Xestión de contedores con Docker

Sitio: Aula Virtual IES Aller Ulloa - Informática
Curso: Deseño de Interfaces Web (2021-2022)
Libro: Xestión de contedores con Docker

Impreso por: Alberto Méndez Taboada
Data: Wednesday, 20 de October de 2021, 13:43

Táboa de contidos

1. Que son os contedores?

Nos últimos anos, a administración de sistemas está sufrindo un cambio moi importante:

- Que pasaría se se puidera crear máquinas virtuais con custe practicamente nulo en tempo e computación?
- Que ocorrería se nunha máquina puidésemos 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, podendo correr en servidores físicos, servidores virtuais, clouds privadas e públicas ou no propio equipo do desenvolvedor?

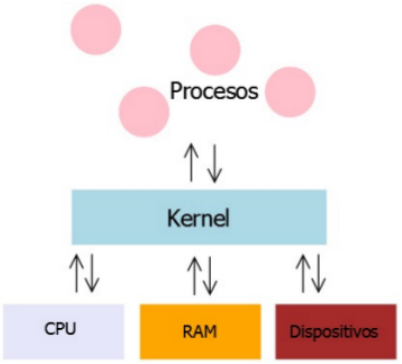
Todo iso é o que posibilita a nova tecnoloxía dos **contedores de software**.

O cambio é dun calado tal que afecta tamén á forma de construír e programar as aplicacións e servizos. Estamos ante un novo paradigma que introduce, por fin, na industria unha nova arquitectura: a dos **microservizos**.

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 isto, créase unha abstracción fundamental: os **recursos**. 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.

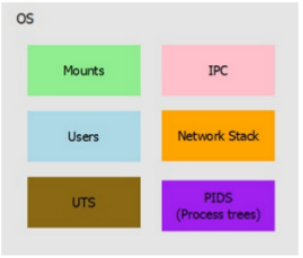


Ademais, o kernel leva a cabo unha importante labor de homoxeneización, ocultando os detalles do funcionamento dos diferentes dispositivos e ofrecendo aos procesos interfaces limpas e unificadas que, estandarizan o emprego de hardware de distinta procedencia. Podemos concluir, por tanto, que os procesos non "ven" os dispositivos que conforman a máquina onde corren, senón a representación que dos mesmos lles ofrece o kernel.

Pero, logran os SO illar todos os recursos dunha máquina? Non. Alomenos nos entornos UNIX, existen unha serie de recursos que son comúns e, polo tanto, globais para todos os procesos.

Estes 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

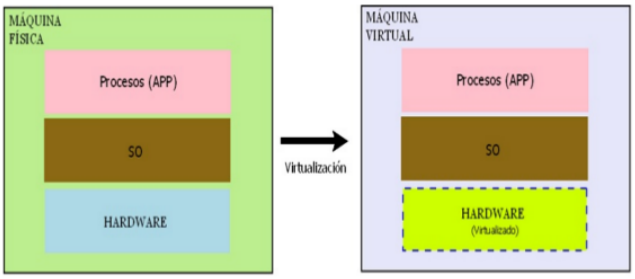


Isto implica que nun sistema non pode haber dous procesos co mesmo PID, ou dos procesos que vexan dos hostnames diferentes na mesma máquina, ou teñan acceso a puntos de montaxe distintos...

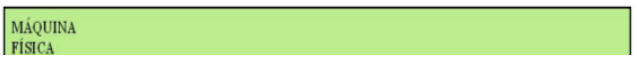
Solución parcial: virtualización da plataforma

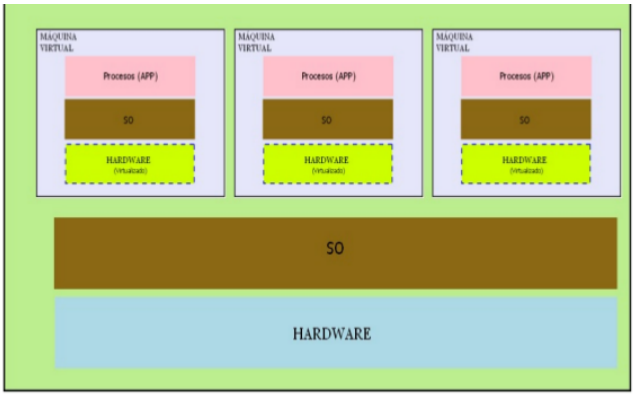
Unha das solucións que se empregan para evitar o problema da falta de illamento do SO dos recursos globais, é a virtualización a nivel de plataforma.

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



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



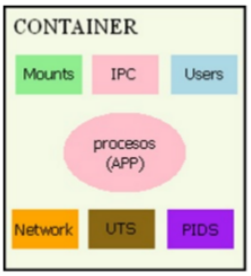


Aínda que esta solución ten virtudes importantes (mellor aproveitamento do hardware, automatización dos despregamentos, portabilidade de MVs, etc.):

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

Solución completa: namespaces e cgroups

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 deses recursos:



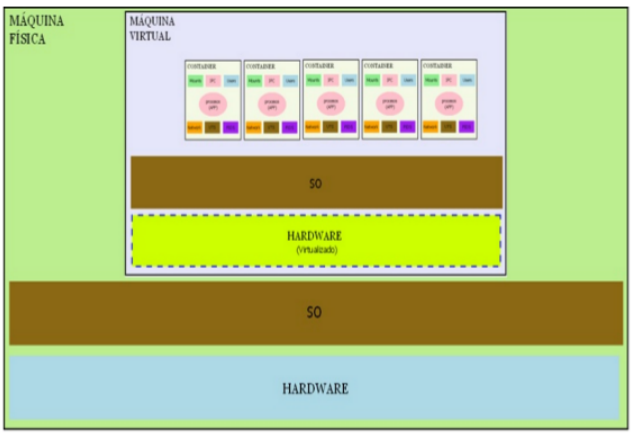
A virtualización deses recursos globais de tal forma que, desde o punto de vista do proceso, sexan privados para el, é no que consiste un contedor. De igual xeito que na virtualización a nivel de plataforma o SO "cree" estarse executando nunha máquina real, na conterización, o proceso "cree" ter un SO para sí mesmo". A técnica de usar contedores é superior a da virtualización de plataforma en que:

- Non supón un custe de recursos adicionais por tener que emular hardware e correr nel un SO: Pódense ter milleiros de contedores nun servidor.
- O arranque/parada dun container é practicamente igual ao arranque/parada dun proceso(<1'').

A costa de:

- Compartir o Kernel do SO.

A conterización non é incompatible coa virtualización da plataforma; ao contrario, empréganse en moitos casos arquitecturas como esta:



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

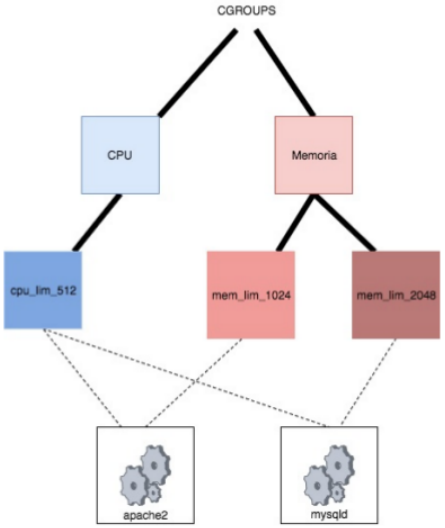
A pregunta que xorde é: permite un control fino de acceso e consumo destes recursos?

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.

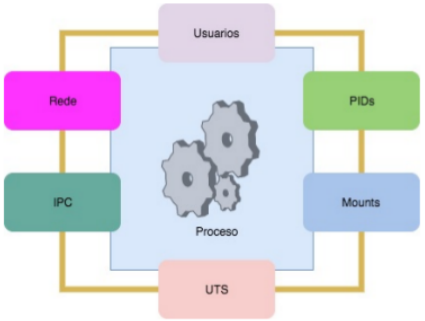
- Xestión a baixo nivel de procesos.

A flexibilidade que permiten é moi grande. Pódense crear distintos grupos de limitacións e control e asignar un proceso, e os seus fillos, adistintos grupos, facendo combinacións que permiten un grao moi alto de personalización.



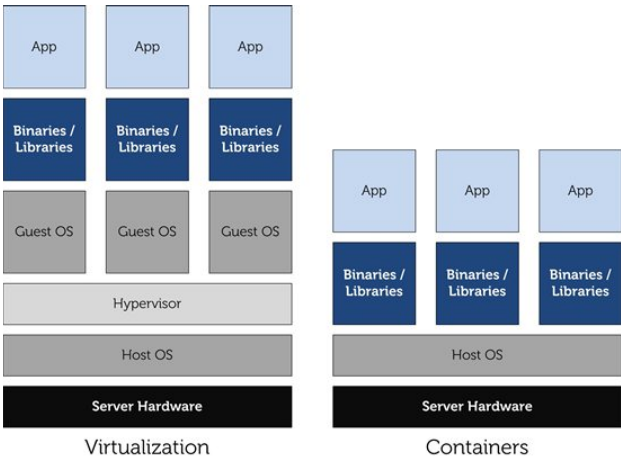
Conclusión

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

Na seguinte imaxe extraída de community.dell.com podemos ver o funcionamento dos contedores fronte á virtualización:



2. Crear un contedor con chroot e namespaces

Empregando as ferramentas que nos da Linux, imos construír un contedor para unha distro de debian (a práctica está baseada na que se pode atopar neste enlace).

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.

Nun directorio do noso sistema de ficheiros, descargamos un 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 zxvf rootfs.tar.gz
```

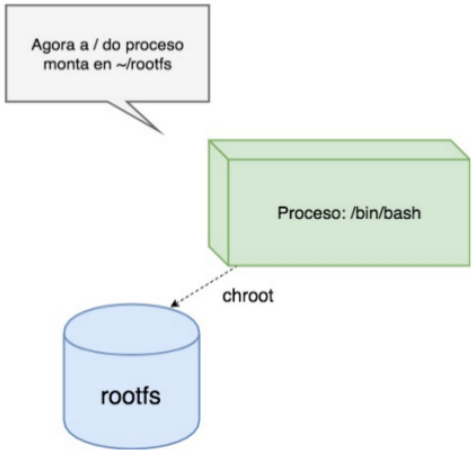
Se listamos os contidos do rootfs resultante, veremos que ten unha estrutura moi similar á dun sistema tradicional debian.

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
```

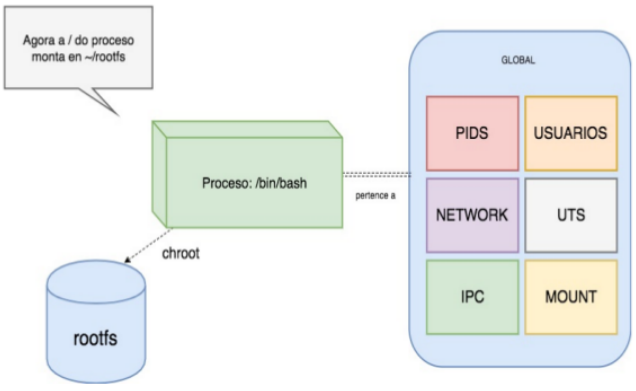
Entraremos nunha nova shell, un novo proceso, que está montado a partires de *rootfs*.



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



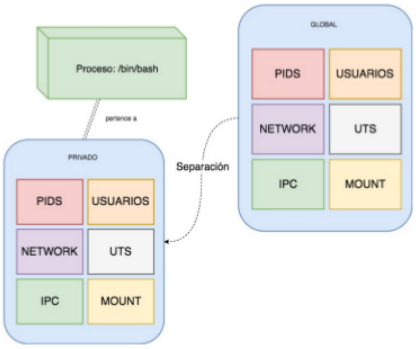
Como podemos ver, este proceso non está realmente "contido":

- Pode crear usuarios no *namespace* xeral da máquina
- Pode ver os procesos de toda a máquina
- Se modifica iptables, se conecta a portos.. estará afectando ao resto dos procesos da máquina

Isto non é un verdadeiro illamento do noso proceso. Para logralo, imos recorrer aos *namespaces*.

Illar o proceso mediante os *namespaces*

Se queremos realmente illar o proceso do resto do sistema, 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 ao sistema:

- Lanza o proceso *chroot rootfs /bin/bash*
- Illa en namespaces novos de mounts, de ipc, de networking, de pids, de UTS ao 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 e, o proceso que creou ese namespace, foi o noso /bin/bash.

Imos facer outra cousa:

```
hostname
```

Veremos o hostname da nosa máquina. Iso é porque 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, tras a chamada a *unshare*, o noso container ten un UTS diferente (privado) con respecto ao global.

3. LXC/LXD

LXC (Linux Containers) é unha tecnoloxía de virtualización a nivel de sistema operativo para Linux. LXC permite que un servidor físico execute múltiples instancias de sistemas operativos Linux illados, cada un co seu propio sistema de ficheiros, espazo de procesos e redes. Pode verse a máquina virtual (contedor) como se unha parte do sistema operativo correse illada do resto, como unha xaula ou *chroot*.

Canonical, a empresa tras Ubuntu, usa contedores dende hai tempo para probas de desevolvemento e lanzou a finais de 2015 LXD, que trata de proporcionar unha mellor interface para traballar con contedores LXC.

4. Instalación e inicialización de LXD

Storage Backends

LXD pode almacenar os contedores e imaxes en directorios (storage de tipo *dir*) e deste xeito todo ó relativo a un contedor está asociado a un cartafol. Trátase da solución máis sinxela; sen embargo, comparada coas outras alternativas é lenta e pouco eficiente á hora de crear clons e instantáneas ao ter que copiar todo o sistema de ficheiros do contedor. LXD pode traballar con sistemas de ficheiros como ZFS, btrfs, LVM e CEPH aproveitando, cando sexa posible, as súas características avanzadas para obter un mellor rendemento.

Os desenvolvedores de LXD sinalan a ZFS e Btrfs como mellores opcións; e no caso de que o sistema teña soporte para ZFS, esta sería a mellor opción. Outra recomendación dos desenvolvedores é a de adicar un disco enteiro ou unha partición para o storage pool. LXD permite crear '*loop based storage*' (arquivo dun tamaño determinado que funciona como un disco co sistema de ficheiros desexado) e aínda que non a recomendan para sistemas en producción funciona moi ben.

Instalando o soporte para ZFS en Ubuntu

Aínda que nos repositorios hai dispoñible unha versión de LXD é recomendable facer a instalación vía paquete snap para ter acceso á máis recente. En Ubuntu 16.04 e posteriores hai que executar o seguinte comando para ter soporte para os paquetes snap e instalar ZFS:

```
$ sudo apt-get install zfsutils-linux snapd
```

Instalando e inicializando LXD

Instalamos LXD vía paquete snap:

```
$ sudo snap install lxd
```

Para facer a configuración inicial de LXD executamos o comando ***lxd init*** e indicamos que:

- A instalación non forma parte dun cluster LXD.
- Queremos crear un novo **storage backend** (onde se almacenarán os contedores e as imaxes) de tipo **ZFS** sen usar un dispositivo de bloque xa existente (un segundo disco duro ou partición). Fixarse que ao ter instalado o soporte para ZFS o tipo por defecto para o *storage backend* é ZFS.
- Indicamos o tamaño para o *loop device* que será o storage backend.
- Non queremos conectarnos a un servidor

.

- Queremos crear unha nova interface de rede **lxdbr0**, que actuará coma un switch-router-nat que dará saída ao exterior aos contedores (semellante ó modo rede NAT de Virtualbox).
- Non queremos que o servizo LXD sexa accesible por rede dende outros equipos.
- Queremos que as imaxes (plantillas a partir das cales se crean os contedores) se actualicen automaticamente.

Aceptamos as opcións por defecto salvo na configuración do *storage backend* onde indicaremos un tamaño axeitado para ó *loop device*, acorde ó tamaño do disco duro do equipo. No exemplo, o novo storage backend de tipo ZFS de tipo *loop based storage* é de 25 GB:

```
$ lxd init

Would you like to use LXD clustering? (yes/no) [default=no]:
Do you want to configure a new storage pool? (yes/no) [default=yes]:
Name of the new storage pool [default=default]:
Name of the storage backend to use (btrfs, ceph, dir, lvm, zfs) [default=zfs]:
Create a new ZFS pool? (yes/no) [default=yes]:
Would you like to use an existing block device? (yes/no) [default=no]:
Size in GB of the new loop device (1GB minimum) [default=15GB]: 25
Would you like to connect to a MAAS server? (yes/no) [default=no]:
Would you like to create a new local network bridge? (yes/no) [default=yes]:
What should the new bridge be called? [default=lxdbr0]:
What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:
What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:
Would you like LXD to be available over the network? (yes/no) [default=no]:
Would you like stale cached images to be updated automatically? (yes/no) [default=yes]
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

Para rematar, os usuarios que van traballar con LXD teñen que pertencer ó grupo *lxd* polo que procedemos a engadir ao noso usuario a ese grupo:

```
sudo usermod -a -G lxd administrador
```

Como a información sobre os grupos ós que pertence un usuario é lida no momento de iniciar a sesión, pode ser necesario saír e volver a entrar.

lxdbr0 e pool ZFS

Durante o proceso de inicialización creouse unha interfaz de rede `lxdbr0` que servirá para ter unha rede que dará servizo aos contedores permitindo a comunicación entre eles e a saída a Internet. A IP asignada pode verse facendo un `ifconfig` e variará dunha instalación a outra.

Tamén podemos coñecer información sobre o pool ZFS creado durante a instalación; e como podemos ver trátase dun arquivo, e non dun disco duro/partición:

```
$ lxc storage list
+---------+-------------+--------+-------------------------------------------+---------+
|  NAME   | DESCRIPTION | DRIVER |                  SOURCE                   | USED BY |
+---------+-------------+--------+-------------------------------------------+---------+
| default |             | zfs    | /var/snap/lxd/common/lxd/disks/default.img | 1       |
+---------+-------------+--------+-------------------------------------------+---------+

$ sudo zpool list
NAME      SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
default    26G   840M  25,2G         -     1%     3%  1.00x  ONLINE  -

$ sudo zpool status
  pool: default
 state: ONLINE
  scan: none requested
config:

    NAME                                        STATE     READ WRITE CKSUM
    default                                     ONLINE       0     0     0
      /var/snap/lxd/common/lxd/disks/default.img  ONLINE       0     0     0

errors: No known data errors

$ sudo zfs list
NAME                    USED  AVAIL  REFER  MOUNTPOINT
default                 729K  25,2G    24K  none
default/containers       24K  25,2G    24K  none
default/custom           24K  25,2G    24K  none
default/deleted          48K  25,2G    24K  none
default/deleted/images   24K  25,2G    24K  none
default/images           24K  25,2G    24K  none
default/snapshots        24K  25,2G    24K  none
magasix@magasix:~$
```

5. Crear e traballar con contedores en LXD

Repositorios de imaxes para contedores LXD

Os contedores lánzanse a partir de imaxes aloxadas en repositorios locais ou remotos; é dicir, que cando creamos un contedor úsase unha destas imaxes para construir unha nova 'máquina virtual' correndo a distribución escollida. LXD permite xestionar os contedores, as imaxes e os seus repositorios. Para ver os repositorios dispoñibles executamos o comando:

```
$ lxc remote list
+-----------------------+-----------------------------------------+---------------+--------+--------+
|         NOMBRE        |                   URL                   |   PROTOCOLO   | PUBLIC | STATIC |
+-----------------------+-----------------------------------------+---------------+--------+--------+
| images                | https://images.linuxcontainers.org      | simplestreams | SÍ     | NO     |
+-----------------------+-----------------------------------------+---------------+--------+--------+
| local (predeterminado) | unix://                                 | lxd           | NO     | SÍ     |
+-----------------------+-----------------------------------------+---------------+--------+--------+
| ubuntu                | https://cloud-images.ubuntu.com/releases | simplestreams | SÍ     | SÍ     |
+-----------------------+-----------------------------------------+---------------+--------+--------+
| ubuntu-daily          | https://cloud-images.ubuntu.com/daily    | simplestreams | SÍ     | SÍ     |
+-----------------------+-----------------------------------------+---------------+--------+--------+
```

Temos tres repositorios remotos e un local (na propia máquina onde está correndo lxd). Se queremos ver as imaxes dispoñibles nun repositorio executamos o comando: `lxc image list <repositorio>`: Por exemplo, para ver as imaxes dispoñibles no repositorio *local*:

```
$ lxc image list local:
+-------+-------------+--------+-------------+-----+--------+-------------+
| ALIAS | FINGERPRINT | PUBLIC | DESCRIPCIÓN | ARQ | TAMAÑO | UPLOAD DATE |
+-------+-------------+--------+-------------+-----+--------+-------------+
```

Podemos apreciar que aínda non temos ningunha imaxe no repositorio local; sen embargo, no repositorio oficial de Ubuntu si hai unha boa cantidade delas (`lxc image list ubuntu:`). Por exemplo, hai imaxes para crear contedores Ubuntu 14.04 LTS (*Trusty Tahr*) e Ubuntu 16.04 LTS (*Xenial Xerus*) de 64 bits. No caso de estar interesados noutras distribucións podemos recorrer ao repositorio *images* onde hai imaxes de Debian, Alpine, Fedora, Ubuntu, ...

Creando o noso primeiro contedor

Unha vez visto que temos á nosa disposición as imaxes nos repositorios remotos procederemos a crear un contedor. O noso contedor chamarase *contedor00* e será un Ubuntu 16.04 LTS de 64 bits creado a partir da imaxe do repositorio oficial de Ubuntu:

```
$ lxc launch ubuntu:x contedor00
Creando contedor00
Iniciando contedor00
```

O que aconteceu foi o seguinte:

- Accedemos ó repositorio *ubuntu* (https://cloud-images.ubuntu.com/releases) a buscar unha imaxe chamada *x* (fixarse na columna ALIAS da listaxe de imaxes do repositorio Ubuntu). No canto de usar o ALIAS tamén podemos usar o FINGERPRINT para indicar a imaxe na que estamos interesados: `lxc launch ubuntu:018d083aec13 contedor00`
- Procédese a descargala ao equipo local.
- Despois úsase para crear un contedor chamado contedor00.

Unha vez rematado o proceso podemos ver que o contedor foi creado e está operativo, recibindo a configuración de rede dende ese 'router-nat' *lxdbr0* creado no *lxd init*:

```
$ lxc ls
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
|  NOMBRE    | ESTADO  | IPV4                 |                     IPV6                      |    TIPO    | SNAPSHOTS |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor00 | RUNNING | 10.122.236.82 (eth0) | fd42:c0b9:6787:9563:216:3eff:fec9:a944 (eth0) | PERSISTENT |           |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
```

Neste momento temos funcionando un contedor que a efectos prácticos é equivalente a unha máquina virtual Ubuntu Server 16.04 de 64 bits co seu propio sistema de ficheiros, configuracións, aplicacións, ...

Creando o noso segundo contedor

No punto anterior descargamos a imaxe plantilla do Ubuntu 16.04 de 64 bits para crear o contedor contedor00. Como esta descarga ralentiza o proceso de creación de contedores automaticamente procedeuse a gardar unha copia desta imaxe no repositorio local. A próxima vez que creemos contedores Ubuntu 16.04, xa non descargaremos a imaxe dende Internet; senón que empregaremos a copia local, acelerando todo o proceso.

```
$ lxc image list local:
+-------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
| ALIAS | FINGERPRINT  | PUBLIC |                DESCRIPCIÓN              |  ARQ   |  TAMAÑO  |         UPLOAD DATE          |
+-------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
|       | 018d083aec13 | no     | ubuntu 16.04 LTS amd64 (release) (20181114) | x86_64 | 158.12MB | Nov 14, 2018 at 12:00am (UTC)|
+-------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
$ lxc launch 018d083aec13 contedor01
Creando contedor01
Iniciando contedor01
magasix@lxd:~$ lxc ls
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
|  NOMBRE    | ESTADO  |        IPV4          |                     IPV6                      |    TIPO    | SNAPSHOTS |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor00 | RUNNING | 10.122.236.82 (eth0) | fd42:c0b9:6787:9563:216:3eff:fec9:a944 (eth0) | PERSISTENT |           |
```

```
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor01 | RUNNING | 10.122.236.224 (eth0)| fd42:c0b9:6787:9563:216:3eff:fecd:78fc (eth0) | PERSISTENT |           |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
```

Poñendo un alias ás imaxes

É posible asignar un nome (alias) ás imaxes en vez de traballar co fingerprint á hora de crear contedores. Podemos consultar a axuda do comando escribindo *lxc image*.

```
$ lxc image alias create xenial64 018d083aec13
$ lxc image list local:
+----------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
|  ALIAS   | FINGERPRINT  | PUBLIC |                DESCRIPCIÓN              |  ARQ   |  TAMAÑO  |         UPLOAD DATE          |
|
+----------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
| xenial64 | 5b8c11faa8e2 | no     | ubuntu 14.04 LTS amd64 (release) (20180913) | x86_64 | 121.77MB | Oct 6, 2018 at 5:36pm (UTC)
|
+----------+--------------+--------+-----------------------------------------+--------+----------+------------------------------+
$ lxc launch xenial64 contedor02
Creando contedor02
Iniciando contedor02
$ lxc ls
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
|  NOMBRE    | ESTADO  |        IPV4          |                    IPV6                       |    TIPO    | SNAPSHOTS |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor00 | RUNNING | 10.122.236.82 (eth0)  | fd42:c0b9:6787:9563:216:3eff:fec9:a944 (eth0) | PERSISTENT |           |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor01 | RUNNING | 10.122.236.224 (eth0) | fd42:c0b9:6787:9563:216:3eff:fecd:78fc (eth0) | PERSISTENT |           |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
| contedor02 | RUNNING | 10.122.236.65 (eth0)  | fd42:c0b9:6787:9563:216:3eff:fe76:992c (eth0) | PERSISTENT |           |
+------------+---------+----------------------+-----------------------------------------------+------------+-----------+
```

Arrincar e parar os contedores

Os contedores pódense parar usando o comando `lxc stop <contedor>`

Para arrincar contedores úsase o comando `lxc start <contedor>`

Se en vez dun único contedor indicamos varios, é posible arrincar/parar varios contedores á vez.

Co comando time podemos comprobar que os tempos de creación, inicio e parada dos contedores son mínimos (*`time lxc lauch|stop|start`*).

Borrar contedores

Para eliminar un contedor hai paralo en primeiro lugar e despois proceder ao seu borrado con `lxc delete` ou forzar o borrado sen apagalo con `lxc delete -f <contedor>`

Pódense indicar varios contedores para borrar varios á vez.

Usando os contedores

Temos que ver un contedor como unha máquina virtual onde poder executar comandos, correr servizos e aplicacións, ... De querer executar un comando nos contedores podemos usar **`lxc exec <contedor> -- <comando>`**

```
magasix@lxd:~$ lxc ls
+----------+---------+----------------------+-----------------------------------------------+------------+-----------+
|   NAME   |  STATE  |         IPV4         |                    IPV6                       |    TYPE    | SNAPSHOTS |
+----------+---------+----------------------+-----------------------------------------------+------------+-----------+
| xenial00 | RUNNING | 10.122.236.3 (eth0)   | fd42:c0b9:6787:9563:216:3eff:fe12:acbf (eth0) | PERSISTENT |           |
+----------+---------+----------------------+-----------------------------------------------+------------+-----------+
| xenial01 | RUNNING | 10.122.236.209 (eth0) | fd42:c0b9:6787:9563:216:3eff:fe0e:9906 (eth0) | PERSISTENT |           |
+----------+---------+----------------------+-----------------------------------------------+------------+-----------+
magasix@lxd:~$ lxc exec xenial00 -- date
Sat Oct  6 18:49:14 UTC 2018
magasix@lxd:~$ lxc exec xenial00 -- free -h
              total        used        free      shared  buff/cache   available
Mem:           2.0G         17M        1.9G          0B         64M        1.9G
Swap:          2.9G          0B        2.9G
magasix@lxd:~$ lxc exec xenial00 -- ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:16:3e:12:ac:bf
          inet addr:10.122.236.3  Bcast:10.122.236.255  Mask:255.255.255.0
          inet6 addr: fd42:c0b9:6787:9563:216:3eff:fe12:acbf/64 Scope:Global
          inet6 addr: fe80::216:3eff:fe12:acbf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:305 errors:0 dropped:0 overruns:0 frame:0
          TX packets:159 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:366976 (366.9 KB)  TX bytes:12664 (12.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

magasix@lxd:~$
```

Se o que desexamos é traballar directamente no contedor podemos lanzar o comando *lxc exec* coa opción *bash* para abrir unha terminal. Desta forma, accederemos á unha liña de comandos dentro do contedor onde poder traballar, ata que salgamos escribindo *exit* (volvendo ó equipo anfitrión).

```
magasix@lxd:~$ lxc exec xenial01 bash
root@xenial01:~# cat /etc/issue
Ubuntu 16.04.5 LTS \n \l

root@xenial01:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# Source interfaces
# Please check /etc/network/interfaces.d before changing this file
# as interfaces may have been defined in /etc/network/interfaces.d
# See LP: #1262951
source /etc/network/interfaces.d/*.cfg

root@xenial01:~# exit
exit
magasix@lxd:~
```

Instantáneas e clonación

Despois de estar traballando cun contedor pode xurdir a necesidade de facer unha instantánea para poder recuperar ese estado un futuro. En lxd usando ZFS as instatáneas son inmediatas e fanse co comando `lxc snapshot`:

```
magasix@lxd:~$ lxc snapshot xenial00 snap_06_10_18
magasix@lxd:~$ lxc info xenial00
Name: xenial00
Remote: unix://
Architecture: x86_64
Created: 2018/10/06 18:09 UTC
Status: Stopped
Type: persistent
Profiles: default
Snapshots:
  snap_06_10_18 (taken at 2018/10/06 21:04 UTC) (stateless)
magasix@lxd:~$
```

Como pode observarse, o comando `lxc info` proporciona información sobre o contedor incluíndo información sobre as instantáneas. No caso de querer recuperar o estado da máquina no momento da instantánea temos que executar o comando: `lxc restore <contedor> <instantánea>`.

Unha limitación que ten o sistema ZFS (que non de LXD) é que únicamente permite recuperar á derradeira instantánea con lxc restore. A solución a este problema é facer un clon a partir da instantánea. Se queremos facer un clon ou copia dun contedor executamos o comando `lxc copy`:

```
magasix@lxd:~$ lxc copy xenial00 clon00
magasix@lxd:~$ lxc ls
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
|   NAME   |  STATE  |         IPV4         |                     IPV6                     |    TYPE    | SNAPSHOTS |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| clon00   | STOPPED |                      |                                              | PERSISTENT | 1         |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| xenial00 | STOPPED |                      |                                              | PERSISTENT | 1         |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| xenial01 | RUNNING | 10.122.236.209 (eth0) | fd42:c0b9:6787:9563:216:3eff:fe0e:9906 (eth0) | PERSISTENT |           |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
magasix@lxd:~$
```

No caso de querer facer unha copia a partir dunha instantánea sería: `lxc copy <contedor>/<instantánea> <nom do clon>`

```
magasix@lxd:~$ lxc copy xenial00/snap_06_10_18 clon01
magasix@lxd:~$ lxc ls
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
|   NAME   |  STATE  |         IPV4         |                     IPV6                     |    TYPE    | SNAPSHOTS |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| clon00   | STOPPED |                      |                                              | PERSISTENT | 1         |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| clon01   | STOPPED |                      |                                              | PERSISTENT |           |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
| xenial00 | STOPPED |                      |                                              | PERSISTENT | 1         |
+----------+---------+----------------------+----------------------------------------------+------------+-----------+
```

```
| xenial01 | RUNNING | 10.122.236.209 (eth0) | fd42:c0b9:6787:9563:216:3eff:fe0e:9906 (eth0) | PERSISTENT |           |
+----------+---------+-----------------------+-----------------------------------------------+------------+-----------+
```

6. Introdución a Docker

Docker é unha plataforma de conterización que:

- Facilita enormemente a xestión de contedores.
- Ofrece utilidades de creación e mantemento de imaxes de contedores.
- Aporta ferramentas de orquestación de contedores.
- Esfórzase por manter unha serie de estándares de conterización.

Docker ten tres compoñentes básicos:

Compoñentes de Docker

- O docker-cli:intérprete de comandos que permite interactuar con todo o ecosistema do Docker.
- Unha API REST: que permite dar resposta aos clientes: tanto o docker-cli como calquera outra libraría ou cliente de terceiros. A API está ben documentada.
- O dockerd: un daemon que corre no host e que xestiona todos os contedores, volumes e imaxes do host.

Estes tres compoñentes forman o **docker engine**. Polo tanto,nós imos instalar o docker-engine nunha máquina e interactuar con el dende o propio docker-cli desa máquina. Pero nada nos impide, coa configuración axeitada, poder interactuar dende o noso docker-cli con docker-daemons doutros hosts.

Na actualidade, na web de Docker podemos atopar dúas versións do docker engine:

- **Docker Engine - Community**: Versión gratuíta e aberta do motor, dispoñible para as principais distribucións de Linux (CentOS, Debian, Fedora e Ubuntu).
- **Docker Engine - Enterprise**: Versión de pago, dispoñible tamén para RedHat Enterprise Linux e Windows Server, que ofrece maior soporte, plugins e funcionalidades de cifrado e sinatura das imaxes.

Pero ademais do motor, Docker ofrece tres produtos máis que completan o funcionamento da plataforma:

- **Docker Enterprise**: É unha plataforma que inclúe a versión enterprise do docker engine e unha serie de módulos adicionais para mellorar a seguridade nas comunicacións e autenticación das imaxes, xestión do ciclo de vida dos contedores, etc.
- **Docker Desktop**: É unha aplicación dispoñible para Windows e Mac que permite utilizar docker de forma sinxela. Tamén se integra coas ferramentas de orquestación *docker swarm* e *kubernetes*.
- **Docker Hub**: É un repositorio de imaxes para poder crear a partir delas os contedores que queiramos.

7. Instalación de Docker

No sitio web de Docker podemos atopar a guía de instalación do Docker Community (CE) sobre distintas plataformas. Se queremos instalalo nun sistema con Ubuntu 16.04 ou superior, seguiremos os seguintes pasos:

Desinstalamos versións anteriores se docker se estivesen instaladas:

```
sudo apt-get remove docker docker-engine docker.io
```

Configuramos o repositorio de docker:

```
sudo apt-get update

# Instalamos os paquetes necesarios para usar un repositorio sobre HTTPS

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

# Engadimos a chave do repositorio de docker.

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Engadimos o repositorio

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Instalamos docker-ce

```
sudo apt-get update

sudo apt-get install docker-ce
```

Podemos comprobar que funciona correctamente executando o noso primeiro contedor:

```
sudo docker run hello-world
```

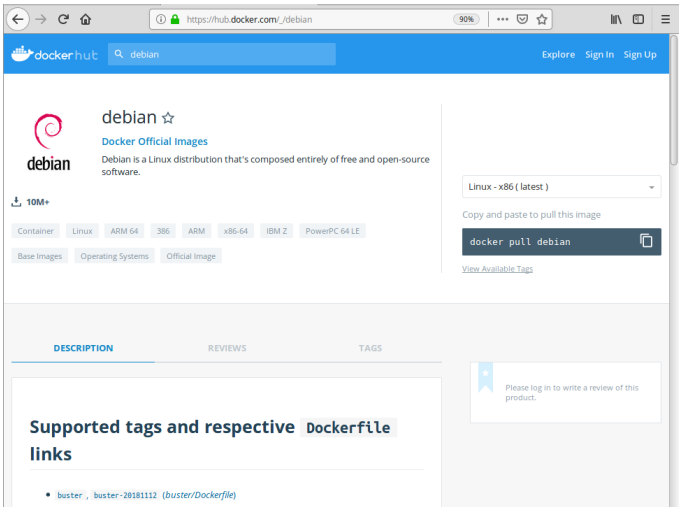
8. Manexo básico de contedores

A interacción con Docker (co seu *daemon*) pódese facer, fundamentalmente por dúas vías:

- A través dunha API REST.
- Mediante a súa liña de comandos: O intérprete de comandos de Docker é o comando `docker`.

Executar un contedor

Podemos executar un primeiro contedor co comando "`docker run`". A ese comando temos que indicarlle a imaxe do contedor que queremos executar, que podemos buscar no docker hub:



No docker hub podemos ver que unha imaxe ademais dun nome ten unha serie de etiquetas (*tags*) asociadas. Estas etiquetas permiten acceder a distintas versións da imaxe; por exemplo neste caso as etiquetas "`jessie`", "`8.11`" e "`8`" descargarían unha Debian 8 mentres "`stretch`" ou "`9`" descargarían unha Debian 9. En moitas imaxes, a etiqueta "`latest`" permite acceder á última versión da imaxe.

Co seguinte exemplo descargaremos do docker hub a imaxe dun contedor coa última versión de Debian e executaremos sobre el o comando "`echo Ola Mundo!`":

```
$ sudo docker run -i debian:latest echo Ola Mundo!
```

Podemos obter axuda das opcións de cada comando de `docker` con "`--help`". Por exemplo, se queremos ver os parámetros que permite "`docker run`":

```
$ docker run --help
```

Listaxe dos contedores do sistema

Podemos listar os contedores existentes nunha máquina con "`docker ps`". Coa opción "`-a`" podemos listar todos os contedores, incluíndo aqueles que xa non están en execución. Desta maneira, despois de executar varias veces contedores da imaxe de debian, podemos ter unha saída como a seguinte:

```
$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS               NAMES
0ec2f2db3074        debian:latest       "echo Ola Mundo!"   13 hours ago        Exited (0) 13 hours ago
dreamy_herschel
7a2a37b8a129        debian:latest       "echo Ola Mundo!"   2 days ago          Exited (0) 2 days ago
practical_sammet
aa9d02e909ec        debian:latest       "echo Ola Mundo!"   2 days ago          Exited (0) 2 days ago
compassionate_kilby
5b055ecd6461        debian:latest       "echo Ola Mundo!"   2 days ago          Exited (0) 2 days ago
inspiring_kalam
```

O que vemos é o seguinte:

- Docker asigna un uuid aos contedores.
- Infórmanos sobre a imaxe que monta cada contedor.
- Indícanos o comando que lanzou este contedor.
- Os seus tempos de arranque (*CREATED*) e o tempo que leva aceso (*STATUS*).
- A conectividade do contedor.
- O nome asignado ao contedor en caso de que non llo poñamos nós.

Creando e arrancando contedores

O comando básico para crear un contedor é "`docker create`". Como mínimo teremos que indicarlle a imaxe que queremos usar para a creación do contedor. Polo tanto, podemos facer:

```
$ sudo docker create debian:latest
3614ad77dca3290587f65838eae1dd3d95f1d8159f27623b4dd063fb82dc17ea
```

A resposta é unha cadea hexadecimal que será o identificador único do contedor creado. Como o contedor non está iniciado, non aparecerá se facemos "`docker ps`", pero si se facemos "`docker ps -a`".

Se queremos arrancar o contedor, usaremos "`docker start`", indicando o uuid do contedor (podemos usar o formato longo do uuid que nos devolve o comando "`create`" como o formato curso que devolve o "`ps`"):

```
$ sudo docker start 3614ad77dca3290587f65838eae1dd3d95f1d8159f27623b4dd063fb82dc17ea
3614ad77dca3290587f65838eae1dd3d95f1d8159f27623b4dd063fb82dc17ea
```

Neste caso, se agora facemos "`docker ps`" tampouco veremos o contedor, xa que despois de executar o comando "bash" o contedor remata a súa execución. Podemos usar outra imaxe como proba que está xa preparada para manter a súa execución e comprobar así o efecto do comando "`start`": prefapp/debian-formacion.

O comando "`run`" nos permite facer a creación e arranque do contedor nun único paso.

Detendo os contedores

Para deter un contedor que está en execución, usamos o comando "`docker stop`" indicando o uuid do contedor. Por exemplo:

```
$ sudo docker stop c7ea90ab4435
c7ea90ab4435
```

Borrando contedores

O comando "`docker rm`" elimina un contedor:

```
$ sudo docker rm c7ea90ab4435
c7ea90ab4435
```

Opcións na execución dos contedores

Xa vimos que se iniciamos un contedor con "`docker run`" podemos indicarlle o comando que queremos executar sobre o contedor. Se queremos iniciar un contedor e utilizalo de xeito interactivo, para poder introducir comandos sobre, podemos usar os parámetros "`-ti`", e indicar como comando a executar un intérprete de comandos como por exemplo "`/bin/bash`". A modo de exemplo, podemos usar o parámetro "--rm" para borrar o contedor cando remate a execución do intérprete de comandos:

```
$ sudo docker run --rm -ti debian:latest /bin/bash
root@4ad094c32820:/# cat /etc/issue
Debian GNU/Linux 9 \n \l

root@4ad094c32820:/# exit
exit
$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
$
```

Outra opción moi habitual é executar un contedor como un servizo (*daemon*), para que estea executándose de forma permanente ata que fagamos un "`docker stop`". Para iso, podemos usar as seguintes opcións:

```
$ sudo docker run --name dserver00 -d debian:latest tail -f /dev/null
8b3ec8f95b2b8290a5de1bf8b1ba69e6379d2286ab61b13ef5ef0a47d6a6564c
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS               NAMES
8b3ec8f95b2b        debian:latest       "tail -f /dev/null"   18 seconds ago      Up 17 seconds                           dserver00
$
```

A destacar que o uso dos seguintes parámetros:

- "`--name nome_contedor`": Desta forma lle damos un nome ao contedor que nos permitirá facer referencia a el sen ter que usar o uuid.
- "`-d`": Para que o contedor se execute en segundo plano.
- "`tail -f /dev/null`": Facemos que o contedor execute este comando para que a súa execución continúe permanentemente.

Se agora queremos executar un comando sobre ese contedor, podemos utilizar "`docker exec`". Por exemplo:

```
$ sudo docker exec -ti dserver00 /bin/bash
root@8b3ec8f95b2b:/# cat /etc/issue
Debian GNU/Linux 9 \n \l

root@8b3ec8f95b2b:/# exit
exit
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS               NAMES
8b3ec8f95b2b        debian:latest       "tail -f /dev/null"   5 minutes ago       Up 5 minutes                            dserver00
$
```

Desta maneira iniciamos unha sesión no intérprete de comandos dentro do proceso do contedor. Como se pode comprobar, agora ao pechar a sesión do intérprete de comandos o contedor continúa a súa execución, que podemos deter con "`docker stop`".

Por último, na execución dun contedor podemos pasar parámetros a variables de contorno con "-e". Por exemplo, o seguinte exemplo asigna no contedor as variables "ROOT_LOGIN" e "ROOT_PASSWORD":

```
$ sudo docker run --name dserver00 -d  -e ROOT_LOGIN=admin -e ROOT_PASSWORD=abc123. debian:latest tail -f /dev/null
```

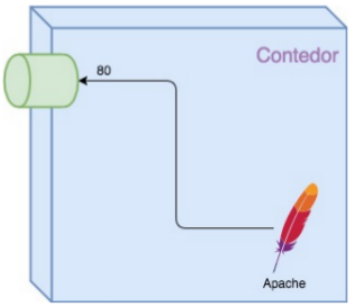
9. Xestión da rede en Docker

Por defecto os contedores Docker poden acceder ao exterior (teñen conectividade entre eles e ao exterior mediante NAT, se a ten a máquina anfitriona). Porén, o contedor está illado con respecto a entrada de datos. É dicir, por defecto, e coa excepción do comando `exec`, o contedor constitúe unha unidade illada á que non se pode acceder. Por suposto, compre poder acceder aos contedores para poder interactuar dalgún xeito con eles.

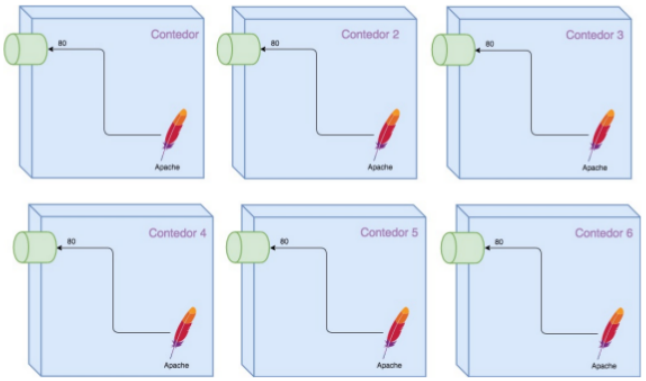
A regra básica aquí é: salvo que se estableza o contrario, todo está pechado ó mundo exterior.

Un contedor está dentro do seu *namespace* de rede polo que pode establecer regras específicas sen afectar ao resto do sistema. Deste xeito, e sempre dende o punto de vista do contedor, podemos establecer as regras que queiramos e conectar o que desexemos aos 65535 do contedor sen temor a colisións doutros servizos.

Así, poderíamos ter un contedor cun apache conectado ó porto 80:



Ou poderíamos ter varios contedores con servizos apache conectados ao porto 80:



Dado que cada un deles te o seu propio *namespace* de rede, non habería ningún problema de colisión entre servizos conectados ao mesmo porto en contedores diferentes.

O problema é que, pese a ter esta liberdade dentro do contedor, aínda precisamos conectar o extremo do contedor cun porto da máquina anfitriona para que as conexións externas poidan chegar ao noso contedor.

Iso poderemos facelo pasándolle ao comando "`docker run`" o parámetro "`-p`". Por exemplo, a continuación móstrase como lanzar un contedor Debian no que se vai instalar o servizo ssh (no porto 22 do contedor), e ao que se conecta o porto 2222 da máquina anfitriona:

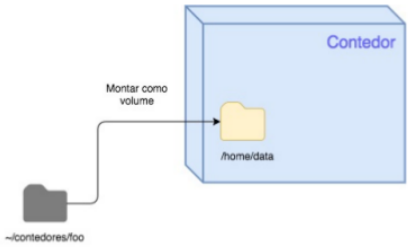
```
$ sudo docker run --name dserver02 -d -p 2222:22 debian:latest tail -f /dev/null
c17f6a488e0f0b7159b9c391afdef33af559cb3658b57135ea136a27b724ee0a
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                   NAMES
c17f6a488e0f        debian:latest       "tail -f /dev/null"   4 seconds ago       Up 3 seconds        0.0.0.0:2222->22/tcp    
dserver02
d14ed340d135        debian:latest       "tail -f /dev/null"   24 hours ago        Up 8 minutes                                
dserver01
$
```

Docker permite configurar outros modos de conexión para os contedores, crear redes para conectalos directamente á rede física cun enderezo MAC distinto ao da máquina anfitriona, etc. Na documentación móstranse os pasos para a configuración.

10. Xestión do almacenamento en Docker

Como norma xeral, non debemos usar os contedores para almacenar datos. Hai que ter en conta que un contedor se crea a partir dunha imaxe, e todo o que gardemos nel desaparecerá cando o contedor se destrúa. Cando creemos un novo contedor a partir da imaxe, eses datos xa non estarán dispoñibles.

Unha das solucións principais para o problema da falta de persistencia dos contedores é a dos volumes. Podemos pensar nun volume como un directorio do noso anfitrión que se "monta" como parte do sistema de ficheiros do contedor. Este directorio pasa a ser accesible por parte do contedor e, os datos almacenados nel, persistirán con independencia do ciclo de vida do contedor.



Para acadar isto, abonda con indicar no comando "`docker run`" co parámetro "`-v`" que directorio do noso anfitrión queremos montar como volume e en que ruta queremos montalo no noso contedor. Nun exemplo:

```
$ sudo docker run --rm -ti -v ~/dserver03_datos:/var/datos debian:latest /bin/bash
root@1157ee88d448:/# echo "Ficheiro persistente" > /var/datos/ficheiro.txt
root@1157ee88d448:/# exit
exit
$ cat dserver03_datos/ficheiro.txt
Ficheiro persistente
$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS
PORTS                  NAMES
c17f6a488e0f        debian:latest       "tail -f /dev/null"   28 minutes ago      Up 28 minutes
0.0.0.0:2222->22/tcp   dserver02
d14ed340d135        debian:latest       "tail -f /dev/null"   24 hours ago        Up 36
minutes                                       dserver01
8b3ec8f95b2b        debian:latest       "tail -f /dev/null"   24 hours ago        Exited (137) 31 minutes
ago                          dserver00
$
```

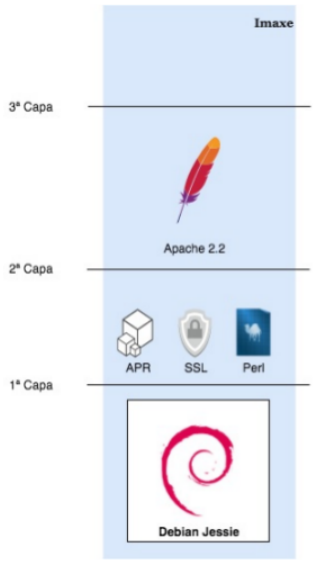
Como vemos, tras crear un contedor da imaxe de debian que se elimina tras a execución dun intérprete de comandos, a montaxe da carpeta "`dserver03_datos`" da máquina anfitriona na carpeta "`/var/datos`" do contedor permite que o ficheiro creado nesa carpeta do contedor persista aínda que o contedor fose destruído.

11. Xestión das imaxes en Docker

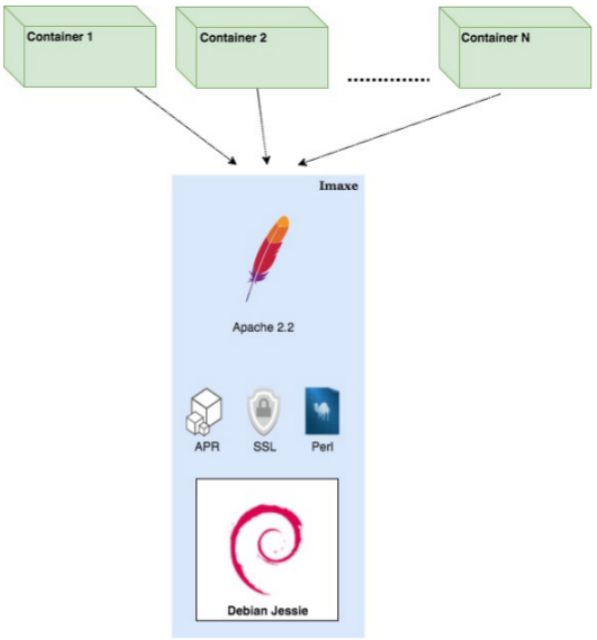
Que é unha imaxe?

A imaxe é un dos conceptos fundamentais no mundo da conterización. Unha imaxe abrangue o conxunto de software específico a empregar polo contedor unha vez arrancado. Intuitivamente, podemos comprender que se trata de algo estático e inmutable, como pasa por exemplo cunha ISO, que temos que ter almacenado na máquina anfitriona para poder lanzar contedores baseados nesa imaxe.

As imaxes en Docker están formadas por **capas** o que permite a súa modularidade e reutilización. Por exemplo, podemos crear unha imaxe para un servidor web Apache partindo dunha imaxe base Debian, sobre a que instalaremos nunha segunda capa as dependencias de software que ten Apache e nunha terceira capa o servidor Apache:



Agora xa podemos empregar esa imaxe para lanzar contedores dela:



Imaxes e contedores

Unha vez que temos unha imaxe, os contedores que se crean a partir dela teñen unha capa de lectura/escritura para que todos os cambios que se poidan facer no sistema de ficheiros do contedor se fagan nesta capa, mentres que a capa da imaxe só é de lectura e non pode ser modificada por ningún dos contedores que se crean dela (mecanismo coñecido como *copy-on-write*):

Capas dos contedores e imaxe

É por iso que non debemos gardar nun contedor datos de importancia e, por normal xeral utilizaremos, os volumes de almacenamento montados nunha carpeta do anfitrión para almacenar os datos relevantes que queremos que persistan.

Comandos para a xestión das imaxes

Para poder iniciar un contedor a partir dunha imaxe, esta ten que estar instalada localmente na nosa máquina. Podemos usar o comando "`docker images`" para ver as imaxes que temos descargadas:

```
$ sudo docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
debian                    latest              4879790bd60d        7 weeks ago         101MB
prefapp/debian-formacion  latest              ea566f92532b        11 months ago       123MB
```

Podemos borrar unha imaxe con "`docker rmi`" (só poderemos borrar unha imaxe se non temos contedores dependentes dela):

```
$ sudo docker rmi prefapp/debian-formacion
Untagged: prefapp/debian-formacion:latest
Untagged: prefapp/debian-formacion@sha256:819f6ef512892c39d3125d9640c91934db969445c95a7f5d3deb2e3106a7ad58
Deleted: sha256:ea566f92532b4965c7d4cd0027da2021bc8933ec95a8e2f985bcde47145965ce
Deleted: sha256:4bcdffd70da292293d059d2435c7056711fab2655f8b74f48ad0abe042b63687
```

As imaxes pódense descargar e subir a un rexistro, que pode ser o Docker Hub ou un rexistro propio creado por nós (Dentro de Docker, o proxecto Docker Registry permite crear sobre un contedor un rexistro privado no que subir as nosas imaxes). Os seguintes comandos permiten realizar as operacións básicas sobre as imaxes:

- `docker pull`: Descarga unha imaxe dun rexistro (en caso de que non se indique ningún rexistro, usarase o Docker Hub).
- `docker push`: Sube unha imaxe a un rexistro.
- `docker login`: Para poder subir unha imaxe, necesitaremos iniciar sesión no rexistro con este comando.
- `docker tag`: Crea unha copia dunha imaxe cunha etiqueta distinta para poder subila a outro rexistro diferente.
- `docker commit`: Crea unha imaxe a partir dun contedor. Isto é útil para poder actualizar unha imaxe, xa que o proceso que podemos facer é o seguinte: Creamos un contedor a partir da imaxe, actualizamos o contedor e facemos un "commit" para crear unha nova versión da imaxe.

Definir imaxes con Dockerfile

Docker ofrece unha ferramenta para a creación de imaxes máis sinxela que facer uso do comando "`docker commit`" para crear unha imaxe a partir dun contedor: O Dockerfile. O Dockerfile é un ficheiro de texto que contén a receita de como construír unha imaxe. O conxunto de instrucións do Dockerfile constitúen un DSL que nos vai permitir expresar dun xeito razoablemente doado os pasos que hai que dar para producir a imaxe.

Unha vez redactado o Dockerfile, basta con empregar o comando "`docker build`" para producir unha imaxe con el. Docker creará (e destruirá) os contedores de traballo que precise para construír a nosa imaxe mantendo unicamente o resultado final: a imaxe que queremos.

A continuación móstrase un exemplo de Dockerfile que permite crear unha imaxe de debian co servizo ssh, e na que se inclúen as instrucións máis frecuentes nun Dockerfile:

```
# FROM: Imaxe da que partimos. Neste caso a imaxe de debian
FROM debian:latest

# Con RUN executamos comandos. Actualizamos a lista de paquetes e instalamos o ssh
RUN apt-get update -y && apt-get install -y ssh

# Pasamos ao directorio /etc/ssh con WORKDIR
WORKDIR /etc/ssh

# Configuramos o servizo ssh, executando de novo comandos con RUN
RUN echo "PermitRootLogin yes" >> sshd_config
RUN mkdir /var/run/sshd

# Con EXPOSE indicamos que o contedor escoitara no porto 22
EXPOSE 22

# CMD e ENTRYPOINT permiten indicar un comando a executar no inicio do contedor
CMD ["/usr/sbin/sshd", "-D"]
```

Así que o que teremos que facer para construír unha imaxe é gardar o contido anterior nun ficheiro co nome "`Dockerfile`" e executar o comando "`docker build`" (a opción -t permite poñer a etiqueta da nova imaxe):

```
$ sudo docker build -t debian-ssh .
```

Podemos probar a imaxe con "`docker run`". Se usamos a opción "`-P`" o contedor exporá o porto indicado no Dockerfile en calquera porto libre da máquina anfitrión. Con "`docker port`" podemos ver cal é o porto asignado e usalo para conectarnos por ssh:

```
$ sudo docker run --name dserver00 -d -P debian-ssh
370d9d56adfc89d4736337af889cc80da07ed7067c9c03720968f10b9121fdb1
$ sudo docker port dserver00
22/tcp -> 0.0.0.0:32768
$ ssh -p 32768 root@localhost
```

Se queremos utilizar un porto concreto da máquina anfitrión, podemos usar "`-p`":

```
$ sudo docker run --name dserver01 -d -p 2222:22 debian-ssh
ed654729c7f48fc507c9a3561fac2366b3f73e8d1d6c39819a0f7e4fb3051878
$ sudo docker port dserver01
22/tcp -> 0.0.0.0:2222
$ ssh -p 2222 root@localhost
```

12. Orquestración de servizos multicontedor

Por que precisamos a orquestración?

Co uso de contedores, a idea é a de romper o noso sistema en unidades funcionais pequenas e manexables que se comunican entre elas para facer un traballo máis grande. Un símil ao que se recorre moitas veces é ó da programación orientada a obxectos:

- Igual que na POO temos clases e instancias, na conterización temos imaxes e obxectos.
- A idea clave é encapsular funcionalidades en unidades independentes que falan entre sí mediante paso de mensaxes.
- A evolución do sistema faise mediante a extensión/creación de novas clases e obxectos non mediante a modificación dos existentes.

Polo tanto, a conterización introduce un novo paradigma de deseño de aplicacións e infraestruturas no que:

- O sistema fragméntase en unidades moi pequenas, sendo o bloque mínimo de construción o contedor.
- Cada unidade ten unha funcionalidade concreta e ben definida (Separación de intereses).
- Acadando esta modularidade, as diferentes partes pódense ver como **microservizos** que expoñen una interface clara para comunicarse e que traballan conxuntamente para asegurar o cumprimento dos obxectivos do sistema como entidade.
- Ademais as nosas aplicación son por fin escalables horizontalmente, engandindo novas instancias (contedores) podemos aumentar a potencia da nosa aplicación sen necesidade de recorrer á gaiola do escalado vertical.

Imos comenzar conha simple aplicación en Php dentro dun contedor:

orquestación

Queremos que a nosa aplicación teña **estado**. A solución obvia é agregar unha base de datos dentro do container:

orquestación

Isto, a pesar de que sería unha solución obvia, é unha ruptura do paradigma da containerización:

- O container non ten unha única preocupación, ten dúas (xestión de bbdd e aplicación en Php).
- O cambio na bbdd ou na aplicación implica cambiar o resto de unidades.
- Introducimos dependencias do software de Mysql con respecto ao Php e viceversa.

O problema sería peor se queremos, por exemplo, engadir soporte para SSL, un servidor web, soporte para métricas e logs... O noso contedor crecería e crecería, polo que non amañariamos ningún dos problemas que queremos resolver o por riba perderíamos as vantaxes da conterización.

A solución axeitada sería esta:

orquestación

Agora temos dúas unidades independentes en dous containers. Podemos modificar unha sen afectar á outra. As preocupacións están correctamente separadas.

Neste momento, os nosos containers poden ademais escalar, basta con engadir novos containers de aplicación se é necesario:

orquestación

E, por suposto, podemos engadir os servicios auxiliares que creamos convenientes, sen necesidade de modificar os contedores que xa temos (encapsulación)

orquestación

Como podemos ver, a nosa aplicación crece engadindo novas unidades funcionais, non modificando as existentes. Pero, xorden preguntas:

- Como coñece o noso container de Php a dirección e o porto no que escoita o Mysql?
- Como facemos para parar/arrincar todos os containers de vez?
- Se un container cae e ao levantalo cambiou de IP, como o sabe o resto?
- Como inxectamos configuracións comúns a todos os containers?

A estas e outras moitas preguntas trata de dar resposta a **orquestración de contedores**, que é un conxunto de técnicas e ferramentas que buscan asegurar:

- O correcto aprovisionamento de hosts.
- A correcta xestión (arrincado, detención) dun grupo de containers.
- O re-arrincado de containers caídos.
- A conexión de containers a través dunha serie de interfaces estándar ou ben establecidas.

- A correcta conexión de determinados containers ao mundo exterior de tal xeito que poidan comunicarse cos clientes do sistema.
- O escalado e dimensionamento do grupo de containers de tal xeito que se creen e destrúan containers segundo as necesidades puntuais do sistema en cada momento.

A orquestración con Docker-compose

O docker-compose é unha ferramenta de Docker deseñada para permitir unha orquestración a nivel dunha soa máquina. O funcionamento é relativamente sinxelo:

- Os distintos containers da aplicación organízanse en **servizos**.
- Pódense crear **redes privadas** para conectar os containers da aplicación.
- Aporta a posibilidade de crear volumes para aportar persistencia.

O docker-compose aporta unha DSL que permite expresar estas funcionalidades nun ficheiro de YAML que despois interpreta para crear a nosa infraestrutura tal e como a establezamos.

Dentro do noso compose, imos definir tres entidades fundamentais:

- **Servizos**: Trátase dunha definición dos contedores, no que se establece a imaxe a montar, as variables de contorno, o número de réplicas do contedor a correr, os portos de conexión e os volumes que monta.
- **Redes**: Son unha construción que permite que os distintos contedores pertencentes se "vexan" uns ós outros sen necesidade de coñecer as súas IPs. Imaxinemos que temos un servizo en PHP que se conecta a outro que corre unha bbdd en Mysql. Se o metemos nunha rede común, o servizo de PHP vai a ter definido un host (bbdd) que coincide co nome do servizo de bbdd sen ter que preocuparse da IP do contedor.
- **Volumes**: Os volumes e almacenamento poden ser directamente definidos no docker-compose a través da propia DSL da ferramenta.

Instalación de Docker-compose

Na documentación de Docker podemos atopar as instrucións para a instalación de Docker-compose. Por exemplo, os seguintes comandos nos permitirían instalalo sobre Linux:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.25.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local
/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Podemos comprobar que temos Docker-compose instalado con:

```
$ sudo docker-compose --version
```

Exemplos de uso de Docker-compose

Imos probar un exemplo sinxelo do uso de Docker-compose usando a imaxe oficial de wordpress que se pode atopar no Docker Hub. Para que o contedor de wordpress funcione correctamente, precisa de outro contedor que execute un xestor de base de datos mysql, así que creamos unha carpeta e creamos un ficheiro "`docker-compose.yml`" co seguinte contido que se propón na propia documentación da imaxe:

```
version: '3.1'

services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb

  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
```

Basicamente o que se fai é definir un servizo con dous contedores, "`wordpress`" e "`db`", partindo das imaxes "`wordpress`" e "`mysql:5.7`", conectando o porto 80 do contedor de wordpress ao porto 8080 da máquina anfitriona e configurando unha serie de variables de contorno cos valores dos usuarios, contrasinais e base de datos a usar. Neste caso non se definen nin redes nin volumes externos.

Para iniciar o servizo definido no ficheiro, executamos o comando:

```
$ sudo docker-compose up -d
```

Unha vez iniciado o servizo, podemos ver os contedores en execución con:

```
$ sudo docker-compose ps
           Name                         Command               State          Ports
-------------------------------------------------------------------------------------------
probas-compose_db_1          docker-entrypoint.sh mysqld      Up      3306/tcp, 33060/tcp
probas-compose_wordpress_1   docker-entrypoint.sh apach ...   Up      0.0.0.0:8080->80/tcp
```

E podemos comprobar que wordpress está dispoñible conectándonos a "http://localhost:8080"

Podemos deter o servizo con:

```
$ sudo docker-compose stop
```

Neste enlace pódese atopar un exemplo dun ficheiro docker-compose máis completo que ademais de servizo define distintas redes e volumes de almacenamento.

13. Orquestración de clusters con Docker Swarm

Alternativas para a creación de clusters

Ata o de agora vimos como lanzar contedores nun único anfitrión Docker. Imos agora a abordar Docker Swarm como unha opción integrada dentro do propio Docker engine, para lanzar contedores e orquestralos nun cluster de anfitrións Docker (nun conxunto de máquinas que actúan coma se fosen unha).

Kubernetes, a alternativa open source de Google, é outra opción moi utilizada para a orquestración de contedores nun cluster de máquinas. Intégrase perfectamente con Docker, así que son dúas opcións posibles para a configuración de clusters de contedores. Neste enlace pódese atopar unha comparativa entre Docker Swarm e Kubernetes.

Conceptos de Docker Swarm

Dentro dun cluster Swarm de anfitrións Docker debemos ter claros unha serie de conceptos:

Nodos

Cada un dos "anfitrións" docker que están participando no Docker Swarm. Un nodo pode ser calquera tipo de servidor co demonio de docker instalado e vinculado a un cluster Swarm: dende unha máquina física, unha máquina virtual, ata un contedor docker!

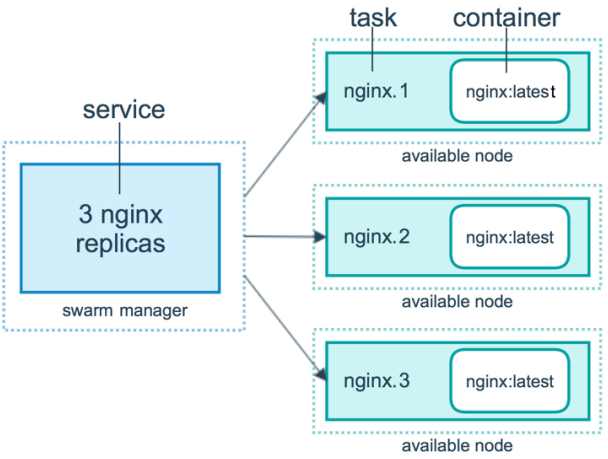
Un nodo no cluster Swarm pode ser de tipo **Manager**, **Worker** ou ambos a vez. O caso máis común en pequenos clusteres e ter un nodo Manager que actúa tamén como Worker, e os demais exclusivamente funcionando como Worker.

Para xestionar o cluster é preciso conectar contra un dos nodos Manager, ao que lle enviaremos comandos ben directamente co cliente de Docker, ou ben mediante a api. Os Manager se encargan de xestionar os recursos do cluster, as novas alta e baixas de nodos, e monitorizar o estado dos servicios swarm que se executan dentro del, e, os Workers de aloxar os contedores de estes servizos.

Servizos e Tarefas

Un servizo dentro do cluster swarm enténdese como a declaración dun estado desexado dos elementos que compoñen a nosa aplicación (o servizo web, o servizo de base de datos, etc.). Unha tarefa (*task*) é a unidade atómica de planificación, un espazo (*slot*) onde o planificador vai a acabar poñendo un container. A idea do swarm é ser un orquestrador e planificador de propósito xeral, polo que, tanto as *tasks* como os servicios (*services*) son as abstraccións coas que traballa.

A partir de ahí, os services definen o estado desexado dos diferentes elementos que compoñen a nosa aplicación, e as tasks acaban encarnándose en contedores que levan a cabo a tarefa de conformar este estado desexado. Un servizo descríbese utilizando a linguaxe que coñecemos de docker-compose, pero con algunhas características extra. A principal é a sección de **deploy** onde se especifican as características de despregue que se quere aplicar a ese servizo (réplicas, política de actualización, etc.).



Stacks

Unha pila (*stack*) é unha colección de servizos que representan unha aplicación nun determinado entorno, que comparten dependencias e poden ser orquestrados e escalados conxuntamente. Un ficheiro de stack é un ficheiro YAML co formato do docker-compose v3. Realmente é o equivalente ao *docker-compose.yaml* nun cluster Swarm; unha maneira conveniente de despregar servizos que están relacionados entre si para conformar unha aplicación.

Montar un cluster swarm

Para a creación dun primeiro cluster de nodos con Docker Swarm, imos a usar 2 máquinas virtuais correndo sobre VirtualBox. Docker ofrécenos a ferramenta Docker Machine, que permite instalar e xestionar, dende o noso equipo local, nodos Docker (servidores virtuais co docker-engine instalado) tanto en máquinas virtuais locais (HyperV, VirtualBox, VMWare Player) como en proveedores remotos (AWS, Azure, DigitalOcean, etc.).

Para crear o noso primeiro cluster Swarm, imos a empregar esta ferramenta para crear 2 novas máquinas virtuais correndo no noso VirtualBox, que van a formar o noso cluster Swarm. En primeiro lugar, teremos que instalar docker-machine:

```
$ base=https://github.com/docker/machine/releases/download/v0.16.0 &&
curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Podemos comprobar que docker-machine está instalado con "`sudo docker-machine version`".

Creamos as máquinas virtuais con docker-machine:

```
$ docker-machine create -d virtualbox host01
$ docker-machine create -d virtualbox host02
```

Con -d se especifica o driver a empregar, e host01 e host02 serán os nomes das vms xeneradas. Este comando descarga unha imaxe de VirtualBox dunha distribución moi lixeira de Linux (*boot2docker*) co motor de Docker instalado, e crea a máquina virtual con el.

Con "docker-machine ls" podemos listas as máquinas creadas e ver as súas direccións IP (tamén podemos parar e iniciar as máquinas con "docker-machine stop" e "docker-machine start", e borralas con "docker-machine rm"):

```
$ docker-machine ls
NAME     ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
host01   -        virtualbox   Running   tcp://192.168.99.100:2376           v18.09.1
host02   -        virtualbox   Running   tcp://192.168.99.101:2376           v18.09.1
```

Iniciamos o swarm nunha das máquinas (por exemplo na vbox01):

```
$ docker-machine ssh host01 "docker swarm init --advertise-addr 192.168.99.100"
Swarm initialized: current node (7v4sgaqpffckpopqj6gu107vw) is now a manager.

To add a worker to this swarm, run the following command:
    docker swarm join --token SWMTKN-1-2t9349e0vtu2s43q35vmqdj9o7g84r4up3m8997d9p3okdy0wz-6qq2kzefqu2qn64sf0le5p8f0
192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Como podemos ver na resposta, xa está inicializado o cluster swarm e indícasenos como podemos engadir novos *workers* e *managers*, usando un token que se xenera para cada cluster e a IP que usamos para iniciar swarm, que se vai a usar para conectarse cos demais nodos.

Seguindo as instrucións, unimos a outra máquina ao swarm:

```
$ docker-machine ssh host02 "docker swarm join --token SWMTKN-1-2t9349e0vtu2s43q35vmqdj9o7g84r4up3m8997d9p3okdy0wz-
6qq2kzefqu2qn64sf0le5p8f0 192.168.99.100:2377"
This node joined a swarm as a worker.
```

Xa temos un cluster swarm creado con dúas máquinas. Executando o comando "docker node ls" no manager podemos ver os seus nodos:

```
$ docker-machine ssh host01 "docker node ls"
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
7v4sgaqpffckpopqj6gu107vw *   host01              Ready               Active              Leader              18.09.1
wuowqtvufhsinde639gkb10p2     host02              Ready               Active                                  18.09.1
```

Agora podemos agregar máis máquinas ao cluster, ou lanzar sobre el unha aplicación.

Despregar unha aplicación no cluster

En primeiro lugar, temos que configurar o noso cliente Docker para que fale co host "host01", que é o manager do cluster. O comando "docker-machine env" indícanos como facelo:

```
 $ docker-machine env host01
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="-----/.docker/machine/machines/host01"
export DOCKER_MACHINE_NAME="host01"
# Run this command to configure your shell:
# eval $(docker-machine env host01)
```

Así que executamos no host o comando:

```
$ eval $(docker-machine env host01)
```

Podemos comprobar con "docker-machine ls" que agora o host "host01" está marcado como activo:

```
$ docker-machine ls
NAME     ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
host01   *        virtualbox   Running   tcp://192.168.99.100:2376           v18.09.1
host02   -        virtualbox   Running   tcp://192.168.99.101:2376           v18.09.1
```

Basicamente todas as configuracións relativas ao modo swarm están concentradas, dentro de cada servizo, no apartado **deploy** do ficheiro "docker-compose.yml", de tal maneira que o único comando que interpreta estas opcións é "docker stack deploy", mentres que "docker-compose" simplemente se salta esta sección. A continuación podemos ver unha modificación do ficheiro "docker-compose" de wordpress para despregar o servizo nun cluster:

```
version: '3.1'

services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    deploy:
      replicas: 2
      restart_policy:
         condition: on-failure
```

```
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: exampledb
      MYSQL_USER: exampleuser
      MYSQL_PASSWORD: examplepass
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    deploy:
      placement:
         constraints: [node.role == manager]
```

As opcións máis básicas que se inclúen dentro da sección "`deploy`" son:

- `replicas`: Permite establecer o número de contedores que se deben crear para o servizo. Se se establece a opción "`mode`" co valor "`global`" o cluster creará un contedor en cada nodo.
- `placement`: Permite establecer restricións (constraints) ou preferencias na creación dos contedores. Por exemplo, neste caso úsanse para que o contedor da base de datos se cree só no nodo manager.
- `restart_policy`: Establece se se deben reiniciar os contedores en caso de que rematen. Permite tamén establecer un retardo para reincialos, ou un número máximo de intentos.

Usamos o comando "`docker stack deploy`" para despregar a aplicación:

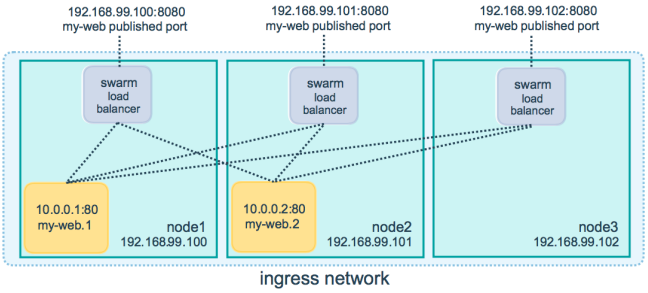
```
$ docker stack deploy --compose-file docker-compose.yml wordpress
Ignoring unsupported options: restart

Creating network wordpress_default
Creating service wordpress_wordpress
Creating service wordpress_db
```

Xa temos a nosa aplicación funcionando sobre o cluster. Con "`docker stack ps`" podemos ver os contedores creados, o seu estado e o nodo sobre o que se están executando:

```
$ docker stack ps wordpress
ID                  NAME                  IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
x05d5za8radg        wordpress_db.1        mysql:5.7           host01              Running             Running 4 minutes
ago
knvvwd9mo1eh        wordpress_wordpress.1   wordpress:latest    host01              Running             Running 4 minutes
ago
t1z4qhqg39s4        wordpress_wordpress.2   wordpress:latest    host02              Running             Running 4 minutes
ago
```

Neste momento xa temos o noso servizo funcionando, e accesible no porto publicado no ficheiro "`docker-compose`" en todos os hosts do cluster. É dicir, poderemos acceder ao wordpress no porto 8080 tanto do host01 como do host02. Isto é así porque Docker Swarm monta unha rede en malla entre todos os nodos do cluster para poder acceder dende calquera deles a calquera contedor dos que están executando o servizo, como se pode ver no esquema:



Con "`docker stack services`" podemos ver o funcionamento dos servizos do cluster, cos portos conectados, imaxes usadas e número de réplicas:

```
$ docker stack services wordpress
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
4kbkgvxcvhgq        wordpress_db        replicated          1/1                 mysql:5.7
j8rpsdtq58mc        wordpress_wordpress   replicated          2/2                 wordpress:latest    *:8080->80/tcp
```

14. Recoñementos

O material recollido neste libro está baseado no material dos cursos:

- "Tecnoloxía de contedores con Dockers", creado por Francisco Maseda Muiño e Javier Gómez Rodríguez.
- "Escenarios prácticos de seguridade e alta dispoñibilidade", creado por Manuel González Regal.