Virtualización, contenedores, cloud y TEE

Diego Fernandez Slezak[1]

[1]Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, 2do cuatrimestre de 2025

(2) Virtualización

- Definición: es la posibilidad de que un conjunto de recursos físicos se vean como varias copias de recursos lógicos.
- La acepción más común es pensar en una computadora realizando el trabajo de varias.

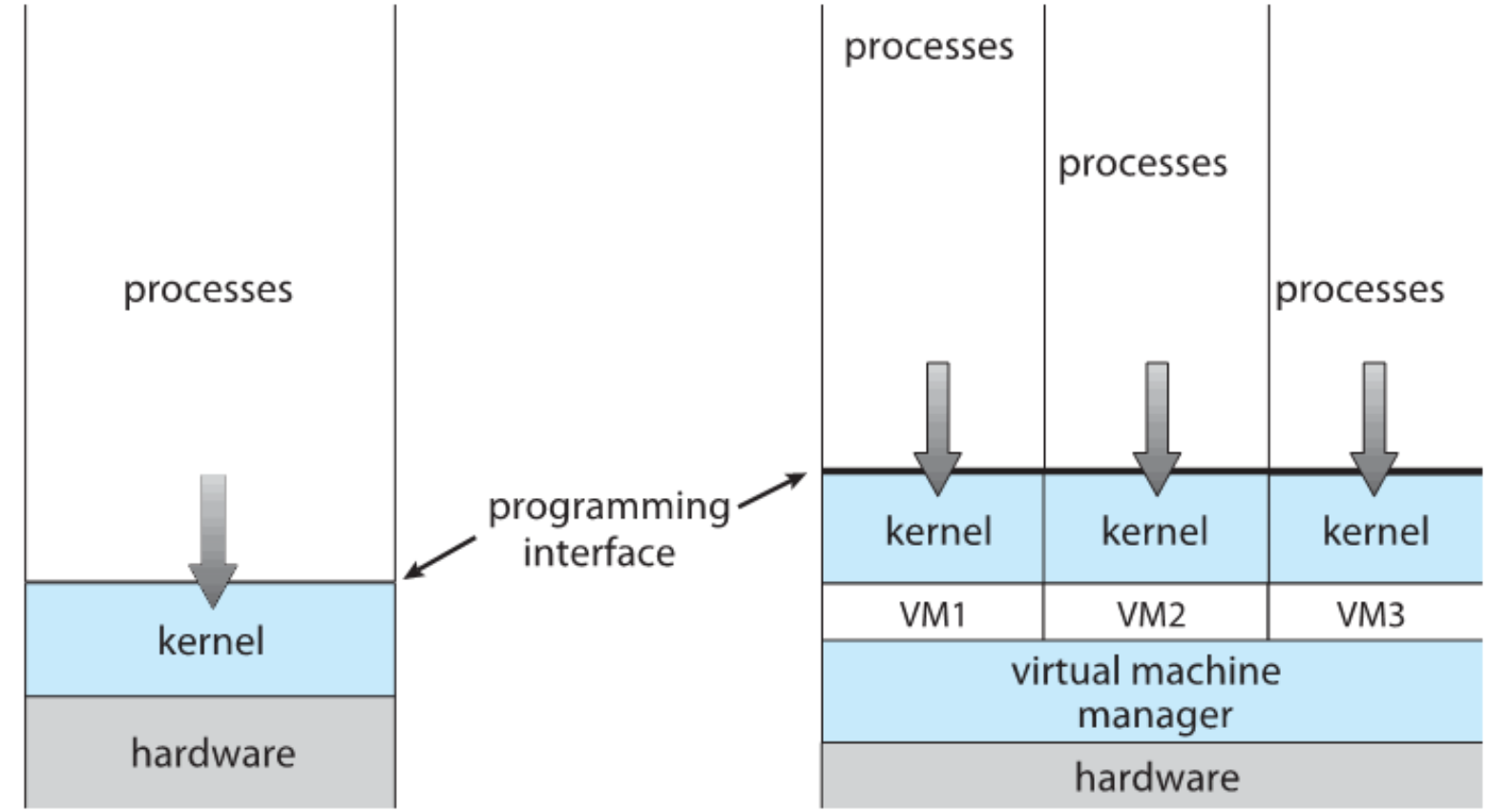
(3) Algo de historia

- Desde hace rato (por lo menos 1960) resulta tentadora la idea de tener *máquinas virtuales*.
- Ie, de mentira.
- Los objetivos son variados:
  - Portabilidad (à la Java Virtual Machine).
  - Simulación/testing.
  - Aislamiento (como chroot, jail y cía).
  - Particionamiento de HW.
  - Agrupamiento de funciones ("*consolidation*").
  - Protección ante fallas de HW (migración de HW).
  - Migración entre HW sin pérdida de servicio.

(4) Virtualización

- Concepto de VMM/Hypervisor
- Características esenciales de los VMM (Popek y Goldberg, 1974)
- Fidelidad
- Performance
- Safety

(5) Virtualización



Non-virtual machine

Virtual machine

(6) Simulación y emulación

- Una forma posible de lograr esto es mediante la *simulación*:
  - En el sistema *anfitrión* se construye una variable de estado artificial que representa al sistema *huésped*.
  - Se lee cada instrucción y se modifica el estado como si ésta se ejecutase realmente.
- Sin embargo:
  - El mecanismo puede ser muy lento.
  - ¿Cómo se simulan las interrupciones, DMA, concurrencia, etc.?

(7) Simulación y emulación (cont.)

- Otra forma es mediante la *emulación de HW*:
  - Acá el sistema emulado se ejecuta realmente en la CPU del anfitrión.
  - Se emulan componentes de HW.
  - le, cuando la máquina virtual cree que está haciendo E/S de un dispositivo, en realidad lo está haciendo contra el controlador de máquina virtuales.
  - Éste, a su vez, hace de proxy contra el dispositivo real o la emulación que se esté usando.
  - El grueso del código se corre mediante *traducción binaria*.
- Problemas:
  - ¿Cómo logro separación de privilegios? Toda la máquina virtual corre en modo usuario.
  - ¿Qué pasa con la velocidad de acceso a los dispositivos?

(8) Virtualización asistida por HW

- En este contexto nace la intención de lograr virtualización asistida por HW, especialmente para lograr evitar los siguientes problemas (retengo los nombres de Intel):
  - Ring aliasing: tengo programas escritos para modo kernel, pero en realidad se están ejecutando en modo usuario. Puedo tener problemas de permisos para ejecutar ciertas instrucciones.
  - Address-space compression: ¿cómo hago para que la máquina virtual no pueda pisar memoria del propio emulador? Recordemos que desde el punto de vista del anfitrión son un único proceso.
  - Non-faulting access to privileged state: algunas instrucciones privilegiadas generan un trap cuando se ejecutan sin permiso. Eso es bueno porque puedo atrapar el trap y simularlas. Pero otras no. ¿Cómo hago?
  - Interrupt virtualization: hay que simularle las interrupciones al SO huésped.

(9) Virtualización asistida por HW (cont.)

- seguimos con los problemas...
  - Access to hidden state: hay parte del estado del procesador que no es consultable por software.
  - Ring compression: como tanto el kernel huésped como sus programas corren en realidad en el mismo nivel de privilegio, no hay protección entre kernel y programas de usuario.
  - Frequent access to privileged resources: si bien el controlador de máquinas virtuales puede bloquear el acceso a ciertos recursos, haciendo que se genere un trap, esto puede ser un cuello de botella para recursos accedidos frecuentemente.

(10) Virtualización asistida por HW (cont.)

- Para solucionar estos problemas los fabricantes agregaron soporte para la virtualización en el HW.
- En el caso de Intel, agregaron al procesador las extensiones *VT-x*, que proveen dos modos:
  - VMX root: Las instrucciones se comportan de manera similar, pero hay algunas extensiones (anfitrión).
  - VMX non-root: El mismo set de instrucciones pero con comportamiento restringido (huésped).
- Se proveen (10) instrucciones para alternar fácilmente entre ambos modos.

(11) Virtualización asistida por HW (cont.)

- Se agrega la *Virtual Machine Control Structure* (en memoria).
  - Campos de control: indican qué interrupciones recibe el huésped, qué puertos de E/S, etc.
  - Estado completo del huésped.
  - Estado completo del anfitrión.
- La idea es que el HW sale automáticamente de modo VMX non-root cuando el huésped realiza alguna acción que está "prohibida" de acuerdo a la VMCS.
- En ese momento, el controlador de la máquina virtual recibe el control y emula, ignora o termina la acción "prohibida".

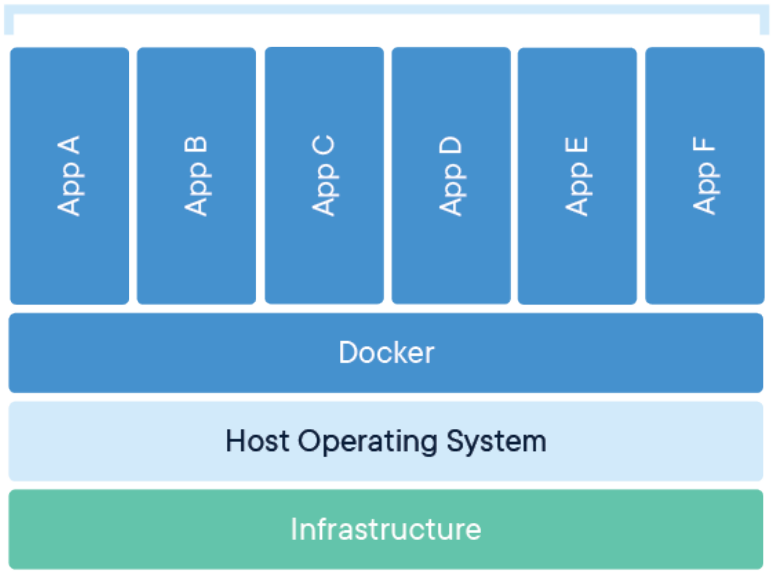
(12) Desafíos/problemas

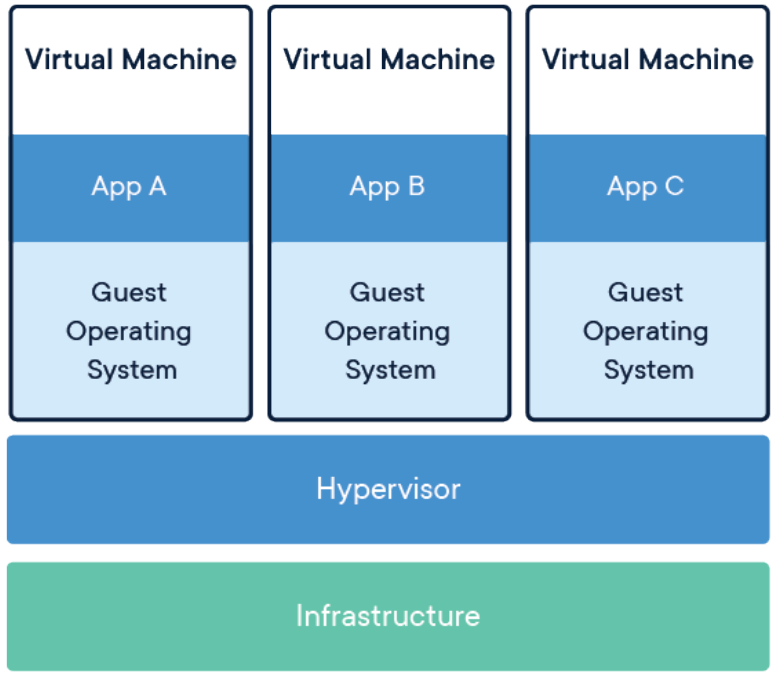
- ¿Qué pasa con las optimizaciones que tenían el kernel y el FS para acceder al disco de manera eficiente?
- ¿Y con picos de carga en más de una CPU?
- Único punto de falla: falla una pieza de HW real, caen varias máquinas virtuales.
- Pero también al revés.

(13) Escenarios de uso de la virtualización

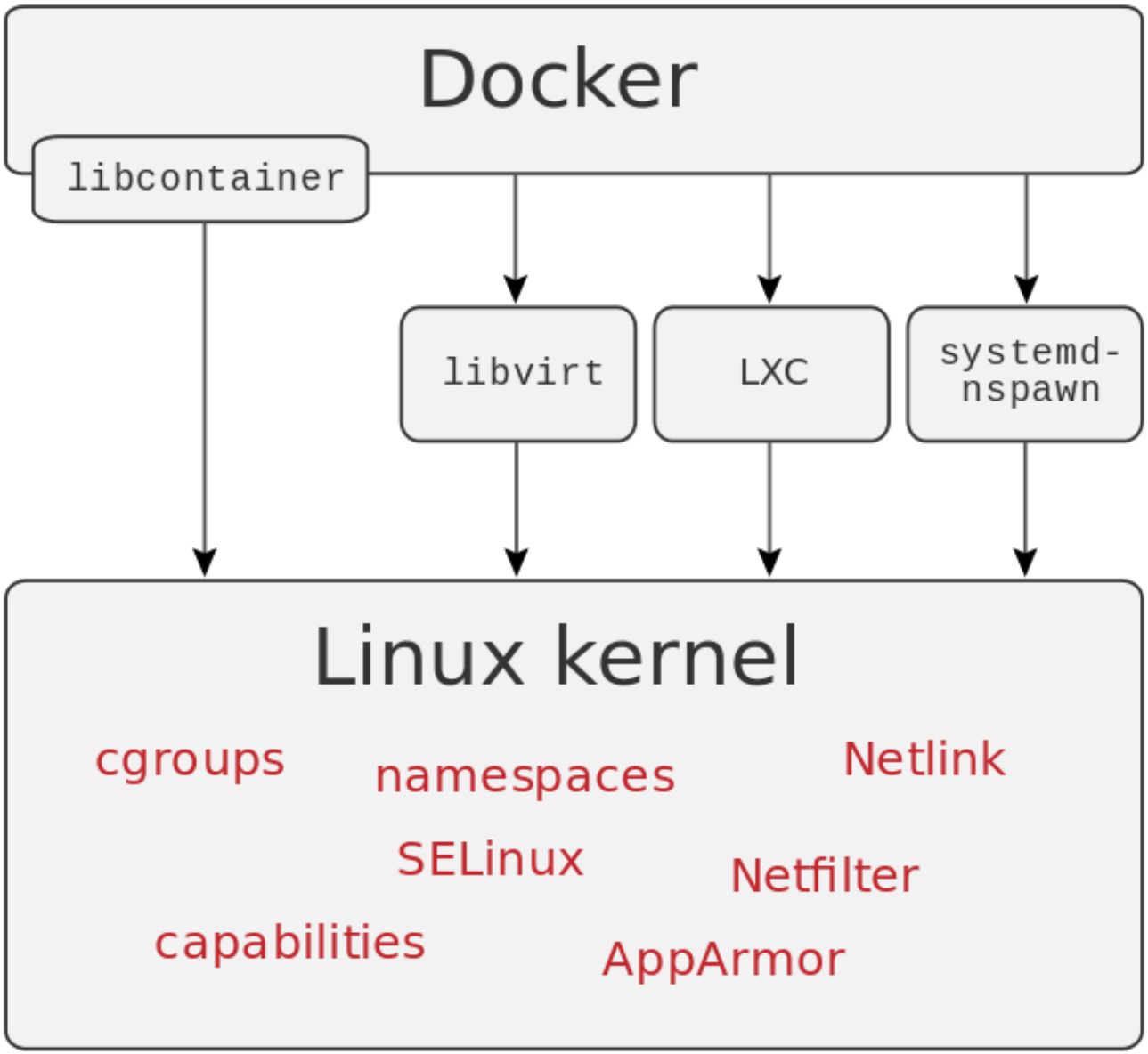
- Correr sistemas viejos.
- Aprovechamiento de equipamiento
- Desarrollo/testing/debugging.
- ¿Abaratar costos?

(14) Contenedores





(15) Contenedores



(16) ¿Qué hace posible a los contenedores en Linux?

- Los contenedores no son una virtualización completa.
- Se apoyan en funciones clave del kernel de Linux.
- Aislamiento, limitación de recursos y sistemas de archivos eficientes.
- Componentes fundamentales:
  - Namespaces
  - Cgroups
  - Layered FS

(17) Namespaces: Aislamiento

- Proveen aislamiento entre procesos.
- Tipos de namespaces:
  - **PID**: espacio de procesos
  - **NET**: red (interfaces, rutas, iptables)
  - **MNT**: sistema de archivos
  - **UTS**: nombre del sistema
  - **IPC**: distintos mecanismos de IPC
  - **USER**: privilegios y UID/GID
  - **TIME**: manejo del clock
- Cada contenedor ve una "realidad" distinta del sistema.

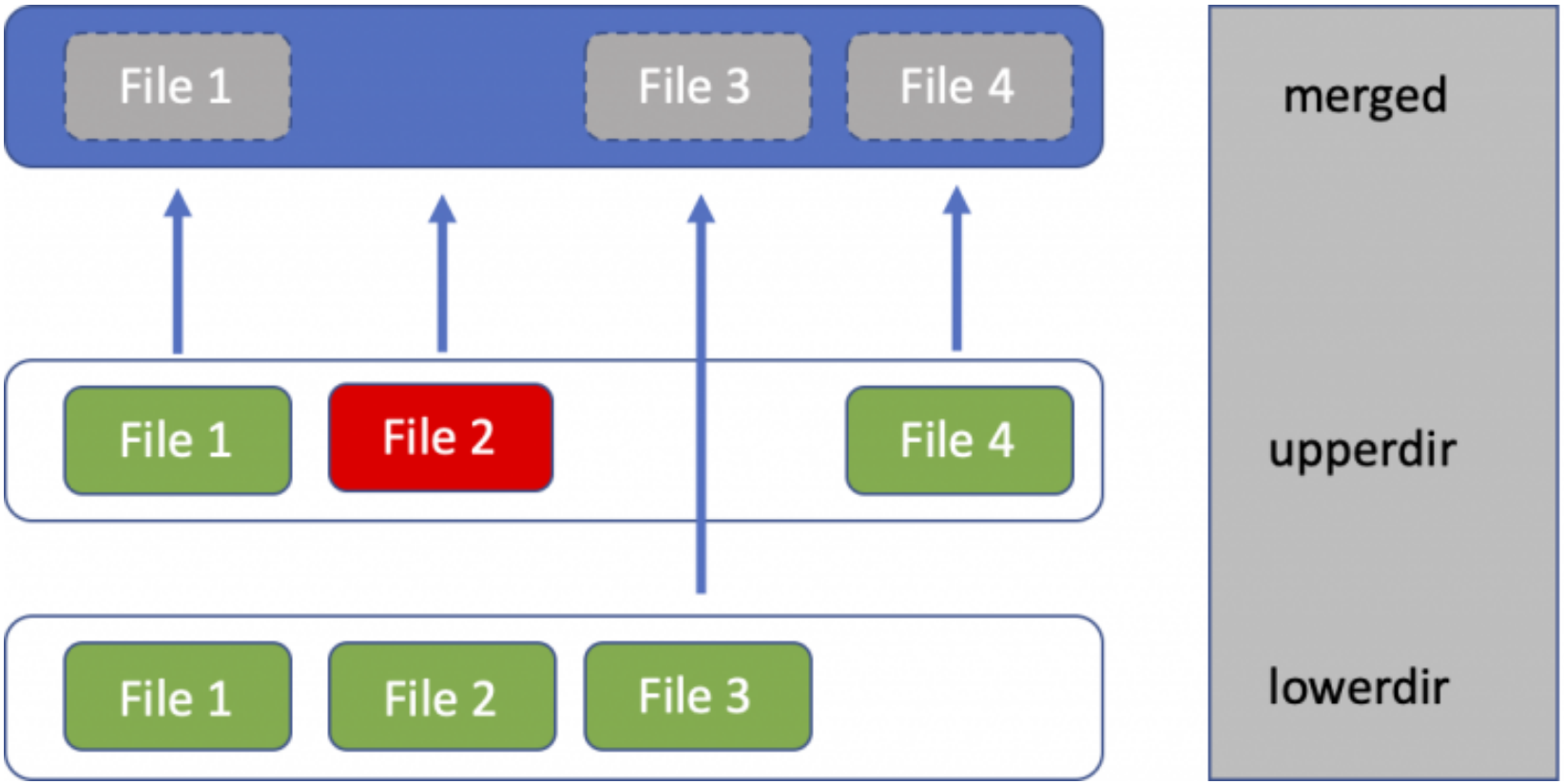
(18) Cgroups: Control de Recursos

- Control groups permiten limitar y contabilizar recursos:
  - CPU
  - Memoria
  - E/S de disco
  - Red (parcialmente)
- Evita que un contenedor consuma todo el sistema.
- Versión actual: **cgroups v2**.

(19) ¿Qué es un sistema de archivos en capas (Layered FS)?

- Modelo donde se apilan capas de solo lectura y una capa superior de escritura.
- Cada capa representa cambios respecto a la anterior.
- Se usa para mejorar eficiencia, reutilización y almacenamiento.
- Base de tecnologías como OverlayFS, UnionFS y AUFS.

(20) OverlayFS



(21) Uso en la arquitectura de Docker

- Docker utiliza OverlayFS (o UnionFS en versiones antiguas) como backend de almacenamiento.
- Cada imagen se construye en capas: base, dependencias, app, configuraciones.
- Los contenedores escriben en una capa superior temporal.
- Esto permite:
  - Ahorro de espacio.
  - Rápida creación de contenedores.
  - Compartir capas comunes entre contenedores.

(22) Imágenes de Docker y el estándar OCI

- Una imagen de Docker es una plantilla inmutable usada para crear contenedores.
- Contiene todo lo necesario para ejecutar una aplicación: código, dependencias, herramientas, bibliotecas y configuración.
- Las imágenes están compuestas por capas que se apilan sobre una capa base común.

(23) OCI: Open Container Initiative

- Estándar abierto para contenedores, promovido por la Linux Foundation.
- Define dos especificaciones principales:
  - **OCI Image Spec:** Formato estándar de las imágenes.
  - **OCI Runtime Spec:** Comportamiento estándar del runtime (como runc).
- Docker contribuyó con sus especificaciones iniciales a OCI. Más info en: `https://opencontainers.org/`

(24) Otras funciones del kernel

- **Seccomp**: restringe llamadas al sistema.
- **Capabilities**: Permiten asignar a un aplicativo ciertos privilegios usualmente reservados al administrador.
- **AppArmor/SELinux**: control de acceso mandatorio (MAC).
- **Netfilter**: Filtrado de paquetes (fw), nateo
- **Netlink**: comunicación entre el kernel y procesos de usuario

(25) Orquestación de aplicaciones contenerizadas

- Kubernetes: Plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores.
- Openshift (y OKD): usa Kubernetes de base, pero agrega restricciones de seguridad por defecto, interface web mas completa, manejo de roles, facilidades para el desarrollador.

(26) Cloud


PR OVIDER
SCALABILITY
UTILITY
INTERCLOUD
USER
VIRTUALIZA TION
RELIABILITY
DATA
COMPUTING
CLIENT
COMMUNITY
INFRASTRUCTURE
PHONE
SaaS
SECURITY
IaaS
APPLICA TION
CONTENT
CLOUD
MOBILE
DEVICE
TABLET
SERVICE
ONLINE
COST
PaaS
PRIV AT E
PLA TFORM
SLA
PUBLIC
SERVER
HYBRID
REMOTE
API
ARCHITECTURE
CENTER
NETWORK
ENGINEERING
BROWSER
WEB
MONIT ORING
DATABASE
ST ORAGE
IDENTITY

(27) Trusted Execution Environment

- Área protegida dentro de una CPU.
- Ejecuta código y protege datos confidenciales.
- Aísla procesos incluso del sistema operativo y del hipervisor.
- Diseñado para resistir ataques físicos y de software.

(28) Implementaciones en CPUs

- **Intel SGX (Software Guard Extensions)**: enclaves en CPUs Intel.
- **AMD SEV (Secure Encrypted Virtualization)**: cifrado de memoria para VMs.
- **ARM TrustZone**: partición segura del sistema.

(29) Ejemplos en Servidores

- **Microsoft Azure Confidential Computing**
  - Usa Intel SGX y AMD SEV para proteger cargas en la nube.
- **Confidential Containers (CoCo)**
  - Contenedores protegidos con SEV y SGX.
- **IBM Cloud Hyper Protect**
  - Seguridad en el hardware con procesadores IBM Z.

(30) Ejemplos en Teléfonos Móviles

- **Android con TrustZone**
  - Almacén seguro para claves biométricas, PIN y certificados.
- **Apple Secure Enclave**
  - Chip dedicado en iPhones y iPads.
  - Protege Face ID, Touch ID, y claves privadas.
- **Samsung Knox**
  - Entorno seguro basado en TrustZone.

(31) Ventajas

- Alta protección contra accesos no autorizados.
- Aislamiento robusto.
- Mejora la confianza en plataformas en la nube y móviles.

(32) Desafíos

- Complejidad en el desarrollo.
- Posibles vulnerabilidades en el hardware.
- Limitaciones en recursos y memoria del enclave.

(33) Bibliografía

- "Formal Requirements for Virtualizable Third Generation Architectures", de Popek y Goldberg.
- "Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization". `http://www.intel.com/technology/itj/2006/v10i3/1-hardware/1-abstract.htm`
- "A comparison of software and hardware techniques for x86 virtualization", de Adams y Agesen.
- "Linux Containers and Virtualization: A Kernel Perspective", Mohan Jain, Apress, 2020.
- "Trusted Execution Environments", Shepherd y Markantonakis, Springer, 2024.