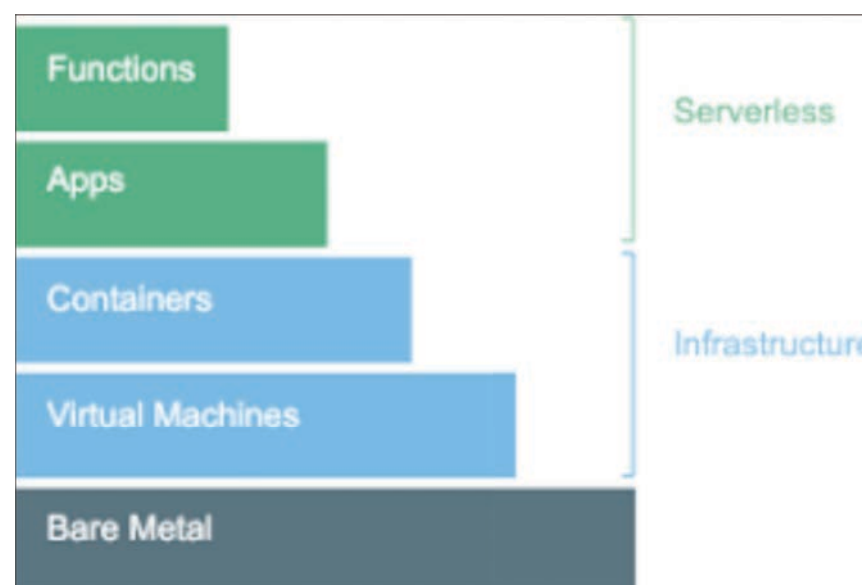


Introducción a la Virtualización

Objetivos

- Conocer el funcionamiento básico de las máquinas virtuales y los contenedores
- Distinguir entre la virtualización a nivel de sistema y de aplicación/sistema operativo
- Administrar de manera básica máquinas virtuales y contenedores
- Adquirir unas nociones básicas de seguridad con máquinas virtuales y contenedores



Fuente: google cloud



Índice

Máquinas Virtuales

-  Fundamentos

-  Técnicas de implementación

-  Administración de MV con qemu/libvirt/kvm en debian

-  Seguridad básica con MV

Virtualización de Sistema Operativo: Contenedores

-  Fundamentos

-  Soporte del Sistema operativo

-  Administración básica de contenedores con docker

-  Seguridad básica con contenedores

Máquinas Virtuales

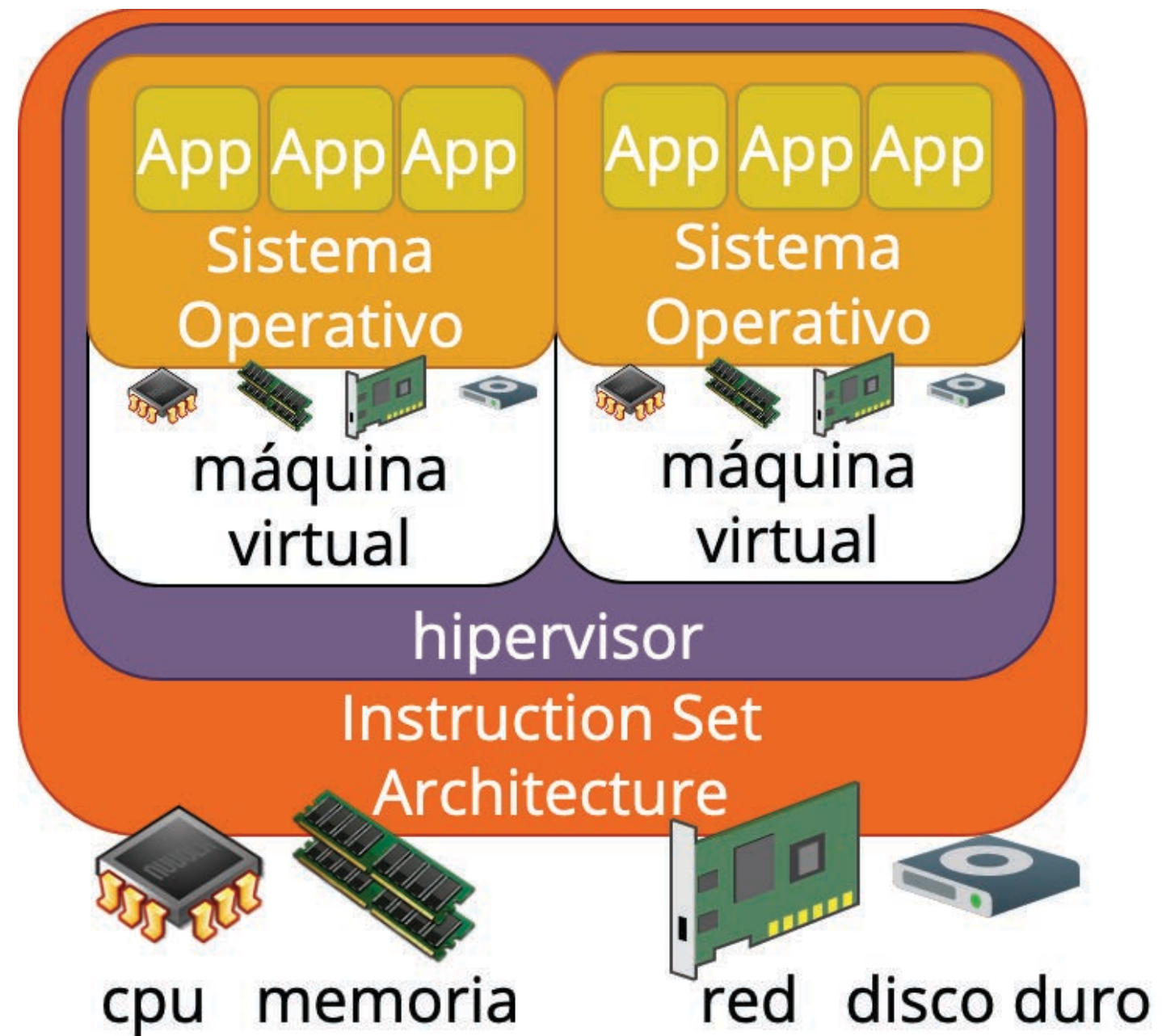
An isometric illustration on a dark blue background with a glowing circuit pattern. In the center, a large, glowing wireframe brain sits atop a cluster of server racks. To the left, a laptop and a tablet are shown. To the right, a desktop monitor, keyboard, and a smartphone are visible. Various gears and a stack of coins are scattered around the central server area, symbolizing technology, data, and virtualization.

fuelle: Nadine_C, Getty Images/iStockphoto

Máquina Real vs. Virtual



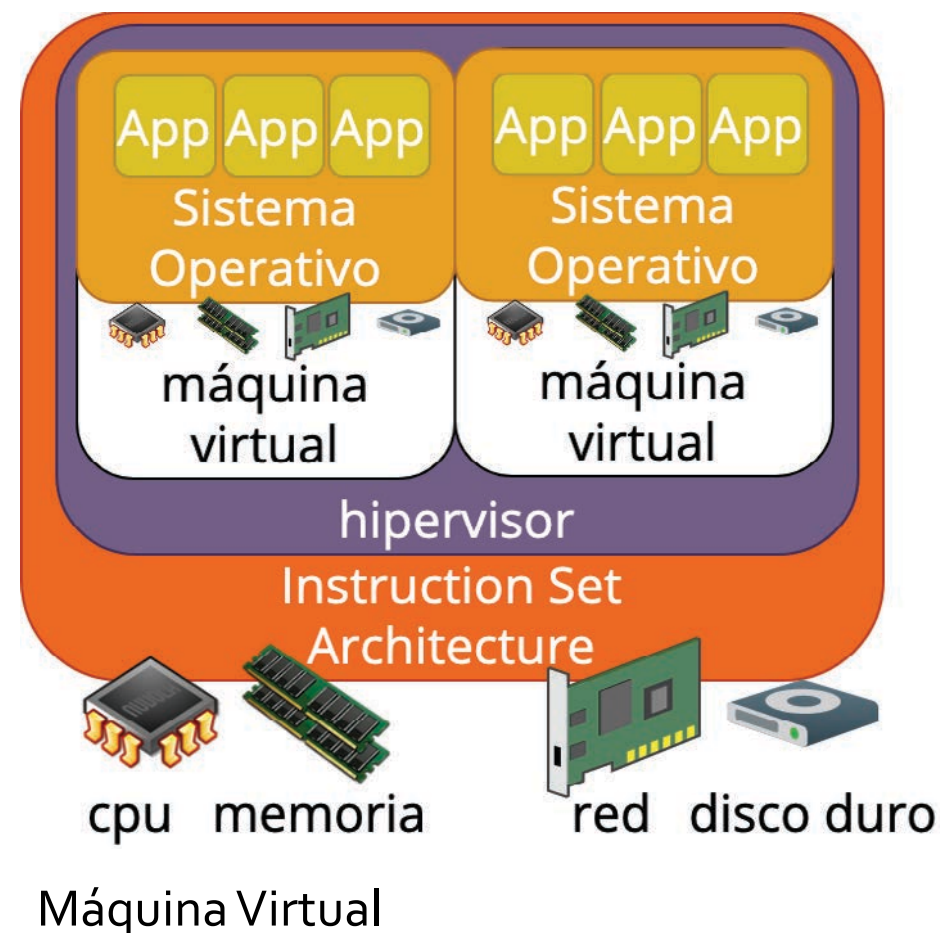
Máquina Real




Máquina Virtual

Hipervisor o Virtual Machine Monitor

 **Hipervisor o Virtual Machine Monitor:** componente, normalmente software, que intermedia entre las máquinas virtuales que aloja y el hardware real. Además, presenta un hardware virtual al sistema operativo virtualizado



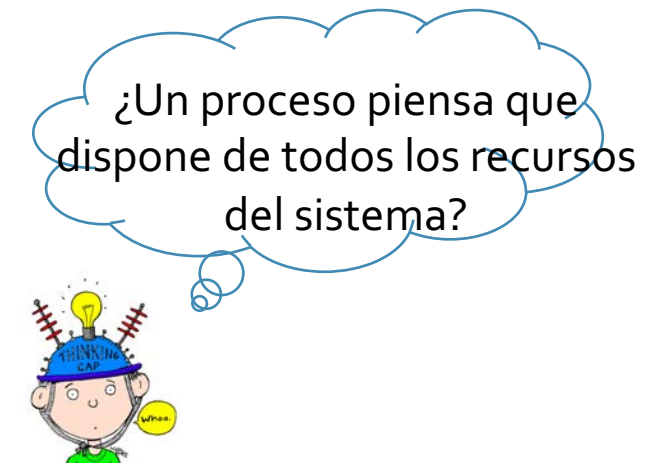
Ventajas de las máquinas virtuales







 **Seguridad:** las máquinas virtuales se ejecutan aisladas gracias a la capa de abstracción extra, los hipervisores son más sencillos que los sistemas operativos y suelen presentar menos vulnerabilidades, permiten monitorización de manera sencilla

 **Eficiencia:** multiples sistemas operativos corriendo sobre el mismo hardware aumentan la utilización


 **Flexibilidad:** ajuste dinámico de recursos entre máquinas virtuales, migración de máquinas incluso en ejecución, recuperación de estado, repositorio de imágenes, ...

Interfaz del SO vs Hipervisor





-  El Sistema operativo ya “virtualiza” los recursos del sistema mediante la abstracción
-  Los hilos piensan que se ejecutan en una CPU propia
-  La memoria se virtualiza (direcciones lógicas -> direcciones físicas)
-  Se accede a los dispositivos mediante abstracciones de alto nivel. Por ejemplo, leer en un fichero o escribir un socket...
-  El hipervisor ofrece una abstracción de más bajo nivel
-  El sistema operativo cree que se ejecuta en hardware real y no es consciente de la virtualización, en principio, ...


Implementación de Virtualización

 La virtualización requiere “simular” un computador completo, procesador, memoria, almacenamiento, redes, timers, todo tipo de dispositivos, ...

 Requerimientos para un hipervisor:

 **Seguridad:** El hipervisor tiene que tener control completo de los recursos virtualizados

 **Fidelidad:** El comportamiento de un programa sobre un hipervisor tiene que ser idéntico a su comportamiento al ejecutarse en hardware real





 **Eficiencia:** Buena parte del código de la máquina virtual debe ejecutarse sin intervención del hipervisor (requiere misma ISA en anfitrión y huésped)

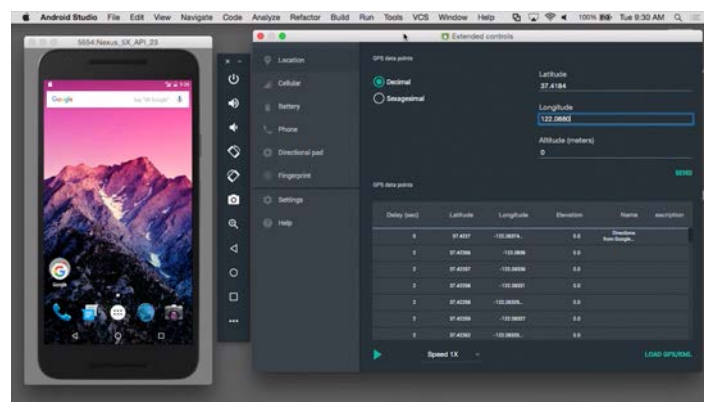
Fuente: G. Popek y R. Golberg. Formal Requirements for Virtualizable Third Generation Architectures, Communications of the ACM, 1974

Técnicas en la implementación de Máquinas Virtuales

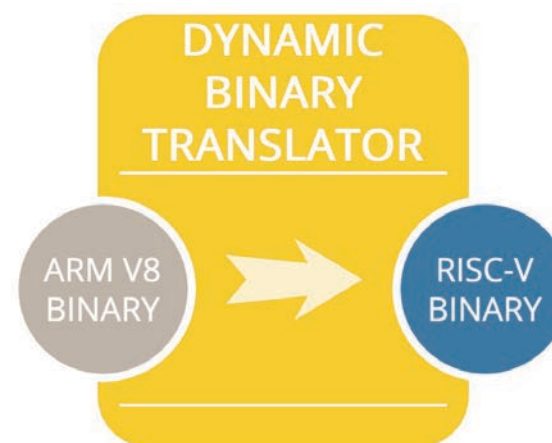
¿emuladores consolas,
OS móviles, ...?



-  **Emuladores:** Herramienta hardware o software que permiten a un sistema informático comportarse como si fuera otro para permitir la ejecución/utilización de aplicaciones/dispositivos del segundo en el primero
-  **Traducción binaria dinámica:** traductor de una ISA a otra
-  **Soporte Hardware:** Gestión de memoria en huésped/anfitrión, captura de instrucciones ...
-  **Paravirtualización:** El SO huésped es consciente de la virtualización y solicita tareas/ayuda al hipervisor



Emulador



Traductor binario



Paravirtualización

Tipos de hipervisores y ejemplos

🏛️ Nomenclatura un poco difusa

🏛️ Tipo 1 (se ejecutan directamente sobre el hardware)



código abierto



código propietario



Tipo 1

🏛️ Tipo 2 (se ejecutan sobre un sistema operativo)



código abierto



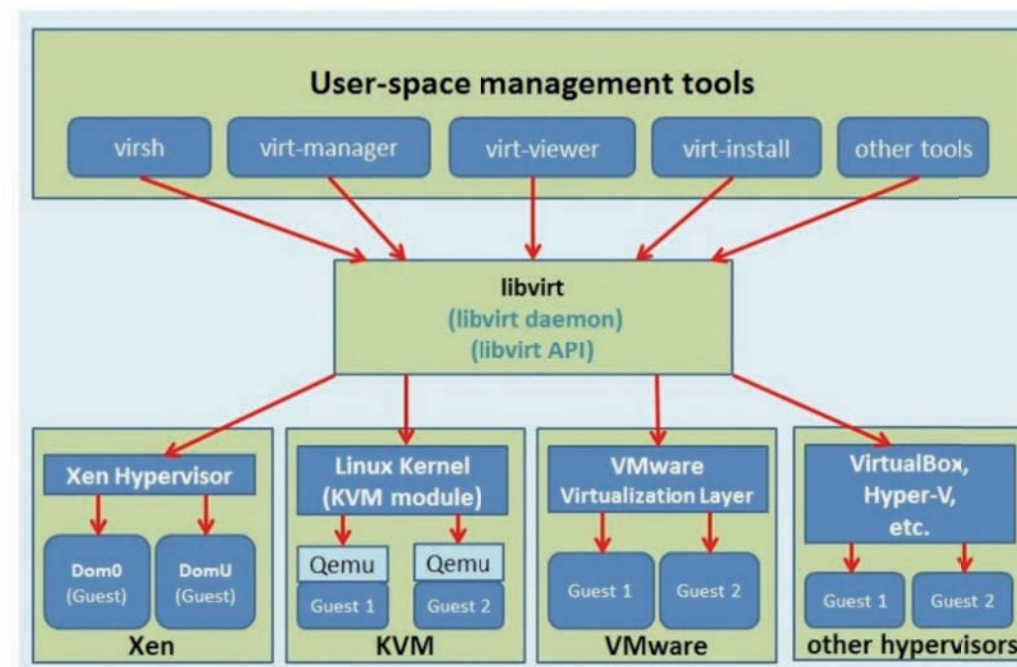
código propietario



Tipo 2

Ejecución de máquinas virtuales en Linux


- El kernel da soporte a la paravirtualización y suele emplear el soporte hardware
- En espacio usuario hay multitud de herramientas:
 - Muy habitual emplear libvirt, <https://libvirt.org/>, + herramientas adicionales como Virtual Machine Manager, <https://virt-manager.org/>
- Estas herramientas servirán para crear, lanzar, borrar MVs y dar soporte a drivers, ...



Fuente: Peter Larsen

Pasos previos para ejecutar MVs en debían (kvm/qemu)

Instalación de paquetes requeridos

```
 # apt-get install --no-install-recommends qemu-kvm  
libvirt-clients libvirt-daemon-system
```

Pertenencia a grupos requeridos

```
 #adduser <user> libvirt
```


Listado de MV (*guest domains* en terminología libvirt)

```
 # virsh list --all
```

```
 # sudo virsh --connect qemu:///system list --all  
// todas las máquinas de root
```

Creación de una máquina virtual

 Si disponemos de la imagen ISO

```
 # virt-install --virt-type kvm --name buster-amd64 \
  --cdrom ~/iso/Debian/debian-10.0.0-amd64-netinst.iso \
  --os-variant debian10 --disk size=10 --memory 1000
```

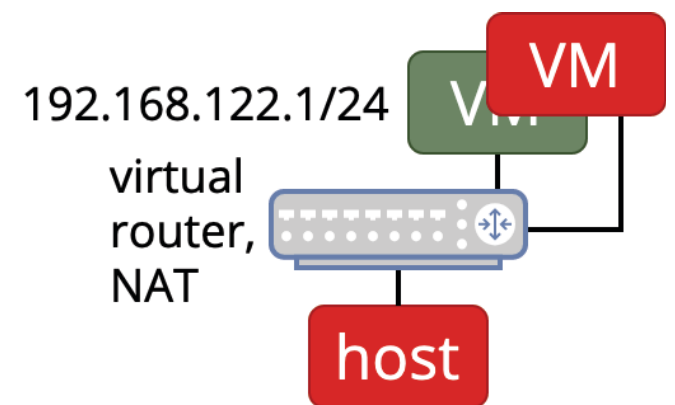
 También podemos emplear `--location` para no descargar la imagen

```
 # virt-install --virt-type kvm --name buster-amd64 \
  --location
  http://deb.debian.org/debian/dists/stable/main/installer-amd64/ \
  --extra-args "console=ttyS0" -v --os-variant debian9 \
  --disk size=10 --memory 1000
 --extra-args "console=ttyS0": active la consola
```

Gestión de redes con libvirt: NAT

- 🏢 IPs dinámicas para las MV dentro de una red privada virtual
- 🏢 Misma IP pública, limitación: conexiones entrantes a las MV
- 🏢 La red se define mediante un fichero xml:

```
<network>
  <name>default</name>
  <bridge name="virbr0"/>
  <forward mode="nat"/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254"/>
      <host mac="52:54:00:6f:78:f3" ip="192.168.122.222"/>
      <host mac="52:54:00:91:ed:23" ip="192.168.122.233"/>
    </dhcp>
  </ip>
</network>
```



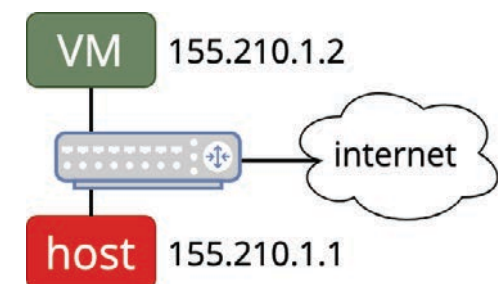
- 🏢 Luego podemos activar la nueva red mediante:
virsh net-define <fichero_xml>
virsh net-start default
virsh net-autostart default

Gestión de redes con libvirt: **bridged mode**










- Se comparte un interfaz ethernet real con las MV
- Cada MV puede estar asociada a una dirección IP como si fuera una máquina física
- Limitación, es necesario disponer de una IP por MV
- En debian, necesario instalar el paquete bridge-utils
- Primero hay que crear el bridge en el host y luego activarlo para la MV

```
# virsh edit <nombre de la MV>
```

```
<interface type="bridge">  
  <source bridge="br0"/>  
  <mac address="52:54:00:4f:47:f2"/>  
</interface>
```




Algunos comandos útiles con MVs

-  `virsh` es el comando principal para interactuar con MV
-  Encendido de una MV: `# virsh start <MV>`
-  Apagado de una MV: `# virsh shutdown <MV>`
-  Apagado forzado de una MV: `# virsh destroy <MV>`
-  Suspensión de una MV: `# virsh suspend <MV>`
-  Listado de MVs: `# virsh list`
-  Acceso a la MV: `# virsh console <MV>`
-  Manipulación de dispositivos: `# virsh attach-device|attach-disk <VM> <...>`
-  Clonación de una MV: `# virt-clone --original demo - -auto-clone`

Ajuste de parámetros de MV

 Cómo editar la configuración de una MV: # virtsh edit <MV>

 Ejemplo número de CPUs virtuales:

```
<cputune>
```

```
<vcpupin vcpu='0' cpuset='0' />
```






```
<vcpupin vcpu='1' cpuset='4' />
```

```
</cputune>
```

 vcpuin es la CPUs virtual y cpuset es la CPU física asignada

 Tambien se puede ajustar el disco (cuello de botella habitual), memoria, red...

Notas sobre seguridad en Máquinas Virtuales

-  Comprometer el hipervisor puede suponer tener acceso a todas las MV, con máquinas físicas no ocurre
-  Es crítico garantizar la seguridad del anfitrión
-  Activar el control de acceso a libvirt para minimizar los privilegios de los clientes al conectarse a libvirt
-  Emplear módulos de seguridad como SELinux junto con sVirt para asegurar el aislamiento de las MV y controlar la compartición de datos, red, ...
-  Emplear canales de comunicación seguros con TLS o SSL junto con cortafuegos



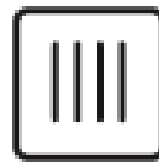
Contenedores

Objetivo de los contenedores

¿Son las máquinas virtuales la mejor manera de compartir, empaquetar y ejecutar aplicaciones?



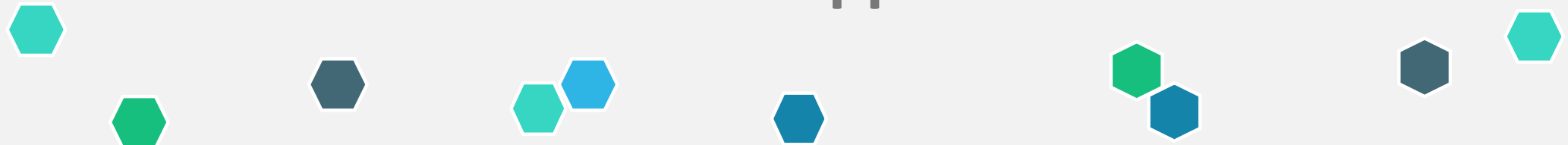
Build



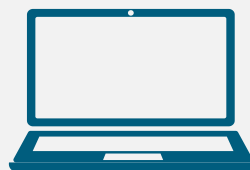
Ship

Run

Distributed Applications

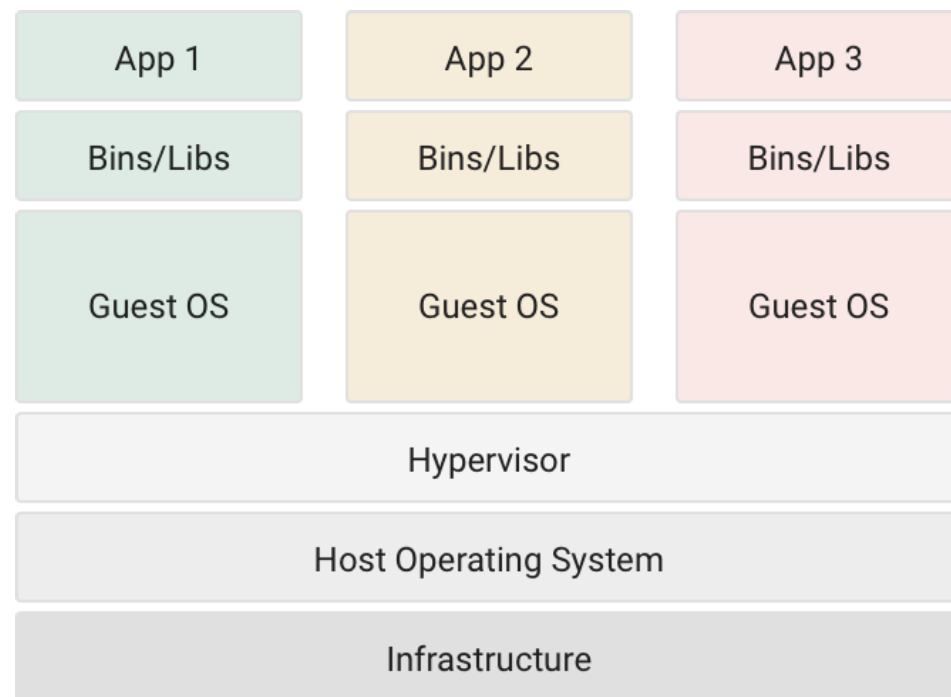


Anywhere

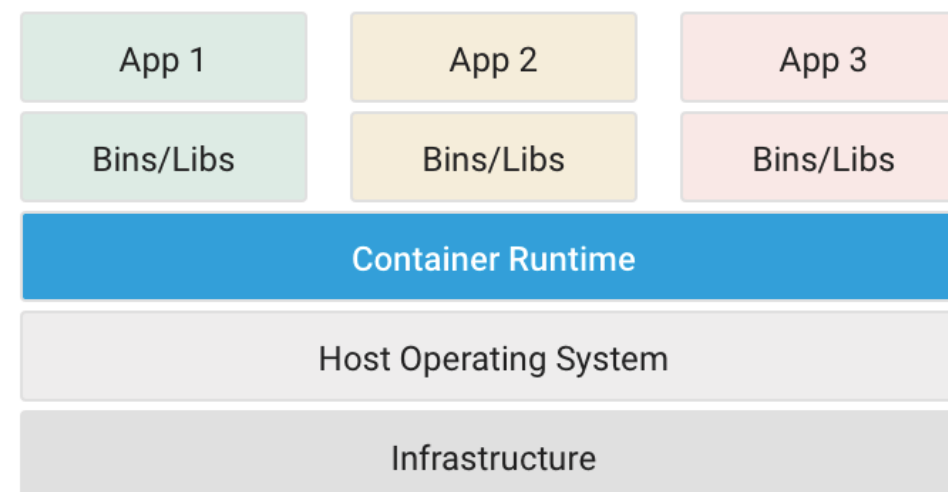


fuelle: <https://github.com/docker/labs/blob/master/slides/docker-introduction.key>

Máquinas virtuales vs Contenedores



Virtual Machines



Containers

¿Puedes explicar las diferencias entre ambas figuras?







MV: virtualización a nivel de sistema, contenedor: virtualización a nivel de sistema operativo






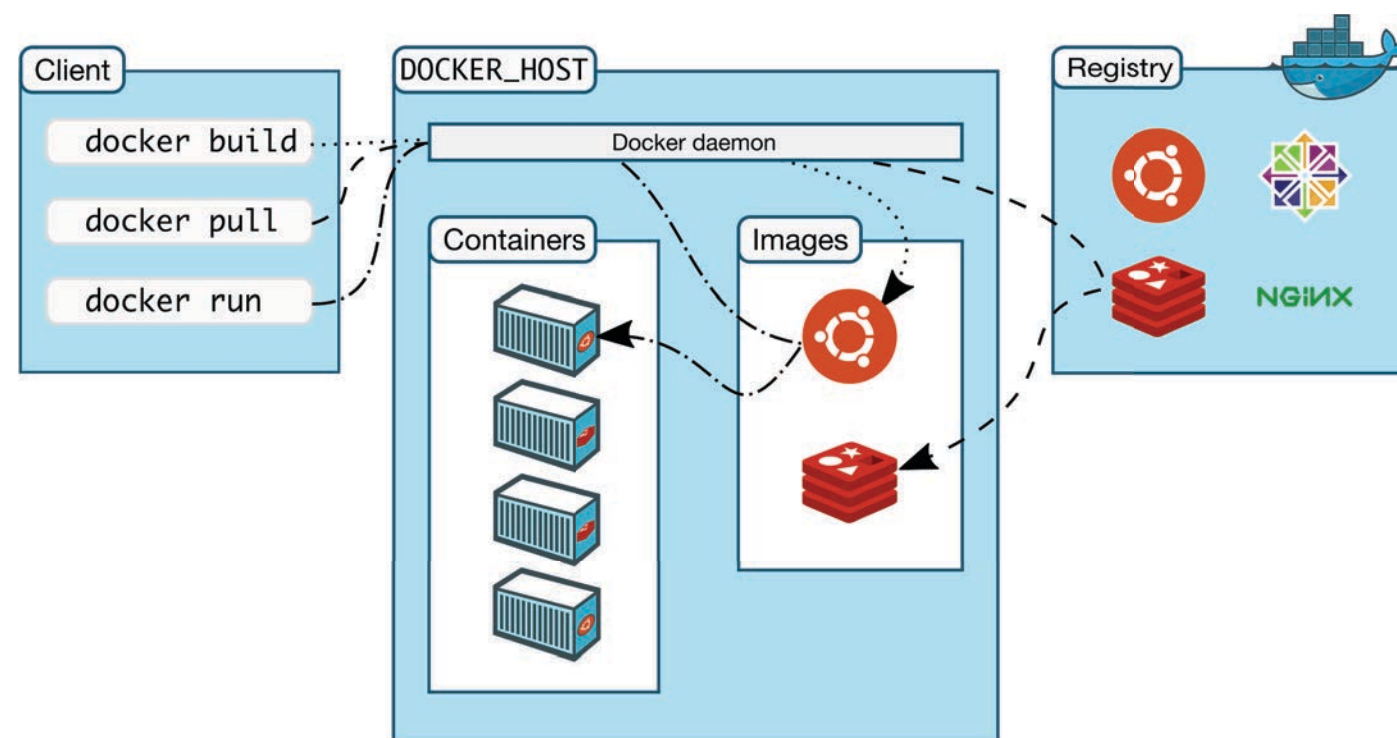
fuelle: <https://cloud.google.com/containers>

Soporte del núcleo de SO para contenedores

-  **Namespaces:** Proveen el aislamiento de los contenedores hacia los sistemas de ficheros (puntos de montaje), gestión de procesos y redes. Por ejemplo, cada contenedor puede ver una parte distinta del sistema de ficheros (similar a chroot, pero más potente)
-  **Control groups (cgroups):** limita, audita, y aísla el uso de recursos (CPU, disco, memoria, red) para colecciones de procesos
-  **Capabilities:** permiten especificar los permisos a la hora de realizar operaciones sensibles y llamadas al sistema. Los UNIX tradicionales solo distinguen 2 categorías de procesos: privilegiados (UID = 0) y noprivilegiados. Las capabilities extienden el número de categorías (control de grano fino). Ejemplos: CAP_CHOWN, CAP_SETUID, ...
-  **Secure computing mode (seccomp):** Restringe el acceso a las llamadas al sistema. Una granularidad más fina que las capabilities. Por ejemplo SECCOMP_SET_MODE_STRICT únicamente permite a un hilo ejecutar las llamadas: read, write, _exit y sigreturn...

Elementos principales en contenedores (Docker)

-  **Contenedor:** Grupo aislado de procesos con acceso restringido al sistema de ficheros y a los recursos del sistema
-  **Imagen:** Plantilla de solo lectura con instrucciones para crear un contenedor. Puede depender de otra imagen
-  **Registro:** Almacén desde donde puede descargarse imágenes



Fuente: <https://docs.docker.com/engine/images/architecture.svg>

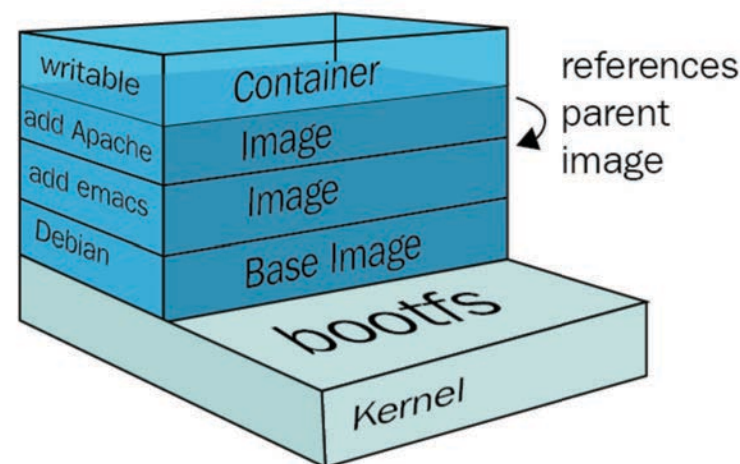
Ejemplo de ejecución de un contenedor (debian 10)

- 🏢 Instalación de docker, revisar instrucciones de <https://docs.docker.com/install/linux/docker-ce/debian/>
- 🏢 Descarga de una imagen (Alpine Linux): # docker pull alpine
- 🏢 Visualización de las imágenes en disco: # docker images
as@as:~\$ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	a187dde48cd2	11 days ago	5.6MB
- 🏢 Ejecución de un shell interactivo : # docker run -it alpine /bin/sh
- 🏢 Los shells no interactivos terminan inmediatamente después de ser ejecutados

Sistema de ficheros en un contenedor

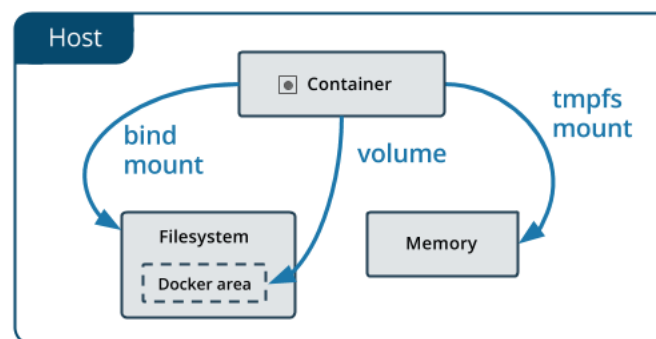
- Una imagen es un conjunto de capas (*layers*) de solo lectura donde cada una representa una instrucción del fichero generador de la imagen (Dockerfile)
- El sistema de ficheros, visto por el contenedor, está compuesto por las capas de la imagen y una capa de lectura/escritura mediante *copy-on-write*
- Cada capa (*layer*) es una colección de cambios sobre ficheros
- Distintos contenedores pueden compartir las mismas imágenes base inmutables



Fuente: Denis Zuev, Artemii Kropachev, Aleksey Usov, Learn OpenShift, Julio 2018

Gestión de datos con Docker


- Por defecto, todos los ficheros creados por un contenedor se almacenan en su capa de escritura:
 - La capa de escritura no es persistente, los datos se pierden al apagar
 - Es difícil mover los datos de la capa de escritura
- En Linux, 3 alternativas para disponer de ficheros persistentes: volúmenes, bind mounts y tmpfs
- Los volúmenes se almacenan en un directorio del host gestionado por docker, `/var/lib/docker/volumes/`. No deben ser modificados por procesos no-docker
- Los bind mounts se almacenan en cualquier directorio del host. Modificables por cualquiera en cualquier momento
- Tmpfs se almacenan en la memoria del host y nunca se escriben




fuelle: <https://docs.docker.com/storage/>

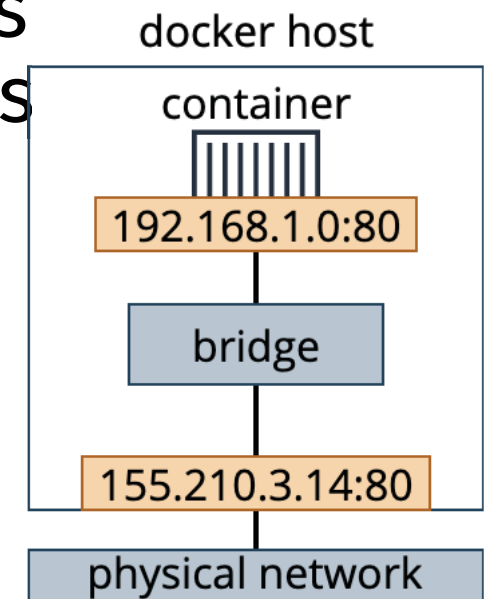
Gestión de redes

 Docker emplea un modelo modular con varios drivers de red, modos más empleados: bridge y host

 **Bridge** (defecto): Los contenedores disponen de redes privadas para comunicarse exclusivamente entre ellos (permite mapear puertos al exterior)

 **Host:** El contenedor emplea la pila de red del host directamente (no hay aislamiento)




 Se pueden publicar puertos del contenedor para que sea accesible desde el exterior, por ejemplo para un servidor web:



```
# docker run -p 80:80 --hostname nginx --name nginx -d nginx
```













 `<host_port>:<container_port>`

Creación de una imagen con fichero Dockerfile










-  Un fichero Dockerfile es una receta para construir una imagen. Cada paso es una instrucción y/o comandos Shell
-  Las imágenes se generan con: `# docker build`
-  Ejemplo, recortado, de Dockerfile para el servidor web NGINX

```
FROM debian:jessie
MAINTAINER unizar "as@unizar.es"
ENV NGINX_VERSION 1.10.3-1~jessie
RUN apt-get update && apt-get install -y ca-
certificates \
nginx=${NGINX_VERSION} && rm -rf /var/lib/apt/lists/*
# redireccionar las salidas a un fichero
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log
EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"]
```


Comandos principales de Docker

-  Información sobre el demonio: `# docker info`
-  Ejecución de un nuevo contenedor: `# docker run <COMMAND>`
-  Lanzamiento/Parada de un contenedor: `# docker start/stop`
-  Visualización de contenedores en ejecución: `# docker ps`
-  Borrado de un contenedor: `# docker rm <CR>`
-  Visualización procesos contenedor: `# docker top <CR>`
-  Descarga/Carga imagen en registro remoto: `# docker pull/push <IMG/REPO>`
-  Borrado de una imagen: `# docker rmi `
-  Etiquetado de una imagen: `# docker tag ...`
-  Listado imágenes locales: `# docker images`
-  Visualización de la salida estándar de un contenedor: `# docker logs <CR>`
-  Listado de redes: `# docker network ls`


Algunas buenas prácticas de seguridad en contenedores


-  Las MV y los contenedores pueden desplegarse juntos para aumentar el aislamiento y la seguridad
-  Mantener actualizado el anfitrión y el gestor de contenedores (docker)
-  Nunca permitir acceso al socket de docker, `/var/run/docker.sock`
-  Emplear un usuario no privilegiado en el contenedor
-  Limitar las *capabilities* que necesite el contenedor
-  Emplear `--security-opt=no-new-privileges` para prevenir escalada de privilegios
-  Desactivar la comunicación entre contenedores
-  Emplear módulos de seguridad (seccomp, AppArmor o SELinux)
-  Para más información consultar <https://github.com/OWASP/Docker-Security>

Comparación de Máquinas Virtuales y Contenedores

	Máquinas Virtuales	Contenedores
Resumen	Múltiples SO compartiendo recursos hardware vía Hypervisor	Grupos de procesos aislados gestionados por el mismo núcleo de SO
Arranque	completo y lento (1-2 minutos)	Rápido, proceso listo en < 1 sec
Tiempo de vida	grande	pequeño
Almacenamiento	uno o varios discos virtuales	sistema de ficheros esta definido por el gestor de contenedores
Tamaño imagen	GB	MB
Nº por anfitrión	Decenas por servidor físico	Muchos por servidor
Aislamiento	Completo entre VM	SO núcleo y servicios compartidos con anfitrión
Migración	Sistema Completo (procesos, ficheros, ...)	Sólo ficheros
ISA diferente	Si	No

Referencias

 Capítulos 24 y 25, Unix and Linux System Administration Handbook, Nemeth *et al.*, 5th Ed.

 Apéndice “Virtual Machines”, Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, <http://pages.cs.wisc.edu/~remzi/OSTEP/vmm-intro.pdf>

 Jamie Nguyen, Libvirt Networking Handbook, <https://jamielinux.com/docs/libvirt-networking-handbook/>

 Documentación de docker, <https://docs.docker.com>