteriormente que existe una gran cantidad de XaaS y, en este apartado, revisaremos dos que tienen alto impacto sobre los anteriores, o que pueden ser comercializados por separado o unidos a los anteriores:

Storage as a service

En este apartado, han surgido diferentes tipos de negocio como ya se ha mencionado anteriormente, como Amazon S3 o Dropbox/Drive (algunos autores engloban todo esto como *cloud storage*) que, teniendo como premisa el almacenamiento, los modelos de explotación son diferentes; el primero ofrece los

servicios de almacenamiento en internet con interfaz *web services* (REST, SOAP y BitTorrent), mientras que el segundo/tercero son explotados básicamente mediante SaaS.

No obstante, y más allá de los servicios y modelos de negocio utilizados, una parte importante para todo tipo de *cloud* es el software necesario para lograr este servicio. En este sentido, describiremos los paquetes *open source* que permiten generar un servicio de almacenamiento distribuido, fiable y seguro.

a) GlusterFS: utiliza FUSE (sistema de archivos en el espacio de usuario) para generar un VFS (*virtual file system*) y permite crear un sistema de archivos de red en un clúster en espacio de usuario utilizando sistemas de archivos existentes como ext3, ext4, xfs, etc. para almacenar datos. La popularidad de GlusterFS viene de las capacidades de escalado y accesibilidad, ya que puede proporcionar petabytes de datos en un único punto de montaje distribuyendo los archivos a través de una colección de subvolúmenes, haciendo unidades mayores de la unión de espacios pequeños y distribuidos, permitiendo replicación; tiene por ello redundancia y alta disponibilidad.

b) Ceph: la base de este sistema de almacenamiento es *Reliable Autonomic Distributed Object Store* (RADOS), que proporciona aplicaciones con almacenamiento de objetos, bloques y archivos en un único clúster de almacenamiento unificado. Con las bibliotecas que se ofrecen a las aplicaciones cliente, los usuarios pueden aprovechar RADOS *Block Device* (RBD), RADOS *Gateway*, así como el sistema de archivos Ceph. El RADOS *Gateway* proporciona interfaces a Amazon S3 y OpenStack y compatibles con el almacén de objetos RADOS. Además, Ceph provee una interfaz Posix, por lo cual las aplicaciones que utilizan sistemas de ficheros compatibles con este estándar pueden utilizar fácilmente el sistema de almacenamiento de objetos de Ceph. Este sistema de archivo de altas prestaciones permite la lectura/escritura parcial o completa, instantáneas, asignaciones de valor-clave de nivel de objeto y transacciones atómicas.

c) OpenStack: esta arquitectura proporciona almacenamiento de objetos escalable y redundante utilizando clústeres de servidores; puede escalar hasta petabytes de datos. A través de este sistema de almacenamiento distribuido ofrece a los usuarios de esta infraestructura escalabilidad, redundancia sobre los datos almacenados y además dispone de interfaces con Ceph, NetApp, Nexenta, SolidFire y Zadara. Las características incluyen instantáneas, escalado en caliente, soporte para almacenamiento en bloque, *self-healing* y herramientas potentes para la administración, uso, rendimiento y auditoría.

d) Sheepdog: permite almacenamiento de objetos distribuidos y se caracteriza por su pequeño tamaño, simplicidad y facilidad de uso. Este sistema gestiona volúmenes y servicios que, de forma inteligente, maneja los discos y nodos con un escalado óptimo que puede llegar a cientos o miles de dispositivos. Sheepdog se puede anexar a MV de QEMU y los *targets* SCSI de Linux, tiene

soporte para `libvirt` y OpenStack y puede interactuar con HTTP Simple Storage. A nivel del sistema de archivos permite hacer instantáneas, clonación y aprovisionamiento delegado y puede anexarse a MV y SO para que se ejecuten en *bare-metal* (con interfaz iSCSI).

Network as a service

Frecuentemente, se asocia el término a otros servicios con el *cloud* o también con *communication-as-a-service* (CaaS), pero en un sentido amplio este incluye la provisión de un servicio de red virtual desde la red específica hacia terceros y se utilizan protocolos como OpenFlow. Entre el software *open source* utilizado para proveer este servicio en el *cloud* podemos enumerar:

a) Floodlight: es un controlador OpenFlow de altas prestaciones en Java y bajo licencia de Apache. Es un controlador SDN (*software defined networks*) abierto que funciona con dispositivos (*switches*) físicos y virtuales que interactúan a través del protocolo OpenFlow; se puede escoger el protocolo para interactuar con los restantes dispositivos remotos de red (*switches*, *routers*, *virtualswitches*, o *accesspoints*). Al utilizar OpenFlow permite explotar toda la funcionalidad de este y controlar remotamente (tablas de reenvío de paquetes, reglas de flujo, reenvío o bloqueo de tráfico) y aprovechar interfaces personalizadas y lenguajes de secuencias de comandos. Uno de los promotores de proyecto es Big Switch Networks, que ofrece su SDN versión *community* sin costo.

b) OpenStack Networking «Neutrón»: es una parte de proyecto OpenStack y proporciona una «red como un servicio» entre los NIC gestionados por Nova. Si bien es parte de núcleo de OpenStack, Neutron puede ser considerado por su tamaño y funcionalidad como un producto NaaS donde los usuarios pueden crear topologías de aplicaciones de múltiples niveles, utilizar capacidades avanzadas de red como la supervisión de QoS o NetFlow de extremo a extremo; también se pueden añadir servicios avanzados mediante el uso de conectores (*plugins*).

c) Open vSwitch: es un *switch* virtual multicapas *open source* (ASL) de altas prestaciones y funcionalidad extendida con una amplia gama de características incluyendo VLAN 802.1Q con puertos de troncal y de acceso, enlace NIC (con y sin LACP *upstream*), NetFlow/sFlow, QoS, GRE, GRE sobre IPSEC, VXLAN y LISP *tunnling*, gestión de fallos de conectividad 802.1ag, OpenFlow, reenvío a través del *kernel* de Linux y una base de datos de configuración transaccional. Open vSwitch puede funcionar completamente en el espacio de usuario con un módulo de *kernel* o como un conmutador basado en *kernel* que soporta múltiples tecnologías de virtualización incluyendo Xen/XenServer, KVM y VirtualBox.

Resumen

Como se ha podido observar en este módulo, el mundo *cloud* goza de una perspectiva alentadora de futuro y las tendencias (informe de RighScale) son hacia estrategias *multi-clouds*, centrándose en una integración (*clouds* híbridos) donde los entornos DevOps toman fuerza dentro del ecosistema *cloud* como PaaS y con actores muy definidos dentro de cada rubro. En EuroStat se pueden observar las tendencias y cómo esta tecnología está siendo usada por las empresas y se pueden extraer las tendencias del mercado en el futuro próximo.

También es necesario contemplar las recomendaciones de la EC (*Opinion 05/2012 on Cloud Computing, Opinion 02/2015 on C-SIG Code of Conduct on Cloud Computing*) en relación con la protección de datos, así como la Estrategia *Cloud* en Europa y la financiación a esta tecnología en las propuestas en curso o en las que todavía se encuentran abiertas.

Actividades

1. Analizar dos hipervisores diferentes y extraer conclusiones de sus ventajas/desventajas.

2. Utilizar dos plataformas IaaS públicas analizando sus prestaciones, ventajas y desventajas.

3. Lo mismo para PaaS. Lo mismo para SaaS (a ser posible dentro del mismo ámbito de negocio/funcionalidad)

4. Comparar y extraer conclusiones sobre las ventajas de una plataforma PaaS orientada al desarrollo de aplicaciones y un *cloud* IDE.

5. Realizar un cuadro comparativo sobre las diferentes opciones del *cloud*, ventajas y desventajas; proponer un modelo de negocio basado en una de ellas teniendo en cuenta los condicionantes, modelo de monetización, infraestructura, «nicho» y todos los elementos que consideréis necesarios y que pueden condicionar un negocio basado en esta tecnología.

Glosario

AJAX Asynchronous JavaScript And XML (cast. JavaScript asíncrono y XML) Técnica de desarrollo web para crear aplicaciones interactivas o RIA.

appliances Entorno que incluye todo el software preconfigurado y listo para utilizar incluido el SO que se puede ejecutar sobre una máquina bare-metal o sobre un hipervisor.

bare-metal Expresión utilizada para referirse a una máquina sin SO instalado.

CAPEX Costo de inversión

cloud computing (cast. computación/servicios/informática en la nube) Propuesta tecnológica que permite acceder a aplicaciones o servicios remotos en forma ubicua a través de un navegador y que pueden ser aprovisionados/liberados bajo demanda y en un tiempo reducido.

containers, jails (cast. contenedores, jaulas) Técnica de virtualización del sistema operativo que permite altas prestaciones (ejemplo: Docker, LXC/LXD).

data lock-in Situación en la que una empresa proveedora cierra su actividad sin aviso previo y los datos del cliente quedan «atrapados» en los servidores del proveedor sin posibilidad de acceso.

high performance computing (cast. cómputo de altas prestaciones) Recursos dedicados a la ejecución de aplicaciones complejas con grandes necesidades de potencia de cómputo.

hipervisor Capa de software que se debe poner entre el SO de la máquina sobre la cual se desean virtualizar los recursos.

IaaS (cast. infraestructura como servicio) Se encuentra en la capa inferior y es un medio de provisionar cómputo, almacenamiento y red como servicios estandarizados en la red.

infiniband Red de comunicación de altas prestaciones (por ejemplo, 40 Gbits).

kernel Núcleo de la aplicación o del sistema operativo, es decir, las funciones esenciales que conforman la aplicación y el SO.

licencia *open source* Licencia que da soporte al código *open source*. Ejemplos de estas son GPL (GNU General Public License), ASL (Apache Software License), CC (Creative Commons), BSDL (Berkeley Software Distribution License).

multitenency (cast. multitenencia) Método que permite ejecutar una sola instancia del software en la infraestructura del proveedor y sirve a múltiples clientes manteniendo el aislamiento de los datos de cada uno.

software *open source* (cast. software de código abierto) Software que dispone una licencia de forma que los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente.

OpenMPI, OpenMP Librerías para la programación de aplicaciones distribuidas y de memoria compartida respectivamente.

OPEX Costos de operación.

PaaS (cast. plataforma como servicio) Encapsulación de un entorno de desarrollo (SO + aplicación + entorno) y una serie de módulos/complementos que proporcionan todos los elementos para que un desarrollador, por ejemplo, pueda trabajar inmediatamente sin preocuparse por la infraestructura hardware/software.

pay-as-they-go, pay-as-they-grow, pay-per-user, pay-as-xx Modelos de monetización de recursos y servicios en el cloud.

SaaS (cast. software como servicio) Aplicación completa ofrecida como un servicio, bajo demanda, accesibles a través de un navegador y donde el usuario no tiene control sobre ellas eliminando la necesidad de instalar cualquier software y sin preocuparse por el mantenimiento de la aplicación.

RIA (*rich internet applications*; cast. aplicación de internet enriquecida) Aplicación web que tiene la mayoría de las características de las aplicaciones de escritorio tradicionales (por ejemplo, implementadas en AJAX).

SLA (*service level agreement*; cast. acuerdo de nivel de servicio) Acuerdo escrito entre un proveedor de servicio y su cliente con objeto de fijar el nivel acordado para la calidad de dicho servicio.

SOA (*service-oriented architecture*; cast. arquitectura orientada a servicios) Arquitectura para diseñar y desarrollar sistemas distribuidos y que se utiliza para el descubrimiento dinámico y el uso de servicios en una red.

standby services Servicios preparados listos para ponerse en ejecución y que estarán disponibles en un tiempo mínimo.

REST (*representational state transfer*; cast. transferencia de estado representacional) Interfaz entre sistemas que utilizan HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc.).

vendor lock-in Dependencia de un proveedor de productos y servicios, que hace que no sea posible cambiar a otro proveedor sin costos de cambio sustanciales, aunque el cambio signifique una reducción de costos o mejores prestaciones.

virtualización Técnica mediante la cual se presenta una visión virtual de las capas subyacentes a la aplicación o al SO. En el caso de la virtualización hardware el SO «ve» una máquina hardware equivalente pero que es virtual.

web services (cast. servicios web) Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones desarrolladas en lenguajes de programación diferentes.

Bibliografía

Todos los enlaces han sido visitados en octubre de 2016.

[Acs] All AWS Customer Stories. <https://aws.amazon.com/solutions/case-studies/all/>

[Cca] Cloud Application Architectures. George Reese. 2009. O'Reilly Media, Inc.

[Cca] Considering Cloud Appliances for Private Cloud Deployments. Would you prefer to build a cloud from components or plug it in and go? Vince Vasquez. Cloudbook Journal. Vol. 2. Issue 3, 2011. <http://www.cloudbook.net/resources/stories/considering-cloud-appliances-for-private-cloud-deployments>

[Ccp] Cloud Computing: Paradigms and Technologies. A. Shawish, M. Salama. Inter-cooperative Collective Intelligence: Techniques and Applications. 2013. Springer. 495. Págs. 39-67. <http://link.springer.com/chapter/10.1007%2F978-3-642-35016-0_2>

[Cis] Magic Quadrant for Cloud Infrastructure as a Service. L. Leong, G. Petri, B. Gill, M. Dorosh. 2016. ID: G00278620. Gartner. <https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519>

[Cmh] Magic Quadrant for Cloud-Enabled Managed Hosting. D. Toombs, B. Gill, M. Dorosh. 2015. ID: G00269143. Gartner. <https://www.gartner.com/doc/reprints?id=1-2K50B8G&ct=150729&st=sb>

[Cvh] Containers vs Hypervisors: The Battle Has Just Begun. Russell Pavlicek. 2014. Linux.Com. <https://www.linux.com/news/containers-vs-hypervisors-battle-has-just-begun>

[Dcc] The NIST Definition of Cloud Computing. P. Mell and T. Grance. National Institute of Standrs & Technology. Special Publication 800-145. 2011. <http://dx.doi.org/10.6028/NIST.SP.800-145>

[Eoc] The Evolution of the Cloud. The Work, Progress and Outlook of Cloud Infrastructure. Ari Liberman García. MIT. 2015. https://dspace.mit.edu/bitstream/handle/1721.1/100311/932065967-MIT.pdf?sequence=1

[Iar] Iconos con licencia de uso libre. <http://www.iconarchive.com> <http://www.customicondesign.com> <http://icons8.com>

[Idd] Info World Cloud Computing Deep Dive. 2009. <http://www.infoworld.com/resources/15675/cloud-security/download-the-cloud-security-deep-dive>

[Mcc] A Survey of Mobile Cloud Computing Application Models. A. Khan, M. Othman, S. Madani, S. Khan. IEEE Communications Surveys & Tutorials. 2013. Vol. 16 (1). http://dx.doi.org/10.1109/SURV.2013.062613.00160

[Ona] OpenNebula. Open Cloud Reference Architecture. 2015. OpenNebulaSystems. <https://support.opennebula.pro/hc/en-us/article_attachments/202208405/OpenNebula-Open_Cloud_Reference_Architecture_Rev1.0_20150421.pdf>

[Osc] A Guide to Open Source Cloud Computing Software. C. BryantJune. 2014. <http://www.tomsitpro.com/articles/open-source-cloud-computing-software,2-754.html>

[Osd] OpenStack Documentation. <http://docs.openstack.org/#docs-main-body>

[Ovf] Open Virtualization Format Specification. DSP0243. 2015. <http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.1.pdf>

[SaO] SaaS. Apprenda. 2016. <https://apprenda.com/library/software-on-demand/>

[Saa] SaaS. J. Moreno Martín. 2016. SaaSMania. <http://www.saasmania.com>

[Tif] The Information Factories. George Gilder. 2006. Wired. <https://www.wired.com/2006/10/cloudware/>

[Tch] Timeline of Computer History. <http://www.computerhistory.org/timeline/2015/>

[Wtn] 15 Ways to Tell its Not Cloud Computing. James Governor. 2008. RedMonk. <http://redmonk.com/jgovernor/2008/03/13/15-ways-to-tell-its-not-cloud-computing/>

[Vcc] A View of Cloud Computing. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. 2010. Communications of the ACM. Vol. 53. Núm. 4. Págs. 50-58. https://doi.org/10.1145/1721654.1721672