Introducción aos contedores de software

Nos últimos catro anos, o mundo dos sysadmin está a asistir a unha **revolución** sen precedentes.

- Qué pasaría se poidesemos crear máquinas virtuais con coste prácticamente nulo en tempo e computación?
- Qué ocorrería se nunha máquina poidesemos despregar en cuestión de segundos centos ou incluso miles de máquinas virtuais co seu propio SO, sistema de ficheiros, usuarios, árbores de PIDs, stacks de rede...?
- E se, por enriba, esta tecnoloxía fose compatible con todo o stack actual, puidendo correr en servidores físicos, servidores virtuais, clouds privadas e públicas ou no propio laptop do desenvolverdor?

Todo iso é o que posibilita a nova tecnoloxía dos contedores de software.

O cambio é dun calado tal que afecta non só as coitas e preocupacións propias dos devops e sysadmins senón tamén á forma de construir e programar as aplicacións e servizos. Estamos ante un novo paradigma que introduce, por fin, na industria unha nova arquitectura: a dos **microservicios**.

No presente curso imos adentrarnos no novo paradigma dos contedores para darlle ó docente unha visión o máis completa e práctica posible desta nova tecnoloxía e das súas posibilidades.

Obxectivos

Obxectivos

- Repasar as funcións tradicionais dun SO e entender as suas **limitacións**, no illamento de certos recursos.
- Comprende-la razón pola que as técnicas de virtualización, ata o de agora, non deron una resposta completa ao problema dos **recursos globais**.
- Explorar os contedores de software como unha técnica de virtualización a nivel de SO, que permite *resolver o problema do illamento dos procesos*.

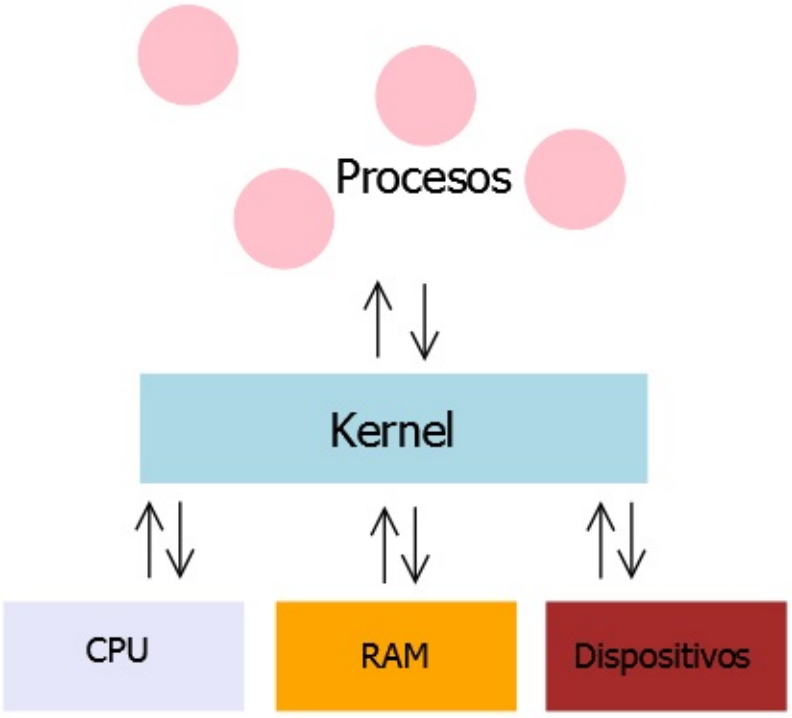
Proceso e Kernel: os recursos

O kernel dun SO ten como misión fundamental a de mediar entre as aplicacións que se executan nunha máquina, os procesos, e os distintos dispositivos físicos e virtuais que a compoñen. Para esto, se crea unha abstracción fundamental: o **recurso**.

Os recursos poden ter:

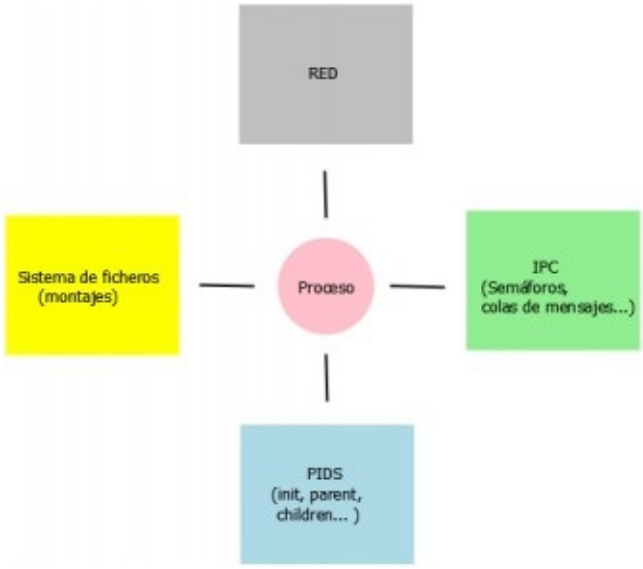
- una dispoñibilidade limitada
- un acceso restrinxido
- un empleo privativo.

É tarefa do kernel impedir que se monopolice o uso dos mesmos, que se produzan accesos indebidos ou que se simultanee o uso de recursos de utilización exclusiva.



Ademais, o kernel leva a cabo unha importante labor de homoxeneización, ocultando os detalles do funcionamento dos diferentes dispositivos e ofrecendo aos procesos interfaces limpas e unificadas que, estandarizan o emprego de hardware de distinta procedencia.

Podemos concluir, por tanto, que **os procesos non "ven" os dispositivos que conforman a máquina onde corren, senón a representación que dos mesmos lles ofrece o kerne**l.

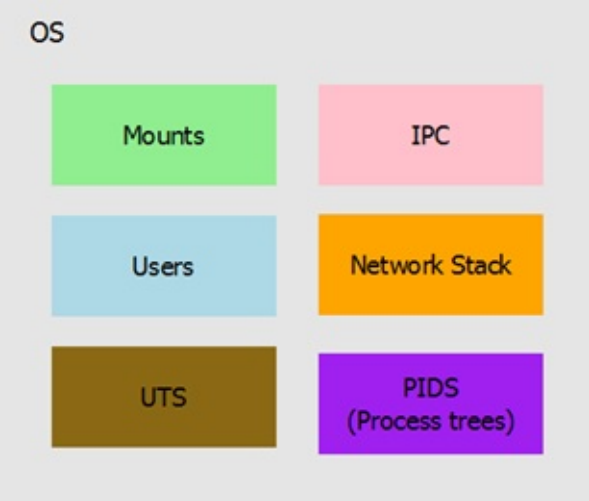


A "pingueira" no illamento do Kernel: recursos globais

Vimos na sección anterior que o kernel illa unha serie de recursos de forma que poden usarse privativamente, ainda que sexa de forma ilusoria, por un proceso ou grupo de procesos.

Pero, ¿logran os SO illar todos los recursos dunha máquina?

La respuesta é que non. Alomenos nos entornos UNIX, existen unha serie de recursos que son comúns e, polo tanto, globais para todos os procesos.



Estos recursos son:

- Usuarios
- Comunicacións entre procesos (IPC: Sockets, pipes...)
- Puntos de montaxe (Sistemas de ficheiros)
- Pila de rede (interfaces, bridges, iptables...)
- UTS (hostname, domainname...)

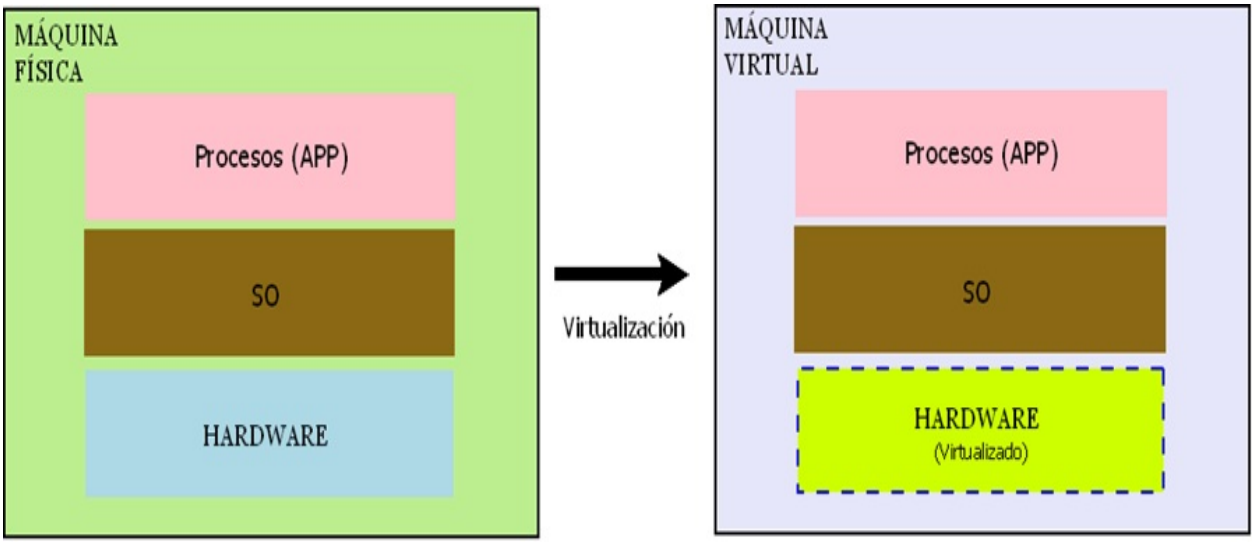
- PIDs

Esto implica que, nun sistema non pode haber dous procesos co mesmo PID, ou dos procesos que vexan dos hostnames diferentes na mesma máquina, ou teñan acceso a puntos de montaxe distintos...

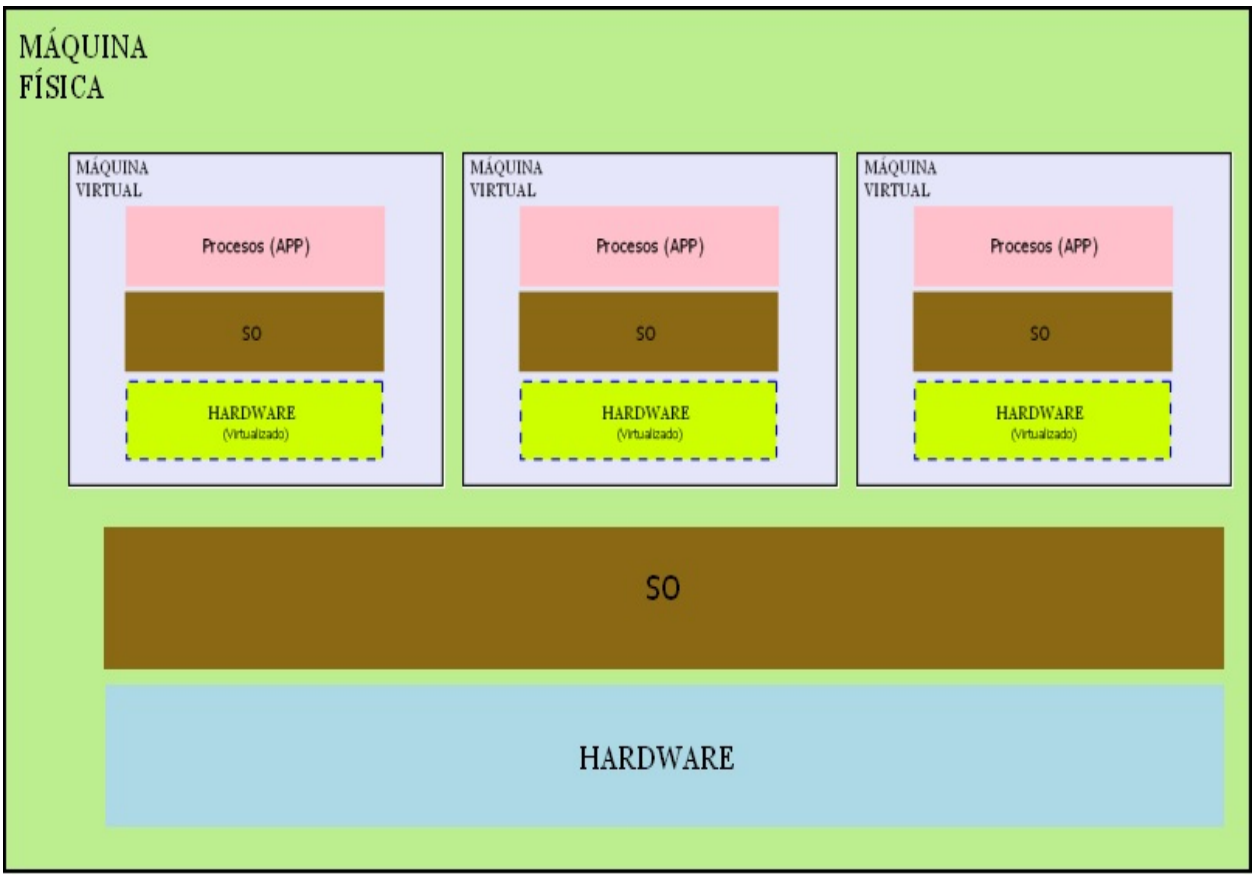
Solución parcial: virtualización da plataforma

Unha das solucións que se empregan para evitar o *problema da falta de illamento do SO dos recursos globais, é* a virtualización a nivel de plataforma.

Existen diversas técnicas (paravirtualización, full virtualization) pero a idea á a mesma: virtualizar o hardware, esto é, simular un ordenador mediante software.



De esta forma, e dado que temos varias máquinas virtuais, cada unha co seu sistema operativo, podemos illar os recursos globaies antes mencionados.



Non obstante,

- O coste en CPU/RAM da emulación do hardware é elevado
- O tempo de arranque/parada dunha máquina virtual é considerable (> 1')

Por soposto a virtualización de plataforma sigue tendo virtudes importantes:

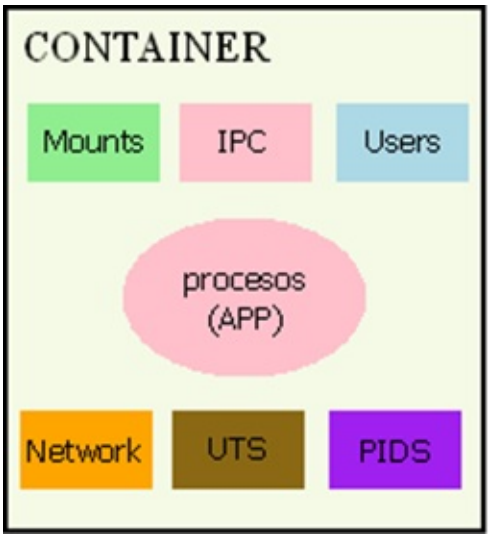
- Mellor aproveitamento do hardware
- Automatización dos despregues
- Portabilidade de VMs

Con todo, o su emprego con fins de illamento de procesos é **excesivamente caro en térmos de tempo e recursos**.

Solución completa: os namespaces

Os contedores de software son unha técnica de virtualización a nivel de SO, tamén se coñocen como virtualización a nivel de proceso.

A idea é sinxela, dado que o SO é, desde o punto de vista do proceso, un conxunto de recursos, podemos ofrecerlle unha vista "privada" ou virtual desos recursos.



A virtualización desos recursos globais de tal forma que, desde o punto de vista do proceso, sexan privados para él, **é no que consiste un contedor.**

> "De igual xeito que na virtualización a nivel de plataforma o SO "cree" estarse executando nunha máquina real, na containerización, o proceso "cree" ter un SO para sí mesmo"

A técnica de usar contedores é superior a da virtualización de plataforma

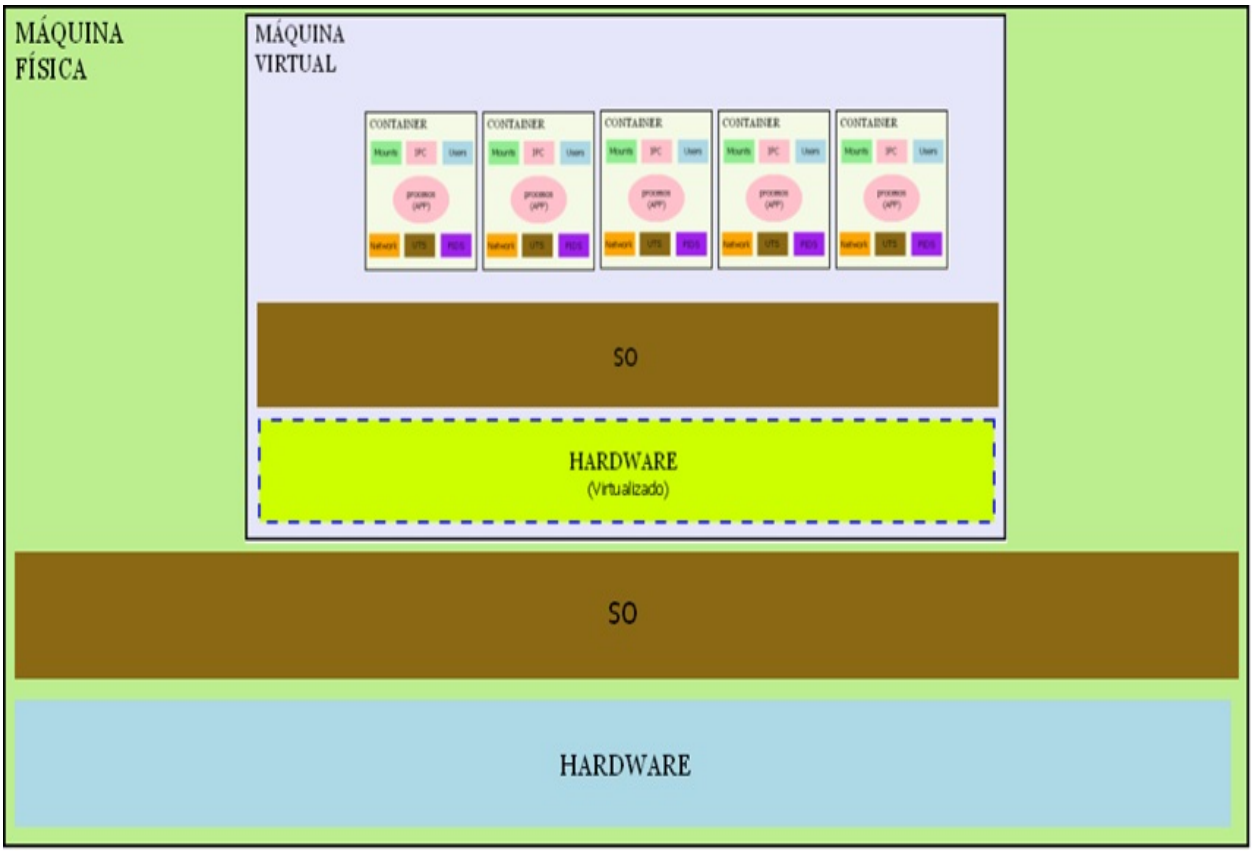
en que:

- No supón un custe de recursos adicionais por tener que emular hardware e correr en él un SO: se poden ter milleiros de contedores n un servidor
- El arranque/parada dun container é prácticamente igual ao arranque/parada dun proceso (< 1")

A costa de:

- Compartir o Kernel do SO

Ademais, no é alternativa a técnica de virtualización de plataforma: ao contrario, é totalmente compatible e como se está empregando en moitos sitios:



Solución completa (2): os cgroups

Sabemos que o Kernel de Linux ten como traballo evitar a monopolización por parte dos procesos de recursos básicos tales como:

- CPU
- Memoria
- Operacións E/S

A pregunta que xorde é: permite un control fino de acceso e consumo destes recursos?

A resposta é que, previamente á versión 2.6.24, existían mecanismos de control (fundamentalmente o comando nice) pero eran moi limitados.

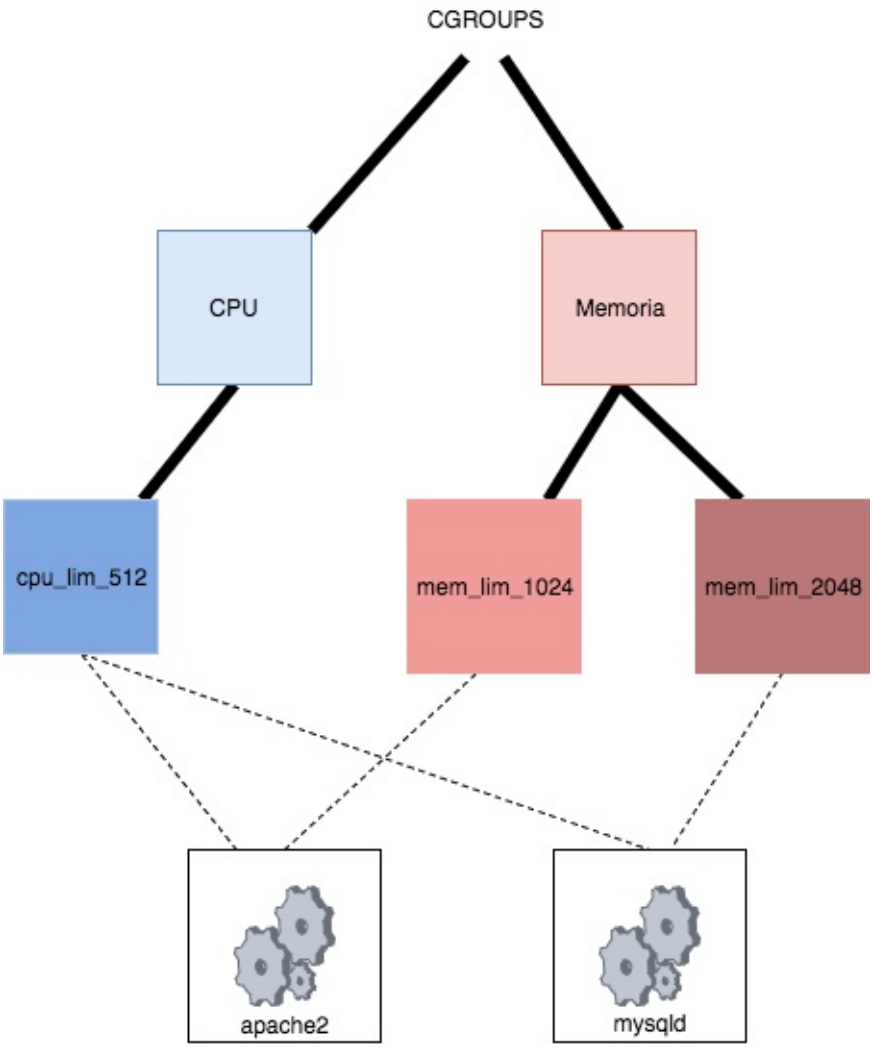
Todo isto cambia coa adopción polo kernel de Linux, en xaneiro de 2008, dos control groups (abreviado cgroups) impulsados principalmente polos enxeñeiros de Google.

Os cgroups pódense ver como unha árbore en que os procesos están pendurados dunha pola de control de tal xeito que podense establecer, para ese proceso e os seus fillos:

- Limitacións de recursos.
- Prioridades de acceso a recursos.
- Monitorización do emprego dos recursos.
- Xestión a baixo nivel de procesos.

A flexibilidade que permiten é moi grande. Pódense crear distintos grupos de limitacións e control e asignar un proceso, e os seus fillos, a

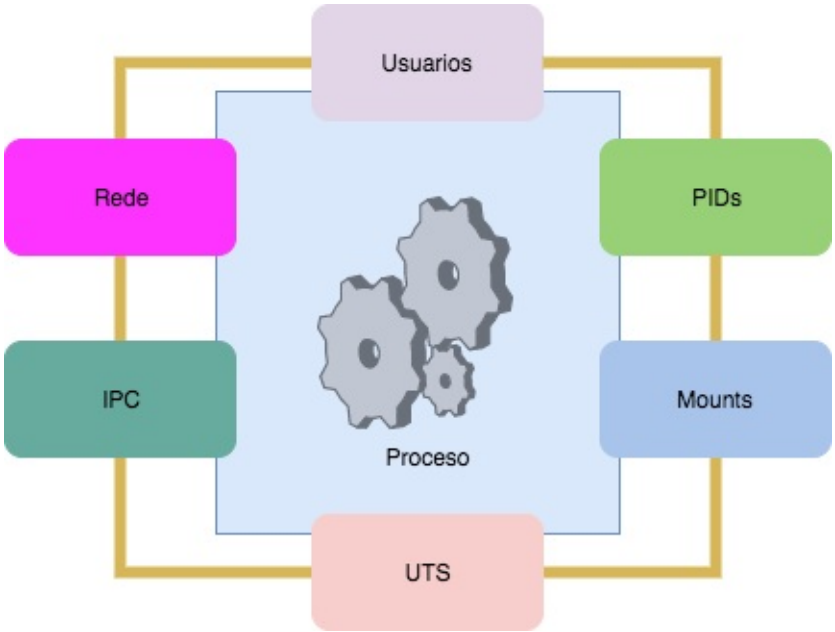
distintos grupos, facendo combinacións que permiten un grao moi alto de personalización.



Entón, ¿qué é un contedor?

Un contedor é unha técnica de virtualización a nivel de sistema operativo que illa un proceso, ou grupo de procesos, ofrecéndolles un contexto de execución “completo”. Se entendo por contexto, ou entorno de execución, o conxunto de recursos (PIDs, red, sockets, puntos de montaje…) que lle son relevantes ao proceso.

O contedor está baseado nos namespaces que o manteñen "separado" do resto do sistema.



A medio camiño entre o chroot e as solucións del virtualización completas (KVM, VirtualBox, VMWare, Xen) o contedor no incurre no custo de virtualizar o hardware ou o kernel do SO ofrecendo, así e todo, un nivel de control e illamento moi superior ao do chroot.

O contedor é moito máis rápido en ser aprovisionado que a máquina virtual (VM), non precisa arrancar unha emulación de dispositivos nin o

núcleo do sistema operativo, a costa dun nivel de illamento menor: procesos en distintos contedores comparten o mismo kernel.

Practica guiada: construindo o noso contedor

Empregando as ferramentas que nos da Linux, imos construir un contedor para unha distro de debian.

Nota: esta práctica está baseada nesta

1. O sistema de ficheiros

Para correr un sistema debian, obviamente, precisamos do conxunto de ferramentas, útiles e demonios que ten unha distro deste tipo.

Imos baixar un sistema de ficheiros con tódolos elementos necesarios.

Collemos o tar deste link.

Nun directorio do noso sistema de ficheiros, descargamos o tar co sistema debian:

```
wget https://github.com/ericchiang/containers-from-scratch/releases/download/v0.1.0/rootfs.tar.gz
```

Imos extraer os contidos do tar:

```
tar xfz rootfs.tar.gz
```

Se listamos os contidos do rootfs resultante, veremos que ten unha estrutura moi similar á dun sistema tradicional debian.

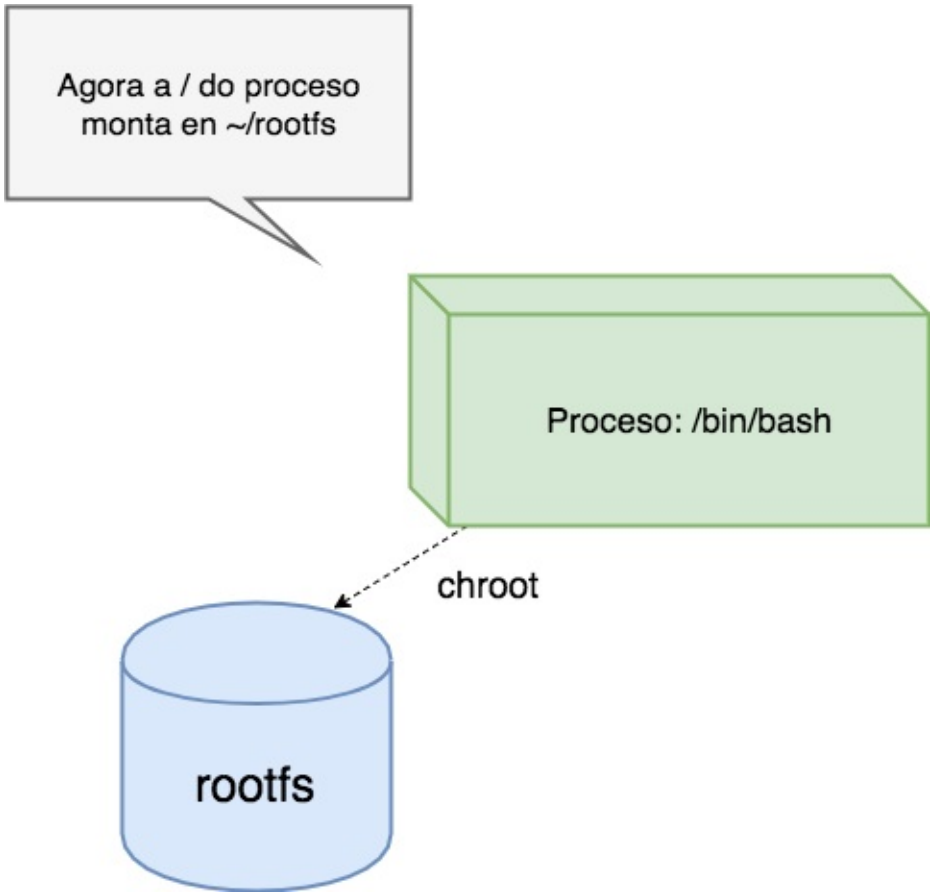
2. Engaiolar o proceso no sistema de ficheiros mediante chroot

A ferramenta chroot, permítenos illar un proceso con respecto a unha ruta concreta dentro do noso sistema de ficheiros.

Se lanzamos este comando dende a ruta onde desentarramos o noso sistema de ficheiros:

```
chroot rootfs /bin/bash
```

Entraremos nunha nova shell, un novo proceso, que está montado a partires de ~/rootfs.



Temos un container real?

A resposta é que non.

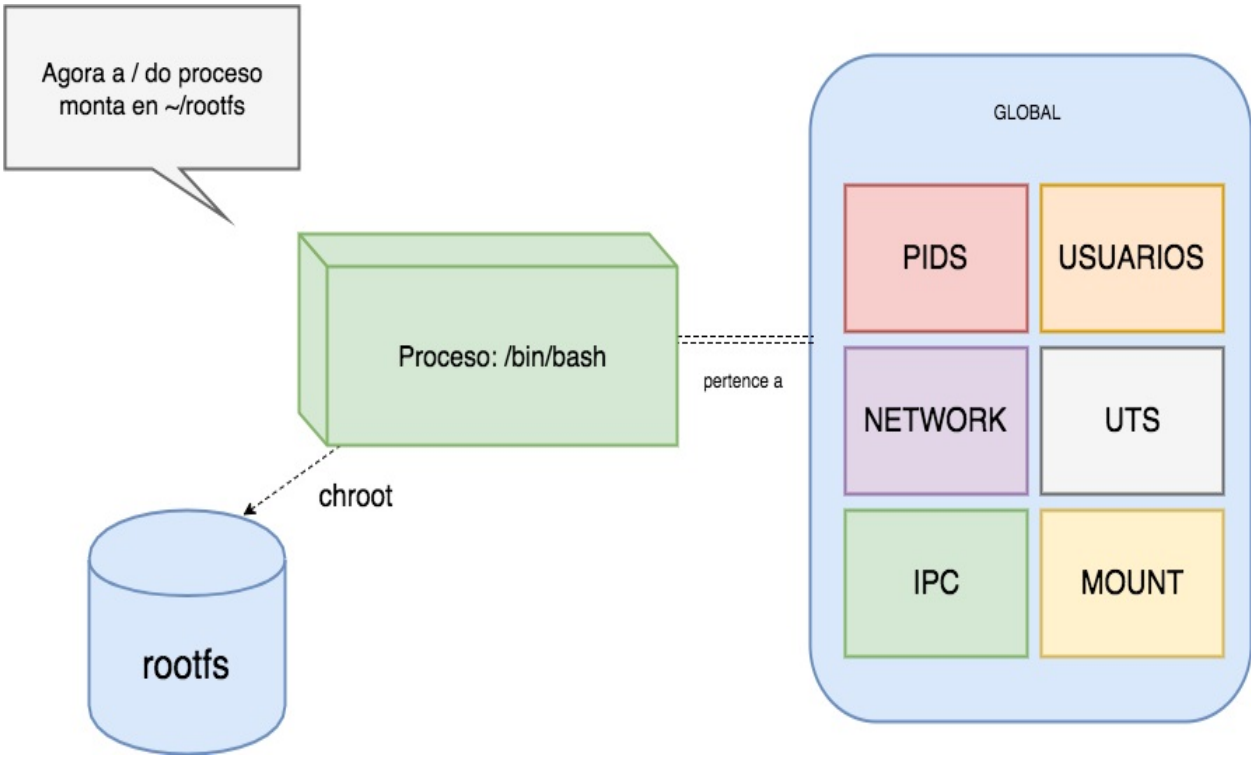
Se montamos o proc nunha ruta do noso proceso:

```
mount -t proc proc /proc
```

E facemos un ps ou un top, seguimos a ver tódolos procesos do sistema.

Polo tanto, o noso proceso, ainda que ten como raíz o rootfs, non está realmente illado do resto do sistema, posto que segue a pertencer ós namespaces globáis.

É dicir, o noso proceso **segue a estar no namespace global compartido polo resto dos procesos do sistema**.



Como podemos ver, este proceso non está realmente "contido":

- Pode crear usuarios no namespace xeral da máquina
- Pode ve-los procesos de toda a máquina
- Se modifica iptables, se conecta a portos.. estará afectando ó resto dos procesos da máquina

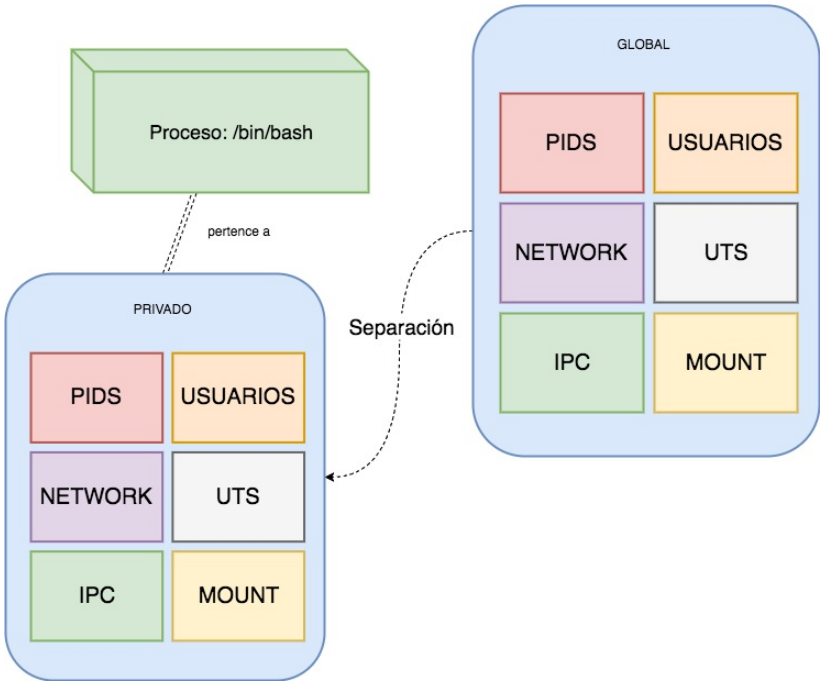
Isto non é un verdadeiro isolamento do noso proceso.

Para logralo, imos recorrer ós namespaces.

3. Illa-lo proceso mediante os namespaces

Como vimos no paso 2, realmente, estamos a lanzar un proceso que ten unha nova raíz no sistema de ficheiros, pero non está realmente illado do resto dos recursos do sistema. Se o queremos realmente illar, teríamos que crear unha serie de namespaces privados dese proceso (e do resto de fillos do mesmo).

Nun diagrama:



O comando unshare permítenos lanzar un comando ou proceso especificando os namespaces que queremos que sexan privados do mesmo.

Imos lanzar de novo un proceso, pero esta vez mediante unshare:

```
unshare -m -i -n -p -u -f chroot rootfs /bin/bash
```

Neste comando estamos a dicirlle ó sistema:

- Lanza o proceso chroot rootfs /bin/bash
- Illa en namespaces novos de mounts, de ipc, de networking, de pids, de UTS ó proceso

Sinxelo, non? Imos ver se realmente é así.

Montamos o proc:

```
mount -t proc proc /proc
```

Se introducimos o comando top, veremos que temos dous procesos.

Vemos que proceso que ten o pid 1, é o noso /bin/bash.

Cómo é iso posible?

Recordemos que agora estamos dentro dun novo namespace de PIDS é, o proceso que creou ese namespace, foi o noso /bin/bash.

Imos facer outra cousa:

```
hostname
```

Veremos o hostname da nosa máquina. O cal non é extraño, dado que o namespace de UTS clonóuse do noso UTS global.

Pero se agora cambiamos o hostname dentro do noso container:

```
hostname contedor-a-man
```

Veremos que, efectivamente, o noso hostname mudou a "contedor-a-man".

Se abrimos outra terminal na nosa máquina, e facemos hostname, veremos que conserva o hostname orixinal.

Isto é posible porque, tra-la chamada a **unshare**, o noso container ten un UTS diferente (privado) con respecto ó global.

4. Conclusión

Fomos quen de crear un proceso que está dentro dun contedor (dun conxunto de namespaces privados a ese proceso).

Deste xeito, accións que normalmente afectarían á tódolos procesos da máquina:

- Cambiar o hostname
- Creación de usuarios
- Facer mounts
- Conectar aplicacións a escoitar en portos concretos

Están illados dentro dos namespaces do contedor sen afectar á máquina.

Por último, notar que a creación do contedor levou prácticamente o mesmo tempo que lanzar un proceso novo (faga a proba) e que o custo en memoria e insignificante.

Cuestionario

Preguntas de Elección Múltiple

Pregunta

¿Un SO illa completamente os recursos que expón aos procesos que corren nel?

Respuestas

Sí, totalmente, é a sua misión principal

Non, existen unha serie de recursos (puntos de montaxe, usuarios, pids, IPC...) que siguen sendo globais.

Un SO non illa recursos.

Retroalimentación

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)

Pregunta

A técnica da virtualización a nivel de plataforma...

Respuestas

Busca emular con software o hardware dunha máquina para crear un computador virtual

É moi pouco costosa en térrmos de tempo de arranque e de uso de CPU/RAM

É incompatible con calquer outra forma de virtualización

Retroalimentación

Opción correcta

Incorrecto

Incorrecto

Solución

1. Opción correcta (Retroalimentación)
2. Incorrecto (Retroalimentación)
3. Incorrecto (Retroalimentación)

Pregunta

Os contedores de software son unha técnica de virtualización...

Respuestas

Que busca emular o hardware de forma similar ao que fan as técnicas de virtualización de plataforma

Ofrece unha vista "privada" dunha serie de recursos tradicionalmente globais de forma que o proceso, ou grupo de procesos, ten a ilusión de ter un SO para él

Non pode usarse en máquinas virtuais

Retroalimentación

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto (Retroalimentación)
2. Opción correcta (Retroalimentación)
3. Incorrecto (Retroalimentación)