Contenedores - Docker

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2025

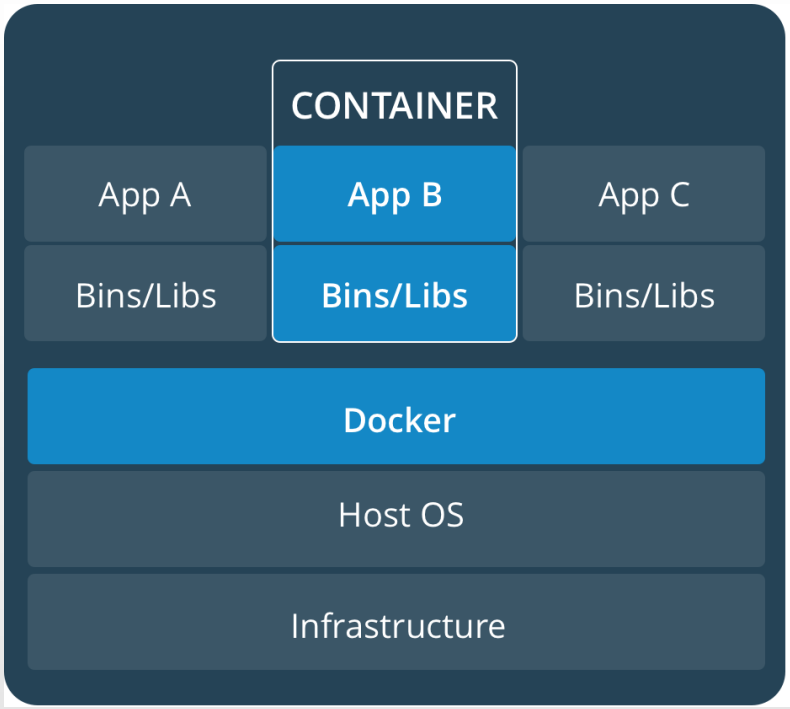
Agenda

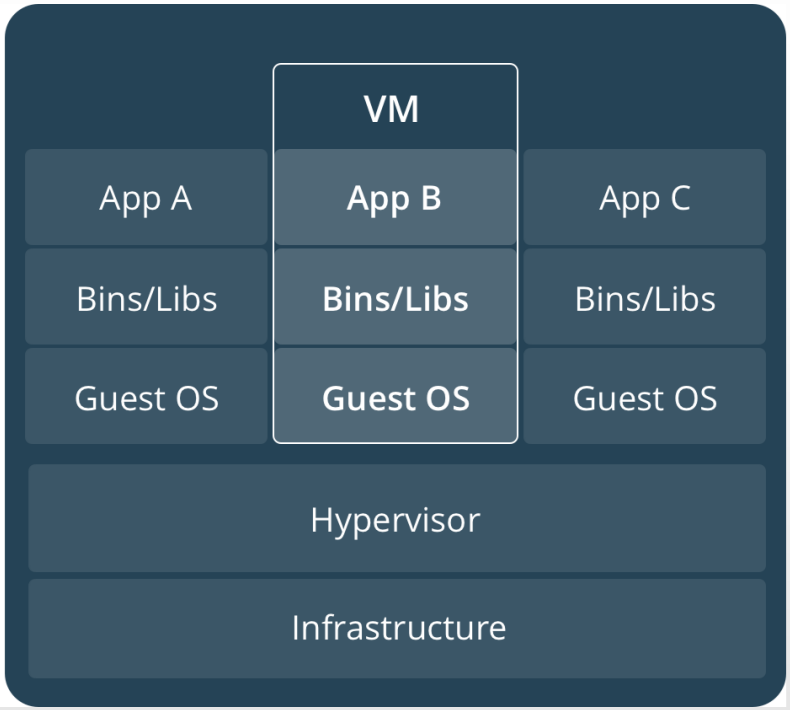
1. Docker

¿Qué es Docker?

- Docker es una plataforma open-source que permite empaquetar y ejecutar una aplicación en containers livianos.
- Docker Engine está dividido en 3 componentes:
  - **Docker Daemon (dockerd):** es el servidor. Responsable por crear, ejecutar y monitorear los contenedores, construcción de imágenes, etc.
  - **API:** especifica la interface que los programas pueden usar para interactuar con el servidor.
  - **CLI:** cliente. Permite a los usuarios interactuar con el servidor mediante comandos.
- Usos posibles:
  - En desarrollo/testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.

Containers vs. VMs





1

[1] https://docs.docker.com/get-started/#containers-and-virtual-machines

¿Qué es Docker?

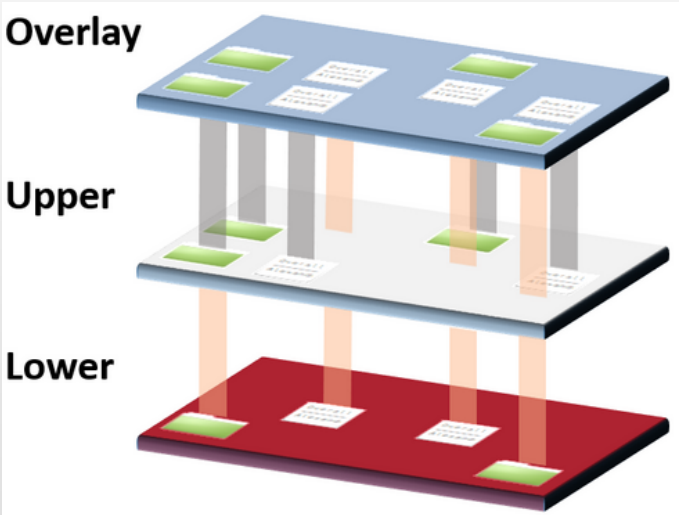
- Utiliza una architectura cliente-servidor.
- Cliente y servidor pueden ejecutar en el mismo sistema (IPC o socket domain) o en diferentes nodos (socket TCP/IP).
- Docker "daemon" escucha por "API requests".
- Se comunican usando la REST API de Docker. Cliente envía comandos HTTP.
- Cliente y servidor se distribuyen como binarios.
- Ambos ejecutan en el espacio del usuario.
- Además de la CLI existe una interface gráfica: *Docker Desktop*.

Docker es una herramienta que utiliza una serie de características del kernel para proveer containers:

- *Namespaces:* Docker lo utiliza para proveer el espacio de trabajo aislado que denominamos container. Por cada container Docker crea un conjunto de espacios de nombres (entre ellos **pid**, **net**, **ipc** y **mnt**).
- *Control groups:* Para, opcionalmente, limitar los recursos asignados a un contenedor.
- *Union file systems:* Se utilizan como filesystem de los containers. Docker puede utilizar **overlay2**, **AUFS**, **btrfs**, **vfs** y **DeviceMapper**(deprecated).

Union Mount FileSystems

- Es un mecanismo de montajes, no un nuevo file system
- Permite que varios directorios sean montados en el mismo punto de montaje, apareciendo como un único file system
- Read-only capas inferiores, writable capa superior
- Existen varias alternativas: UnionFS, AuFS, overlayFS, etc



Ref:
https://www.datalight.com/blog/2016/01/27/explaining-overlayfs- %E2 %80 %93-what-it-does-and-how-it-works/

imagen: template (molde) de sólo lectura con todas las instrucciones para construir un contenedor.

container: Es una instancia de una imagen en ejecución.

registry: Es un almacén de imágenes de Docker. Puede ser público o privado. Por defecto, docker utiliza *Docker Hub*.

Dockerfile: Archivo de texto que indica los pasos necesarios para construir una imagen.

Una imagen puede basarse en otras. Por ejemplo:
httpd → **debian:jessie-backports** → **debian:jessie** → **scratch**
donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.

imagen: template (molde) de sólo lectura con todas las instrucciones para construir un contenedor.

container: Es una instancia de una imagen en ejecución.

registry: Es un almacén de imágenes de Docker. Puede ser público o privado. Por defecto, docker utiliza *Docker Hub*.

Dockerfile: Archivo de texto que indica los pasos necesarios para construir una imagen.

Una imagen puede basarse en otras. Por ejemplo:
httpd → **debian:jessie-backports** → **debian:jessie** → **scratch**
donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.

Imágenes y capas

- Cada imagen está compuesta de una serie de capas que se montan una sobre otra (stacking).
- Cada capa es un conjunto de diferencias con la capa previa.
- Solo la última es R/W (la capa del container). Las demás, de solo lectura (inmutables).
- Capas pueden ser reutilizadas entre las imágenes.
- Capa escribible permite almacenar datos generados durante la ejecución del contenedor.
- Docker usa "storage drivers" para almacenar capas de una imagen y para almacenar datos en la capa escribible de un contenedor.
- Distintos tipos de storage drivers: overlay2, brtfs, zfs, etc. En Windows, windowsfilter.

- Capas pueden ser reutilizadas entre las distintas imágenes.
- Permiten extender las imágenes base de otros agregando nuevos datos
- Apilando las capas:
  - Cada capa que se baja, se extrae el contenido en un directorio del filesystem del nodo.
  - Al ejecutar el contenedor desde una imagen, se genera un union-filesystem donde las capas se apilan una sobre otra.
  - Usando *chroot*, se estable el union-filesystem creado como directorio raíz del contenedor.
  - Por último, se crea un nuevo directorio para el contenedor que permite modificar el filesystem (capa escribible de contenedor)
- Directorio modificable es el que permite correr múltiples contenedores a partir de las mismas capas (uno nuevo por cda contenedor)´

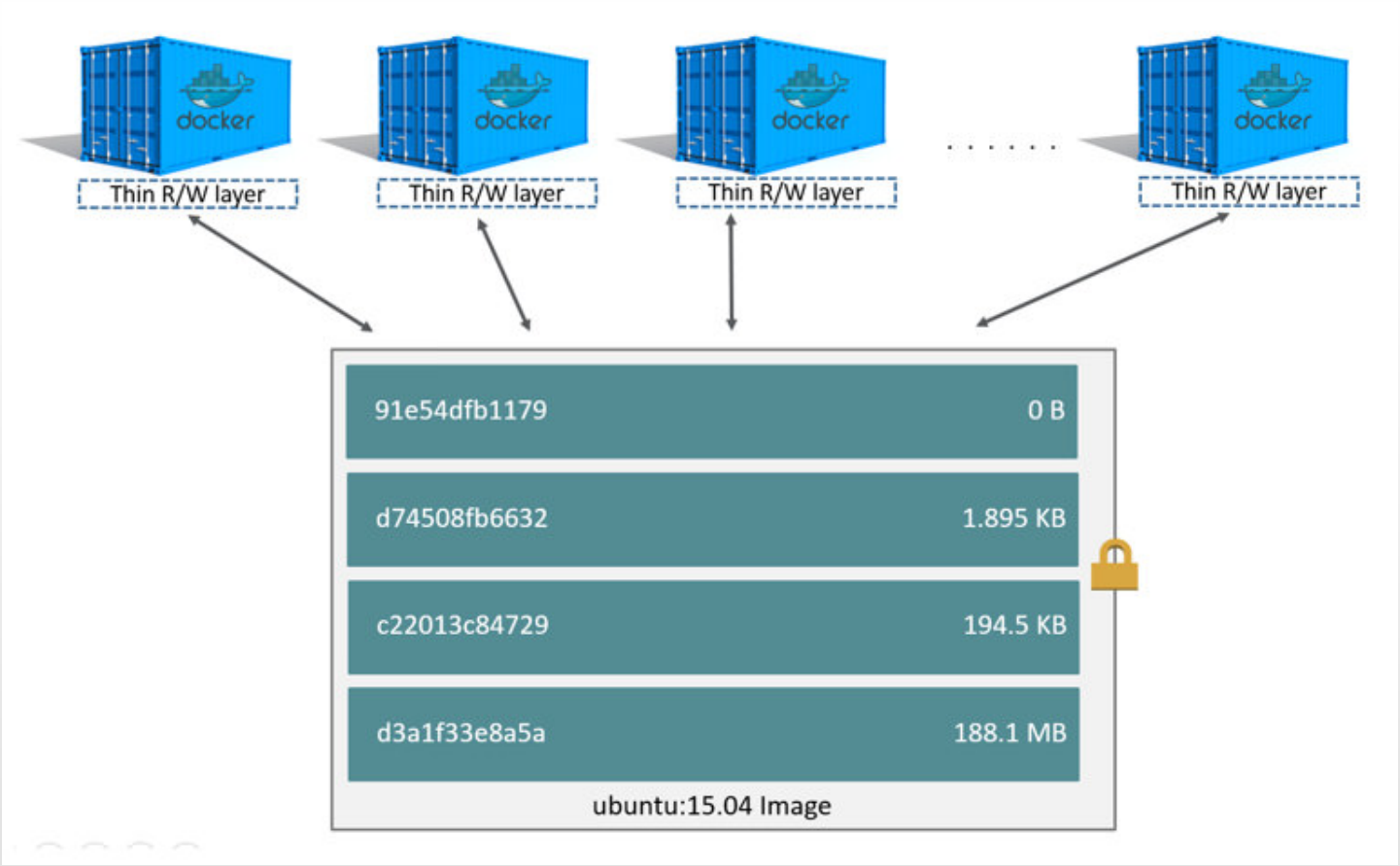
- Imágenes se contruyen siguiendo las instrucciones de un Dockerfile.
- docker build -t nginx-aplicacion:2025.01 .
- docker run -d -p 80:80 –name mi-aplicacion nginx-aplicacion:2025.01

```
FROM ubuntu:24.10
MAINTAINER user1 <user1@example.com>
RUN apt-get -y update && apt-get -y install nginx
COPY index.html /usr/share/nginx/html/
ENV APP_USER=nginx_user
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Cada capa en la imagen representa una instrucción (build instruction) en el archivo Dockerfile.
- Si un comando modifica el filesystem se genera un nueva capa.
- Al remover un archivo (RUN rm -f /home/user1/index.html) se genera una nueva capa. Archivo sigue existiendo en la capa anterior (sigue ocupando espacio).

- Un contenedor es una instancia de una imagen.
- La principal diferencia entre un contenedor y una imagen es la capa *escribible*
- Al eliminar un contenedor esa capa también se elimina. Las capas inferiores se mantienen sin modificaciones.
- Desde una imagen es posible generar varios contenedores.
- Cada contenedor es autónomo y ejecuta en su propio entorno aislado.
- Todos los contenedores comparten el mismo kernel y ejecutan en su propios namespaces.
- Contenedores pueden ser iniciados, detenidos, pausados y destruidos usando la CLI de Docker.

Imágenes y contenedores . . .



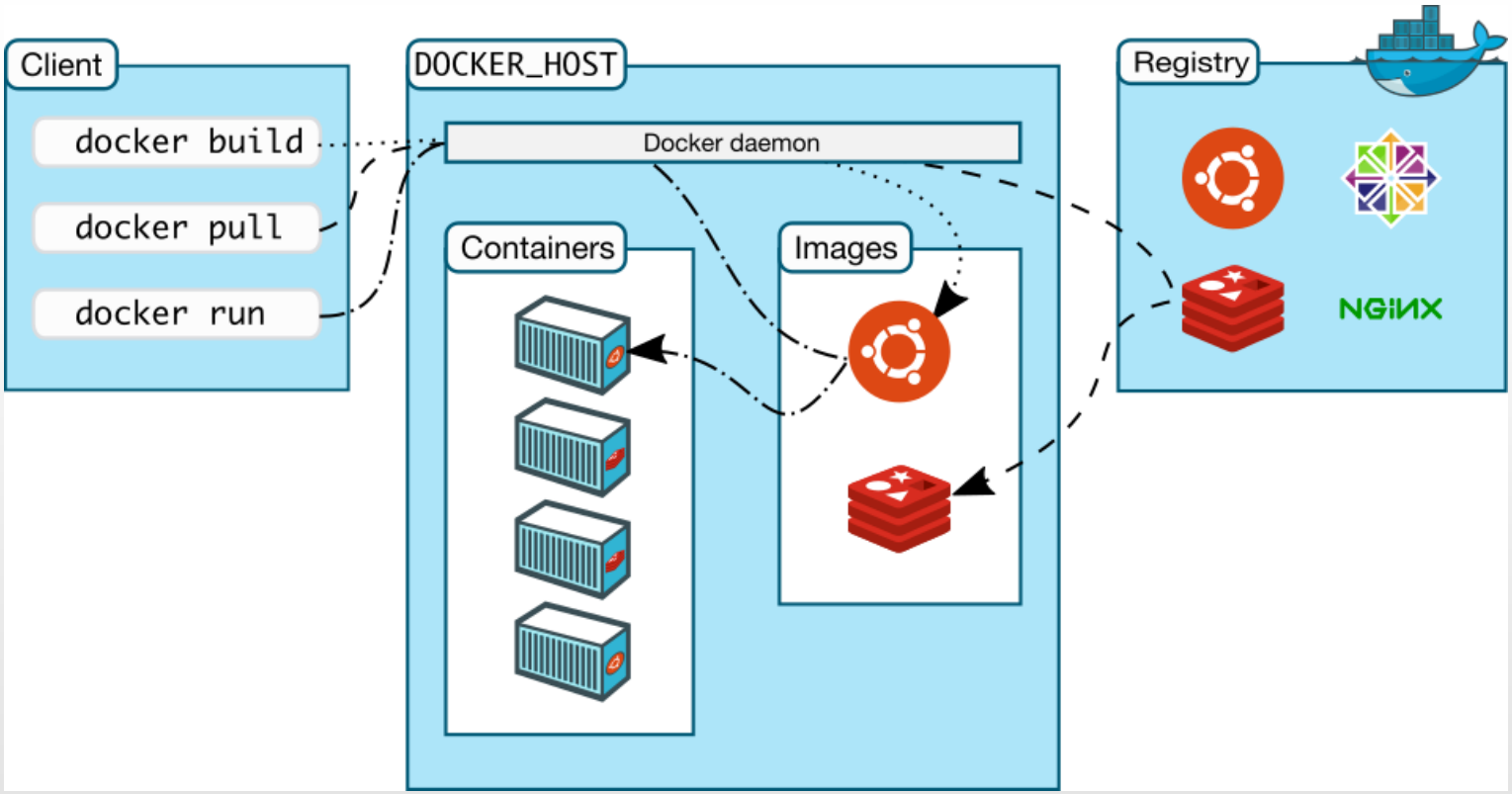
2

[2] https://docs.docker.com/storage/storagedriver/#container-and-layers

Almacenamiento (Storage)

- Archivos creados dentro de un contenedor son almacenados en una capa escribible:
- Datos escritos en el contenedor no persisten cuando es destruido.
- Docker tiene dos opciones para almacenar datos en el host para que sean persistentes:
  - **Volumes:** almacenados en una parte del filesystem administrada por Docker (por default: /var/lib/docker/volumes)
  - **Bind Mounts:** pueden estar en cualquier parte del filesystem. Pueden ser modificados por procesos que no sean de Docker
- Ambos deben ser montados en el contenedor
- Volumes permiten una mejor portabilidad entre sistemas-

- Contenedores tienen el networking habilitado por default (puede deshabilitarse).
- Es posible realizar conexiones salientes.
- Usuarios pueden definir nuevas redes.
- Múltiples contenedores pueden conectarse a la misma red y comunicarse usando direcciones IP y/o nombre.
- Un contenedor se puede conectar a varias redes simultáneamente.
- Para hacer disponible un servicio el contenedor debe publicar el correspondiente puerto.
- Dos contenedores en el mismo host no pueden publicar el mismo puerto.
- Contenedores usan los mismos servidores DNS que el nodo host, pero se pueden modificar.



[3] https://docs.docker.com/engine/docker-overview/#docker-architecture

Ejemplo - Docker, cgroups, namespaces

- Al iniciar el sistema existe al menos un namespace de cada tipo
- Todos los procesos pertenecen a un namespace de cada tipo

```
root@so:~# lsns
NS         TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup     88     1 root /sbin/init
4026531836 pid        88     1 root /sbin/init
4026531837 user       88     1 root /sbin/init
4026531838 uts        88     1 root /sbin/init
4026531839 ipc        88     1 root /sbin/init
4026531840 mnt        86     1 root /sbin/init
4026531860 mnt         1    20 root kdevtmpfs
4026531992 net        88     1 root /sbin/init
4026532147 mnt         1   293 root /lib/systemd/systemd-udevd
```

Ejemplo - Docker, cgroups, namespaces

```
root@so:~# docker run -d --name contenedor1 busybox /bin/sh -c "sleep 5000"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
a58ecd4f0c86: Pull complete
Digest: sha256:9e2bbca079387d7965c3a9cee6d0c53f4f4e63ff7637877a83c4c05f2a666112
Status: Downloaded newer image for busybox:latest
3d6db57dd18c91d694a3e801f2ae41381d6e29a5f3f3a375228e5e096b9640c8
root@so:~# docker image ls
REPOSITORY    TAG         IMAGE ID        CREATED       SIZE
busybox       latest      af2c3e96bcf1    2 days ago    4.86MB
root@so:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
      PORTS     NAMES
3d6db57dd18c   busybox   "/bin/sh -c 'sleep ...5"   33 seconds ago   Up 29 seconds
                 contenedor1
```

Ejemplo - Docker, cgroups, namespaces

```
root@so:~# lsns
NS         TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup     88     1 root /sbin/init
4026531836 pid        87     1 root /sbin/init
4026531837 user       88     1 root /sbin/init
4026531838 uts        87     1 root /sbin/init
4026531839 ipc        87     1 root /sbin/init
4026531840 mnt        85     1 root /sbin/init
4026531860 mnt         1    20 root kdevtmpfs
4026531992 net        87     1 root /sbin/init
4026532147 mnt         1   293 root /lib/systemd/systemd-udevd
4026532236 mnt         1  1722 root /bin/sh -c sleep 5000
4026532237 uts         1  1722 root /bin/sh -c sleep 5000
4026532238 ipc         1  1722 root /bin/sh -c sleep 5000
4026532239 pid         1  1722 root /bin/sh -c sleep 5000
4026532241 net         1  1722 root /bin/sh -c sleep 5000
```

Ejemplo - Docker, cgroups, namespaces

```
root@so:~# docker exec 3d6db57dd18c ls -l /proc/1/ns
total 0
lrwxrwxrwx  1 root  root  0 May 14 01:15 cgroup -> cgroup:[4026531835]
lrwxrwxrwx  1 root  root  0 May 14 01:15 ipc -> ipc:[4026532238]
lrwxrwxrwx  1 root  root  0 May 14 01:15 mnt -> mnt:[4026532236]
lrwxrwxrwx  1 root  root  0 May 14 01:15 net -> net:[4026532241]
lrwxrwxrwx  1 root  root  0 May 14 01:15 pid -> pid:[4026532239]
lrwxrwxrwx  1 root  root  0 May 14 01:15 pid_for_children -> pid
    :[4026532239]
lrwxrwxrwx  1 root  root  0 May 14 01:15 user -> user:[4026531837]
lrwxrwxrwx  1 root  root  0 May 14 01:15 uts -> uts:[4026532237]
```

Ejemplo - Docker, cgroups, namespaces

- PID Namespace: mismo proceso, diferente PID.

```
root@so:~# ps -ef | grep sleep
root      1722  1701  0 21:48 ?        00:00:00 /bin/sh -c sleep 5000
root      1808  1036  0 22:13 pts/0    00:00:00 grep sleep
root@so:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
              PORTS     NAMES
3d6db57dd18c   busybox   "/bin/sh -c 'sleep ...5"   25 minutes ago   Up 25
    minutes              contenedor1
root@so:~# docker exec 3d6db57dd18c ps -ef
PID   USER     TIME   COMMAND
 1    root     0:00   /bin/sh -c sleep 5000
 7    root     0:00   ps -ef
```

Ejemplo - Docker, cgroups, namespaces

```
root@so:/sys/fs/cgroup# docker run -d --memory 256m --name contenedor2 busybox /
      bin/sh -c "sleep 4000"
WARNING: Your kernel does not support swap limit capabilities or the cgroup is
not mounted. Memory limited without swap.
a08cb831097ed4168e7fddb4c8e49ef21d14b9962168aa5708e412a4d0a79bb2
root@so:/sys/fs/cgroup# docker ps
CONTAINER ID    IMAGE      COMMAND                    CREATED          STATUS
              PORTS      NAMES
a08cb831097e    busybox    "/bin/sh -c 'sleep ...4"   7 minutes ago    Up 7 minutes
                  contenedor2
root@so:/sys/fs/cgroup# cd memory/
root@so:/sys/fs/cgroup/memory# ls -l
total 0
-rw-r--r--   1 root root 0 may 13 23:10 cgroup.clone_children
--w--w--w-   1 root root 0 may 13 23:10 cgroup.event_control
-rw-r--r--   1 root root 0 may 13 21:05 cgroup.procs
-r--r--r--   1 root root 0 may 13 23:10 cgroup.sane_behavior
drwxr-xr-x   4 root root 0 may 13 21:06 docker
-rw-r--r--   1 root root 0 may 13 23:10 memory.failcnt
--w-------   1 root root 0 may 13 23:10 memory.force_empty
-rw-r--r--   1 root root 0 may 13 23:10 memory.kmem.failcnt
.......
-rw-r--r--   1 root root 0 may 13 21:05 memory.limit_in_bytes
.....
```

Ejemplo - Docker, cgroups, namespaces

```
root@so:/sys/fs/cgroup/memory/docker/
      a08cb831097ed4168e7fddb4c8e49ef21d14b9962168aa5708e412a4d0a79bb2#
ps -ef | grep sleep
root       1722   1701   0 21:48 ?        00:00:00 /bin/sh -c sleep 5000
root       2529   2508   0 23:10 ?        00:00:00 /bin/sh -c sleep 4000
root       2574   1036   0 23:11 pts/0    00:00:00 grep sleep
root@so:/sys/fs/cgroup/memory/docker/
      a08cb831097ed4168e7fddb4c8e49ef21d14b9962168aa5708e412a4d0a79bb2# more
      cgroup.procs
2529
root@so:/sys/fs/cgroup/memory/docker/
      a08cb831097ed4168e7fddb4c8e49ef21d14b9962168aa5708e412a4d0a79bb2# more
      memory.limit_in_bytes
268435456
```

```
# Descargar imagen de Apache de DockerHUB
docker pull httpd
# Ejecutar imagen
docker run httpd
# Crear una imagen a partir de un Dockerfile
docker image build -t NOMBRE_TAG .
# Ejecutar la imagen creada
docker run NOMBRE_TAG
# Subir la imagen a DockerHUB
# antes hay que ejecutar `docker login`
docker push NOMBRE_TAG\
        USUARIO_DOCKERHUB/REPOSITORIO
```

```
# Información general y configuración
docker info
# Containers en ejecución
docker ps
# Imágenes y containers
docker image ls
docker container ls -a
# Ejecutar el container de Ubuntu en modo
    interactivo (bash)
docker pull ubuntu &&\
    docker run -v ./dir_comp:/mnt -it ubuntu
# Crear una nueva imagen con los cambios del
    container
docker commit CONTAINER REPOSITORY:TAG
```

Referencias

- https://docs.docker.com/engine/docker-overview/#docker-objects
- https://docs.docker.com/engine/reference/commandline/
- https://medium.com/@nagarwal/understanding-the-docker-internals-7ccb052ce9fe
- http://docker-saigon.github.io/post/Docker-Internals/
- https://www.safaribooksonline.com/library/view/using-docker/9781491915752/
- https://washraf.gitbooks.io/the-docker-ecosystem

¿Preguntas?