TACS

Qué es un Linux Container? (LXC)

Tecnología de virtualización en el nivel de sistema operativo (SO) para Linux.

LXC permite que un **servidor** ejecute **múltiples instancias de sistemas operativos** aislados, conocidos como:

- **Servidores Privados Virtuales** (VPS)
- **Entornos Virtuales** (VE)

Qué es un Linux Container? (LXC)

LXC **no** provee de una **máquina virtual**, más bien provee un **entorno virtual que tiene su propio espacio de procesos y redes.**

Qué es Docker?

Docker posibilita **empaquetar** una **aplicación con todas sus dependencias** en una **unidad estandarizada** de desarrollo de software. Incluye:

- código
- runtime
- system tools
- system libraries
- cualquier otra cosa que se pueda instalar en un server

Esto **garantiza** que el software **siempre correrá igual**, independientemente de su ambiente

Conceptos de docker

image

- **image**
- registry
- container

Docker Images

- **Templates read-only** desde los que se crean containers
- Derivados de una **imagen base**(BusyBox, Alpine o Ubuntu)
- Armada por un **conjunto de capas** que combinadas, conforman el filesystem
- **Construida** a partir de un *Dockerfile*
- Cada capa es **hasheada** y **cacheada**

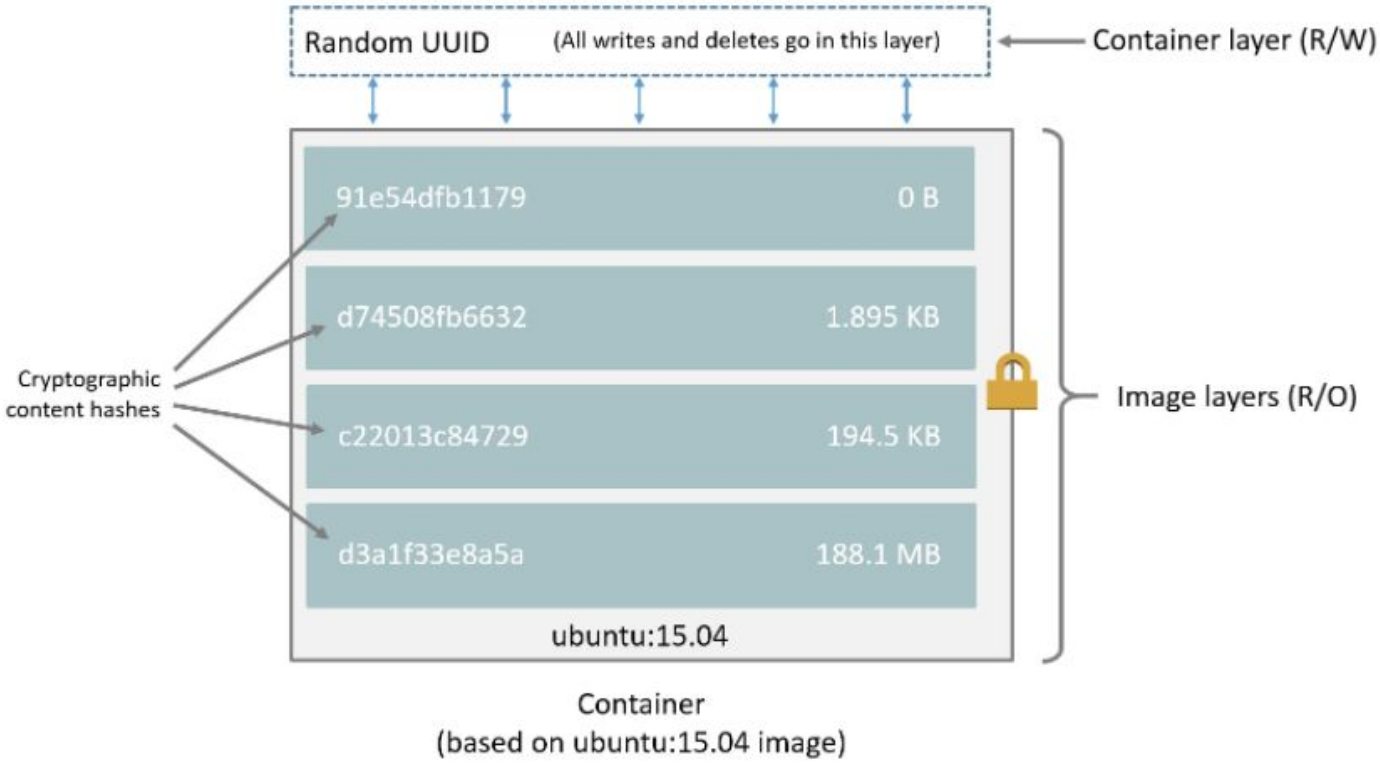
```
FROM ubuntu:16.10

RUN apt-get update && apt-get install -y apache2

ENTRYPOINT [ "apache2ctl" ]
```

Docker Images

Cada **imagen docker** referencia una **lista de capas read-only** que representan deltas de filesystem.

Estas capas son apiladas una sobre otra, sirviendo como base para formar el **sistema de archivos raíz** de un **container**.

Docker File System Layers



Conceptos de docker

registry

- image
- **registry**
- container

Docker registry

- almacena docker images
- docker hub
- private registries
- open source
- similar a git repositories

Conceptos de docker

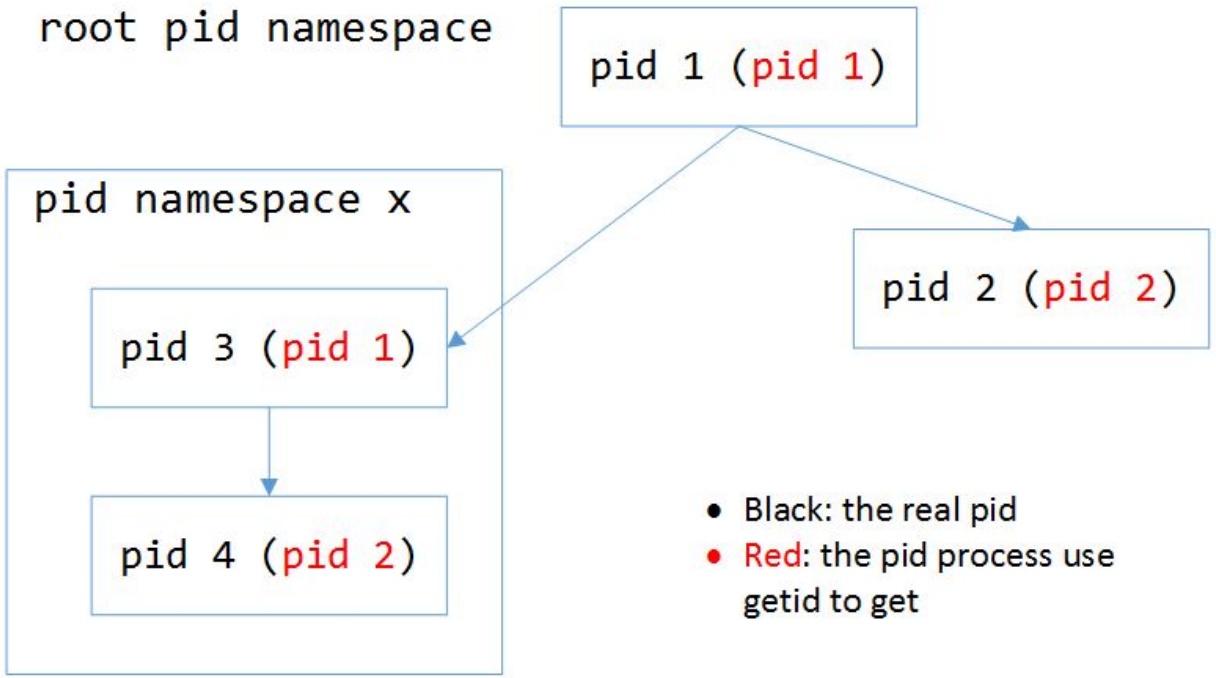
container

- image
- registry
- **container**

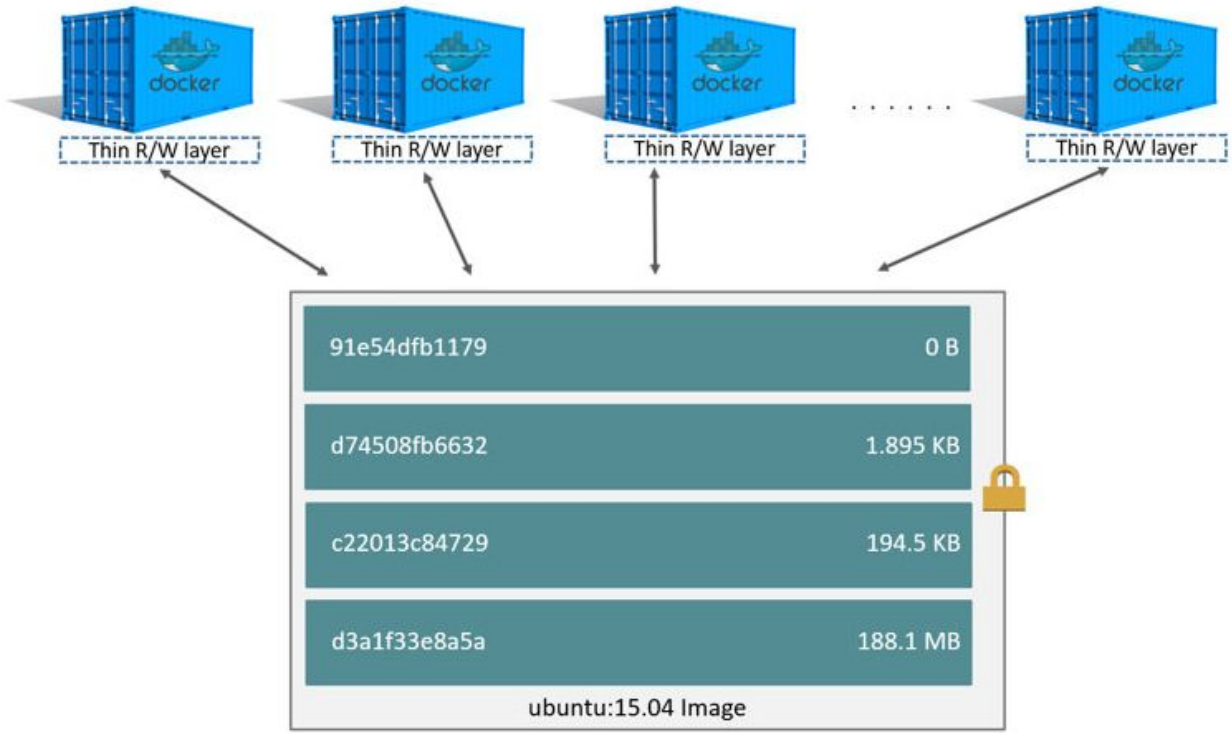
Docker container

- **instancia runtime** de una **docker image**
- **capa r/w** agregada **arriba** de la imagen
- **efímero**
- **aislado** del resto
  - no puede ver/modificar/dañar el host u otros containers
  - **namespaces**, con los que logran tener su vista privada del sistema (network interfaces, árbol PID, mountpoints..)
  - **cgroups**, para tener recursos limitados, y mitigar el efecto ***bad neighbor***

namespaces


root pid namespace
pid 1 (pid 1)
pid namespace x
pid 3 (pid 1)
pid 2 (pid 2)
pid 4 (pid 2)
Black: the real pid
Red: the pid process use getid to get

Docker Containers / FileSystem Layers



Container y layers

- La mayor diferencia es la capa R/W que tiene el container
- Al eliminar un container:
  - Esta capa R/W se elimina
  - La imagen queda sin cambios
- Múltiples containers pueden compartir la misma imagen

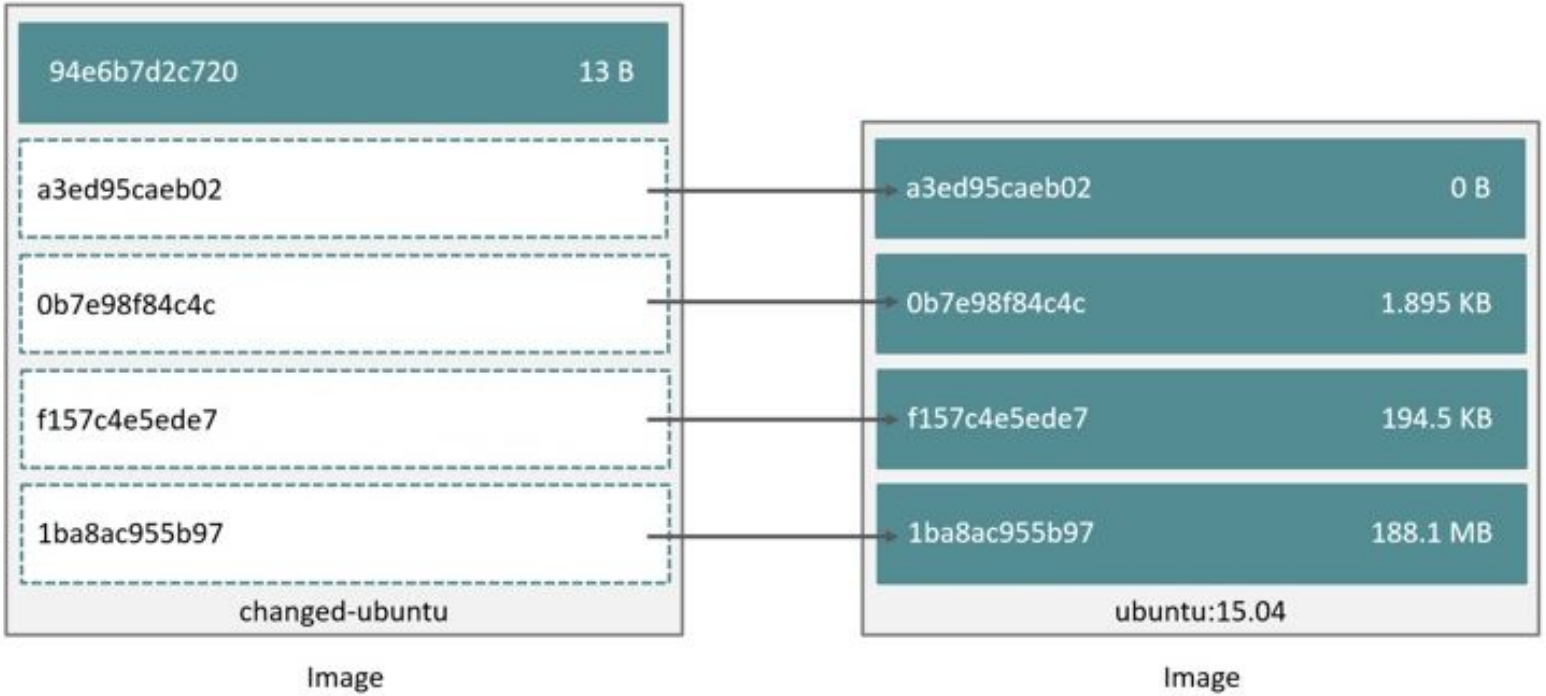
Eficiencia en el manejo de disco

Eficiencia en el manejo de disco

Docker basa su manejo de imágenes y administración de containers en 2 tecnologías:

- Stackable image layers
- copy-on-write

copy-on-write



copy-on-write

Cuando un archivo se modifica en un container:

1. Se **busca el archivo a través de las capas**. El proceso comienza en la capa superior, más nueva, y **va bajando capa por capa** hasta la capa base.
2. Realiza una operación **“copy-up”** con la **primer copia encontrada**. “copy up” copia el archivo a la **capa writable** del container.
3. **Modifica la copia** del archivo en la **capa writable** del container.

copy-up Performance

---

esta operación puede degradar notablemente la performance, especialmente cuando hay:

- grandes archivos
- gran cantidad de capas
- deep directory trees

Afortunadamente, sólo ocurre la primera vez un archivo en particular es modificado.

Espacio ocupado por containers

- Los containers que **escriban gran cantidad** de datos, **consumirán más espacio** que los containers que no.
- Esto es porque la mayoría de las operaciones de escritura consumirán nuevo espacio **en la capa writable** del container.
- Si tenemos que escribir mucho dentro de un container, debemos considerar utilizar un **data volume.**

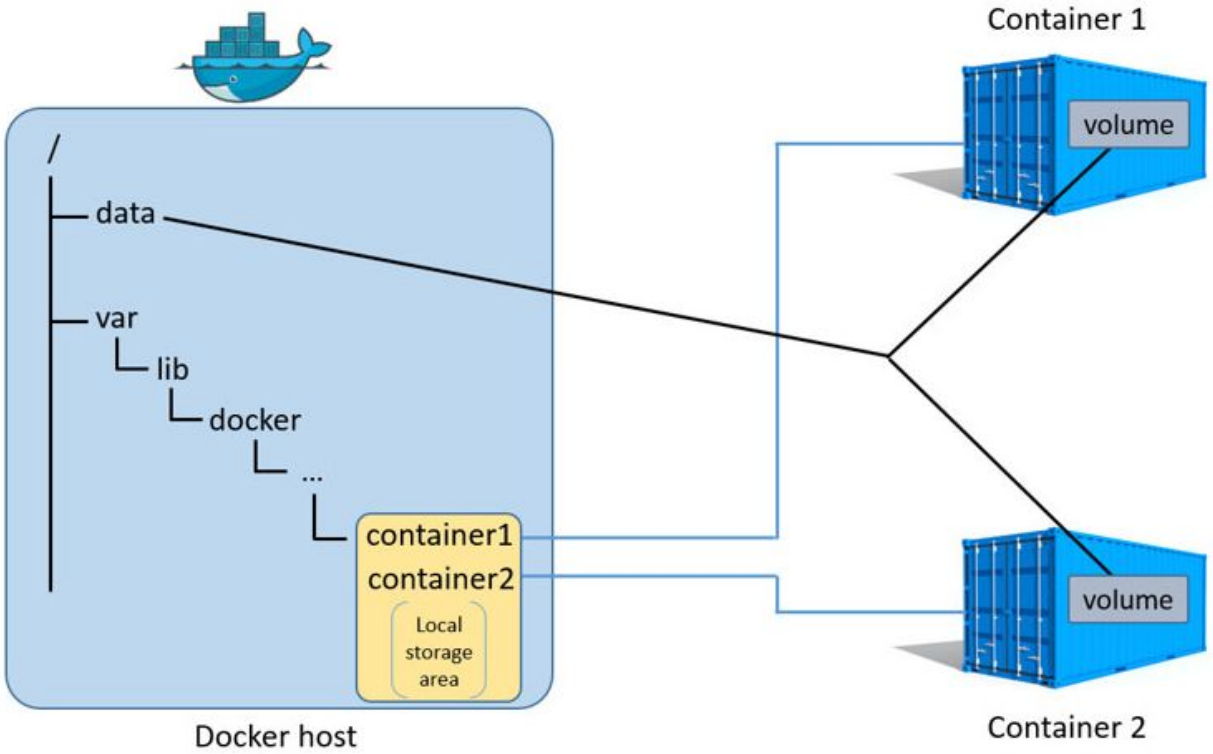
Data Volumes

Cuando se borra un container, cualquier dato escrito en el container, que no es almacenado en un **data volume, es eliminado junto con el container.**

Data Volumes

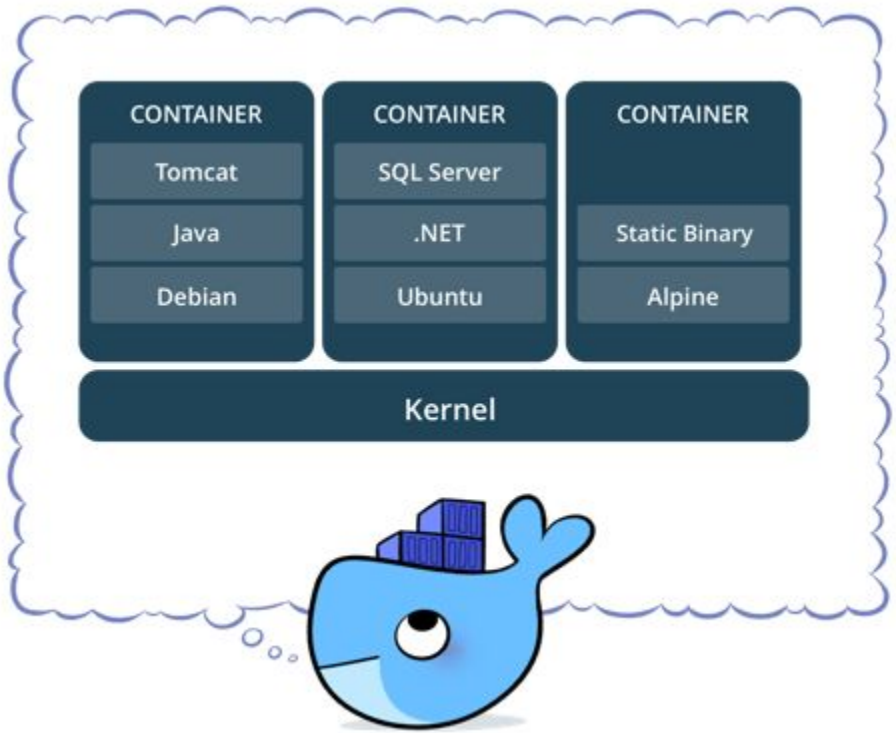
- **Directorio o archivo** en el filesystem del **host**, **montado** directamente dentro de un container.
- **No** son controlados por el **storage driver**
- Operan a la **velocidad nativa** del host
- Se puede montar cualquier número de **data volumes** en un container
- **Multiples containers** pueden **compartir** uno o más **data volumes**

Data Volumes

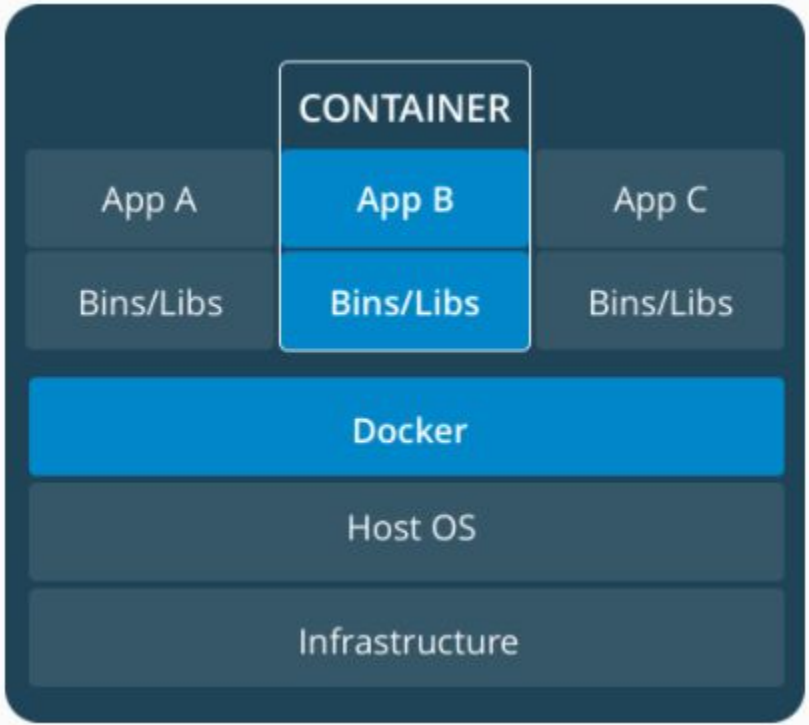


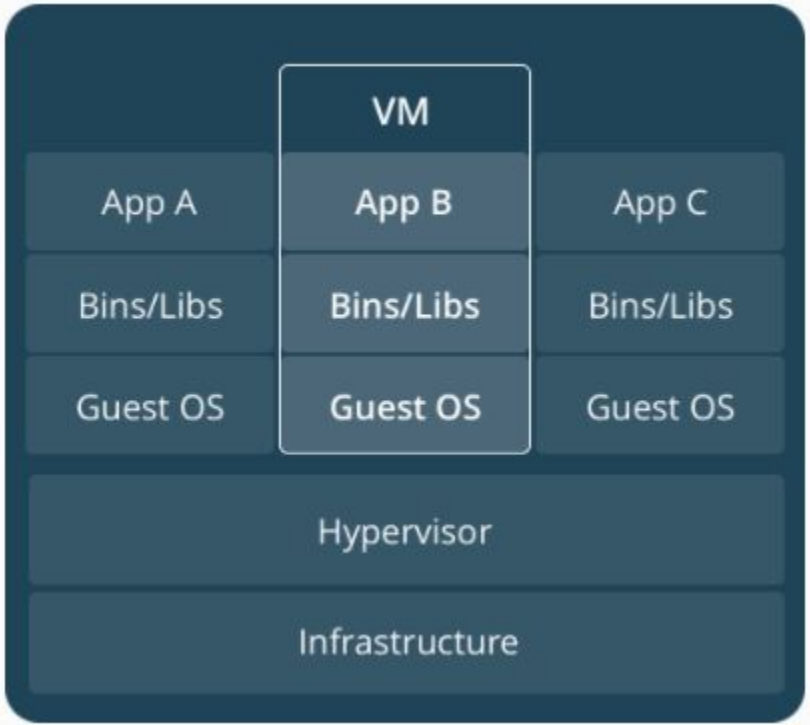
Containers vs VMs

Docker Containers



Containers vs VMs





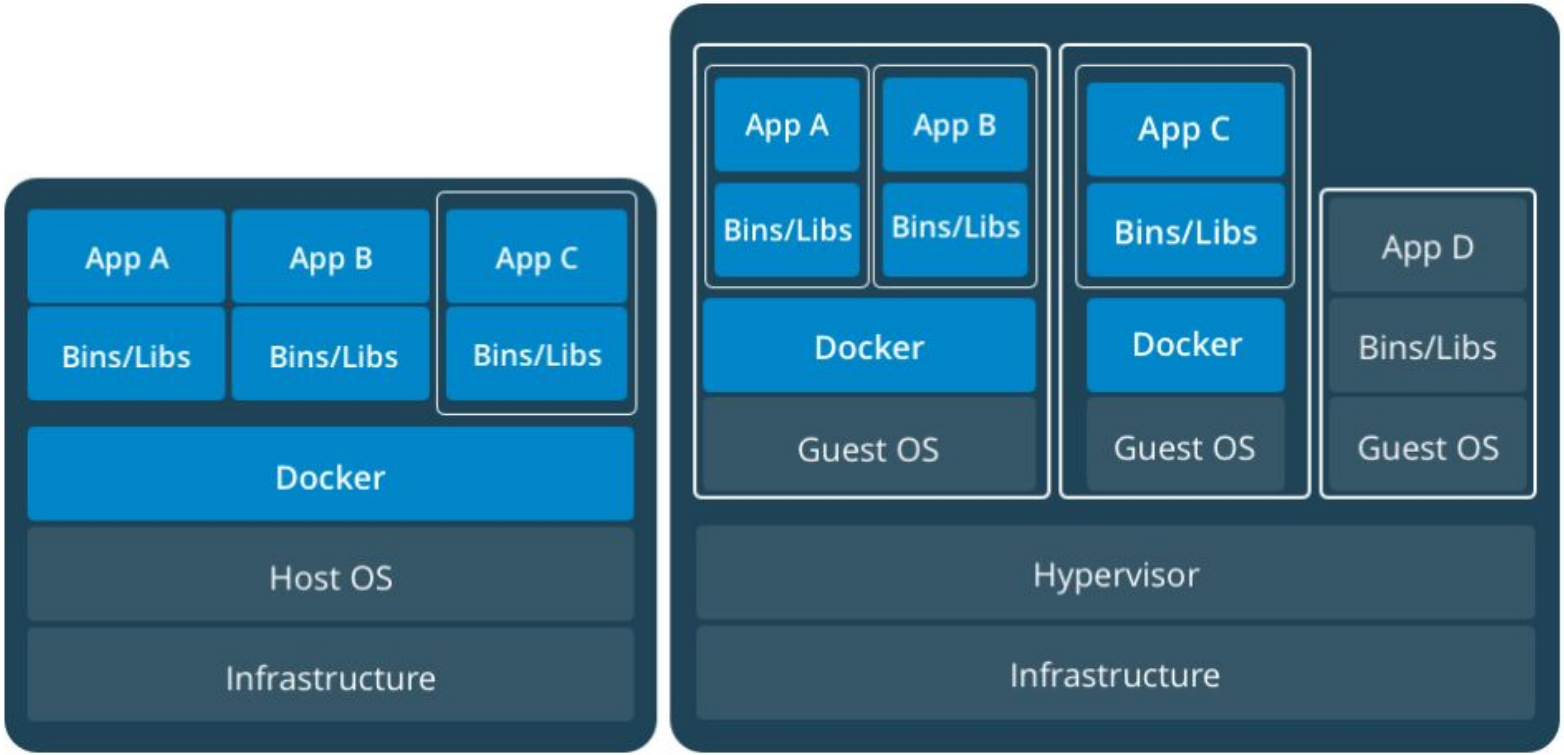
Containers no son VMs

Los Containers nos proveen un **ambiente lo más cercano posible** al que obtendríamos de una **VM**, pero **sin el overhead** que implica correr **kernels** por separado y **simular** todo el **hardware**.

Obtenemos **aislamiento** en

- Process
- network
- filesystem
- resources (cpu, memoria, etc)

Containers Vs VM



Docker life cycle

build

build
deploy
run
monitor

build

- Dockerfile
- cada **comando** es una **nueva capa**
- **capas jerárquicas**: cambiar una capa baja, destruye el cache de las capas superiores
- **download**: todo esto se descargará una y otra vez

Dockerfile - Best Practices

- Containers deben ser efímeros
- Reducir tamaño de las imagenes:
  - Usar un archivo .dockerignore para archivos innecesarios en runtime
  - Evitar instalar paquetes innecesarios
  - Cada container un concern
  - Minimizar la cantidad de **capas**

Docker life cycle

deploy

build
deploy
run
monitor

Deploy

Consideraciones con el **registry**

- **Autenticación**
  - devs
  - c.i. como *jenkins*
  - prod
- **Robustez**
  - el registry se convierte en parte vital del ciclo de vida
  - que pasa si se cae?
- **Versionado**
  - **versionado semántico**
  - **tag latest en producción**

Deploy / Registries

---

registries

- docker hub
- AWS Elastic Container Registry
- registry propio (es opensource)
  - High availability
  - docker engine concurrency problems

Docker life cycle

run

build
deploy
run
monitor

run

- pull vs push
- seguridad
- cómo proveemos la configuración?
  - El mismo artefacto que se testea, es el que corre en producción
- service discovery

Docker life cycle

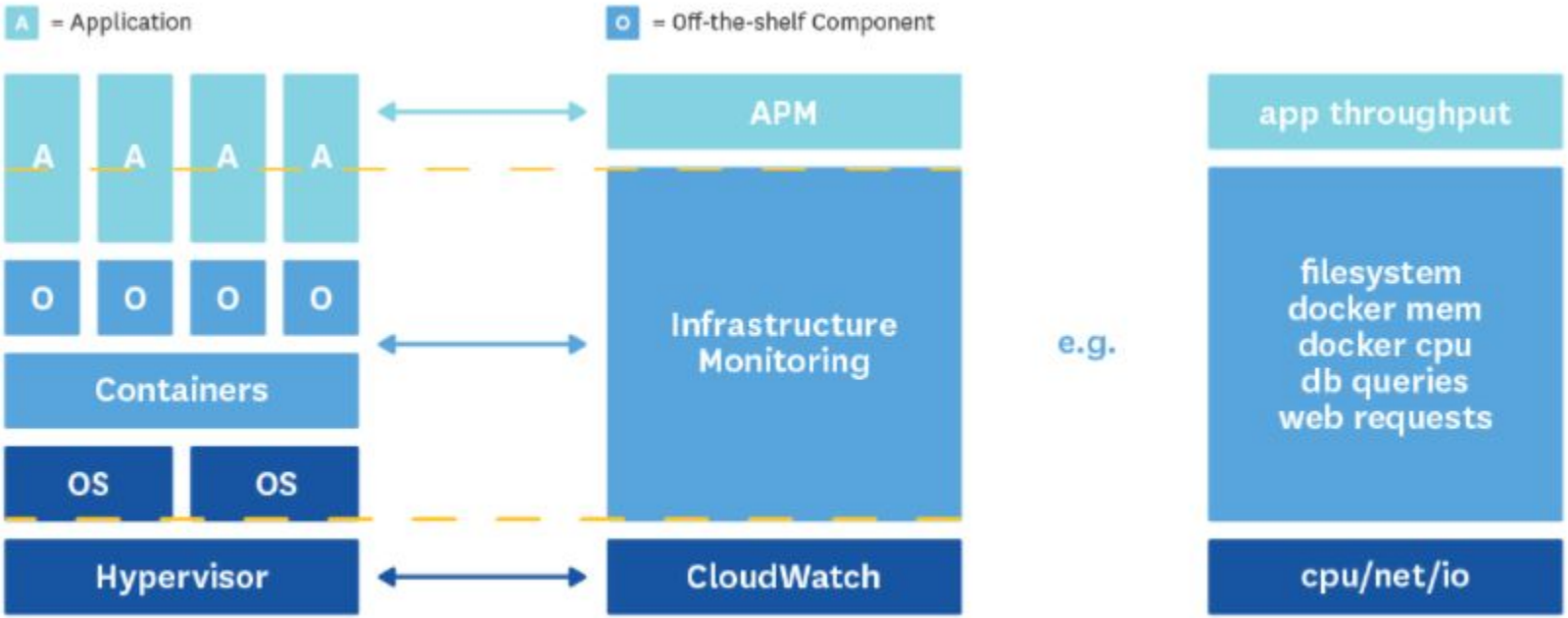
monitor

build
deploy
run
monitor

Monitoreo

- **sin monitoreo** es como **volar sin instrumentos**
- instancias **efímeras**
- **logs**
- abnormal **termination**
- espacio en **disco**
- **métricas** empiezan a tener **significados confusos**: uso cpu
- **monitor daemons**: dentro o fuera de los containers?
- **No gaps**

Monitoreo - No gaps



Monitoreo - No gaps

Debemos poder:

- Ver qué está pasando a través de todas las capas simultáneamente
- Determinar cómo los problemas en una capa afectan a otras

Ej: Poder determinar si **tiempo de respuesta lento** se debe a un **pico de IO** al nivel de la VM

Docker en producción

Docker en producción

En qué:

- VM
- Equipos físicos
- Híbridos

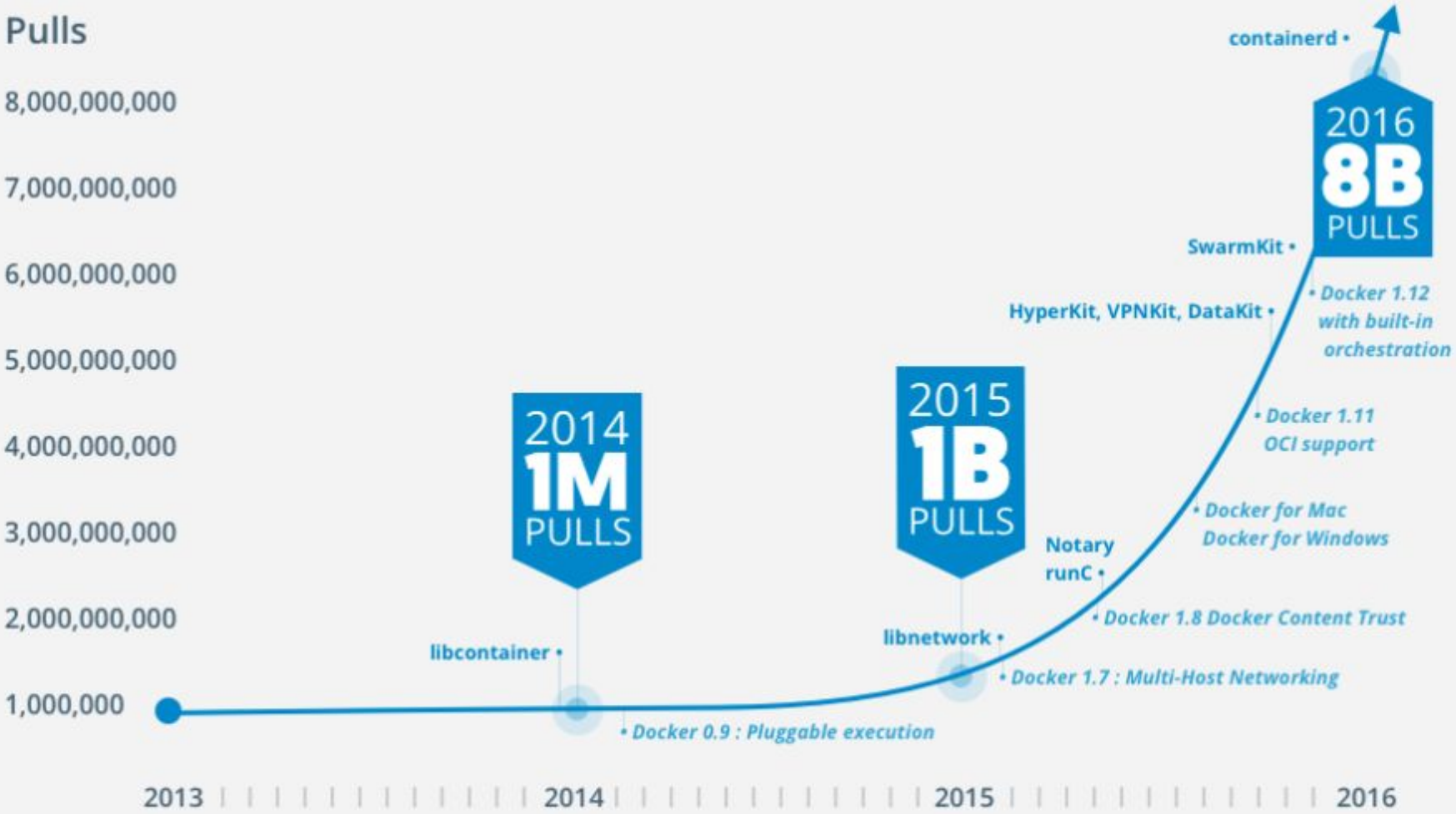
Donde:

- On premise
- Cloud privado o comercial
- Híbridos

Pros de Docker

Pros de Docker

- **Ambientes virtuales**, livianos, rápidos y desechables
- App agnóstica del entorno
- **Isolation** contra el host u otros containers
- **Arquitectura en código**
- A pattern for **scale**
- Escape from **dependency hell**
- Simplifica configuración del ambiente de desarrollo, **todas las apps son iguales desde fuera**
- **Comunidad**, docker hub


Pulls
8,000,000,000
7,000,000,000
6,000,000,000
5,000,000,000
4,000,000,000
3,000,000,000
2,000,000,000
1,000,000
2013
2014
2015
2016
2014 1M PULLS
2015 1B PULLS
2016 8B PULLS
libcontainer •
• Docker 0.9 : Pluggable execution
libnetwork •
• Docker 1.7 : Multi-Host Networking
Notary
runC •
• Docker 1.8 Docker Content Trust
• Docker for Mac
Docker for Windows
• Docker 1.11
OCI support
HyperKit, VPNKit, DataKit •
SwarmKit •
• Docker 1.12
with built-in
orchestration
containerd •

qué más?

Docker compose

Herramienta que nos permite definir y correr aplicaciones docker multi containers

Nos permite definir su arquitectura mediante código:

- services
  - building
  - links
  - ports
- volumes
- networks

más...

- docker-compose
- Orchestration
  - Kubernetes
  - Docker Swarm
  - Mesos & Marathon
  - CoreOs
- Docker in Production: Security issues
- Docker networking
- Service discovery

Preguntas?

Gracias!

TACS

Referencias

- https://docs.docker.com/
- https://www.slideshare.net/TriNimbus/from-your-desktop-to-the-cloud-things-you-need-to-consider-before-you-run-docker-in-aws
- https://github.com/DataDog/the-monitor/blob/master/docker/1_the_docker_monitoring_problem.md
- https://docs.docker.com/get-started
- https://labs.play-with-docker.com