Control Groups, Namespaces, Docker

Explicación de práctica 4

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2024

Agenda

1. Linux cgroups y Namespaces
2. Containers - Docker

Agenda

1. Linux cgroups y Namespaces
2. Containers - Docker

Service Isolation - chroot

- Chroot es una forma de aislar aplicaciones del resto del sistema
- Introducido en la versión 7 de UNIX, 1979
- Cambia el directorio raíz aparente de un proceso. Afecta sólo a ese proceso y a sus procesos hijos
- Al entorno virtual creado por chroot a partir de la nueva raíz del sistema se le conoce como “jail chroot”
- No se puede acceder a archivos y comandos fuera de ese directorio
- “chroot /new-root-dir comando”
- chroot /usr/local/so ls /tmp

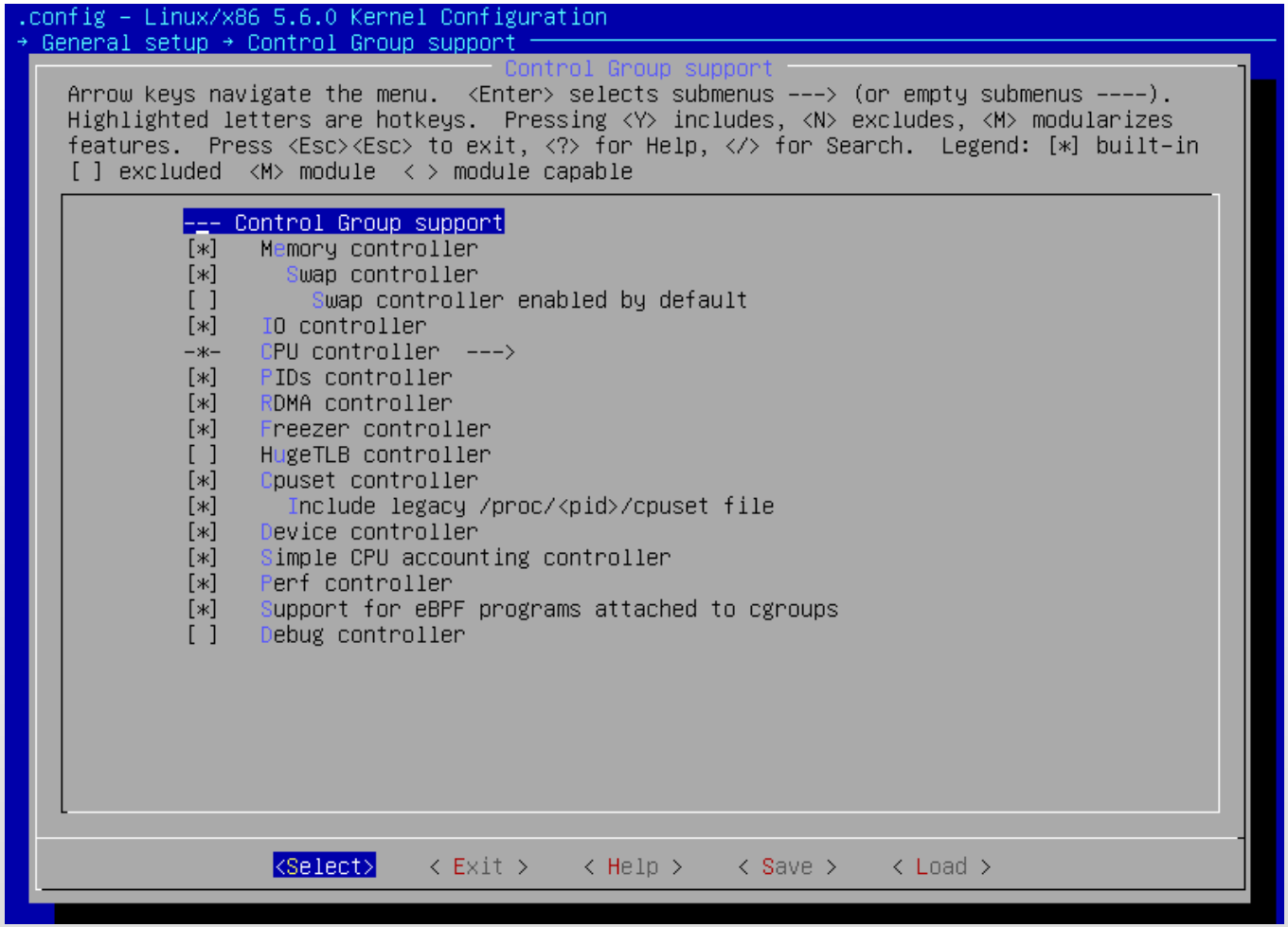
- En un sistema operativo se ejecutan varios procesos en forma concurrente
- En Linux, por defecto, todos los procesos reciben el mismo trato en lo que respecta a tiempo de CPU, memoria RAM, "I/O bandwidth".
- ¿Qué sucede si se tiene un proceso importante que requiere prioridad? O, ¿como limitar los recursos para un proceso o grupo de procesos?
- El kernel no puede determinar cual proceso es importante y cual no
- Existen algunas herramientas, como Nice, CPULimit o ulimit, que permiten controlar el uso de los recursos por un proceso, pero, ¿son suficientes?

Control Groups

- Control Groups, *cgroups*, son una característica del kernel de Linux que permite que los procesos sean organizados en grupos jerárquicos cuyo uso de varios tipos de recursos (CPU, memoria, I/O, etc.) pueda ser limitado y monitoreado.
- Desarrollo comenzado en Google por Paul Menage y Rohit Seth en el 2006 bajo el nombre de “process containers”
- Renombrado en 2007 como “Control Groups”. Disponible desde la versión del kernel 2.6.24
- Actualmente, Tejun Heo es el encargado del desarrollo y mantenimiento de CG.
- Versión 2 de CG con el Linux Kernel 4.5 de Marzo de 2016. Ambas versiones se habilitan por defecto
- La interface de cgroups del kernel es provista mediante un pseudo-filesystem llamado *cgroups*

- Permiten un control "fine-grained" en la alocación, priorización, denegación y monitoreo de los recursos del sistema
- cgroups provee lo siguiente:
  - **Resource Limiting:** grupos no pueden excederse en la utilización de un recurso (tiempo de CPU, cantidad de CPUs, cantidad de memoria, I/O, etc.)
  - **Prioritization:** un grupo puede obtener prioridad en el uso de los recursos (tiempo de CPU, I/O, etc.)
  - **Accounting:** permite medir el uso de determinados recursos por parte de un grupo (estadísticas, monitoreo, billing, etc.)
  - **Control:** permite freezar y reiniciar un grupo de procesos
- Procesos desconocen los límites aplicados por un "cgroup"

- Actualmente hay 12 subsistemas definidos



- cgroups v1
  - Distintos controladores se han ido agregando en el tiempo para permitir la administración de distintos tipos de recursos
  - En cgroups v1, desarrollo de los controladores fue muy descoordinado
  - Administración de las distintas jerarquías se hizo cada vez más complejo
  - Diseño posterior a la implementación
- cgroups v2
  - cgroups v2 pensado como un reemplazo de cgroups v1
  - Controladores también agregados en el tiempo
- Ambos controladores pueden ser montados en el mismo sistema
- Una jerarquía de un controlador no puede estar en ambos cgroups simultáneamente

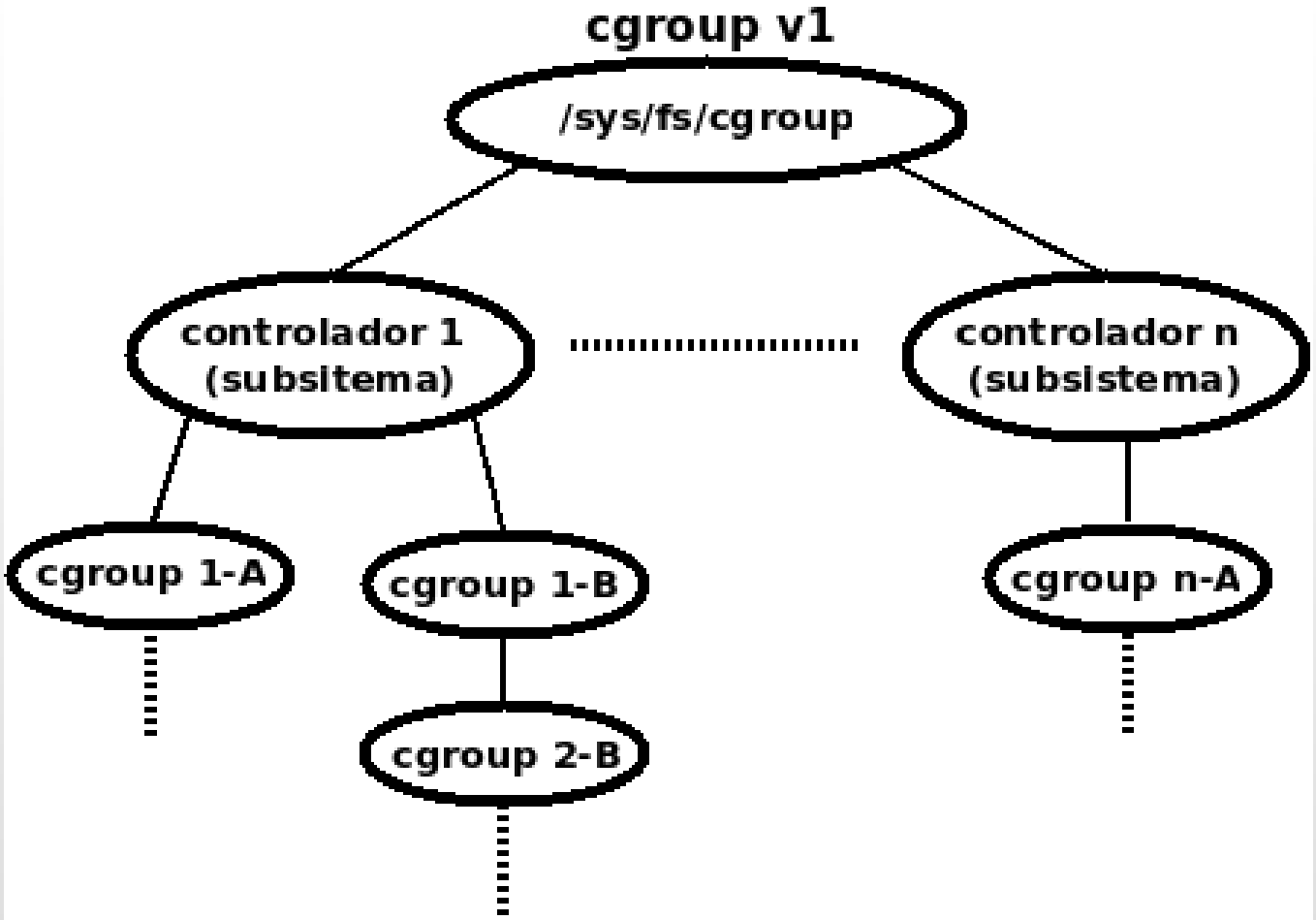
- **cgroup:** asocia un conjunto de tareas con un conjunto de parámetros o límites para uno o más subsistemas.
- **Subsistema:** componente del kernel que modifica el comportamiento de los procesos en un cgroup. También llamado *resource controllers* o simplemente *controllers*.
- **Jerarquía:** es un conjunto de cgroups organnizados en un árbol.
- Cada subsistema representa un único recurso: tiempo de CPU, memoria, I/O, etc.
- Cada jerarquía es definida mediante la creación, eliminación y renombrado de subdirectorios dentro del pseudo-filesystem
- Cada proceso del sistema solo puede pertenecer a un cgroup dentro de una jerarquía (pero a varias jerarquías)

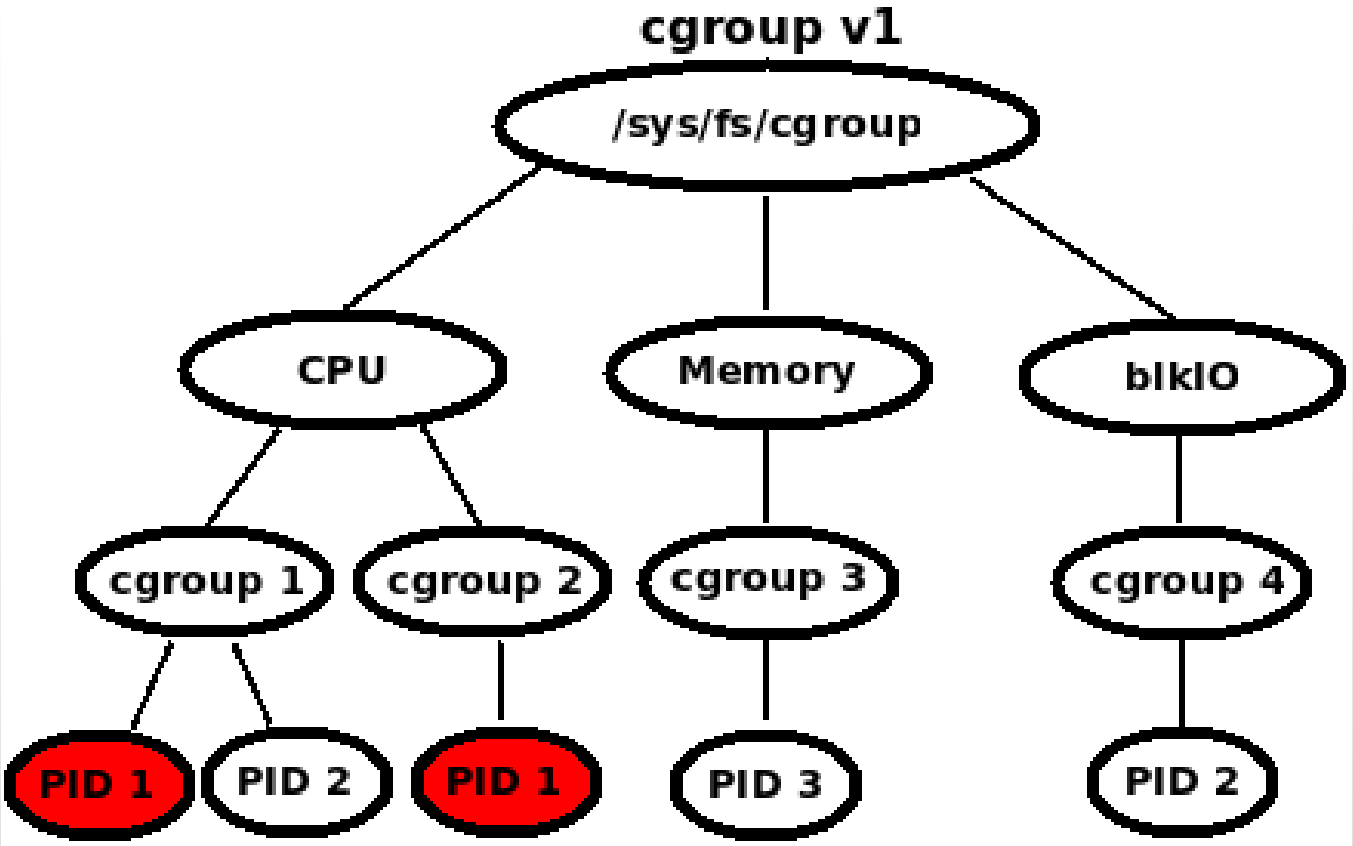
cgroup v1 - Características

- Controladores pueden ser montados en pseudo-filesystems individuales o en un mismo pseudo-filesystem.
- Por cada jerarquía, la estructura de directorios refleja la dependencia de los cgroups.
- Cada cgroup es representado por un directorio en una relación padre-hijo. Por ejemplo: cpu/procesos/proceso1.
- En cada nivel de la jerarquía se pueden definir atributos (por ej. límites). No pueden ser excedidos por los cgroups hijos.
- Un proceso creado mediante un "fork"pertenece al mismo cgroup que el padre.
- Cada directorio contiene archivos que pueden ser escritos/leídos.
- Una vez definidos los grupos se le agregan los IDs de procesos.
- Posible asignar threads de un proceso a diferentes cgroups. Deshabilitado en la v2, restaurado luego, pero con limitaciones

cgroup v1 - Características

- Un controlador v1 debe ser montado contra un filesystem de tipo cgroup. Usualmente es mediante un filesystem *tmpfs* montado en /sys/fs/cgroup.
- *mount -t cgroup -o cpu none /sys/fs/cgroup/cpu* para montar un controlador en particular (CPU en este caso)
- *mount -t cgroup -o all cgroup /sys/fs/cgroup* para montar todo los controladores
- Un controlador puede ser desmontado si no está ocupado: no tiene cgroups hijos (umount /sys/fs/cgroup/cpu)
- Cada cgroup filesystem contiene un único cgroup raíz al cual pertenecen todos los procesos
- Un proceso creado mediante un "fork" pertenece al mismo cgroup que el padre
- *libcgroup:* tools para administrar los cgroups


cgroup v1
/sys/fs/cgroup
controlador 1
(subsitema)
controlador n
(subsistema)
cgroup 1-A
cgroup 1-B
cgroup n-A
cgroup 2-B


cgroup v1
/sys/fs/cgroup
CPU
Memory
blkIO
cgroup 1
cgroup 2
cgroup 3
cgroup 4
PID 1
PID 2
PID 1
PID 3
PID 2

cgroup v1 - Características

```
root@so2020:~# mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=100824k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=sy
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuac
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
....
```

cgroup v1 - Características

```
root@so2020:/sys/fs/cgroup# ls -l
total 0
dr-xr-xr-x 2 root root  0 abr 20 08:28 blkio
lrwxrwxrwx 1 root root 11 abr 20 08:28 cpu -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 abr 20 08:28 cpuacct -> cpu,cpuacct
dr-xr-xr-x 2 root root  0 abr 20 08:28 cpu,cpuacct
dr-xr-xr-x 2 root root  0 abr 20 08:28 cpuset
dr-xr-xr-x 4 root root  0 abr 20 08:28 devices
dr-xr-xr-x 2 root root  0 abr 20 08:28 freezer
dr-xr-xr-x 4 root root  0 abr 20 08:28 memory
lrwxrwxrwx 1 root root 16 abr 20 08:28 net_cls -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 abr 20 08:28 net_cls,net_prio
lrwxrwxrwx 1 root root 16 abr 20 08:28 net_prio -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 abr 20 08:28 perf_event
dr-xr-xr-x 4 root root  0 abr 20 08:28 pids
dr-xr-xr-x 2 root root  0 abr 20 08:28 rdma
dr-xr-xr-x 5 root root  0 abr 20 08:28 systemd
dr-xr-xr-x 5 root root  0 abr 20 08:28 unified
```

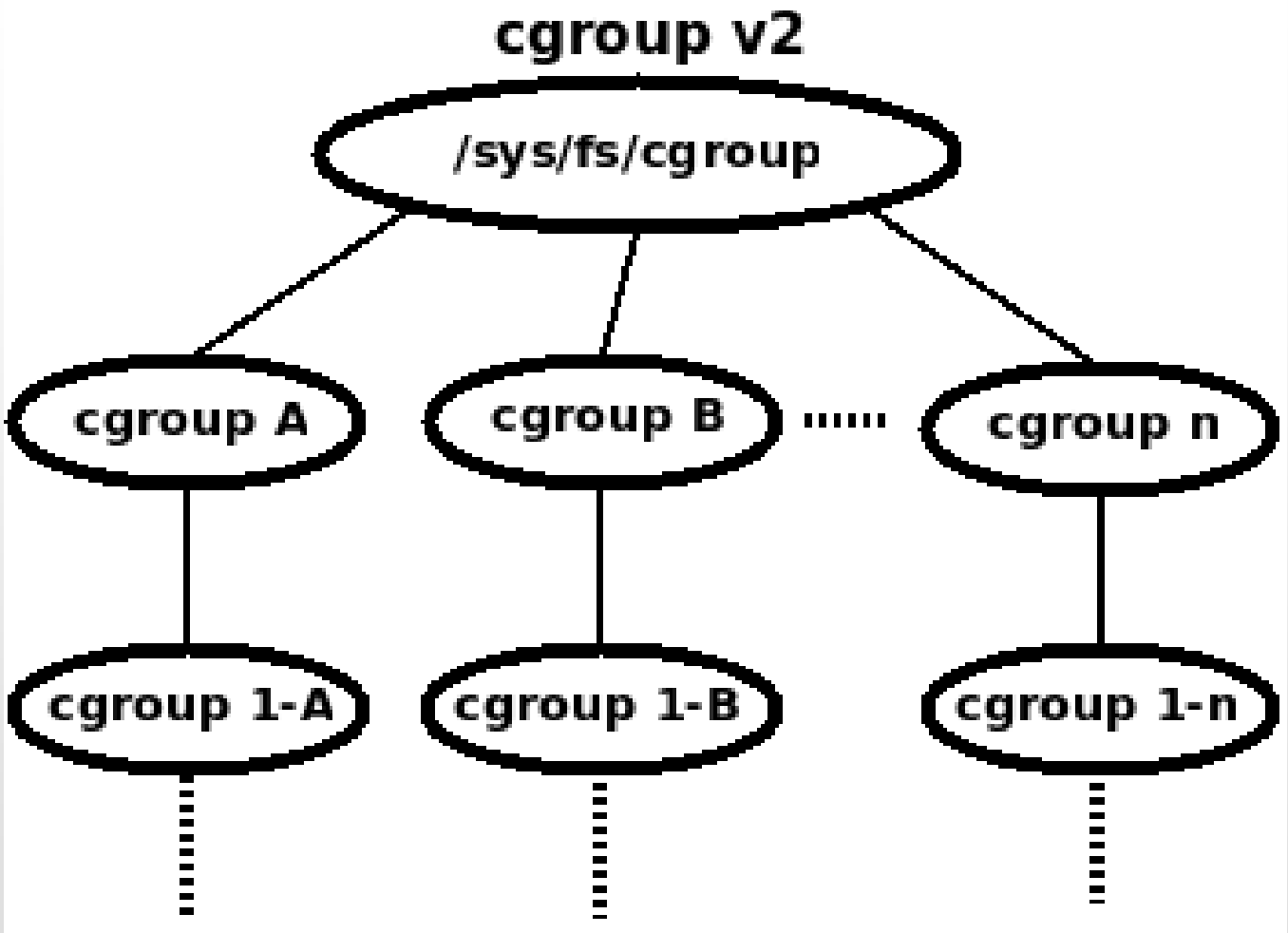
- cgcreate, o mkdir dentro de la estructura, para crear un cgroup
- Systemd alternativa para administra los cgroups

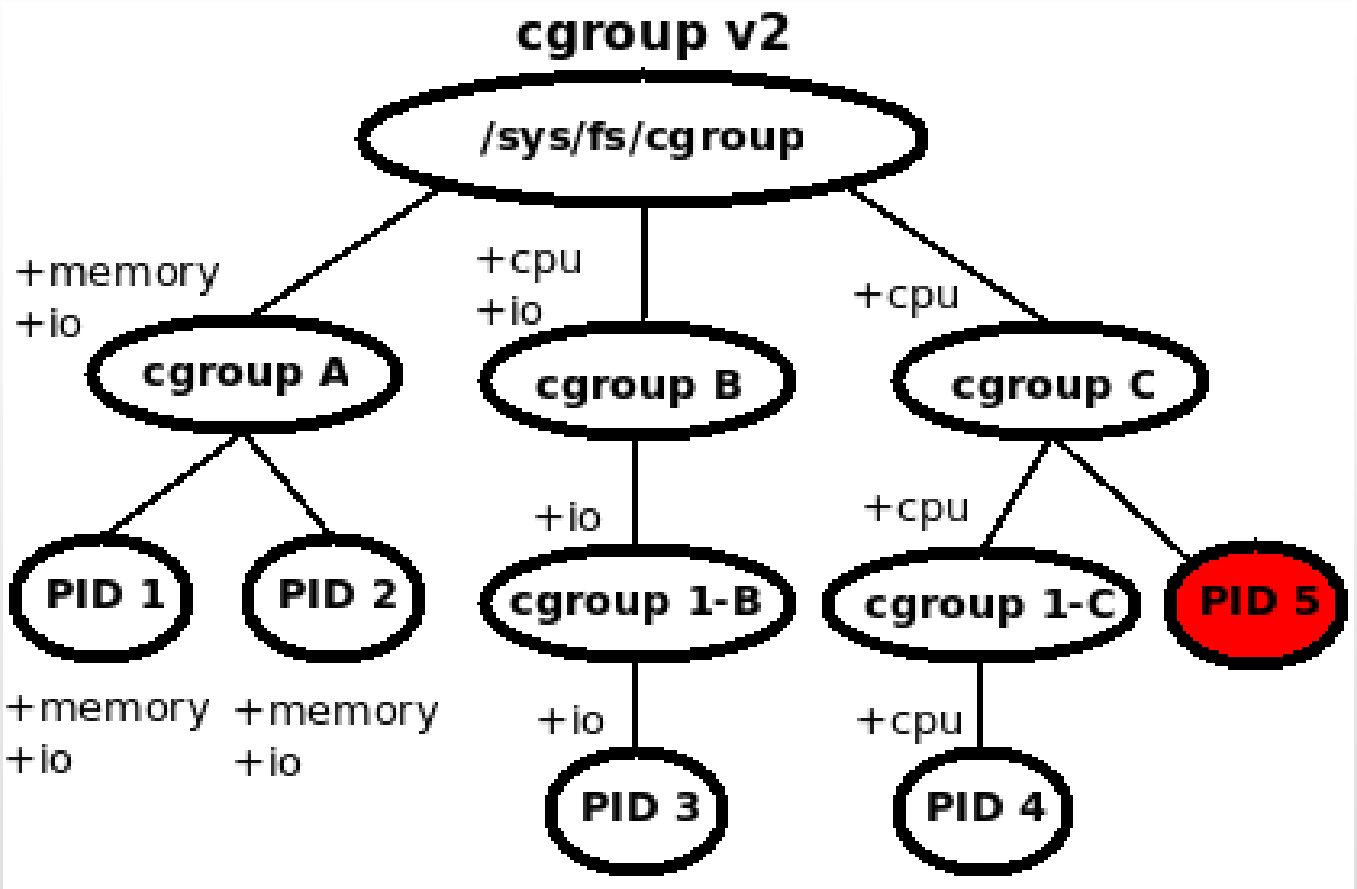
```
so@so:/$sudo cgcreate -g cpuset:my_group
so@so:/$ ls -l /sys/fs/cgroup/cpuset/my_group/
total 0
-rw-r--r-- 1 root root 0 jun  5 23:18 cgroup.clone_children
-rw-r--r-- 1 root root 0 jun  5 23:18 cgroup.procs
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.cpu_exclusive
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.cpus
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.mem_exclusive
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.mem_hardwall
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.memory_migrate
-r--r--r-- 1 root root 0 jun  5 23:18 cpuset.memory_pressure
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.memory_spread_page
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.memory_spread_slab
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.mems
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.sched_load_balance
-rw-r--r-- 1 root root 0 jun  5 23:18 cpuset.sched_relax_domain_level
-rw-r--r-- 1 root root 0 jun  5 23:18 notify_on_release
-rw-r--r-- 1 root root 0 jun  5 23:18 tasks
```

- echo "0-2,4" > /sys/fs/cgroup/cpuset/my_group/cpuset.cpus
- echo "PID" > /sys/fs/cgroup/cpuset/my_group/cgroup.procs

- Todos los controladores son montados en una jerarquía unificada
- No se permiten procesos internos, a excepción del cgroup root. Procesos deben asignarse a cgroups sin hijos (hojas)
- No es posible especificar un controlador en particular para montar
- *mount -t cgroup2 none /mnt/cgroup2*
- Todos los controladores son equivalente a los de cgroups v1, excepto *net_cls* y *net_prio*
- Cada cgroup en la jerarquía v2 contiene, entre otros, los siguiente archivos:
  - **c**group.controllers: archivo de solo lectura que indica los controladores disponibles en este cgroup
  - **c**group.subtree_control: lista de controladores habilitados en el cgroup.

- Conjunto de controladores activos es modificado escribiendo el nombre de los controladores en el archivo correspondiente. + para habilitar, - para deshabilitar
- *echo '+pids -memory' > cgroup.subtree_control*
- cgroup.subtree_control determina el conjunto de controladores que son empleados en los cgroups hijos.
- Procesos solo en las hojas. Por ej. si /procesos/proc1 son cgroups entonces un proceso puede residir en /procesos/proc1, pero no en /procesos.
- Inicialmente, solo el cgroup root existe y todos los procesos pertenecen a él
- mkdir CG_NAME y rmdir CG_NAME para crear y eliminar cgroups.
- Cada cgroup tiene un archivo lectura/escritura *cgroup.procs*


cgroup v2
/sys/fs/cgroup
cgroup A
cgroup B
cgroup n
cgroup 1-A
cgroup 1-B
cgroup 1-n


cgroup v2
/sys/fs/cgroup
+memory
+io
+cpu
+io
+cpu
cgroup A
cgroup B
cgroup C
+io
+cpu
PID 1
PID 2
cgroup 1-B
cgroup 1-C
PID 5
+memory
+io
+memory
+io
+io
+cpu
PID 3
PID 4

```
root@so:/sys/fs/cgroup# ls -l
total 0
-r--r--r--  1 root root 0 abr 20 08:16 cgroup.controllers
-rw-r--r--  1 root root 0 abr 20 08:20 cgroup.max.depth
-rw-r--r--  1 root root 0 abr 20 08:20 cgroup.max.descendants
-rw-r--r--  1 root root 0 abr 20 08:16 cgroup.procs
-r--r--r--  1 root root 0 abr 20 08:20 cgroup.stat
-rw-r--r--  1 root root 0 abr 20 08:16 cgroup.subtree_control
-rw-r--r--  1 root root 0 abr 20 08:20 cgroup.threads
-r--r--r--  1 root root 0 abr 20 08:20 cpuset.cpus.effective
-r--r--r--  1 root root 0 abr 20 08:20 cpuset.mems.effective
drwxr-xr-x  2 root root 0 abr 20 08:16 init.scope
drwxr-xr-x 19 root root 0 abr 20 08:16 system.slice
drwxr-xr-x  3 root root 0 abr 20 08:18 user.slice
```

Namespace Isolation

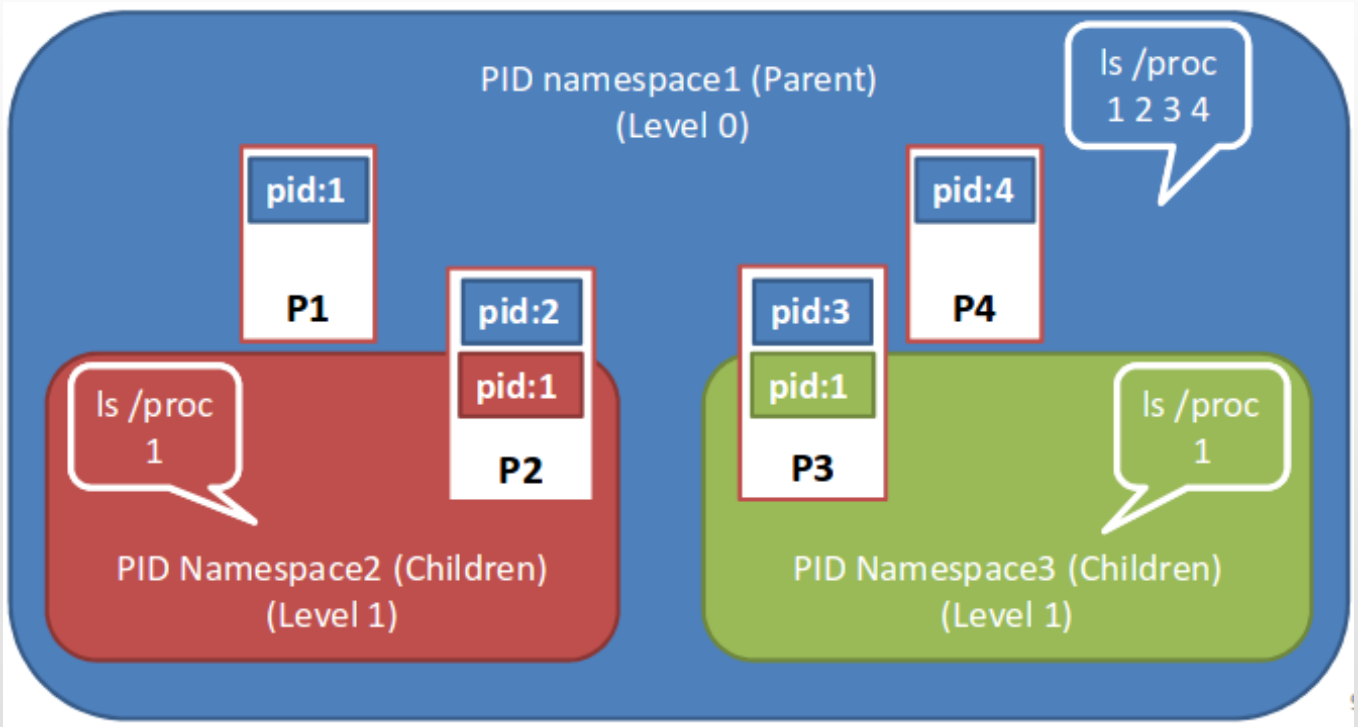
Permite abstraer un recurso global del sistema para que los procesos dentro de ese "namespace" piensen que tienen su propia instancia aislada de ese recurso global

- Limitan lo que un proceso puede ver (y en consecuencia lo que puede usar)
- Modificaciones a un recurso quedan contenidas dentro del "namespace"
- Un proceso solo puede estar en un namespace de un tipo a la vez
- Un namespace es automáticamente eliminado cuando el último proceso en él finaliza o lo abandona
- Un proceso puede utilizar ninguno/algunos/todos de los namespace de sus padre.

- Entre los namespaces provistos por Linux:
  - **IPC:** Flag: CLONE_NEWIPC. System V IPC, cola de mensaje POSIX
  - **Network:** Flag: CLONE_NEWNET. Dispositivos de red, pilas, puertos, etc
  - **Mount:** Flag: CLONE_NEWNS. Puntos de montaje
  - **PID:** Flag: CLONE_NEWPID. IDs de procesos
  - **User:** Flag: CLONE_NEWUSER. IDs de usuarios y grupos
  - **UTS:** Flag: CLONE_NEWUTS. HostName y nombre de dominio
  - **Cgroup:** Flag: CLONE_NEWCGROUP. Cgroup root directory
  - **Time:** Flag: CLONE_NEWTIME. Distintos offsets al clock del sistema por namespace (Marzo 2020)
- Flag: usado para indicar el namespace en las system calls

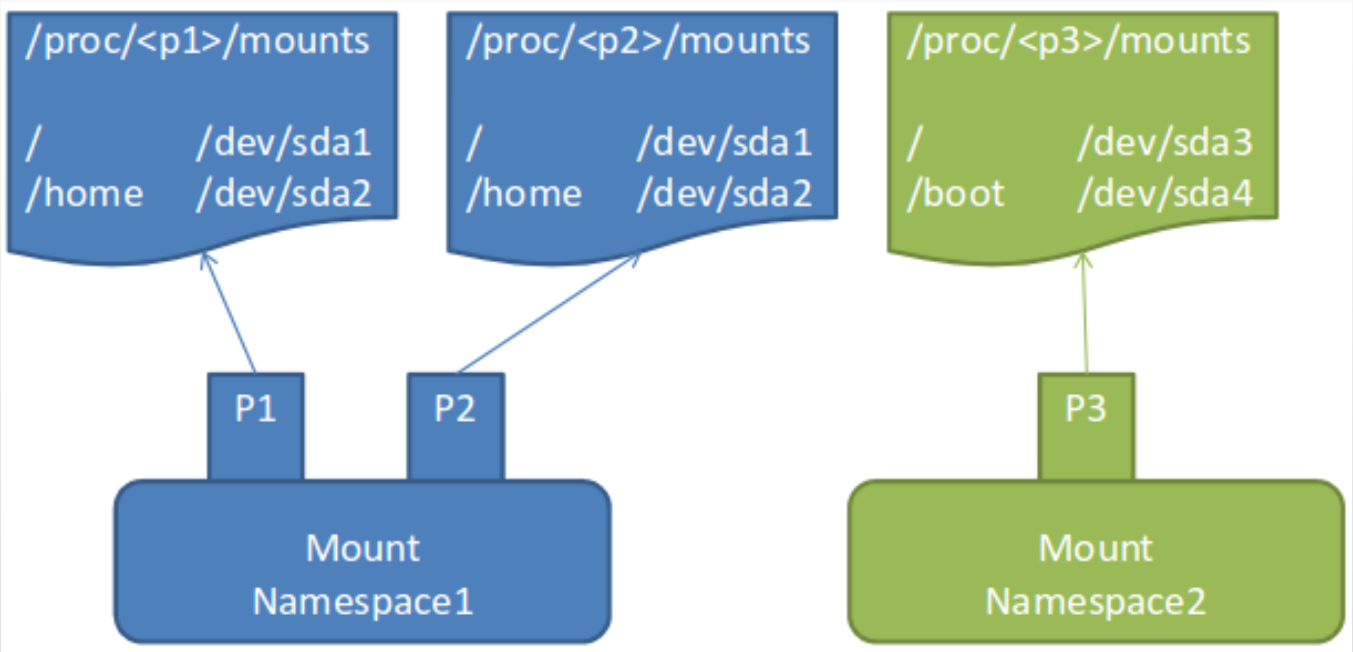
- Tres nuevas systems-calls:
  - **clone():** similar al fork. Crea un nuevo proceso y lo agrega al nuevo namespace especificado. Su funcionalidad puede ser controlada por flags pasados como argumentos
  - **unshare():** agrega el actual proceso a un nuevo namespace. Es similar a clone, pero opera en al proceso llamante. Crea el nuevo namespace y hace miembro de él al proceso llamador
  - **setns():** agrega el proceso actual a un namespace existente. Desasocia al proceso llamante de una instancia de un tipo de namespace y lo reasocia con otra instancia del mismo tipo de namespace
- Cada proceso tiene un subdirectorio que contiene los namespaces a los que está asociado: */proc/[pid]/ns*
- *unshare también es un herramienta de Linux*

- Posibilidad de tener múltiples árboles de procesos anidados y aislados



Fuente imagen: http://events.linuxfoundation.org/sites/events/files/cojp13_feng.pdf

- Permite aislar la tabla de montajes (montajes por namespace)
- Cada proceso, o conjunto de procesos, tiene una vista distinta de los puntos de montajes



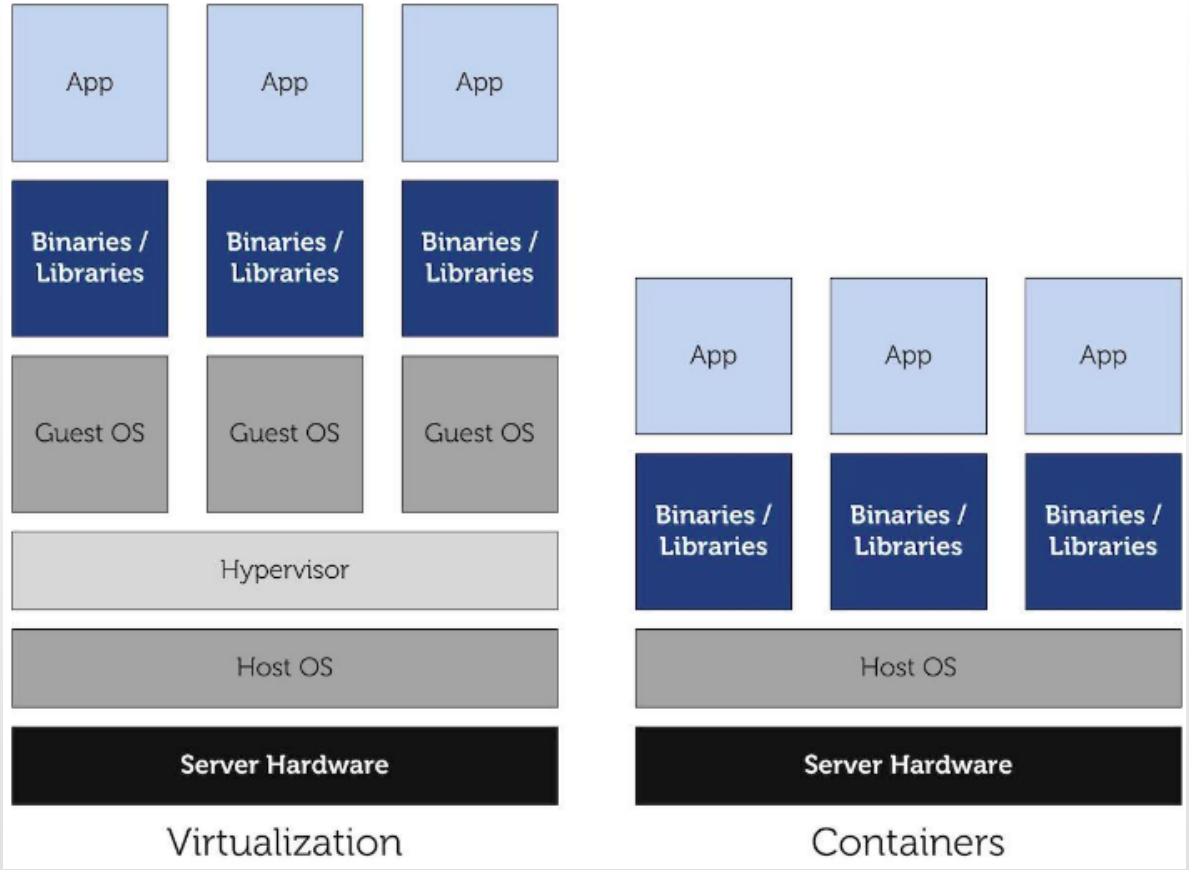
Fuente imagen: http://events.linuxfoundation.org/sites/events/files/cojp13_feng.pdf

Agenda

1. Linux cgroups y Namespaces
2. Containers - Docker

- Tecnología liviana de virtualización (lightweight virtualization) a nivel de sistema operativo que permite ejecutar múltiples sistemas aislados (conjuntos de procesos) en un único host
- Instancias ejecutan en el espacio del usuario. Comparten el mismo kernel (el del SO base)
- Dentro de cada instancia son como máquinas virtuales. Por fuera, son procesos normales del SO.
- Método de virtualización más eficiente: mejor performance, booteo más rápido
- No es necesario un software de virtualización tipo hypervisor.
- No es posible ejecutar instancias de SO con kernel diferente al SO base (por ej. Windows sobre Linux)
- LXC, Solaris Zones, BSD Jails, Docker, Podman, etc.

Linux Containers - Hypervisor vs. Container



Fuente Imagen:
https://wiki.aalto.fi/download/attachments/109397667/Linux%20containers.pdf?version=2&modificationDate=1447254317

¿Qué es Docker?

¿Qué es Docker?

- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Containers son livianos y contienen todo lo necesario para que la aplicación se ejecute
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.

¿Qué es Docker?

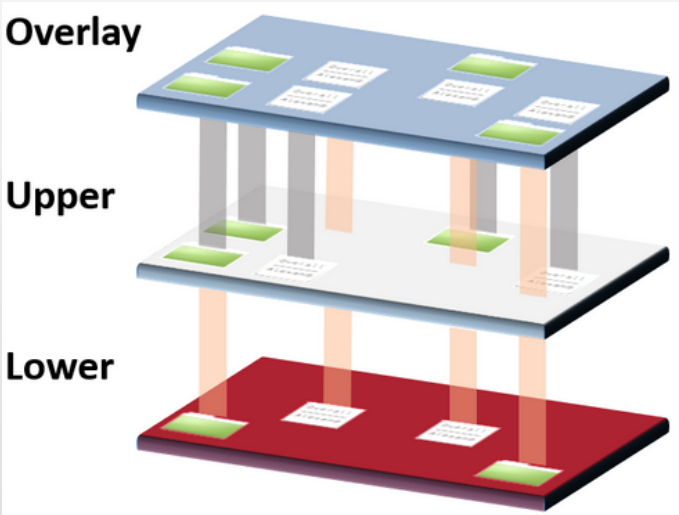
- Utiliza una architectura cliente-servidor.
- Cliente y Servidor pueden ejecutar en el mismo sistema o en diferentes
- Se comunican usando una REST API
- Docker “daemon” escucha por “API requests” y administra los objetos de Docker (imágenes, containers, red, etc,)
- Cliente Docker envía comandos al demonio, dockerd, usando las Docker API.

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**.

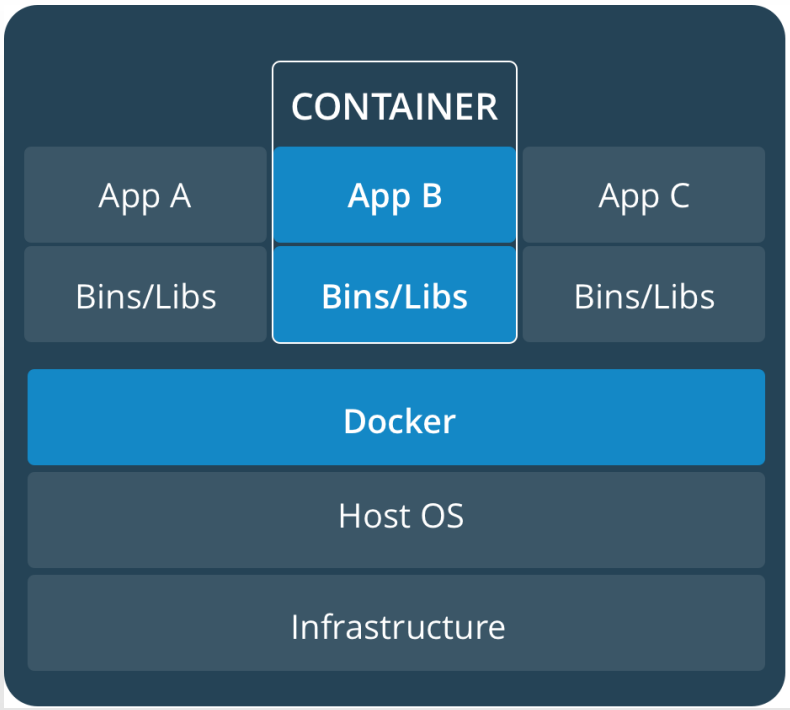
Union Mount - Unificar filesystems

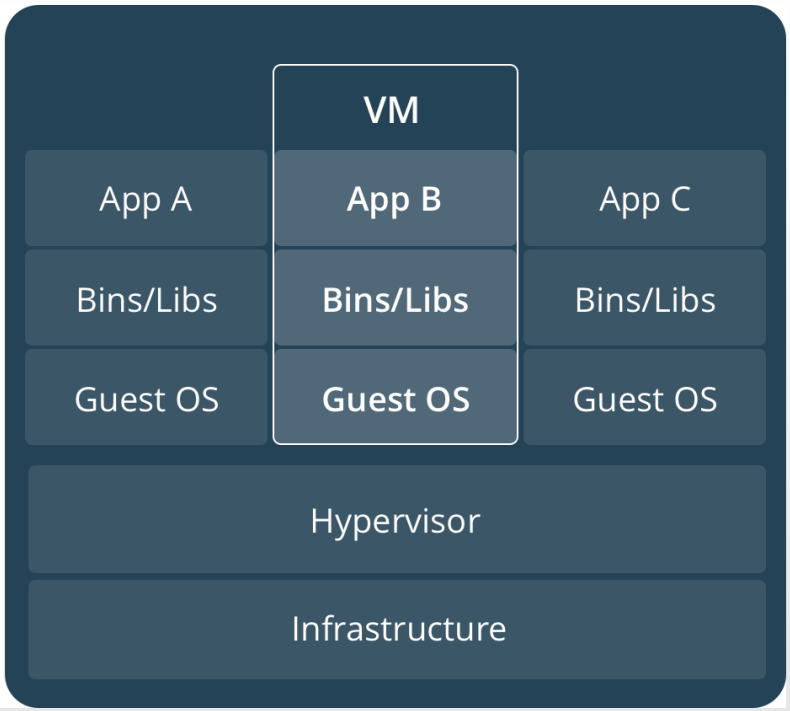
- Es un mecanismo de montajes, no un nuevo file system
- Permite que varios directorios sean montados en el mismo punto de montaje, apareciendo como un único file system
- Read-only capas inferiores, writable capa superior
- Existen varias alternativas: UnionFS, AuFS, overlayFS, etc



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

Containers vs. VMs





1

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

imagen: Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...)

registry: Es un almacén de imágenes de Docker. Por defecto, Docker utiliza *Docker Hub*.

container: Es una instancia de una imagen en ejecución.

Dockerfile: Archivo que define como 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 writeable imagen vacía.

imagen:	Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...)
registry:	Es un almacén de imágenes de Docker. Por defecto, Docker utiliza *Docker Hub*.
container:	Es una instancia de una imagen en ejecución.
Dockerfile:	Archivo que define como 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 writeable imagen vacía.

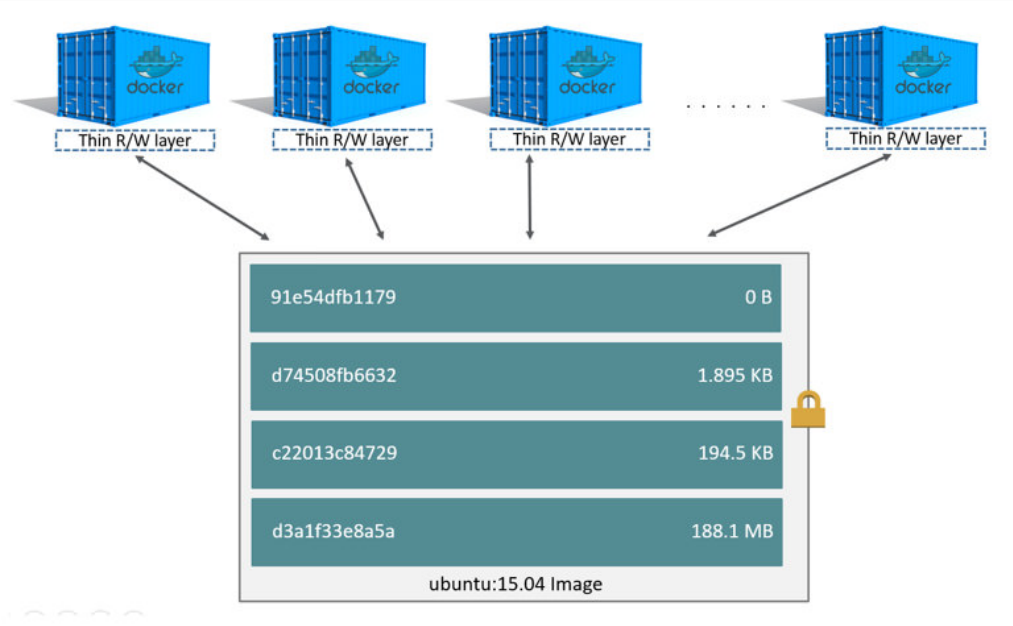
Capas

- Cada imagen está compuesta de una serie de capas.
- Las capas se montan una sobre otra (stacking)
- Solo la última es R/W (la capa del container). Las demás de solo lectura
- Docker usa "storage drivers" para almacenar capas de una imagen y para almacenar datos en la capa *escribible* de un contenedor
- Cada capa representa una instrucción en el archivo Dockerfile de la imagen
- Cada capa es un conjunto de diferencias con la anterior

2

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

Capas (cont.)

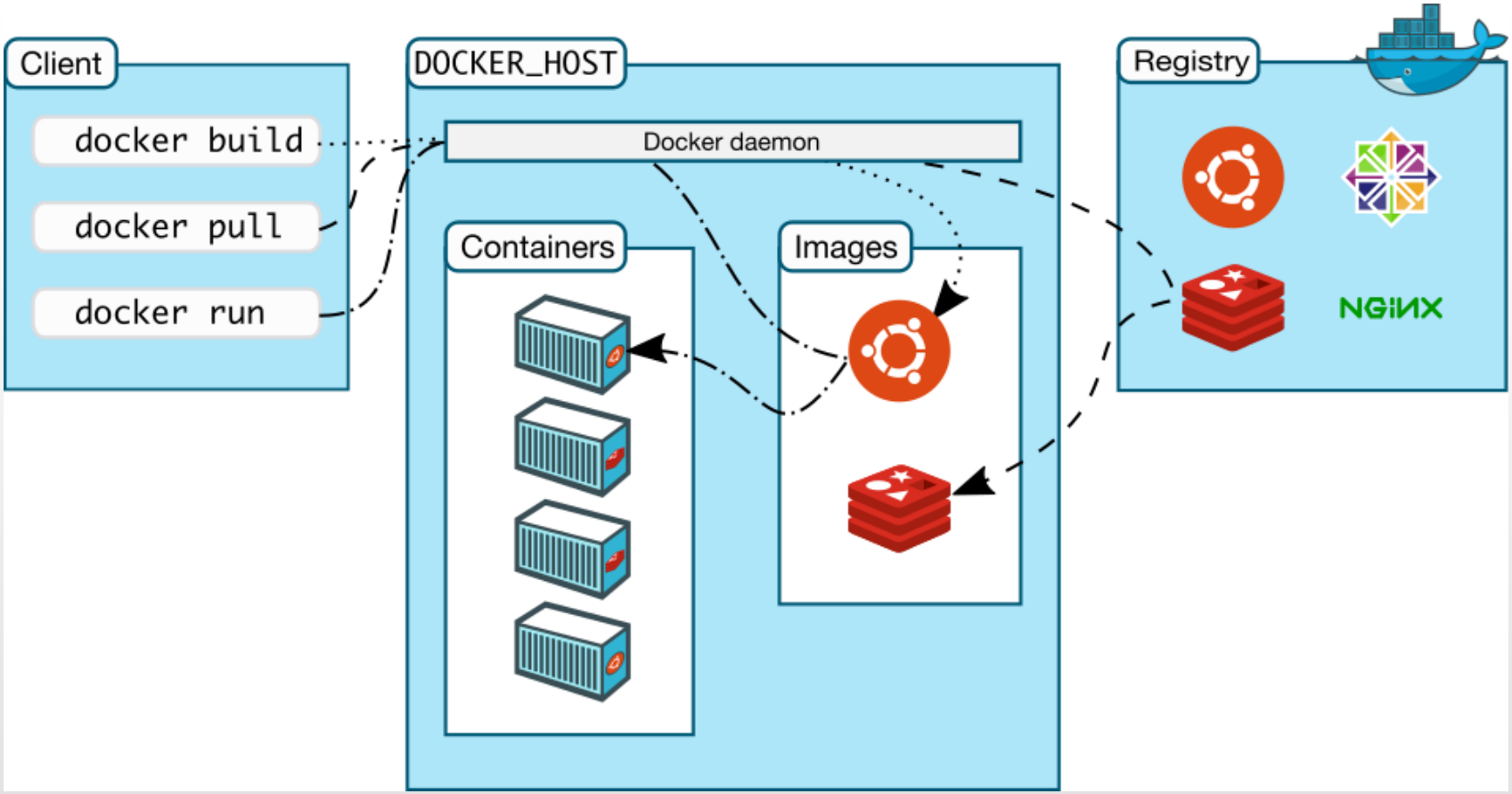


3

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

- Una imagen consiste de una colección de archivos (capas) que empaquetan todo lo necesario -dependencias, código fuente, librerías, etc. - para ejecutar una aplicación
- 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 esta 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

- Archivos creados dentro de un contenedor son almacenados en una capa "escribible"
  - Datos no persisten cuando el contenedor deja de existir
  - Escribir dentro del contenedor requiere de un "storage driver" que provee un "union filesystem" usando el kernel de Linux
- Docker tiene dos opciones para almacenar datos en el host y que los datos sean persistentes:
  - Volumes: almacenados en una parte del filesystem administrada por Docker.
  - Bind Mounts: pueden estar en cualquier parte del filesystem. Pueden ser modificados por procesos que no sean de Docker



[4]

[4] 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@so2022:~# 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@so2022:~# 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: sha256:9
    e2bbca079387d7965c3a9cee6d0c53f4f4e63ff7637877a83c4c05f2a666112
Status: Downloaded newer image for busybox:latest
3d6db57dd18c91d694a3e801f2ae41381d6e29a5f3f3a375228e5e096b9640c8
root@so2022:~# docker image ls
REPOSITORY    TAG         IMAGE ID        CREATED       SIZE
busybox       latest      af2c3e96bcf1    2 days ago    4.86MB
root@so2022:~# 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@so2022:~# 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@so2022:~# 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@so2022:~# 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@so2022:~# 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@so2022:~# 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@so2022:/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@so2022:/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@so2022:/sys/fs/cgroup# cd memory/
root@so2022:/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@so2022:/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@so2022:/sys/fs/cgroup/memory/docker/
     a08cb831097ed4168e7fddb4c8e49ef21d14b9962168aa5708e412a4d0a79bb2#
    more cgroup.procs
2529
root@so2022:/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

cgroups v1 - Kernel Documentation
https://www.kernel.org/doc/Documentation/cgroup-v1/

cgroups v2 - Kernel Documentation
https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html

cgroups v1/v2 - Linux Man Pages
http://man7.org/linux/man-pages/man7/cgroups.7.html

Namespaces - Linux Man Pages
http://man7.org/linux/man-pages/man7/namespaces.7.html

Referencias

Namespaces in Operation
https://lwn.net/Articles/531114/

Linux Containers
https://linuxcontainers.org/

Linux Containers
https://ubuntu.com/server/docs/containers-lxc

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?