ADMINISTRACIÓN DE INFRAESTRUCTURAS Y SISTEMAS INFORMÁTICOS (AISI)

Grado en Ingeniería Informática

Roberto R. Expósito (roberto.rey.exposito@udc.es)

TEMA 2

Tecnologías de virtualización

Contenidos

- Introducción
- Tipos de virtualización
- Virtualización de servidores
- Beneficios/inconvenientes
- Estándares

Contenidos

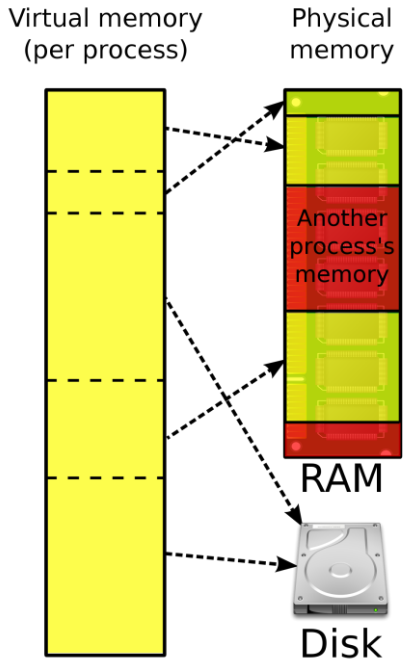
- **Introducción**
- Tipos de virtualización
- Virtualización de servidores
- Beneficios/inconvenientes
- Estándares

Definición de virtualización

- Abstracción lógica de una infraestructura física
  - La virtualización permite abstraer los recursos físicos reales de las aplicaciones o usuarios que los usan, proporcionando un entorno lógico o virtual que elimina la dependencia del sistema físico subyacente
    - El entorno virtual proporcionado puede ser diferente del sistema o recurso físico que abstrae
    - Se usa desde los comienzos de la computación
  - Algunos ejemplos de virtualización son:
    - Memoria virtual
    - Sistemas lógicos de ficheros
    - Gestión de volúmenes lógicos
    - VLAN
    - Virtualización de servidores
    - *Java Virtual Machine* (JVM)

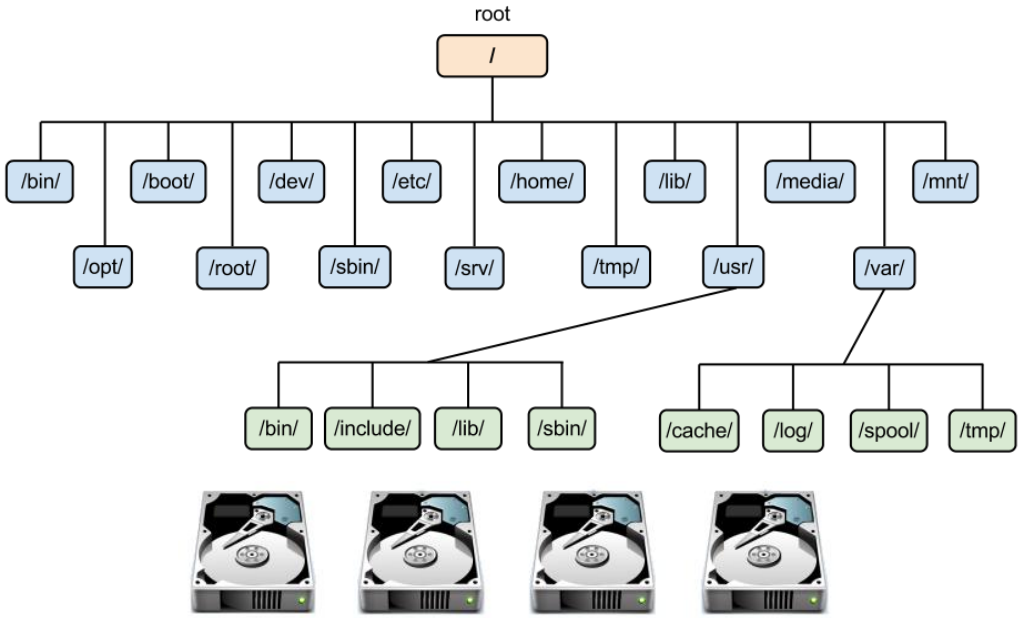
Ejemplos de virtualización

- Memoria Virtual
  - Técnica de gestión de la memoria que proporciona a las aplicaciones una **abstracción idealizada** de los recursos de almacenamiento
  - Combina la memoria RAM y espacio en disco (*swap*) para proporcionar "la ilusión" a las aplicaciones de disponer de un espacio de memoria contiguo más grande del que realmente hay
    - Cada proceso "ve" una memoria virtual mayor que la memoria RAM física disponible
    - Los procesos comparten la memoria física
    - Parte de la memoria virtual de los procesos se guarda en RAM y parte en un fichero *swap* en disco
    - El SO intercambia la información entre RAM y disco a medida que los procesos la van necesitando (se mantiene en RAM la información que se accede con mayor frecuencia)



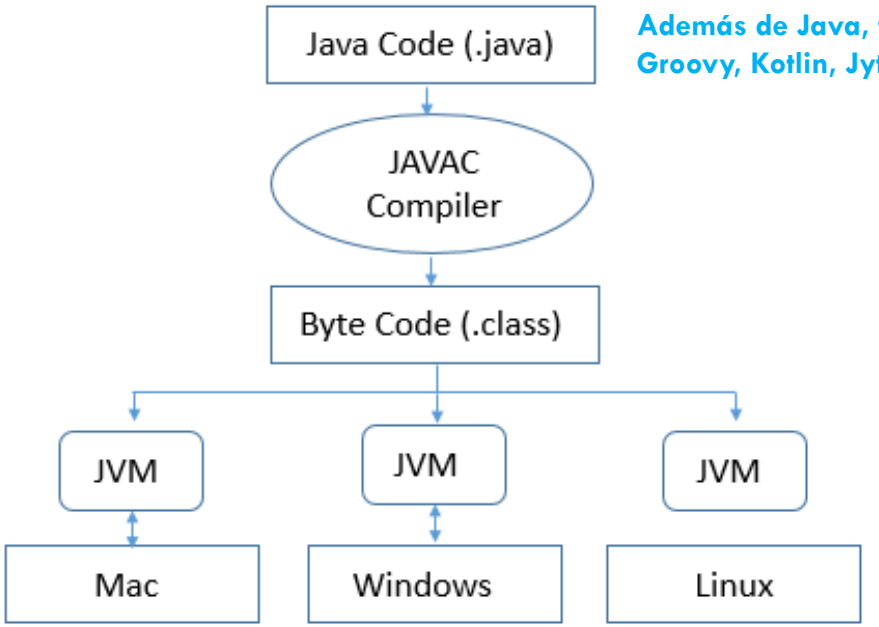
Ejemplos de virtualización

- Sistemas Lógicos de Ficheros
  - Proporcionan una visión abstracta de los datos almacenados en bloques en uno o más discos físicos
    - Ofrecen una estructura jerárquica en árbol en forma de directorios y ficheros
  - P.e. FAT, NTFS, EXT4, HFS+, XFS



Ejemplos de virtualización

- *JVM (Java Virtual Machine)*
  - Proporciona el nivel de abstracción necesario para permitir la portabilidad del código compilado de aplicaciones Java (*bytecode*)
    - El *bytecode* puede ejecutarse en cualquier combinación de SO y arquitectura HW para la que exista una implementación del entorno de ejecución de Java: *Java Runtime Environment* (JRE)



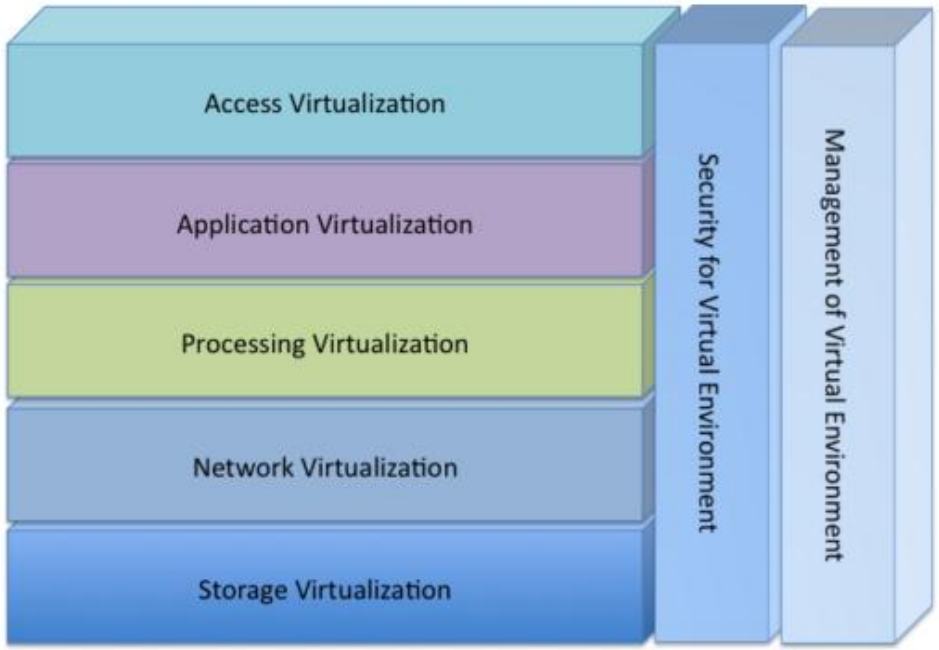
Además de Java, también aplicaciones Scala, Groovy, Kotlin, Jython, entre otros

Contenidos

- Introducción
- **Tipos de virtualización**
- Virtualización de servidores
- Beneficios/inconvenientes
- Estándares

Tipos de virtualización

- Acceso
- Aplicación
- Servidor
- Red
- Almacenamiento

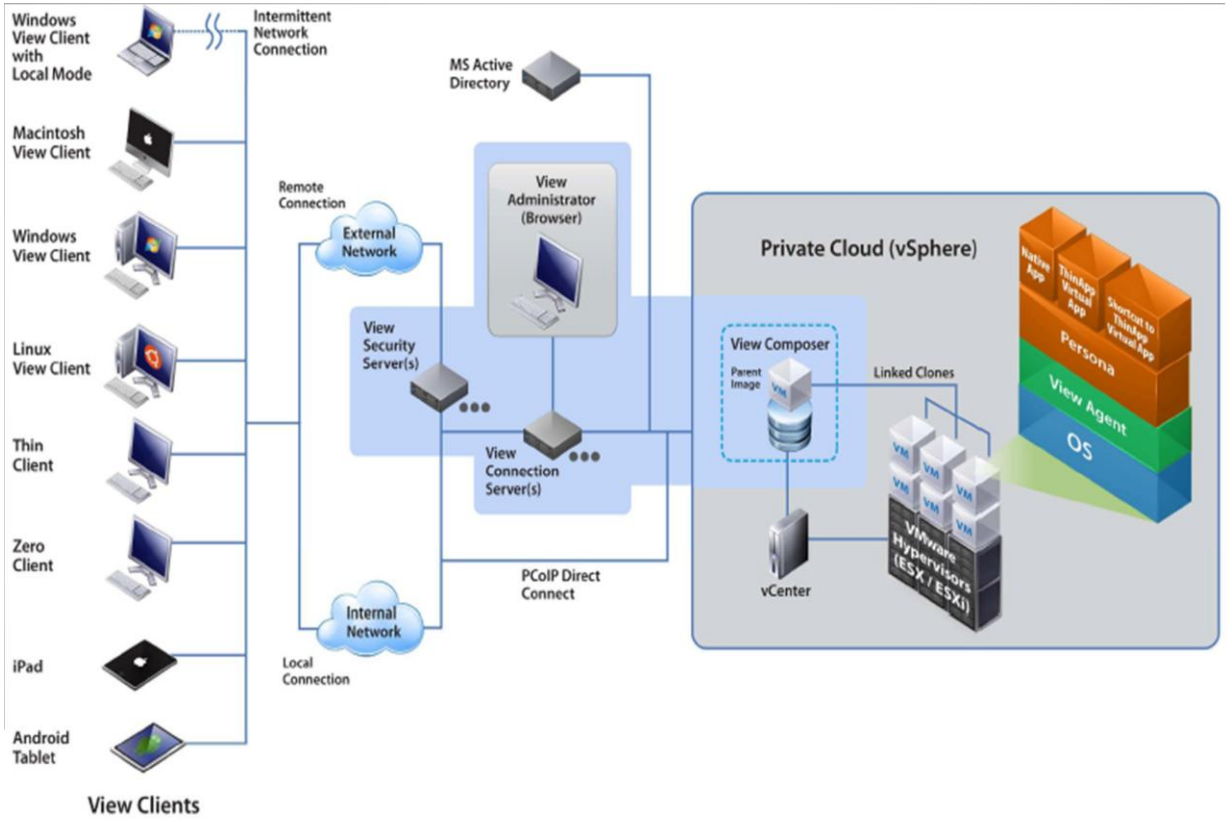


Virtualización de acceso

- En este tipo de virtualización se visualiza la interfaz del entorno virtualizado desde un equipo remoto
- Dentro de esta categoría podemos incluir
  - Protocolos de escritorio remoto
    - Visualización remota de la interfaz de una aplicación o equipo
    - Ejemplos: X-Windows, VNC (*Virtual Network Computing*), RDP (*Remote Desktop Protocol*)
  - Virtualización de escritorios (VDI, *Virtualization Desktop Infrastructure*)
    - Gestión centralizada de escritorios virtuales personalizados
      - Usado por las organizaciones que quieren simplificar la gestión del *software* de escritorio de sus usuarios
      - El *software* que virtualiza el acceso se ejecuta en los servidores
      - El acceso se hace desde clientes ligeros (*Netbooks, tablets, smartphones*...)
    - Ejemplos: Citrix XenDesktop, VMware Horizon View, EyeOS

Virtualización de acceso: VDI

- Ejemplo: VDI na UDC
  - Utiliza VMware Horizon View: https://wan.vdi.udc.es

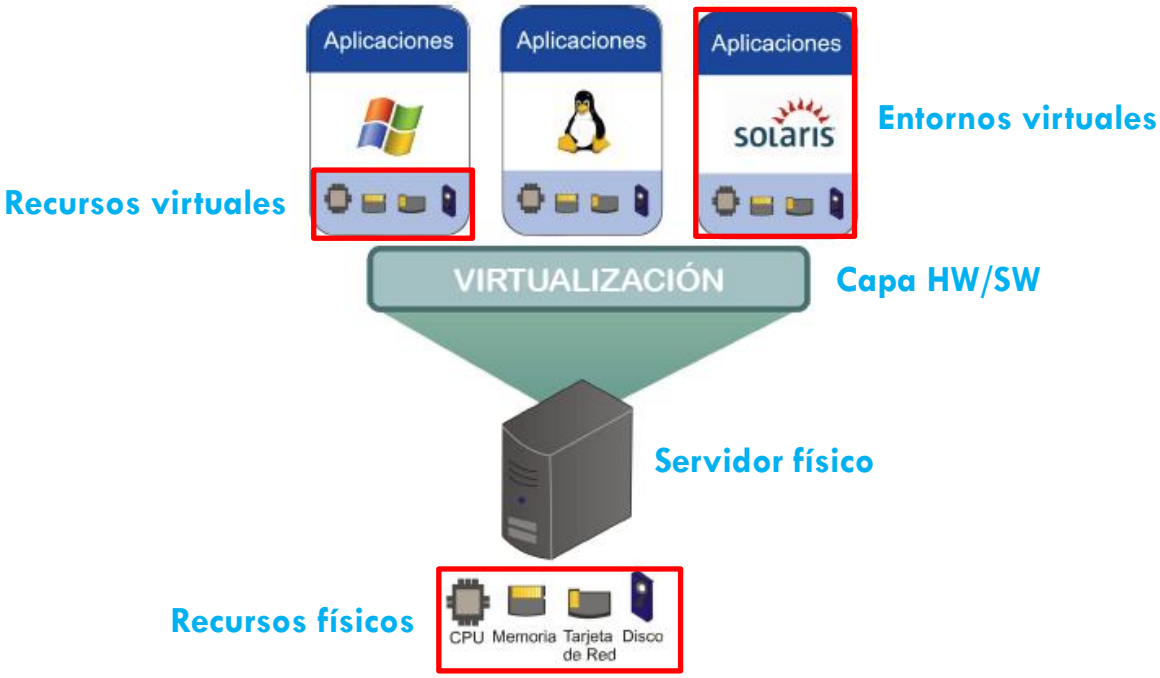


Virtualización de servidores

- Permite compartir los recursos físicos de un servidor entre múltiples entornos o ambientes virtuales de ejecución **aislados** entre sí
  - Se implementa como una capa intermedia entre el servidor físico y los entornos virtuales usando técnicas SW, HW o una combinación de ambas
  - Estos entornos virtuales reciben diferentes nombres dependiendo de la tecnología o capa de virtualización que se utilice
    - Máquina Virtual (VM), contenedor (container), zonas…

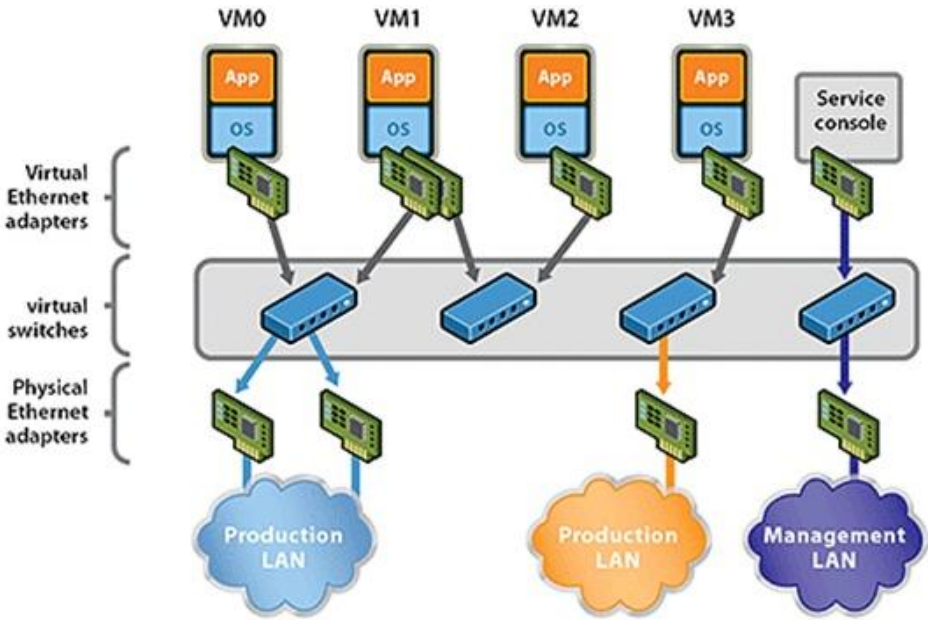
Virtualización de servidores

- Permite compartir los recursos físicos de un servidor entre múltiples entornos o ambientes virtuales de ejecución **aislados** entre sí
  - Cada entorno virtual tiene asignados sus propios recursos virtuales y un SO, el cual puede ser diferente al del servidor físico y al de otros entornos
  - Tanto el SO como las aplicaciones están aisladas en cada entorno y no tienen acceso a los recursos y a los datos de otros entornos



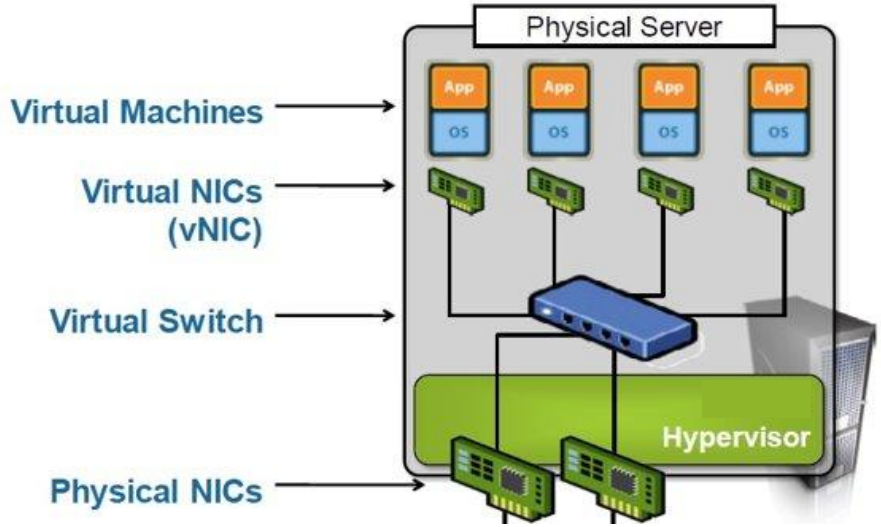
Virtualización de red

- Abstracción de los recursos físicos de red en recursos virtuales
- Se aplica en diferentes dispositivos y niveles del modelo OSI
  - En un servidor para crear redes virtuales entre VM (vNIC y *vSwitches*)
  - En *switches* y *routers* para crear particiones virtuales
    - VLAN
    - VRF (*Virtual Routing and Forwarding*)
  - Para crear versiones virtuales de dispositivos y funciones de red
    - vSwitches
      - Open vSwitch
      - VMware vSwitch
    - Balanceador de carga
      - KEMP Virtual LoadMaster
    - *Firewalls*
      - Cisco ASAv
  - Para agregar recursos
    - *Switches* distribuidos
      - VMware vNetwork
      - Cisco Nexus 1000V



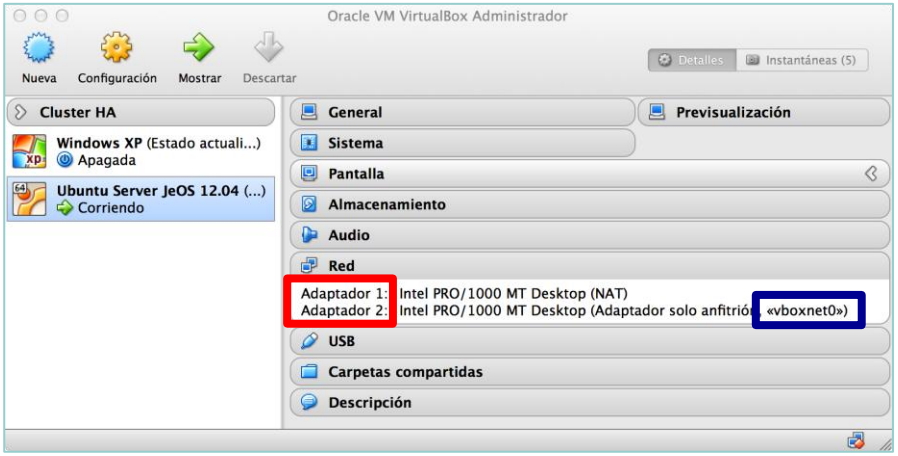
Virtualización de red

- Ejemplo: *Switch* virtual (vSwitch o *virtual bridge*)
  - Versión virtual de un *switch* físico que se ejecuta en el servidor físico bajo el control del hipervisor
    - Las VM se conectan al vSwitch mediante una o más NIC virtuales (vNIC)
    - Los *switches* virtuales pueden estar conectados a NIC físicas o servir para crear redes internas al servidor entre las VM

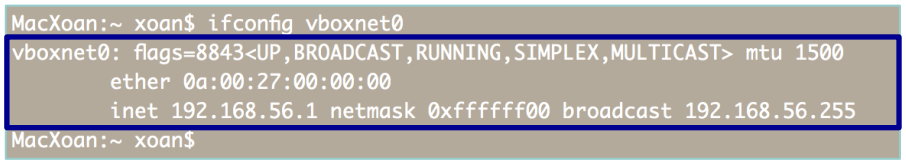


Virtualización de red

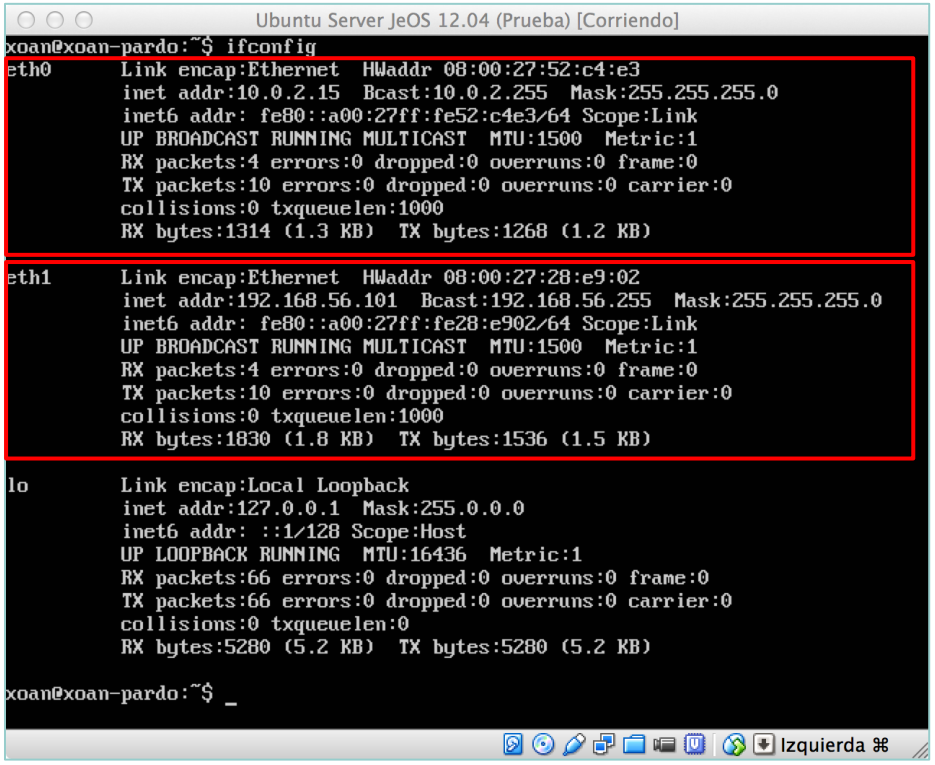
- Ejemplo: red virtual en VirtualBox



NICs virtuales en VirtualBox



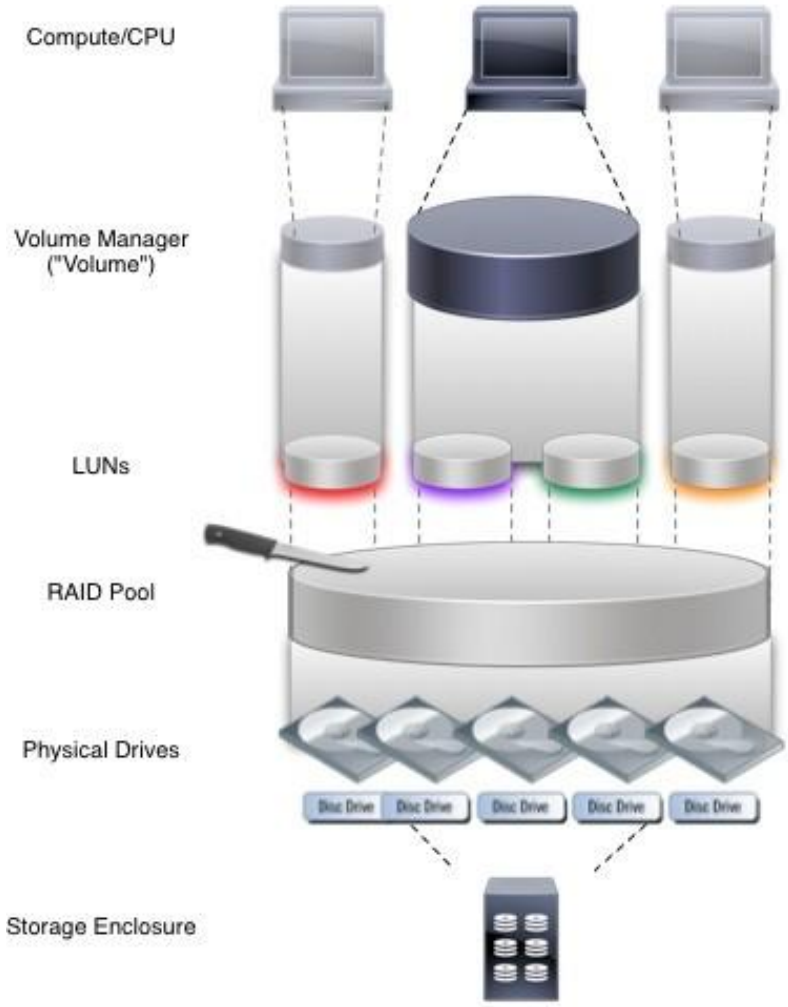
Switch virtual en el servidor físico



NIC virtuales (vNIC) en la VM

Virtualización de almacenamiento

- Capa/s de abstracción entre el almacenamiento físico y los servidores
- Permiten abstraer, emular, agregar y asignar recursos dinámicamente
- Los discos físicos se combinan mediante esquemas RAID para formar *pools* de almacenamiento
- Las unidades lógicas (LUN) son particiones del espacio de almacenamiento disponible
- Los volúmenes que se montan en los servidores se forman combinando 1 o más LUN
- El contenido de los volúmenes se organiza usando un sistema lógico de ficheros

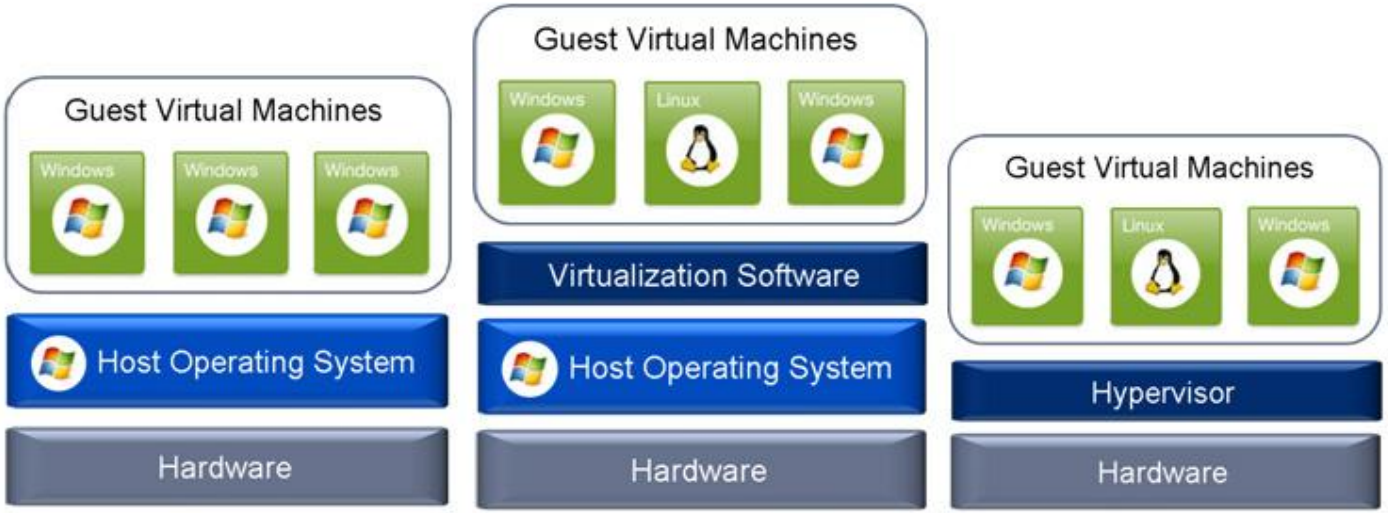


Contenidos

- Introducción
- Tipos de virtualización
- **Virtualización de servidores**
- Beneficios/inconvenientes
- Estándares

Virtualización de servidores

- Existen diferentes técnicas de virtualización de servidores
  - Emuladores de *hardware*
  - Virtualización basada en hipervisor
  - Virtualización a nivel de *kernel*
  - Virtualización por compartición de *kernel*



Emulación de *hardware*

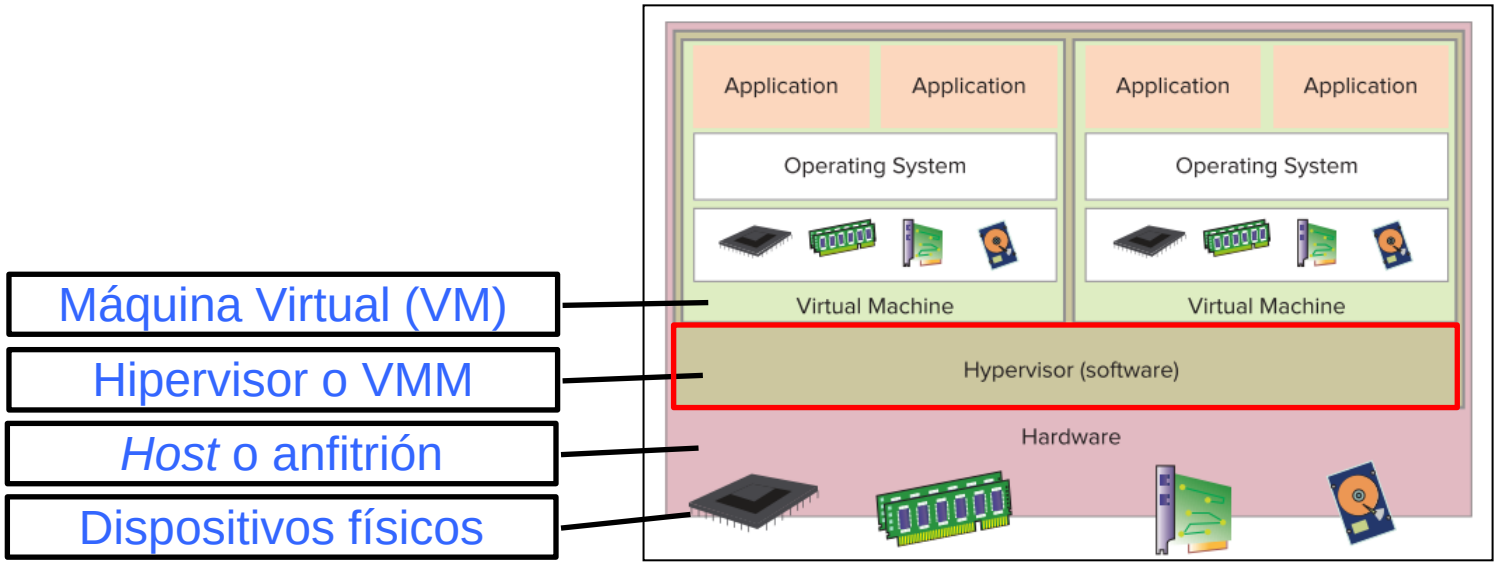
- Permiten emular un sistema en otro diferente
  - También llamada virtualización a nivel de ISA (*Instruction Set Architecture*)
  - Traducción del **ISA virtual al ISA real** de la máquina física
- Tiene dos aplicaciones principales:
  - Ejecutar programas/SO compilados para una arquitectura en otra diferente
    - Jugar a un juego de la PlayStation 3 (CPU Cell) en un PC con CPU x86
  - Emular la arquitectura HW **completa** de un sistema (E/S incluida)
    - Si ejecutamos un SO en el sistema emulado, obtenemos una VM
    - La VM no sabe que se ejecuta sobre HW emulado

Emulación de *Hardware*

- Técnicas de emulación
  - Interpretación de código (*fetch-decode-execute loop*)
  - Traducción binaria (estática/dinámica)
- Su principal desventaja es la sobrecarga que introducen
  - Emular componentes HW mediante SW es lento
- En la actualidad se mejoró mucho su rendimiento
  - Sacan provecho del soporte HW para la virtualización de los procesadores
  - Usan *drivers* de E/S "especiales" (paravirtualizados) que se instalan en el SO de la VM y que permiten la comunicación directa de ésta con los dispositivos de E/S del servidor físico
  - Gracias a esto, las operaciones que antes eran emuladas por SW ahora se aceleran ejecutándolas directamente en el HW del servidor o de las interfaces de E/S
- Ejemplos:
  - Bochs (x86), QEMU (x86, PowerPC, SPARC, MIPS, ARM), Rosseta (x86)

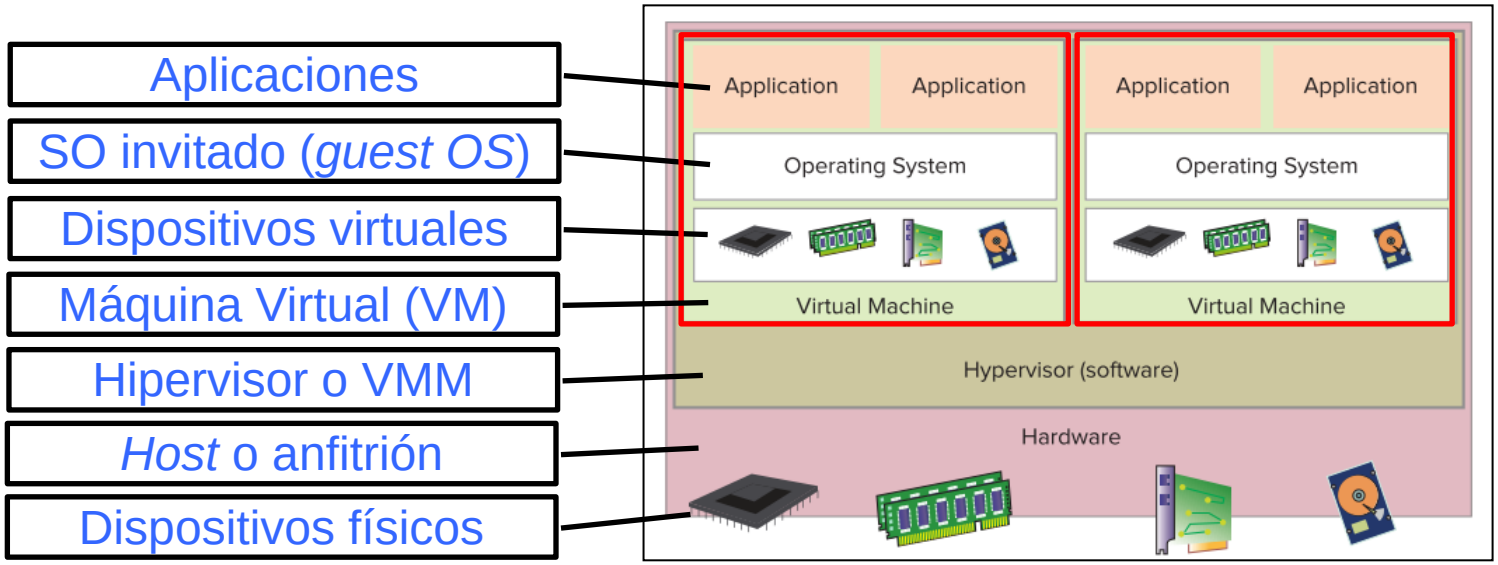
Virtualización basada en hipervisor

- Un **hipervisor** o **Virtual Machine Monitor (VMM)** es una **capa SW** que virtualiza el HW de un servidor físico (*host* o anfitrión) y gestiona las VM que se ejecutan en él
- Permite ejecutar VM con **SO diferentes** compartiendo el HW de un mismo servidor físico



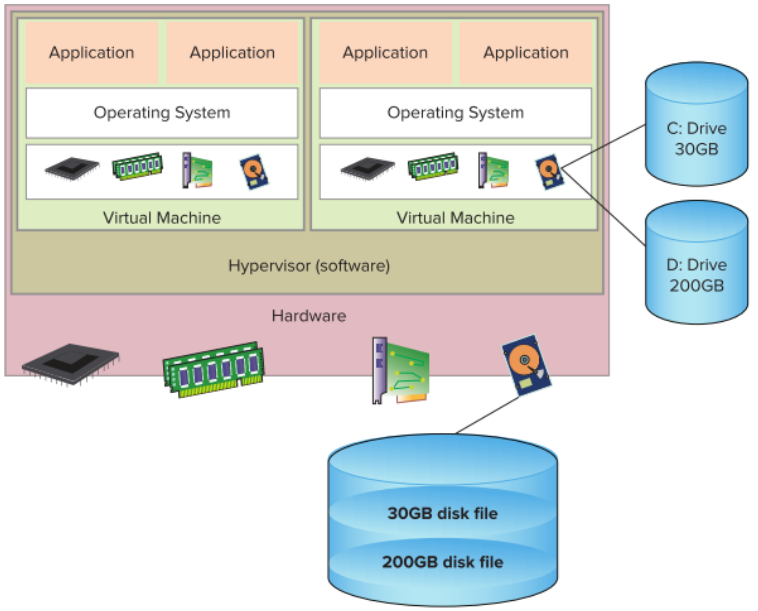
Virtualización basada en hipervisor

- Una **Máquina Virtual (VM)** es una abstracción de los recursos HW de un servidor físico, del SO y aplicaciones que se ejecutan en él
  - Las VM ejecutadas en un servidor físico están aisladas entre sí
  - Los recursos virtuales que una VM "ve" (y puede usar) pueden ser diferentes a los disponibles en el servidor y a los que otras VM "ven"



Virtualización basada en hipervisor

- Funciones del hipervisor
  - Abstrae los recursos dando la ilusión a las VM de que acceden directamente y en exclusiva a los mismos
  - Se encarga de repartir los recursos existentes entre las diferentes VM
    - Cores físicos, memoria RAM, espacio en disco, NIC, …
  - Controla el acceso a los recursos balanceando la carga de trabajo, actuando a modo de intermediario entre estos y las VM
  - Permite crear y gestionar las VM y proporciona seguridad y aislamiento entre ellas

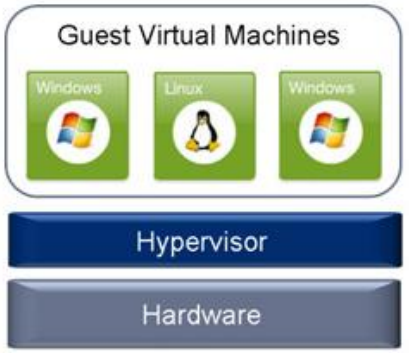


Virtualización basada en hipervisor

- Propiedades de un hipervisor
  - Los principios formales de la virtualización basada en hipervisor fueron propuestos por **Gerald J. Popek** y **Robert P. Goldberg** en su artículo publicado en 1974:
    - "*Formal requirements for virtualizable third generation architectures*"
    - *https://dl.acm.org/doi/10.1145/361011.361073*
  - Definieron las tres propiedades que debe cumplir un hipervisor
    1. **Equivalencia.** Proporcionar un entorno idéntico/equivalente al del HW físico que se virtualiza (procesador, memoria, almacenamiento, red, …)
       - La ejecución de un binario en una VM debe producir los mismos resultados que su ejecución en un entorno no virtualizado (es decir, directamente en el *host* o anfitrión)
    2. **Seguridad.** Debe tener el control completo de los recursos físicos
       - Debe proporcionar **aislamiento** entre las VM (y entre el hipervisor y las VM)
    3. **Rendimiento.** Debe proporcionar un rendimiento próximo al nativo
       - Una fracción estadísticamente significativa de las instrucciones máquina deberían ejecutarse **directamente sobre el HW sin intervención del hipervisor**
  - Se denominan hipervisores eficientes los que cumplen la tercera propiedad

Virtualización basada en hipervisor

- Tipos de hipervisores (Popek & Goldberg)
  - Tipo 1 (nativo o *bare metal*)
    - Se ejecuta directamente sobre el HW del servidor controlando todos los recursos físicos
    - Implementa funciones básicas de un SO (*microkernel*)
    - Más eficientes y seguros que los de tipo 2, pero más complejos de desarrollar/mantener
    - Ejemplos
      - Xen, VMware ESXi, Microsoft Hiper-V
  - Tipo 2 (*hosted*)
    - Necesita un SO anfitrión sobre el que instalarse
    - Añade una capa intermedia entre el SO anfitrión y las VM (más cambios de contexto)
    - Ejemplos
      - VMware WorkStation, Parallels Desktop, VirtualBox



Virtualización basada en hipervisor

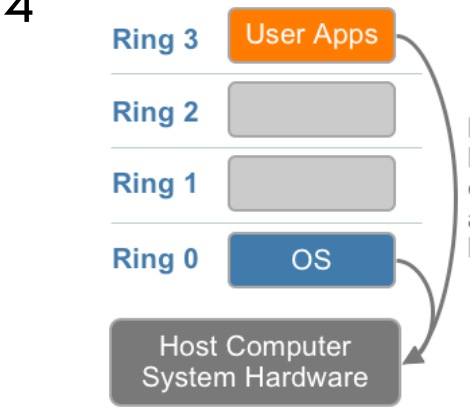
- Virtualización de una arquitectura de instrucciones
  - Popek & Goldberg propusieron un **teorema** que enuncia un **requisito suficiente** para que una arquitectura sea virtualizable
    - Asumen que la arquitectura dispone de al menos dos niveles de privilegio:
      - Modo usuario, en el que se ejecutan las aplicaciones
      - Modo *kernel* (o supervisor), en el que se ejecuta el *kernel* del SO en un entorno nativo (no virtualizado)
    - El teorema establece que una arquitectura es virtualizable siempre que las **instrucciones sensibles** sean un **subconjunto** de las **privilegiadas**
      - Una instrucción es **sensible** si:
        - Modifica el estado del procesador (p.e. cambiar el modo de ejecución, modificar el contador de programa o PC)
        - Accede o expone la configuración de los recursos del sistema (p.e. obtener el modo de ejecución actual)
        - Su comportamiento depende del estado actual del procesador
      - Una instrucción es **privilegiada** si solo puede ejecutarse en modo *kernel* y lanza una excepción (***trap***) si se intenta ejecutar en modo usuario

Virtualización basada en hipervisor

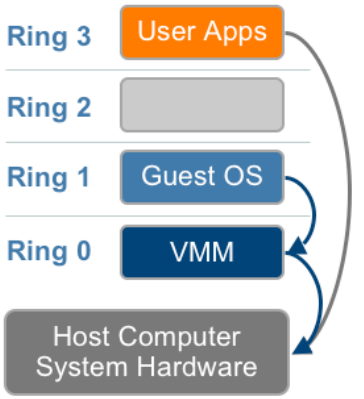
- Virtualización de una arquitectura de instrucciones
  - Popek & Goldberg también propusieron un modelo de virtualización para implementar un hipervisor eficiente: ***trap-and-emulate***
  - Este modelo de virtualización se basa en que:
    - Todas las instrucciones que afecten al funcionamiento "correcto" del VMM (sensibles) deben lanzar excepciones (***traps***) que puedan ser capturadas y gestionadas por este (**emuladas**) para conseguir así el control total sobre los recursos físicos
    - El resto de instrucciones (que son **inocuas**) deben **ejecutarse de forma directa** en el procesador (sin emulación ni intervención del VMM) para maximizar el rendimiento
      - Esto implica que la arquitectura virtualizada debe ser la misma que la arquitectura del *host*

Virtualización basada en hipervisor

- Virtualización de la arquitectura x86
  - La arquitectura x86 ofrece un modo protegido con 4 niveles (anillos) de privilegio
    - El SO se ejecuta en el nivel 0 (el más privilegiado)
    - Las aplicaciones en el nivel 3
  - Hay un subconjunto de **instrucciones privilegiadas**
    - Sólo pueden ejecutarse en el nivel 0
    - Si se ejecutan fuera de él provocan una excepción
  - En un entorno virtualizado mediante *trap-and-emulate*
    - El VMM se ejecuta en el nivel 0
    - El SO de la VM en el nivel 1 (menos privilegiado)
    - Las aplicaciones en el nivel 3
    - Cuando el *kernel* del SO invitado ejecuta una instrucción privilegiada se producirá una excepción que será capturada por el VMM para su emulación



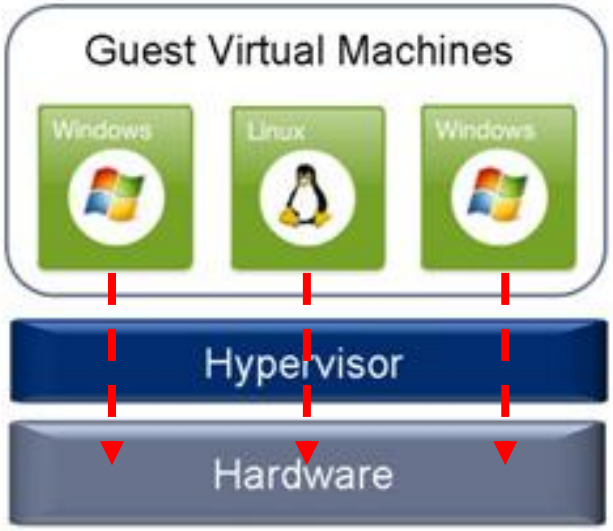
Entorno nativo (no virtualizado)



Entorno virtualizado

Virtualización basada en hipervisor

- Virtualización de la arquitectura x86
  - **Problema**
    - La arquitectura x86 tiene instrucciones **sensibles no privilegiadas**: no lanzan excepciones al ejecutarse en un nivel de usuario
      - El SO invitado de una VM podría usar esas instrucciones "problemáticas" (denominadas **críticas o no virtualizables**) para acceder a la configuración de los recursos físicos sin que el VMM pueda evitarlo
      - La arquitectura x86 no se diseñó desde un principio para ser virtualizable

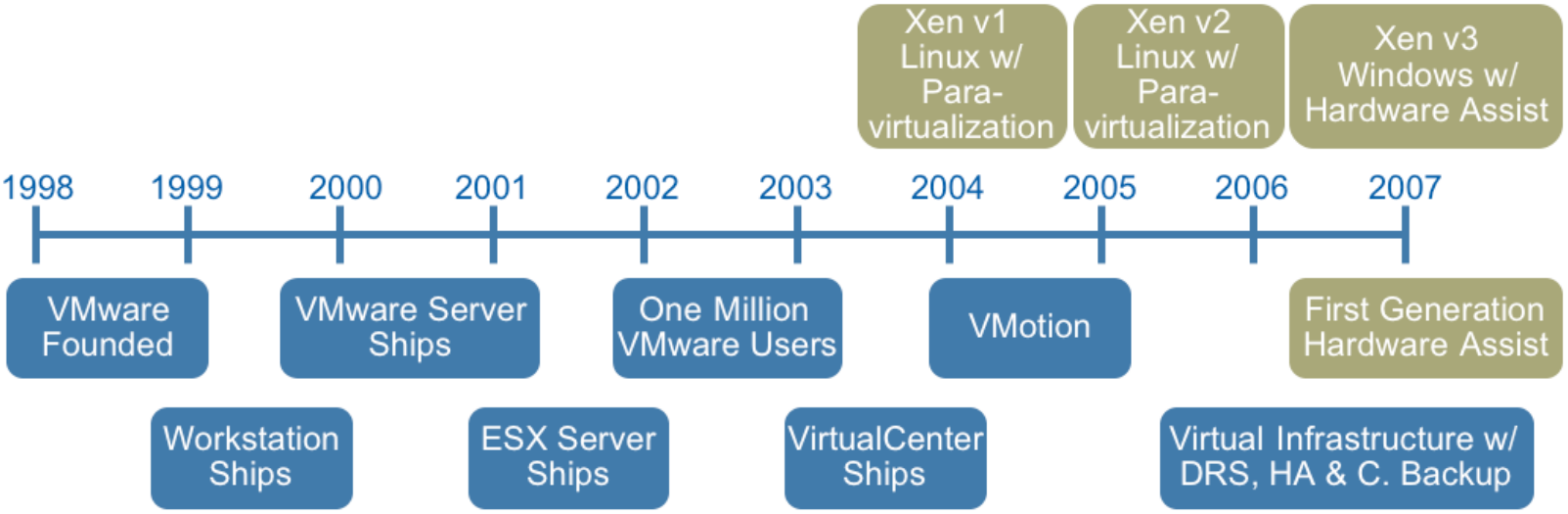


Virtualización basada en hipervisor

- Virtualización de la arquitectura x86
  - **Problema**
    - La arquitectura x86 tiene instrucciones **sensibles no privilegiadas**: no lanzan excepciones al ejecutarse en un nivel de usuario
      - El SO invitado de una VM podría usar esas instrucciones "problemáticas" (denominadas **críticas o no virtualizables**) para acceder a la configuración de los recursos físicos sin que el VMM pueda evitarlo
      - La arquitectura x86 no se diseñó desde un principio para ser virtualizable
    - Esto, según el teorema de Popek & Goldberg, impide su virtualización usando la técnica *trap-and-emulate*, ya que esas instrucciones podrían acceder a los recursos sin estar bajo el control del VMM
  - Se tardó **casi 25 años** en tener una solución a este problema
  - En la actualidad hay varias tecnologías o **modos de virtualización** de la arquitectura x86

Virtualización basada en hipervisor

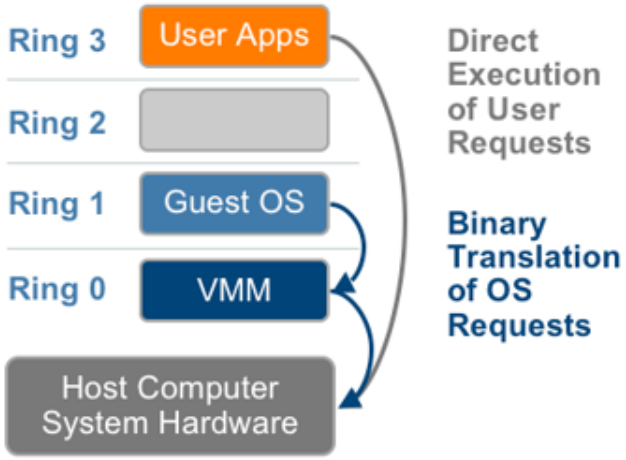
- Modos de virtualización de la arquitectura x86
  - Virtualización completa
  - Paravirtualización
  - Virtualización nativa o asistida por hardware



Evolución temporal de los modos de virtualización de la arquitectura x86

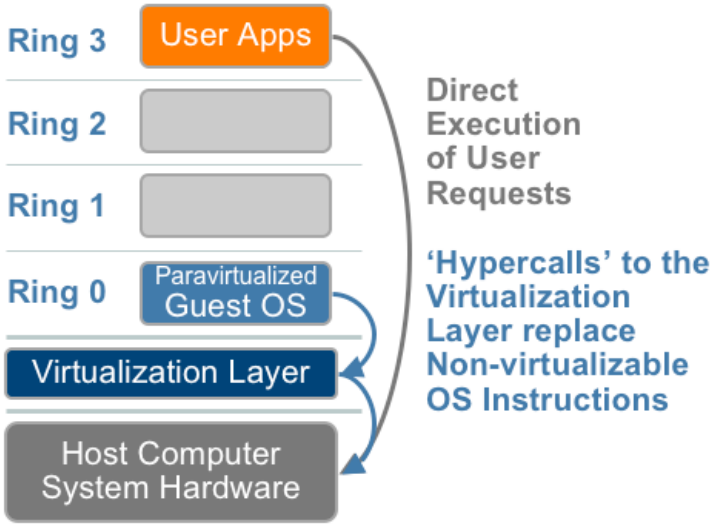
Virtualización basada en hipervisor

- Virtualización completa
  - Combina dos técnicas
    - Instrucciones no privilegiadas ni sensibles ("no problemáticas" o inocuas) se ejecutan **directamente en el *hardware***
    - Se detectan las instrucciones "problemáticas" y se substituyen por secuencias de instrucciones que **emulan su funcionamiento por *software*** en el VMM
      - Se usa **traducción binaria dinámica**
  - Características
    - Transparente para el SO invitado (no requiere modificaciones para virtualizarlo)
    - No requiere soporte del HW
    - Rendimiento algo más pobre
    - VMM complejos de desarrollar/mantener
  - Ejemplo: VMware Workstation/ESXi



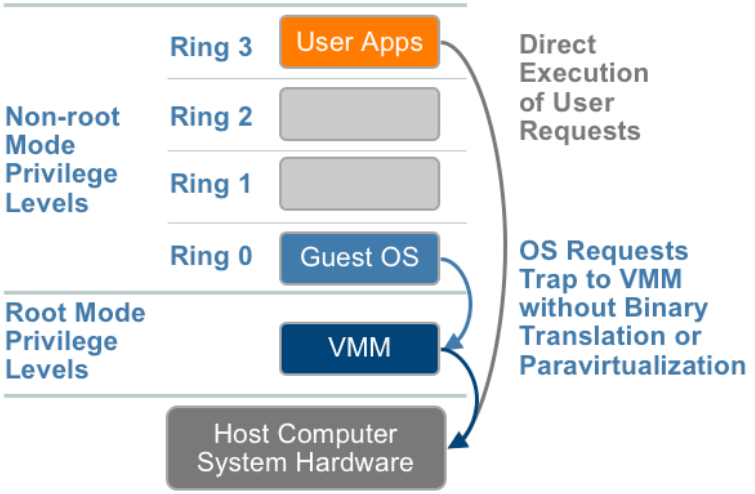
Virtualización basada en hipervisor

- Paravirtualización
  - El VMM proporciona una API a los SO invitado de las VM
    - Las instrucciones "problemáticas" se substituyen por llamadas a esa API (*hypercalls*)
    - Ejecución directa en el HW (sin emulación)
  - Características
    - El *kernel* del SO invitado tiene que ser modificado (y recompilado)
      - El SO invitado "sabe" que está siendo virtualizado
    - No requiere soporte del HW
    - Menor portabilidad (p.e. Windows)
    - Mejor rendimiento
  - Ejemplo: Xen (en CPUs sin soporte HW para virtualización)



Virtualización basada en hipervisor

- Asistida por *hardware* (*native virtualization* o HVM)
  - Los procesadores x86 modernos incluyen extensiones de virtualización HW
    - **Intel VT-x, AMD-V**
  - El VMM se ejecuta en un nuevo modo *root* más privilegiado
  - El SO invitado se ejecuta en el nivel 0 (más privilegiado) del modo *no-root*
  - Las instrucciones privilegiadas y sensibles provocan excepciones en el nivel de *root* que son capturadas por el VMM sin el *overhead* de técnicas previas
  - Estas extensiones aceleran por HW otras operaciones básicas de los VMM (p.e. traducción de direcciones de memoria)

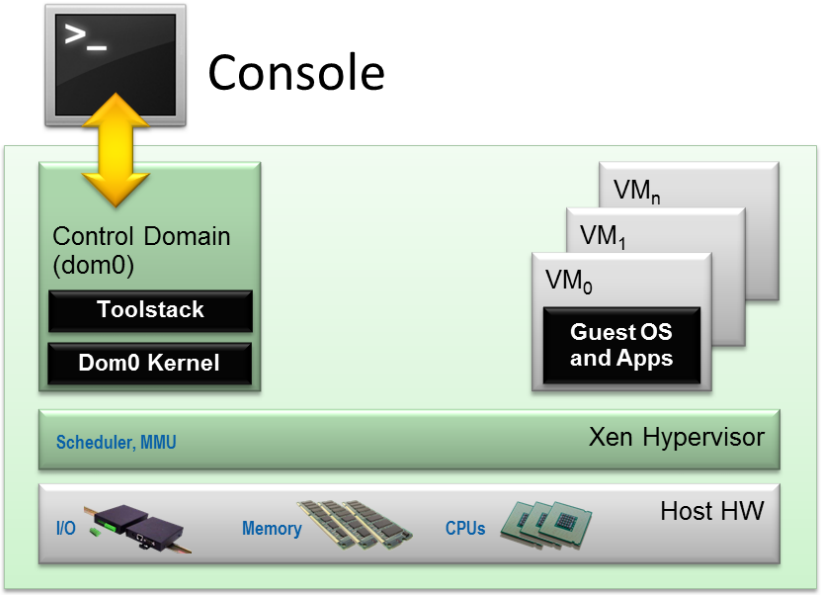


Virtualización basada en hipervisor

- Asistida por *hardware* (*native virtualization* o HVM)
  - Características
    - No requiere modificar el SO invitado pero necesita CPUs con soporte HW para virtualización
    - Buen rendimiento, aunque es dependiente de la frecuencia y *overhead* de las transiciones entre el modo *root* y el modo *no-root*
      - La primera generación de soporte HW para virtualización imponía un *overhead* significativo (y fijo) en cada transición
      - En algunos escenarios podía tener menor rendimiento que la paravirtualización
      - Generaciones posteriores mejoraron enormemente este aspecto
  - La virtualización asistida por *hardware* es complementaria de las anteriores
    - Los principales hipervisores soportan diferentes modos de virtualización
  - Ejemplos: Xen (PV+HVM), VMware ESXi (FV+HVM)

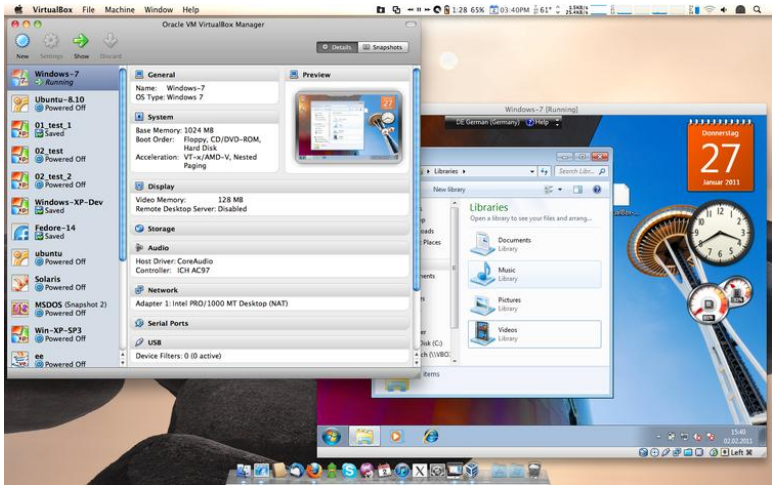
Hipervisores

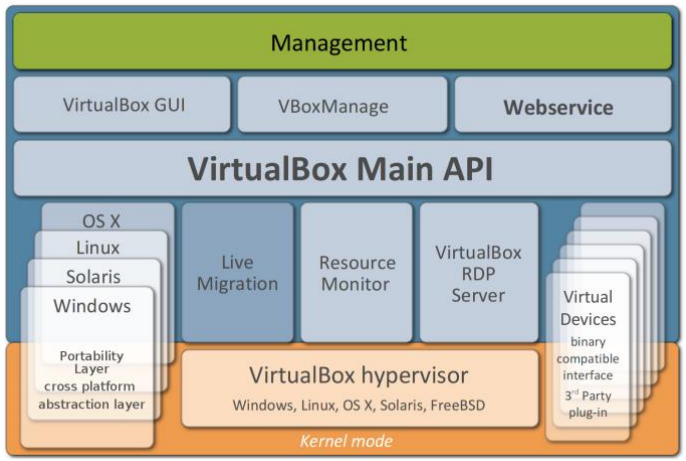
- Xen
  - Xen es un hipervisor tipo 1 que introdujo la paravirtualización
    - Actualmente puede ejecutar *guests* PV o HVM
  - El hipervisor se ejecuta directamente en el HW y gestiona la CPU, la memoria y las interrupciones, pero no la E/S
  - Las VM se denominan "**dominios**"
  - Hay un dominio privilegiado denominado **dom0** y el resto son dominios sin privilegios (**domU**)
    - dom0 contiene los *drivers* de dispositivo, puede acceder directamente al *hardware*, controla el acceso a las funciones de E/S y expone una interfaz de control para la gestión de los domU



Hipervisores

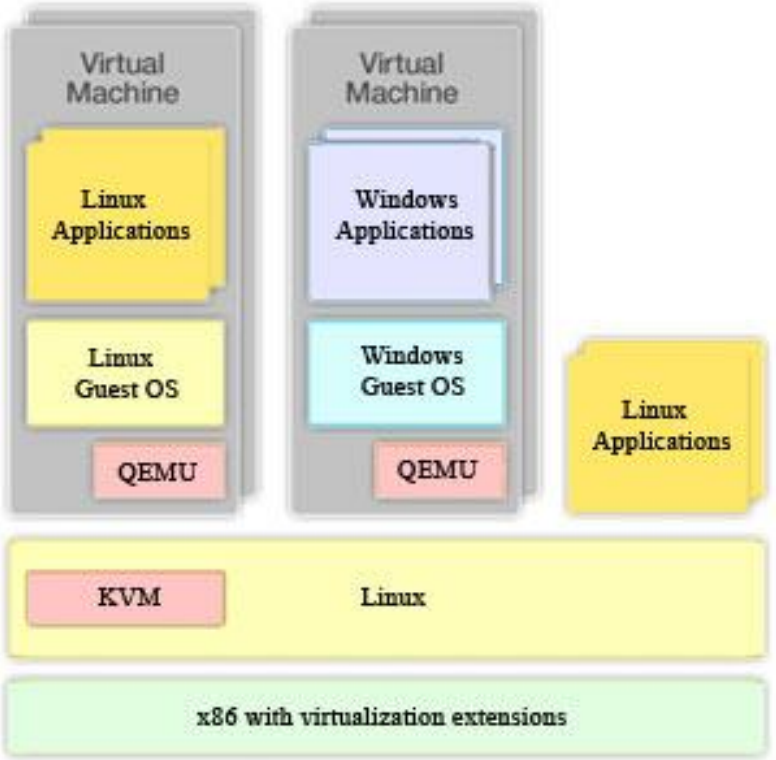
- Oracle VirtualBox
  - Hipervisor tipo 2 de código abierto y multiplataforma desarrollado en la actualidad por Oracle
    - Disponible para Linux, macOS, Windows, Solaris, FreeBSD…
  - Desde la versión 4.0, el "*core*" se distribuye bajo GPLv2
    - El *extension pack* con soporte para otras características (USB 2.0/3.0, encriptación de discos, PXE) se distribuye bajo licencia PUEL (*Personal Use and Evaluation License*): gratuito para uso personal (sin propósitos comerciales), educacional o para su evaluación





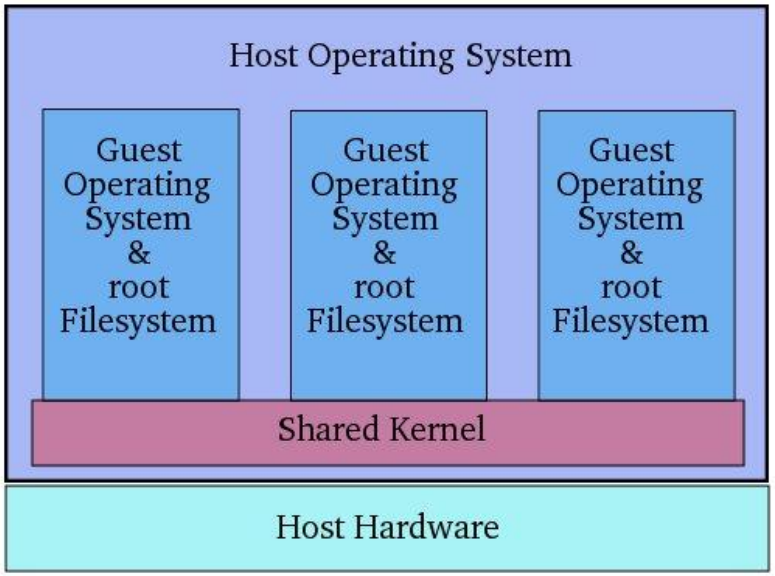
Virtualización a nivel de *kernel*

- Técnica propia de sistemas Linux que no requiere de un hipervisor
- Las VM se ejecutan como procesos en espacio de usuario
- KVM (*Kernel Virtual Machine*)
  - Módulo que permite usar el *kernel* Linux a modo de hipervisor tipo 2
  - Requiere un procesador con extensiones de virtualización (Intel VT-x, AMD-V)
  - Puede ejecutar VM con diferentes SO invitado en sistemas Linux
    - Windows, Linux, Solaris, BSD, …
  - Usa una versión modificada de QEMU para emular el HW de las VM
    - Las VM se ejecutan directamente en el HW sin usar traducción binaria



Virtualización por compartición de *kernel*

- Técnica propia de sistemas UNIX y Linux
  - El concepto surge desde los inicios de UNIX con el comando *chroot*
- El SO proporciona capacidades de virtualización que permite múltiples entornos/ambientes virtuales ejecutarse sobre el **mismo *kernel***
  - Los entornos se ejecutan en espacio de usuario y aislados entre sí
  - Cada entorno sólo "ve" una versión virtual propia de los recursos disponibles
  - También se conoce como virtualización a nivel de sistema operativo

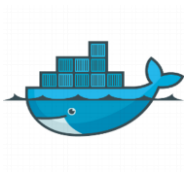


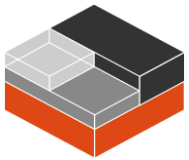
Virtualización por compartición de *kernel*

- El *kernel* del SO anfitrión debe proporcionar, entre otros, dos mecanismos:
  - Particionado de los recursos y limitación/control de acceso a los mismos
  - Aislamiento entre procesos
- Los recursos del *kernel* se comparten entre los entornos virtuales que tienen la ilusión de que se ejecutan con su propio sistema de ficheros, CPU, memoria y E/S
  - Usan la interfaz de llamadas al sistema del *kernel* del SO anfitrión sin necesidad de emulación HW ni de un hipervisor
  - Los entornos virtuales de ejecución reciben diferentes denominaciones:
    - Contenedores, zonas, particiones, jails, servidores privados virtuales (VPS), …
- Ventajas
  - Se pueden ejecutar un gran número de entornos virtuales con un rendimiento muy cercano al nativo
  - No es necesario lanzar una VM con un SO completo para ejecutar un simple servicio
- Desventajas
  - Los entornos virtuales tienen que ser compatibles con el *kernel* del SO anfitrión: no es posible ejecutar diferentes SO a la vez

Virtualización por compartición de *kernel*

- Ejemplos
  - Docker
  - Linux Containers (LXC/LXD)
  - Solaris Containers (Zones)
  - FreeBSD Jails
  - Linux-VServer
  - Open Virtuozzo (OpenVZ)
  - Singularity/Apptainer
  - Podman
  - udocker
  - CoreOS Rkt (Rocket)



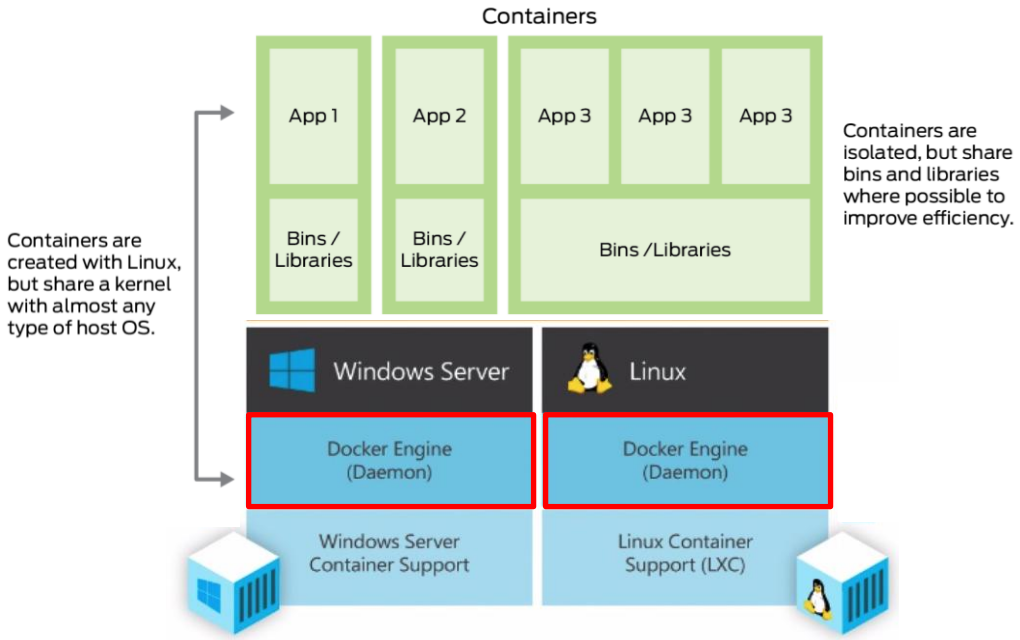


Virtualización por compartición de *kernel*

- Linux Containers (LXC/LXD)
  - LXC permite ejecutar múltiples instancias de Linux aisladas (contenedores)
  - Se basa en tres funcionalidades proporcionadas por el *kernel* de Linux
    - ***Namespaces***: aislamiento de recursos entre procesos independientes
      - Particionado de recursos de forma que un proceso "ve" un determinado conjunto de recursos distinto al que "ve" otro proceso
      - Ejemplos de *namespaces*: *process IDs, user IDs, file names,…*
      - El *namespace* "*process IDs*" permite que dos procesos ejecutándose en la misma máquina puedan tener el mismo identificador o PID
    - ***pivot_root***: mecanismo *chroot-like* para cambiar el sistema de ficheros raíz que un determinado proceso puede "ver" y usar (proceso "enjaulado")
    - ***control groups* (*cgroups*)**: organización de procesos en grupos jerárquicos cuyo uso de recursos (p.e. CPU) se puede monitorizar, limitar y contabilizar
      - Un *cgroup* es un conjunto de procesos que están limitados por el mismo criterio y tienen asociado un determinado límite en el uso de recursos
  - LX Daemon (LXD) permite gestionar contenedores a través de la red de forma transparente mediante una API REST que LXC usa por debajo

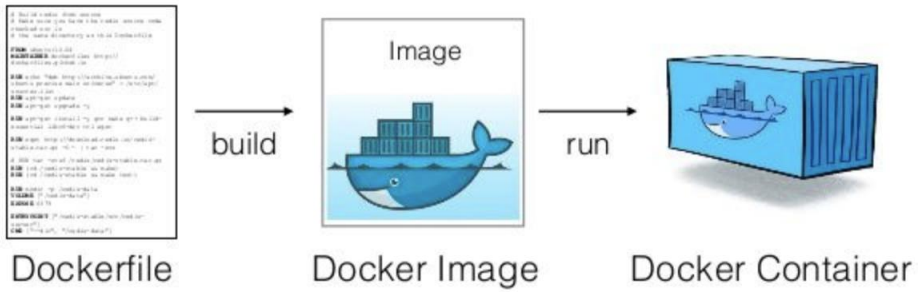
Virtualización por compartición de *kernel*

- Docker
  - Liberado como proyecto *open-source* por dotCloud en 2013
  - Al igual que LXC, utiliza las características del *kernel* de Linux (*namespaces*, *cgroups*,…) para proporcionar los entornos virtuales o contenedores
    - Originalmente, Docker Engine se implementó sobre Linux usando LXC
    - Actualmente disponible para Linux, Windows y macOS



Virtualización por compartición de *kernel*

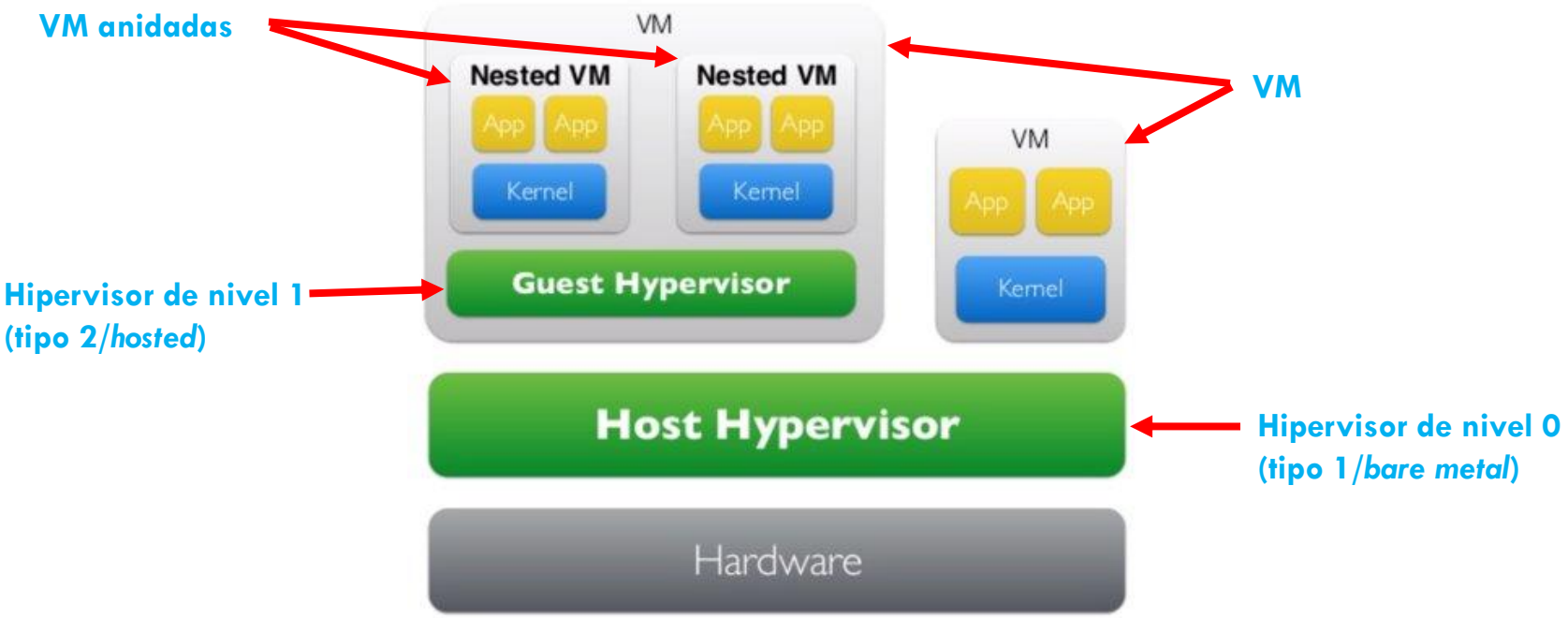
- Docker
  - Docker se basa en el concepto de **imágenes**
    - Una imagen es un componente SW que contiene todo lo necesario para ejecutar una aplicación (código, binarios, librerías, ficheros de configuración…)
      - Se crean a partir de un fichero ***Dockerfile*** que contiene la "receta" (instrucciones) para crear una imagen
      - Una imagen se puede basar en otra imagen con personalización adicional
        - P.e. se puede crear una nueva imagen basada en la imagen de Ubuntu pero con el servidor web Apache y una aplicación web pre-instaladas
    - Un **contenedor** Docker es una instancia en ejecución de una imagen



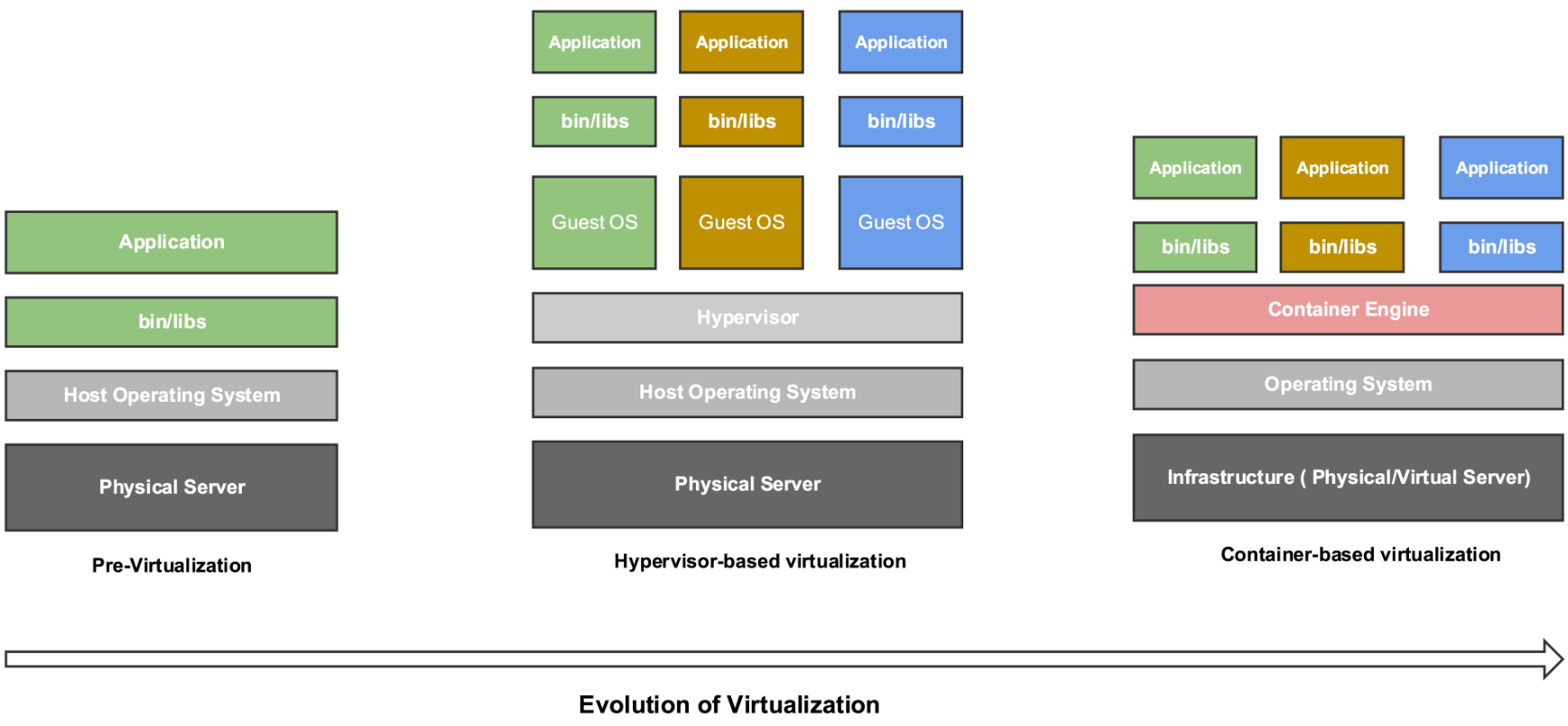
Virtualización por compartición de *kernel*

- Docker
  - Docker se basa en el concepto de **imágenes**
    - Una imagen es un componente SW que contiene todo lo necesario para ejecutar una aplicación (código, binarios, librerías, ficheros de configuración…)
      - Se crean a partir de un fichero ***Dockerfile*** que contiene la "receta" (instrucciones) para crear una imagen
      - Una imagen se puede basar en otra imagen con personalización adicional
        - P.e. se puede crear una nueva imagen basada en la imagen de Ubuntu pero con el servidor web Apache y una aplicación web pre-instaladas
    - Un **contenedor** Docker es una instancia en ejecución de una imagen
    - Un registro Docker (**Docker Registry**) es un repositorio de imágenes
      - Docker Hub y Docker Cloud son repositorios públicos que cualquiera puede usar, pero también es posible crear repositorios privados
    - *Union File System* (UnionFS) proporciona la capacidad de crear imágenes en capas
      - Permite montar un sistema de ficheros formado a partir de varios sistemas de ficheros: los archivos y directorios de los distintos sistemas se superponen de forma transparente formando un único sistema de ficheros virtual

- Virtualización anidada (*nested virtualization*)
  - Ejecución de una capa de virtualización dentro de otra
    - Ejemplo: un hipervisor ejecutándose dentro de una VM
    - Puede aplicarse tanto para ejecutar VM como contenedores dentro de una VM
    - No todos los hipervisores ni SO lo permiten



Virtualización de servidores: Evolución



Contenidos

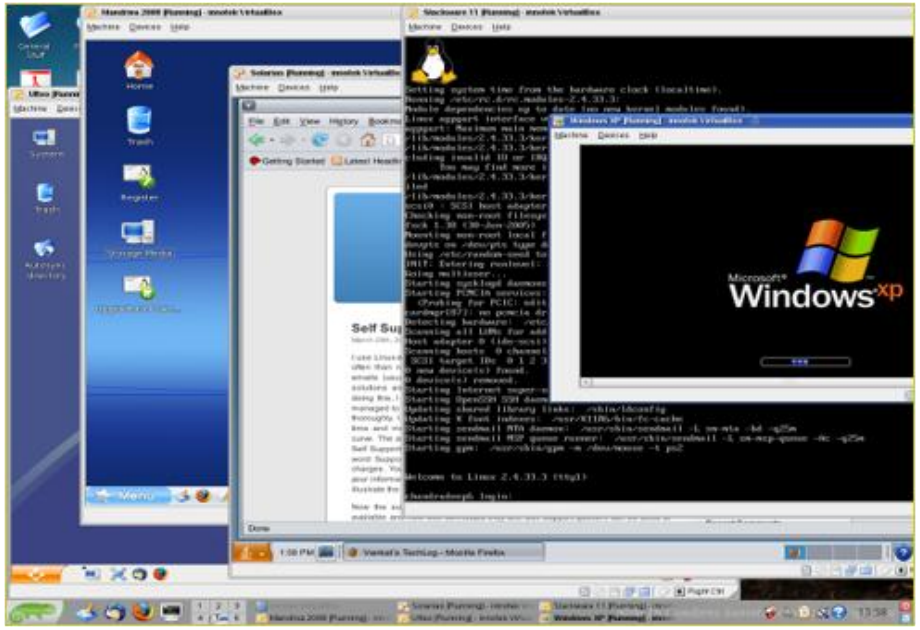
- Introducción
- Tipos de virtualización
- Virtualización de servidores
- **Beneficios/inconvenientes**
- Estándares

Algunos beneficios de la virtualización

- Tener múltiples SO en un mismo servidor
- Tener servidores virtuales con diferentes configuraciones SW o HW
- Consolidación de servidores
- Agregación de servidores (*Server pooling*)
- Entornos de desarrollo y prueba
- Formación
- Tolerancia a fallos
- Balanceo de carga
- Recuperación ante desastres

Algunos beneficios de la virtualización

- Múltiples SO en un mismo servidor
  - El servidor puede dar soporte a múltiples usuarios que usen SO diferentes
  - Puede ejecutar aplicaciones que no estén disponibles en su propio SO
  - Puede ejecutar el mismo SW que otros servidores con SO distinto



Diferentes SO ejecutándose simultáneamente en el mismo servidor

Algunos beneficios de la virtualización

- Servidores virtuales con diferentes configuraciones SW o HW
  - El servidor puede ejecutar diferentes versiones de un mismo SO o aplicación
    - Ejemplos
      - Soporte y mantenimiento de versiones anteriores de las aplicaciones
      - Ejecución de aplicaciones antiguas (*legacy*) en HW nuevo
  - También puede ejecutar variantes de una misma VM con configuraciones HW distintas

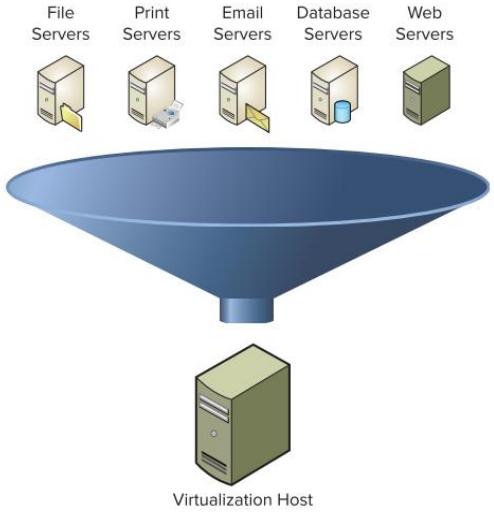
Diferentes versiones de Windows ejecutándose simultáneamente

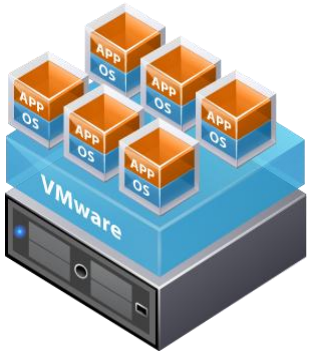
Algunos beneficios de la virtualización

- Consolidación de servidores
  - Algunos problemas en los centros de datos (CPD) tradicionales
    - Aproximación monolítica: una aplicación, un servidor ➔ proliferación de servidores (*Server Sprawl*)
      - Desperdicio de energía y espacio
    - Los servidores cada vez son más potentes ➔ recursos infrautilizados
      - No es raro un 90%-95% CPU idle time, memoria libre desperdiciada…
    - *Lost Servers*: servidores de los que no se sabe qué hacen o a quién pertenecen ➔ miedo a pararlos por el riesgo de afectar a algo crítico
    - Problemas de espacio en el CPD, consumo de energía, complejidad de la administración, …

Algunos beneficios de la virtualización

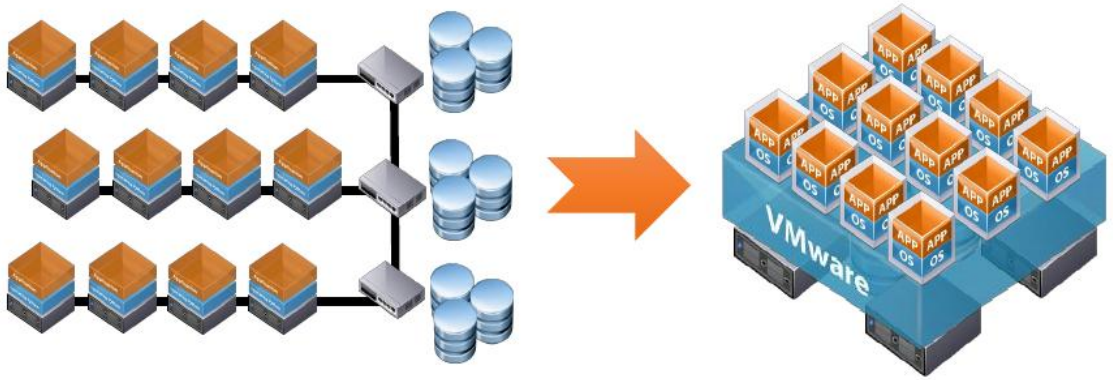
- Consolidación de servidores
  - Substituir varios servidores físicos "monolíticos" por VM/contenedores ejecutados en un servidor de mayores prestaciones virtualizado
    - La consolidación reduce el número de servidores físicos y, por tanto, los costes, el consumo de energía y espacio en el CPD, la complejidad de la administración, el impacto medioambiental (emisiones CO2),…
  - La **ratio de consolidación** es una medida que indica el número de VM que se ejecutan por servidor físico (ratio 4:1 ➔ 4 VM por servidor)





Algunos beneficios de la virtualización

- Agregación de servidores (*Server pooling*)
  - La virtualización de servidores puede aplicarse a un grupo de servidores físicos (*clúster*) que son gestionados como una entidad única
  - El SW de virtualización se encarga automáticamente de:
    - Distribuir las VM entre los servidores físicos
    - Migrar las VM para optimizar el uso de recursos o proporcionar mecanismos de tolerancia a fallos
    - Gestionar de forma transparente la incorporación o eliminación de servidores al grupo



Algunos beneficios de la virtualización

- Entornos de desarrollo y prueba
  - Reproducir mediante virtualización entornos de prueba complejos
    - Entornos distribuidos, diferentes SO, diferentes versiones de librerías, parches, configuraciones, etc
  - Un único servidor físico puede reproducir uno o varios entornos de desarrollo y prueba diferentes
  - Los entornos virtuales están aislados entre sí y aislados del servidor (no afectan a su configuración)
    - Fallos severos en los entornos de pruebas no afectan al servidor físico
  - La recuperación del entorno de pruebas después de un fallo que provoque su caída es muy ágil
    - Cuestión de segundos o minutos

Algunos beneficios de la virtualización

- Formación
  - Cada estudiante tiene su propio entorno de pruebas, en el que tiene libertad para modificar lo que quiera (privilegios de *root*)
  - Cada entorno está aislado de los demás, por lo que lo que haga un estudiante no afecta a los otros
  - No es necesario disponer de *hardware* específico, cada estudiante puede hacer las pruebas en su propio equipo o mediante un escritorio virtual
  - La virtualización permite que la preparación y/o recuperación de un entorno de pruebas sea muy fácil

Algunos beneficios de la virtualización

- Tolerancia a fallos
  - Configuración simple: recuperación (casi inmediata) del estado de una VM después de un fallo (p.e. uso de *snapshots*)
    - Problema: el servidor físico es un SPOF
  - Configuración HA: configuraciones de *clustering* con servidores físicos de reserva (p.e. activo/pasivo)
    - Mediante el *software* de virtualización adecuado, las VM se recuperan en un servidor diferente
    - Requiere recuperar la configuración de red y almacenamiento de la VM
  - Balanceo de carga: configuraciones de *clustering* con balanceo de carga (p.e. activo/activo)
    - Las VM se ejecutan repartidas entre 2 o más servidores y se reparte la carga entre ellas

Algunos beneficios de la virtualización

- Recuperación ante desastres
  - La virtualización permite migrar VM (*offline* o en vivo)
    - Las VM se pueden migrar a cualquier servidor del CPD
  - Disponer de capacidad reservada en otro CPD para migrar las VM y los datos
  - Se elimina la necesidad de que la infraestructura del CPD de reserva sea una réplica de la original
  - Pueden crearse configuraciones de *clustering* y balanceo de carga entre CPD

Inconvenientes de la virtualización

- Los fallos *hardware* pueden ser más críticos
  - Consolidación de servidores → servidor es un SPOF
- Problemas de rendimiento, falta de capacidad y congestión en el uso de los recursos
  - Múltiples VM con uso intensivo de CPU, memoria o red
- La capa de abstracción añade mayor complejidad y dificultad en la depuración de errores
  - Más componentes que depurar (p.e. en redes virtuales: vNIC, *vSwitches*, vIP, vLAN, …)
- Si no se dispone de las herramientas de automatización adecuadas, la administración de entornos virtualizados puede complicarse

Contenidos

- Introducción
- Tipos de virtualización
- Virtualización de servidores
- Beneficios/inconvenientes
- **Estándares**

Estándares

- OVF (*Open Virtualization Format*)
  - Estándar para el empaquetamiento y la distribución de VM
  - Propuesta realizada por el *Distributed Management Task Force* (DMTF) en cooperación con múltiples empresas:
    - VMware, XenServer, Microsoft, Citrix, IBM…
    - Especificación: https://www.dmtf.org/standards/ovf
  - Define un formato abierto, extensible, eficiente e **independiente de la plataforma**
  - Soportado en varios productos de virtualización
    - VirtualBox, VMware, XenServer, OpenStack…
  - Adoptado en el 2010 como estándar ANSI INCITS 469-2010

Estándares

- OVF (*Open Virtualization Format*)
  - Define un formato estándar de empaquetado de VM para la distribución segura de aplicaciones virtuales (*virtual appliances*)
    - Tiene en cuenta aspectos como
      - Compresión, integridad (firmas digitales), gestión de licencias (EULA)
      - Soporta configuraciones con varias VM (p.e. aplicaciones *multi-tier*)
      - Soporta los formatos de disco más habituales y proporciona medios para incorporar otros nuevos (URI a la especificación del formato)
    - Un paquete OVF contiene
      - Un descriptor: fichero XML con los metadatos del paquete
      - Un manifiesto (opcional): contiene los *hashes* SHA1 de los ficheros del paquete
      - Un certificado opcional
      - Un conjunto de imágenes de disco
      - Un conjunto de recursos adicionales (p.e. imágenes ISO)
  - Define un formato estándar de archivo: ***Open Virtualization Archive* (OVA)**
    - Permite la conveniente distribución de paquetes OVF en un único fichero
    - Es un formato *tarball* de los ficheros individuales que forman un paquete OVF

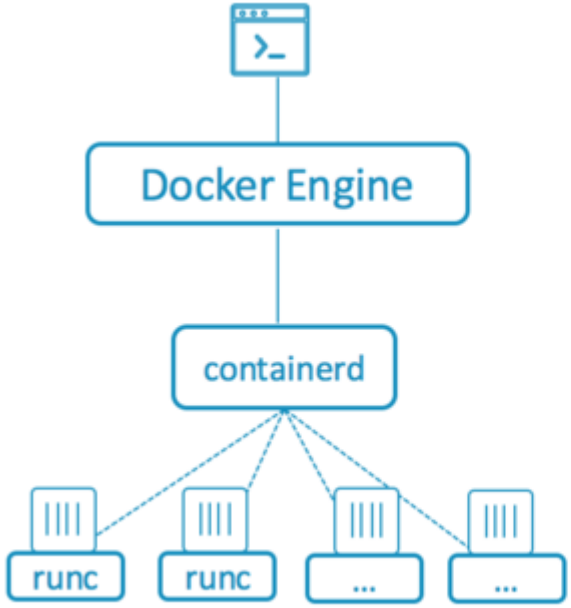
Estándares

- En el ámbito de la virtualización de contenedores, han surgido varios estándares para garantizar interoperabilidad y uniformidad
  - Estos estándares son promovidos, en su mayoría, por proyectos de la *Linux Foundation*, principalmente OCI y CNCF
- OCI (*Open Container Initiative*)
  - Define estándares abiertos para la creación y ejecución de contenedores
  - Sus especificaciones más importantes son:
    - OCI *Runtime Specification* (**runc**): define cómo un contenedor debe ser empaquetado, desplegado y ejecutado
      - runc: motor de ejecución de contenedores de "bajo nivel" e implementación de referencia desarrollada por OCI que crea y ejecuta los contenedores a nivel del SO
      - Existen otros motores OCI como crun y runsc
    - OCI *Image Specification*: estandariza el formato de las imágenes de contenedores para garantizar compatibilidad entre diferentes motores de ejecución
    - OCI *Distribution Specification*: estandarizar cómo se distribuyen las imágenes de contenedores y otros artefactos relacionados a través de los registros o *registries*

Estándares

- CNCF *(Cloud Native Cloud Foundation)*
  - Proyecto iniciado en 2015 para ayudar a impulsar las tecnologías de contenedores y alinear la industria tecnológica en torno a su evolución
    - En la actualidad alberga importantes proyectos como **Kubernetes**
  - Sus especificaciones más importantes son:
    - *Container Runtime Interface* (CRI): API desarrollada por Kubernetes que permite integrar diferentes motores de ejecución de contenedores
      - Implementaciones CRI populares: containerd (mantenido por la CNCF), CRI-O, MCR
        - Estos motores de ejecución de "alto nivel" pueden delegar en runc la ejecución
        - Docker utiliza internamente containerd como motor de ejecución
    - *Container Networking Interface* (CNI): estándar para gestionar las redes de contenedores, permitiendo conectar y desconectar redes de forma dinámica
      - Calico es una implementación popular del estándar CNI
    - *Container Storage Interface* (CSI): estándar que define una interfaz para integrar soluciones de almacenamiento con plataformas de orquestación de contenedores
      - Permite que los sistemas de almacenamiento dinámicos (NFS, Ceph, AWS EBS, …) se conecten con Kubernetes u otras plataformas

Estándares



Same Docker UI and commands

User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers