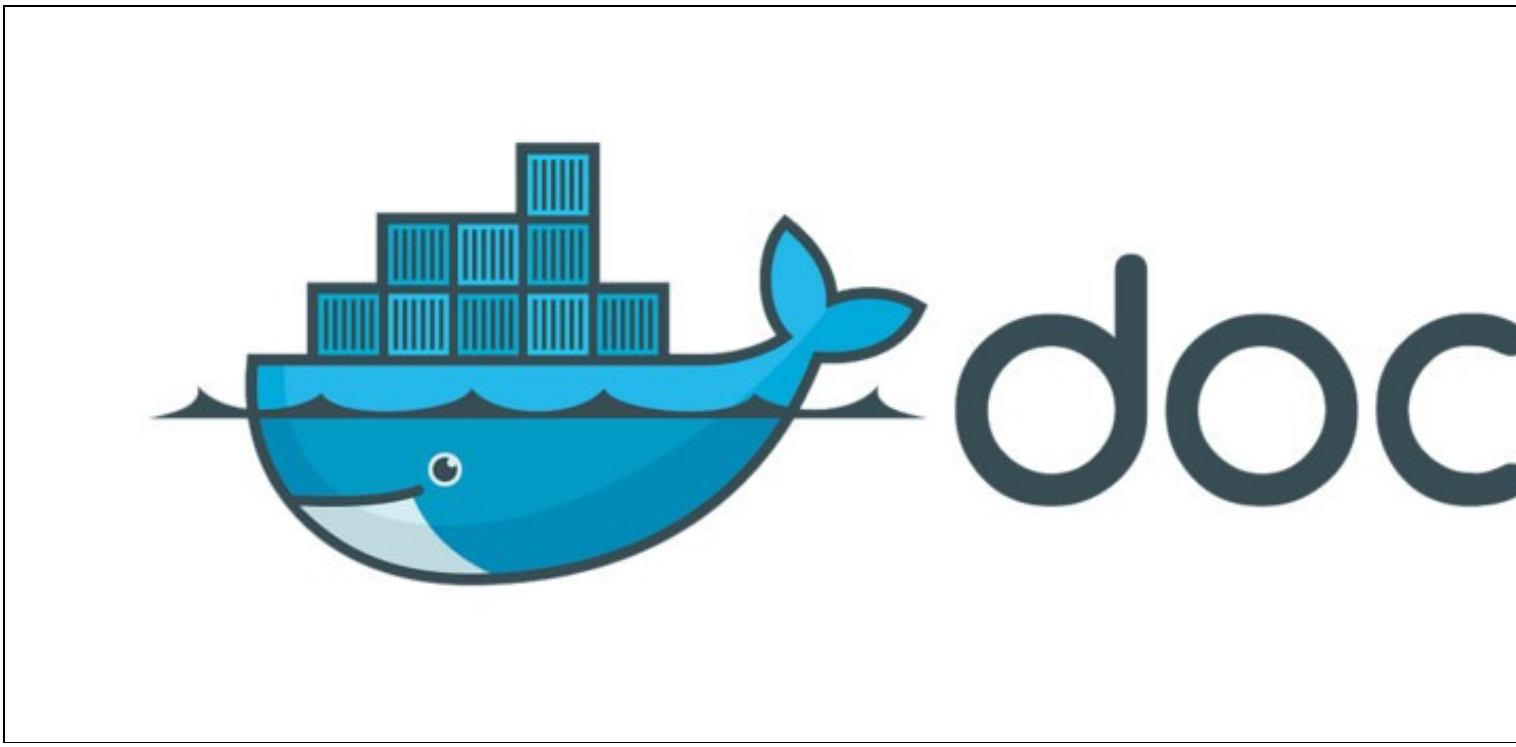
Docker

Segundo a Wikipedia *Docker é un proxecto de código aberto capaz de automatizar o despregamento de aplicacións dentro de contenedores de software, proporcionándonos así unha capa adicional de abstracción e automatización no nivel de virtualización do sistema operativo.?*

Sumario

- 1 Introdución
- 2 Instalación
  - 2.1 Permitir a execución de docker a usuarios non *root*
- 3 Xestión de contedores
  - 3.1 Comandos básicos
    - 3.1.1 Listaxe dos contedores do sistema
    - 3.1.2 Creación de contedores
    - 3.1.3 Detendo o contedor
    - 3.1.4 Borrando o contedor
  - 3.2 Comando *run* avanzado
    - 3.2.1 Executar un comando instantáneo nun contedor
    - 3.2.2 Executar un comando de xeito interactivo
    - 3.2.3 Crear e lanzar un contedor demonizado
    - 3.2.4 Executando un comando dentro dun contedor
  - 3.3 Xestión da rede
    - 3.3.1 Portos do contedor
    - 3.3.2 Conectar portos de docker ó exterior
  - 3.4 Paso de argumentos ó contedor
  - 3.5 Xestión de volumes
- 4 Xestión de imaxes
  - 4.1 Listado de imaxes
  - 4.2 Creación de imaxes propias
    - 4.2.1 Sintaxe dun Dockerfile
    - 4.2.2 Instruccións do Dockerfile
- 5 Orquestación de contedores
  - 5.1 Instalación do docker-compose
  - 5.2 Fundamentos do docker-compose
    - 5.2.1 Servizos
    - 5.2.2 Redes
    - 5.2.3 Volumes
  - 5.3 Ciclo de vida dunha aplicación de compose

Introdución



A partir desta definición podemos interpretar que docker virtualiza o ambiente que necesitamos para a nosa aplicación utilizando contedores sobre o kernel do sistema operativo, e debemos agregar que estes contedores teñen os seus procesos illados do host onde se montan, pero que significa todo iso?

Docker é unha plataforma para desenvolvedores e administradores de sistemas para desenvolver, desprega, e executar aplicacións con colectores. O uso de contedores de Linux para despregar as aplicacións é chamada containerización. Os contedores non son novos, mais o seu uso para despregar aplicacións de forma sinxela si que o é.

A containerization é cada vez máis popular porque os colectores son:

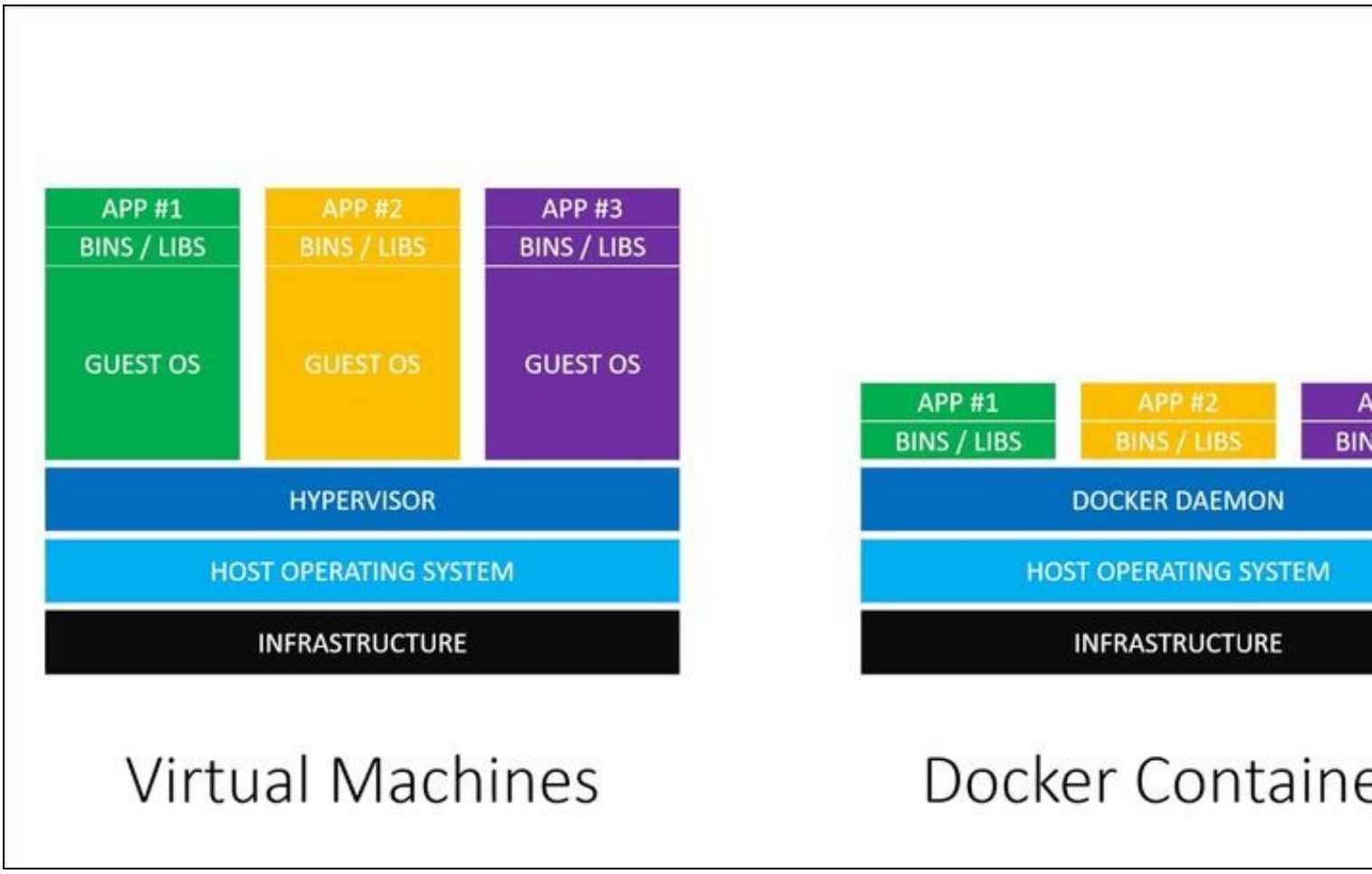
- *Flexibles*: A maioría de aplicacións complexas poden ser containeirizadas.
- *Lixeiros*: Os contedores teñen a vantaxe de que comparten o kernel co anfitrión.
- *Intercambiables"*: podes despregar actualiza e upgrades sobre a marcha.
- *Portátiles*: Podes construír localmente, despregar na nube, e executar en calquera sitio.
- *Escalables*: Podes aumentar e distribuír de xeito automático réplicas do contedor.
- *Apilables*: Podes apilar servizos verticalmente e sobre a marcha.

Lánzase un contedor executando unha imaxe. Unha imaxe é un paquete executábel que inclúe todo o necesario para executar unha aplicación: o código, a contorna de execución, librarías, variábeis de contorna, e ficheiros de configuración.

Un contedor é unha instancia en tempo de execución dunha imaxe, que a súa vez permanece na memoria namentres se executa (o que ven sendo, unha imaxe con estado, ou un proceso de usuario).

Para entendelo mellor imos ver unha comparación dos contedores coas máquinas virtuais tradicionais.

No caso da máquina virtual temos o sistema operativo HOST sobre o servidor, logo un Hypervisor e enriba outro sistema operativo e cada unha das máquinas virtuais ten librarías e por ultimo a nosa aplicación. Isto significa que temos varias capas que consomen recursos e non comparten nada. Ao contrario dos contedores onde non hai un Hipervisor senón un Docker Engine, non hai un sistema operativo completo correndo enriba do outro, hai librarías compartidas e enriba as nosas apps ou servizos illados uns doutros. Todo isto resúmese a unha mellor utilización de recursos, maior velocidade á hora de subir contedores illamento e separación de responsabilidades.



Todo isto dános unha serie de vantaxes á hora de traballar con sistemas que necesitan moitos recursos e sobre todo que necesitan escalar.

Para ter en conta!.

> Ao compartir o kernel do sistema operativo HOST os contendores podes ser vistos como menos flexibles xa que xeralmente só poden correr un sistema operativo que sexa o mesmo ou similar ao do HOST. Por exemplo poderíase correr Red Hat nun servidor Ubuntu pero non Windows nese mesmo servidor. Con todo estas limitacións non son tan importantes na maioría de casos.

Docker prové o necesario para automatizar o despregamento de aplicacións en contedores e está deseñado para proporcionar un ambiente lixeiro e rápido onde podemos executar o noso código, así mesmo provenos un fluxo eficiente para levar este código a ambientes desde o noso computador local a testing ou produción.

Instalación

Docker está dispoñible en dúas edicións:

- Community Edition (CE)
- Enterprise Edition (EE)

Docker Community Edition (CE) é ideal para desenvolvedores individuais e equipas pequenas que queiran iniciarse con Docker e experimentar con aplicacións containerizadas.

Docker Enterprise Edition (EE) está deseñado para o desenvolvemento empresarial e equipas que construen, desenvolven e executan aplicacións críticas a nivel de produción.

Para instalar docker en Debian 9 (stretch) ou Raspian necesitamos ter instalada previamente unha versión de 64 bits de Debian 9. Docker CE está soportado en arquitecturas x86_64 (ou amd64), armhf, e arm64.

O primeiro paso é desinstalar dodas as versións anteriores

```
apt remove docker docker-engine docker.io containerd runc
```

A continuación procedemos á instalación. Temos tres posibles opcións:

- Configurar os repositorios de debian e instalar desde alí para facilitar a instalación e a actualización. Esta é a opción máis axeitada excepto para Raspbian.
- Descargar o paquete DEB e instalalo manualmente, así coma as súas actualizacións.
- Instalar a partires de scripts. É a única forma de instalalo en Raspbian.

Nesta guía instalaremos a partires do repositorio. Previamente deberemos configurar 'apt' para empregar HTTPS:

```
 apt install apt-transport-https \
    ca-certificates curl gnupg2 \
    software-properties-common
```

A continuación engadimos a chave GPG e engadimos o repositorio ás listas de APT.

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

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

Por último instalamos a derradeira versión de docker

```
apt update
apt install docker-ce
```

Para probar se todo se instalou correctamente, probamos a descargar e executar unha imaxe de proba. Esta execúase, imprime información e remata a súa execución.

```
docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Permitir a execución de docker a usuarios non *root*

Por defecto docker so permite ao usuario root (ou mediante o comando sudo) traballar con contedores.

Para permitir aos demais usuarios so temos que engadilos ao grupo *docker* (debe existir previamente)

```
usermod -aG docker usuario
```

Para que os cambios estén aplicados o usuario deberá saír da súa sesión e volver a entrar.

Xestión de contedores

Comandos básicos

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

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

Nesta guía, imos a empregar a liña de comandos. O intérprete de comandos de Docker é o comando ***docker***.

Listaxe dos contedores do sistema

Para saber os contedores existentes que se están executando nunha máquina, imos a empregar o comando **ps** de Docker

```
#docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS                     NAMES
424258ec9b54        portainer/portainer    "/portainer"        9 months ago        Up 7 hours          0.0.0.0:9000->9000/tcp    compas
```

Da saída deste comando podemos comprobar:

- Docker asigna un uuid ós contedores.
- Infórmanos sobre a imaxe que monta.
- 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.

Creación de contedores

O comando básico de creación de contedores e **create**.

Compre enviarlle un elemento obligatorio: a imaxe de creación(trátase dunha plantilla a partir da cal se montará o contedor).

Deste xeito, o seguinte comando:

```
#docker create smartentry/debian
Unable to find image 'smartentry/debian:latest' locally
latest: Pulling from smartentry/debian
cd8eada9c7bb: Already exists
6f5536442030: Pull complete
Digest: sha256:6e5dcc023466dd77348c544e0d731c060144151632fa0d32e03106642c7b34c7
Status: Downloaded newer image for smartentry/debian:latest
4250dcbcbb3e98d631d1d8df546612dcecf5049d41af419e3e029f4ade36c2f8
```

Devolveranos unha cadea en hexadecimal que será o identificador único do contedor creado. Nembargantes, se facemos `docker ps` non o veremos na táboa de contedores activos. A razón é que o contedor creado estará parado. Para arrincalo, compre executar o comando **start**.

```
docker start <uuid do contedor ou nome>
```

Con isto, teremos o novo contedor funcionando, e se facemos un `docker ps` poderémolo ver na táboa de contedores activos. O normal é facer estos dous pasos nun, mediante o comando *run*. Estecomando, crea e arrinca o contedor.

```
docker run smartentry/debian
```

Detendo o contedor

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

```
#docker stop <uuid do contedor ou nome>
```

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.

```
#docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS                      PORTS
c31770124660        portainer/portainer    "/portainer -H unix:?"   25 hours ago        Up 25 hours                 0.0.0.0:9000->9000
b0318b313dbb        fce289e99eb9           "/hello"                 2 days ago          Exited (0) 2 days ago
```

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 revisitar o 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  smartentry/debian cat /etc/issue
smartentry> running main program(UID=0 GID=0 USER=root)
Debian GNU/Linux 9 \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 smartentry/debian.
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: creámolos e destruímolos en cuestión de décimas de segundos e empregámolos 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 smartentry/debian /bin/bash
smartentry> running main program(UID=0 GID=0 USER=root)
root@0bb85ada5756:/#
```

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 lanzar 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-proba -d smartentry/debian tail -f /dev/null
826b8a6a1354dd1ccef09b7426ba1f03dc5d7b30f8f8e2f7db3c0ef6f39095c1
```

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 smartentry/debian

- 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 creadoe co nome que establecemos.

```
#docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                    
826b8a6a1354        smartentry/debian      "/sbin/smartentry.sh?"   6 seconds ago       Up 5 seconds                                 
c31770124660        portainer/portainer    "/portainer -H unix:?"   26 hours ago        Up 25 hours         0.0.0.0:9000->9000/tcp   
```

Unha pregunta que nos podemos facer agora é: cómo entro nese contedor agora? A solución é `docker exec`.

Executando un comando dentro dun contedor

A solución que nos ofrece docker para "entrar dentro dun contedor" 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-proba". Para poder correr un proceso noso dentro, cun intérprete de comandos,facemos:

```
#docker exec -ti contedor-proba /bin/bash
826b8a6a1354dd1ccef09b7426ba1f03dc5d7b30f8f8e2f7db3c0ef6f39095c1
```

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-proba
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.

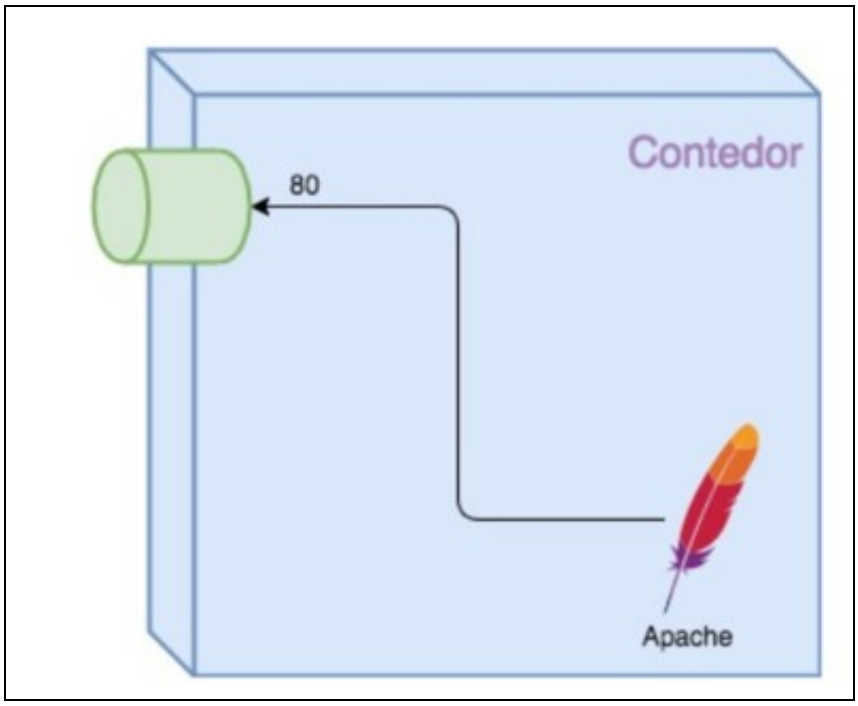
A regra básica aquí é: salvo que se estableza o contrario, todo está pechado ó mundo exterior.

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

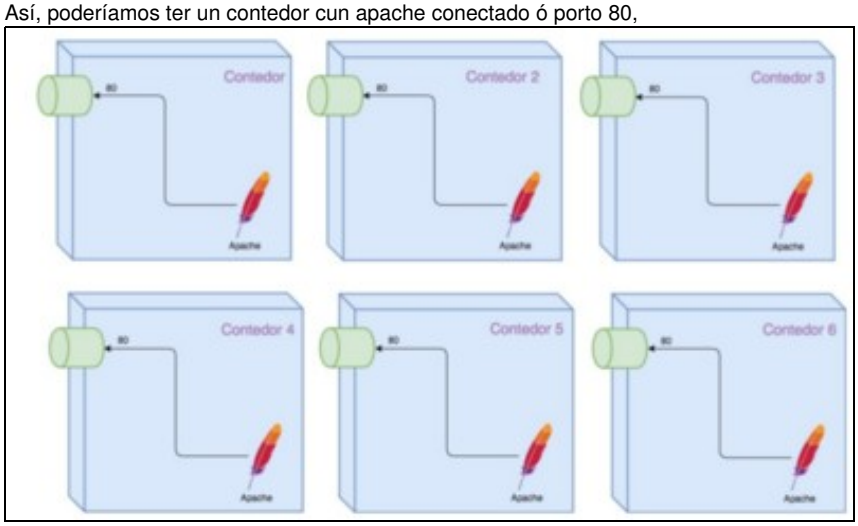
Portos do contedor

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,

- 

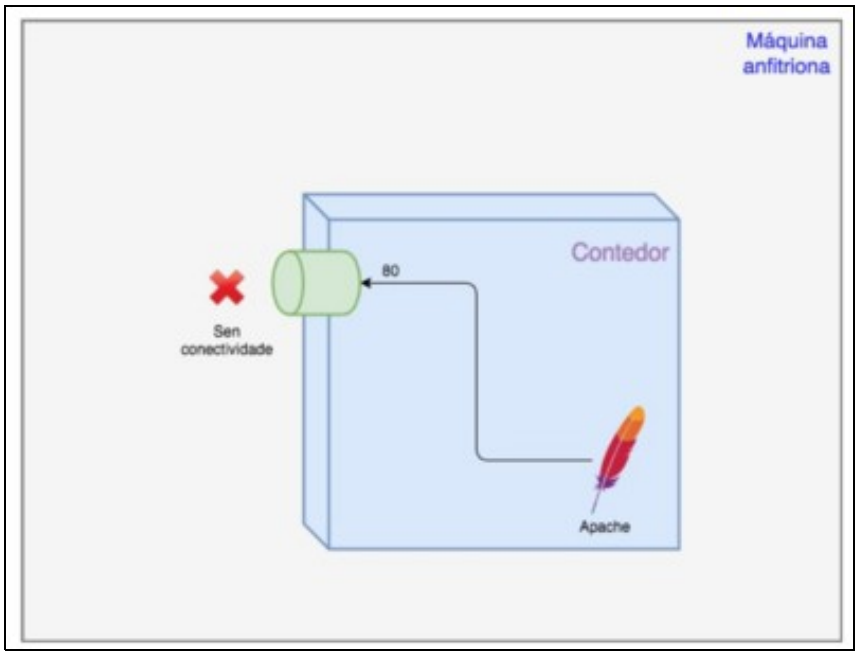
ou poderíamos ter varios contedores con servizos apache conectados ao porto 80.

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.

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 chegar ao noso contedor. Por sorte, ese traballo vaínolo facilitar moito Docker.

Conectar portos de docker ó exterior

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:

- 

O 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

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.

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

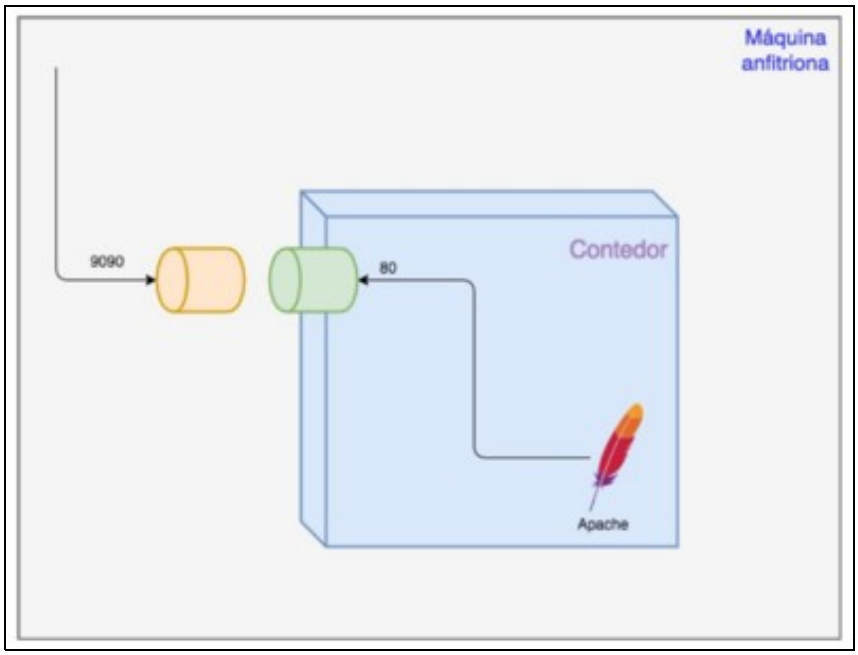
Se facemos agora un `docker ps`, verémos o contedor funcionado. O problema é que non podemos chegar de ningunha forma ó servizo de apache que corre escoitando no porto 80, porque o noso contedor nonten conectividade co exterior.

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

```
#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)

- 

O porto 80 do noso contedor está conectado ó porto 9090 da máquina anfitriona.

Podemos, por tanto, podemos chegar ó apache que corre dentro sen problema dende o exterior.

Paso de argumentos ó contedor

Un contedor ven sendo unha unidade illada dende o punto de vista dos recursos tradicionalmente compartidos nunha máquina: usuarios, pids, mounts, rede...

Este illamento é unha das claves da creación de contedores. Nembargantes, as 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 nesta maneira)

Antes de lanzar un contedor, Docker permite establecer cal será o valor do seu contorno mediante o paso de pares clave=valor.

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

```
#docker run -d -e ROOT_LOGIN=admin -e ROOT_PASSWORD=abc123. 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 contorno (ROOT_LOGIN con valor admin)
- O mesmo (-e) para a variable ROOT_PASSWORD (valor abc123.)

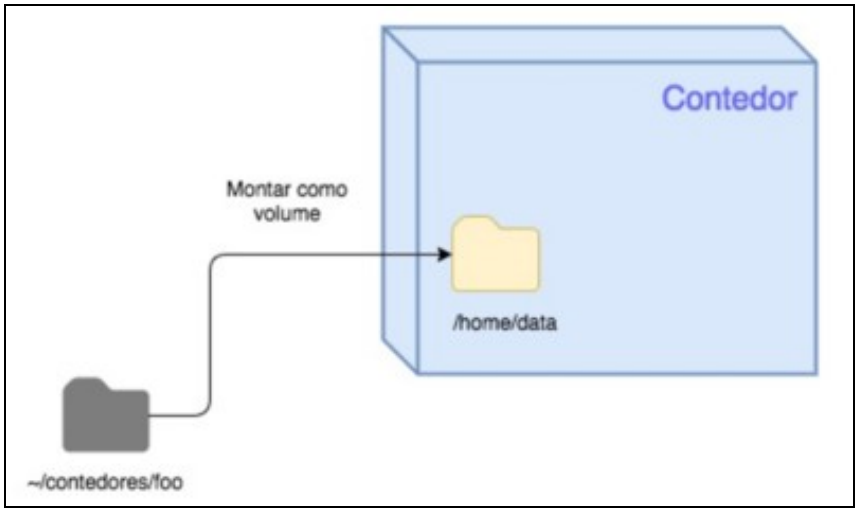
Docker, vai inxectar estas variables dentro do contedor antes de arrincalo de xeito que, se o programa está preparado para facelo, pode recolle-la súa configuración do ENV

Se queremos mudar o valor das variables de contorno dun contedor en funcionamento, compre reinicialo e/ou 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.

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.

- 

Este directorio pasa a ser accesible por parte 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 smartentry/debian bash
```

Como podemos ver:

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

Xestión de imaxes

Qué é unha imaxe? A imaxe é un dos conceptos fundamentáis no mundo da containerización.

Tal e como viramos, a containerización é unha técnica de virtualización que permite illar un proceso dentro dun SO de tal xeito que este último "pensa" que ten toda a máquina para él, puidendo executar versións específicas de software, estableéce-lo seu stack de rede ou crear unha serie de usuarios sen afecta-lo resto dos procesos do sistema. Unha imaxe abrangue o conxunto de software específico a empregar polo container 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 anfitrioa para poder lanzar containers baseados nesa imaxe.

A xestión de imaxes é un dos puntos fortes de Docker.

As imaxes en Docker están formadas por capas o que permite a súa modularidade e reutilización.

Para este exemplo, imos montar unha imaxe co servidor web Apache2.

- Primeira capa: O sistema base

  Xa sabemos que un container está completamente illado, coa excepción do Kernel, do Sistema Operativo anfitrión. Isto implica que, para que o container funcione, compre ter unha primeira capa na imaxe coas utilidades e programas fundamentais para garantir o funcionamento do software que queremos correr dentro do container. Noutras palabras, precisamos un Sistema Operativo como base da imaxe do container. Partiremos dunha imaxe base con SO Ubuntu

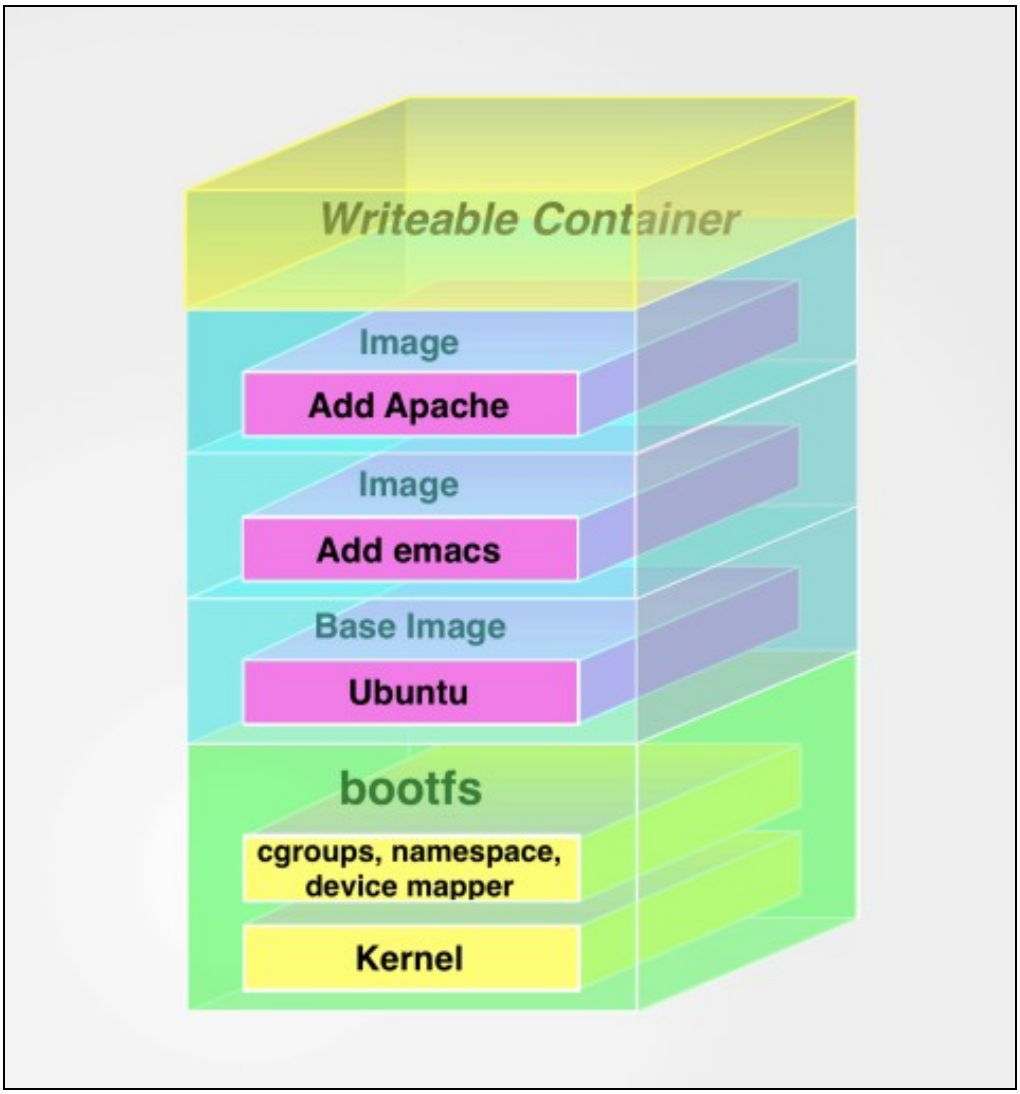
- Segunda capa: As dependencias de Apache 2.4

Neste caso é probable que previamente á instalación de Apache, teñamos que instalar outro software ou ter satisfeitas certas dependencias.

- Terceira capa: Instalación de Apache

  Nesta capa, instalariamos o software Apache 2.4

A imaxe ao final quedaría algo similar a



Nesta sección afirmamos que a imaxe é algo estático e inmutable, polo que as imaxes non se poden cambiar, agás a través dos métodos establecidos para o desenvolvemento e mantemento de imaxes por parte da suite de Docker.

Significa isto que un container non pode escribir en disco? Dentro do container, poderemos crear, borrar ou modificar ficheiros? Se a imaxe é algo inmutable, cómo se pode facer todo isto? Por suposto, Docker permite que os containers modifiquen o seu sistema de ficheiros, puidendo, se quixer, borrar todas as carpetas e os seus contidos.

Para poder facer isto, Docker emprega un mecanismo coñecido como copy-on-write (COW)

O "truco" é concetualmente sinxelo: Docker non corre o noso container directamente sobre a imaxe, senón que, por enriba da última capa da mesma, crea unha nova: a capa de container.

Realmente, a imaxe está formada polas capas propias da imaxe e por unha capa de container. Tan só a capa de container é de ESCRITURA/LECTURA. Deste xeito, os programas correndo no container poden escribir no sistema de ficheiros de xeito natural sen ser conscientes de que, realmente, están a escribir nunha capa asociada ó container e non na imaxe que é inmutable.

Isto posibilita que, cada container, poida face-las súas modificacións no sistema de ficheiros sen afectar a outros containers que estén baseados na mesma imaxe, dado que, cada container ten asociada unha capa de container específica para él.

Tal e como podemos ver, este mecanismo é moi útil. Non obstante, esto produce un problema: a volatilidade dos datos. Se se destrúe o container, destrúense os datos, xa que so son persistentes na capa de container e non na imaxe.

A forma correcta de facer que eses datos sexan persistentes é o uso de volumes de datos. Deste xeito os datos están nunha carpeta da máquina anfitrión e, o container, fai cambios no sistema de ficheiros que noné volátil, polo que, ainda que o container desapareza os datos está almacenados nun lugar estable e persistente.

Listado de imaxes

Para ver as imaxes que tempos dispoñibles no noso host anfitrión podemos executar

```
#docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
smartentry/debian     latest              89cd3d2371c1        5 days ago          101MB
debian                stretch             de8b49d4b0b3        3 weeks ago         101MB
portainer/portainer   latest              a01958db7424        5 weeks ago         72.2MB
```

Podemos ver que temos unha imaxe chamada *smartentry/debian*. Pero, de onde ven? Recordemos que previamente executamos un comando `docker run`, e nunha parte do seu proceso, descargou unha imaxe. No noso caso a imaxe *smartentry/debian*

 As imaxes almácenanse localmente en `/var/lib/docker`

Esa imaxe foi descargada dun repositorio. As imaxes residen en repositorios, e os repositorios, residen en rexistros.O rexistro por defecto é o rexistro público xestionado por Docker Inc. chamado Docker Hub

Dentro do Docker Hub, as imaxes almácenanse en repositorios. Podemos pensar en repositorios de iaxes en algo similar aos repositorios Git. Os repositorios conteñen imaxes, capas, e metadatos sobre esas imaxes. Cada repositiorio pode conter múltiples imaxes (por exemplo, o repositiorio ubuntu contén imaxes para Ubuntu 12.04, 12.10, 13.04, 13.10, 14.04, 16.04 ...). Podemos descargar unha imaxe en concreto dun repositorio.

```
#docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
Digest: sha256:
c6674c44c6439673bf56536c1a15916639c47ea04c3d6296c5df938add67b54b
Status: Downloaded newer image for ubuntu:16.04
```

Podemos identificar as imaxes dentro dun repositorio con etiquetas. Por exemplo, a imaxe debian ou ubuntu sempre se lista coas etiquetas que se lle poden aplicar, como pode ser 12.04, 14.10 quantal ou precise.

Para referirnos en concreto a unha imaxe, separamos o nome do repositorio da etiqueta co caracter ":".

```
#docker run -t -i --name new_container ubuntu:16.04 /bin/bash
root@79e36bff89b4:/#
```

Por defecto, se non especificamos ningunha etiqueta, Docker automáticamente descarga a etiqueta **latest**.

Por exemplo:

```
#docker run -t -i --name new_container ubuntu /bin/bash
root@79e36bff89b4:/#
```

descargará a imaxe coa etiqueta *latest*

Podemos buscar imaxes no rexistro público usando o comando `docker search`:

```
#docker search puppet
NAME                    DESCRIPTION           STARS OFFICIAL AUTOMATED
macadmins/puppetmaster Simple puppetmaster   21    [OK]
devopsil/puppet        Dockerfile for a      18    [OK]
. . .
```

É recomendable sempre que descarguemos imaxes, obter as que teñen máis estrelas e maior número de descargas.

Creación de imaxes propias

Xa vimos como descargar imaxes preparadas con contidos customizados. Pero como podemos modificalas para obter as nosas propias imaxes, e actualizalas e mantelas?

Hai dúas maneiras de crear imaxes de Docker: ? Via o comando de docker **commit** ? Via o comando de docker **build** con un ficheiro *Dockerfile*

O primeiro método non é o máis recomdendado, xa que construir as imaxes cun *Dockerfile* é moito máis flexible e poderoso.

Os ficheiros *Dockerfile* usan unha linguaxe básica chamada DSL (Domain Specific Language) con instrucións para construir as imaxes de Docker.

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

Sintaxe dun Dockerfile

O Dockerfile agrupa un conxunto de sentencias nun ficheiro de texto. Cada unha das sentencias son interpretadas e executadas por orde polo Docker producindo unha imaxe de saída.

Cada sentencia implica unha nova capa na imaxe.

So debe haber un ficheiro Dockerfile por directorio

Por exemplo:

```
# Version: 0.0.1
#imaxe da que partimos
FROM ubuntu:18.04
#mantedor do Dockerfile
LABEL maintainer="james@example.com"
#actualizamos as orixes de paquetes
RUN apt-get update
#instalamos o apache2
RUN apt-get install -y apache2
#Declaramos o punto de entrada ao contedor. É a execución de apache en FOREGROUND
ENTRIPOINT ["apachectl", "-D", "FOREGROUND"]
#O porto para conectar o contedor ao mundo exterior.
EXPOSE 80
```

Para construír a imaxe executamos:

```
docker build -t imaxe_propia
```

Este comando buscará un ficheiro con nome Dockerfile no directorio actual.

O Docker vainos a producir unha imaxe de nome imaxe_propia que terá un apache2 correndo no porto 80 e baseada na imaxe de Ubuntu. O DSL do Dockerfile é sinxelo pero moi completo. Aquí pódese ver unha relación dos comandos.

Instruccións do Dockerfile

FROM

Indica a imaxe orixinal da que se parte.

RUN

Especifica un comando a executar no proceso de construción da imaxe. Pode aparecer tantas veces como sexa necesario.

CMD

Permite especificar a instrución a executar cando se lanza un contedor. É similar á instrucción *RUN*, pero o comando execútase cando se lanza o contedor do mesmo xeito que fariamos con **docker run**. Tanto o comando coma os seus argumentos especifícanse nun formato de array cos argumentos entrecomiñados e separados por comas.

Exemplo:

```
CMD ["apachectl", "-D", "FOREGROUND"]
:No caso de especificar un comando con '''docker run''' deixaría de executarse o comando especificado na sección CMD do ''Do
```

ENTRYPOINT

É moi similar ao CMD. Diferenciase en que a súa execución é sempre obrigatoria e nunha é sobreescrita polo instrucción '*docker run*. Ao igual que a instrución CMD, soe indicarse coma un array cos argumentos entrecomiñados e separados por comas.

Exemplo:

```
ENTRYPOINT ["/usr/sbin/nginx", "-g", "daemon off;"]
```

ADD

Permite copiar ficheiros e directorios desde o anfitrión ao contedor. As rutas relativas no orixe (non se poden poñer absolutas), sono con respecto ao contexto de construción da imaxe. Como curiosidade, a orixe pode ser unha URL. Tamén se se especifica un arquivo comprimido, este será descomprimido no destino.

Exemplo:

```
ADD hom* /mydir/        # adds all files starting with "hom"
ADD dir/ /mydir/        # adds the dir/ directory.
ADD hom?.txt /mydir/    # ? is replaced with any single character, e.g., "home.txt"
ADD http://wordpress.org/latest.zip /root/wordpress.zip
```

COPY

É moi similar á instrución ADD, pero non permite empregar URL's nin descomprime os arquivos no destino. No caso de non empregar nin URL's nin descomprimir arquivos, recoméndase empregar COPY.

Exemplo:

```
COPY hom* /mydir/        # adds all files starting with "hom"
COPY dir/ /mydir/        # adds the dir/ directory.
COPY hom?.txt /mydir/    # ? is replaced with any single character, e.g., "home.txt"
```

WORKDIR

Esta instrución establece o directorio de traballo para as instrucións RUN, CMD, ENTRYPOINT, COPY e ADD. Pode empregarse múltiples veces, e se se emprega de forma relativa, sempre será con respecto á instrución WORKDIR previa.

Exemplo:

```
WORKDIR /opt/web
RUN install.sh
WORKDIR htdocs    #relativo a /opt/web.
CMD ['python', 'app.py']
```

EXPOSE

Informa a Docker que o contedor escoita nun determinado porto durante a súa execución. Pode especificarse tanto portos TCP coma UDP. Por defecto enténdese que son TCP. Non ten ningún efecto na execución, máis ben funciona como documentación. Hai que especificar os portos na instrución **docker run**

Exemplo:

```
EXPOSE 80/tcp 443 421/udp
```

ENV

Permite establecer variables de contorno que se poderán empregar nas instrucións subseguintes.

Exemplo:

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
RUN echo $myCat > /var/cat.txt
```

Orquestación de contedores

A orquestación de containers 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 ó 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.

O ***docker-compose*** é unha ferramenta de Docker deseñada para permitir unha orquestación a nivel dunha soa máquina.

O funcionamento é relativamete sinxelo:

- Os distintos containers da aplicación organízanse en servizos.
- Pódense crear redes privadas para conecta-los containers da aplicación.
- Aporta a posibilidad 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 infraestructura tal e como a establezcamos.

Instalación do docker-compose

Para instalar o docker-compose so hai que seguir as intrucións da documentación oficial dependendo do SO.

No caso de Debian 9, o proceso é o seguinte:

```
curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker
chmod +x /usr/local/bin/docker-compose
#opcionalmente podemos instalar o autocompletado de comandos para o ''docker-compose''. Supoñendo que o interprete de coandos é Bash
curl -L https://raw.githubusercontent.com/docker/compose/1.23.2/contrib/completion/bash/docker-compose -o /etc/bash_completion.d/doc
```

Fundamentos do docker-compose

Dentro do noso compose, imos definir tres entidades fundamentais:

- **Servizos**: trátase de definicións de contedores.
- **Redes**: definicións de rede que poden ser empregados polos contedores.
- **Volumes**: un directorio accesible polos contedores.

Servizos

Trátase dunha definición de contedor, no que se establece:

- A imaxe a montar.
- As variables de entorno.
- O número de réplicas do contedor a correr.
- Os portos de conexión.
- Os volumes que monta.

Exemplo:

```
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:80"
    volumes:
     -data:/var/wwww/data
```

Para arrancar o contedor, so temos que executar

```
docker-compose up -d
```

Redes

As redes en Docker son unha construcción que permite que os distintos contedores pertencentes se "vexan" uns ós outros sen necesidade de coñecer os seus enderezos IP.

Imaxinemos que temos un servizo en PHP que se conecta a outro que ten lanzada 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 do enderezo IP do contedor.

```
version: '3'
services:
  app:
    image: php
    network:
     - rede-foo
  bbdd:
    image: mysql
    network:
     - rede-foo
networks:
  rede-foo:
```

Vemos aquí como o contedor de app e o de bbdd están na mesma rede: a rede-foo. Esta rede atópase definida no mesmo docker-compose. Agora, e dado que os dous servizos están nesta rede, nun contedor de app existe definido o host con nome *bbdd* que corresponde co enderezo IP do contedor do servizo bbdd.

Volumes

Os volumes poden ser directamente definidos no docker-compose a través da propia DSL da ferramenta.

Os volumes poden empregarse para facer persistentes datos empregados no contedor, ou para compartir datos entre diferentes servizos ou contedores.

O seguinte sería un exemplo para o primeiro caso, de facer persistentes os datos dun contedor:

```
version: '3'
ghost:
  image: ghost
  volumes:
    - ./ghost/config.js:/var/lib/ghost/config.js
```

Neste caso, o ficheiro ghost/config.js, estará montado no directorio /var/lib/config.js do contedor.

Tamén nos permite facer persistentes os datos dunha BD e que non se perdan cando se destrúe o contedor.

```
version: '3'
services:
 mysql:
  image: mysql
  container_name: mysql
  volumes:
    - ./mysql:/var/lib/mysql
```

O directorio mysql, permitirá facer persistente a BD anque se destrúa o contedor.

No seguinte caso, faremos que dous contedores compartan información entre eles. Vai existir un volume, que ten o nome de "datos" que vai estar compartido polos dous contedores.

```
version: '3'
services:
 web1:
  image: ubuntu
  container_name: web1
  volumes:
    - datos:/var/lib/html
 web2:
  image: debian
  container_name: web2
```

```
    volumes:
      - datos:/var/www/html

volumes:
  datos:
```

Ciclo de vida dunha aplicación de compose

O docker-compose é unha ferramenta de liña de comando.

Para funcionar compre ter definido un **docker-compose.yaml** que é un ficheiro onde se expresa no DSL de compose unha infraestructura de contedores, redes e volumes. Unha vez que temos ese ficheiro creado, podemos:

- Lanzar a aplicación (**up**).
- Detela e borrala (**down**).
- Inspeccionala (**ps** e **top**).
- Construí-la súas imaxes (**build**).

Como xa sabemos o compose lee o noso ficheiro de docker-compose.yaml e comeza a crear:

- volumes
- redes
- contedores

O problema parece obvio: cómo nomea todos eses artefactos en Docker? A solución que adopta o docker-compose e crear un nome composto: Dominio ou App + Nome

Dado que cada contedor, volume e rede ten un nome (sexa o do servizo,ou o propio do artefacto) para evitar unha colisión de nomes, o que fai o docker-compose é determinar un dominio por aplicación e engadirlle o nome concreto do artefacto. O dominio ou nome da app virá dado polo nome do directorio onde se atope o docker-compose.yaml. Deste xeito, un ficheiro ~/foo/docker-compose.yaml terá

- Un contedor "app" chamarase foo_app
- Unha rede "privada" chamarase foo_privada