Docker

Servicios de Red e Internet – 2º ASIR

Introducción

Desarrollo de aplicaciones y Despliegue de aplicaciones web.

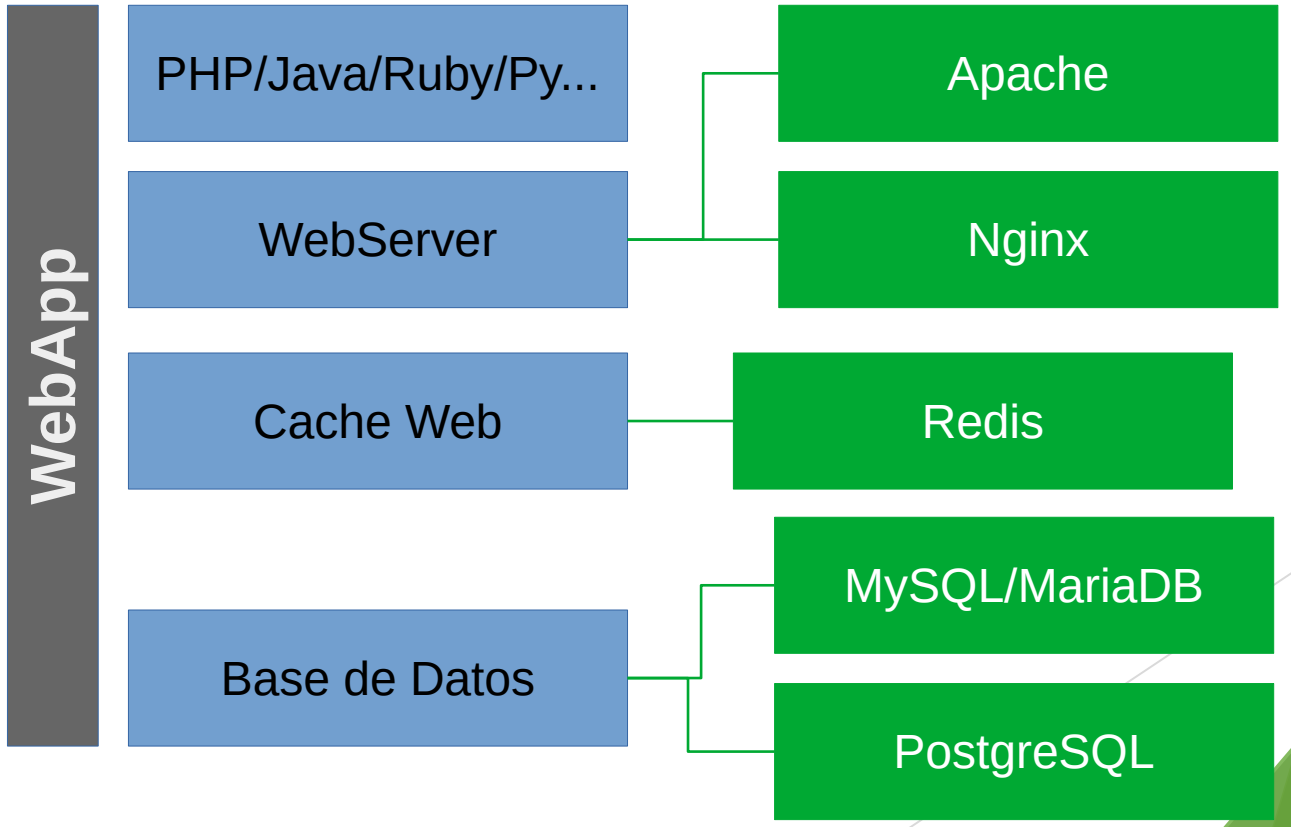
Los programadores necesitan desarrollar sus aplicaciones en un entorno de prueba

Dev – Prod parity: similitud entorno de desarrollo y producción

Alternativas:

Desarrollo Local Hypervisores Contenedores

Escenarios complejos



Introducción

Alternativas:

Desarrollo Local: Instalación de las aplicaciones directamente en el PC de trabajo

- Solución válida y sencilla
- Problemas de compatibilidad SOs usados (ficheros, codificación ficheros,...)
- Problemas de escalabilidad tanto de servicios como de equipo de desarrollo

Hypervisores: Desarrollo de máquinas virtuales compartidas

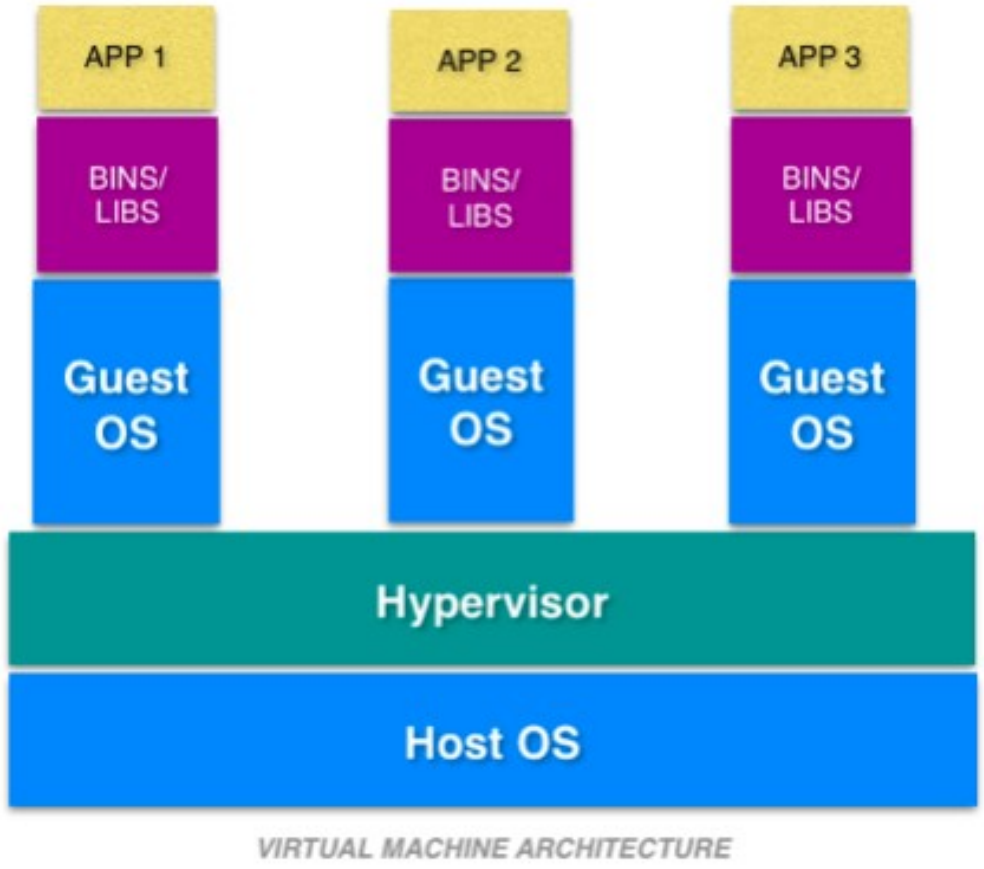
- Es posible hacer réplicas de las MV para cada miembro del equipo (Boxes de Vagrant)
- Separación robusta entre el Host y el entorno de desarrollo
- Problemas de desvío de la configuración con el tiempo. Entorno mutable
- Penalización en recursos. Penalización del SO y de licencias

Contenedores:

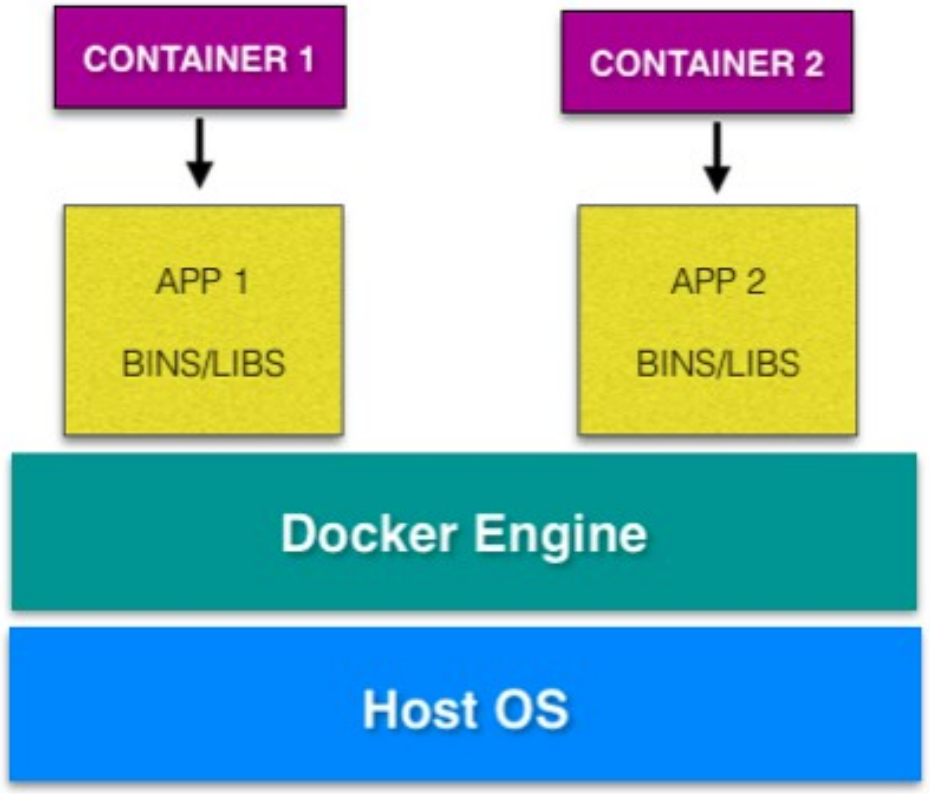
- Compartimenta funciones y puede usarse desde el desarrollo hasta la producción
- Aisla actualizaciones y cambios de configuración
- Despliegue rápido
- Contenedores enfocados en microservicios. Una “sola” función separada.

Contenedores vs Máquinas Virtuales

Penalización del SO



VIRTUAL MACHINE ARCHITECTURE



DOCKER ARCHITECTURE

Instalación Docker

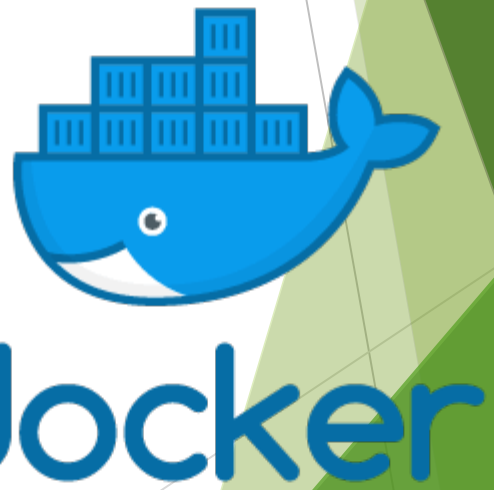
Está compuesto por diversos elementos:

Docker Desktop – Para W10Home y MAC
Docker Engine - Directamente en Linux
Docker Compose - Herramienta de Python para la creación de servicios
Docker Hub - Crear cuenta

Instalación:

Docker Engine Ubuntu: https://docs.docker.com/engine/install/ubuntu/

Docker Compose: https://docs.docker.com/compose/install/



Ejecutando una instancia de Docker

Vamos a ver distintas opciones de arrancar contenedores que nos permitirán lanzar infraestructuras más o menos complejas

docker run: Es la aproximación más sencilla y arranca un contenedor

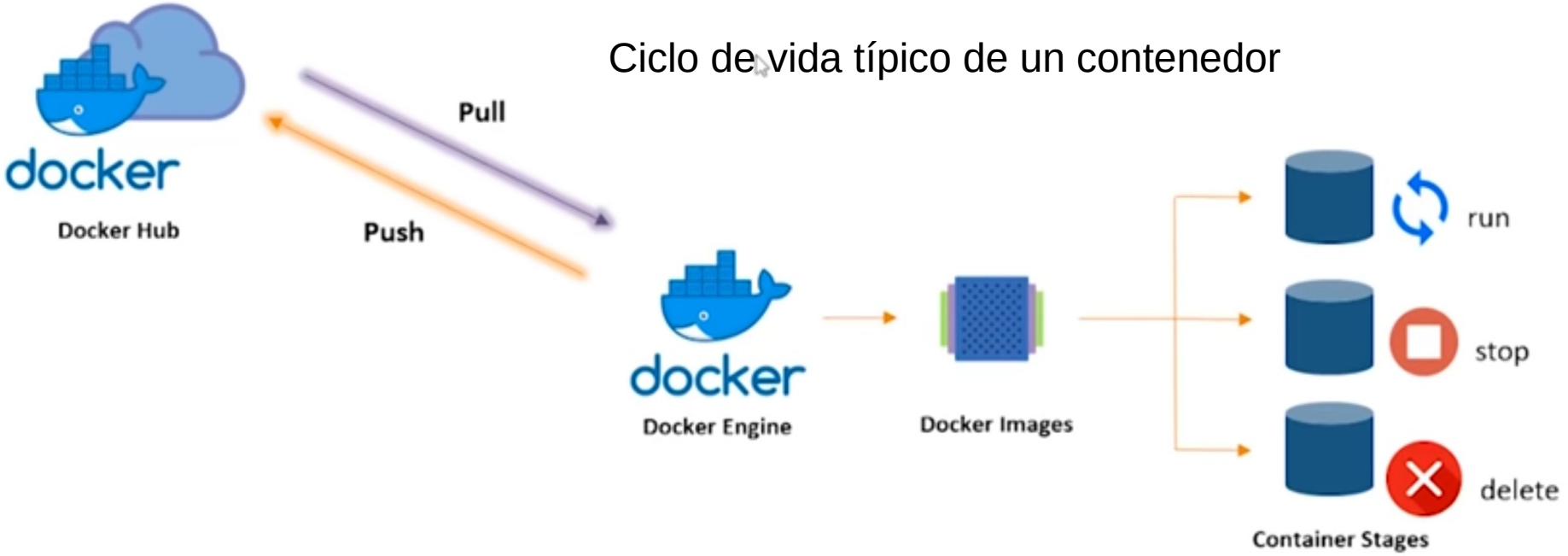
```
docker run -it ubuntu /bin/bash
    docker container ls
    docker container ls -a
    docker container
```

```
docker run -it --rm ubuntu /bin/bash
docker run -it -d --rm --name=UBU1 ubuntu /bin/bash
    docker attach UBU1
```

```
docker image ls
```

Ciclo de vida contenedor Docker

Ciclo de vida típico de un contenedor



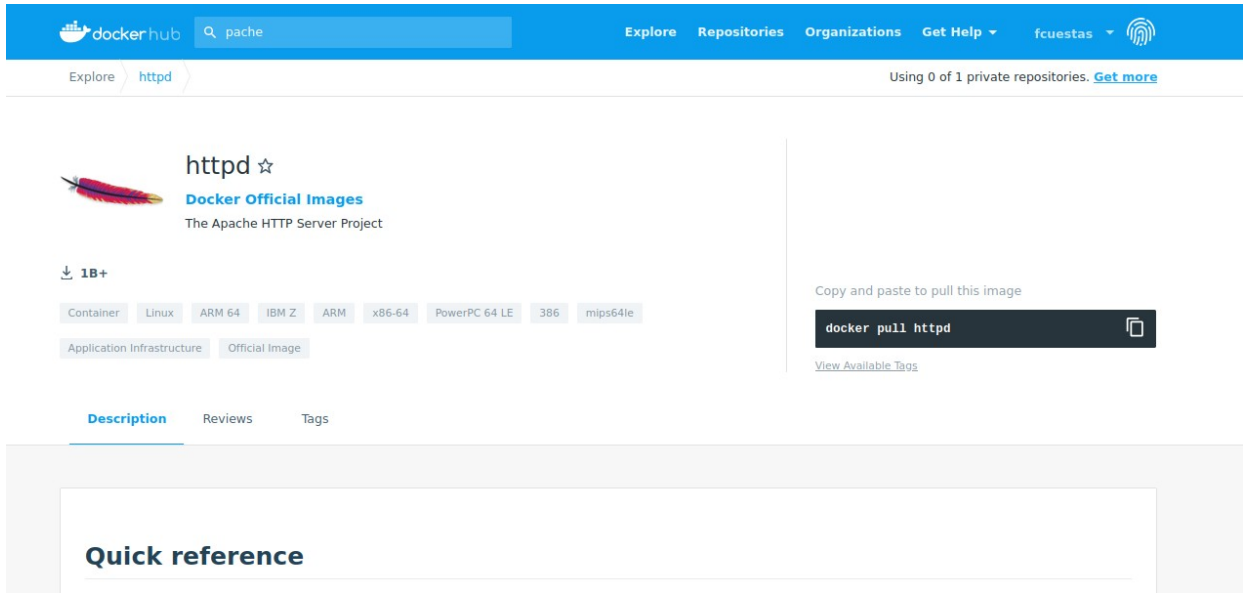
Encontraremos situaciones más avanzadas:

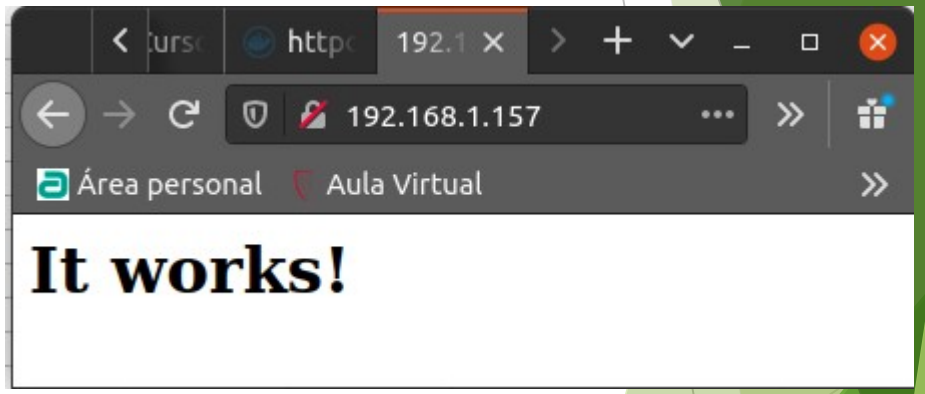
- Servicios: Varios contenedores desplegados en una red
- Stacks: Pilas de servicios desplegadas en red para aplicaciones/infraestructura compleja

Ejecutando una instancia de Docker

Ejemplo contenedor Apache: https://hub.docker.com/_/httpd

docker run -dit --name appico -p 80:80 httpd





Elementos en Docker

Imágenes:
- Build – Capas Dockerfile
- Push/Pull

Contendores: Creación-ejecución, interacción, parada y eliminación

Volúmenes:
- Capas RO
- Capa RW
- Datos persistentes

Redes: tipos de redes y puertos

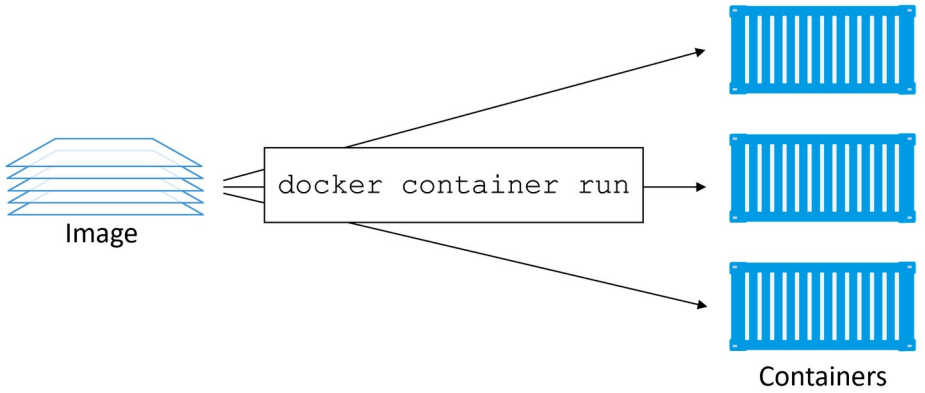
Swarm:

Servicios: Docker-compose.yaml

Stack:

Contenedores en Docker

Un contenedor es una instancia de la imagen ejecutándose. Parte de la plantilla imagen y comienza la ejecución




```
docker container run -it mcr.microsoft.com/powershell:nanoserver pwsh.exe
```

Ciclo de vida ligado al proceso principal

Imágenes

Creación y Gestión

Imágenes en Docker

Imagen: Unidad básica que tiene todos los elementos necesarios para que una aplicación se ejecute.

Es como una plantilla que define SO, código, dependencias etc…
Un contenedor se crea a partir de una imagen.

(Build time) (Runtime)

Diseño mínimo: sólo contiene lo esencial para la aplicación para la que se ha diseñado.
(Código y sus dependencias)
Sin Kernel: se aprovecha del host anfitrión

Generalmente tamaños muy bajos. (Ej: alpine Linux ~ 5MB, Ubuntu ~ 73 MB)

Gestión de Imágenes en Docker

Pull: Para descargar imágenes de repositorios (/var/lib/docker/<driver-almacen>)

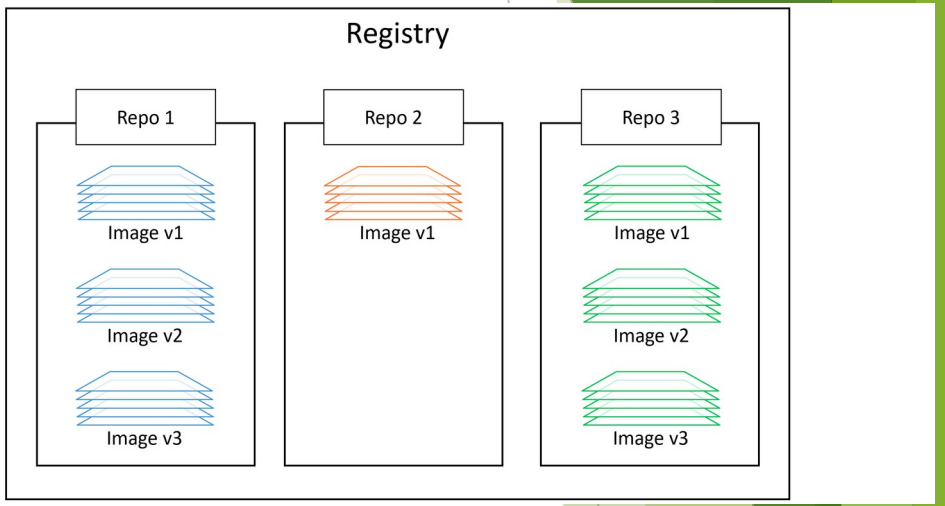
docker image pull <repositorio>:<tag>

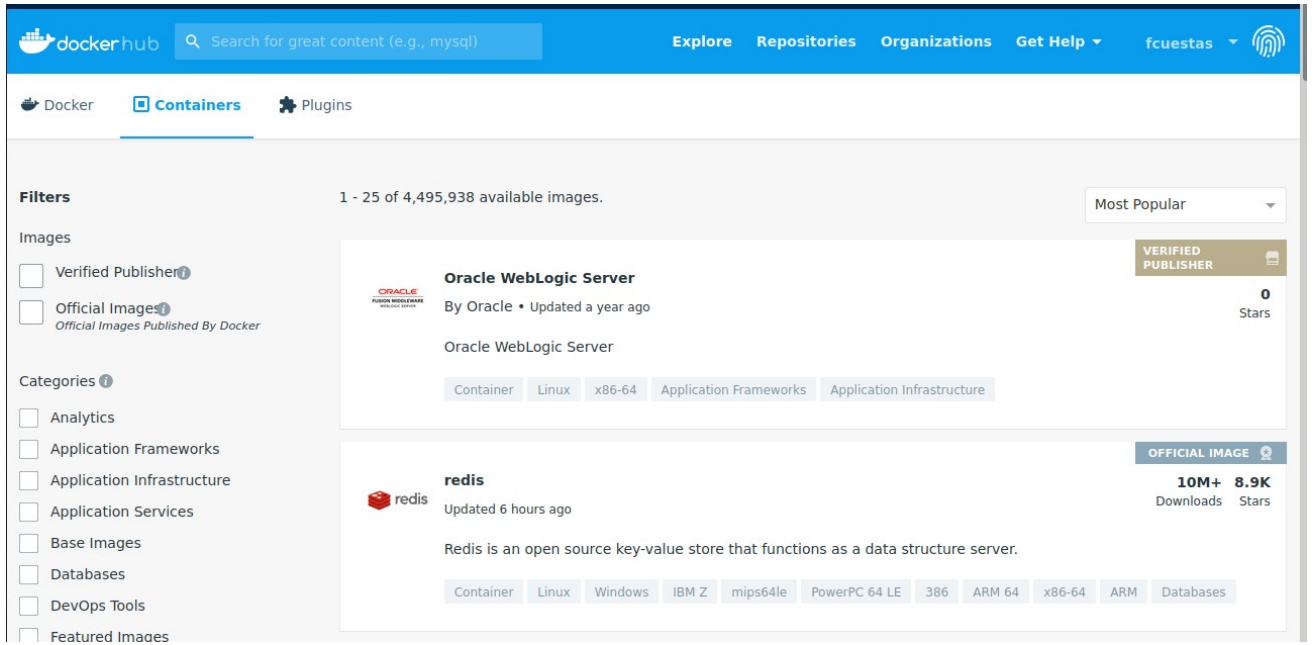
docker image ls

Push: Para subir imágenes a un repositorio

Registro principal: DockerHub: link

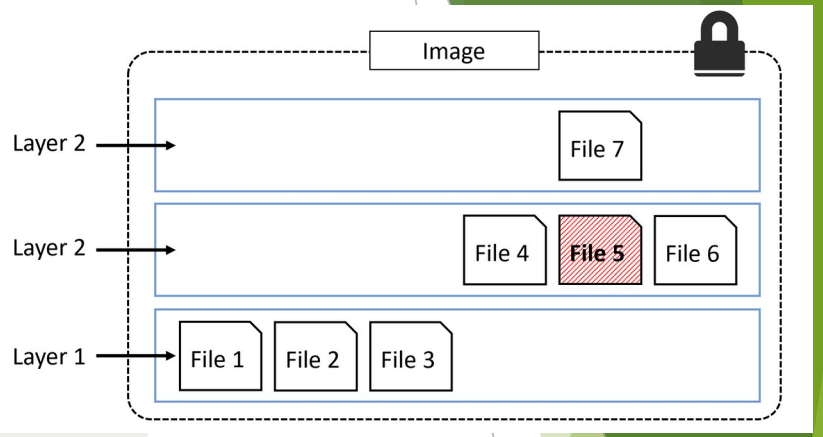
Otros: Google cloud link





Creación de Imágenes en Docker

Imágenes formadas a partir de capas RO



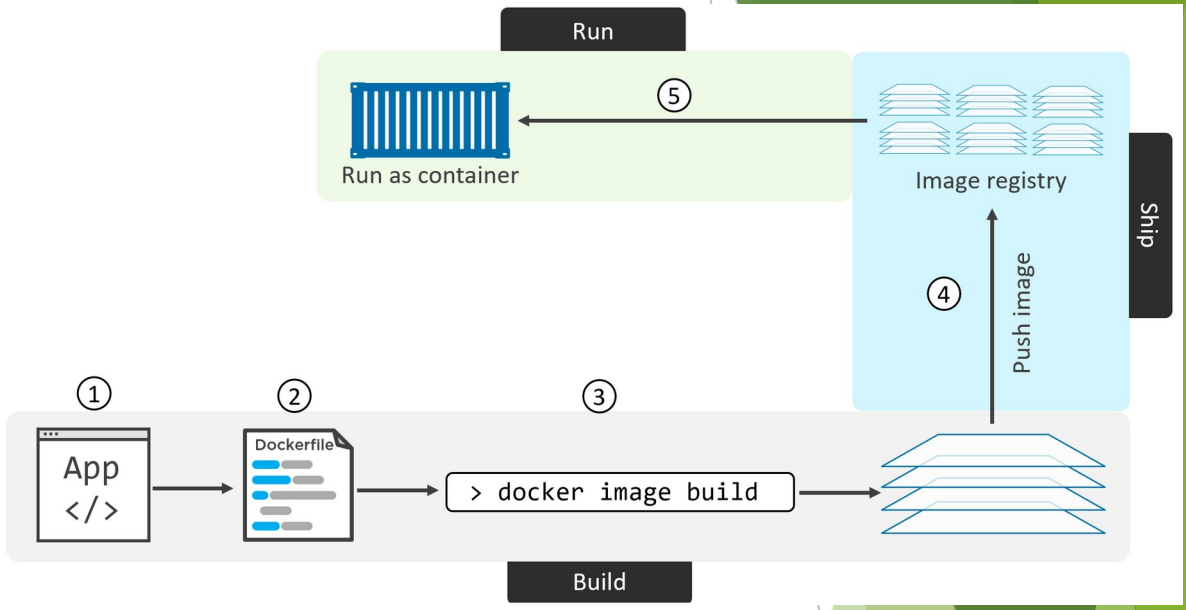

```
ico@dock:~$ docker image pull httpd
Using default tag: latest
latest: Pulling from library/httpd
6ec7b7d162b2: Pull complete
17e233bac21e: Pull complete
130aad5bf43a: Pull complete
81d0a34533d4: Pull complete
da240d12a8a4: Pull complete
Digest: sha256:a3a2886ec250194804974932eaf4a4ba2b77c4e7d551ddb63b01068bf70f4120
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

Las capas pueden ser compartidas por distintas imágenes

Creación de Imágenes en Docker

Ejemplo:
Podemos descargar este ejemplo
git clone https://github.com/nigelpoulton/psweb.git

Donde encontramos el fichero Dockerfile



Etiquetas:
FROM: Etiqueta que indica la imagen de partida
LABEL: Información sobre el autor
COPY: Permite copiar ficheros en la imagen
WORKDIR: Establece el directorio de trabajo
ENTRYPOINT: Establece el comando o aplicación por defecto de la imagen

Las capas pueden ser compartidas por distintas imágenes

Chuleta con distintas etiquetas: link

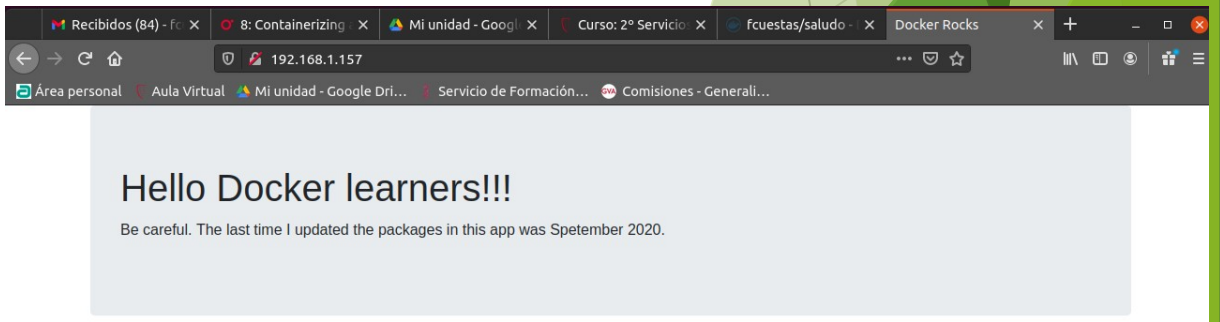
Creación de Imágenes en Docker

Ejemplo:
Podemos descargar este ejemplo, echa un vistazo a los archivos descargados
git clone https://github.com/nigelpoulton/psweb.git

A continuación creamos la imagen:
docker build -t fcuestas/poulton .
Esto nos permite crear la imagen fcuestas/poulton (Nombre repositorio=fcuestas)
docker image ls

Probamos un contenedor con esta imagen:
docker run --rm --name c1 -p80:8080 fcuestas/poulton

Ahora podemos comprobar que funciona conectando por web desde el navegador a la dirección del host.

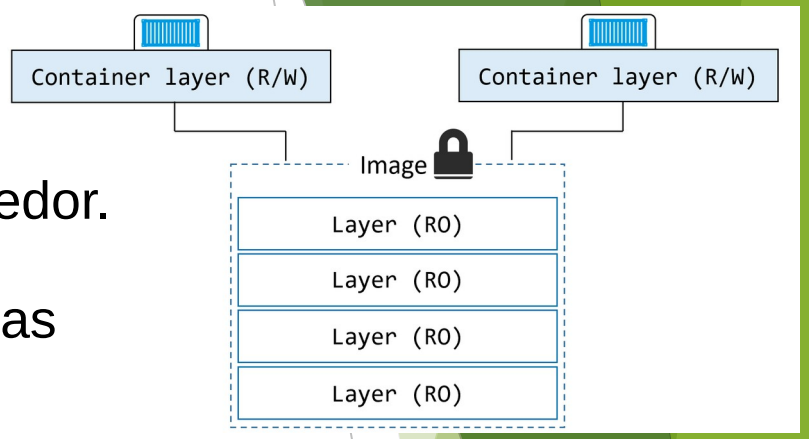


Esta imagen la podemos subir a nuestro repositorio
Haciendo:
docker login
docker push fcuestas/poulton

Volúmenes

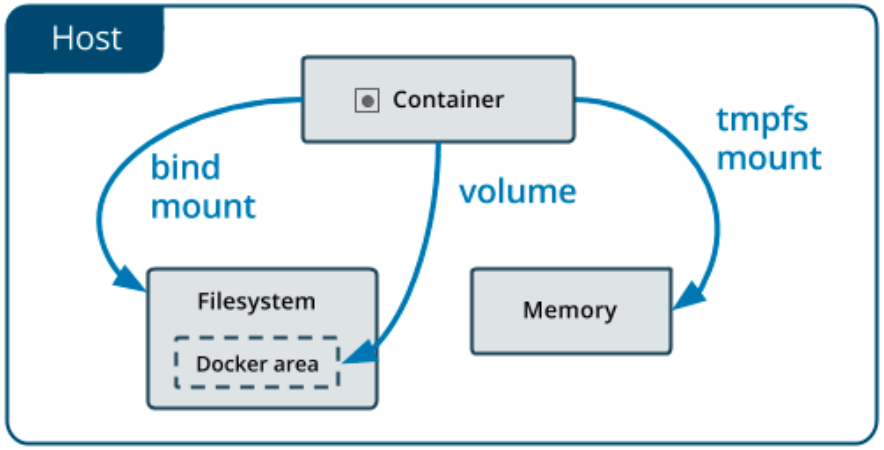
Persistencia de datos

Volúmenes en Docker



Las capas de la imagen son inmutables (RO)
La capa superior es de lectura/escritura pero **se elimina** con el contenedor.

Si queremos conservar los datos debemos recurrir a los montar sistemas de almacenamiento externos.
Dentro del contenedor se verán como un directorio más.



Alternativas:

- tmpfs: Esta es la capa R/W que se elimina
- **Volúmenes**: opción recomendada gestionada por Docker
- Bind: Enlaza a carpetas del host. Es peligroso y no recomendado
- Named pipes: Alternativa de comunicación

Volúmenes en Docker

Los volúmenes permiten persistencia en los datos. Son unos elementos de docker independientes de los contenedores

Creación de un volumen:

docker volume create myvol

Gestión volúmenes:

docker volume ls / docker volume inspect myvol / docker volume rm myvol

Usamos un volúmen en un contenedor: Montamos el volumen en /vol dentro de c1

**docker container run -dit –name c1 **

--mount source=myvol,target=/vol alpine #NOTA: si no existe el volumen lo crea

Eliminación de un volumen: No se elimina con el contenedor, hemos de eliminarlo manualmente

docker volume rm myvol

NOTA: Por defecto se usan volúmenes locales, existe la posibilidad de crear volúmenes con plugins especiales que permitan almacenamiento centralizado o características específicas.

Docker Compose

Composición de aplicaciones

Servicios en Docker

Docker compose: Herramienta externa a Docker Engine que permite desplegar servicios. Ayuda a la gestión de múltiples “microservicios”

Herramienta de Python: De fichero YAML → Despliega los servicios
Se instala por separado: link

Permite el despliegue de múltiples contenedores coordinadamente

```
version: "3.8"
services:
  web-fe:
    build: .
    command: python app.py
    ports:
      - target: 5000
        published: 5000
    networks:
      - counter-net
    volumes:
      - type: volume
        source: counter-vol
        target: /code
  redis:
    image: "redis:alpine"
    networks:
      counter-net:

networks:
  counter-net:

volumes:
  counter-vol:
```

Servicios en Docker

Imagen para web-fe [Web front-end]
- build: se crea a partir de un Dockerfile .
- publica un puerto
- Define su red
- Define volumen de datos y lo monta

Chuleta de etiquetas: link

Imagen Redi: [Base de datos/cache]
- image: descarga una imagen delregistro
- La conecta a la misma red

Definición de las redes

Definición de los volúmenes

```
version: "3.8"
services:
  web-fe:
    build: .
    command: python app.py
    ports:
      - target: 5000
        published: 5000
    networks:
      - counter-net
    volumes:
      - type: volume
        source: counter-vol
        target: /code
  redis:
    image: "redis:alpine"
    networks:
      counter-net:

networks:
  counter-net:

volumes:
  counter-vol:
```

App en Docker Compose

Ejemplo de despliegue de una app con Docker compose

```
git clone https://github.com/nigelpoulton/counter-app.git  #Ficheros necesarios
```

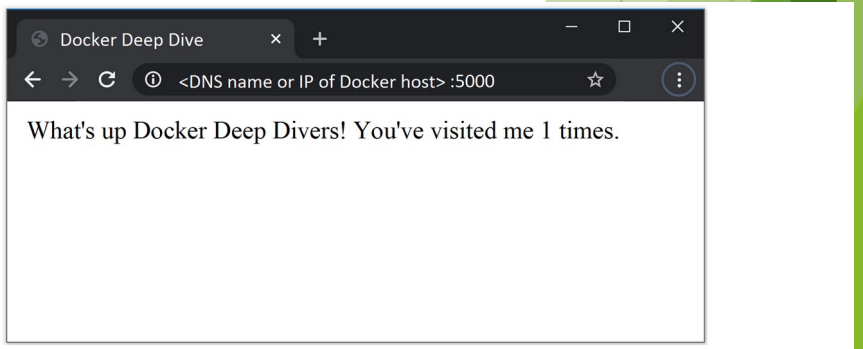
1.-Dockerfile: Crea la imagen web-fe
2.-app.py: Para la imagen anterior
3.-requirements.txt: Lista de requisitos de python para la app en la imagen web-fe

4.-Docker-compose.yml: documento con la información del despliegue

Arrancamos la aplicación:

```
docker-compose up &          #Ejecutar dentro del archivo con los ficheros
```

Resultado: (conter-app es el nombre del directorio donde se arrancó la app)
- Se automáticamente crea la imagen counter-app_web-fe (docker image ls)
- Se crea la red counter-app_counter-net
- Se crea el volumen conter-app_counter-vol



docker-compose ps → Servicios en marcha
docker-compose top → Procesos en marcha
Paramos la App: docker-compose down / stop / restart

docker-compose rm

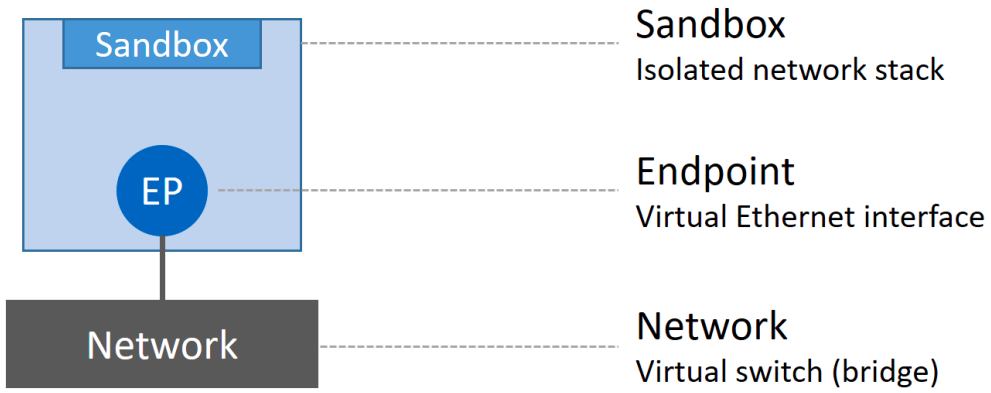
Redes

Creación y Gestión

Redes en Docker

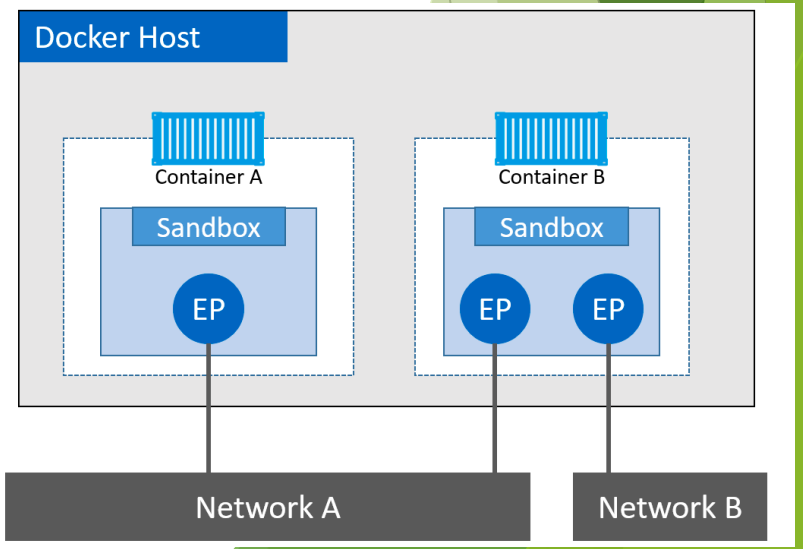
Son un elemento fundamental que se implementa por SW – CNM (Container Network Model)
Elementos:

Sandbox → Interfaces Ethernet, puertos, tablas de routing, DNS.
Endpoints → Interfaces de red virtuales veth. Realizan las conexiones
Networks → Impletación bridge virtual (802.1d): agrupa y aisla endpoints



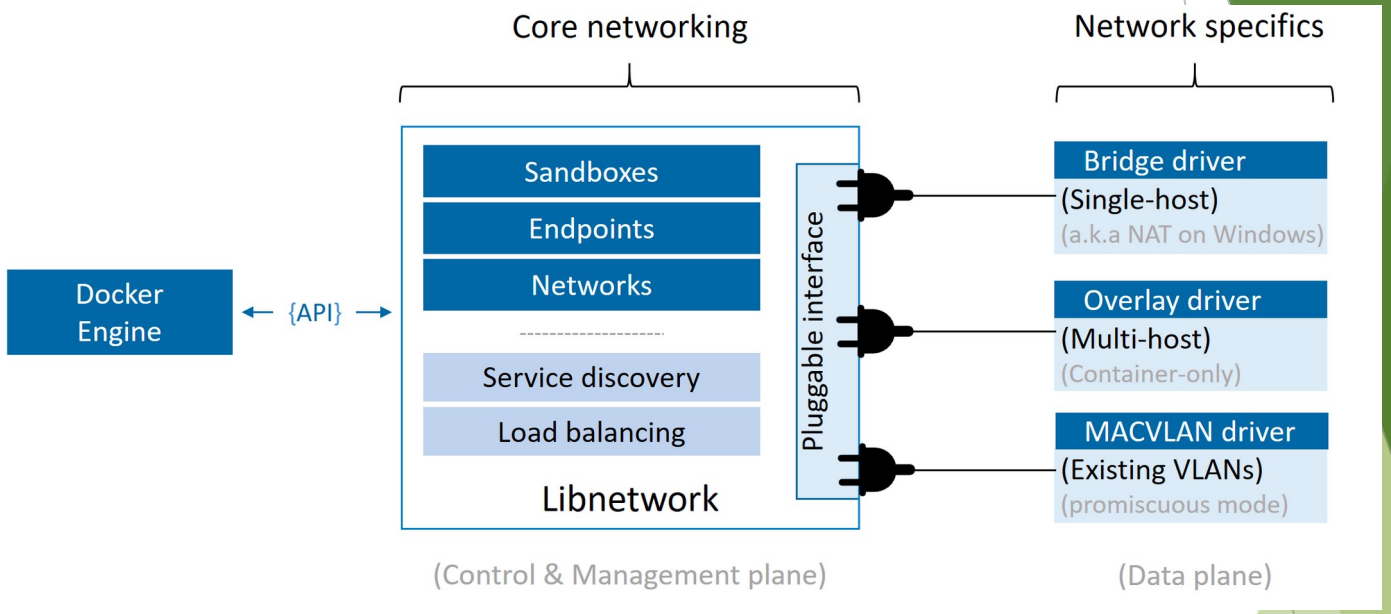
libnetwork

Ejemplo: 2 contenedores en un mismo Host de Docker



Redes en Docker

libnetwork: librería que implementa el plano de control y gestión de las funciones



Añade las funciones de:

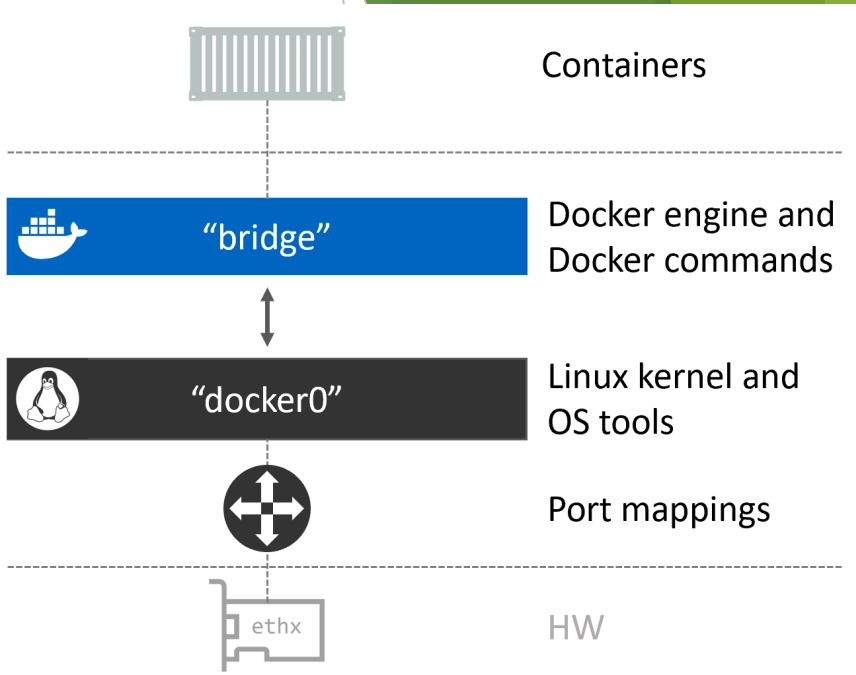
- Service discovery
- Balanceo de carga

Permite tres tipos de redes eligiendo el driver correspondiente. Pueden haber varias redes

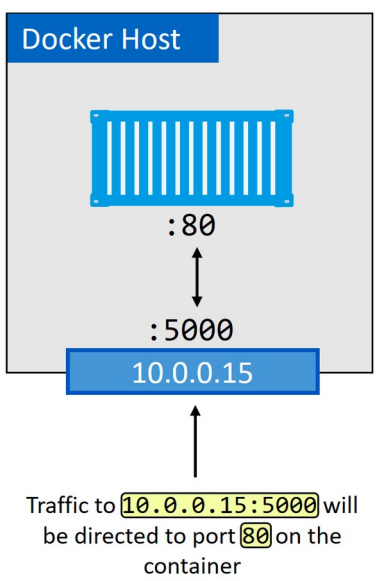
- docker network ls
- docker network inspect → Relacionada con la interfaz docker0
- docker network create -d bridge localnet [docker network create -d nat localnet (W10)]

Redes en Docker

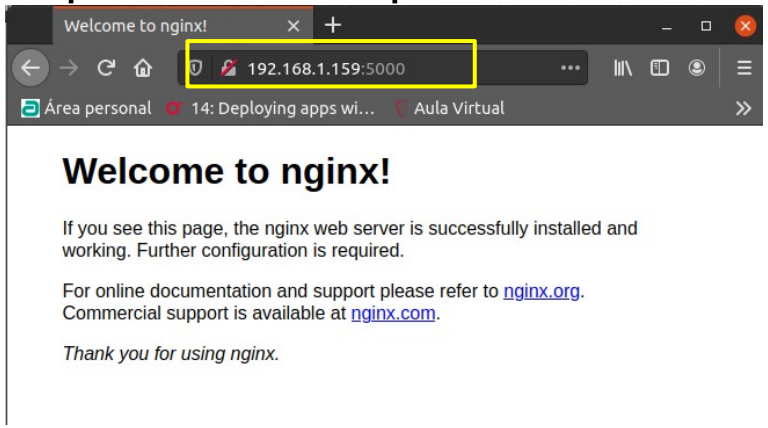
Por defecto se crea red Bridge. → Exponer puertos



docker run --name lco-nginx -d -p 5000:80 nginx:stable



Mapeamos a un puerto del Host



Otra opción:

docker network create -d bridge localnet → Creamos primero una red
docker container run -d --name web --network localnet --publish 5000:80 nginx
docker run -it --name pine --network localnet alpine sh → Por defecto se ven por nombre*

*solo cuando la red no es la opción por defecto

Redes Overlay en Docker

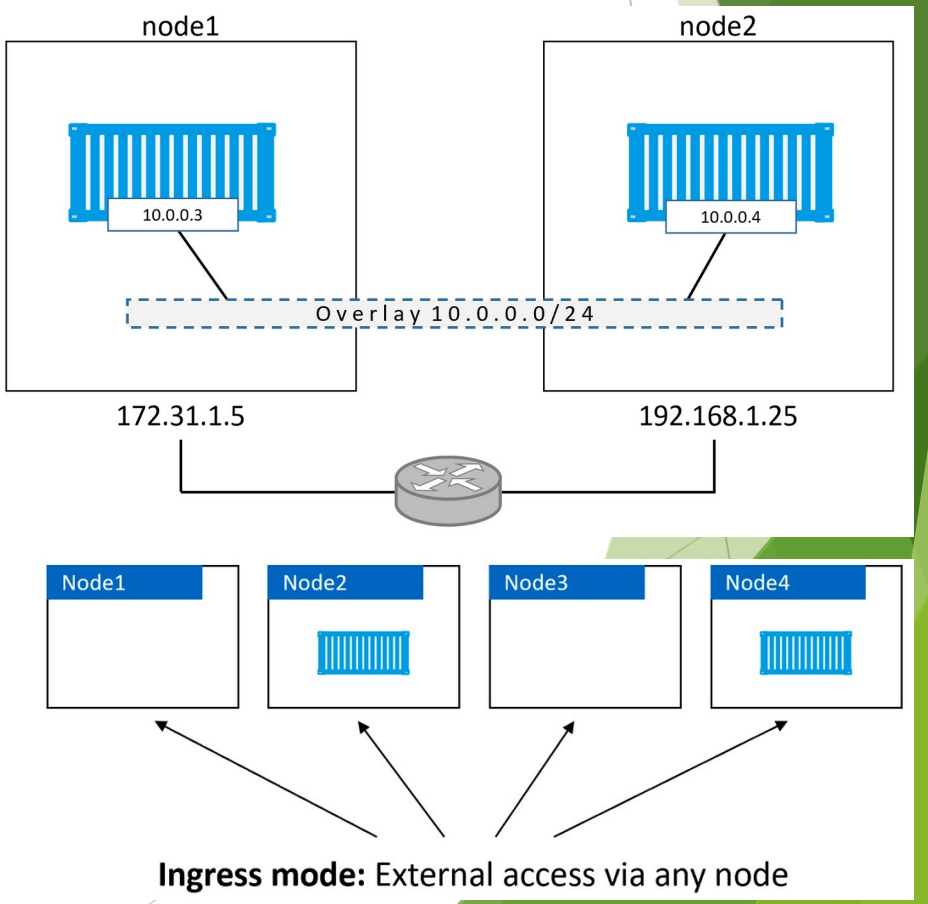
En aplicaciones reales, es necesario que se comuniquen los contenedores desde distintos anfitriones.

Estas redes funcionan con Docker Swarm
Se crea una red de capa 2 virtual que facilita la conexión entre contenedores y oculta la complejidad de la red real subyaciente.
Se utilizan para el despliegue de servicios

Paso 1: Iniciar Swarm
Paso 2: Gestión red overlay

```
docker network create -d overlay uber-net
docker network ls
docker service create --name test \
    --network uber-net --replicas 2 \
    ubuntu sleep infinity
```

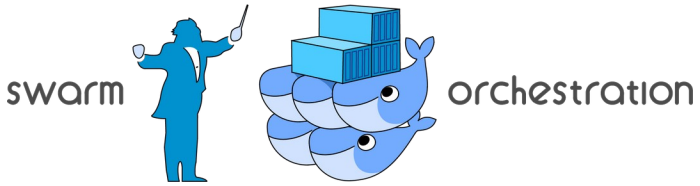
Funcionamiento **ingress load balancing** por defecto
Opciones: ingress/host mode



Swarm
Cluster de contenedores

Creación y Gestión

Swarm en Docker

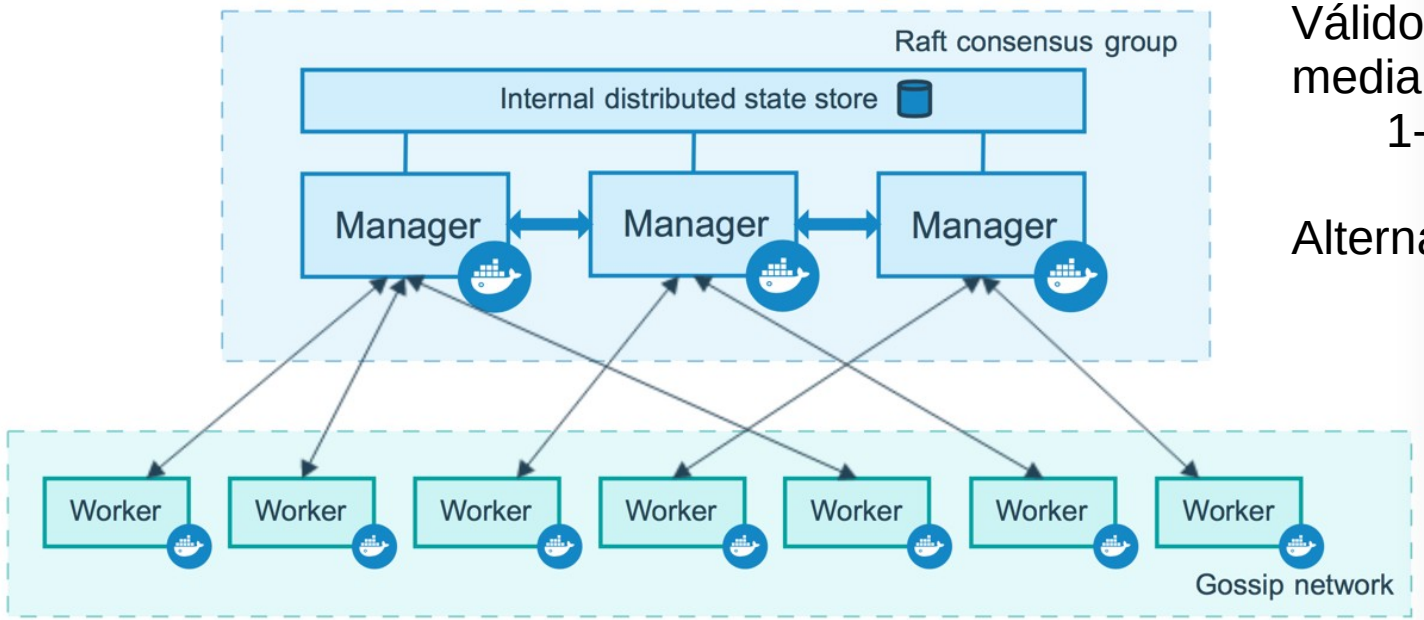


El cluster de contenedores permite el desplieue de contenedores en múltiples equipos host. link

Son la base de los servicios

Proveen balanceo de carga y alta disponibilidad.

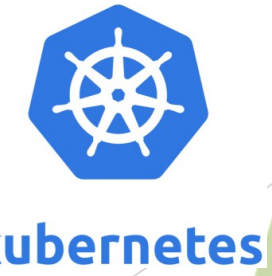
Swarm = conjunto de nodos = manager nodes + worker nodes



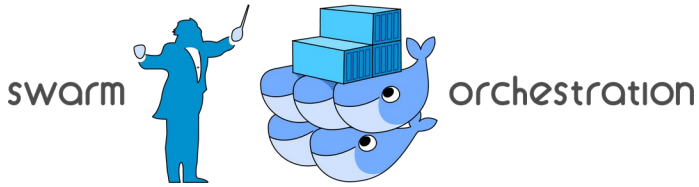
Válido para sistemas pequeños y medianos:

1-3-5-7 nodos de gestión

Alternativa a mayor escala: **Kubernetes**



Swarm en Docker



Gestión del swarm:

1) Iniciar el cluster: [En un nodo que vaya a ser manager]

docker swarm init --advertise-addr 10.0.0.1:2377 --listen-addr 10.0.0.1:2377

#Indicamos la ip del host y el puerto (usado por defecto)

#Nos devuelve el token y la instrucción para añadir nodos

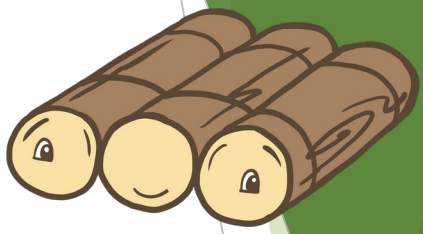
2)Listar los nodos:

docker nodes ls

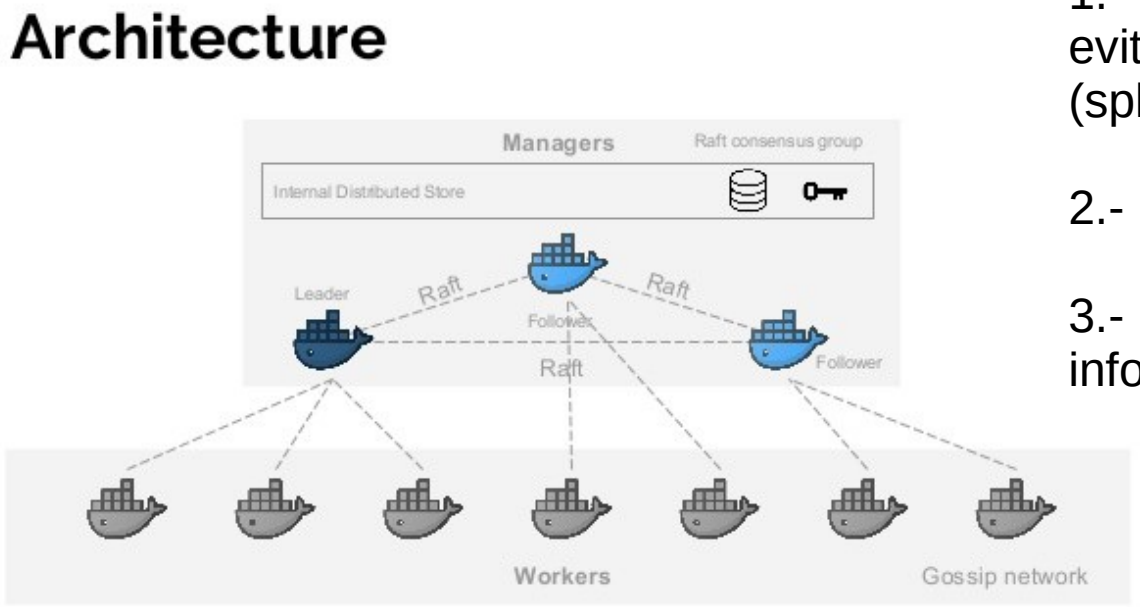
3)Agragar nodos: La sigientes instrucciones para añadir nodos

1) Nodo worker: docker swarm join-token worker
2) Nodo Manager: docker swarm join-token manager

Swarm en Docker



Gestión del swarm: Gestión con el modelo Raft



1.- Número de Managers impar: Para evitar problemas de división del cluster (split brain)

2.- 1-3-5-7 managers: más no es opertivo

3.- Un nodo toma el liderazgo y va informando al resto.

Funciones del cluster:
- Estudia la salud de los nodos y actúa en consecuencia
- Escalado de la capacidad
- Alta disponibilidad
- Red compartida: Cualquier nodo responde con su ip

Docker Service

Creación y Gestión

Docker service

Un servicio requiere:

- Un cluster para funcionar: swarm
- El contenedor del servicio con su imagen asociada en TODOS los nodos
- Una red de acceso [ingress/host based]
- Un proceso de inspección del estado del servicio

Gestión del servicio (link):

- docker service create --name servicio_apache -p 8008:80 –replicas 2 php:apache
- docker service ls // docker service ps servicio_apache
- docker service scale servicio_apache=3
- docker service inspect –pretty servicio_apache #muestra información también de la imagen y se puede cambiar
- docker service update --image php:8.0.1-alpine servicio_apache
- docker service rm servicio_apache

Docker stack

Un stack es una infraestrctura de servicios coordinados entre sí. Necesita por tanto de un cluster donde desplegarse.

Utiliza un fichero muy similar a docker-compose.yaml para lanzar todos los servicios de la infraestrcutura. **NOTA**: Hay pequeñas diferencias.

Gestión del stack (link):

- docker stack deploy -c fichero_despliegue.yaml → Arrancar el stack
- docker stack ls → Lista los stacks en marcha
- docker stack ps pila1 → Lista los servicios del stack con el nombre pila1
- docker stack rm → Para y elimina el stack

Ejemplo AtSea Shop

Creación y Gestión

AtSea shop

Ejemplo App: https://github.com/dockersamples/atsea-sample-shop-app

git clone https://github.com/dockersamples/atsea-sample-shop-app.git