Cloud computing

Tema 2. Fundamentos

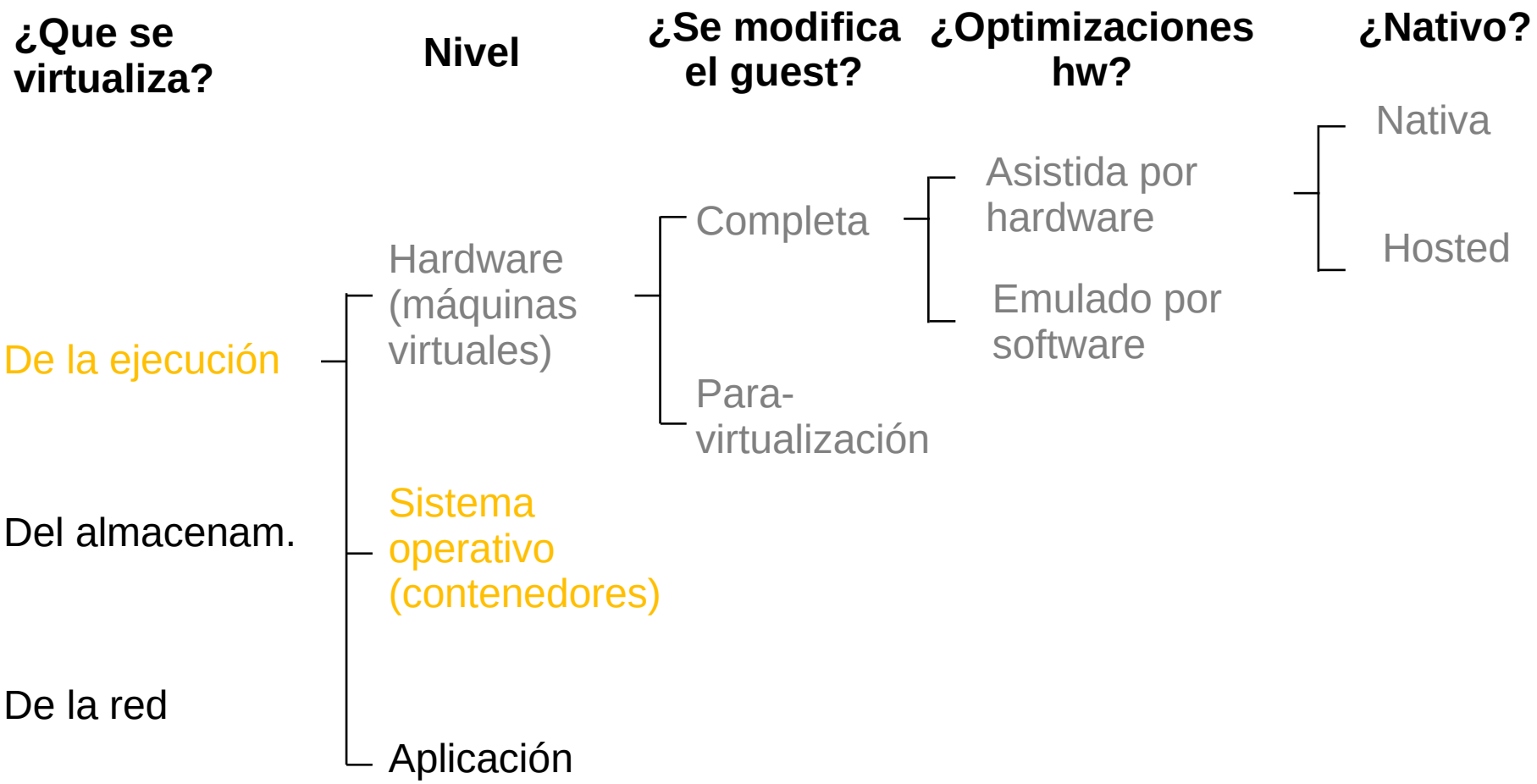
Contenido

- Introducción
- SOA
  - Servicios
  - Principios de diseño
  - Servicios RESTful
- Virtualización
  - Virtualización hardware
  - Libvirt
- Contenedores
  - Docker

Virtualización del sistema operativo

Virtualización del sistema operativo

- No completa



Contenedores

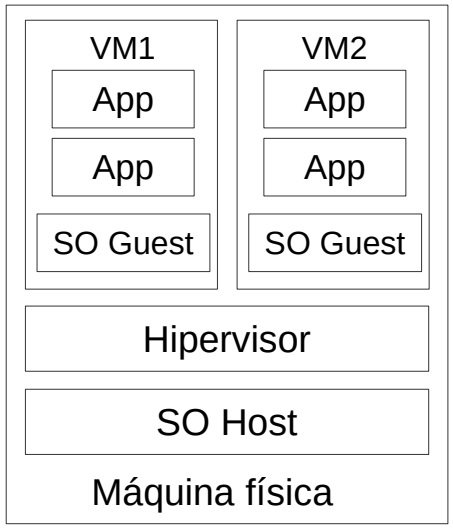
¿Qué es?

- Es un paquete autocontenido que contiene un entorno de ejecución completo: código, dependencias, ficheros, etc.
- Cuando se ejecuta, es un proceso aislado del resto, no se ven entre sí
- OCI (Open Container Initiative)

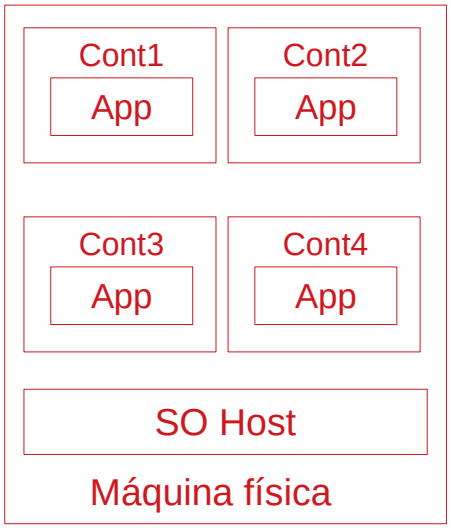
Contenedores

¿Qué es?

- Virtualización a nivel de sistema operativo
- Todos los contenedores comparten el núcleo del sistema operativo: son máquinas virtuales ligeras!!!



Máquinas virtuales



Contenedores

Contenedores

Ventajas

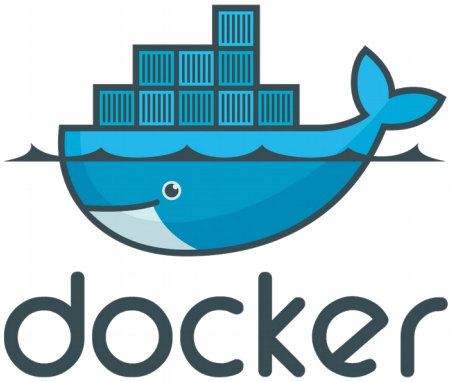
- Consumo de recursos muy bajo: cientos de contenedores por máquina física
- Es posible reproducir el mismo entorno (mismas herramientas, librerías, etc.) en todo momento
- Muy útil en Devops, todo el mundo trabaja en el mismo entorno de ejecución
- Muy útil en producción: se despliega el mismo entorno en cualquier plataforma

Contenedores

Secretos principales

- Kernel de Linux:
  - Linux namespaces: particionado de recursos
  - Cgroups: limitar consumo de recursos
- Sistemas de ficheros estructurados en capas
- Gestionar estos instrumentos es complejo
- Existen soluciones que simplifican la gestión de contenedores: LXC, Docker

Docker



https://www.docker.com/

Docker

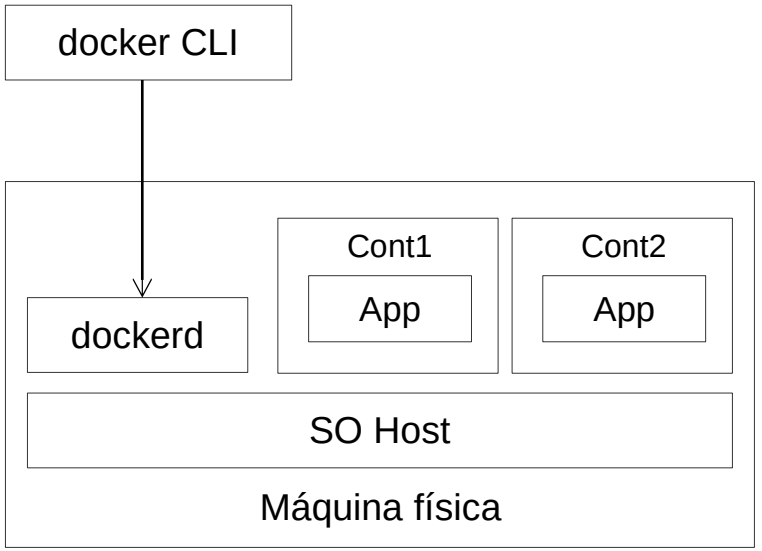
Introducción

- Docker Desktop vs Docker Engine
- Es la tecnología de contenedores más utilizada
- Proporciona herramientas muy útiles para trabajar con contenedores, utilizando por debajo las herramientas del núcleo de Linux (Linux namespaces, cgroups) y sistemas de ficheros a capas
- Disponible en distintos sistemas operativos

Docker

Introducción

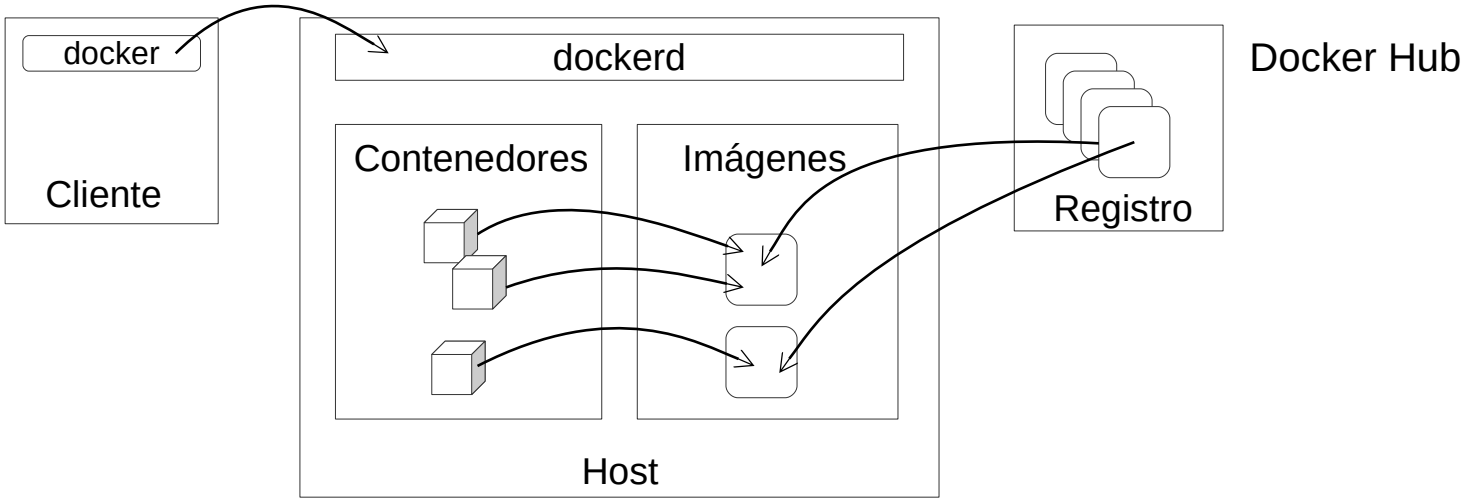
- Solución cliente-servidor
- Demonio *dockerd*
  - Se ejecuta como root
  - Publica API RESTful
  - Gestiona los contenedores
- Herramienta CLI *docker*
  - Envía comandos a *dockerd*



Docker

Conceptos básicos

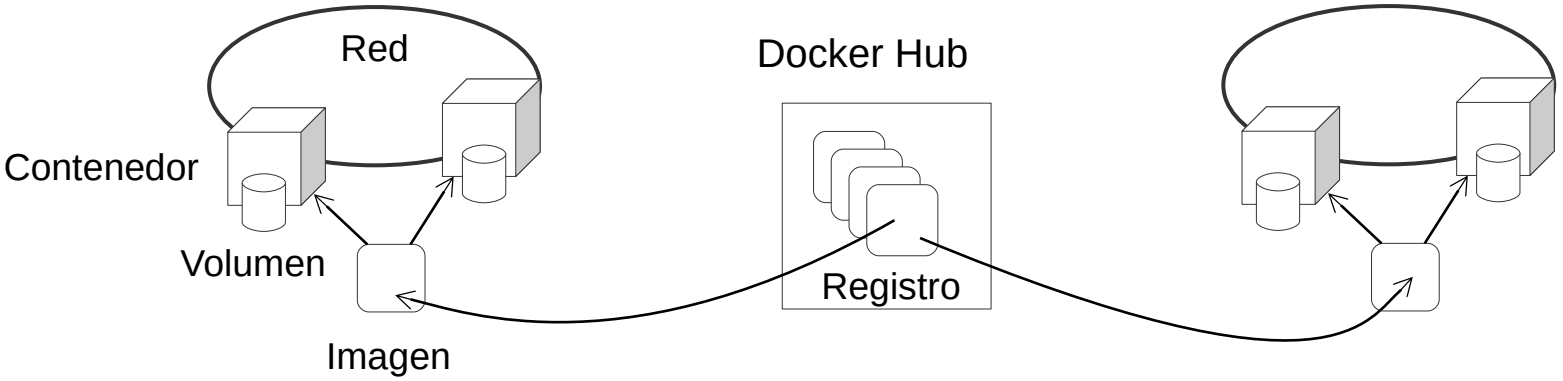
- Un contenedor es un proceso aislado
- Cada contenedor se crea a partir de una imagen
- Las imágenes se descargan de un registro



Docker

Conceptos básicos

- Los contenedores se conectan entre sí por medio de redes
- Los contenedores almacenan datos en volúmenes



Docker

Hola de ruta

- Contenedores
- Redes
- Almacenamiento
- Imágenes

Docker

Contenedores > Ejemplo

```
> docker run hello-world
```

1. Conecta con el Docker Hub y descarga la imagen
2. Instala la imagen localmente
3. Crea un contenedor a partir de la imagen
4. El contenedor se ejecuta y finaliza

```
> docker run hello-world
```

1. Crea un contenedor a partir de la imagen
2. El contenedor se ejecuta y finaliza

Docker

Contenedores > Docker Hub

- Los contenedores se crean a partir de imágenes
- Podemos obtener las imágenes de distintas fuentes
  - Un fichero
  - Un registro de imágenes privado
  - El Docker Hub
    - Imágenes estándar (e.g. hello-world, debian, mongo, …) vs imágenes de usuario (e.g. microsoft/aspnet, jboss/wildfly, ...)

Docker

Contenedores > Comandos

```
> docker container --help
```

- Arrancar un contenedor

```
> docker [container] run [--rm][--name <name>] <img>
> docker run <img> [cmd]
```

- Listar contenedores

```
> docker container ls [-a] / docker ps [-a]
```

- Eliminar contenedor

```
> docker [container] rm [--force] <cont>
```

Docker

Contenedores > Comandos

- Modo interactivo (captura entrada y terminal)

```
> docker run -it <img>
  Ctrl+P Ctrl+Q
> docker [container] attach <cont>
```

- Modo detached (en background)

```
> docker run -d <img>
```

Docker

Contenedores > Comandos

- Inspeccionar contenedores

```
> docker [container] inspect <cont>
```

- Inspeccionar salida

```
> docker [container] logs [--tail n] <cont>
```

- Ejecutar comando

```
> docker [container] exec <cont> <cmd>
```

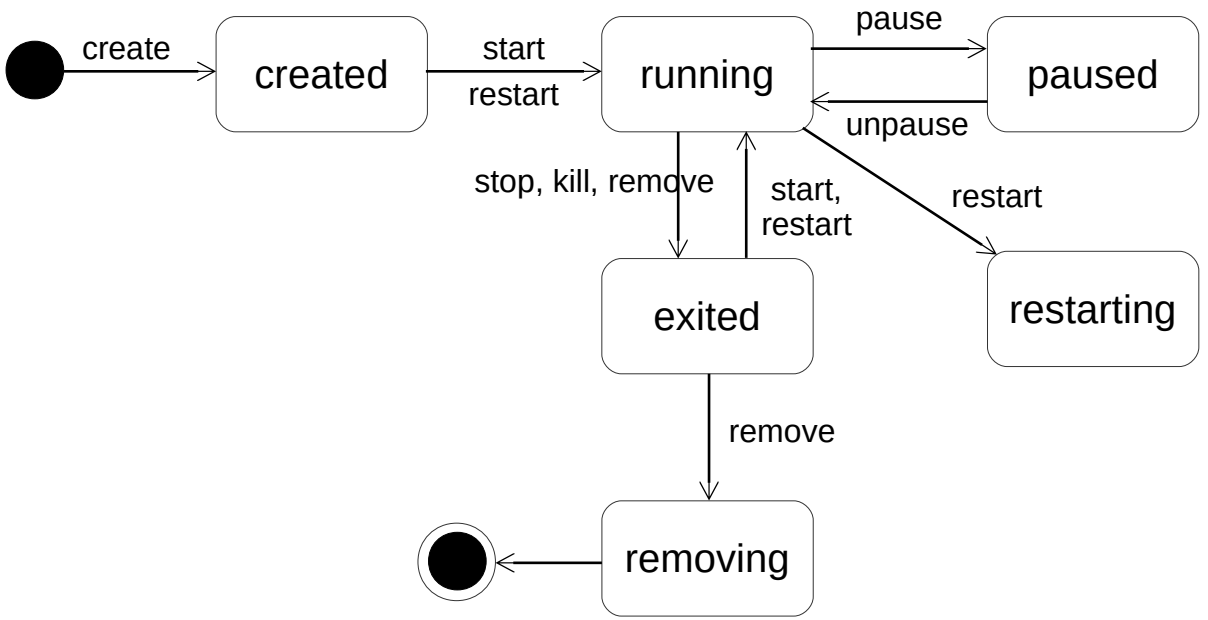
- Pasar variables de entorno

```
> docker run -e <VAR>=<VALUE> <img>
```

Docker

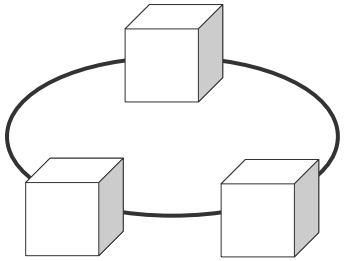
Contenedores > Ciclo de vida

```
> docker [container] create <img>
> docker [container]
          start/pause/unpause/restart/stop/rm <cont>
```



Docker

Redes



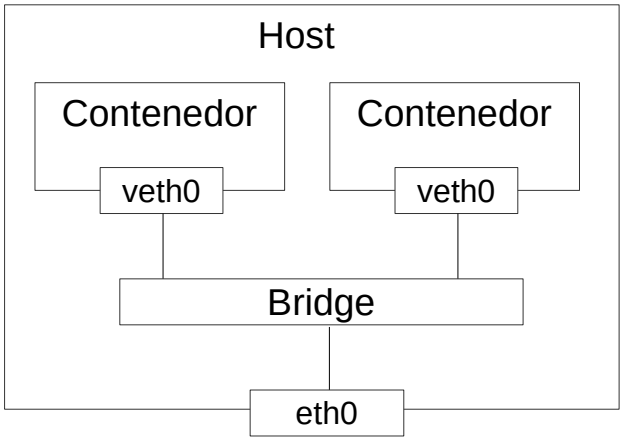
- Interconectan a los contenedores
- Tipos
  - bridge: varios contenedores en una red local
  - host: se fusiona el contenedor con el host
  - none: no hay red
  - etc.
- Listar las redes

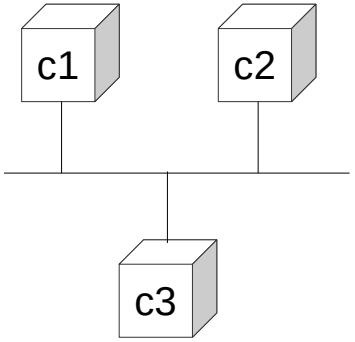
```
> docker network ls
```

Docker

Redes > Bridge

- Red interna privada, con conectividad hacia afuera
- No se puede recibir tráfico del exterior





Docker

Redes > Bridge

- Existe una red bridge por defecto a la que se conectan todos los contenedores
- Se pueden crear nuevas redes bridge de usuario

```
> docker network create <net>
```

- Conectar contenedores nuevos

```
> docker run --network <net> <image>
```

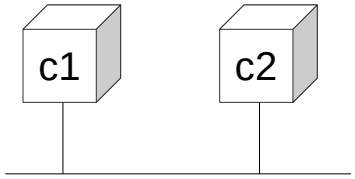
- (Des)Conectar contenedores ya arrancados

```
> docker network (dis)connect <net> <container>
```

Docker

Redes > Bridge

- Ejemplo: diseñar una red con dos contenedores




```
> docker network create mynet
> docker run -it --name c1 --network mynet debian
> docker run -it --name c2 --network mynet debian
```

Docker

Redes > Puertos

- Los puertos de un contenedor no son visibles desde fuera
- Mapear puertos de un contenedor en el host

```
> docker run -p <p-host>:<p-container> <image>
```

- Consultar puertos mapeados

```
> docker port <container>
```

Docker

Redes > Puertos

- Ejemplo: crear un contenedor Nginx (servidor web) y mapear puertos

```
> docker run --name nginx -p 8080:80 nginx
```

```
> docker port nginx
```

Docker

Redes > Host

- El contenedor y el host comparten la misma pila de red
- Todo lo que publica el contenedor es visible desde fuera

```
> docker run --network host <image>
```

Docker

Almacenamiento

- Por defecto todos los ficheros creados/modificados por un contenedor son temporales
- Dos mecanismos de persistencia
  - Volúmenes: los datos son gestionados por Docker
  - Mounts: los datos se guardan en un directorio del host
- Es más recomendable trabajar con volúmenes

Docker

Almacenamiento > Volúmenes

- Los datos se guardan en /var/lib/docker/volumes
- Crear volumen

```
> docker volume create <vol>
> docker run -v <vol>:<path> <img>
```

- Listar volúmenes

```
> docker volume ls
```

- Eliminar volumen

```
> docker volume rm <vol>
```

Docker

Almacenamiento > Mounts

- Los datos se montan directamente desde el host

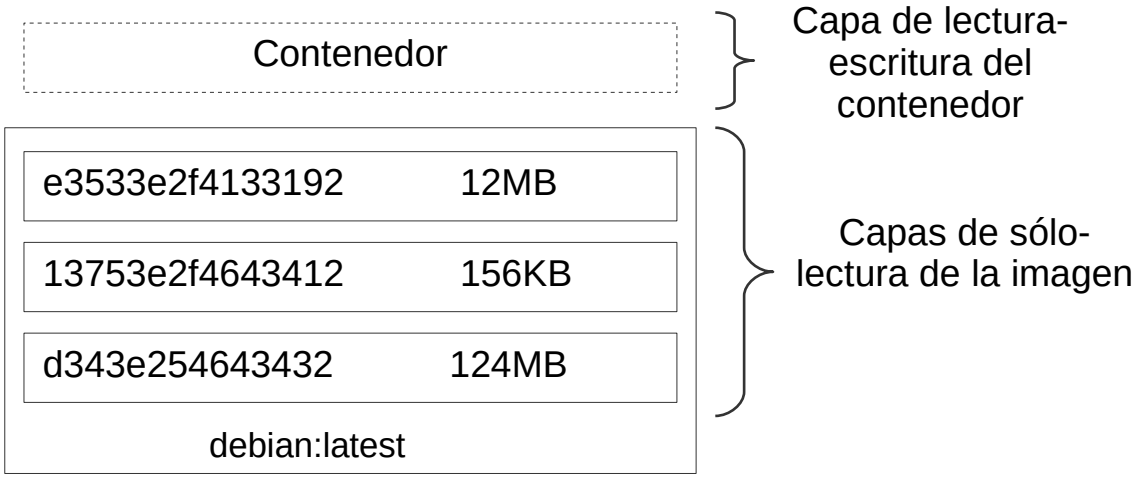
```
> docker run -v <path-host>:<path-cont> <img>
```

- Útil para compartir datos/ficheros con el host
- Compromete la portabilidad del contenedor

Docker

Imágenes

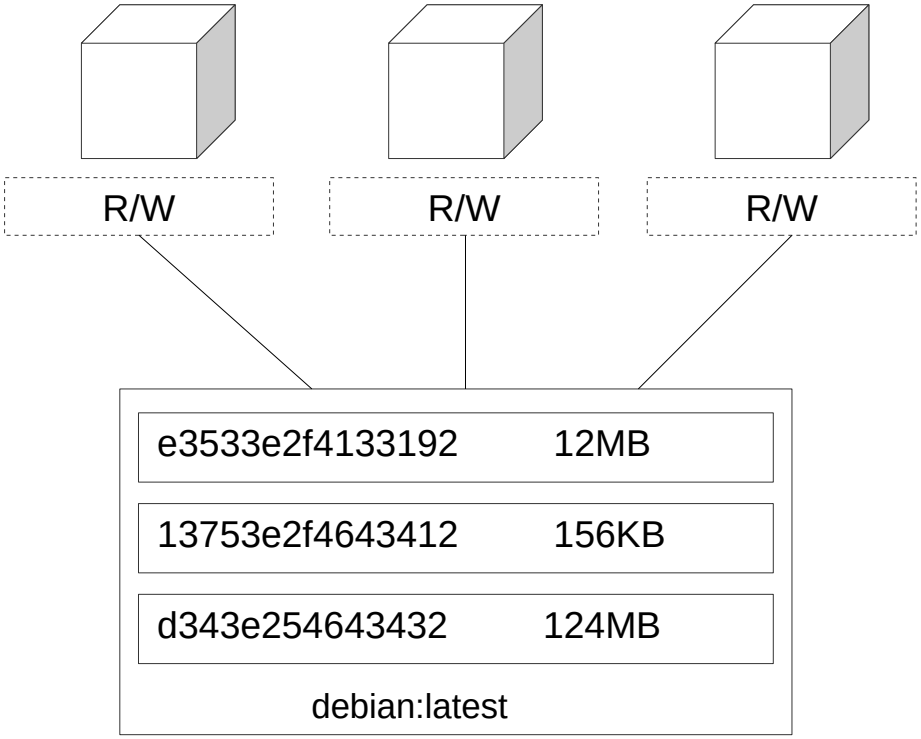
- Cada contenedor se crea a partir de una imagen
- Una imagen se compone de una pila de capas de sólo lectura
- El contenedor añade una capa fina de lectura-escritura



Docker

Imágenes

- Todos los contenedores con la misma imagen comparten todas las capas de sólo-lectura



Docker

Imágenes > Comandos

- Instalar una imagen del Docker Hub

```
> docker [image] pull <image>
```

- Listar imágenes

```
> docker image ls / docker images
```

- Eliminar imágenes

```
> docker image rm <image> / docker rmi <image>
```

Docker

Imágenes > Crear una imagen

- De manera interactiva
- A partir de un fichero Dockerfile

Docker

Imágenes > Crear una imagen

- De manera interactiva
  - Descargar una imagen base y crear contenedor
  - En el contenedor efectuar los cambios que se desee
  - Comprobar los cambios efectuados

```
> docker diff <container>
```

  - Confirmar la nueva capa del contenedor

```
> docker commit [-a] [-m] <container> <img-name>
```

  - Se añade una nueva capa de sólo lectura a una nueva imagen

Docker

Imágenes > Dockerfile

- Para construir una imagen de manera automática
- Se ejecutan secuencialmente las instrucciones del fichero Dockerfile
- La creación de la imagen es determinista, repetible y portable
- La imagen se crea a partir de Dockerfile y un contexto, que contiene todos los recursos para construir la imagen (directorio, git, .tar)

```
> docker image build <ctxt> / docker build <ctxt>
> docker build . -t <name>
```

Docker

Imágenes > Dockerfile

- Se envía todo el contexto (directorio?) a *dockerd* y éste procesa las instrucciones de Dockerfile
- La primera instrucción FROM determina la imagen base

```
FROM debian
```

- Cada instrucción se ejecuta en un contenedor y produce cambios en una capa de lectura-escritura

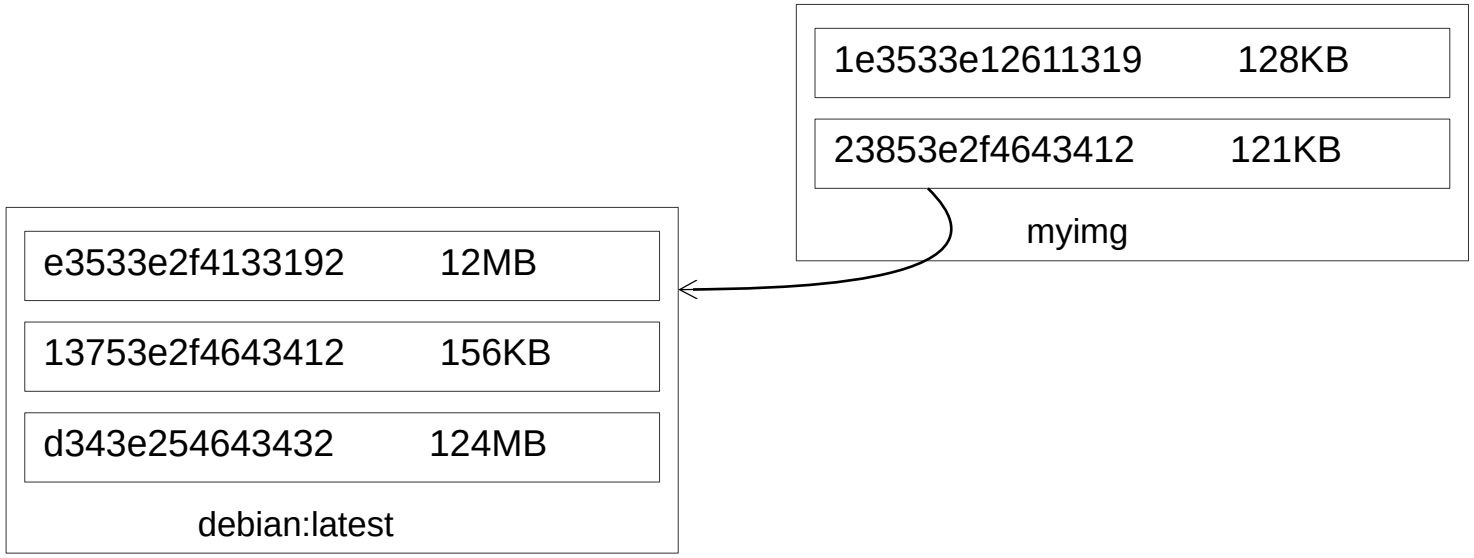
```
RUN apt-get update && apt-get install -y vim
```

Docker

Imágenes > Dockerfile

- Al finalizar cada instrucción, se confirman los cambios en una capa de sólo-lectura

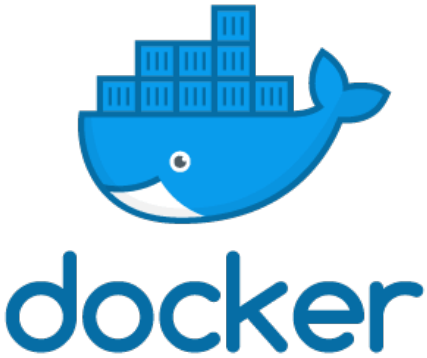
```
FROM debian
RUN apt-get update && apt-get install -y vim
RUN apt-get update && apt-get install -y netcat
```



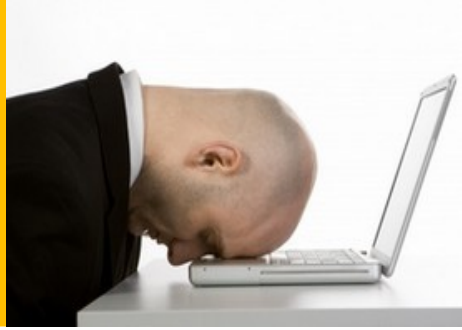
Docker

Imágenes > Dockerfile

- FROM
- LABEL
- RUN
- COPY, ADD
- ENTRYPOINT, CMD
- ENV, ARG
- EXPOSE
- VOLUME

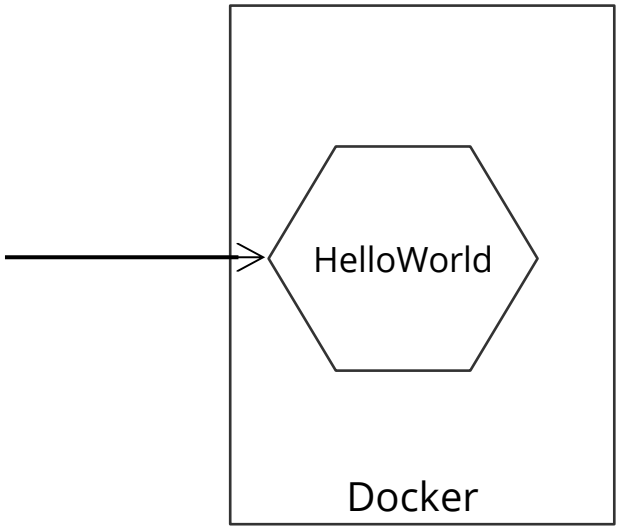


Docker



Ejercicio 3

- Dockerizar (crear una imagen de) un servicio HelloWorld FLASK utilizando Dockerfile



Docker

Imágenes > Repositorios

- Tres maneras de obtener imágenes
  - Docker Hub
  - Ficheros
  - Docker Registry

Docker

Imágenes > Repositorios

- Ficheros
  - En el host donde está instalada la imagen

```
> docker image save -o debian.tar debian
```

  - Instalar imagen en otro host

```
> docker image load -i debian.tar
```

Docker

Imágenes > Repositorios

- Docker Registry
  - Crear repositoro privado de imágenes

```
> docker run -d -p 5000:5000 --name registry registry:2
```

  - El nombre completo de una imagen indica dónde reside la imagen

```
<registry-host>:<port>/<repository>/<name>:<tag>
```

```
debian → docker.io/library/debian:latest
```

```
localhost:5000/mydebian:latest
```

Docker

Imágenes > Repositorios

- Docker Registry
  - Asignar nombres a una imagen

```
> docker commit <container> <img-name>:<tag>
> docker build -t <img-name>:<tag> <path>
> docker tag <image> <img-name>:<tag>
```

Docker

Imágenes > Repositorios

- Docker Registry
  - Registrar imagen en registro privado

```
> docker pull debian   # Docker Hub debian:latest
> docker tag debian:latest localhost:5000/mydebian
> docker push localhost:5000/mydebian
> docker rmi localhost:5000/mydebian
> docker pull localhost:5000/mydebian
```

Docker Compose

- Docker Compose permite definir y ejecutar aplicaciones con múltiples contenedores (en una sóla máquina)
- Para desplegarlos en múltiples máquinas: Docker Swarm o Kubernetes
- Todos los servicios de la aplicación se definen en docker-compose.yml
- Compose automatiza todos los comandos que ejecutaría docker CLI

Docker Compose

Proceso

1. Dockerizar (creamos imágenes de) todos los componentes de la aplicación con Dockerfile
2. Definir la aplicación con docker-compose.yml
3. Arrancar la aplicación con el comando:

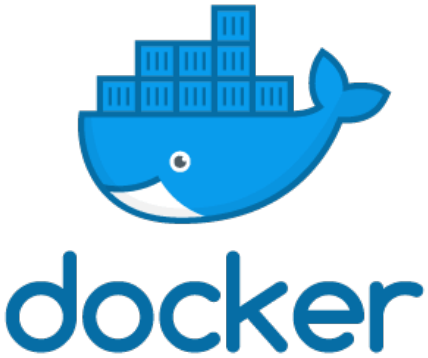
```
> docker compose up
```

Docker Compose

docker-compose.yml

- Estructura básica

```
version:"3.8"
services:
  service1: ...
  service2: ...
volumes:
  vol1: ...
  vol2: ...
networks:
  net1: ...
  net2: ...
```



Docker Compose

docker-compose.yml

- Contenedores


```
services:
  webapp:
    image: debian
```

```
services:
  webapp:
    build: ./dir
```

```
services:
  webapp:
    build: ./dir
    entrypoint: npm run serve
```

```
services:
  webapp:
    image: debian
    environment:
      PUBLISH_URL: '/webapp'
      SSL: true
```

```
services:
  webapp:
    image: debian
    ports:
      - "8080:80"
      - "10022:22"
```

Docker Compose

docker-compose.yml

- Volúmenes


```
services:
  db:
    image: db
    volumes:
      - /var/lib/log               # crea un nuevo volumen automáticamente
      - data-volume:/var/lib/db    # monta un volumen con nombre
  backup:
    image: backup-service
    volumes:
      - /opt/data:/var/lib/mysql   # monta un directorio del host

volumes:
  data-volume:
```

Docker Compose

docker-compose.yml

- Redes


```
services:
  webapp:
    image: debian
    networks:
      - net1
      - net2

networks:
  net1:
  net2:
```

```
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

Docker Compose

CLI

- Múltiples comandos

> docker compose

- Los más útiles son:

> docker compose up

> docker compose down

> docker compose build


The End