Control Groups, Namespaces, Docker

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2025

Agenda

1. Linux cgroups y Namespaces
2. Contenedores

Agenda

1. Linux cgroups y Namespaces

2. Contenedores

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

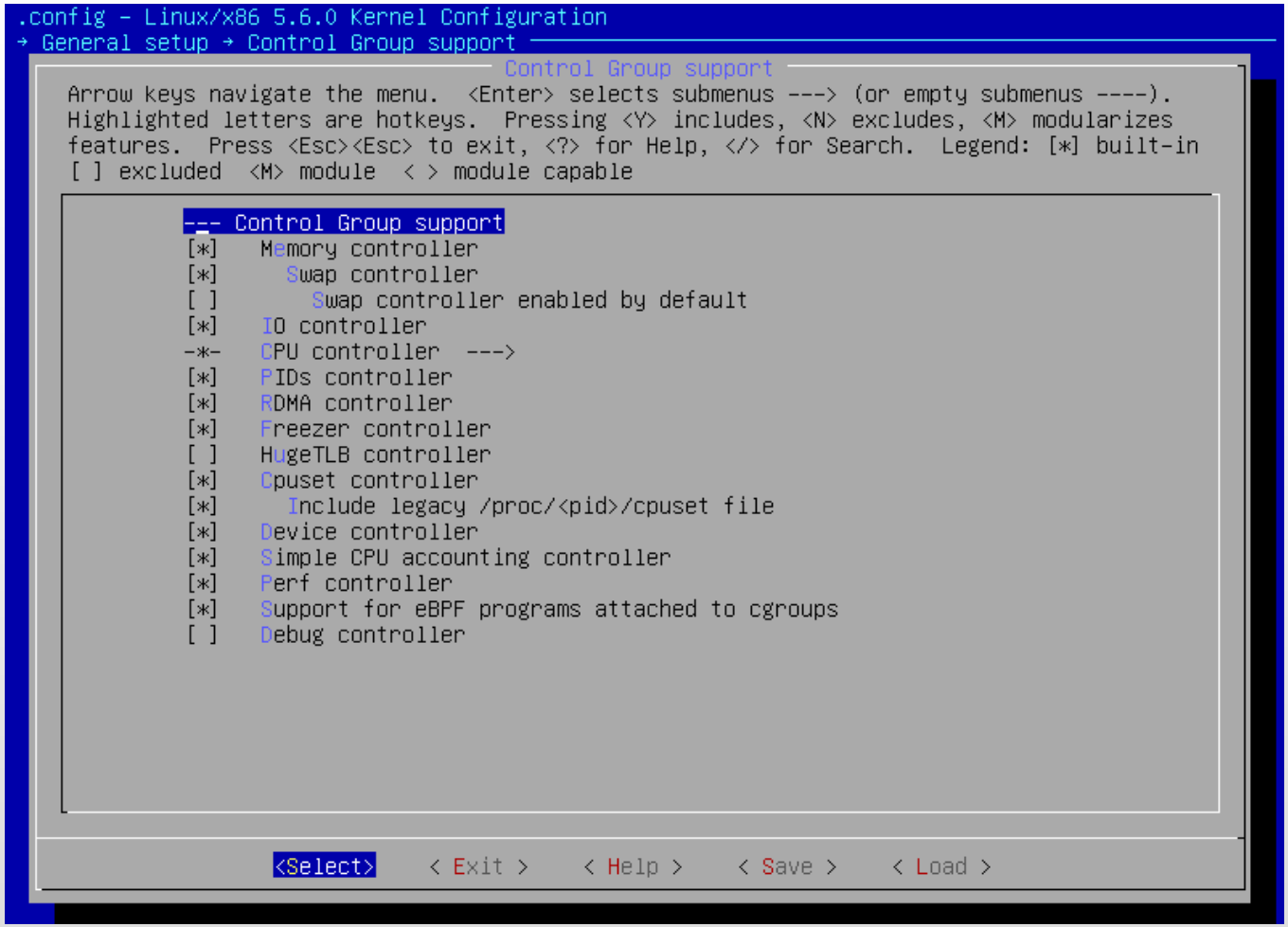
- En un sistema operativo se ejecutan varios procesos en forma concurrente
- En Linux, por defecto, todos los procesos reciben el mismo trato en lo que respecta a tiempo de CPU, memoria RAM, "I/O bandwidth".
- ¿Qué sucede si se tiene un proceso importante que requiere prioridad? O, ¿como limitar los 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

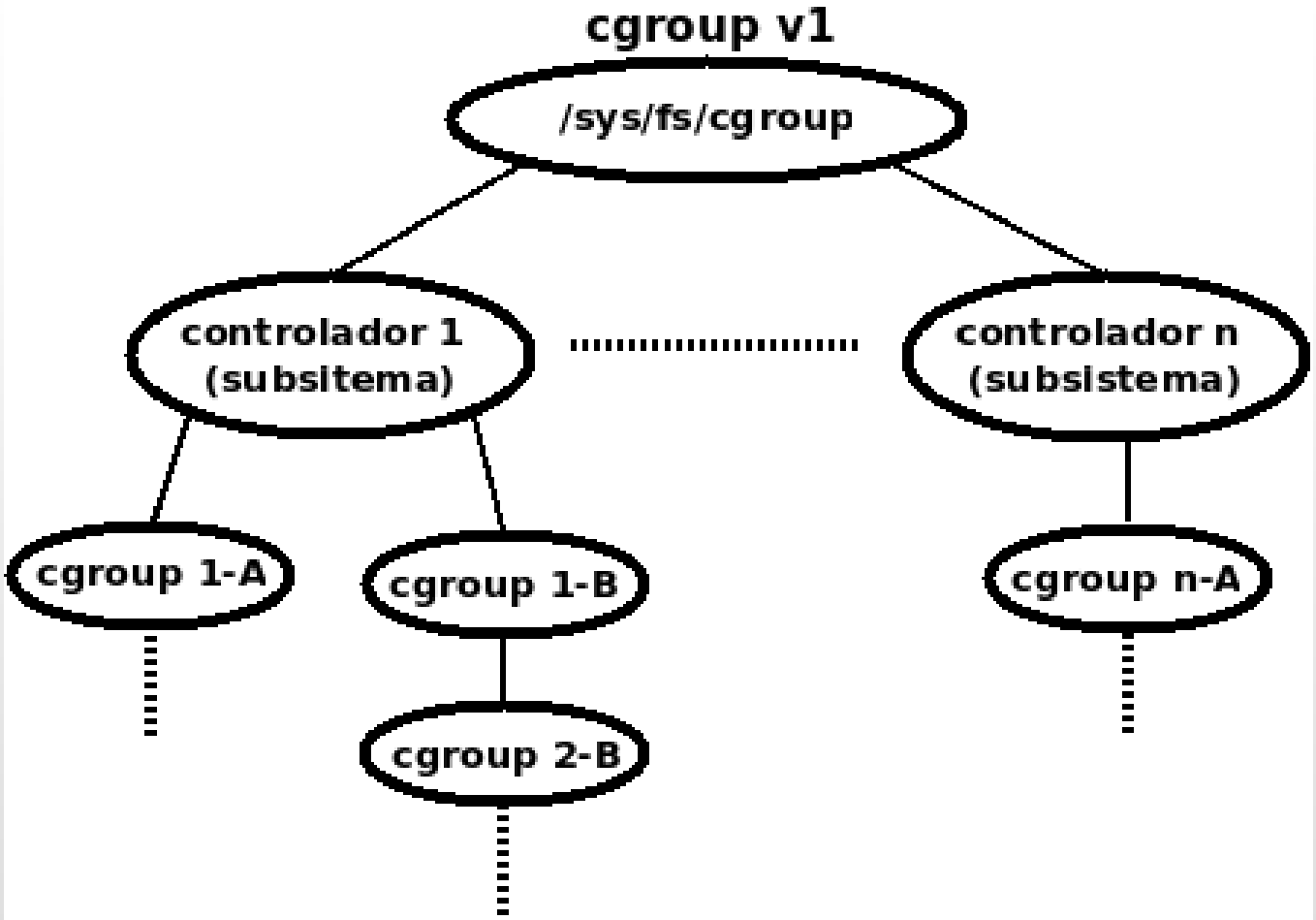
- **cgroup:** asocia un conjunto de procesos con un conjunto de parámetros o límites para uno o más subsistemas.
- **Subsistema:** componente del kernel que modifica el comportamiento de los procesos en un cgroup. También llamado *resource controllers* o simplemente *controllers*.
- **Jerarquía:** es un conjunto de cgroups organizados en una jerarquía.
- 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)

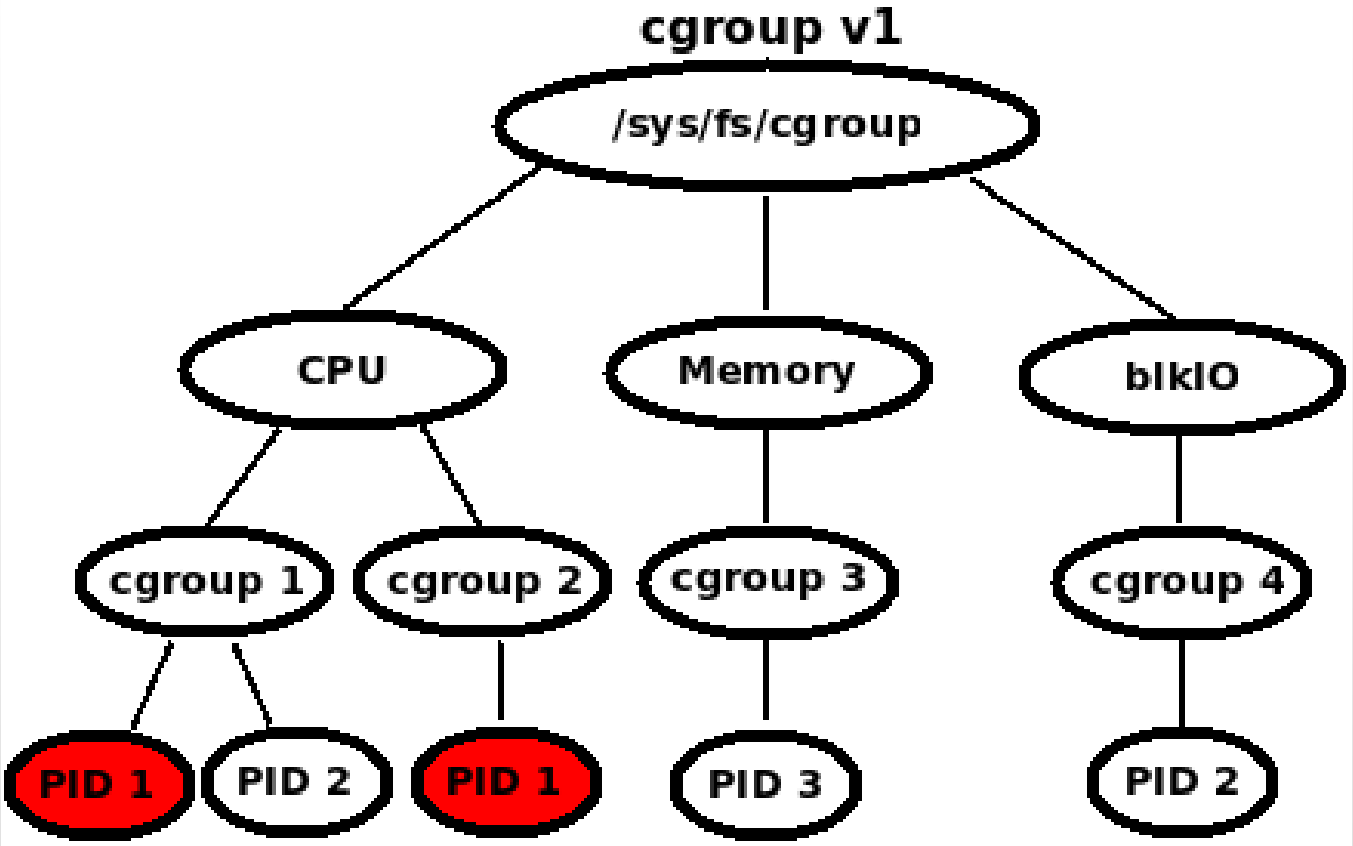
cgroup v1 - Características

- 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

cgroup v2 - Características

- Todos los controladores son montados en una jerarquía unificada.
- No es posible especificar un controlador en particular para montar.
- Por ej.: *mount -t cgroup2 none /sys/fs/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 un cgroup
  - **c**group.subtree_control: archivo de lectura/escritura que indica los controladores que se habilitaran en los cgroups hijos. Inicialmente vacío.

- *cgroup.subtree_control* determina el conjunto de controladores que pueden ser usando en los cgroups hijos.
- Controladores que se pasan a los cgroups se indican en el archivo *cgroup.subtree_control*. + para habilitar, - para deshabilitar.
- Por ej.: *echo '+pids -memory' > cgroup.subtree_control*
- Un controlador que no está presente en el archivo *cgroup.controllers* no se puede agregar al archivo *cgroup.subtree_control*.
- Controladores disponibles en el archivo *cgroup.controllers* de un cgroup son idénticos a los existentes en el archivo *cgroup.subtree_control* del cgroup padre.
- Si un controlador se remueve de un cgroup no puede ser rehabilitado en un cgroup hijo.

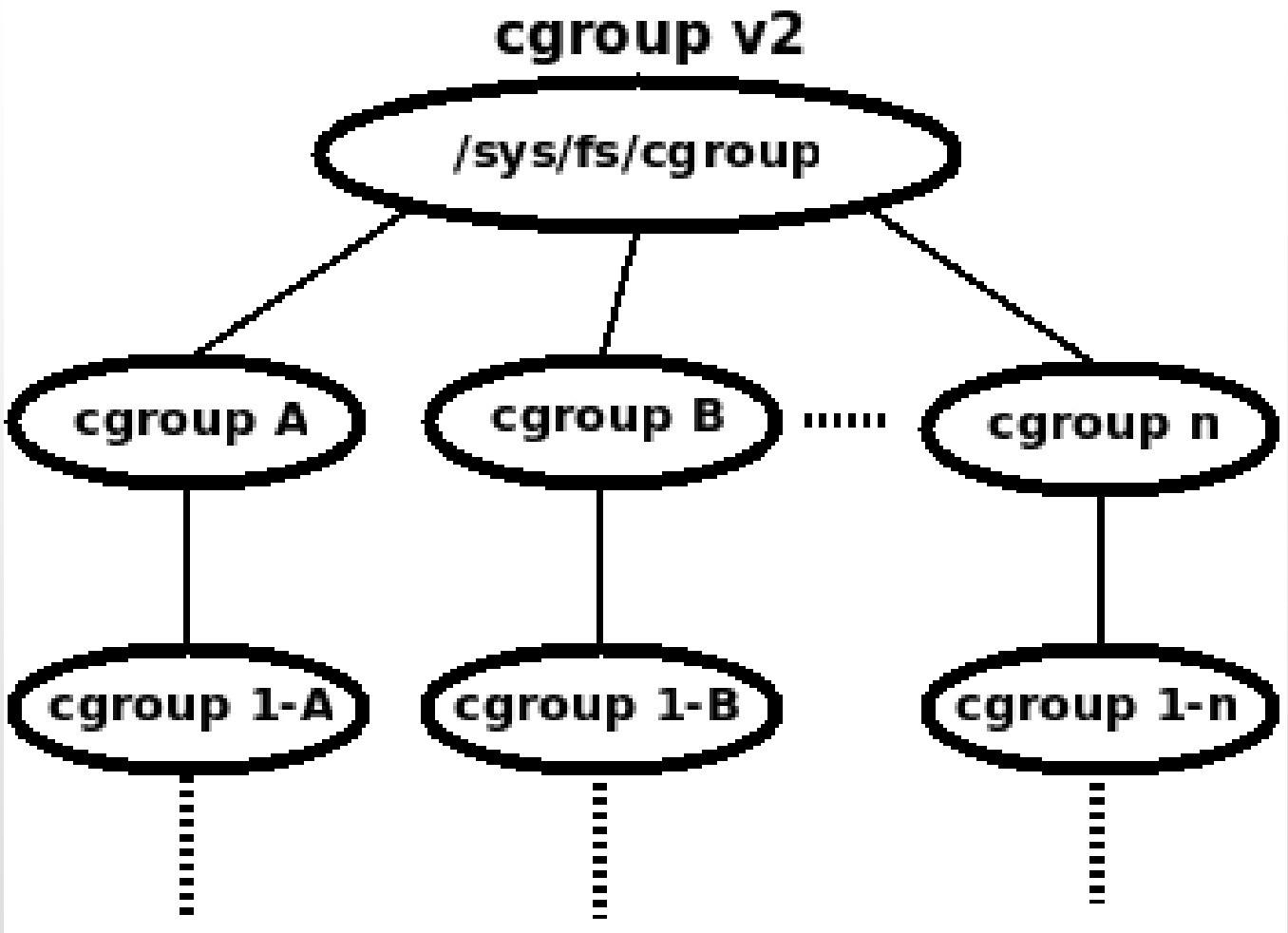
cgroup v2 - Características. . .

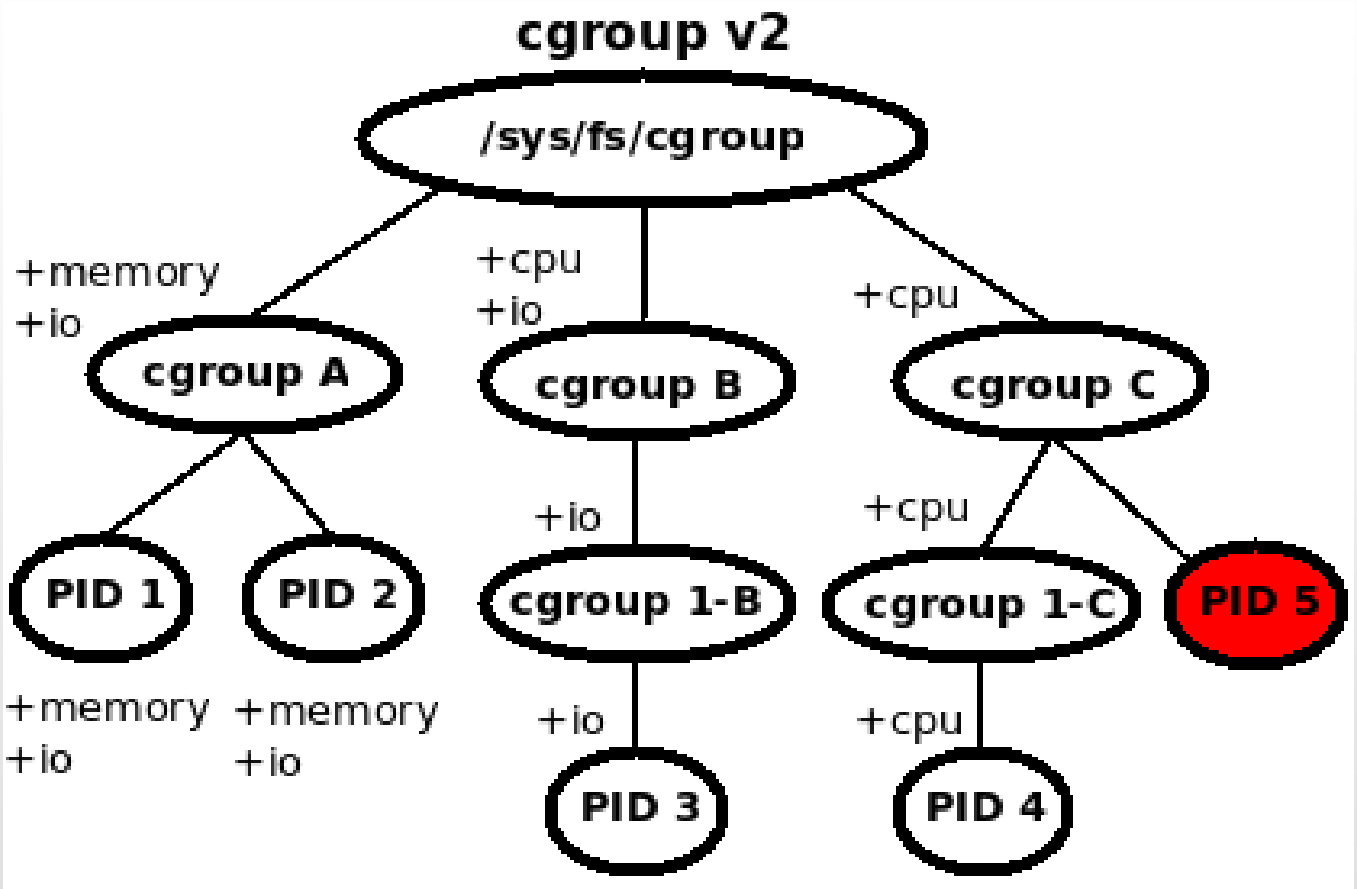
- No se permiten procesos internos, a excepción del cgroup raíz. Procesos deben asignarse a cgroups sin hijos (hojas)
- Por ej. si /procesos/proc1 son cgroups entonces un proceso puede residir en /procesos/proc1, pero no en /procesos.
- Inicialmente, solo el cgroup raíz existe y todos los procesos pertenecen a él.
- mkdir CG_NAME y rmdir CG_NAME para crear y eliminar cgroups.
- Cada cgroup tiene un archivo lectura/escritura *cgroup.procs* con los procesos pertenecientes a ese cgroup. Inicialmente vacío.
- Archivos correspondientes a un controlador habilitado en el archivo *cgroup.subtree_control* de un cgroup son automáticamente generados al crearse un cgroup hijo (contrador: memory, archivo: memory_max)

- En cada cgroup diferente al raíz existe un archivo *cgroup.events* de solo lectura.
- Contiene pares de clave-valor:

```
$ cat proc1/cgroup.events
   populated 1
   frozen 0
```

- *Populated:* si es 1, este cgroup o alguno de sus descendientes tiene procesos miembros.
- *Frozen:* si es 1, este cgroup está freezado.
- Permite notificar cuando un cgroup está vacío.


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

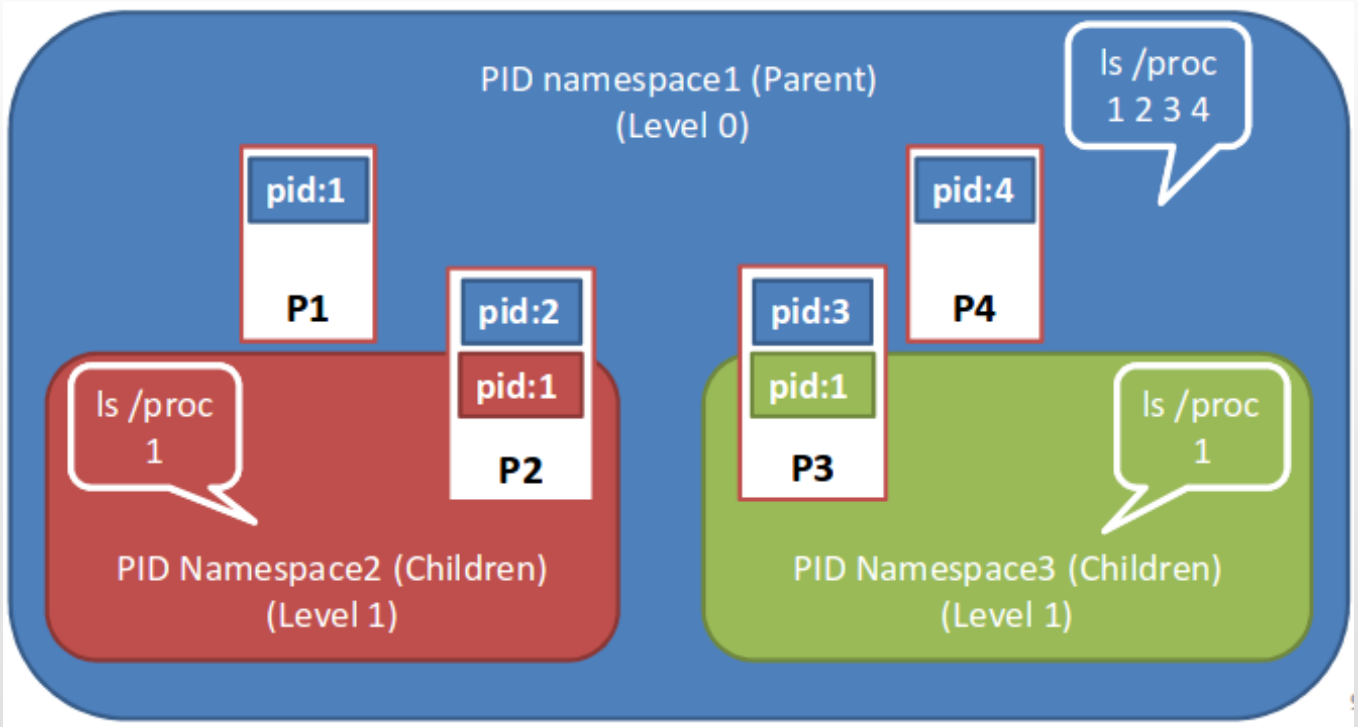
Permite abstraer un recurso global del sistema para que los procesos dentro de ese "namespace" piensen que tienen su propia instancia aislada de ese recurso global

- 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 su 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

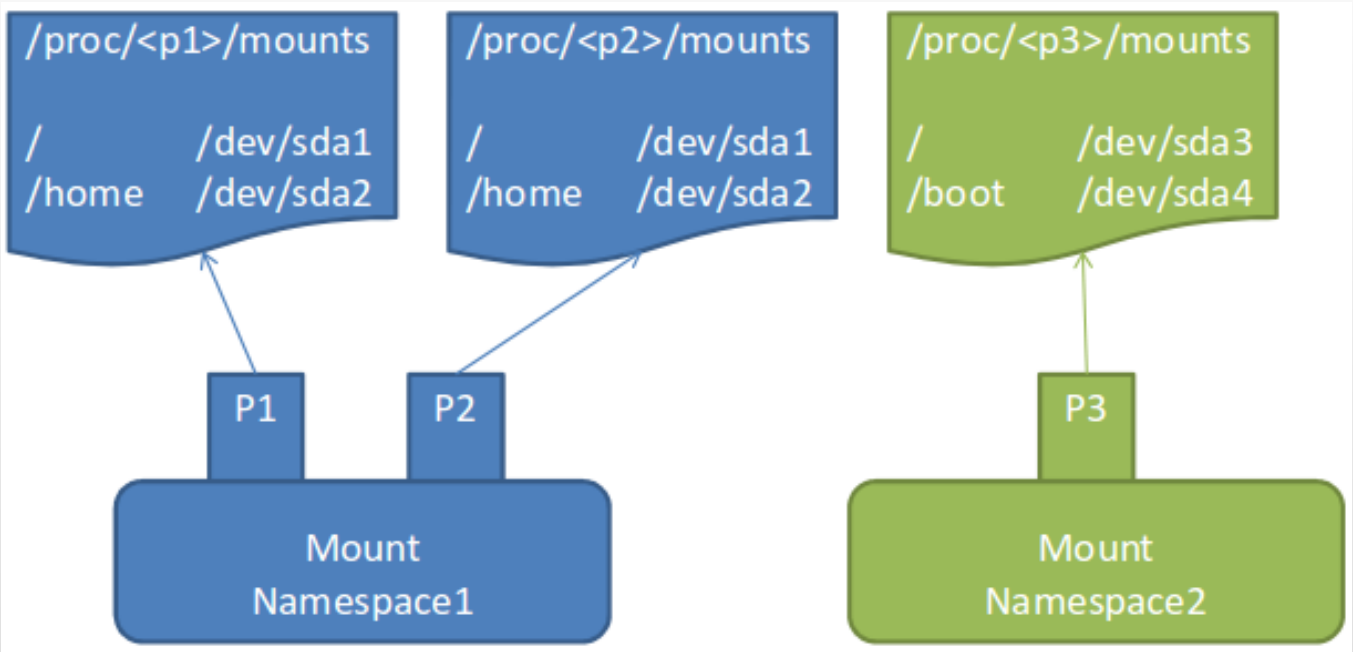
- Tres nuevas systems-calls:
  - **clone():** similar al fork. Crea un nuevo proceso y lo agrega al nuevo namespace especificado. Su funcionalidad puede ser controlada por flags pasados como argumentos
  - **unshare():** agrega el actual proceso a un nuevo namespace. Es similar a clone, pero opera en 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*
- Un proceso hijo hereda todos los namespaces de su proceso padre.

- 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

User Namespace

- Permite que los usuarios y grupos tengan su propios IDs de usuario y grupo.
- Usuarios y grupos también existen en el hosts, pero con diferentes IDs.
- Usuario con ID 0 en el contenedor puede tener una identidad no root, no privilegiada, en el host.
- No es necesario ser root para crear un *user namespace*.
- Por default, usuario *nobody* en el contenedor.
- $/proc/PID/uid\_map$ y $/proc/PID/gid\_map$ para indicar el mapping entre usuario y grupo en el host y en el contenedor.
- *ID-inside-ns ID-outside-ns length*:
  - ID-inside-ns: indica el UID inicial en el namespace
  - ID-outside-ns: indica el UID fuera del namespace
  - length: indica el número de mapeos de UID subsiguientes
- Por ej: 0 1000 100

Agenda

1. Linux cgroups y Namespaces
2. Contenedores

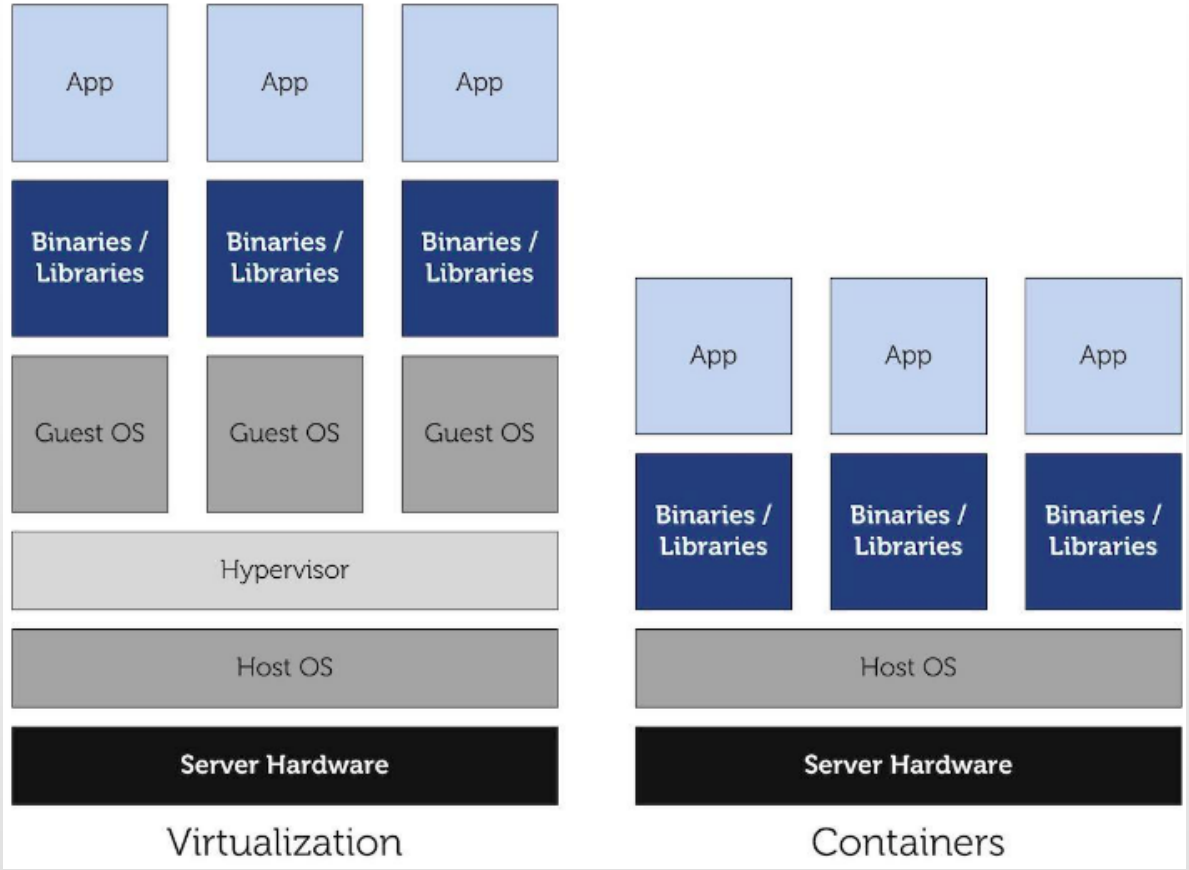
Historia (solo algunos hitos)

- 1979: UNIX v7. Implementa la system call *chroot*
- 2000: FreeBSD Jail. Extiende el *chroot*: dirección IP, hostname, usuarios y procesos propios.
- 2004: Solaris Zones. Pueden contener diferentes binarios, toolkits e, inclusive, diferente OS.
- 2008: Linux Containers (LXC)
- 2013: Docker entra en escena
- 2014: Proyecto Kubernetes anunciado. Docker 1.0 liberado
- 2015: Open Container Initiative creado. Kubernetes 1.0 lanzado.
- 2018: Google Kubernetes Engine se vuelve disponible
- 2019: RedHat lanza la versión 1.0 de Podman

- 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.

- Varias definiciones de container:
  - Una forma de empaquetar aplicaciones
  - Una máquina virtual liviana
  - Un conjunto de procesos aislados del resto
- Dos tipos de contenedores:
  - De sistemas operativos: ejecutan un SO completo (menos el kernel). LXC, BSD Jails, Solaris Zone, etc.
  - De aplicaciones: empaqueta una aplicación o proceso. Docker, Podman, etc.
- Sus principales características:
  - Autocontenidos: tiene todo lo que necesita para funcionar
  - Aislados: mínima influencia en el nodo y otros contenedores
  - Independientes: administración de un contenedor no afecta al resto
  - Portables: desacoplados del entorno en el que ejecutan. Pueden ejecutar de igual manera en diferentes entornos.

Hypervisor vs. Container



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

- *Containerization* es la habilidad de construir y empaquetar aplicaciones como contenedores *shippables*.
- Un contenedor contiene un código específico y todas las librerías y dependencias necesarias para ejecutarse.
- Ejecutan de manera aislada en modo usuario usando un kernel compartido.
- Procesos en un contenedor tienen 2 IDs: uno en el contenedor y otro en el host (PID Namespace).
- Típicamente, cada contenedor provee un único servicio ("micro-servicio").
- Desde el lado del nodo host, un contenedor es un proceso (o conjunto de procesos) ejecutándose.
- En el nodo host se ven los procesos de todos los contenedores. Esto no es posible a la inversa ni entre distintos contenedores.

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?