INSTITUTO LINUX

LINUX COLLEGE

05.01

# VIRTUALIZACIÓN EN REDES LINUX

# CONCEPTOS GENERALES DE VIRTUALIZACIÓN

## EXPERTO EN ADMINISTRACIÓN DE REDES LINUX

-. MÓDULO 05 · CAPÍTULO 01 .-

LINUX FOUNDATION AUTHORIZED TRAINING PARTNER
Linux Professional Institute
UTN FACULTAD REGIONAL DELTA
Universidad Nacional de San Luis
debian

# Capítulo 01 - CONCEPTOS GENERALES DE VIRTUALIZACIÓN

## ÍNDICE

# Capítulo 01 - CONCEPTOS GENERALES DE VIRTUALIZACIÓN

## ¿Qué es la virtualización?

Es una técnica que posibilita la ejecución de una o más máquinas (denominadas máquinas virtuales) sobre una única máquina física.

Cada máquina tiene asignada de forma independiente un conjunto de recursos de hardware y ejecuta su propia copia del OS. Software de virtualización. Este software se conoce de manera genérica como **Hypervisor**.

Este Hyperisor se encarga de la planificación de:

- Ejecución de las máquinas virtuales.
- Gestiona acceso compartido a los recursos de hardware.

El Hypervisor generará que los recursos reales de nuestro equipo físico se abstraigan y se utilicen en conveniencia.

## Historia

Contrario de lo que la mayoría piensa, el tema de la virtualización no es nuevo ni relativamente nuevo tampoco. Durante la década de los 60 los equipos de informática de muchas empresas y entidades tenían un problema similar: contaban con súper-computadoras o “mainframes” de alto rendimiento que deseaban “particionar lógicamente”, o utilizar para múltiples tareas simultaneas (lo que hoy conocemos como “multi-tasking”, trabajar más de una aplicación o proceso simultáneamente).

Es por esto que IBM desarrolló un método para crear múltiples “particiones lógicas” (similar a lo que conocemos hoy como “máquinas virtuales”), las cuales trabajaban independientemente una de las otras, y cada una utilizando los recursos provistos por el “mainframe”.

Cabe aclarar que podemos definir esto como los **inicios de la virtualización**; pero no es comparable con el crecimiento o como fue adoptado durante la década del 2000.

Ya para la década de los 80 y con la llegada de las relativamente económicas máquinas x86, comenzó una nueva era de micro computadoras, aplicaciones cliente-servidor, y “computación distribuida”; en donde los enormes y potentes “mainframes” con mil y una tareas y utilidades en una sola caja gigantesca, costosa y que ocupaba mucho espacio en la sala de servidores, se comenzaron a cambiar por relativamente pequeños servidores y computadoras personales de arquitectura x86 con “una caja diferente para cada uso”. Esto se convirtió rápidamente en el estándar de la industria.

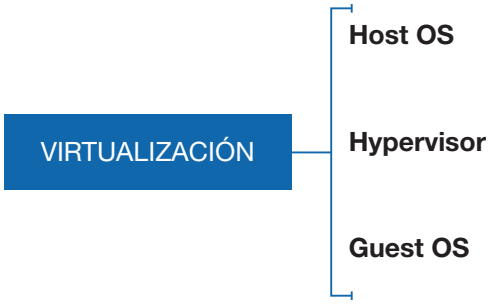
Entonces, en lugar de tener un solo mainframe que dividiera recursos para 3 tareas (por ejemplo) era mucho menos costoso tener 3 equipos que realizaran cada uno una tarea separada. Debido a esto, una vez más, el tema de la virtualización vuelve a quedar prácticamente en el olvido y no es hasta finales de la década de los 90 que gracias al alto desarrollo del hardware volvemos a caer en un predicamento similar al que estábamos en los años 60: el hardware existente es altamente eficiente, y utilizar cada “caja” para una sola aplicación sería un desperdicio de recursos, espacio, energía y dinero; y tampoco es conveniente asignarle múltiples usos o instalar varias

aplicaciones en un solo servidor convencional por más de una razón; pero podemos indicar una de las más importantes es que se vuelve cada vez más complejos de administrar.

Es por esto que vuelve a resurgir la idea de **dividir el hardware**, de manera tal que funcione como múltiples servidores independientes pero compartiendo los recursos de un mismo servidor físico. Y es de aquí que nace lo que hoy todos conocemos como "virtualización".

## Conceptos de virtualización

Para entender un poco más lo que es virtualización vamos a decir que, del Sistema Operativo al usuario se puede segmentar en **3 partes**:




- Host OS

Es el **Host donde reside el hardware junto con su sistema operativo** y proveerá los recursos a las máquinas virtuales y compartir los mismos.

- Guest OS

Es el **"sistema operativo" que está instalado dentro de la máquina virtual**. Esta puede ser distinta o no del sistema operativo del Host OS, o puede estar obligada a ser la misma que el sistema operativo de Host OS. Noten que utilicé comillas al momento de decir "sistema operativo"; ya que como veremos más adelante, no necesariamente para que la máquina virtual funcione es necesario un OS completo.

· Hypervisor

También se lo conoce como **Virtual Machine Manager** y es la **aplicación que permite utilizar distintas técnicas de virtualización para ejecutar al mismo tiempo diferentes sistemas operativos**. Nuevamente, estamos utilizando la palabra "sistema operativo" por ahora para facilitar la comprensión; pero como indiqué anteriormente no necesariamente vamos a encontrar todo un sistema Operativo instalado para correr una máquina virtual.
El concepto viene de principio de los 70's donde en el mainframe se consolidaban varias computadoras en una sola.

Dada la extensión de los sistemas basados en Unix, muchos fabricantes de hardware han estado vendiendo hardware ya virtualizado; pero es costoso. Por ejemplo los Blades Center de IBM.

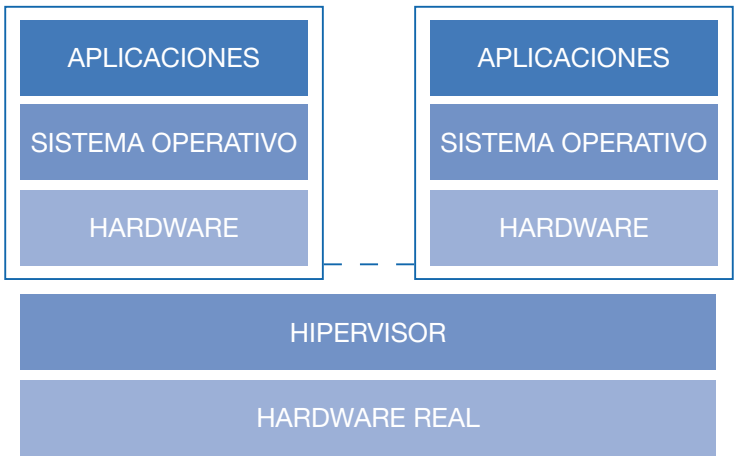
El primer hypervisor para PC fue la máquina virtualWare a fines de los 90's y ahora tanto intel como AMD están incorporando extensiones para virtualizar de mejor manera lo referente a arquitectura x86 para proporcionar un apoyo adicional a lo que es el hypervisor.

## Tipos de hipervisores

Los hipervisores pueden clasificarse en **dos tipos**:

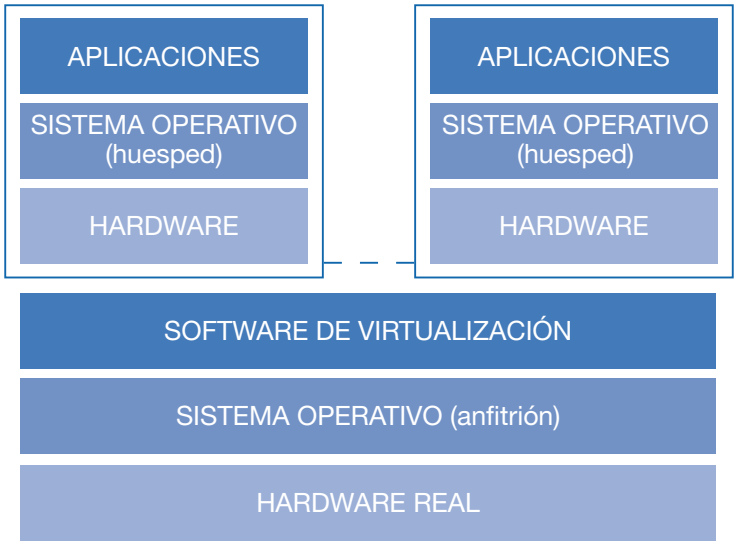
· Tipo 1: nativo, unhosted, bare metal. En este caso, el Hypervissor **se ejecuta directamente sobre el hardware para controlarlo y a su vez también controla el Guest OS**. Lo primero que se nota es la falta de OS. Como dijimos, el hypervisor se sienta directamente sobre el hardware y el sistema operativo del Host OS sobre este Hypervisor. De esta manera se reduce el overhead que genera que una máquina virtual deba transitar el sistema operativo para llegar al hardware. Podríamos decir que para el Hyervisor en este caso, el sistema operativo del Host OS lo toma como si fuera una máquina virtual más. Claro que desde aquí podremos controlar al resto de las máquinas virtuales a través del Hypervisor.

ESX, XEN, HYPERV.




· Tipo 2: también denominado **hosted**. El Hypervisor se ejecuta dentro del sistema operativo, por lo que el OS del Host OS se encargará de coordinar las llamadas para el uso de CPU, memoria, disco, red y otros recursos. Este tipo de virtualización hace que los usuarios finales corran una máquina virtual dentro de un dispositivo personal. Pero no confundan, no es que **solo** se utiliza dentro de los usuarios finales, sino que **los usuarios finales, si tienen que virtualizar, usarán este tipo de virtualización**.

Típico ejemplo de esto es Virtualbox.



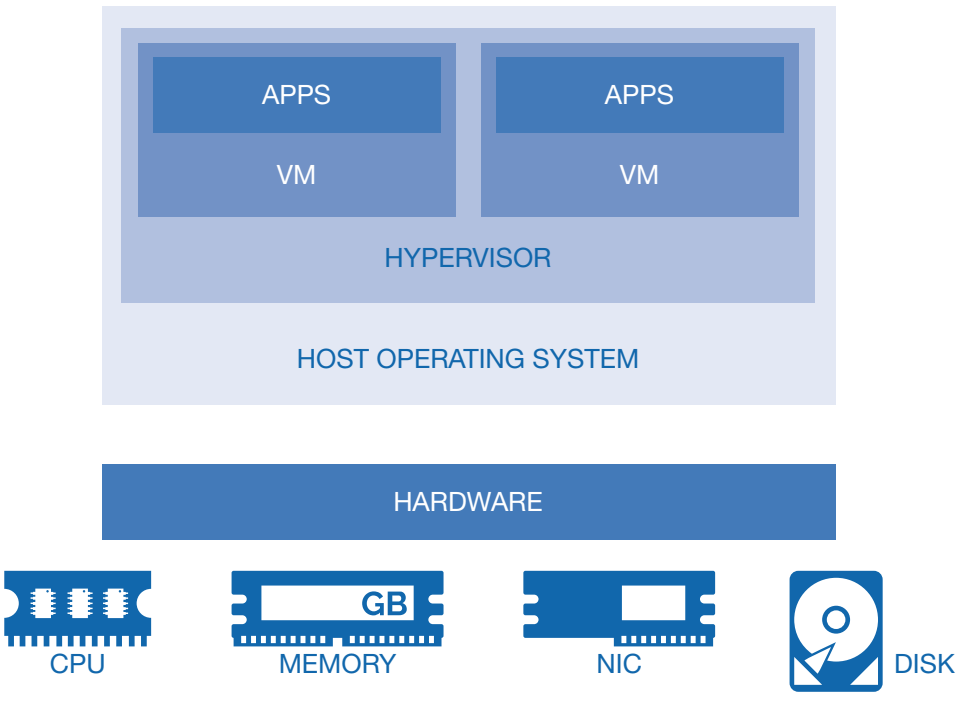
## Extensión de microprocesadores para virtualización

> Las extensiones de microprocesadores **permiten ejecutar una máquina virtual en el Guest OS sin incurrir en penalizaciones de emulación**; o sea, que al estar virtualizando un sistema operativo se reduzca el rendimiento.

No todos los procesadores cuentan con este tipo de extensiones. Para los procesadores Intel, la extensión se la conoce como **IVT** (**Intel Virtualization Technologies**) y se la referencia con el nombre de **Vanderpool**. Básicamente divide todo su poder en procesadores lógicos y se los asigna a las diferentes máquinas virtuales.

Por otro lado, para lo que es el mundo AMD, las extensiones se las conocen bajo el nombre de **Pacifica** y se publicó con el nombre de **Secure Virtual Machine** de ahí que más adelante buscaremos el svm dentro del **/proc/cpuinfo** para saber si es que contamos con la extensión para virtualización.

## Full Virtualization



Existen diferentes técnicas, o lo que es lo mismo, diferentes maneras de lograr la virtualización: **Full virtualization**, **Para virtualization**, **OS Virtualization**.

Tomemos el caso de Full Virtualization. En esta técnica, las máquinas virtuales emulan hardware real o ficticio que requiere de recursos reales desde el host. Este enfoque es utilizado por la mayoría de los sistemas de virtualización.

Permite ejecutar un sistema operativo invitado sin que éste sepa que no se ejecuta en hardware real.

Por lo que en este tipo de técnica de virtualización, requeriremos que cada feature de hardware sea reflejado en las máquinas virtuales. Como estamos emulando o reflejando el Hardware dentro del entorno de la máquina virtual, cualquier software (recuerden que el sistema operativo también es una aplicación) es capaz de ejecutarse.

Ejemplos de software de full Virtualization:

- VirtualBox
- Hyper-V
- QEMU
- ESXi
- Vmware Workstation

Entonces, tenemos que en Full Virtualization. El **Host OS emula suficiente hardware para permitir a los Guest OS correr separados unos de otros.**

También debemos agregar que los sistemas operativos Guest no sufren ninguna modificación, esto significa que **el proceso de instalación es el mismo que si instalaremos sobre un hardware real**. Es por eso que decimos también que el Guest OS no tiene conocimiento de que está siendo virtualizado, ya que si nos conectáramos a una Vm con Full Virtualization, no tenemos a nivel funcional forma de darnos cuenta que estamos corriendo en un entorno virtualizado.

Como puede observer en el gráfico, el Hypervisor (por ejemplo VirtualBox) corre dentro del sistema operativo, y es quien administra cada una de las máquinas virtuales. Cuando una máquina virtual necesita acceder a Hardware **se comunica con el hypervisor, y el hypervisor con el sistema operativo como si fuera una aplicación más**.

Pero, ¿qué pasa con las instrucciones privilegiadas? **Las instrucciones privilegiadas dentro de una máquina virtual son traducidas en instrucciones para el hypervisor**. Esta es otra de las razones principales por la que cualquier sistema operativo con arquitectura x86 puede ser virtualizado sin problemas.

## Paravirtualization

Requiere un hypervisor también; pero la mayor parte de su trabajo se realiza en el código del sistema operativo Guest, que es modificado para apoyarse sobre este hypervisor y evitar el uso innecesario de las instrucciones privilegiadas.

Entonces, una de las primeras diferencias que encontramos es que **el sistema operativo guest es modificado para poder apoyarse sobre este tipo de Hypervisor.**

También permite ejecutar diferentes sistemas operativos en un único server; pero los sistemas virtualizados tienen conocimiento que se están ejecutando dentro de un hipervisor, ya que estos son modificados. **El kernel del guestOS es modificado para correr en el hypervisor**.

Esto limita el soporte a sistemas operativos open source, que puede ser alterado o sistemas privativos donde los dueños concordaron en realizar estas modificaciones. Esto genera que el Guest OS se comunique directamente con el hypervisor, resultando una gran performance.

Como pueden notar en el gráfico **tanto las máquinas virtuales como el Host OS se apoyan en el Hypervisor, y es este quien se comunica directamente con el hardware**. Por lo que **el hypervisor no utiliza el sistema operativo del Host para comunicarse con el hardware**.

Entonces, la paravirtualización requiere que el sistema operativo Guest sea portado específicamente para utilizar la API del Hypervisor, por lo que no es el kernel original del Host OS en el que trabajaremos, sino que es uno modificado.

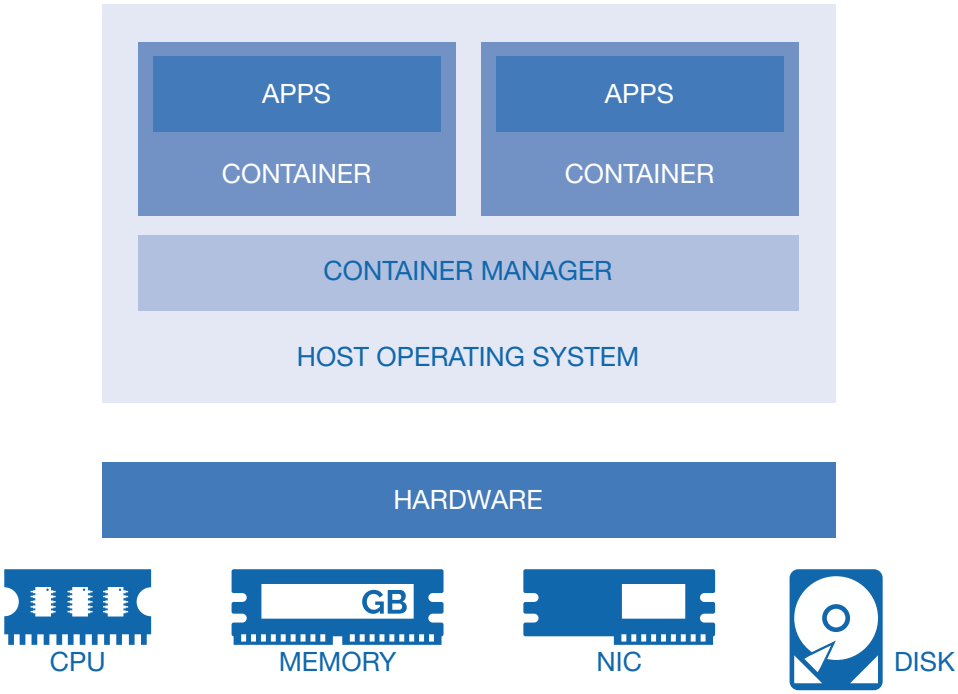
Este modelo modificado intenta reducir la porción del tiempo de ejecución utilizado para realizar operaciones en un entorno virtualizado que en uno no virtualizado; pero tiene como limitante que el Guest OS debe estar preparado para correr por encima del Hypervisor, y no por encima de un kernel como los que conocemos.

La reducción de tiempo de ejecución se logra porque elimina la necesidad de la máquina virtual de atrapar las instrucciones privilegiadas y traducirlas, ya que estas son reemplazadas por las **hypercalls**. **Estas hypercalls se comunican de manera directa con el hypervisor, por lo que no hay necesidad de traducir nada**.

Recuerden que el Guest OS fue modificado, por lo que puede realizar hypercalls y no system calls o llamadas al sistema.

Un ejemplo de ParaVirtualization es Xen.

## OS Virtualization




Proporciona la seguridad para ejecutar múltiples aplicaciones o copias del mismo OS (diferentes distros) en el mismo servidor.

En lugar de virtualizar un sistema en donde hay un OS completamente instalado, lo que hace es armar un container (o VE) de un solo OS.

En casos donde un solo sistema operativo es necesitado, los beneficios de estos VE es que no duplican funcionalidades y aumenta la performance.

Por lo que **en este tipo de Virtualizaciones el sistema operativo será particionado creando múltiples máquinas virtuales separadas unas de otras**. Pensemos esto como si fueran entornos que se ejecutan de manera separada.

Cuando nos referimos a que no duplican funcionalidades o que el sistema operativo es particionado implica que **todos los containers comparten un solo kernel**.

Para particionar una única máquina física en varios containers, el primer paso es crear un filesystem separado para cada una de ellas. La llamada al sistema **chroot()** setea el directorio raíz del proceso que lo está llamando, junto con el de sus hijos. Al modificar el directorio raíz, establecemos nuestro propio filesystem del cual no podremos mirar más afuera de él.

Esto también nos lleva a entender que habrá actividades que no pueden ser contenidas en el jail, como ser modificaciones de kernel accediendo de manera directa (/proc) o cargar módulos.

Como podemos observar, solo instancia por cada máquina virtual el filesystem y las librerías necesarias para ejecutarse; o por explicarlo simple, si hay algo que se pueda utilizar común a todos los containers, lo utilizamos.

Esto nos da mucha flexibilidad; pero más importante, impone casi nada de overhead ya que las aplicaciones en cada uno de estos jails utilizan las llamadas al sistema normales, y **no necesitan conversiones ni emulaciones de ningún tipo**. **Tampoco requiere soporte de hardware**.

Ejemplos de esto son:

- OpenVZ
- FreeBSD Jails
- Docker

## ¿Qué hay de lo que es cloud computing?

La computación en la nube o cloud computing es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.

Noten que hablamos de servicios de computación y no solamente de máquinas virtuales.

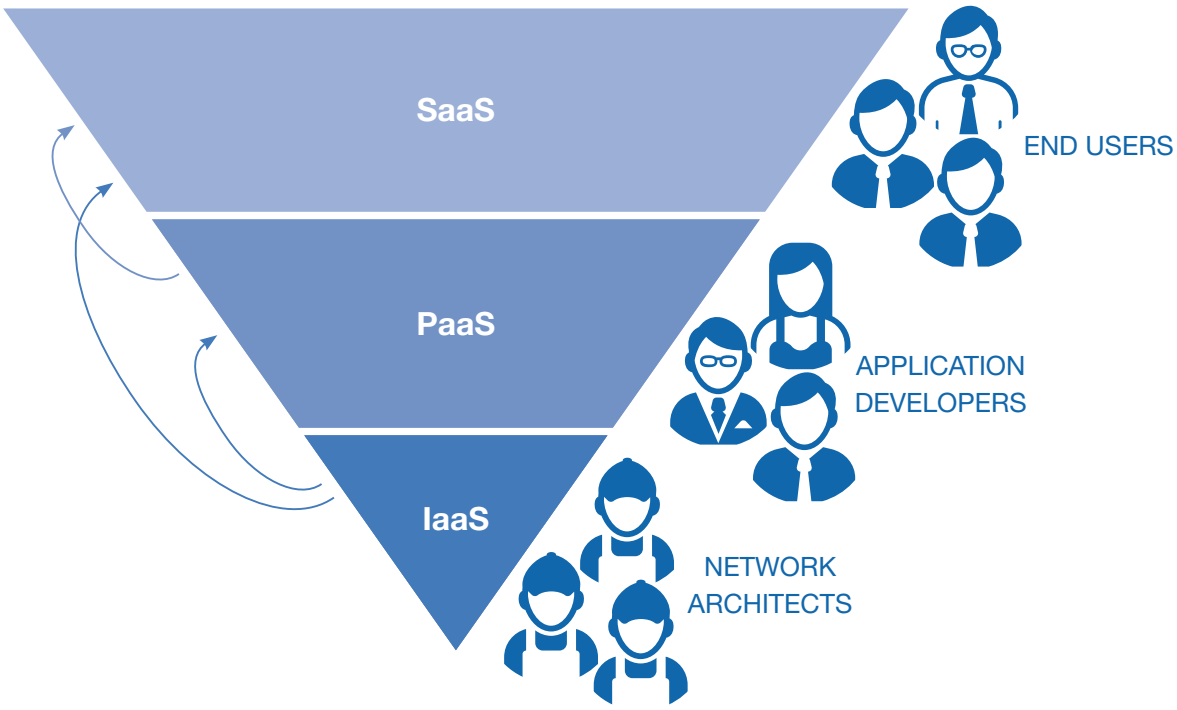
**El concepto principal es que todo lo que puede ofrecer un sistema informático se ofrece como servicio**, de modo que los usuarios puedan acceder a los servicios disponibles "en la nube de Internet" sin conocimientos (o, al menos sin ser expertos) en la gestión de los recursos que usan.

Cloud computing es un **nuevo modelo de prestación de servicios de negocio y tecnología**, que permite incluso al usuario acceder a un catálogo de servicios estandarizados ya que encontraremos que entre los diferentes proveedores de cloud computing, aunque con diferentes nombres, ofrecen los mismos servicios.

Esta segmentación en los servicios permite responder con lo necesario a las necesidades de su negocio de forma adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado. O sea, no es lo mismo una pequeña empresa que una multinacional.

## Modelos de servicio

Cada uno de los servicios que ofrecen las empresas de cloudcomputing caerá en alguno de estos:




### SOFTWARE COMO SERVICIO

El **software como servicio** (en inglés **software as a service, SaaS**) se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, por-demanda, vía multitenencia, que significa que una sola instancia del software que corre en la infraestructura del proveedor y sirve a múltiples organizaciones.

Las aplicaciones que suministran este modelo de servicio son accesibles a través de un navegador web, cliente o a veces hasta una API y el usuario no tiene control sobre ellas, aunque en algunos casos se le permite realizar algunas configuraciones. Esto le elimina la necesidad al cliente de instalar la aplicación en sus propios computadores, evitando asumir los costos de soporte y el mantenimiento de hardware y software. Software como servicio satisface la necesidad de usuarios finales que precisen de una aplicación sin mayores necesidades de configuración.

Ejemplos de esto son:

- Google Apps (sheet, docs)
- Office 365
- Cisco WebEX

## PLATAFORMA COMO SERVICIO

La capa del medio, que es la **plataforma como servicio** (en inglés **platform as a service, PaaS**), es la **encapsulación** de una abstracción de un ambiente de desarrollo y el empaquetamiento de una serie de módulos o complementos que proporcionan, normalmente, una funcionalidad horizontal como ser la persistencia de datos, autenticación, mails.

En este modelo de servicio al usuario se le ofrece la plataforma de desarrollo y las herramientas de programación por lo que puede desarrollar aplicaciones propias y controlar la aplicación, pero no controla la infraestructura. Generalmente son elemento de los que hacen uso las aplicaciones; pero no es donde corren las aplicaciones.

## INFRAESTRUCTURA COMO SERVICIO

La **infraestructura como servicio** (**infrastructure as a service, IaaS**) -también llamada en algunos casos **hardware as a service, HaaS**) se encuentra en la capa inferior y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red.

Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran (por ejemplo a través de la tecnología de virtualización) para manejar tipos específicos de cargas de trabajo, desde procesamiento en lotes (“batch”) hasta aumento de servidor/almacenamiento durante las cargas pico. El ejemplo comercial mejor conocido es Amazon Web Services, cuyos servicios EC2 y S3 ofrecen cómputo y servicios de almacenamiento esenciales (respectivamente). Otro ejemplo es Joyent, cuyo producto principal es una línea de servidores virtualizados, que proveen una infraestructura en demanda altamente escalable para manejar sitios web, incluidas aplicaciones web complejas escritas en Python, Ruby, PHP y Java.