Docker

Obxectivos

Obxectivos

- Ver os orixes da técnica de contedores de software, dende unha perspectiva histórica.
- Explorar a plataforma que Docker ofrece para la xestión de contedores.
- Familiarizarse cos comandos básicos de Docker para administrar un contedor ao longo do seu ciclo de vida.
- Crear un contedor e xestionalo.

Orixes dos contedores de software

Xaulas chroot (1979)

No desenrolo do sistema Unix V7, agregouse unha nova chamada de sistema (chroot) que permitía cambiar o directorio raíz dun proceso, e os seus descendentes, a unha nova localización do sistema de ficheiros.

Este avance foi o comezo do illamento a nivel de proceso, posibilitando controlar o acceso aos ficheiros por proceso.

FreeBSD Jails (2000), Solaris Zones (2004)

Duas décadas despois, un proveedor de hosting sacou un servizo sobre xaulas BSD para lograr unha separación de recursos clara entre os seus servizos e os dos seus clientes, e deste xeito mellorar a seguridade e facilitar a administración dos mesmos.

As xaulas de FreeBSD permiten a un administrador, particionar un sistema FreeBSD en varios sistemas BSD independentes e con menos recursos, illados, e coa capacidade de agregar unha IP a cada un deles.

Oracle agregou unha característica similar a Solaris, que combina o control de recursos que pode consumir un proceso, ca separación en zonas dos procesos que se executan na máquina.

OpenVZ (2005)

Un ano máis tarde, a compañía Virtuozzo creou un novo sistema de virtualización basado en contedores, sobre o kernel de Linux, que permitía crear multiples contedores illados e seguros, nunha mesma máquina, como se dun servidor privado virtual se tratara.

O código de OpenVZ, aínda que está aberto, nunca formou parte da distribución oficial do kernel de Linux e se incorpora como unha serie de parches extra que se aplican sobre o código do kernel.

LXC (2008) e Docker (2013)

A primeira ferramenta de contedores de software que encontrou ampla acollida dentro da comunidade de Linux foi LXC, principalmente polas tecnoloxías que aglutinaba e que foron introducidas dentro do kernel de linux (**Namespaces e cgroups**)

O seu obxectivo e dispor dun contedor como un entorno o máis similar posible a unha máquina virtual, pero sen o sobrecoste de emular hardware. A xestión de este contedor se fai mediante unha serie de ficheiros de configuración, donde o usuario ten que encargarse de descargar os ficheiros do definir os namespaces onde quere conectar ao container, o que o convirte en unha ferramenta moi pouco amigable.

A compañía dotCloud empregaba LXC para dar o servizo de PaaS que ofrecía aos seus clientes, é, buscando unha forma de convertirlo en algo máis sinxelo de empregar, así como agregarlle novas funcionalidades para a xestión das aplicacións dos seus clientes, creou o proxecto que sería o precursor de Docker.

Docker se libera como proxecto de código aberto el 13 de marzo de 2013.

Saber máis

A brief history of containers: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Confining the omnipotent root: http://phk.freebsd.dk/pubs/sane2000-jail.pdf

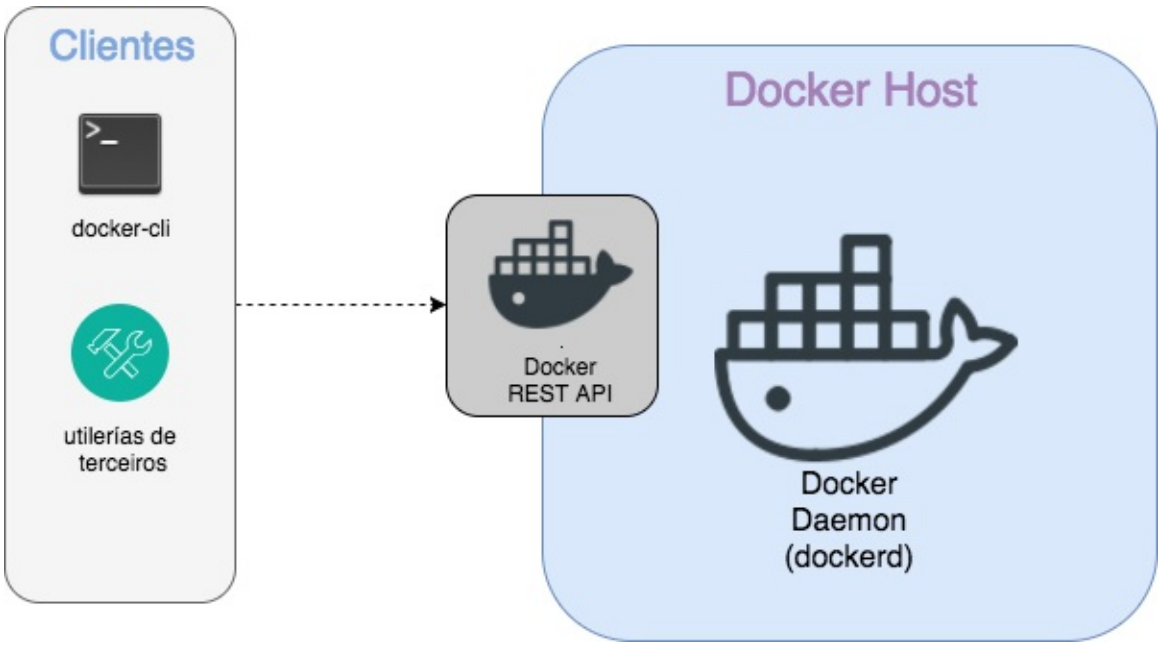
Características de Docker

Trátase dunha plataforma de containerización que:

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

O ecosistema do Docker

O Docker está baseado nunha serie de artefactos:



Como podemos ver, existen tres compoñentes básicas:

- O docker-cli: intérprete de comandos que permite interactuar con todo o ecosistema do Docker.
- Unha api REST: que permite dar resposta ós clientes: tanto o docker-cli como calqueira outra libraría ou cliente de terceiros. A API está ben documentada.
- O dockerd: un daemon que corre no host (pode sé-la nosa máquina) e que xestiona tódolos contedores, volumes e imaxes do host.

Estas tres compoñentes (docker-cli, API-REST e o dockerd) forman o **docker engine**.

Polo tanto, nós imos instalar o docker-engine nunha máquina e interactuar con él dende o propio docker-cli de esa máquina. Pero nada nos impide, coa configuración axeitada, poder interactuar dende o noso docker-cli con docker-daemons doutros hosts.

O interior do DockerD

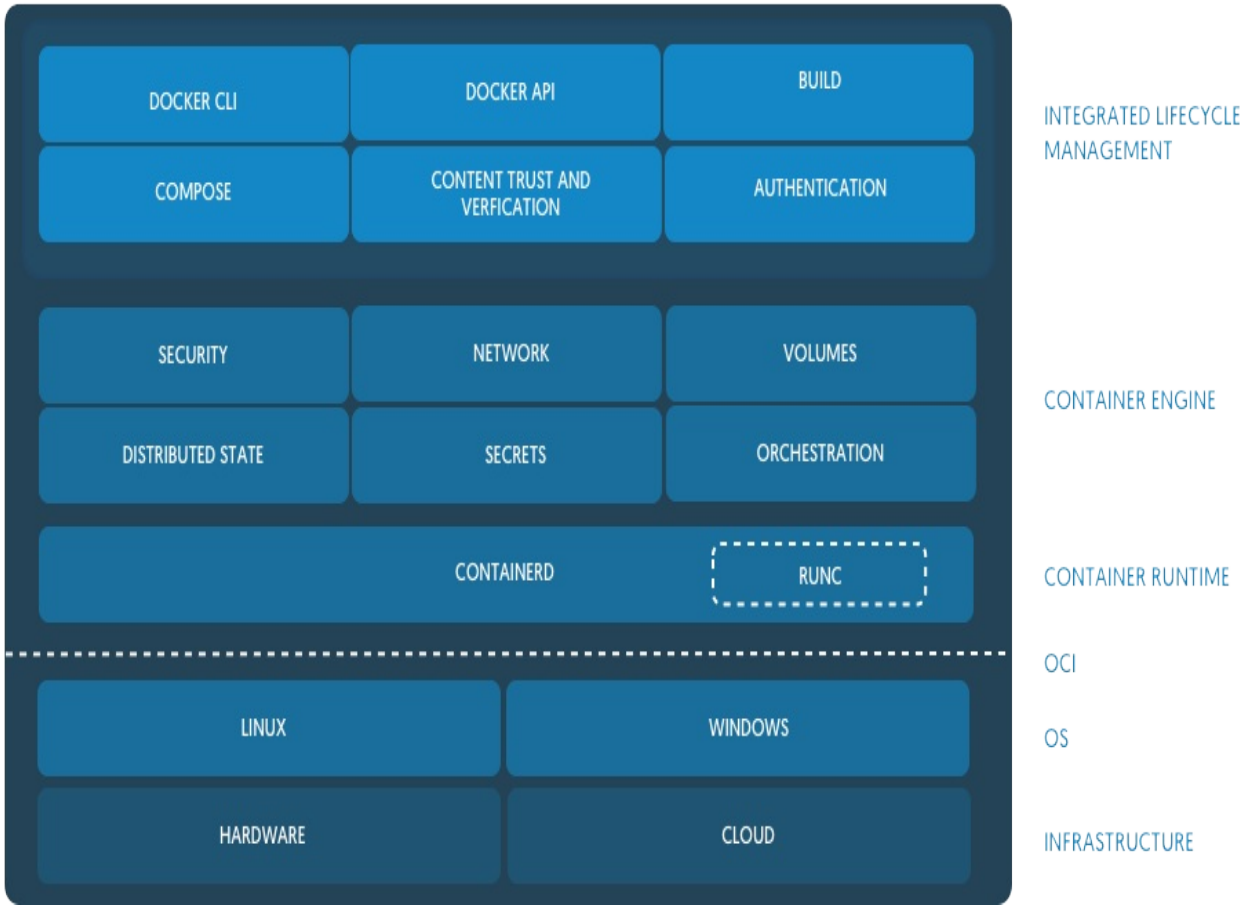
 A estrutura interna do DockerD sufriu unha profunda transformación a mediados do 2016. Máis información aquí.

O proxecto Docker, nos últimos dous anos, fixo importantes cambios para facilita-la súa adopción por parte de empresas e organismos:

- A doazón á comunidade do seu motor a baixo nivel (o containerd) responsable da xestión dos contedores. Constitúe agora o proxecto container-D.

- Creación do proxecto Moby que permite investigar e realizar probas coas funcionalidades básicas dos contedores sen recorrer a Docker.
- Colaboración estreita co organismo de recente creación, a OCI (Open Container Initiative) para estandariza-los contedores e imaxes.
- O emprego da libraría runc como endpoint para comunicarse co sistema operativo do host.

Esto implica que, a baixo nivel, o demonio de docker "tira" de varios proxectos para poder face-lo seu traballo.



Imaxe cortesía do bloque de docker.

O noso primeiro contedor: Instalando docker

Antes de comezar a traballar co ecosistema de Docker, temos que instalalo.

Tipos de instalación

Existen duas versións de Docker, unha de comunidade (Community Edition) e unha para empresas (Enterprise Edition)

Dentro da versión de comunidade, que sería a de empregar neste curso, temos varias opcións:

- A: Instalación de Docker no noso laptop
- B: Creación dunha máquina virtual (servidor linux) e instalar o Docker dentro dela.

A. Instalación de Docker no noso laptop

A plataforma de Docker permite a súa instalación nos principais sistemas:

- Windows
- Mac
- Distros de linux: aplícase o do punto B

B. Instalación de Docker nun servidor linux

Segundo a distro da nosa elección, temos distintas posibilidades.

Actividade

Realice unha instalación de Docker na súa plataforma de elección. Vai a necesitala para o resto do curso

Consello: para testear a instalación, pode empregar o seguinte comando:

```
# docker run --rm hello-world
```

O noso primeiro Contedor: ola mundo!

O traballo

Precisamos levantar un proceso que, empregando a comando de shell echo, saque unha mensaxe por pantalla dicindo: Ola Mundo!

Obviamente, queremos facelo dentro dun contedor, e que monte unha distro Debian.

I. A forma tediosa e lenta de facelo...

Tal e como viramos na práctica da lección anterior (*práctica guiada: construindo o noso contedor*) poderíamos seguir eses pasos para face-lo traballo:

- Descargar un sistema de ficheiros de Debian
- Montar un contedor con unshare, limitando os namespaces..
- ....

II. A forma Docker

Con docker, básicamente, temos que segui-los mesmos pasos da forma anterior, pero, afortunadamente, todo iso está automatizado dentro das súas utilidades.

Abóndanos con facer:

```
# docker run -i debian echo Ola Mundo!
```

Probablemente, antes da saida, vexamos a seguinte mensaxe por pantalla:

```
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
```

Docker, está a informar de que precisa descargar a imaxe de debian antes de lanza-lo contedor.

Xestión de contedores

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

- A través dunha api
- Mediante a súa liña de comandos

Neste curso, imos a empregar a liña de comandos.

O intérprete de comandos de Docker é o comando docker.

Comandos básicos

Listaxe dos contedores do sistema

Para saber os contedores existentes nunha máquina, imos a empregar o comando ps

```
# docker ps
```

O que obtemos é:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
b3b0f3dff0cd        mongo               "docker-entrypoint..."   7 seconds ago       Up 6 seconds        27017/tcp           mongo
```

Como podemos comprobar:

- Docker asigna un uuid ós contedores.
- Infórmanos sobre a imaxe que monta (máis sobre imaxes no seguinte tema).
- Fálanos do 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 ó contedor en caso de que non llo poñamos nós.

Creando e arrincando contedores

O comando básico de creación de contedores e create.

Compre enviarlle un elemento obligatorio:

1. Imaxe de creación(trátase dunha planiña a partir da que se montará o contedor).

Deste xeito, o seguinte comando:

```
# docker create prefapp/debian-formacion
```

Voltaranos unha cadea en hexadecimal que será o identificador único do contedor creado.

Nembargantes, se facemos **docker ps** non o veremos na táboa de contedores.

A razón é que o contedor creado estará parado. Para arrincalo, compre executar o comando start.

```
# docker start <uuid do contedor>
```

Con isto, teremos o novo contedor funcionando, e se facemos un **docker ps** poderémolo ver na táboa de contedores funcionando.

O normal é facer estos dous pasos nun, mediante o comando run. Este

comando, crea e arrinca o contedor.

Detendo o contedor

Para deter un contedor que está a funcionar, abonda con empregar o comando stop.

```
# docker stop
```

Unha vez executado, o contedor está detido, de tal xeito que non aparecerá xa na táboa de **docker ps**. Compre empregar **docker ps -a** para que o listado inclúa os contedores detidos.

Borrando o contedor

O comando docker rm elimina o contedor.

Comando run - avanzado

Unha vez comprendido o ciclo básico de **crear-correr-deter-borrar** contedores, imos revisita-lo comando run para vez algunhas das súas opcións.

Executar un comando instantáneo nun contedor

Imaxinemos que queremos sabe-la versión da nosa debian, para iso, teríamos que facer un cat do ficheiro **/etc/issue**.

Imos crear un contedor que faga este traballo, e despois que desapareza.

```
# docker run --rm prefapp/debian-formacion cat /etc/issue
```

Isto nos sacará por pantalla, a seguinte información:

```
Debian GNU/Linux 8 \n \l
```

O que acaba de pasar é o seguinte:

1. Docker crea e arrinca (**run**) un novo contedor.
2. O sistema vai a borrar o container ó final da súa execución (**--rm**).
3. Este contedor vai a estar baseado na imaxe **prefapp/debian-**

formacion.

4. O comando a executar no contedor é **cat /etc/issue**.

Notése que o /etc/issue non é o da nosa máquina senón o da imaxe que monta o contedor!

Neste exemplo pódese ver o lixeiros que son os contedores: os creamos e destruimos en cuestión de décimas de segundos e os empregamos para executar tareas triviais (como facer un cat)

Executar un comando de xeito interactivo

Imaxinemos que queremos executar unha sesión dentro da nosa imaxe debian e poder executar comandos dentro dela.

Para iso temos que crear un contedor (que arrinque un intérprete de comandos), e interactuar con él.

```
# docker run --rm -ti prefapp/debian-formacion /bin/bash
```

Se executamos este comando, a noso prompt cambiará a un cadea hexadecimal (o uuid do propio contedor).

Agora poderíamos interactuar co contedor, e cando rematemos, basta escribir **exit** ou Ctrl+D para sair da shell e, ó rematar o programa de lanzamento do contedor, o propio contedor morre e recupera o control a shell da nosa máquina.

Crear e correr un contedor demonizado

Cando queremos correr un contedor que da un servizo, o normal é executalo como un daemon.

Imos crear un contedor que funcione como si de un servidor Debian se tratase. Este contedor vai correr demonizado (independiente da nosa sesión na máquina). Además, ímoslle poñer un nome para poder xestionalo dun xeito máis sinxelo.

```
# docker run --name contedor-formacion -d prefapp/debian-formacion tail -f /dev/null
```

Neste caso, decímoslle a docker

- Crea e arrinca un contedor (**run**)
- Ponlle o nome contedor-formacion (--**name**)
- Queremos que corra en segundo plano, como daemon (**-d**)
- Baseámolo na imaxe prefapp/debian-formacion
- O comando a executar é tail -f /dev/null

 O emprego do tail -f /dev/null empregase para que, ó ser o proceso iniciador do contedor, perviva indefinidamente, ata que se remate explícitamente mediante docker stop, rematando así o contedor enteiro.

Se executamos o comando, veremos que seguimos na shell da nosa máquina. Ó facer **docker ps**, veremos que o noso container está creado

e co nome que establecemos.

```
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
21295036cca5        prefapp/debian-formacion   "tail -f /dev/null"   2 seconds ago       Up 1 second                             contedor-formacion
```

Unha pregunta que nos podemos facer agora é: cómo entro nese contedor agora?

A solución é **docker exec**.

Executando un comando dentro dun contedor: o docker-exec

Na sección anterior vimos que podiamos crear un contedor como un daemon que correse na nosa máquina.

Preguntábamos cómo podíamos "acceder" a ese contedor.

A solución que nos ofrece docker é o comando exec: a idea é crear un proceso que se "inserte" dentro do contedor e se execute dentro do mesmo.

Para poder acadar isto, Docker emprega a chamada ó sistema nsenter que lle permite que o proceso "entre" nos namespaces doutro proceso, neste caso, os do contedor no que queremos introducirnos.

Se lembramos o exemplo anterior, tiñamos un contedor correndo a nosa imaxe de debian co nome "contedor-formacion".

Para poder correr un proceso noso dentro, cun intérprete de comandos, facemos:

```
# docker exec -ti contedor-formacion /bin/bash
```

O que estamos a dicirlle a Docker é:

1. Queremos que corras un proceso dentro dun contedor (**exec**)
2. Ese proceso ten que correrse en formato interactivo (**-ti**)
3. O contedor se identifica co nome contedor-formacion
4. O proceso a executar é un /bin/bash

Xestión da rede

Por defecto os contedores Docker poden acceder ó exterior (teñen conectividade, dependendo sempre da máquina anfitriona)

Nembargantes, o contedor está aillado con respecto a entrada de datos. É decir, por defecto, e coa excepción do comando exec, *o contedor constitúe unha unidad aillada á que non se pode acceder*.

Por suposto, compre poder acceder ós 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**.

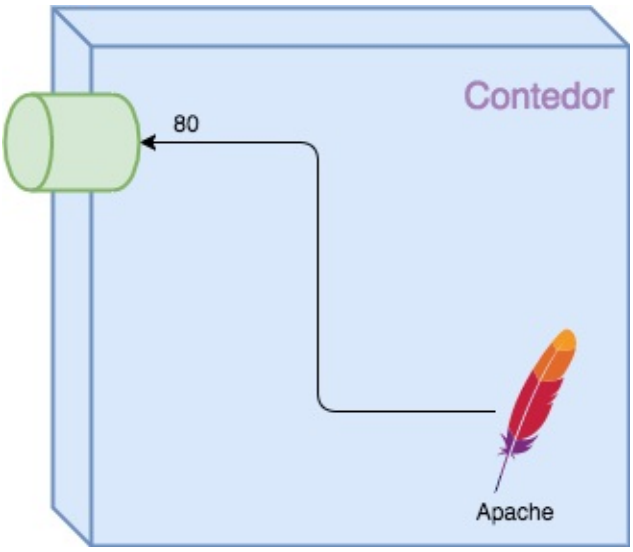
Nesta sección imos aprender cómo resolve Docker o problema de dar conectividade ós contedores.

Portos do contedor

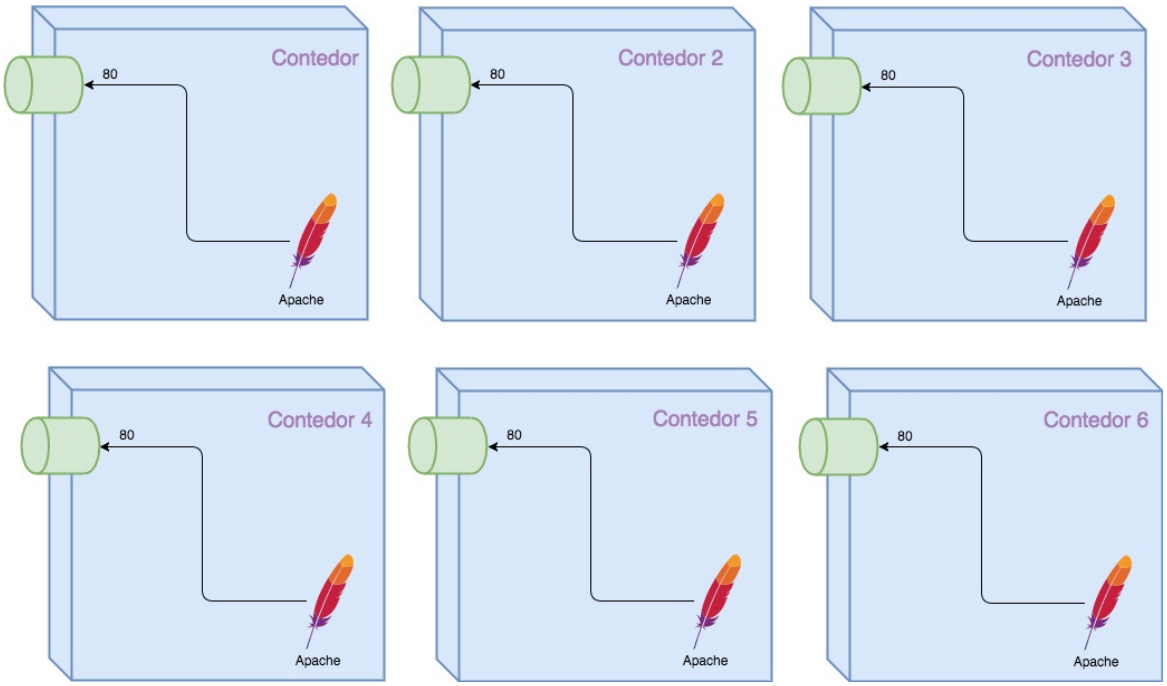
Un contedor está dentro do seu namespace de rede polo que pode establecer regras específicas sen afectar ó resto do sistema.

Deste xeito, e sempre dende o punto de vista do contedor, podemos establecer as regras que queiramos e conectar o que desexemos ós 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 ó porto 80:



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

⚠️ Por suposto, se dentro dun mesmo contedor poñemos dous servizos á escoita no mesmo porto, produciráse un erro de rede.

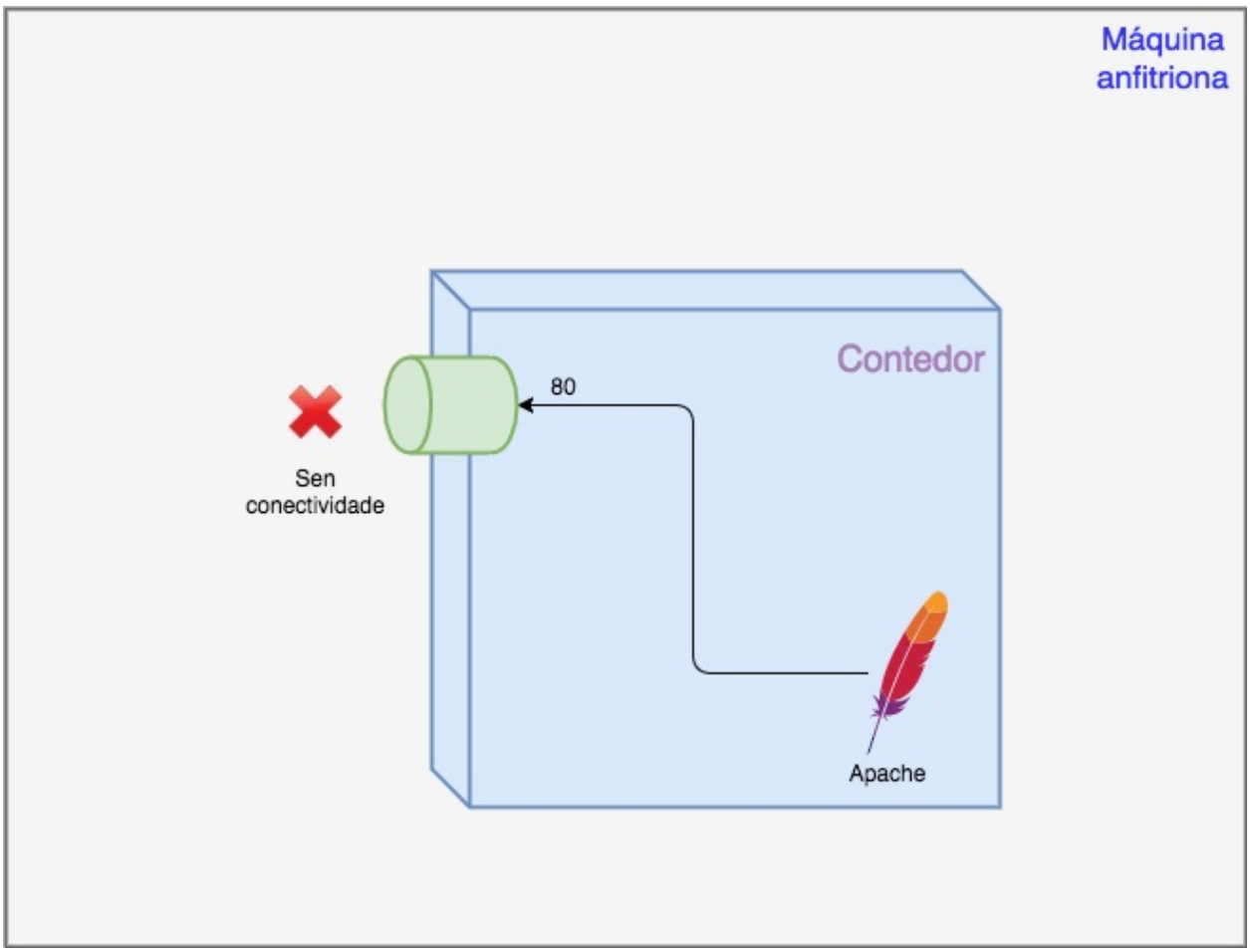
O problema é que, pese a ter esta liberdade dentro do contedor, aínda precisamos conecta-lo extremo do contedor cun porto da máquina anfitriona para que as conexións externas poidan chega-lo noso contedor.

Por sorte, ese traballo vaínolo facilitar moito Docker.

Conectar portos de docker ó exterior

Tal e como viramos na sección anterior, dentro dun contedor dispoñemos de todo o stack de rede para facer o que queiramos.

Nembargantes, con iso non é suficiente para que o se poida chegar ó noso contedor dende o mundo exterior. Así:



Como vemos, o noso contedor está aillado do mundo exterior, por moito que o apache esté presente e conectado ó porto 80 dentro do contedor.

Imos probar a arrincar unha imaxe que temos preparada para este curso:

- Está baseada en debian:8
- Ten instalado o apache2
- O proceso de arranque consiste en poñer a escoitar o apache no porto 80

```
# docker run -d --name apache-probas prefapp/apache-formacion
```

Se facemos agora un **docker ps**, verémo-lo noso contedor funcionado. O problema: non podemos chegar de ningunha forma ó servizo de apache que corre escoitando no porto 80, porque o noso contedor non ten conectividade co exterior.

O único xeito para poder conectarse ó apache, é entrar no contedor:

```
# docker exec -ti apache-probas /bin/bash
```

E dende dentro do mesmo, xa poderíamos facer, por exemplo un curl ó porto 80:

```
root@378cc9f54153:/# curl localhost:80
```

E obteríamos unha resposta axeitada do apache.

Docker permítenos conectar os portos do contedor con portos do anfitrión de tal xeito que este último sexa accesible dende fora.

Imos borra-lo noso container:

```
# docker rm -f -v apache-probas
```

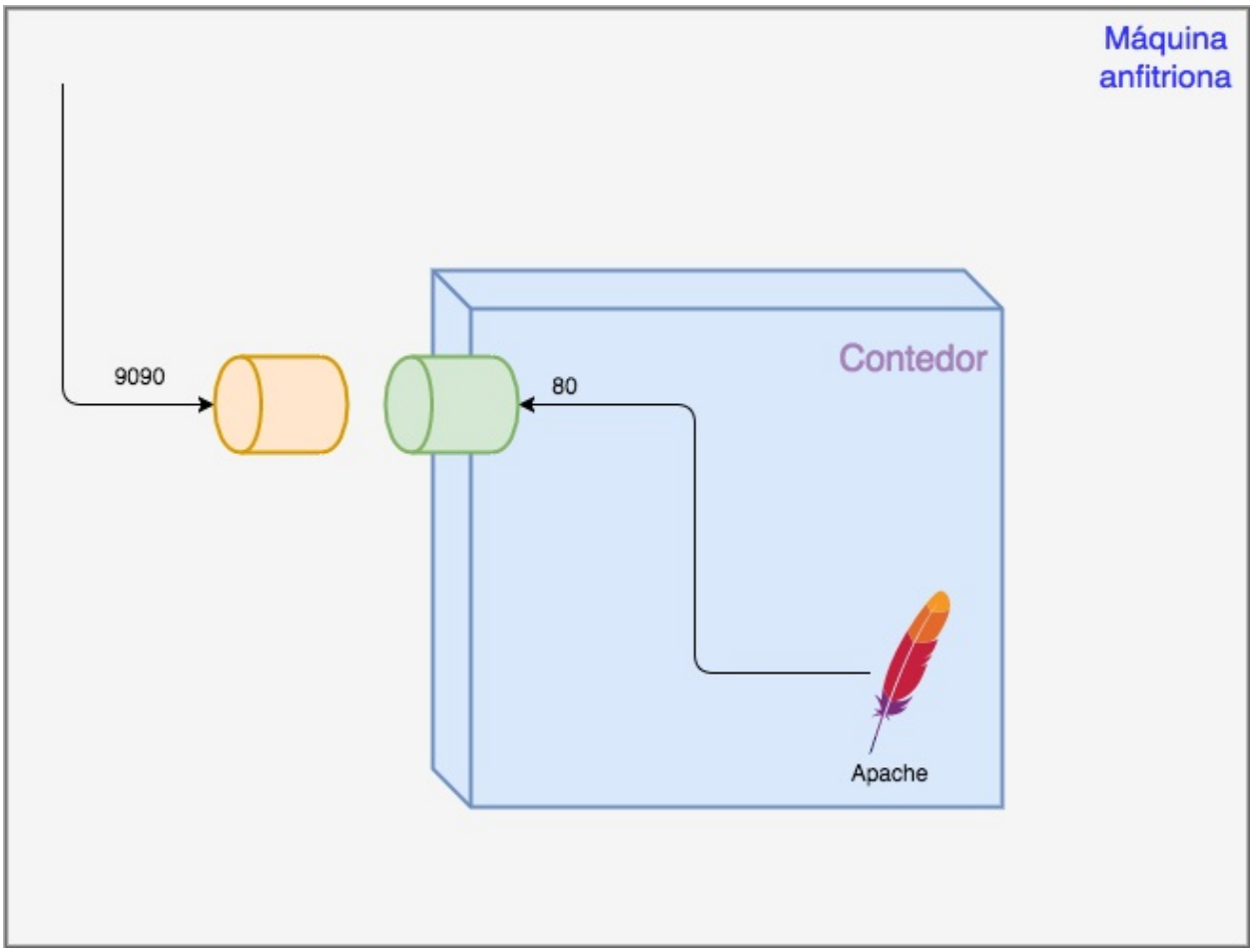
E redesplegalo cunha nova opción:

```
# docker run -d -p 9090:80 --name apache-probas prefapp/apache-formacion
```

Como vemos, a novidade aquí é o **-p 9090:80**:

- Estalle a dicir ó Docker que precisa conectar un porto do anfitrión cun do container.
- O porto 9090 corresponde cun porto do anfitrión.
- O porto 80 é o de dentro do contedor (onde escoita o noso apache)

Nun diagrama:



Agora o porto 80 do noso contedor está conectado ó porto 9090 da máquina anfitriona. Podemos, por tanto, chegar ó apache que corre dentro sen problema dende o exterior:

```
# curl localhost:9090
```

 O porto do anfitrión é totalmente controlable mediante reglas de iptables ou de firewall para determinar a súa accesibilidade dende o exterior.

 Aínda que teñamos liberdade nos portos dentro do contedor, os portos do anfitrión son un recurso limitado e de uso alternativo. Isto é, non se pode asignar o mesmo porto a varios portos de contedor. Isto supón un problema de xestión a resolver por parte das ferramentas de orquestación, obxecto do tema 6 do presente curso.

Paso de argumentos ó contedor

A estas alturas da lección, xa sabemos que o contedor é unha unidade illada dende o punto de vista dos recursos tradicionalmente compartidos nunha máquina: usuarios, pids, mounts, rede...

Este isolamento é unha das claves da creación de contedores. Nembargantes, a veces compre poder controlar dalgún xeito os procesos que corren dentro do contedor. Isto, é: controlar os programas mediante o paso de parámetros.

Existen duas formas principais de paso de parámetros a un contedor:

- Mediante os argumentos que lle chegan ó programa que lanzamos mediante docker-run ou docker-create.
- Asignando ou mudando os valores que reciben as variables de entorno dentro do contedor.

Imos centrarnos na segunda das posibilidades nesta sección.

Control das variables de entorno dun contedor

Antes de correr un contedor, Docker permite establecer cal será o valor do seu entorno mediante o paso de binomios clave=valor.

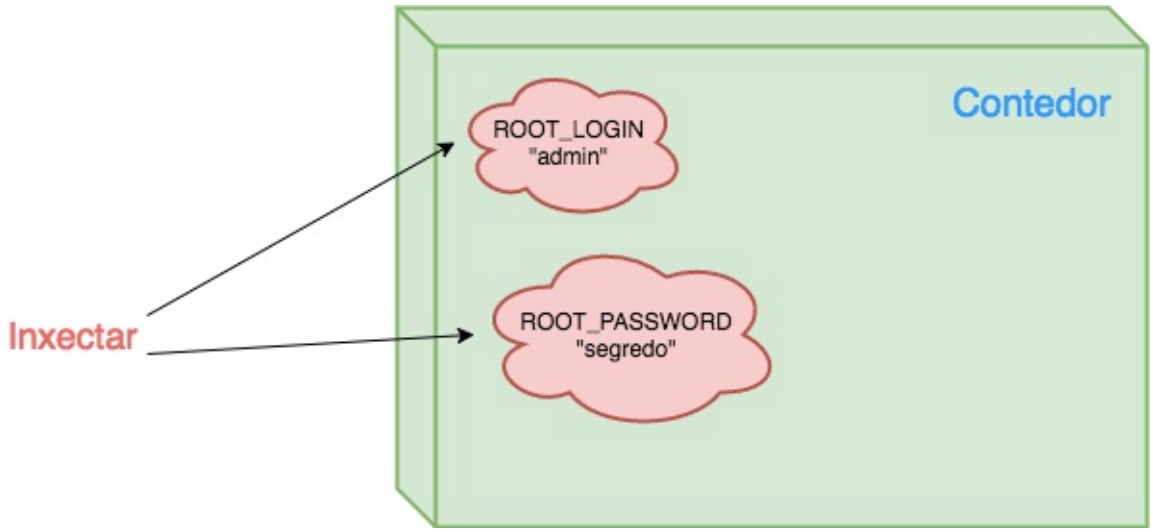
Deste xeito, se temos unha aplicación que precisa un login/password de administrador e que os recolle de variables de entorno, por exemplo (ROOT_LOGIN, ROOT_PASSWORD), poderíamos facer o seguinte:

```
# docker run -d -e ROOT_LOGIN=admin -e ROOT_PASSWORD=segredo imaxe-de-app-con-admin
```

Neste exemplo:

- Créase e lanzase un container en modo daemon (**run -d**)
- Cunha imaxe ficticia (*imaxe-de-app-con-admin*)
- Establécese que se cree ou mude (**-e**) o valor dunha variable de entorno (ROOT_LOGIN con valor admin)
- O mesmo (**-e**) para a variable ROOT_PASSWORD (valor segredo)

Docker, vai inxectar estas variables dentro do contedor antes de arrincalo de xeito que, se o programa está preparado para facelo, pode recolle-la súa configuración do ENV.



Obviamente, se queremos mudar o valor das variables de entorno dun contedor en funcionamento, compre reinicialo o recrealo.

Xestión de volumes

Sabemos que os contedores son efémeros, isto é, unha vez rematado o contedor (cando o proceso que o arrincou finalize) tódolos datos que teñamos no contedor desaparecen.

Para velo, podemos facer a seguinte práctica:

Actividade

Arrinquemos un container con run da nosa debian de formación:

```
docker run --rm -ti prefapp/debian-formacion bash
```

Agora, actualizamo-las fontes con **apt-get update.** De seguido instalamo-lo programa curl.

```
# apt-get update && apt-get install --yes curl
```

Comprobamos que funciona.

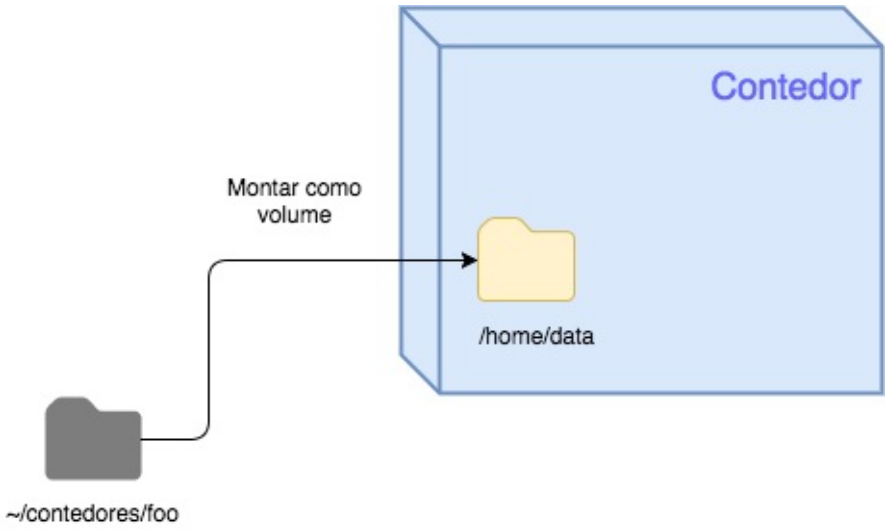
Salimos do container (a opción --rm provocará o seu inmediato borrado)

Se volvemos a lanzar un container co comando de enriba, temos curl? por qué?

Volumes: a conexión do contedor co sistema de ficheiros do anfitrión

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 indicarlle a Docker qué directorio do noso anfitrión queremos montar como volume e en qué ruta queremos montalo no noso contedor.

Nun exemplo:

```
docker run --rm -ti -v ~/meu_contedor:/var/datos prefapp/debian-formacion bash
```

Como podemos ver:

- Indicamoslle ó Docker que queremos un contedor interactivo que se autodestrúa (**run --rm -ti**)
- Imos corre-la imaxe prefapp/debian
- O comando de entrada é o bash
- Montamos un volume: **-v,** indicándolle ruta_anfitrion:ruta_contedor (**~/meu_contedor:/var/datos**)

Actividade

Probemos este comando. Lanzamos un contedor que cree un volume e, unha vez dentro do contedor, creamos tres ficheiros en /var/datos.

Pechamola-sesión do contedor e miramos no directorio do anfitrión, qué ten dentro?

Volvemos lanzar un contedor co mesmo comando, se vamos a /var/datos: están os ficheiros? Se están, borramos o /var/datos/a e o /var/datos/b.

Pechamos outra vez a sesión, miramos no directorio do anfitrión, qué ficheiros hai?

Portainer: a administración visual do Docker

O proxecto de portainer consiste na creación dunha servicio web que permite interactuar coa API de Docker dun xeito visual e intuitivo.

Para instala-lo, basta con facer:

```
docker run -d -p 9000:9000 --restart always -v /var/run/docker.sock:/var/run/docker.sock -v /opt/portainer:/data portainer/portainer
```

E xa teríamos o portainer correndo nun contedor, bastaría con abrir o navegador no porto elixido e veríamos a pantalla de inicio.

Fornecemos un usuario e un contrasinal, e xa teríamos un sistema para administrar visualmente o noso Docker.

Actividade

Probemos a instalar o portainer.

A partir de agora, ó longo do resto do curso aparecerá esta icona  en varios lugares. Cando sexa así, estamos a recomandar que se vexan os resultados no portainer.

Consideramos que a información que aporta facilita o proceso de aprendizaxe.