Fundamentos y plataformas de *cloud computing*

PID_00286170

Remo Suppi Boldrito

Tiempo mínimo de dedicación recomendado: 5 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.

La revisión de este recurso de aprendizaje UOC ha sido coordinada por el profesor: Josep Jorba Esteve

Segunda edición: febrero 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Remo Suppi Boldrito
Producción: FUOC

Índice

Introducción

El *cloud computing* (o sus equivalentes en español como *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, administraciones y usuarios finales para prestar servicios y utilizar recursos en el ámbito de las tecnologías de la información y la comunicación, que está sustentada por un modelo de negocio viable económicamente.

Una definición interesante que aporta más precisión sobre el *cloud computing* es la que formula el Instituto Nacional de Estándares y Tecnología (NIST) de Estados Unidos, que afirma lo siguiente:

> «el *cloud computing* es un modelo que permite el 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 a proveedores, 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), como por ejemplo correo, redes sociales y 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 el otro, la interconectividad y la usabilidad de diferentes interfaces y medios de acceso son esenciales para su quehacer cotidiano (compras, ocio, servicios administrativos, negocio y una larga lista más).

1. Breve historia

Si bien tenemos presentes como precursoras a las grandes compañías que ofrecieron este tipo de servicios (por ejemplo, Yahoo! en 1994 y Google en 1998 [motor de búsqueda], Hotmail en 1997 [correo] y Amazon en 2002 y AWS en 2006), el *cloud computing* es un concepto que tiene sus raíces en los años sesenta y que está basado en las ideas de John McCarthy, profesor emérito de la Universidad de Stanford y cofundador del Laboratorio de Inteligencia Artificial del MIT, donde predijo, en 1961 durante un discurso para el centenario de la institución, que la tecnología de tiempo compartido (que estaba en auge en ese momento) de las computadoras podría conducir a un futuro en el cual el poder del cómputo e incluso aplicaciones específicas podrían venderse 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 fue una figura muy importante en el ámbito de la ciencia computacional, 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 en proyectos como ARPANET (predecesor de internet) o la interfaz gráfica de usuario, que permitió el acceso a servicios y recursos a personas no especialistas; son ideas que los expertos actuales coinciden en afirmar que se asemejan mucho a la idea de *cloud computing* tal 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 y procesamiento de la información, sino que ahora lo es el *cloud*, y que este será el responsable de almacenar los datos de cada individuo en el planeta y que se accederá a ellos por lo menos una vez en la vida. En 2013, Gilder ha publicado un libro, *Knowledge and Power. The Information Theory of Capitalism and How It is Revolutionizing Our World*, en el cual relaciona la economía, la 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 puede verse en el *Timeline of Computer History* [Tch15], pero probablemente los desarrollos más vinculados al *cloud* actual son el progreso de la web (Tim Berners-Lee propone las ideas en 1989 pero hasta 1990 no hay un prototipo del

World Wide Web) y su evolución más reciente en la web 2.0 y el despliegue 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 la búsqueda y el correo y páginas como por ejemplo Wandex en 1993, que pretendía ser un programa para medir el tamaño de internet, desarrollado en el MIT, pero que acabó siendo el primer buscador, y luego Yahoo! en 1994, Altavista en 1995 (que ha ido cambiando de propietarios: Overture, Yahoo! y ahora Microsoft), Hotmail en 1996 (posteriormente comprado por Microsoft en 1997, cuando ya contaba con 6 millones de usuarios de correo), o más recientemente también 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 la más innovadora del mundo por Forbes en los últimos cuatro años), que fue una de las primeras en ofrecer a sus clientes lo que hoy se llama *software como servicio* (SaaS) y que era un software para automatizar las ventas mediante una simple página web. Algunos expertos creen que este servicio actualmente puede ser una *plataforma como servicio* (PaaS) 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 en 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, que se amplía en 2006 con el lanzamiento de Amazon Web Services (AWS): Elastic Computer y Storage Cloud (EC2 y S3) son los servicios más importantes de AWS, para proveer instancias de un servidor bajo demanda para el cómputo y el almacenamiento, respectivamente. Se trata de un servicio orientado al negocio que permitió a las pequeñas empresas y a los 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 el navegador: su uso se popularizó y ganaron en seguridad y servicio. También implementaron sus propias infraestructuras *cloud*, por ejemplo, Microsoft Azure, que fue anunciado en octubre de 2008 (pero no comienza a prestar servicios hasta 2010); IBM en 2011 (lanza SmartCloud Framework para dar soporte a su proyecto Smarter Planet), o Oracle en 2012 (Oracle Cloud).

En cuanto a las plataformas más representativas (no las únicas) para desplegar *clouds*, en 2005 comienza a trabajarse en un proyecto de investigación orientado a este ámbito basado en software de código abierto, y nace en 2008 OpenNebula. En este año se forma el consorcio OpenNebula.org y es financiado por un proyecto de la UE (7.º Programa Marco): llegó a tener, en 2010, 16.000 máquinas virtuales configuradas. Los datos actuales para la versión actual de

OpenNebula (V6.x) son 100.000 descargas en el último año, 2.500 *clouds* conectados al *marketplace* y 300.000 *cores* gestionados en la instalación más importante con esta plataforma.

En 2010 Rackspace y la NASA lanzaron una iniciativa de *cloud-software* de código abierto y la llamaron OpenStack con el objetivo de ofrecer a las organizaciones la posibilidad de montar sus servicios de *cloud* sobre hardware estándar. En 2012 se creó la OpenStack Foundation para promover este software en la comunidad, con 200 socios (una gran parte de todos los actores importantes del mundo TIC). Openstack se ha popularizado y hoy puede encontrarse en las distribuciones más conocidas (Redhat, Ubuntu, SuSE...) o también en 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), con GPLv3. En 2011 Cloud.com es adquirida por Citrix Systems, y CloudStack pasa a ser un proyecto de la Fundación Apache y se distribuye mediante la licencia Apache Software License. CloudStack cuenta con un gran conjunto de usuarios y es una plataforma sólida y actualizada (la versión 4.15 es de 2021).

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

2. Características, ventajas y desventajas del *cloud computing*

Teniendo en cuenta las opiniones de los expertos (por ejemplo, Jamie Turner), además de la web 2.0, las grandes revoluciones que han facilitado la evolución del *cloud computing* son el avance de las tecnologías de virtualización, la proliferación a costos aceptables del ancho de banda y los estándares de interoperabilidad de software. En la opinión de estos expertos, todos estos aspectos han facilitado que hoy en día cualquier recurso 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 hay que creer que el *cloud* puede contenerlo y proveerlo todo, ya que, como todo paradigma, tiene sus elementos a favor (básicamente, el acceso a recursos y servicios por parte de los usuarios sin ser expertos en su instalación o administración, la flexibilidad de uso y adaptativa y los 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*.

Son interesantes el artículo *15 Ways to Tell Its Not Cloud Computing* [Wtn08], que expresa claramente lo que no es *cloud computing*, y los tres criterios expresados por George Reese [Cca09] sobre cuándo un servicio es *cloud*: accesibilidad vía navegador web (no propietario) o una API de servicio web, inicio sin aportar capital y pago únicamente por lo que se usa cuando se use.

De forma descriptiva, podemos enunciar las siguientes **características** clave para el *cloudcomputing* [Dcc11]:

1) Agilidad y autoservicio: se trata de la rapidez y capacidad de proveer recursos (de forma casi inmediata) sin grandes intervenciones ni acciones por ninguna 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, de forma no asistida, obtener capacidades de cómputo, almacenamiento o redes, según lo que requiera, 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), por medio de la red.

2) Costo: a causa de la eficiencia y la automatización en la gestión y administración de recursos, los precios resultantes para los proveedores de *cloud* son reducidos y pueden trasladarlos al usuario final, que no debe hacer frente a los gastos derivados de la implantación de infraestructura y servicios ni 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, pueden aplicarse medidas con un cierto nivel de abstracción apropiado para el tipo de servicio (por ejemplo, cuentas / cuentas activas, almacenamiento, procesamiento o ancho de banda) y es posible ajustar el costo a modelos de, por ejemplo, pago por uso, pago por peticiones, etc., lo que proporciona 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 la adaptabilidad a sus necesidades de carga y uso. Es decir, si puedo prever la carga o la 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: el recurso se independiza del acceso y se facilita que este acceso pueda ser desde una red local, corporativa o de otro tipo y desde diferentes dispositivos (de distintas capacidades y tipologías).

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

6) Rendimiento y gestión de recursos: se refiere al control exhaustivo y la monitorización eficiente de los servicios para lograr una alta disponibilidad y una 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 pueden configurarse para servir a múltiples usuarios en un modelo de multitenencia, los recursos físicos y virtuales pueden asignarse dinámicamente y reasignarse 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, el país o el centro de datos).

7) Seguridad: puede incrementarse, dada la particularidad de tener los datos centralizados. Al responsable de la aplicación le corresponderá la seguridad de esta, pero el proveedor deberá ocuparse de la seguridad física. Con ello, la seguridad será al menos igual que en los sistemas tradicionales, pero podrá mejorarse (y deberá estipularse que nivel se desea en el acuerdo de nivel de servicio, o SLA), ya que al proveedor le interesa que su infraestructura sea se-

gura para los clientes, como base de la calidad del negocio, y podrá invertir en ello globalmente, mientras que esta inversión puede ser inaceptable para un cliente si debe emprenderla 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 por el servidor y el espacio, lo que facilita el despliegue rápido de servicios y garantiza una alta disponibilidad (servicios en *stand by*) y movilidad por carga (movimiento de máquinas virtuales a servidores más liberados cuando la carga aumenta).

Para convencer a los más reticentes, puede enumerarse un conjunto de **ventajas** que aporta el *cloud* como paradigma. Son las siguientes:

9) Fácil integración y aceptación: teniendo en cuenta su desarrollo y base en estándares, puede integrarse 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.

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

11) Simplicidad: los roles y las responsabilidades están muy bien definidos. La separación de las actividades está bien estipulada (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 una menor inversión por todas partes.

12) Reducción de riesgos y rapidez: no se necesita ni una gran inversión ni la adecuación de entornos. Es posible acelerar al máximo la implantación de nuevos servicios en las diferentes etapas de desarrollo hasta la producción.

13) Reducción del uso de energía (consumo eficiente): a causa de la tecnología utilizada y el uso eficiente de los recursos (favorecido por la virtualización), solo se consume lo necesario, a diferencia de los centros de datos tradicionales, en los que que existe un consumo fijo independiente de la carga o 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 difundirse opiniones en contra que consideran que un *cloud público* quizás no es la opción más adecuada para un determinado tipo de servicios o infraestructura. Entre las principales causas de **desventajas** que esgrimen algunos expertos están las siguientes:

1) Centralización: tanto de los datos como de las aplicaciones. Genera una dependencia en relación con el proveedor, que si no dispone de la tecnología adecuada (monitorización y detección) y los recursos apropiados (alta disponibilidad), puede generar cortes o inestabilidades en el servicio. En estos casos toma especial relevancia el SLA, que especificará a qué está obligado el proveedor y las indemnizaciones a las que deberá hacer frente por ello.

2) Confiabilidad: la «salud» tecnológica y financiera del proveedor será un elemento clave en la continuidad de su negocio y del nuestro, por lo cual las decisiones que tome el proveedor afectarán directamente al negocio del cliente, y si no son las adecuadas, impactarán directamente en la empresa. También la empresa quedaría a la merced de un mercado muy dinámico en cuanto a fusiones y monopolios (o pseudomonopolios), con el impacto que podría tener en los costos de los servicios. Un problema importante es que el proveedor, por varios motivos (legales, financieros, económicos), cierre su actividad de forma abrupta y se produzca el *datalock-in* de la empresa en los servidores del proveedor sin posibilidad de poder recuperarlos (son situaciones que han ocurrido a raíz del 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 disponga de más clientes, más usuarios habrá sobre el hardware, la sobrecarga 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, puede llegarse 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 podría tener una prioridad muy baja (y su consiguiente retardo en el despliegue) para el proveedor si no son requeridos por suficientes clientes, lo cual puede afectar 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 este de manera confiable y con un ancho de banda aceptable, el *cloud* deja de tener efectividad.

6) Riesgo y privacidad: los datos «sensibles» del negocio no están en las instalaciones de las empresas y la seguridad no depende de sus recursos humanos, 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 resolverse mediante canales y servicios seguros, la posibilidad

de un fallo en la cadena de cifrado de la información es real, 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 (en la actualidad se ha demostrado que es uno de los más frecuentes) que origina que un cliente dependa de un proveedor de productos y servicios y sea 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 que si después desea cambiarse, comportará costos muy elevados.

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

Como puede verse, 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), y el equipo de toma de decisiones de la empresa deberá analizar cuidadosamente las posibilidades y qué ventajas existen frente a los riesgos que asumirá. Desde un punto de vista global, es una tecnología que aporta beneficios y que, con una planificación y toma de decisiones adecuadas, valorando todos los factores que influyen en el negocio y escapando de conceptos superficiales (todos lo tienen, todos lo utilizan, bajo costo, expansión ilimitada, etc.), puede ser una elección apropiada para los objetivos de la empresa, su negocio y la prestación de servicios TIC que necesita o que forma parte de sus fines empresariales.

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

- **Infraestructura**: recursos de 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.

- **Administración del *cloud***: cuerpo técnico de profesionales que garantizará las prestaciones, seguridad, estabilidad, disponibilidad, etc., de los servicios comercializados en función del SLA firmado con cada cliente.

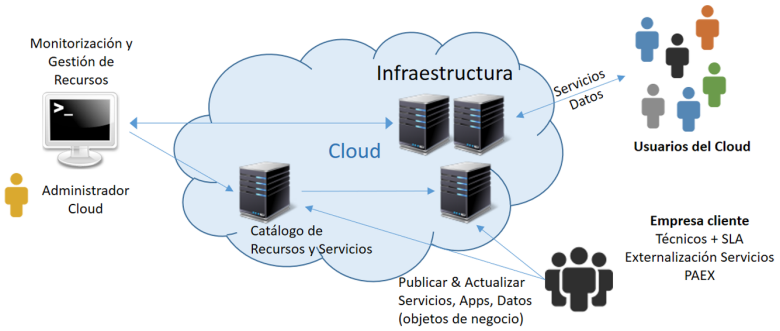
- **Empresa cliente**: usuarios que externalizan sus servicios TIC y publican y actualizan sus aplicaciones o datos en el *cloud*. Se basan en una garantía

del servicio que ofrece el proveedor, que forma parte de un SLA. Generalmente, el costo se basará en políticas similares al pago por uso.

- **Usuarios**: clientes de la empresa que ha puesto sus servicios en el *cloud* y que pueden saber o no que estos se proveen desde el *cloud*. Solo acceden a un servicio, aplicación o datos como si entraran 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 [Acs] (es decir, las empresas que tienen la etiqueta «millones de dólares» asociada a alguna característica [ingresos, activos totales, financiación, valoración o beneficios]), encontramos a Adobe Systems, Airbnb, Alcatel-Lucent, Aon, Autodesk, BMW, Bristol-Myers Canon, Capital One, Comcast y Docker, entre otros muchos. Si tomamos, por ejemplo, a Adobe Systems, podremos ver el rol de cada actor: el **proveedor del *cloud*** y la infraestructura es **AWS**, la **empresa** que los contrata es **Adobe**, y esta comercializa los productos por medio de **Creative Cloud**, que permite que **usuarios** (finales) accedan a software de diseño gráfico, edición de vídeo, 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 el 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 pueden enumerarse:

1) *Cloud* público: en este caso, el sistema está abierto para su uso general mediante diferentes modelos de negocio (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 a través de internet. Generalmente, un cliente no sabe con qué otros usuarios comparte la infraestructura, su utilización se contrata por medio de un catálogo de recursos disponibles y su aprovisionamiento es automático (o semiautomático) después de haber cumplido con una serie de trámites en cuanto al modelo de pago y el SLA.

El propietario del *cloud* (proveedor de recursos y servicios) puede ser una empresa privada, una institución académica, una organización gubernamental o una combinación de ellas, generalmente, en función del tipo de clientes y de los recursos que deberán proveerse. Técnicamente, puede haber pocas diferencias (o ninguna) con alguno de los otros modelos (especialmente con el privado); sin embargo, puede haber diferencias sustanciales en relación con la seguridad: el *cloud* público puede estar disponible en una red abierta como internet y sin mecanismos de cifrado.

> Como ejemplos de estos servicios, podemos mencionar AWS, Microsoft Azure, Google Compute, Rackspace e IBM SoftLayer (hasta 2014, IBM SmartCloud), que operan en su propia infraestructura y cuyos recursos son accesibles en internet, y el Consorcio de Servicios Universitarios de Cataluña (CSUC), que ofrece, con diferentes modelos de explotación, recursos de *cloud* para instituciones académicas y de investigación de Cataluña (https://www.csuc.cat/es/servicios/infraestructura-en-la-nube).

Es interesante mencionar que algunos proveedores intentan reducir la probable inseguridad mediante servicios de conexión directa (por ejemplo, AWS Direct Connect y 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 puede analizarse en [Cis16] (donde examinan los puntos fuertes y débiles de cada operador, elaborado 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).

2) *Cloud* privado: en este supuesto, la infraestructura funciona exclusivamente para una organización o empresa y es utilizada por sus unidades de negocio o departamentos. La organización o empresa puede ser propietaria, administradora y operadora, o alguna de estas funciones puede subcontratarse a un tercero. Es la opción más favorable para las compañías que necesitan una alta protección de datos y un alto nivel de servicio, ya que los recursos y su gestión están bajo el control y el cuidado de la propia empresa u organización.

Si bien resuelve los 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 siguiendo sus criterios) y que controla qué aplicaciones deben ejecutarse y dónde deben hacerlo, y que mantiene la privacidad de su información, lo que permite definir las políticas de acceso y evita el *datalock-in* y el *vendor lock in* mencionados anteriormente.

3) *Cloud* híbrido: en esta organización del cloud, se combinan modelos públicos y privados. El propietario dispone de una parte privada (con acceso restringido y bajo su control) y comparte otras, aunque de una manera controlada. Los *clouds* híbridos 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.

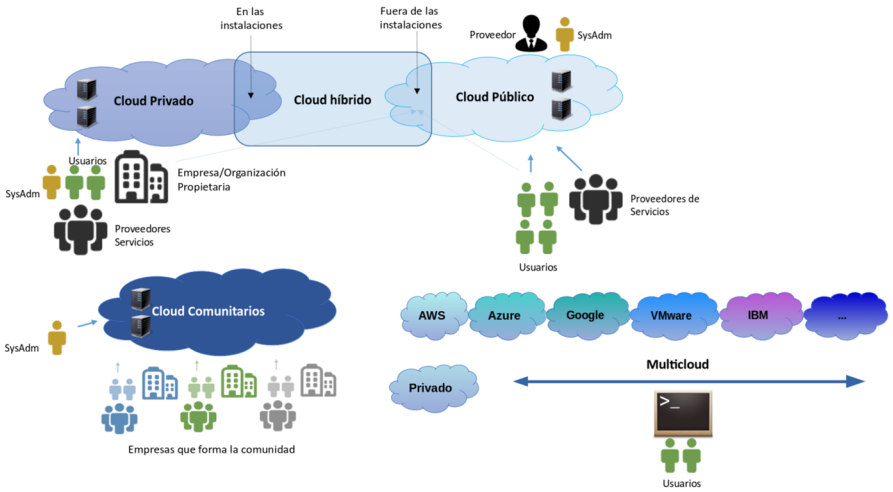
Un ejemplo de ello son algunas implementaciones de correo empresarial o aplicaciones de ofimática.

4) *Cloud* comunitario: los servicios están diseñados para que puedan utilizarse por una comunidad, organizaciones o empresas con objetivos o negocios alineados (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 la infraestructura, así como los gestores u operadores, o algunos de estos roles pueden subcontratarse a terceros, pero siguiendo las indicaciones y reglas que se aplican a la comunidad.

5) *Multicloud*: si bien este modelo de desplegamiento no está considerado en la taxonomía del NIST, se trata de una variación que tiene mucha aceptación actualmente. El *multicloud* representa la implementación del *cloud* por distintos proveedores o de diversas soluciones dentro del cloud privado, integrados y administrados por un solo panel de control. Las ventajas comentadas para los distintos *clouds* se amplían: aumentan la flexibilidad, la escalabilidad, la seguridad y la agilidad para hacer frente a las distintas necesidades de la organización.

Además, este tipo de organización ofrece ventajas en el costo de los servicios, ya que como cada proveedor aportará diferentes precios sobre el uso del cómputo o almacenamiento, ello permitirá al usuario utilizar los recursos más adecuados y con un menor costo en el despliegue de soluciones, para hacer un uso más eficiente de la inversión. En relación con la seguridad, se incrementa por la distribución de los datos, lo que mitiga el efecto de ataques de denegación de servicio u otros que solo afectarían a un único proveedor (es muy poco probable que todos los proveedores de *cloud* se vean afectados por un mismo incidente de seguridad). Funcionalmente, no se observarán muchas diferencias respecto a un *cloud* híbrido, más allá de la utilización múltiple tanto de proveedores públicos como privados.

La siguiente figura muestra un esquema de las cinco formas de implantación y desarrollo de los clouds.



Otra de las clasificaciones habituales es por el nivel de servicio que se presta. Tradicionalmente, se distinguían tres categorías: *infraestructura como servicio* (**IaaS**), *software como servicio* (**SaaS**) y *plataforma como servicio* (**PaaS**). Hoy pueden encontrarse otras extensiones o derivadas de las habituales: *business as a service* (**BaaS**), *storage as a service* (**StaaS**), *desktop as a service* (**DaaS**), *disaster recovery as a service* (**DRaaS**), *marketing as a service* (**MaaS**), o algunos autores le dan un nombre global con *everything as a service* (**XaaS**), para referirse a la creciente diversidad de servicios disponibles en el *cloud* a través de internet (puede consultarse [Idd09]). Un caso que afirma estas consideraciones es el de *application platform as a service* (**aPaaS**), que permite un rápido desarrollo y aprovisionamiento de aplicaciones, como plataforma específica para codificar, aprovisionar y desplegar *apps*, y que es compatible con el ciclo de vida completo, proporcionando una forma más rápida de crear aplicaciones.

Ejemplos de aPaaS

Su aceptación se comprueba por la amplia propuesta del mercado: AWS con Elastic Beanstalk, CenturyLink con AppFog, Google con App Engine e 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 colocar sus SO y sus aplicaciones e implementar un servicio o utilizarlos para ejecutar sus aplicaciones. El cliente no podrá manejar o con-

trolar el hardware subyacente, pero, a partir del SO, sí que podrá manejar el almacenamiento y las aplicaciones o servicios implementados sobre esas máquinas, así como algunos aspectos de la conectividad (subredes, cortafuegos, dominios...).

Ejemplos de grandes proveedores de IaaS (decenas de miles de máquinas virtuales) son Amazon EC2, Google Compute Engine o Digital Ocean, como servicio representativo para pymes (en unidades de máquinas virtuales).

La reflexión del usuario en esta modalidad sería: «¿Por qué comprar, instalar y probar infraestructura cada X años [obsolescencia], y no alquilarla y pagar por uso?». Los recursos necesarios se obtienen sin acometer obras (centro de datos), con los mínimos recursos humanos (no especialistas en TIC), sin una gran inversión inicial y de manera escalable y eficiente y a costos aceptables.

La **plataforma como servicio** (PaaS) es la encapsulación de un entorno 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 o aplicaciones, librerías y API para un fin específico.

Por ejemplo, para una tecnología de desarrollo en particular, sobre Linux, un servidor web, una base de datos y un entorno 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 de su entorno de ejecución (bases de datos y *middlewares*). Es habitual que una PaaS pueda prestar servicios en todos los aspectos del ciclo de desarrollo y las pruebas de software o en desarrollo, gestión y publicación de contenidos.

Ejemplos de ello serían Cloud9, Google App Engine, Heroku o OpenShift.

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

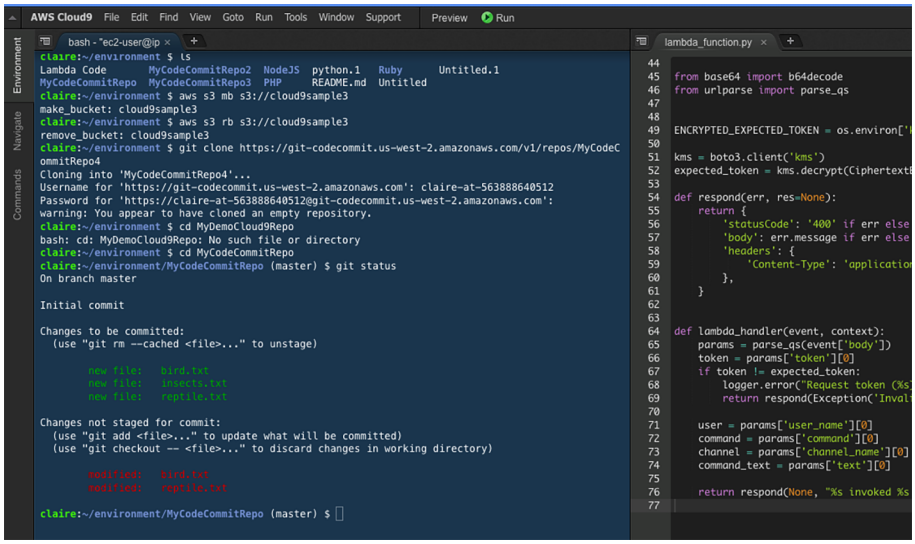
Finalmente, el **software como servicio** (SaaS) se encuentra en la capa abstracta (si bien existe una tendencia bastante extendida a considerar que el SaaS es la capa más simple del PaaS, o la más baja) y caracteriza una aplicación completa ofrecida como un servicio, generalmente bajo demanda con multitenencia (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 con 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 aplicar algunas configuraciones. Esto elimina al cliente

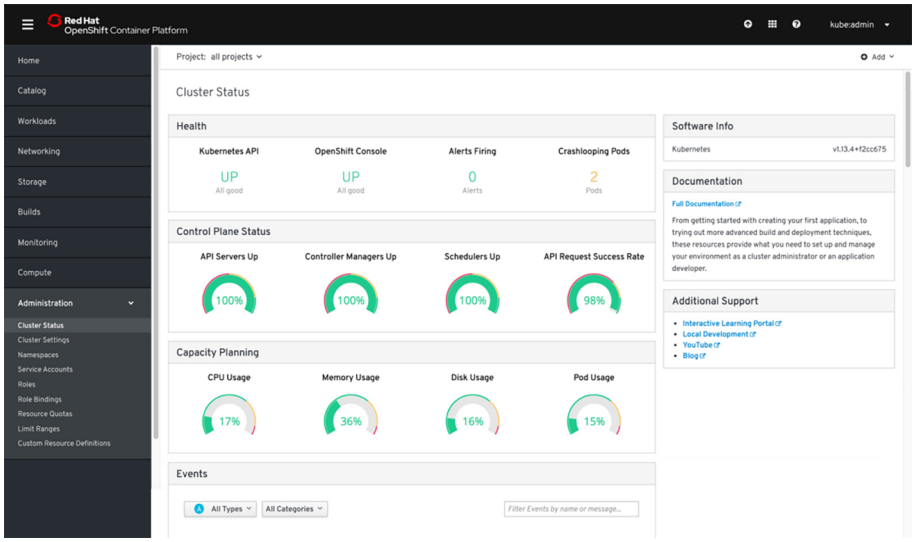
la necesidad de instalar la aplicación en sus propios ordenadores, suprime el soporte y el mantenimiento del hardware y software y mejora el control y la gestión de las licencias, si son necesarias.

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

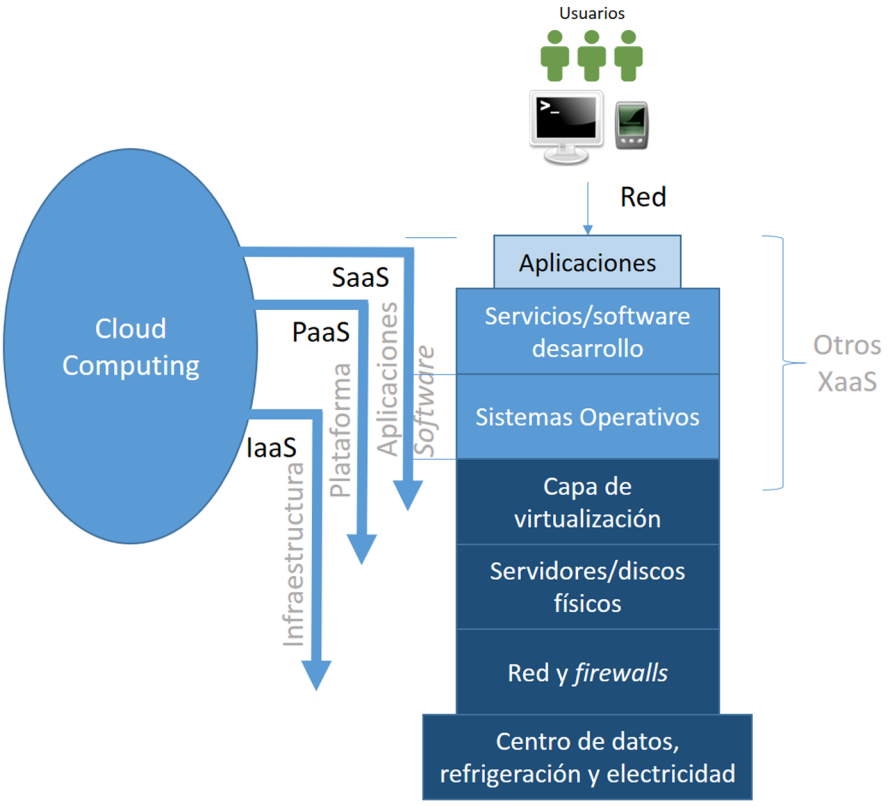
En esta modalidad, podría resumirse así el pensamiento del usuario: «Ejecute esto por mí», sin responsabilidades en la gestión de hardware o software, utilizando un navegador web o evitando la instalación de software cliente, basado en un servicio bajo demanda, con escalabilidad casi inmediata, acceso desde cualquier sitio y con cualquier dispositivo, con 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 (propiedad de AWS desde 2016) y de Openshift (donde pueden obtenerse cuentas gratuitas para pruebas de concepto con diferentes modalidades de funcionamiento y desarrollo), como ejemplos de la facilidad y simplicidad de poner en marcha entornos complejos solo en unos segundos, preparados para desarrollar aplicaciones o servicios sobre ellos.





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



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

1) Clúster: 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. Se utilizan 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 una red de altas prestaciones (p.ej. InfiniBand), que, utilizando en su gran mayoría software de código abierto (Linux; NFS —aunque cada vez más está reemplazándose por otros sistemas de archivos distribuidos como Ceph, Lustre y GlusterFS—; 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*), permiten desarrollar y ejecutar aplicaciones concurrentes o distribuidas de altas prestaciones. Esto permite disponer de una infraestructura que provee de alto rendimiento, alta disponibilidad, con equilibrio de carga, eficiente, escalable y a costos razonables. Existe una clasificación de los clústeres en función de qué característica predomina: alto rendimiento (*high performance computing*, o HPC), alta disponibilidad (*high availability computing*, o HA/HAC) y alta eficiencia (*high throughput computing*, o HT/HTC).

2) Superordenador: es una evolución funcional a gran escala de un clúster (si bien puede tener diferentes arquitecturas) que posee una capacidad de cómputo mucho más elevada que este (y, por lo tanto, que un ordenador de propósito general). Su rendimiento de mide en teraflops (10^{12} operaciones de coma flotante por segundo) y el más potente en junio de 2021, publicado en la lista del Top 500 (los 500 superordenadores más potentes del mundo), es el superordenador Fugaku, en el Centro de Ciencias de la Computación RIKEN (Japón), con 7.630.848 *cores*, 442,010 teraflops y con un consumo de 29,9 megavatios.

Su historia se inicia a finales de los cincuenta. IBM, Sperry Rand y Cray fueron los grandes actores de aquella época con arquitecturas específicas (inicialmente con Univac, como el primer ordenador comercial, y posteriormente IBM7030 o Cray-1). Les siguieron 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 (*symmetric multiprocessor*, o SMP) y luego sin compartición de memoria (*massively parallel processor*, o MPP). Luego, en los noventa derivan en clústeres de miles de procesadores, y a finales de la década y de forma masiva, en los superordenadores con millones de *cores* de los sistemas actuales.

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

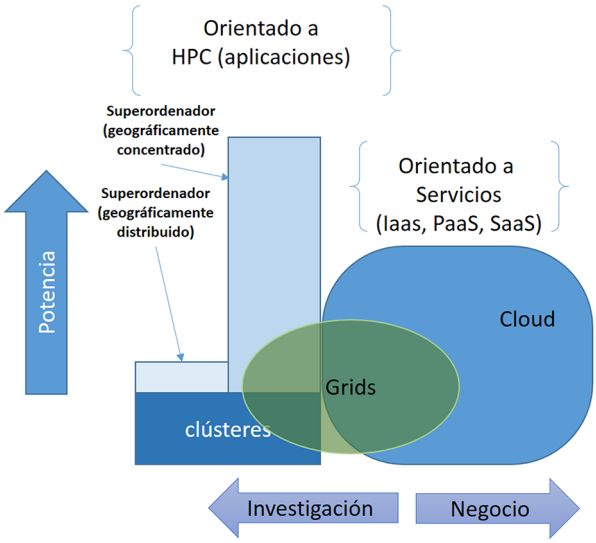
a) La evolución del clúster (como por ejemplo en el MareNostrum, en el BSC [Cataluña]; el Finisterrae, en el CESGA (Galicia), o el Juwels, en el Jülich Supercomputing Centre [Alemania]). Se dedica al cómputo de altas prestaciones, comparte un espacio físico y dispone de una serie de recursos comunes como redes de comunicación de alta velocidad o espacio de almacenamiento compartido y distribuido.

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

Ejemplos representativos de este último son proyectos como Seti@home, Folding@home, World Community Grid y ClimatePrediction.net. Muchos de ellos están basados en Boinc (una arquitectura de software de código abierto que permite que voluntarios aporten sus recursos a un problema común, o paradigma conocido como *volunteer computing*).

3) *Grid*: se refiere a una infraestructura que posibilita 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 fueran 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 los *grids* fueron establecidas por Ian Foster y Carl Kesselman, que fueron los precursores con la creación de Globus Toolkit (hoy en día obsoleto), considerado como la primera herramienta para construir *grids*. Entre los grandes proyectos de *grid*, pueden encontrarse CrossGrid, EU-DataGrid o el reciente EGI-InSPIRE (o simplemente EGI, hoy en día transformado en un servicio a los investigadores como EOSC-Hub), todos ellos financiados por la UE. Es fundamental destacar que el *grid* es un tipo de infraestructura de cómputo cuyo ámbito de utilización y propósito natural es el científico (*e-science*), en las universidades y los laboratorios de investigación, mientras que el *cloud* nace de la prestación de servicios y 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. Son considerados por algunos expertos como las dos vías del uso masivo de recursos (con diferentes modelos): una exclusivamente para la ciencia (*grid*) y otra para las empresas (*cloud*). Es importante destacar que comienzan a ofrecerse servicios cruzados, es decir, altas prestaciones (HPC) en el *cloud* (por ejemplo, instancias con base en hardware de GPU).

Con esta caracterización de los recursos y su utilización, puede situarse gráficamente cada una de estas arquitecturas en función de la potencia de cómputo involucrada y su tipo de dedicación (puede verse la siguiente figura).



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

1) 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). Este tipo de paradigma se utilizará 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.

2) SOA y WS: no son conceptos nuevos (se definieron en la década de los noventa). La arquitectura orientada a servicios (SOA) se define 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. Los servicios web (WS) son 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* se prestan generalmente por medio de estas tecnologías, las cuales también pueden utilizarse para su organización y gestión interna.

3) 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 World Wide Web. Evidentemente, esta tecnología aporta mucho al *cloud computing*, ya que la interacción y la prestación de servicios con el usuario y los clientes (y entre aplicaciones) serán a través de las posibilidades que brinda mediante los estilos (CSS), lenguajes (HTML5, XML, XHTML, Javascript, RoR…), aplicaciones RIA (aplicaciones de internet enriquecidas, AJAX), agregación (RSS, ATOM), JavaScript Client Communication, interoperabilidad (transferencia de estado representacional, o REST), API (WebGL, DOM…), for-

matos de datos (JSON, XML) o *mashups* (aplicaciones web híbridas). Para algunos investigadores y expertos, ya se ha llegado a la web 3.0, pero todavía hay muchas opiniones sobre qué es y qué no es 3.0.

Dentro de esta comparación, puede argumentarse que la tecnología principal que sustenta el *cloud computing* es la **virtualización**. La capa de virtualización (hipervisor; por ejemplo, Xen, VirtualBox, OracleVM, KVM, VMware ESX/ESXi y Hyper-V, entre otros) separa un dispositivo físico en uno o más dispositivos «virtuales» (llamados *máquinas virtuales*): cada uno de ellos puede ser asignado, utilizado y gestionado 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 clonación o *pool of VM in stand-by*) para asignarse a un usuario bajo demanda y en pocos segundos.

Un paso adicional que ha permitido un incremento notable de la eficiencia y la disponibilidad y una revolución en los entornos de desarrollo ha sido la **virtualización a escala de sistema operativo**, para crear 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: 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 *contenedores*, *virtualization engines* (VE) o *jaulas* (por ejemplo, FreeBSD Jails or Chroot Jail). A ojos de un usuario de 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 los sistemas Linux, es una evolución del mecanismo de Chroot más un sistema de control o 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 pueden tenerse contenedores con sistemas operativos que no compartan el mismo núcleo (por ejemplo, si el *host* es Linux, no se dispondrá de un contenedor con Windows, cosa que sí es posible si fuera una máquina virtual). Debe tenerse en cuenta que Windows incluye Windows Subsystem for Linux, que puede utilizarse para ejecutar Linux sobre Windows o disponer entornos de máquinas virtuales y contenedores Linux sobre el sistema operativo Windows 10. Entre el software de virtualización del SO de código abierto podemos encontrar Docker, LXC/LXD, OpenVZ y FreeBSD Jails, y en software propietario, Virtuozzo (basado en OpenVZ).

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

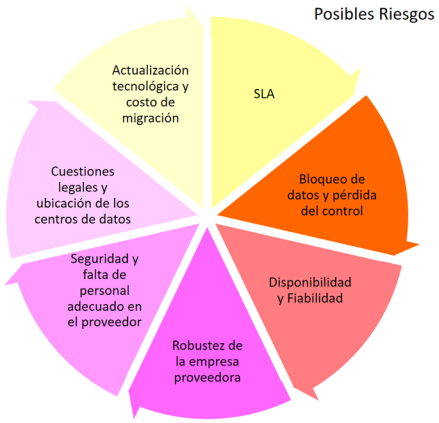
4) ***Dew computing***: es un nuevo paradigma de cómputo distribuido que considera que los equipos locales (ordenadores de sobremesa, portátiles y dispositivos móviles) conceden un entorno propicio para microservicios independientes de los servicios *cloud*, y que estos pueden colaborar con ellos. Es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

5) ***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 el *cloud* para ello. Se considera una forma de computación distribuida de proximidad, en la que cada uno de los dispositivos conectados a la red puede procesar los datos y transmitir solo un conjunto de ellos que sean de interés, o hacerlo únicamente en situaciones excepcionales (alarmas o notificaciones), teniendo una capacidad autónoma y sin dependencias del servidor en el *cloud*. Es un modelo que, con el crecimiento que se espera del IoT (internet de las cosas), permitirá que los sistemas *cloud* y las redes no se colapsen ante el incremento exponencial de datos por transmitir, almacenar y procesar. Algunos autores lo diferencian del ***fog computing***, en el que se utilizan agrupaciones o multitudes de usuarios finales (o dispositivos cercanos al usuario) para almacenar datos (en lugar de hacerlo sobre el *cloud*), comunicar (en lugar de hacerlo a través de internet), controlar, configurar, medir y gestionar (en lugar de que controlen principalmente los *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 Consortium).

6) ***Peer-to-peer computing***: este paradigma plantea el uso de una arquitectura distribuida sin la necesidad de una coordinación central para llevar a cabo el cómputo y el 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 de forma igualitaria. Los *peers* 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…), compartición de archivos (Gnutella, eDonkey), dinero digital (Bitcoin, Peercoin) y anonimización (I2P), entre otros.

7) ***Volunteer computing***: como se ha mencionado anteriormente en el *grid*, este tipo de procesamiento distribuido se basa en que 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 de 1996, con Great Internet Mersenne Prime Search (búsqueda de números primos que cumplan con $2^n - 1$; por ejemplo, 3, 7, 31...), seguido en 1997 por distributed.net, y a continuación muchos otros, entre ellos Bayanihan, cuyos desarrolladores propusieron el nombre de *volunteer computing*. En la actualidad, muchos de estos proyectos están desplegados a partir de la arquitectura Boinc mencionada anteriormente (desarrollada por la Universidad de California en Berkeley), como por ejemplo Seti@home, y también están aquellos que han desplegado su propia arquitectura de software, como Folding@home (desarrollado inicialmente en la Universidad de Stanford).

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



4. Plataformas más representativas

Más allá del extenso catálogo en las diferentes plataformas y opciones propietarias, existe un gran conjunto de plataformas basadas en el software libre que permiten desarrollar y poner en marcha infraestructura de *cloud computing* sobre hardware comercial, pero sin grandes requerimientos. Estos aspectos deben valorarse 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; dependiendo del tamaño de la empresa, las necesidades y el control de la información que desee tenerse es totalmente viable y muchas de ellas son utilizadas por incluso por 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 de código abierto en las empresas, que trabajan con una versión *community* y otra versión empresarial con mayor soporte o adaptaciones a las necesidades específicas del cliente, o con SLA o QoS diferenciados, pero utilizando el mismo software que la versión de código abierto. Generalmente, se observa en las versiones de código abierto un factor de innovación y crecimiento continuo que presenta nuevas adaptaciones, servicios y mejoras en un ciclo de evolución más rápido y constante que el de sus correspondientes propietarios, con la producción de nuevos estándares y nuevas API públicas, lo que favorece la generación de conocimiento y posibilidades adaptadas a todas las necesidades [Bcs].

No obstante, esta proliferación de plataformas y servicios puede desconcertar, originar cierta confusión y complicar la elección, pero debe tomarse desde un punto de vista pragmático para analizar cuál de ellos se acerca más a las necesidades y se adecúa a la infraestructura de hardware de la 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 otra parte, ha evolucionado en todos los aspectos y hoy es posible encontrar gran cantidad de proveedores (algunos de ellos ya se han mencionado), como puede observarse en las recomendaciones y la lista 50 Best Cloud Computing Companies to Work for in 2021 Based on Glassdoor de Forbes, o la The 100 Coolest Cloud Computing Companies of 2020 de CRN.

4.1. Hipervisores

El hipervisor (monitor de máquina virtual, *virtual machine monitor* o VMM) es la primera capa que debe colocarse 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 de software, firmware o hardware, es la responsable de crear y ejecutar máquinas virtuales (llamadas *guest*) sobre el ordenador base (llamado *host*). El hipervisor permite (y gestiona) que las máquinas virtuales puedan ejecutarse simultáneamente, cada una con su sistema operativo, y compartir los recursos de hardware de 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 (los 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* (aunque este concepto se ha transformado en el caso de Windows 10 desde la integración de WSL en el *kernel* del sistema operativo).

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 AMD, con AMD-V (Pacifica) (debe destacarse que, según los datos del primer trimestre de 2021, provenientes de MindFactory en Europa, AMD con su procesador Ryzen 5 3600/3700 está dominando el mercado europeo y se ha reducido mucho el mercado de Intel, que había sido dominante hasta el lanzamiento de la nueva arquitectura de AMD basada en Zen 3).

Otra técnica alternativa para la virtualización es la llamada **paravirtualización**, la cual presenta una interfaz de software para 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) sea más simple y pue-

da 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 *paraAPI* para que disponga de las extensiones necesarias para comunicarse con el hipervisor. Xen, la plataforma más conocida y utilizada por los grandes proveedores, implementa esta técnica y puede ejecutarse con dos tipos diferentes de *guest*: sistemas paravirtualizados y virtualización completa con asistencia de hardware. Ambos tipos pueden utilizarse simultáneamente en un mismo sistema Xen y también pueden usarse técnicas de paravirtualización en un *guest* con asistencia de hardware.

Los diferentes tipos de plataformas de virtualización utilizadas en la actualidad pueden clasificarse en dos tipos: aquellas en las que el hipervisor gestiona el hardware directamente, es decir, hipervisor y SO *host* forman un conjunto indivisible (*bare-metal hypervisor* o tipo 1), o aquellas en las que 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 puede apreciarse, hay algunas opciones, como Linux's Kernel-based Virtual Machine (KVM) y FreeBSD bhyve, que son módulos del *kernel*. Por lo tanto, instalarlos sobre Linux o sobre BSD convierten el SO en hipervisor de la primera categoría, pero algunos autores, al ser módulos, lo clasifican como de la segunda categoría. A continuación, extenderemos algunas consideraciones sobre los hipervisores más representativos.

1) Xen Project Code: este hipervisor ha sido utilizado por los grandes proveedores, como AWS, Rackspace Hosting y Verizon Cloud, entre otros, y puede instalarse desde las distribuciones de Linux que contienen los paquetes, instalar sus 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 Fundación Linux como proyecto colaborativo, el hipervisor ha pasado por diferentes fases (hoy en https://

xenproject.org/). En relación con el *cloud*, en 2009 nace XCP (Xen Cloud Plataform), una distribución binaria de Xen Project Management API (o XAPI), y surgen diferentes integraciones del proyecto Xen utilizando XAPI, como Eucalyptus, Apache CloudStack, OpenNebula y OpenStack; aparecen los primeros *clouds* públicos con XAPI y OpenStack. En 2012 los paquetes XAPI se incluyen en Debian y Ubuntu Server, lo que permite crear los primeros *clouds* utilizando distribuciones Linux. En 2013 XenServer (un *superset* de XCP) es liberado como código abierto por Citrix (desde 2019 se considera obsoleto http://xenserver.org/). Hoy en día, el proyecto Xen, además de una nueva versión 4.15 en 2021, tiene un área centrada en 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 un hardware mínimo. Los proyectos de *unikernels* Mirage OS, Unikraft y OSv son algunos de los más representativos y conocidos como *Xen Project-powered*.

2) KVM: es la sigla 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 al entorno de virtualización (kvm-intel.ko o kvm-amd.ko). Con KVM pueden ejecutarse múltiples máquinas virtuales (Linux o Windows) sin modificación alguna sobre ellas, y cada una podrá acceder al hardware virtualizado (red, disco, tarjeta gráfica…). 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 de código abierto que permite la emulación y virtualización de un determinado hardware. La emulación permite que un programa o SO para una arquitectura (por ejemplo, ARM) pueda ejecutarse en otra (por ejemplo, x86), mientras que la virtualización permite la ejecución del código nativo directamente sobre la CPU *host*. QEMU admite la virtualización cuando se ejecuta bajo un hipervisor Xen o cuando utiliza el módulo de KVM en el *kernel* (utilizándolos como aceleradores y sin usar la emulación). La razón de esto se explica por cómo se virtualiza la CPU: con las extensiones de hardware de los procesadores se crea una *physical CPU slice* que puede mapearse directamente en la CPU virtual, lo que permite una considerable mejora, y esto es lo que ocurre, por ejemplo, con el módulo de KVM, y es utilizado por QEMU cuando escoge como tipo de virtualización KVM. Si no se utiliza esta «aceleración» (por ejemplo, porque el procesador no dispone de las extensiones de hardware), QEMU utiliza TCG (Tiny Code Generator) para trasladar y ejecutar instrucciones de la CPU virtual como instrucciones de la CPU física (emulación), con la consiguiente reducción de prestaciones. En cuanto a KVM, utiliza QEMU como programa para crear una instancia de la máquina virtual, y una vez en ejecución esta será como un proceso regular del SO, al cual puede aplicársele el conjunto de comandos habituales, como *top*, *kill*, *taskset* y otras herramientas habituales para gestionar máquinas virtuales.

3) VirtualBox: es un hipervisor de código abierto y de tipo 2; muy flexible para arquitecturas x86 y AMD64/Intel64, tanto para uso empresarial como doméstico; se instala sobre Linux, Windows, Macintosh y Solaris, y es compatible con un gran número de SO *guest* (Windows, DOS, Linux, OpenSolaris, OS/2, OpenBSD, Android, ChromiunOS y otros más). VirtualBox puede complementarse 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 y pequeñas o medianas instalaciones y posibilita que las máquinas virtuales puedan convertirse a otros formatos (por ejemplo, vmdk de VmWare) y reutilizarse 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 o distribuciones ya instalados y listos para cargarse y ejecutarse; entre ellos pueden mencionarse OSBoxes, Virtualboxes y 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 la homogeneización de recursos o el acceso controlado a estos, con el fin de tener un entorno corporativo homogéneo y que puede controlarse y gestionarse desde el departamento de TIC de la empresa por herramientas como VMware VSphere u 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 e inversión en hardware, optimización…). ESXi dispone de una consola para su gestión y control y un entorno web como herramienta para la gestión y control del entorno virtual. Para la gestión avanzada de diferentes servidores ESXi, gestión en caliente de las máquinas virtuales, gestión avanzada del almacenamiento y demás opciones de monitorización y control, debe recurrirse a las herramientas propietarias de VMware, que están disponibles bajo licencia.

5) Microsoft Hyper-V: es un hipervisor para sistemas de 64 bits con procesadores que disponen de extensiones de hardware -VT-x y AMD-V- (no obstante, los programas de gestión pueden instalarse 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 y compatibilidad con Linux (Debian, Ubuntu, Centos/RH, Oracle, SuSE), FreeBSD y obviamente Windows (desde W10 e incluyendo servidores desde 2008). Existen algunas diferencias entre Hyper-V si se ejecuta, por ejemplo, en Windows 10 y sobre Windows Server, que se centran en la gestión de la memoria, la virtualización de GPU, la migración en caliente, las réplicas o la compartición de VHDX (discos virtuales),

pero por lo demás 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 WS2019, y permite que las máquinas creadas puedan exportarse a otros entornos Hyper-V o directamente a Azure (plataforma de *cloud* público de Microsoft).

6) Proxmox: con las facilidades de KVM y otras tecnologías, han surgido diferentes soluciones, como el Proxmox VE, que posibilita tener un entorno virtualizado con KVM y Linux Containers (LXC). Esta solución completa de virtualización de código abierto es adecuada para muchas empresas, ya que permite gestionar y configurar máquinas virtuales a partir de KVM, LXC, almacenamiento y redes virtualizadas y clústeres de alta disponibilidad. Se administra y monitoriza por medio de una interfaz web y admite como SO *guest* tanto Linux como Windows en máquinas virtuales o contenedores como virtualización del sistema Linux (a escala del SO). Esta distribución es un hipervisor tipo 1 (*bare-metal*) y está basada en la combinación de Debian, KVM y LXC; admite la 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...).

Un punto importante sobre los hipervisores, como ya se ha comentado anteriormente, es su evolución hacia la **virtualización del sistema operativo y los contenedores**. Desde hace varios años existe una disputa entre los hipervisores y los contenedores en relación con las prestaciones y otros parámetros de eficiencia, sobre todo en el *cloud* [Cvh14]. Además, en este sentido, Ubuntu ha apostado por un contenedor (LXD), aplicando la idea del contenedor al *cloud* y promocionándolo como The LXD Container hypervisor, con mayor densidad de ESX, un 25 % más de rapidez y sin latencia, indicando que pueden moverse las máquinas virtuales a los contenedores, fácilmente y sin modificar las aplicaciones o sus operaciones, y en el cual LXD es un hipervisor de contenedor puro que ejecuta sistemas operativos y aplicaciones Linux no modificados con operaciones al estilo de las máquinas virtuales a una velocidad y densidad increíbles.

Un ejemplo de esta tendencia (pero no 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 contenedor para cada servicio. Podría pensarse que hay muchos tipos diferentes de contenedores para probar, pero, según los expertos, en su gran mayoría todos tienen el mismo código en la parte inferior (desde Google, LXD y Docker con *cgroups* o *namespaces*, o *beancounters* en OpenVZ), y además han ido convergiendo y en la actualidad no hay prácticamente diferencias entre ellos.

Dos de los entornos de contenedores de más representativos son Docker y LXC/LXD: cuentan con ecosistemas que permiten obtener los contenedores como VA (*Virtual Appliances*), pero sin la carga que representa la máquina virtual y todo el sistema operativo *guest* y sus dependencias. Según Ubuntu, LXD

y Docker son complementarios, ya que Docker está centrado en las aplicaciones y LXD en *host machine containers* (actúa como una máquina virtual): ello permite ejecutar y gestionar contenedores Docker dentro de contenedores LXD, lo que hace a Docker aún mejor, ya que LXD, en relación con KVM, obtiene 10 veces más prestaciones, 14,5 veces más densidad, un 57 % menos de latencia y un 94 % más de rapidez en la carga. Es habitual que los desarrolladores de aplicaciones generen un contenedor 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 código abierto, puede empaquetarse, enviarse y ejecutarse 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 prácticamente ninguna intervención ni configuración, y se desplegará rápidamente en el *cloud* con una portabilidad total y sin invertir el tiempo que cuesta tener una máquina virtual correctamente configurada y enviarla o desplegarla. Según los expertos, esta 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, lo que permitirá 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 los contenedores de la propia comunidad, que pueden descargarse y ponerse en marcha en cuestión de minutos. Lo mismo podemos encontrar para LCC/LXD, y también contenedores creados por particulares y puestos a disposición de la comunidad: por ejemplo, Flockport, en el que se demuestra que LXD también puede centrarse en aplicaciones, las cuales no son un terreno totalmente propio de Docker (que no obstante continúa teniendo un papel protagonista).

4.2. IaaS (infraestructura como servicio)

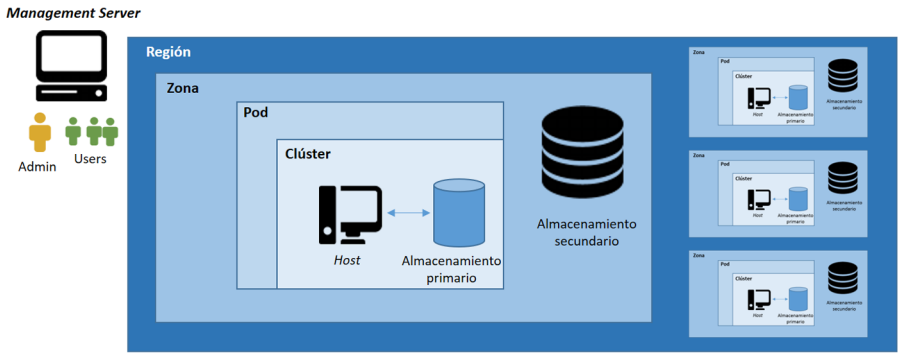
Dentro de las plataformas de código abierto que podemos encontrar con licencia Apache, GPL o similares, tenemos las siguientes:

1) Apache CloudStack: es una plataforma IaaS de código abierto (desde 2010 parcialmente bajo GPL y desde 2011 bajo ASL; véase su historia en el apartado 1), orientada a crear, manejar y aprovisionar infraestructura *cloud* (recursos de cómputo, *pools* de almacenamiento y redes) con diferentes supervisores (KVM, LXC, vSphere —vía vCenter—, XenServer, Xen Project, Hyper-V). Con ello permite inicializar un servicio elástico de *cloud clomputing* y que los usuarios finales obtengan los recursos; es capaz de manejar centenares de servidores físicos distribuidos geográficamente en centros de datos y escalando muy bien (casi linealmente) de servidor de administración, lo que evita la necesidad de servidores de gestión a nivel de clústeres. CloudStack configura automáticamente las redes y el almacenamiento para cada despliegue de máquinas virtuales, 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*; una interfaz web (que puede afinarse) para aprovisionar y administrar el *cloud*, así como una interfaz de usuario (también web) para ejecutar y gestionar las máquinas virtuales.

Esta plataforma provee una *REST-like API* para operar y administrar el *cloud* y es compatible con la EC2 API, lo que permite que herramientas de EC2 puedan utilizarse en ella. La instalación mínima consiste de una máquina que ejecuta CloudStack Management Server y otra máquina que actúa como infraestructura *cloud* (que no es más que una máquina ejecutando el hipervisor), aunque para pruebas de concepto puede montarse todo en una única máquina (*management server* y *KVM hypervisor host*). El *management server* es el único punto de configuración: puede ejecutarse en una máquina dedicada o en una máquina virtual, controla el despliegue de máquinas virtuales 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** (equivalentes a un único centro de datos con uno o más clústeres, o *pods*, y almacenamiento secundario),***pods*** (bastidores con conmutadores de capa 2 y uno o más clústeres), **clústeres** (uno o más *hosts* homogéneos y almacenamiento primario), ***host*** (único ordenador en un clúster, o 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 pueden clasificarse en dos escenarios concretos: **básica** (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) y **avanzada** (utiliza aislamiento en *layer-2*, o VLAN).

Como requerimiento, esta plataforma puede instalarse sobre CentOS/RHEL o Ubuntu. La siguiente figura muestra la arquitectura de esta plataforma.

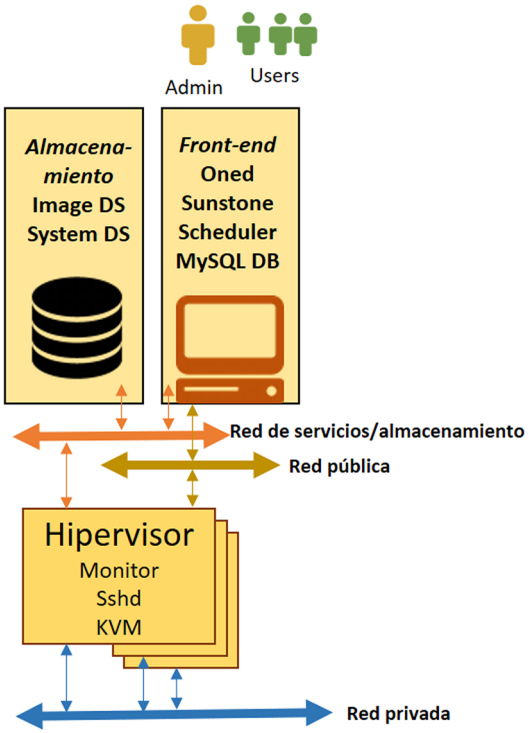


2) OpenNebula: plataforma de código abierto (ASL) para construir *cloud* IaaS tanto privado como público e híbrido, y que permite gestionar infraestructuras de centros de datos heterogéneas. La plataforma combina las tecnologías de gestión de almacenamiento, red, virtualización, monitorización y seguridad 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 interconectarse con diferentes interfaces (EC2 Query, OGF Open Cloud Computing Interface y vCloud) e interactuar con diferentes hipervisores (KVM y VMware), contenedores (Docker) y micro máquinas virtuales (Firecracker).

OpenNebula cuenta con un *marketplace*, que es un catálogo donde los usuarios y las organizaciones pueden distribuir y desplegar rápidamente *appliances ready-to-run* sobre *clouds* OpenNebula. La estabilidad de la plataforma y la calidad de 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 TIC, 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 permite gestionar y administrar todos los recursos para los diferentes niveles o roles desde un único punto. Los requisitos para instalar la plataforma se diferencian en instalaciones básicas o medias (decenas de hipervisores) o avanzadas (centenares de hipervisores). En relación con las redes, las de servicios entre el *front end* y el almacenamiento pueden ser compartidas (dado el bajo volumen de comunicaciones de los servicios), y las máquinas virtuales necesitarán una red pública y otra privada (para implementar VLAN aisladas). En cuanto a los sistemas operativos, pueden 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, puede utilizarse NFS/GlusterFS para las primeras (con imágenes en formato qcow2) y Ceph para las segundas; en cuanto a la autenticación, puede configurarse el sistema nativo de autenticación del que dispone OpenNebula, o Active Directory. La siguiente figura muestra la arquitectura de OpenNebula [Ona21].



OpenNebula utiliza KVM como hipervisor, pero, a causa de su flexibilidad e interoperabilidad, algunas organizaciones y empresas lo usan como *cloud management* en VMware vCenter, llamado vOneCloud, para proporcionar una capa de aprovisionamiento multitenencia por encima de VMware vCenter. Estos despliegues buscan características de provisión, elasticidad y multitenencia para el 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. OpenNebula ha desarrollado MiniONE: a partir de máquinas virtuales sobre KVM, permite poner en marcha una infraestructura de test sobre un nodo para evaluar y analizar sus características; es recomendable como primer paso antes de una instalación completa.

3) OpenQRM (*community edition*): es una plataforma de código abierto (GPL) de *cloud computing* para manejar infraestructura de centros de datos; que permite construir sobre ellos *clouds* privados, públicos e híbridos, y que gestiona la virtualización (mediante 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 tanto 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 máquinas virtuales

(*physical to virtual* [P2V], *virtual to physical* [V2P] y *virtual to virtual* [V2V]), y también es posible hacer una transición de un hipervisor a otros con la misma máquina virtual. La arquitectura de esta plataforma es extensible mediante conectores (*plugins*), que permiten insertar API o conectores hacia el *cloud* público (por ejemplo, AWS) u otros *clouds* (por ejemplo, OpenStack). Los requerimientos para hacer una prueba funcional son un servidor (para el servidor openQRM, virtualización y almacenamiento) con Intel/AMD de 64 bits, *dual/quad core*, extensiones VT-x/AMD-V y Linux (Debian, Ubuntu o CentOS) con acceso a internet. La última versión debe descargarse desde el sitio del desarrollador, pero las versiones anteriores deben descargarse directamente desde SourceForge, donde también puede accederse a la documentación de la comunidad (si bien está poco actualizada).

4) OpenStack: es una plataforma de código abierto (ASL) de *cloud computing*, que permite desplegar IaaS mediante un conjunto de componentes interrelacionados que controlan recursos de hardware (de diferentes proveedores), de cómputo, de red y de almacenamiento de un centro (o más) de datos. Los usuarios pueden gestionar y administrar el *cloud* utilizando un panel de control (*dashboard*) en la web o en CLI, o utilizando una API RESTful. OpenStack se inicia en 2010 como un proyecto conjunto de Rackspace Hosting y la NASA, y en 2014 pasó a ser gestionado por la OpenStack Foundation, con el objetivo de promover esta plataforma y su comunidad, de la cual hoy forman parte más de 500 empresas, organizaciones e instituciones.

La comunidad está organizada en ciclos de desarrollo y actualizaciones semestrales, y se reúne en sesiones de trabajo (multitudinarias) en la OpenStack Summit, para facilitar este desarrollo y generar planes de futuro. En la sesión de octubre de 2020 se reunieron (de forma virtual) 10.000 personas vinculadas a la plataforma; toda la información que se generó (vídeos, tutoriales, casos de uso, etc.) puede encontrarse en el siguiente enlace. Hoy se considera que la comunidad OpenStack está formada por más de 62.000 desarrolladores y usuarios y unas 640 compañías de 187 países, e incluye más de 20 millones de 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, y en 2011 Ubuntu adopta OpenStack para su estrategia de *cloud* (reemplazando a 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, y lo mismo con SuSE, y a partir de 2012 todas las grandes compañías de tecnología comienzan a incluirlo en sus estrategias y a hacerlo disponible para sus clientes, integrado o adaptado junto con sus productos (RHEL, Oracle, HP, etc.); hoy en día ha llegado a tener instalaciones muy particulares, como la del CERN, China Mobile o Walmart, entre otras.

La arquitectura de OpenStack es modular con diversos componentes, entre los que pueden enumerarse los siguientes [Osd]:

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

- ***Compute* (Zun)**: proporciona una API OpenStack para lanzar y administrar contenedores respaldados por diferentes tecnologías. A diferencia de Magnum (Container Orchestration Engine Provisioning), Zun es para usuarios que desean tratar los contenedores como un recurso administrado por OpenStack. Se supone que los contenedores administrados por Zun se integran bien con otros recursos de OpenStack, como la red (Neutron) y el *block storage* (Cinder). Los usuarios disponen de una API simplificada para administrar contenedores sin necesidad de explorar las complejidades de las diferentes tecnologías de contenedores.

- ***Networking* (Neutron)**: gestiona las redes y sus parámetros para que no generen cuellos de botella en una instalación, mediante el 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 anexar servicios de red adicionales, como los sistemas de detección de intrusos (IDS), equilibrio de carga, cortafuegos o redes privadas virtuales (VPN).

- ***Block storage* (Cinder)**: este módulo proporciona dispositivos de almacenamiento persistentes a escala de bloque que pueden ser utilizados por las instancias de Nova, y gestiona la creación, montaje y desmontaje de los dispositivos de bloque en los servidores, 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 plataformas de almacenamiento como Ceph, CloudByte, Coraid, EMC GlusterFS, Hitachi, IBM, LIO, NetApp, Nexenta, Scality, SolidFire y HP, entre otras. El almacenamiento de bloques es apropiado para escenarios donde el rendimiento es sensible, tales como base de datos y sistemas de archivos de crecimiento dinámico, o para permitir al servidor acceder al bloque «en bruto» (*raw*).

- ***Object storage* (Swift)**: es un sistema de almacenamiento redundante y escalable en el que los objetos y archivos se gestionan en diversas unida-

des de disco repartidas 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 de software, por lo cual puede utilizar discos y servidores de bajo costo.

- ***Identity* (Keystone)**: implementa un directorio de identidades para gestionar los servicios y los permisos de los que disponen los usuarios, y actúa además como un sistema de autenticación común en todo el *cloud* integrando servicios de directorio *back-end* existentes, como LDAP. Es compatible con múltiples formas de autenticación, incluyendo usuario y contraseñas, sistemas basados en *tokens* e inicios de sesión (al estilo AWS). También incluye una lista de todos los servicios existentes en el *cloud* OpenStack en un solo registro para que los usuarios y servicios de terceros puedan consultar qué recursos hay y con qué permisos pueden acceder.

- ***Image* (Glance)**: proporciona gestión sobre las imágenes de los discos y servidores (las cuales pueden utilizarse como una plantilla para nuevos despliegues). Entre sus funciones más importantes está la de almacenar y gestionar imágenes de discos y de servidores en una variedad de *back-ends* (incluyendo OpenStack 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.

- ***Dashboard* (Horizon)**: interfaz en el panel de control que permite a administradores y usuarios acceder a la provisión y la automatización de recursos basados en el *cloud* y gestionarlas. Su diseño es abierto y permite que puedan incluirse nuevas funcionalidades (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 la REST API nativa de OpenStack o la API *cloudformation-compatible query*.

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

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

- ***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 máquinas virtuales, utilizando PXE y IPMI para gestionar las máquinas de hardware.

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

- ***Shared file system* (Manila)**: provee una API para gestionar comparticiones de objetos (independientes del proveedor), y permite crear, borrar y denegar objetos y darles acceso 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* y es compatible con otras tecnologías, como PowerDNS y BIND. No provee el servicio como tal, sino que actúa como interfaz de DNS existentes para manejar zonas de DNS.

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

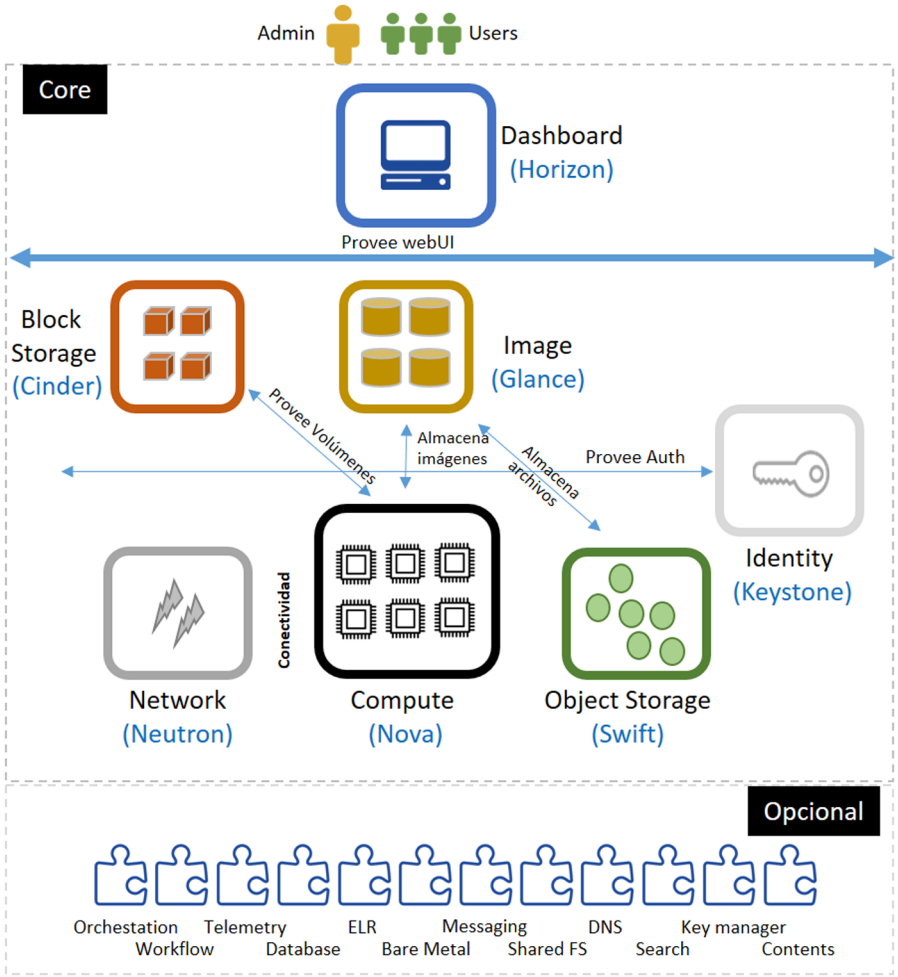
- ***Containers* (Magnum)**: Magnum hace que los motores de orquestación de contenedores, como Docker Swarm, Kubernetes y Apache Mesos, estén disponibles como recursos de OpenStack. Magnum usa Heat (Orchestration) para orquestar una imagen del sistema operativo que contiene Docker y Kubernetes y ejecuta esa imagen en máquinas virtuales o *bare metal* en una configuración de clúster.

Puede obtenerse una lista completa de todos los componentes y servicios de OpenStack en este enlace.

La última versión, de abril de 2021, es **Wallaby**, y en su página web pueden verse todos los componentes y servicios que incluye. Consultando el *marketplace*, pueden verse las distribuciones o *appliances* que lo admiten, los *clouds* públicos y privados que lo implementan, los proveedores de formación o las consultorías relacionadas con OpenStack o los controladores para dispositivos específicos.

Como puede observarse en el *marketplace*, existe una gran cantidad de *appliances* de OpenStack con diferentes objetivos y debe tenerse en cuenta que algunas de estas *appliances* son versiones de prueba por tiempo limitado.

La siguiente figura muestra algunos de los componentes de OpenStack y sus interrelaciones.



La forma más simple de probar OpenStack es mediante DevStack, que es una serie de *scripts* que se utilizan para desplegar rápidamente un entorno OpenStack completo basado en las últimas versiones desde el *git master*. Se utiliza de forma interactiva como entorno de desarrollo y como base para gran parte de las pruebas funcionales del proyecto OpenStack.

5) oVirt: es una plataforma de código abierto (ASL), creada por RH como proyecto de la comunidad y a partir de su distribución y tecnología, de gestión y provisión de entornos virtualizados (con KVM) y formada por dos componentes: oVirt Engine y oVirt Node. Permite, mediante una interfaz web muy sencilla, gestionar de forma centralizada máquinas virtuales y recursos de cómputo, red y almacenamiento de manera remota y sin acceder a los recursos físicos (es el equivalente en código abierto a VMware vSphere).

Sus requerimientos son muy básicos, pero está todo orientado a RH o CentOS, con procesadores con extensiones AMD-V y 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 máquinas virtuales.

El núcleo de oVirt Engine está escrito en Java; la interfaz en GWT Web Toolkit y se ejecuta sobre el servidor de aplicaciones WildFly (antes JBoss), y la arquitectura puede verse en el siguiente enlace. Los usuarios pueden ser gestionados internamente o conectados a un LDAP o AD. Para el almacenamiento, oVirt utiliza PostgreSQL y provee una API RESTful como interfaz a las funcionalidades de núcleo. oVirt Node es un servidor que ejecuta RHEL o CentOS con hipervisor KVM y un *daemon* VDSM (Virtual Desktop and Server Manager). Toda la gestión y administración se lleva a cabo mediante el portal web de oVirt Engine, que se conecta con VDSM para realizar las tareas indicadas. oVirt Engine puede instalarse en un nodo separado o sobre un nodo del clúster como una máquina virtual (*self-hosted engine*), la cual podrá instalarse manualmente o utilizar una *appliance* lista para ejecutarse.

4.3. PaaS (plataforma como servicio)

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, lo que les permite concentrarse únicamente en desarrollar su aplicación. Las ofertas de las plataformas incluyen componentes de muchos servicios utilizados en el desarrollo de software, como bases de datos, servidores web, sistemas operativos y almacenamiento: esto posibilita que en algunas de ellas trabajen múltiples usuarios, que colaboren y que compartan código y recursos.

Entre las más destacadas (o entre las mejor clasificadas en algunos indicadores, como satisfacción, cuota de mercado, soporte y servicio, por ejemplo, en G2 Crowd), están **Heroku, Amazon Lamda, ServiceNow Now Platform, OpenShift y Platform.sh**, entre otras.

Si se consideran las plataformas con un alto grado de satisfacción por parte de sus usuarios pero que todavía no han alcanzado una cuota elevada de mercado, puede mencionarse **Service Now**. Entre los productos que tienen presencia y recursos, pero valoraciones algo inferiores al primer grupo, están **Google App Engine**, **AWS Elastic Beanstalk y Plesk**. Entre las plataformas muy especializadas o por segmentos muy verticales del mercado, encontramos **Cloud Foundry** y **Dokku**.

Es interesante consultar listas de las PaaS, como por ejemplo la mencionada, ya que elaboran una valoración de su adecuación para diferentes tipos de empresas, y en un cuadrante donde se consideran el grado de satisfacción y las preferencias del mercado.

Existen diversas razones para seleccionar las plataformas PaaS, entre las que pueden mencionarse las siguientes:

- Lenguajes de programación: dado que será un proyecto de software, las plataformas escogidas serán aquellas que sean compatibles con el lenguaje en el cual se desarrollará el proyecto y estén dentro de sus requerimientos.

- 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 la adecuación y el despliegue de infraestructura y provisión del servicio, con recursos adicionales, y un departamento de TIC adecuado a los objetivos, pero tiene como contrapartidas el control, la seguridad y la privacidad.

- Tiempo de funcionamiento del servicio: existe una dependencia total, en el caso público, y el tiempo está controlado, en el caso privado; esto puede afectar seriamente a la planificación del proyecto.

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

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

- Integración: el proyecto deberá integrarse después con otros servicios, por lo cual debe analizarse el tiempo necesario para esta integración y de acuerdo con qué política, para evitar inconvenientes después de que se inicie el trabajo.

Entre las plataformas de código abierto que pueden instalarse sobre infraestructura propia (o *cloud*), podemos mencionar las siguientes (hay que tener en cuenta que, si bien todas comparten objetivos similares, la visión y la misión de cada una de ellas son diferentes y escapan al detalle de este apartado):

- **OKD**: OKD es una distribución de Kubernetes optimizada para el desarrollo continuo de aplicaciones y con una implementación multitenencia. OKD agrega herramientas sobre Kubernetes para permitir el desarrollo rápido de aplicaciones, la implementación y el escalado fácil, y el mantenimiento del ciclo de vida a largo plazo para equipos pequeños y grandes. OKD es una distribución de Kubernetes (hermana de Red Hat OpenShift) y lo amplía con seguridad y otros conceptos integrados. OKD también se conoce como Origen en Github y en la documentación. Si se busca sopor-

te empresarial o información, Red Hat también ofrece Red Hat OpenShift Container Platform.

- **Apache Stratos** (antes WS02): es un entorno PaaS extensible que permite ejecutar aplicaciones Apache Tomcat, PHP y MySQL, con la posibilidad de extenderse a otros servicios sobre las infraestructuras *cloud* más importantes. Para los desarrolladores, Stratos proporciona un entorno basado en el *cloud* para desarrollar, probar y ejecutar aplicaciones escalables, y los proveedores de TIC obtienen beneficio de las altas tasas de utilización, la gestión automatizada de recursos y la visión general de la plataforma, incluidas la supervisión y la facturación.

- **Cloudify** (Cloudify Community Version): es un entorno de código abierto 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, permitiendo que se desplieguen soluciones (manualmente o automáticamente) y manejando las tareas de mantenimiento.

- **Tsuru**: PaaS de código abierto, disponible en GitHub, que facilita el despliegue rápido y el control de aplicaciones en hardware propio. Los desarrolladores pueden utilizar Git, Tsuru Cli o el propio panel de control (*dashboard*) para desarrollar aplicaciones web en diferentes lenguajes, y pueden hacerse pruebas y test instalando las máquinas virtuales provistas para Vagrant.

- **Cloud Foundry**: plataforma PaaS de código abierto desarrollada originalmente por VMware, transferida a Pivotal Software (empresa conjunta entre EMC, VMware y General Electric) y ahora gestionada por la Cloud Foundry Foundation. Esta plataforma abarca todo el ciclo de vida de las aplicaciones, desde el desarrollo inicial y las 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 al cual tenga permisos para acceder.

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

- **AppScale**: plataforma de código abierto que automáticamente despliega y escala sobre *clouds* privados y públicos aplicaciones Google App Engine sin modificación. Si bien algunas de ellas **no** pueden calificarse como plataformas PaaS, es útil mencionar plataformas orientadas a desarrolladores (*cloud IDE*), pero debe recordarse que no todas ofrecen acceso gratuito a una serie de recursos:

- **Cloud9.io** (hoy forma parte de AWS): esta plataforma ofrece 40 lenguajes, control sobre la aplicación, espacio de trabajo (*workspaces*), públicos ilimitados gratuitos, depuración de aplicaciones en diferentes entornos (Django, Rails, WordPress…) en más de 300 combinaciones, entre otras características.

- **Red Hat CodeReady Workspaces** (antes **Codenvy**): se basa en el proyecto abierto Eclipse Che y utiliza Kubernetes y contenedores para proporcionar a cualquier desarrollador un entorno coherente, seguro y sin configuración. La experiencia es rápida y sencilla, como un entorno de desarrollo integrado en un ordenador portátil.

- **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. Un proyecto similar es Codebox.

- **Codiad**: plataforma IDE basada en la web con unos requerimientos muy reducidos, que permite disponer de un entorno de desarrollo en el *cloud* sin los potentes recursos que necesitan los entornos IDE habituales. Por ello, 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 72 lenguajes y entornos preconfigurados.

- **Ideone**: compilador y depurador en línea en un modelo de *write-&-run*, con soporte para más de 60 lenguajes y basado en tecnología Sphere Engine.

- **Eclipse Che**: IDE nativo de Kubernetes que permite que este sea accesible para los equipos de desarrolladores, proporcionando espacios de trabajo con un solo clic y eliminando la configuración del entorno local para todo su equipo. Che traslada la aplicación Kubernetes a su entorno de desarrollo y proporciona un IDE en el navegador, lo que permite codificar, construir, probar y ejecutar aplicaciones exactamente como se ejecutan en producción desde cualquier máquina.

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

4.4. SaaS (software como servicio)

Las plataformas SaaS son un modelo de distribución de software en el cual el soporte lógico y los datos que se manejan se alojan en servidores de una compañía de TIC (no necesariamente la proveedora del servicio), y el cliente accede a ellos a través de un software específico (software cliente, hoy en desuso) o a través de un servicio web (hoy en día es la opción más utilizada, ya que no debe instalarse nada en el equipo local y solo hay que disponer de un navegador).

La empresa proveedora se ocupa del mantenimiento, la operación diaria y el 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 están en los servidores del proveedor y el cliente no debe preocuparse por su disponibilidad, acceso, mantenimiento o seguridad, ya que todo ello recae sobre el proveedor en los términos del SLA firmado con el cliente.

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

- El cliente no necesita ni inversiones en infraestructura ni personal técnico especializado en el servicio, lo que reduce los costes y riesgos de inversión.
- La operación (disponibilidad, funcionalidad, seguridad…) recae en la empresa proveedora de acuerdo con un 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 se continúe pagando.
- No es necesario preocuparse por licencias de máquinas (las 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 ni las prestaciones, ya que todo se ejecutará en la máquina del proveedor.

Asimismo, pueden enumerarse un conjunto de **desventajas**, como las siguientes:

- No se tiene acceso a los datos (excepto si el servicio prevé un método de descarga).

- Si no se cuenta con mecanismos de cifrado y control, disminuyen el índice de privacidad, el control y la seguridad.

- El usuario accede a un servicio y no puede modificarlo o configurarlo más allá de las opciones que le permite el proveedor.

- Pueden producirse el *data lock-in* (en caso de cierre de actividades de la empresa proveedora) o el *vendor lock-in* (costes más altos que la competencia, ya que los de migración aún son mayores).

- Puede perderse el servicio o reducirse la calidad si la conexión es deficiente.

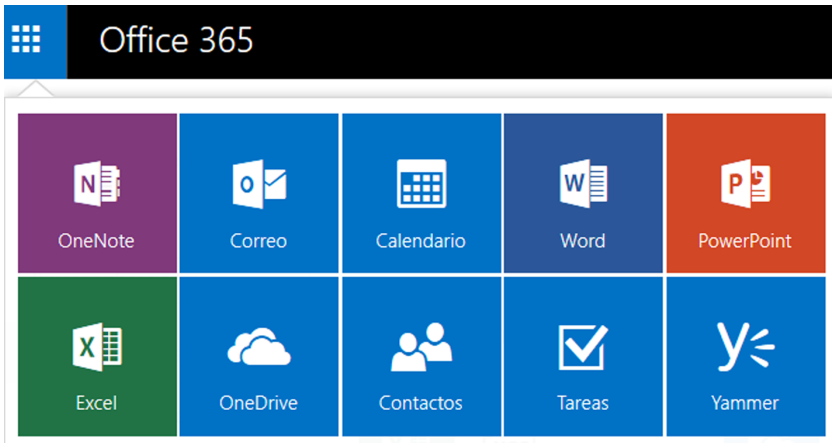
Mediante cuatro preguntas, podemos determinar si el servicio al cual estamos accediendo es SaaS:

- ¿Puede accederse al servicio a través de un navegador o de protocolos normalizados como REST?
- ¿Puede accederse a un modelo de pago como *pay-as-you-go*?
- ¿El software escala bajo demanda?
- ¿El proveedor asume la responsabilidad de la infraestructura, la gestión, el mantenimiento, la mejora y la supervisión del software, y deja al cliente solo utilizar el software?

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

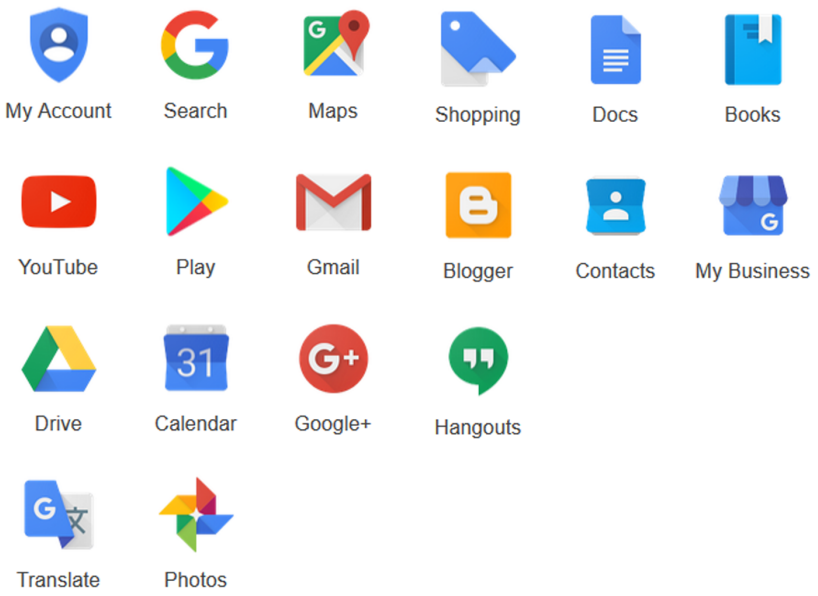
Google y Microsoft

Dos ejemplos que se muestran habitualmente sobre SaaS son dos de las grandes compañías TIC: 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 ofrecen dentro de la licencia SaaS (tanto para ordenador de escritorio como para dispositivo móvil), que incluye herramientas de edición de documentos (Office365), colaboración (SharePoint, Yammer), mensajes instantáneos (Lync), correo (Exchange) y almacenamiento (OneDrive), pero también CRM, comunicación (Skype) y BMI, entre los más destacados. Debe tenerse en cuenta que algunos de ellos son gratuitos (no confundirlos con el código abierto), como por ejemplo el correo y el almacenamiento (hasta un determinado volumen), y otros son con diferentes

modelos de licencia (por ejemplo, para los centros educativos, Office365 está incluido en la licencia del SO Windows).



Google también ofrece sus recursos, con diferentes modelos de negocio, aunque disponiendo de una versión de uso «gratuita».

Además de los que se muestran, de búsqueda, navegadores, mapas, correo, documentos, vídeos y fotos, almacenamiento, calendarios, comunicación y traducción, ofrece redes sociales, compras y servicios para las empresas. En el 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 de plataformas SaaS de éxito, podemos mencionar los siguientes (clicando en el enlace se accede a la página del servicio).

Categorías	Plataformas
Almacenamiento	Dropbox MediaFire Mega
Colaboración	Asana Todoist Trello Evernote
Documentos	Pixlr , Polar (edición de fotos) SiteBuilder (creación web) SlideRocket , Prezzi (creación de presentaciones) Slideshare (presentaciones y documentos) StackEdit (edición en la web)
Medios	Netflix Flickr (fotos) OutBrain (contenidos) Spotify (música) Wikipedia (enciclopedia)
Servicios	bit.ly (URL) Flood (test) Ionic (aplicaciones) Mailchimp (correo) Panda (antivirus)

Categorías	Plataformas
Social	BuzzStream Facebook Instagram LinkedIn LuckyOrange Twitter

Algunos autores consideran que el SaaS es la evolución de un modelo de negocio ampliamente conocido, llamado **ASP** (proveedor de servicio de aplicaciones), que proporciona servicios a los clientes a través de una red y permite el acceso a una aplicación de software en particular. Este modelo surgió a partir de los crecientes costos del software especializado y de la complejidad de su instalación y mantenimiento, que se reducen y se hacen accesibles cuando se funciona en modo ASP.

En principio, podemos afirmar que es un concepto similar al **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) y se basa en una relación «de tú a tú», mientras que SaaS es un modelo global, flexible y que en principio se basa en la ubicuidad y la 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 y el alojamiento en un solo paquete (aunque comienzan a haber desarrolladores que permiten utilizar las IaaS de las que disponga el cliente).

- La mayoría de aplicaciones ASP se ejecutan mediante modelos cliente-servidor, por lo cual se requiere un software específico sobre un SO determinado y se pierde la ubicuidad e independencia del SO (en SaaS solo se necesita un navegador); dado que es tecnología de un tercero, las actualizaciones y las 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, son instancias independientes y muy adaptadas o personalizadas para empresas diferentes, mientras que en SaaS, generalmente, se trata de múltiples clientes con una sola instancia (multitenencia).

- El ASP puede integrar diferentes servicios, como ofimática, directorio y CRM, en uno solo, en función de los acuerdos con los diferentes provee-

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

- Finalmente, en cuanto a la relación con el desarrollador o 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. Podría pensarse, por 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 las pymes.

En cuanto a los modelos de monetización del SaaS, son de diferentes tipos y bastante más difusos que en los 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…— que se han consumido en un periodo —por ejemplo, por hora—) o su equivalente *pay-what-you-want*. Por lo que respecta a los modelos *freemium* y prémium, el *freemium* representa un porcentaje muy alto de usuarios en relación con el prémium, pero tiene limitaciones en las prestaciones u opciones y es la fuente que alimenta el prémium. Los usuarios que realmente prueben y usen la aplicación acabarán convirtiéndose en prémium, pagarán por ello y esto sustentará el modelo de negocio, que el proveedor «cuidará», en el sentido de pertenencia a un grupo con privilegios. Los *freemium* tendrán incentivos para convertirse en prémium y, en determinados momentos, reducciones dinámicas de las prestaciones para favorecer este paso. Otros modelos de monetización son los siguientes:

- *Long tail*, o larga cola: catálogo de muchos productos de los que se venden pocas unidades (es el modelo Amazon).

- *Bait & hook*, o cebo y anzuelo: es similar al *freemium*, se da acceso a unos recursos, pero limitados, para incentivar el paso a pago (por ejemplo, *pay-as-you-go*).

- Suscripción, o *paywall*: distribución de los costos en una gran cantidad de usuarios y distribución del impacto inicial por una larga suscripción.

- Anuncios: método indirecto en el que la publicidad propia o de terceros monetiza el servicio.

- Microtransacciones: generalmente se aplica al ocio, se adquieren recursos o servicios con inversiones de valor muy bajo pero que en su conjunto representan un gran volumen.

- *Tiered service*: variante del *pay-as-you-go*, se incrementa el precio de manera progresiva a medida que se necesitan más recursos.

En relación con la estructura del SaaS, la mayoría se basa en una arquitectura de multitenencia, en la cual una única versión de la aplicación, con una única configuración (hardware, red, sistema operativo), se utiliza para todos los clientes, y para afrontar la escalabilidad, la aplicación se instala en varias máquinas a fin de permitir el escalado horizontal. No obstante, algunos SaaS no utilizan la multitenencia sino modelos más aislados, como los contenedores (modelo de Google) o las máquinas virtuales, ya que mejoran la privacidad y el funcionamiento y usan menos recursos. No obstante, es un tema de gran discusión en foros especializados y cada proveedor tiene su visión particular actualmente entre la multitenencia o los contenedores.

Es importante mencionar que una plataforma SaaS puede desplegarse sobre una plataforma IaaS o PaaS (por ejemplo, OpenStack, OpenNebula, Cloudify, OpenShift, CloudStack…), ya que significa proveer una máquina virtual o contenedor con la aplicación que ejecutará el usuario sobre este recurso. Sin embargo, existen plataformas específicas para construir infraestructuras SaaS, entre las cuales podemos mencionar:

- **Appserver.io**: si bien últimamente no se ha actualizado (https://github.com/appserver-io/appserver), es una plataforma interesante (*multithreaded application server for PHP*) para crear aplicaciones multitenencia basadas en PHP, tiene una funcionalidad estable y no presenta grandes inconvenientes.

- **Drupal**: CMS en SaaS (openSaaS).

- **Elgg**: motor de red social de código abierto para que las empresas e instituciones puedan crear redes sociales específicas con el objeto de fidelizar clientes o movilizar entornos sociales, en paralelo con una presencia en las redes sociales convencionales.

- **ERP5** (plataforma TioLive): servicio ERP en SaaS de código abierto, bien documentado y diseñado para pequeñas empresas.

- **Innomatic**: entorno de código abierto para construir plataformas SaaS de multitenencia, productos y aplicaciones empresariales en PHP. Con esta plataforma, las empresas pueden construir aplicaciones multitenencia preparadas para ejecutarse en el *cloud* y basadas en los entornos Composer y Symfony2.

- **Magento** (comercio electrónico en SaaS): es una plataforma de código abierto para el comercio electrónico escrita en PHP y que utiliza diferentes entornos PHP, como Laminas y Symfony. En 2018 Magento fue adquirida por Adobe, pero continúa teniendo una versión *open source*, si bien en la opinión de algunos expertos ha perdido la potencialidad de las versiones anteriores.

- **OwnCloud**: entorno para el alojamiento de archivos, que permite el almacenamiento en línea y las aplicaciones en línea (*cloud computing*). Puede combinarse con servicios de LibreOffice para editar archivos de ofimática simplemente disponiendo de un navegador.

- **Piwigo**: entorno de galería de fotos de código abierto totalmente reconfigurable y que incluye la posibilidad de gestionar también vídeos, aplicaciones web y juegos, entre otros.

- **WordPress**: plataforma SaaS de código abierto para publicar y gestionar contenido (incluye extensiones para el comercio electrónico), que dispone de un modelo de negocio también con licencia o *cloud*.

En modelo propietario o con licencia, pueden mencionarse Multihoster, SurPaaS, SaaS OpenBox, SaaS Maker (gratuita para desarrolladores) y Koros, entre otras.

4.5. Otros servicios *cloud*

Ya se ha mencionado anteriormente que existe una gran cantidad de XaaS. En este subapartado revisaremos dos de ellos, que tienen un alto impacto sobre los servicios anteriores y que pueden comercializarse por separado o unidos a los anteriores.

1) ***Storage as a service (StaaS)***: en este apartado han surgido diferentes tipos de negocios, como ya se ha comentado anteriormente, como por ejemplo Amazon S3, Dropbox y Drive (algunos autores los engloban todos en el *cloud storage*), que, teniendo como premisa el almacenamiento, presentan modelos de explotación diferentes: el primero ofrece los servicios de almacenamiento en internet por medio de una interfaz de servicio web (REST, SOAP y BitTorrent), mientras que el segundo y el tercero son explotados básicamente mediante SaaS.

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

- **GlusterFS**: utiliza el sistema de archivos en el espacio de usuario (FUSE) para generar un sistema de archivos virtual (VFS), y permite crear un sistema de archivos de red en un clúster en el 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 y distribuye los archivos a través de una colección de subvolúmenes; hace unidades mayores de la unión de espacios pequeños

y distribuidos permitiendo la replicación y, por lo tanto, la redundancia y la alta disponibilidad.

- **Ceph**: la base de este sistema de almacenamiento es la 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) y RADOS Gateway, así como el sistema de archivos Ceph. La RADOS Gateway proporciona interfaces a Amazon S3 y OpenStack, 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 usar fácilmente el sistema de almacenamiento de objetos de Ceph. Este sistema de archivo de altas prestaciones permite la lectura y la escritura parciales o completas, las instantáneas, las asignaciones de valor clave de nivel de objeto y las transacciones atómicas.

- **OpenStack**: esta arquitectura proporciona almacenamiento de objetos escalable y redundante utilizando clústeres de servidores, y puede escalar hasta petabytes de datos. Mediante este sistema de almacenamiento distribuido, ofrece a los usuarios de esta infraestructura escalabilidad y 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.

- **Sheepdog**: permite almacenar objetos distribuidos y se caracteriza por su pequeño tamaño, simplicidad y facilidad de uso. Este sistema gestiona volúmenes y servicios y maneja de forma inteligente los discos y nodos con un escalado óptimo que puede llegar a cientos o miles de dispositivos. Sheepdog puede anexarse a la máquina virtual de QEMU y los *targets* SCSI de Linux, tiene soporte para libvirt y OpenStack y puede interactuar con HTTP Simple Storage. A escala del sistema de archivos, permite hacer instantáneas, clonación y aprovisionamiento delgado, y puede anexarse a máquinas virtuales y SO que se ejecuten en *bare-metal* (con interfaz iSCSI)

2) ***Network as a service (NaaS)***: frecuentemente se asocia el término a otros servicios del *cloud* o también a *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 frecuentemente utiliza protocolos como OpenFlow. Entre el software de código abierto utilizado para proveer este servicio en el *cloud* podemos enumerar:

- **Floodlight**: es un controlador OpenFlow de altas prestaciones en Java y con licencia de Apache. Es un controlador SDN (*software defined networks*) abierto que funciona con dispositivos (*switches*) físicos y virtuales que in-

teractúan a través del protocolo OpenFlow, y puede escogerse el protocolo para interactuar con los restantes dispositivos remotos de red (*switches*, *routers*, *virtual switches* o *access points*). Al utilizar OpenFlow, permite explotar toda su funcionalidad: 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.

- **OpenStack Networking «Neutron»**: es una parte del proyecto OpenStack y proporciona una «red como un servicio» entre los NIC gestionados por Nova. Si bien es parte del núcleo de OpenStack, «Neutron» puede considerarse por su tamaño y funcionalidad como un producto NaaS, en el cual 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, y pueden añadirle servicios avanzados mediante el uso de conectores (*plugins*).

- **Open vSwitch**: es un *switch* virtual multicapa de código abierto (ASL), de altas prestaciones y funcionalidad extendida con una amplia gama de características que incluyen 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 fallas 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 el *kernel* que admite múltiples tecnologías de virtualización, como Xen/XenServer, KVM y VirtualBox.

Resumen

Como ha podido observarse en este módulo, el mundo *cloud* goza de una perspectiva alentadora de futuro y las tendencias (informe de RightScale) van hacia estrategias *multiclouds*, 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 categoría. En Eurostat pueden observarse las tendencias: cómo están usando las empresas esta tecnología y cómo pueden extraerse las tendencias del mercado en el futuro próximo.

También es necesario tener en cuenta las recomendaciones de la Comisión Europea, ya que, si bien tienen unos años, muchas de ellas todavía son válidas (Estrategia Cloud en Europa), y la financiación de esta tecnología en las propuestas en curso o en las que todavía están abiertas.

Actividades

1. Analiza dos hipervisores diferentes y extrae conclusiones de sus ventajas y desventajas.

2. Utiliza dos plataformas IaaS públicas y analiza sus prestaciones, ventajas y desventajas.

3. Repite la actividad 2 con dos plataformas PaaS y dos plataformas SaaS (a ser posible, dentro del mismo ámbito de negocio o funcionalidad).

4. Compara una plataforma PaaS orientada al desarrollo de aplicaciones y un *cloud* IDE, y extrae conclusiones sobre las ventajas de cada uno.

5. Elabora un cuadro comparativo sobre las diferentes opciones del *cloud*, con sus ventajas y desventajas, y propón un modelo de negocio basado en una de ellas, teniendo en cuenta los condicionantes, el modelo de monetización, la infraestructura, el nicho y todos los elementos que consideres necesarios y que pueden condicionar un negocio basado en esta tecnología.

Glosario

acuerdo de nivel de servicio *m* Acuerdo escrito entre un proveedor de servicio y su cliente con el objeto de fijar el nivel acordado para la calidad de dicho servicio.
en service level agreement
sigla **SLA**

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

appliance *m* Entorno que incluye todo el software preconfigurado y listo para utilizar, incluido el SO que puede ejecutarse sobre una máquina *bare-metal* o sobre un hipervisor.

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

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

cloud computing *m* Propuesta tecnológica que permite acceder a aplicaciones o servicios remotos de forma ubicua a través de un navegador: pueden ser aprovisionados o liberados bajo demanda y en un tiempo reducido. Sus equivalentes en español son *computación en la nube*, *servicios en la nube* e *informática en la nube*.

cómputo de altas prestaciones *m* Conjunto de recursos dedicados a ejecutar aplicaciones complejas con grandes necesidades de potencia de cómputo.
en high performance computing

contenedor *m* Técnica de virtualización del sistema operativo que permite altas prestaciones (ejemplos: Docker, LXC/LXD).
sin. jaula
en container, jail

data lock-in *m* Situación cuando 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.

gasto de funcionamiento *m* Costo de operación.
sigla **OPEX**

hipervisor *m* Capa de software que debe ponerse entre el SO de la máquina sobre la cual desean virtualizarse los recursos.

InfiniBand *f* Red de comunicación de altas prestaciones (por ejemplo, 40 Gbit).

infraestructura como servicio *f* Medio que está en la capa inferior y aprovisiona cómputo, almacenamiento y red como servicios estandarizados en la red.
sigla **IaaS**

inversión en bienes de capital *f* Costo de inversión
sigla **CAPEX**

JavaScript asíncrono y XML *m* Técnica de desarrollo web para crear aplicaciones interactivas o RIA.
en asynchronous JavaScript and XML
sigla **AJAX**

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

licencia de código abierto *f* Licencia que da soporte al código *open-source*. Algunos ejemplos son GPL (GNU General Public License), ASL (Apache Software License), CC (Creative Commons) y BSDL (Berkeley Software Distribution License).

multitenencia *f* 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.

en multitenency

OpenMP *f* Librería para la programación de aplicaciones de memoria compartida.

OpenMPI *f* Librería para la programación de aplicaciones distribuidas.

standby service *m* Modelo de monetización de recursos y servicios en el *cloud*. Otros modelos son *pay-as-they-grow*, *pay-per-user* y *pay-as-xx*.

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

servicio web *m* Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones desarrolladas en lenguajes de programación diferentes.
en web service

software como servicio *m* Aplicación completa ofrecida como un servicio, bajo demanda, accesible a través de un navegador y que el usuario no controla, eliminando la necesidad de instalar cualquier software y sin preocuparse por el mantenimiento de la aplicación.
sigla **SaaS**

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

standby service *m* Servicio preparado y listo para ponerse en ejecución y que estará disponible en un tiempo mínimo.

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

vendor lock-in *m* Dependencia de un proveedor de productos y servicios, que comporta 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 *f* 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 del hardware, el SO «ve» una máquina de hardware equivalente pero que es virtual.

Bibliografía

Todos los enlaces han sido visitados en julio de 2021.

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

[Bcs14] The Best Cloud Computing Services of 2020. <https://www.business.com/categories/cloud-computing-services/>

[Cca09] Cloud Application Architectures. George Reese. 2009. O'Reilly Media, Inc.

[Cca11] 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.

[Ccp13] Cloud Computing: Paradigms and Technologies. A. Shawish, M. Salama. Inter-cooperative Collective Intelligence: Techniques and Applications. 2013. Springer. 495. pp 39-67. <http://link.springer.com/chapter/10.1007%2F978-3-642-35016-0_2>

[Cis16] Magic Quadrant for Cloud Infrastructure as a Service. L. Leong, G. Petri, B. Gill, M. Dorosh. 2016 ID: G00278620. Gartner.

[Cmh15] Magic Quadrant for Cloud-Enabled Managed Hosting. D. Toombs, B. Gill, M. Dorosh. 2015. ID: G00269143. Gartner.

[Cvh14] 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>

[Dcc11] 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>

[Idd9] Info World Cloud Computing Deep Dive. 2009. <http://www.infoworld.com/resources/15675/cloud-security/download-the-cloud-security-deep-dive>

[Mcc13] 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>

[Ona21] OpenNebula. Open Cloud Reference Architecture. 2015. OpenNebulaSystems. <https://support.opennebula.pro/hc/en-us/articles/204210319-Open-Cloud-Reference-Architecture-White-Paper>

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

[Ovf15] Open Virtualization Format Specification. DSP0243. 2015. <https://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.1.pdf>

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

[Tch15] Timeline of Computer History. <http://www.computerhistory.org/timeline/2015/>

[Tif06] The Information Factories. George Gilder. 2006. Wired. <https://www.wired.com/2006/10/cloudware/>

[Vcc10] 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 No. 4, Pages 50-58. <https://doi.org/10.1145/1721654.1721672>

[Wtn08] 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/>

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