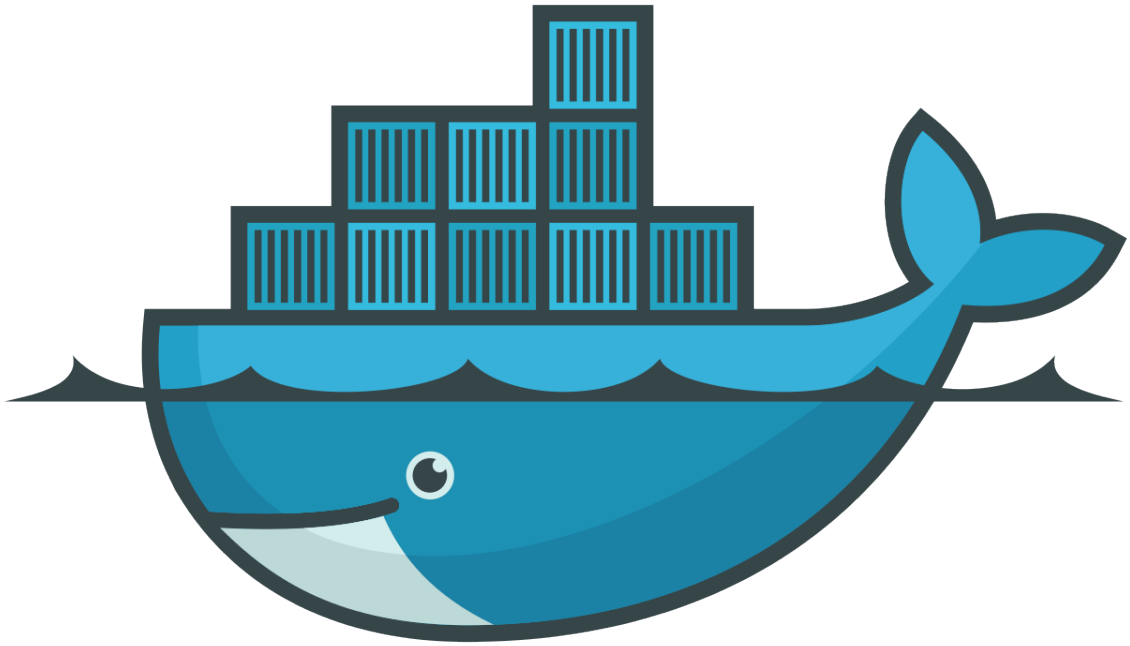


docker

Introducción a los contenedores

Rubén Gómez Olivencia

2023-2024

- Github: https://github.com/yuki

Puedes encontrar la última versión de este libro en formato **HTML** en el siguiente link, así como otros libros que he creado. Para descargar el código fuente en formato **Markdown** visita el repositorio en GitHub.

Información

Por favor, ponte en contacto conmigo si encuentras algún fallo, falta de ortografía o quieres mejorar de alguna manera este libro. Gracias.

I Contenedores

II Docker

Contenedores

1. Introducción

Hoy en día es muy habitual hacer uso de los sistemas de contenedores, siendo el más conocido Docker en el mundo del desarrollo de *software*. Este sistema trae consigo una serie de ventajas que veremos más adelante, que nos permite asegurar, entre otras cosas, que las versiones utilizadas en el entorno de producción son las mismas que durante las etapas de desarrollo.

En este documento se va a explicar cómo realizar la instalación y configuración de un sistema basado en contenedores Docker para poder arrancar servicios, y ciertas configuraciones que son necesarias conocer.

2. Sistemas de contenedores

Los sistemas de contenedores son un método de virtualización (conocido como "virtualización a nivel de sistema operativo"), en el que se permite ejecutar sobre una capa virtualizadora del núcleo del sistema operativo distintas instancias de "espacio de usuario".

Este "espacio de usuario" (donde se ejecutarán aplicaciones, servicios...) se les denomina **contenedores**, y aunque pueden ser como un servidor real, están bajo un mecanismo de aislamiento proporcionado por el *kernel* del sistema operativo, y sobre el que se pueden aplicar límites de espacio, recursos de memoria, de acceso a disco...

Información

Un contenedor es un espacio de ejecución de servicios al que se les puede aplicar límites de recursos (como la memoria, el acceso a disco…)

Desde el punto de vista del usuario, que un servicio se ejecute en una máquina virtual o en un contenedor es indistinguible. En cambio, desde el punto de vista de un administrador de sistemas o de un desarrollador, el uso de contenedores trae consigo una serie de ventajas que veremos en apartados posteriores.

2.1. Un poco de historia

Aunque está muy en boga el despliegue de aplicaciones haciendo uso de contenedores, no es un concepto nuevo, ya que lleva existiendo desde la década de los 80 en sistemas UNIX con el concepto de chroot.

Chroot, también conocido como "jaulas chroot", permitían ejecutar comandos dentro de un directorio sin que, en principio, se pudiese salir de dicha ruta. Tenía muy pocas restricciones de seguridad, pero era un primer paso al sistema de contenedores.

LXC nace en 2008 utilizando distintas funcionalidades del kernel Linux para proveer un entorno virtual donde poder ejecutar distintos procesos y tener su propio espacio de red. Con LXC nacen distintas herramientas para controlar estos contenedores, así como para crear plantillas y una **API que permite interaccionar con LXC** desde distintos lenguajes de programación.

Ha habido otras tecnologías en Linux, como OpenVz, pero nos centraremos en Docker, ya que es lo más conocido actualmente.

2.2. Qué es un contenedor y cómo se crea

Para entender qué es un contenedor dentro de la infraestructura Docker y cómo se crea, tenemos que diferenciar distintos conceptos:

- **Imagen Docker**
- **Contenedor Docker**

A continuación se van a detallar en profundidad.

2.2.1. Imágenes Docker

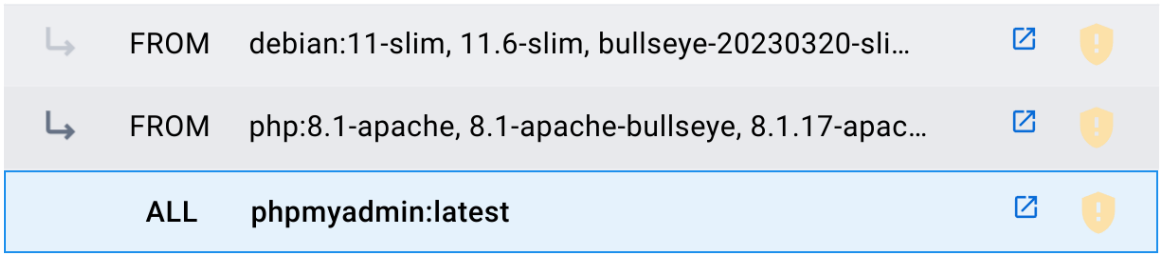
Para crear un contenedor necesitamos hacer uso de una **"imagen", que es un archivo inmutable (no modificable) que contiene el código de la aplicación que queremos ejecutar y todas sus dependencias necesarias**, para que pueda ser ejecutada de manera rápida y confiable independientemente del entorno en el que se encuentre.

Las imágenes, debido a su origen **sólo-lectura**, se pueden considerar como **"plantillas"**, que son la representación de una aplicación y el entorno necesario para ser ejecutada en un momento específico en el tiempo. **Esta consistencia es una de las grandes características de Docker**.

Información

Una imagen contiene el servicio que nos interesa ejecutar junto con sus dependencias, y son independientes del servidor donde se ejecuta.

Una imagen puede ser creada utilizando otras imágenes como base. Por ejemplo, la imagen de PHPMyAdmin empaqueta la aplicación PHPMyAdmin sobre la imagen **PHP** (versión 8.1-apache), que a su vez hace uso de la imagen **Debian** (versión 11-slim).



Jerarquía de imágenes usadas por PHPMyAdmin. Fuente.

A las imágenes creadas se les suele añadir etiquetas (**tags**) para diferenciar versiones o características internas. Cada creador determina las etiquetas que le interesa crear. Por ejemplo:

- **latest**: Se le denomina a la última imagen creada.
- php:**8.1-apache**: Indica que en esta imagen PHP la versión es la 8.1 y además cuenta con Apache.

Podemos utilizar imágenes públicas descargadas a través de un ***registry*** público, que no es otra cosa que un repositorio de imágenes subidas por la comunidad. El *registry* principal más utilizado es Docker Hub.

Información

Las imágenes Docker pueden ser públicas o privadas y se almacenan en un repositorio llamado *registry*, siendo el más conocido Docker Hub

Se pueden crear nuestras propias imágenes privadas, que pueden ser almacenadas en nuestros equipos o a través de un **registry privado** que podemos crear (también existen servicios de pago).

2.2.2. Contenedores Docker

Un contenedor Docker es un **entorno de tiempo de ejecución virtualizado donde los usuarios pueden aislar aplicaciones**. Estos contenedores son unidades compactas y portátiles a las que se les puede aplicar un sistema de limitación de recursos.

Información

Un contenedor se crea a través de una imagen, es la versión ejecutable de la misma que se crea en un entorno virtualizado

Un contenedor se crea a través de una imagen y es la versión ejecutable de la misma. Lo que se hace es crear una capa de escritura sobre la imagen inmutable, donde se podrán escribir datos. Se pueden crear un número ilimitados de contenedores haciendo uso de la misma imagen base.

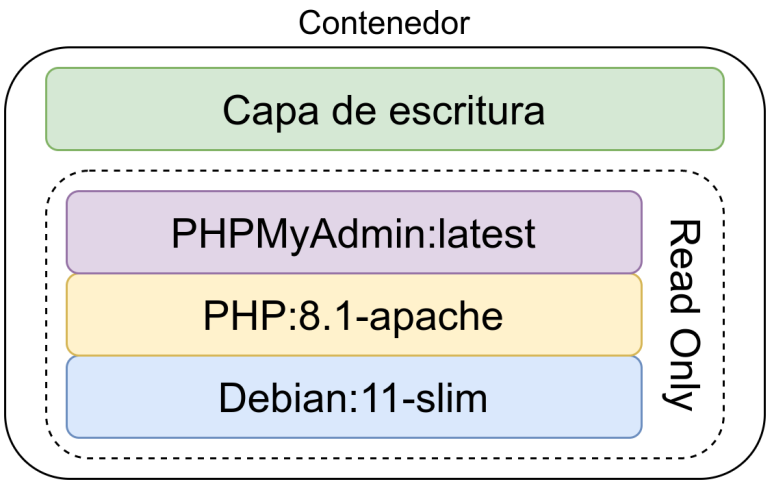


Imagen “interna” de un contenedor

La capa de escritura no es persistente y se pierde al eliminar el contenedor, es decir, **los datos de un contenedor se eliminan al borrar el contendor**. Para evitar este comportamiento se puede hacer uso de un volumen persistente de datos, de esta manera esos datos no se pierden.

¡Cuidado!

Los datos creados dentro de un contenedor se borran al eliminar el contenedor

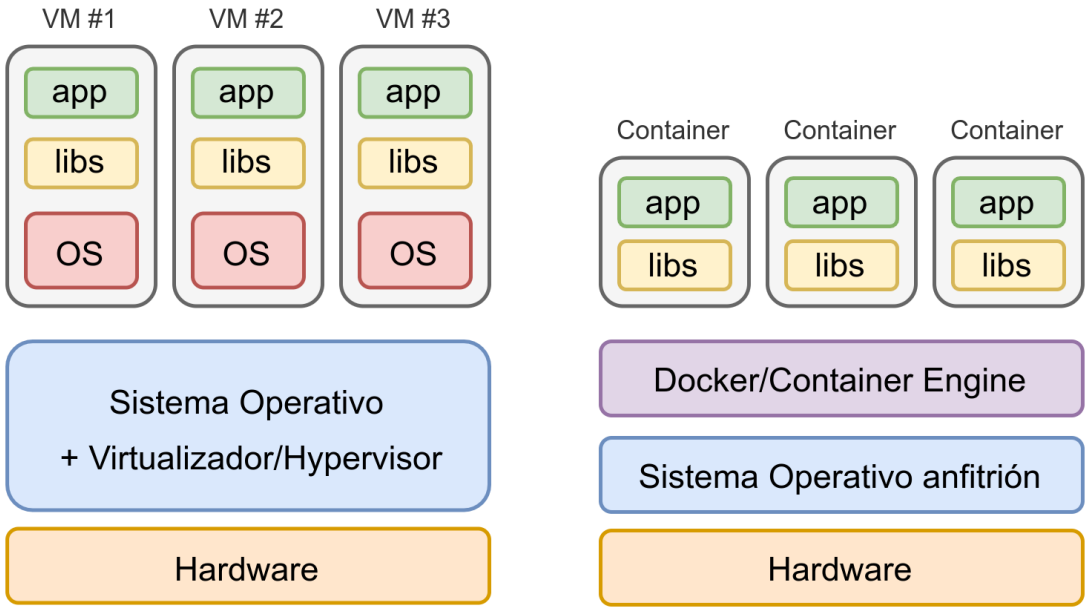
2.3. Contenedores vs. Máquinas virtuales

El uso de máquinas virtuales está muy extendido gracias a que cada vez es más sencillo crearlas. Esto no quiere decir que siempre sea la mejor opción, por lo que se va a realizar una comparativa teniendo en cuenta distintos aspectos a la hora de realizar un desarrollo con máquinas virtuales y con sistemas de contenedores.

2.3.1. Infraestructura

La creación de máquinas virtuales nos permite crear entornos aislados en los que poder instalar el Sistema Operativo que más nos interese y con ello poder instalar el software y los servicios que necesitemos.

Las máquinas virtuales se virtualizan a nivel de hardware, donde debe existir un Sistema Operativo con Hypervisor que permita dicha virtualización. Por otro lado, los contenedores se virtualizan en la capa de aplicación, haciendo que este sistema sea mucho más ligero, permitiendo utilizar esos recursos en los servicios que necesitamos hacer funcionar dentro de los contenedores.



Infraestructura Máquinas Virtuales vs Docker

En la imagen se puede apreciar una comparativa diferenciando cómo quedaría una infraestructura de 3 aplicaciones levantadas en distintas máquinas virtuales o en distintos contenedores.

Tal como se puede ver en la imagen, **al tener cada servicio en una máquina virtual separada**, se va a tener que virtualizar todo el Sistema Operativo en el que se encuentre, con el consiguiente **coste de recursos (memoria RAM y disco duro) y con el coste en tiempo de tener que realizar la configuración y securización del mismo**.

Información

Usando contenedores la infraestructura se simplifica notáblemente

Por otro lado, en un sistema de contenedores, cada contenedor es un servicio aislado, en el que sólo tendremos que preocuparnos (en principio) de configurar sus parámetros.

2.3.2. Ventajas durante el desarrollo

A la hora de desarrollar una aplicación es habitual hacer pruebas utilizando distintas versiones de librerías, *frameworks* o versiones de un mismo lenguaje de programación. De esta manera, podremos ver si nuestra aplicación es compatible.

Cuando se hace uso de una máquina virtual dependemos de las versiones que tiene nuestra distribución y es posible que no podamos instalar nuevas versiones u otras versiones en paralelo.

Por ejemplo, la última versión de PHP actualmente es la 8.4.11 y de Apache la 2.4.65:

- En **Debian 12** sólo se puede instalar PHP 8.2.29 y Apache 2.4.62.
- En **Ubuntu 24.04** la versión de PHP es la 8.3.6 y la de Apache la 2.4.58.

Con Docker, podremos levantar contenedores con distintas versiones del servicio que nos interese en paralelo para comprobar si nuestra aplicación/servicio es compatible.

Información

Con Docker es posible levantar servicios con distintas versiones en paralelo

Por otro lado, si un desarrollador quiere utilizar un sistema operativo distinto, no se tendrá que preocupar de si su distribución tiene las mismas versiones. O en el caso de usar Windows/Mac, no tener que estar realizando instalaciones de las versiones concretas.

2.3.3. Ventajas durante la puesta en producción

Ligado al apartado anterior, durante la puesta en producción es obligatorio hacer uso de las mismas versiones utilizadas durante el desarrollo para asegurar la compatibilidad.

¡Cuidado!

Para asegurar la compatibilidad en producción, siempre se debe usar la misma versión de los servicios que en desarrollo

Si tenemos un servidor que no está actualizado, o en el mismo servidor tenemos distintas aplicaciones que requieren utilizar distintas versiones de software, en un entorno de máquinas virtuales se hace muy complejo, ya que lo habitual será tener que instalar nuevas máquinas virtuales.

¡Atención!

No siempre es posible tener distintas versiones del mismo software en un mismo servidor

En un entorno con contenedores, al igual que se ha comentado antes, esto no es problema.

2.3.4. Rapidez en el despliegue

Ligado a todo lo anterior, realizar el despliegue de un entorno de desarrollo/producción es más rápido utilizando contenedores, sin importar el sistema operativo en el que nos encontremos.

Información

El despliegue con contenedores es más rápido.

Más adelante se verá cómo realizar el despliegue de distintos servicios haciendo uso de un único comando.

II

Docker

1. Docker

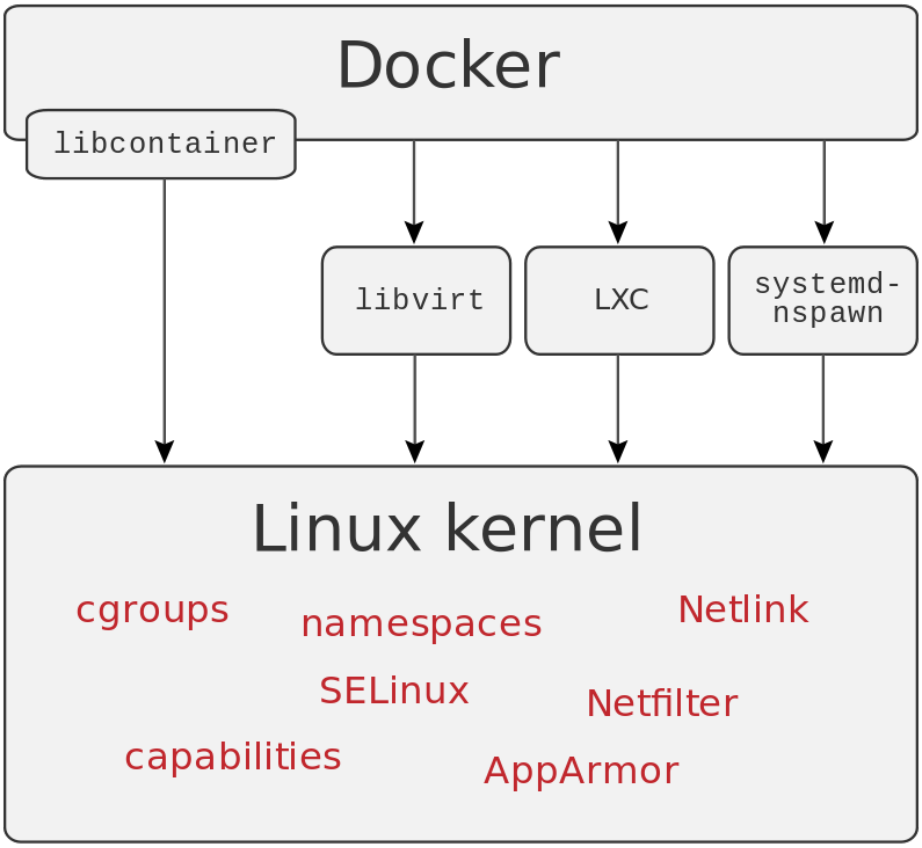
Docker es un proyecto de Software Libre nacido en 2013 que permite realizar el despliegue de aplicaciones y servicios a través de contenedores de manera rápida y sencilla, tal como veremos más adelante.

Estos contenedores proporcionan una capa de abstracción y permiten aislar las aplicaciones del resto del sistema operativo a través del uso de ciertas características del kernel Linux.

Dentro del contenedor, se puede destacar el aislamiento a nivel:

- Árbol de procesos
- Sistemas de ficheros montados
- ID de usuario
- Aislamiento de recursos (CPU, memoria, bloques de E/S...)
- Red aislada

Al igual que sucede con otro tipo de *software*, para que Docker haga uso de todas estas características, está construido haciendo uso de otras aplicaciones y servicios.



Tecnologías usadas por Docker. Fuente: Wikipedia

En el 2015 la empresa Docker creó la ***Open Container Initiative***, proyecto actualemente bajo la Linux Foundation, con la intención de diseñar un estándar abierto para la virtualización a nivel de sistema

operativo.

1.1. Instalación

Dependiendo del sistema operativo en el que nos encontremos, Docker tiene la opción de instalarse de distintas maneras. En sistemas operativos GNU/Linux cada distribución tiene un paquete para poder realizar la instalación del mismo.

Instalación de Docker en Ubuntu

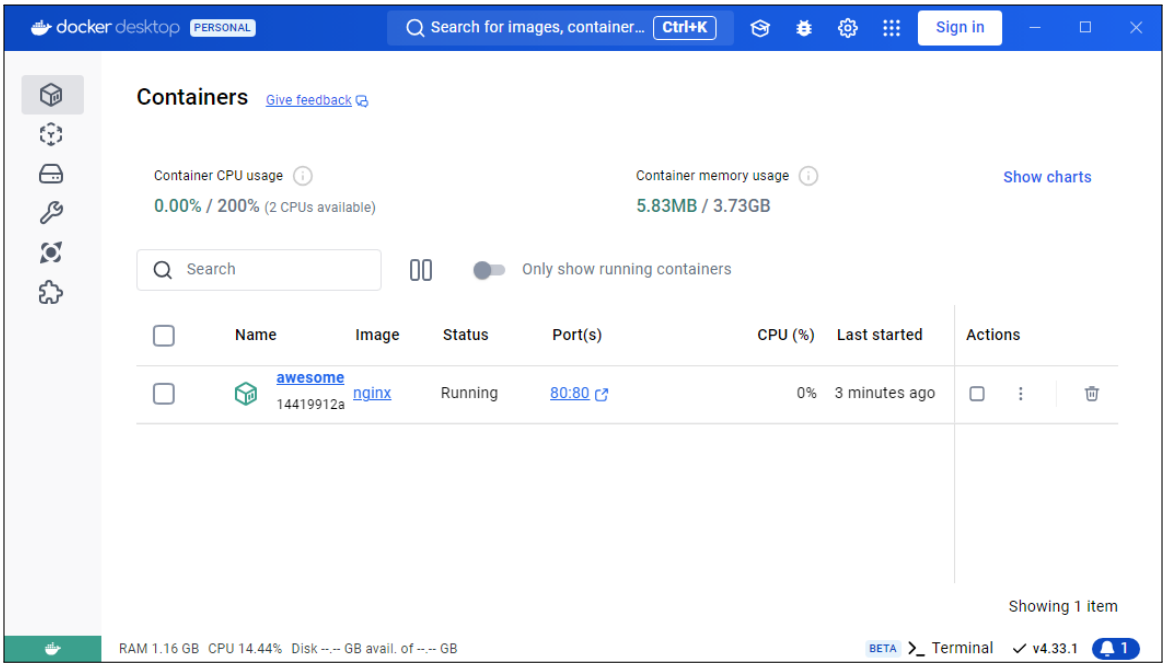
```
ruben@vega:~$ sudo apt install docker.io
```

¡Cuidado!

El nombre del paquete en Ubuntu y Debian es “docker.io”.

1.1.1. Instalación en Windows y/o Mac

En sistemas Windows y MacOS existe la opción de instalar Docker Desktop, una versión que utiliza una máquina virtual para simplificar la instalación en estos sistemas. De todas maneras, también se instala el CLI para poder usar los comandos que veremos a continuación.



Docker Desktop

En caso de Windows, se requiere tener las extensiones de virtualización habilitadas en la BIOS/UEFI y una de estas dos opciones, que habrá que configurar antes de instalar Docker Desktop:

- Usar WSL2.
- Usar Hyper-V y el sistema de contenedores de Windows.

1.2. Configuración

Tras realizar la instalación veremos cómo el servicio Docker ha levantado un interfaz nuevo en nuestra máquina, cuya IP es **172.17.0.1/16**, siendo el direccionamiento por defecto.

Nueva IP en el equipo

```
ruben@vega:~$ ip a
...
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 02:42:9c:1f:e2:90 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
      valid_lft forever preferred_lft forever
```

Esta IP hará de **puente** (similar a lo que sucede con las máquinas virtuales) cuando levantemos contenedores nuevos. Los contenedores estarán dentro de ese direccionamiento 172.17.0.0/16, por lo tanto, aislados de la red principal del equipo.

Información

Los contenedores que levantemos estarán en la red 172.17.0.0/16

1.3. Usar Docker con usuario sin privilegios

Para poder hacer uso de Docker con un usuario sin permisos de root/administrador, se debe añadir a los usuarios no privilegiados dentro de un grupo. Dependiendo de dónde usemos Docker, tendremos que realizarlo de una manera u otra.

1.3.1. Linux

En este caso, el grupo que debe tener el usuario es "**docker**", que se lo podemos añadir al usuario de distintas maneras:

- Editar el fichero `/etc/group`, y añadir el usuario al grupo
- Ejecutar estos comandos que ponemos a continuación:

Añadir el grupo docker al usuario correspondiente

```
ruben@vega:~$ sudo addgroup ruben docker
[sudo] password for ruben:
Adding user `ruben' to group `docker' ...
Adding user ruben to group docker
Done.
ruben@vega:~$ newgrp docker
```

A partir de ahora ya se puede hacer uso de Docker con el usuario al que hayamos añadido al grupo.

1.3.2. Windows

Para que un usuario en Windows pueda usar Docker Desktop tiene que pertenecer al grupo "**docker-users**". Para añadirlo, desde un PowerShell **con permisos de administrador**, ejecutaremos:

Añadir al usuario "usuario" al grupo docker-users

```
PS C:\Users\ruben> net localgroup "docker-users" "usuario" /add
```

1.4. Primeros pasos

El comando `docker` tiene muchas opciones, por lo que es recomendable ejecutarlo sin parámetros. De esta manera se pueden ver todas las opciones y una ayuda simplificada para cada una de ellas.

Algunas de las opciones del comando docker

```
ruben@vega:~$ docker
Usage:  docker [OPTIONS] COMMAND

Management Commands:
builder     Manage builds
completion  Generate the autocompletion script for the specified shell
config      Manage Docker configs
container   Manage containers
context     Manage contexts
image       Manage images
manifest    Manage Docker image manifests and manifest lists
network     Manage networks
node        Manage Swarm nodes
plugin      Manage plugins
secret      Manage Docker secrets
service     Manage services
stack       Manage Docker stacks
swarm       Manage Swarm
system      Manage Docker
trust       Manage trust on Docker images
volume      Manage volumes

Commands:
...
```

Para cada una de estas opciones, se le puede añadir el parámetro `--help` para mostrar la ayuda. Hay un segundo apartado que se ha cortado, en el que se incluyen más comandos.

Para asegurar que el servicio Docker está funcionando, podemos hacer uso de `>_ docker info`, que nos mostrará mucha información acerca del servicio. Pero si lo que queremos es comprobar si tenemos algún contenedor corriendo, es más sencillo hacer `>_ docker ps` (que es la versión simplificada de `>_ docker container ls`):

>_ Comprobar estado de Docker y contenedores levantados

```
ruben@vega:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ruben@vega:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

En este caso, como no hay ningún contenedor levantado, sólo muestra las cabeceras de las columnas del listado.

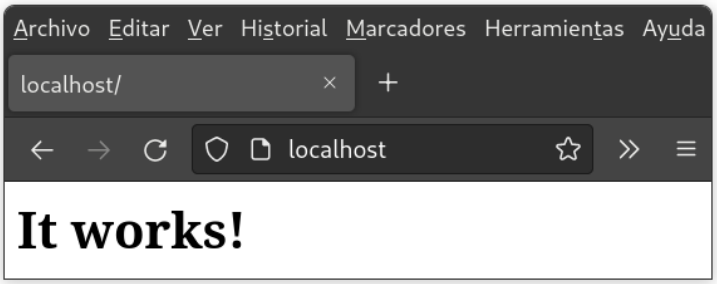
1.5. Levantando nuestro primer contenedor

Es momento de crear nuestro primer contenedor. Para ello, dado que se está usando la consola, hay que hacer uso del comando `>_ docker` con una serie de parámetros. En este caso se ha optado por levantar el servicio **Apache HTTPD**:

>_ Levantando el primer contenedor

```
ruben@vega:~$ docker run -p 80:80 httpd
AH00558: httpd: Could not reliably determine the server's ...
AH00558: httpd: Could not reliably determine the server's ...
[Fri Mar 24 18:25:14.194246 2023] [mpm_event:notice] ...
[Fri Mar 24 18:25:14.194347 2023] [core:notice] [pid  ...
172.17.0.1 - - [24/Mar/2023:18:25:41 +0000] "GET / HTTP/1.1" 304 -
```

Vemos los logs del servicio Apache al arrancar y si vamos al navegador a la dirección http://localhost muestra lo siguiente:



Y para entender lo que hace el comando, los parámetros son:

- **docker**: Cliente de consola para hacer uso de Docker.
- **run**: Ejecuta un comando en un nuevo contenedor (y si no existe lo crea).
- **-p 80:80**: Publica en el puerto 80 del servidor el puerto 80 utilizado en el contenedor. Se puede pensar que es como hacer un **port-forward** en un firewall.
- **httpd**: Es la **imagen** del contenedor que se va a arrancar. En este caso, la imagen del servidor Apache HTTPD.

Y si vemos qué muestra el estado de docker, vemos cómo aparece el contenedor levantado.

Comprobar estado de Docker y contenedores levantados

```
ruben@vega:~$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS                NAMES
a1c3362b0d6c   httpd     "httpd-..."   3 seconds ago   Up 2 seconds   0.0.0.0:80->80/tcp   great
```

En la columna PORTS se puede apreciar cómo aparece que se ha levantado el puerto **`0.0.0.0:80`** (escucha en el puerto 80 para cualquier IP del sistema operativo) que es una redirección al puerto **`80/TCP`** interno del contenedor.

1.6. Contenedores en *background* y más opciones

Tal como se puede ver en el ejemplo anterior, el contenedor se queda en primer plano, viendo los logs del Apache. Esto para ver qué es lo que está sucendiendo durante el desarrollo puede ser útil, pero lo ideal es que el contenedor arranque en modo ***background***, y cuando necesitemos vayamos a ver los logs.

A continuación se va a arrancar un nuevo contenedor de Apache con nuevos parámetros:

Crear un contenedor Web en el puerto 8080

```
ruben@vega:~$ docker run --name mi-apache -d -p 8080:80 httpd
```

Los nuevos parámetros son:

- **`--name mi-apache`**: De esta manera se le da un nombre al contenedor, para poder identificarlo de manera rápida entre todos los contenedores creados.
- **`-d`**: Este parámetro es para hacer el ***detach*** del comando, y de esta manera mandar a ***background*** la ejecución del contenedor.
- **`-p 8080:80`**: Publica en el puerto 8080 del servidor el puerto 80 utilizado en el contenedor. Se puede pensar que es como hacer un **port-forward** en un firewall.

1.7. Parar, arrancar y borrar contenedores

Hasta ahora hemos aprendido a crear contenedores, pero en ciertos momentos nos puede interesar parar un contenedor que no estemos utilizando, o una vez haya cumplido su función, borrarlo.

1.7.1. Parar contenedores

Para parar un contenedor, debemos conocer el nombre del mismo o su ID (que es único). Estos datos los podemos conocer a través del comando `>_ docker ps`.

Con esto, podemos ejecutar:

>_ Parar un contenedor

```
ruben@vega:~$ docker stop mi-apache
```

1.7.2. Arrancar un contenedor parado

Una vez parado un contenedor, o al reiniciar el servidor, si queremos arrancar un contenedor parado, debemos conocer también su ID o nombre.

Para visualizar todos los contenedors (tanto los arrancados como los parados), lo podemos hacer a través del comando `>_ docker ps -a`.

Gracias a ese listado, podemos volver a arrancar un contenedor que esté parado con `>_ docker start mi-apache`, siendo "mi-apache" el contenedor que queremos arrancar.

1.7.3. Borrar un contenedor

Si queremos borrar un contenedor, éste debe estar parado, ya que Docker no nos va a dejar borrar un contenedor que está en ejecución.

Es interesante borrar contenedores que hayamos creado de pruebas o contenedores que ya no se vayan a utilizar más, para de esta manera liberar recursos.

Para borrarlo, similar a los casos anteriores, se hará con `>_ docker rm mi-apache`.

2. Variables de entorno

Algunos contenedores tienen la opción de recibir variables de entorno al ser creados. Estas variables pueden afectar al comportamiento del contenedor, o para ser inicializado de alguna manera distinta a las opciones por defecto.

El creador de la imagen Docker puede crear las variables de entorno que necesite para después utilizarlas en su aplicación. A modo de ejemplo, se va a utilizar la imagen de la aplicación PHPMyAdmin.

A continuación se van a crear 2 contenedores de PHPMyAdmin, diferenciados por el puerto, el nombre, y la variable de entorno **PMA_ARBITRARY**:

- El primer contenedor va a estar en el puerto 8081, se le va a dar el nombre "myadmin-1" y no va a tener la variable de entorno incializada.
- El segundo contenedor usará el puerto 8082, llamado "myadmin-2" y la variable **PMA_ARBITRARY** inicializada a "1", tal como aparece en la documentación de la imagen de PHPMyAdmin.

Para ello, se han ejecutado los siguientes comandos:

Creación de dos contenedores PHPMyAdmin

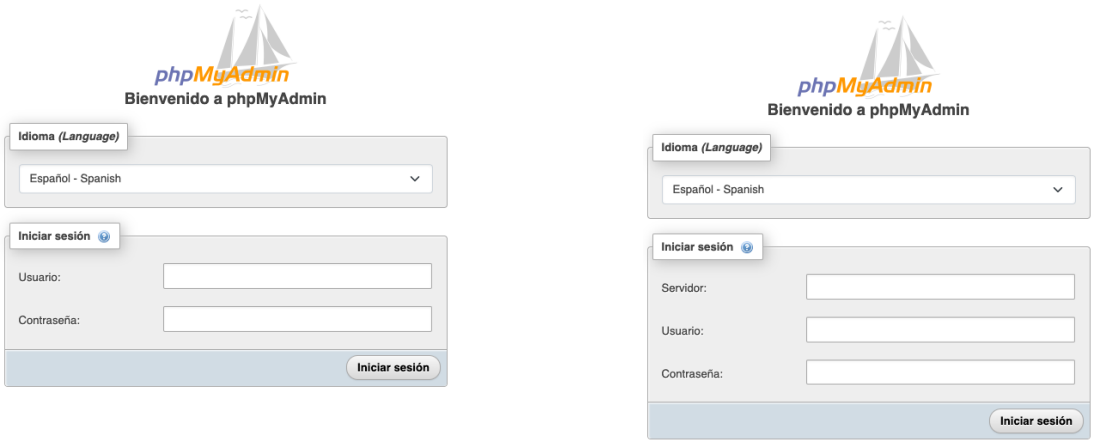
```
ruben@vega:~$ docker run --name myadmin-1 -d -p 8081:80 phpmyadmin

ruben@vega:~$ docker run --name myadmin-2 -e PMA_ARBITRARY=1 -d -p 8082:80 phpmyadmin
```

Tal como se puede ver, al segundo contenedor se le ha pasado un nuevo parámetro **-e**, que significa que lo que viene a continuación es una variable de entorno (en inglés **environment**). En este caso, la variable de entorno es **PMA_ARBITRARY** que se ha inicializado a **1**.

Si ahora en nuestro navegador web apuntamos al puerto 8081 y al puerto 8082 de la IP de nuestro servidor, veremos cómo existe una ligera diferencia en el formulario que nos muestra la web.

En el formulario del puerto 8081 (donde no hemos inicializado la variable) sólo podemos indicar el usuario y la contraseña. Por el contrario, en el formulario del puerto 8082, al inicializar la variable **PMA_ARBITRARY**, y tal como nos dice la documentación de la imagen, nos permite indicar la IP del servidor MySQL al que nos queremos conectar.



A la izquierda formulario del puerto 8081, sin variable inicializada. A la derecha, puerto 8082 con variable inicializada.

Dado que una variable puede afectar al comportamiento (o la creación) del servicio que levantemos a través de un contenedor, es importante leer la documentación e identificar las variables que tiene por si nos son de utilidad.

Información

Es recomendable leer la documentación de las imágenes Docker para identificar las posibles variables de entorno que existen y ver si nos son útiles.

3. Volumen persistente de datos

Hasta ahora hemos levantado un contenedor a través de una imagen que levanta el servicio Apache, mostrando su página por defecto. Podríamos escribir en el contenedor la página HTML que nos interesase,

pero hay que entender que **los datos de un contenedor desaparecen cuando el contenedor se elimina**.

Para que los cambios realizados dentro de un contenedor se mantengan, tenemos que hacer uso de los denominados **volúmenes de datos**. Esto no es más que **hacer un montaje de una ruta del disco duro del sistema operativo dentro de una ruta del contenedor**.

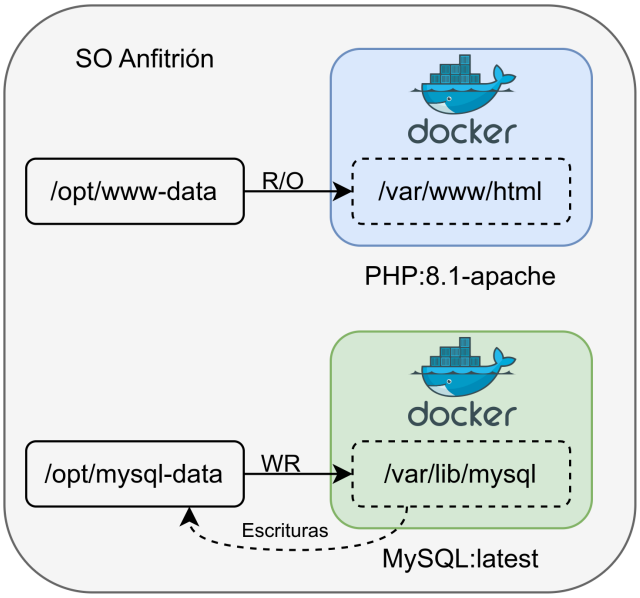
Estos volúmenes que le asignamos al contenedor pueden ser de dos tipos:

- **Sólo lectura**: Nos puede interesar asignar un volumen de sólo lectura cuando le pasamos ficheros de configuración o la propia web que queremos visualizar.
- **Lectura-Escritura**: En este caso se podrá escribir en el volumen. Por ejemplo, el directorio donde una base de datos guarda la información o una web donde deja imágenes subidas por usuarios.

De esta manera, tendremos que asignar el número de volúmenes necesarios a cada contenedor dependiendo de la imagen utilizada, el servicio que se levanta y lo que queremos hacer con los datos que le asignemos o generemos en el contenedor.

En la siguiente imagen se puede ver una infraestructura con dos contenedores y dos volúmenes:

- **Contenedor Web**: Se le asigna un volumen en modo **sólo lectura** cuya ruta original está en `/opt/www-data`, que dentro del contenedor está en `/var/www/html`.
- **Contenedor MySQL**: Dado que los datos de la base de datos deben ser guardados, en este caso se le asigna un volumen que permite escritura. Por lo tanto, lo que se crea dentro del contenedor en `/var/lib/mysql` realmente se estará guardando en el sistema operativo anfitrión en `/opt/mysql-data`.



Ejemplo de dos volúmenes asignados a distintos contenedores

3.1. Añadir volumen de escritura al crear un contenedor

Al añadir un volumen cuando creamos un contenedor hace que por defecto sea en modo lectura-escritura. Cualquier escritura realizada dentro del contenedor en la ruta especificada va a resultar en que el fichero se

creará en la ruta indicada del sistema operativo anfitrión.

Añadir volumen de escritura al crear un contenedor

```
ruben@vega:~$ ls /opt/mysql-data
ruben@vega:~$ docker run -d -p 3306:3306 --name mi-db \
    -v /opt/mysql-data:/var/lib/mysql \
    -e MYSQL_ROOT_PASSWORD=my-secret-pw \
    mysql:latest
ruben@vega:~$ ls /opt/mysql-data
auto.cnf      client-key.pem      '#innodb_temp'      server-cert.pem   ...
```

Para este ejemplo se ha creado un contenedor usando la imagen de MySQL, al que se le ha asignado un volumen (**por defecto se asigna permitiendo la escritura**) y un parámetro necesario para realizar la posterior conexión con contraseña.

- **-v /opt/mysql-data:/var/lib/mysql**: A través del parámetro **-v** se le indica al contenedor que se le va a pasar un volumen. Posteriormente se le indica la ruta del sistema operativo anfitrión `/opt/mysql-data` que se montará dentro del contenedor en `/var/lib/mysql`.

- **-e MYSQL_ROOT_PASSWORD=my-secret-pw**: El parámetro "**-e**" sirve para pasarle al contenedor **variables de entorno**. En este caso, y tal como dice la web de la imagen MySQL, esta es la manera de asignar la contraseña del usuario **root** durante la inicialización de la base de datos.

Tras crear el contenedor, y asegurarnos que está levantado haciendo uso del comando `docker ps`, podemos realizar la conexión desde el sistema operativo anfitrión o desde cualquier otro lugar usando la contraseña indicada previamente.

Comprobar estado de MySQL

```
ruben@vega:~$ mysql -h127.0.0.1 -uroot -P3306 -p
    Enter password:
    ...
    MySQL [(none)]> show databases;
    +--------------------+
    | Database           |
    +--------------------+
    | information_schema |
    | mysql              |
    | performance_schema |
    | sys                |
    +--------------------+
    4 rows in set (0,007 sec)
```

3.2. Añadir volumen en modo sólo-lectura

A continuación se van a explicar los pasos para levantar un contenedor que contiene una web simple creada en PHP, que está alojada en la ruta `/opt/www-data` del sistema operativo anfitrión.

Añadir volumen sólo lectura al crear un contenedor

```
ruben@vega:~$ ls /opt/www-data
index.php
ruben@vega:~$ docker run -d -p 80:80 --name mi-web \
    -v /opt/www-data:/var/www/html:ro \
    php:8.2.4-apache
```

El parámetro nuevo asignado en la creación de este contenedor es:

- **`-v /opt/www-data:/var/www/html:ro`**: Para indicarle que le vamos a asignar un volumen siendo la ruta real en el sistema de ficheros del sistema operativo anfitrión `/opt/www-data` y la ruta destino **dentro del contenedor** y que va a ser en modo **read-only** `/var/www/html`.

Más adelante, cuando veamos cómo entrar dentro de un contenedor Docker, se podría usar el comando para ir a la ruta dentro del contenedor y comprobar que efectivamente está en modo sólo-lectura.

3.3. Entrar dentro de un contenedor Docker

Normalmente no suele ser necesario entrar dentro de un contenedor, ya que, tal como se ha dicho antes, cualquier modificación realizada dentro de él se perderá (salvo que sea dentro de un volumen persistente).

Aún así, para realizar pruebas o comprobaciones del correcto funcionamiento de una imagen puede ser interesante entrar dentro de un contenedor. Para ello, el comando a ejecutar es el siguiente:

Acceder a un contenedor

```
ruben@vega:~$ docker exec -it mi-db /bin/bash
```

Los parámetros utilizados son:

- **`exec`**: Indicamos que queremos ejecutar un comando dentro de un contenedor que está corriendo.
- **`-it`**: Son dos parámetros unidos, que sirven para mantener la entrada abierta (modo interactivo) y crear una TTY (consola)
- **`mi-db`**: Es el nombre del contenedor al que se quiere entrar. También se puede indicar el **ID** del contenedor.
- **`/bin/bash`**: el comando que queremos ejecutar. En este caso, una shell **bash**. En algunos casos esta shell no está instalada y debemos usar **`/bin/sh`**

Hay que tener en cuenta que dentro de un contenedor está el mínimo software posible para que la aplicación/servicio funcione, por lo que habrá muchos comandos que no existan.

4. Otros comandos útiles

Para obtener toda la información de un contenedor, incluido su estado, volúmenes utilizados, puertos, …

Obtener toda la información de un contenedor

```
ruben@vega:~$ docker inspect mi-db
```

Listar las imágenes descargadas en local. Al tener las imágenes en local, no hará falta volver a descargarlas, por lo que crear un nuevo contenedor que haga uso de una de ellas será mucho más rápido.

Listado de imágenes en local

```
ruben@vega:~$ docker image ls
REPOSITORY       TAG             IMAGE ID       CREATED         SIZE
php              8.2.4-apache    de23bf333100   3 days ago      460MB
httpd            latest          192d41583429   3 days ago      145MB
mysql            latest          483a8bc460a9   3 days ago      530MB
```

Borrar una de las imágenes que no se esté utilizando en ningún contenedor.

Borrar una imagen concreta

```
ruben@vega:~$ docker image rm httpd
```

Cuando un contenedor está en modo ***detached*** no aparecen los logs, por lo que para poder visualizarlos tenemos un comando especial para ello.

Ver los logs de un contenedor

```
ruben@vega:~$ docker logs mi-web -f
172.17.0.1 - - [26/Mar/2023:18:05:29 +0000] "GET / HTTP/1.1" 200 248
172.17.0.1 - - [26/Mar/2023:18:05:29 +0000] "GET / HTTP/1.1" 200 248
172.17.0.1 - - [26/Mar/2023:18:05:29 +0000] "GET / HTTP/1.1" 200 248
```

Listar los volúmenes existentes en el sistema. El listado muestra los que se están utilizando en contenedores (activos o parados) o los que se han utilizado en contenedores que ya no existen.

Listar volúmenes

```
ruben@vega:~$ docker volume ls
DRIVER    VOLUME NAME
local     0d6c400a6407f5cdea81a2f0158222fdd87d7f3b3e2b5969ca466d743fc71f5c
local     1d2f52018e17af0689e070a55337154c1dd68517c54435ecc24d597f7509d43c
local     6b72797227ef4708ca23ee1dfcb4b651b42eeacefd4166b898407ad4aadda10c
```

Si queremos realizar una limpieza de todos los recursos (contenedores, imágenes, volúmenes) que no se estén utilizando, se puede utilizar el siguiente comando.

Borrar recursos que no estén activos

```
ruben@vega:~$ docker system prune -a
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all images without at least one container associated to them
- all build cache

Are you sure you want to continue? [y/N] y
```

¡Cuidado!

El comando anterior hace que se borren contenedores parados

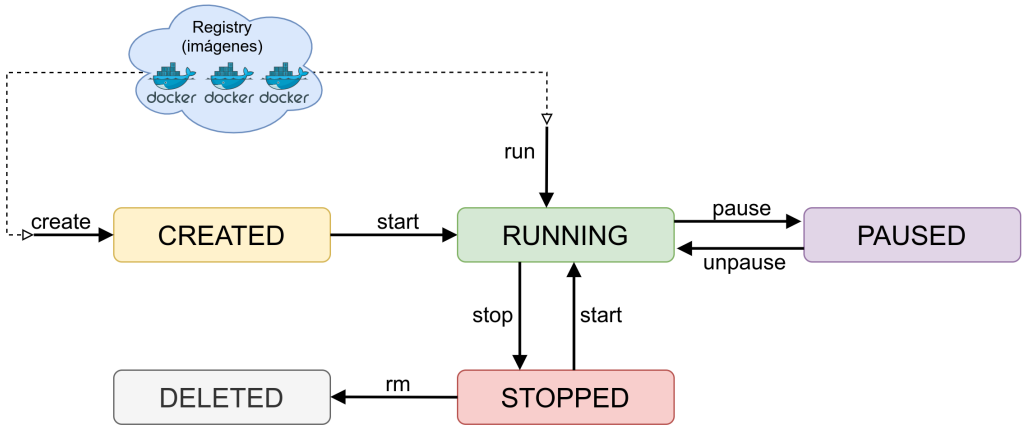
Para conocer las estadísticas de uso de cada contenedor.

Ver las estadísticas de los contenedores

```
ruben@vega:~$ docker stats
CONTAINER ID   NAME    CPU %   MEM USAGE / LIMIT     MEM %   NET I/O          BLOCK I/O   PIDS
2fcf97530766   mi-web  0.00%   54.62MiB / 15.47GiB   0.34%   9.44MB / 172kB   0B / 0B     6
413d6e9f590f   mi-db   0.55%   357.9MiB / 15.47GiB   2.26%   319kB / 280B     0B / 0B     38
```

5. Ciclo de vida de un contenedor Docker

Un contenedor tiene un ciclo de vida que puede pasar por distintos estados. Para pasar entre estados se debe realizar a través de distintos comandos de Docker.



Estados de un contenedor

En la imagen se representa los estados más básicos junto con los comandos para pasar entre ellos.

III

Alternativas a Docker

1. Alternativas a Docker

Aunque Docker actualmente sea la herramienta más extendida para la creación de sistemas de contenedores, en el 2015 se creó la ***Open Container Initiative*** (OCI). Es un proyecto de la Fundación Linux para diseñar un estándar abierto para asegurar que las plataformas de contenedores no estén vinculadas a ninguna empresa o proyecto concreto.

Actualmente se han creado tres especificaciones para el desarrollo y uso de contenedores, cuya documentación se puede encontrar en Github y en la web de la OCI:

- **Formato de imagen**: Es la especificación de cómo debe ser el formato para construir, transportar y preparar la imagen con la que después se creará el contenedor.
- **Entorno de ejecución**: Es la especificación de referencia para el comportamiento y la configuración de bajo nivel de los entornos de ejecución de contenedores. El programa de referencia es **runc**, que normalmente es llamado desde aplicaciones de más alto nivel como containerd o cri-o.
- **Distribución**: Define el protocolo API para facilitar y estandarizar la distribución de contenido. Sería la parte para crear un *registry* (del estilo Docker Hub), o un repositorio de imágenes.

1.1. Containerd y runc

La alternativa más ***cruda*** es hacer uso de los sistemas que nos proporciona la OCI, como son:

- **Containerd**: El servicio de ejecución para los contenedores.
- **runc**: el comando de consola que nos permite crear y lanzar los contenedores.

Este sistema, aunque pueda ser el más "directo", ya que hace uso de los comandos directos del sistema, puede ser el más complejo. En algunos casos para realizar ciertas tareas habrá que ejecutar distintas acciones que con Docker, y otras alternativas, son más directas.

1.2. Nerdctl

Nerdctl es un programa de línea de comandos para **containerd** compatible con Docker. Tal como dice **nerdctl** en su *FAQ*, el objetivo es facilitar y experimentar con las características más punteras que tiene *containerd* que no tiene Docker.

Como a nivel de línea de comandos es compatible con Docker, cualquier tarea que sepamos hacer con Docker, lo podremos realizar con nerdctl:

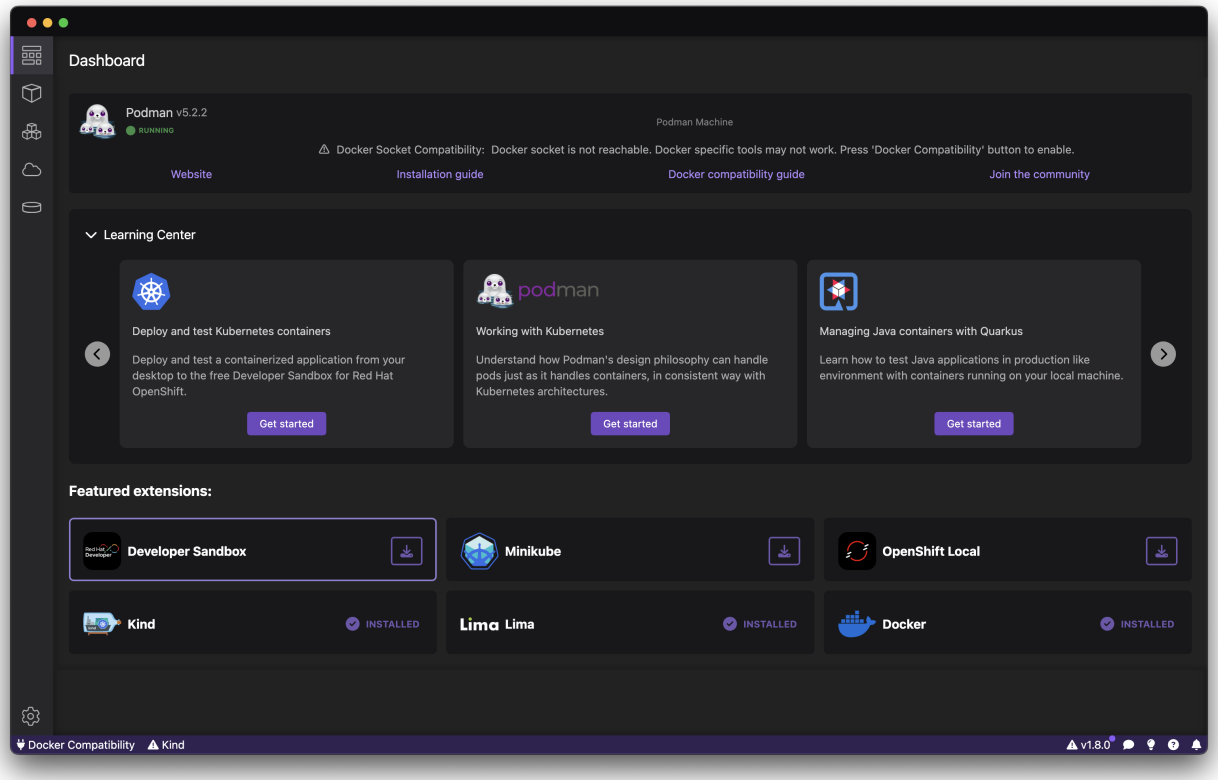
Comprobar estado de contenedores con nerdctl

```
ruben@vega:~$ nerdctl ps
CONTAINER ID    IMAGE     COMMAND    CREATED    STATUS    PORTS     NAMES
```

1.3. Podman

Podman es un sistema para manejar contenedores, imágenes, *pods*, control de Kubernetes de manera local. Es desarrollado por Red Hat.

Podman cuenta con un interfaz gráfico similar a Docker Desktop, y al igual que este, en entornos Windows y MacOS, creará una máquina virtual con un sistema Linux donde correrá internamente Podman.



Aplicación de escritorio de Podman

IV

Imágenes Docker

1. Crear imágenes Docker

Tal como se ha dicho previamente, una imagen Docker es un archivo inmutable que contiene el servicio que queremos ejecutar, junto con sus dependencias necesarias. Aunque en hub.docker.com existe infinidad de imágenes, en algún momento nos puede interesar crear una propia, que contenga un servicio propio.

Para crear una imagen tenemos que crear un fichero llamado **Dockerfile**, y las opciones que podemos utilizar en él están en la documentación oficial.

Es recomendable también tener en cuenta la web donde se comenta hacer uso de buenas prácticas durante la creación de las imágenes.

1.1. Instrucciones en Dockerfile

El número de instrucciones que se pueden utilizar en el fichero Dockerfile es limitada, y están explicadas en la documentación, entre las que podemos destacar:

- **ADD**: Se pueden añadir ficheros o directorios propios a la imagen, para que estén disponibles al levantar el servicio. Por ejemplo, ficheros de configuración o el código de nuestra aplicación. Se pueden utilizar URLs de internet o repositorios GIT como origen
- **CMD**: Este parámetro indica el comando que se va a ejecutar al arrancar el contenedor. Este comando puede contener parámetros. **Sólo puede haber un CMD al crear la imagen**, y de haber más, se usará sólo el último.
- **COPY**: Similar a ADD. Existe una web comparativa donde se explican las diferencias.
- **ENTRYPOINT**: Es utilizado cuando el contenedor va a usarse como un ejecutable. Puede ser un *script* creado por nosotros. En la documentación existe un apartado en el que se explica la interacción entre CMD y ENTRYPOINT.
- **ENV**: Sirve para setear variables de entorno.
- **EXPOSE**: Informa a Docker de los puertos (y el protocolo) que se van a querer utilizar al arrancar la imagen.
- **FROM**: Para que un fichero Dockerfile sea válido tiene que empezar con una directiva FROM, para indicar cuál es la imagen por defecto a utilizar. Lo habitual suele ser hacer uso de imágenes oficiales. Docker recomienda usar imágenes de Alpine Linux debido a su bajo consumo de RAM, entre otras cosas.
- **RUN**: Instrucción para ejecutar cualquier comando, que genera una nueva capa en la imagen.
- **VOLUME**: Sirve para exponer los directorios que deben ser volúmenes externos (por ejemplo: directorios de almacenamiento de bases de datos, directorios de configuración…)

- **WORKDIR**: Setea el directorio para los comandos RUN, CMD, ENTRYPOINT, COPY y ADD que se ejecuten posteriormente.

Existen otras instrucciones, pero estas serían las más básicas para crear una imagen propia.

2. Ejemplo de Dockerfile

Como ejemplo, vamos a utilizar el siguiente fichero **Dockerfile** que tendremos en un directorio de nuestro equipo.

Fichero Dockerfile de pruebas

```
FROM php:8.4-apache
COPY src/ /var/www/html/
```

V

Windows Subsystem for Linux

1. Introducción

WSL (del inglés *Windows Subsystem for Linux*) es una capa de compatibilidad que ha desarrollado Microsoft para correr ejecutables creados para sistemas Linux de manera nativa en Windows.

Desde el año 2019 la versión por defecto es la 2, que introdujo muchos cambios en el sistema, ya que esta versión corre dentro de una capa de virtualización creada a través de un **subconjunto del virtualizador Hyper-V**. Esto hace que el kernel que se está ejecutando sea mucho más compatible con los binarios de Linux que la versión 1. A pesar de usar virtualización, también mejora el rendimiento respecto a la versión anterior.

Por defecto WSL no viene instalado en el sistema en Windows 10, por lo que es necesario realizar la instalación para poder ejecutar las aplicaciones que deseemos (como Docker, por ejemplo). En versiones Windows 11 sí viene instalado.

2. Instalación

Para realizar la instalación necesitaremos una versión compatible de Windows (10 build 19041 o posterior). Hoy en día no debería ser problema si tenemos el sistema actualizado.

Para realizar la instalación necesitaremos de permisos de administrador, y lo realizaremos, para mayor comodidad, desde una consola de PowerShell o la nueva terminal de Windows. Para ello, abrimos la consola con permisos de administrador y ejecutamos:

Instalación de WSL en Windows 10

```
PS C:\Users\ruben> wsl --install
```

Tras la instalación es necesario reiniciar el sistema para que aplique cambios y levante los servicios necesarios. Una vez reiniciado nos aparecerá una ventana donde nos pedirá que introduzcamos el usuario y la contraseña para el sistema Linux recién instalado.

Información

Por defecto, la distribución que se instala es Ubuntu.

3. Configuraciones

Tras realizar la instalación podremos observar que Windows ha realizado una serie de configuraciones para adecuar la nueva instalación del servicio.

- Al tener las instancias levantadas, se genera un nuevo interfaz de red Hyper-V, con una red 172.25.240.0/20.

- Tal como se ha dicho, en WSL-2 las instancias realmente son máquinas virtuales Hyper-V. La configuración de las instancias se encuentran en el directorio `AppData` del usuario que las crea. Por ejemplo, para Debian, se encuentra el disco duro dentro de `./AppData/Local/Packages/TheDebianProject…/LocalState/ext4.vhdx`

  ¡Atención!

  El directorio AppData está oculto por defecto en el explorador de ficheros de Windows.

- Dentro de las instancias se puede acceder al disco duro de Windows a través de `/mnt/c`, o la unidad correspondiente.

- Desde Windows se puede acceder al sistema de ficheros de las instancias a través del explorador de ficheros, ya que nos aparecen las instancias que tenemos creadas.

Para poder realizar una configuración general de todo el ecosistema WSL se puede realizar desde una aplicación (disponible desde verano del 2024):

Aplicación de configuración de WSL

En esta aplicación se pueden modificar aspectos tan interesantes como:

- Procesadores lógicos dentro del WSL
- Tamaño máximo de la memoria
- Modo de red (NAT, mirrored, o proxy)

3.1. Configuración avanzada

Para realizar una configuración avanzada, existe una documentación desde dos puntos de vista:

- `wsl.conf`: fichero de configuración que se sitúa en el directorio `/etc` en las distribuciones. **Esta configuración sólo afecta en la distribución correspondiente** donde se haya realizado. Más adelante veremos un ejemplo para el uso de Docker.
- `.wslconfig`: **fichero en el directorio del usuario de Windows**. Este fichero tendrá la configuración que afectará a todas las distribuciones que hayamos instalado con WSL 2.

4. WSL con usuarios no privilegiados

WSL hace uso de ciertas características que necesitan de permisos de administrador. En caso de no tener permisos de administrador, por defecto sólo se hará uso de WSL versión 1, por lo que el rendimiento de los subsistemas es peor.

Por lo tanto, para poder utilizar WSL2 **se necesita tener acceso a los credenciales de administrador** y ejecutar los siguientes comandos:

Usar WSL2 en usuarios no privilegiados

```
PS C:\Users\usuario> wsl --update

PS C:\Users\usuario> wsl --set-default-version 2
```

5. Comandos útiles

Una vez realizada la instalación, existen ciertos comandos que nos pueden ser útil a la hora de hacer uso de WSL. No se van a detallar todos, ya que con `wsl -help` obtendremos la ayuda del comando y muchas más opciones.

Mostrar todas las distribuciones que se pueden instalar

```
PS C:\Users\ruben> wsl -l -o
```

Instalar una distribución Debian

```
PS C:\Users\ruben> wsl --install -d Debian
```

Mostrar las distribuciones instaladas

```
PS C:\Users\ruben> wsl -l -v
```

Ejecutar una distribución instalada y entrar en ella

```
PS C:\Users\ruben> wsl -d Debian
ruben@DESKTOP-1RVJ3UP:/mnt/c/Users/ruben$
```

Terminar/Apagar una distribución

```
PS C:\Users\ruben> wsl -t Debian
```

Apagar todas las instancias

```
PS C:\Users\ruben> wsl --shutdown
```

Eliminar una distribución instalada

```
PS C:\Users\ruben> wsl --unregister Debian
```

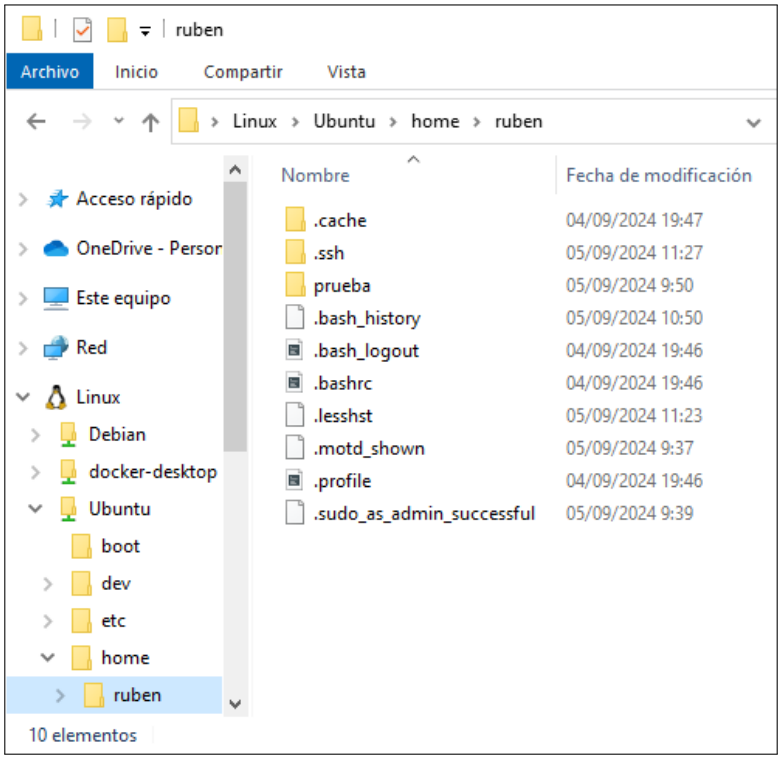
Clonar una instancia de una distribución instalada

```
PS C:\Users\ruben> wsl --export Ubuntu ubuntu.tar

PS C:\Users\ruben> wsl --import Ubuntu2 ubuntu2_files ./ubuntu.tar
```

6. Acceder al sistema de ficheros de los subsistemas

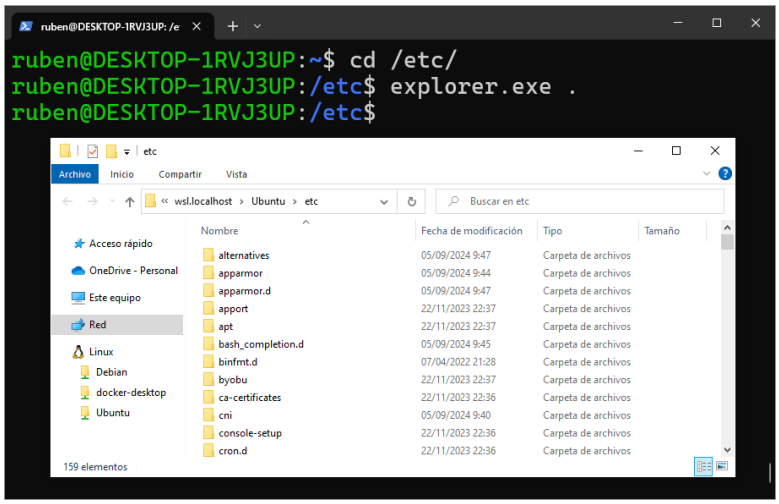
Microsoft ha creado la posibilidad de poder acceder al sistema de ficheros de los Linux que levantemos con WSL a través del explorador de ficheros de Windows. Esto permite copiar/pegar ficheros entre las distribuciones que hayamos creado y el propio sistema base. En la siguiente imagen se ve el explorador de archivos con tres subsistemas Linux creados:



Contenido de "/home/ruben" de Ubuntu desde el explorador Windows

Como alternativa, desde dentro del subsistema Linux podemos ejecutar aplicaciones Windows, por lo

que estando en cualquier ruta, podemos llamar al comando `explorer.exe .` que nos abrirá el explorador de Windows en esa misma ruta:



Abrir explorador Windows desde Linux

6.1. Rendimiento de los sistemas de ficheros en WSL

Por cómo funciona WSL y la gestión de sistemas de ficheros entre el sistema anfitrión Windows y el subsistema Linux, tenemos que tener claro que **existen dos sistemas de ficheros independientes, pero accesibles entre ellos**:

- **Sistema de ficheros de Windows**: Es el sistema de ficheros de nuestro equipo Windows. Hay que tener en cuenta, que cuando entramos al subsistema Linux, por defecto nos encontramos en ese mismo sistema de ficheros:

Al entrar al subsistema Linux, estamos en el sistema de ficheros de Windows

```
PS C:\Users\ruben\Desktop> wsl -d Ubuntu
ruben@DESKTOP-1RVJ3UP:/mnt/c/Users/ruben/Desktop$
```

Tal como se puede ver, al entrar en Ubuntu, la ruta en la que nos encontramos es `/mnt/c/Users/ruben/Desktop`, que es el sistema de ficheros de Windows (`C:`) **montado en la ruta de Linux** `/mnt/c`. Por eso, desde Linux tendremos acceso a todo el sistema de ficheros de Windows desde esa ruta.

- **Sistema de ficheros del subsistema Linux**: La máquina virtual de Linux que hemos creado tiene su propio sistema de ficheros, que como en cualquier Linux, está en `/`.

Pasamos al sistema de ficheros real de Linux

```
ruben@DESKTOP-1RVJ3UP:/mnt/c/Users/ruben/Desktop$ cd
ruben@DESKTOP-1RVJ3UP:~$ pwd
/home/ruben
```

Tal como se puede ver, con el comando `pwd`, ahora nos encontramos en el sistema de ficheros de

Linux real.

A la hora de hacer uso de aplicaciones en el susbsistema Linux, es recomendable hacerlo dentro del sistema de ficheros de Linux, no en el sistema montado, debido a que el rendimiento en el sistema montado es mucho peor. Por tanto, nos debemos asegurar que la aplicación está en la ruta correcta.

¡Cuidado!

Usar el sistema de ficheros de Windows montado en el subsistema Linux perjudica el rendimiento.

7. Docker dentro de WSL

Si queremos tener Docker dentro de un subsistema Linux, existen dos posibilidades completamente diferenciadas:

- Utilizar **Docker Desktop en Windows**. Docker Desktop usará WSL por debajo y tenemos la posibilidad de que los subsistemas hagan uso del Docker engine creado instalado en Docker Desktop. Este es el modo aconsejado por la documentación de Microsoft.
- Instalar el Docker Engine dentro de un subsistema Linux.

Este último método lo explicamos a continuación.

7.1. Instalar Docker Engine en WSL

En algunos casos nos puede interesar no hacer uso de Docker Desktop, porque lo que queremos es tener la posibilidad de un control total de Docker, como si de una instalación de máquina virtual completa de Linux se tratara. Es por ello que debemos realizar una pequña modificación en el funcionamiento del subsistema Linux correspondiente.

¡Atención!

Es más sencillo hacer uso de Docker Desktop en lugar de este sistema.

Supongamos que hemos creado el subsistema Linux de la distribución Ubuntu, deberemos entrar en ella, y tendremos que crear un fichero en `/etc/wsl.conf` con el siguiente contenido.

Configuración del fichero wsl.conf

```
[boot]
systemd=true
```

Salimos de la distribución y tenemos que forzar su reinicio. Una vez realizado estos pasos, si volvemos a entrar en la instancia, `systemd` estará funcionando y por tanto podremos instalar y hacer uso del Docker Engine como si fuese una máquina virtual creada al modo tradicional.