

2.2 Los flags -it

LOS FLAGS -it

Una de las cosas en las que **me atasqué un poco** cuando empecé a trabajar con docker fue el hecho de que **"a veces"** me encontraba con que **los contenedores se paraban y ya no podía acceder a ellos ni iniciarlos de nuevo (con docker start)**. Al principio no sabía el porqué y además me desconcertaba el hecho de que me pasaba con algunos contenedores y con otros no.

Sin entrar en detalles, que derivan de cómo se han construido los contenedores, la solución para que no os pase esto está en:

- **Siempre usar el flag -it** al ejecutar una orden docker run si es un contenedor que no tiene servicios. Este -it es la unión del flag -i (--interactive) y el flag -t (--tty) que lo que hacen es abrir la entrada estándar del contenedor que estamos ejecutando y permitir la posibilidad de abrir un terminal en el contenedor. Es decir, **nos va a permitir interactuar con él**.
- **No debemos añadir al final otra orden que no sea /bin/bash** (u otro shell que puedan contener los contenedores) ya que eso sobrescribe la orden de arranque de algunos contenedores (shell o inicio de servicio dependiendo del tipo de contenedor) y será imposible volver a arrancarlos una vez parados. Hay que tener en cuenta que una vez arrancados, si lo hemos hecho bien, siempre podremos ejecutar órdenes en el contenedor, tal y como veremos posteriormente.

Debemos de tener en cuenta que si los contenedores no son contenedores con servicios se paran tras salir la primera vez de interactuar con ellos. Para iniciarlos de nuevo, como ya he dejado ver en el primer párrafo, tenemos la orden docker start cuyo funcionamiento es muy simple. Lo vamos a ilustrar con un ejemplo.

```
# Arrancar el contenedor con nombre servidorWeb (debe estar parado)
> docker start servidorWeb
# Arrancar el contenedor con ID 511eed5992d
> docker start 511eed5992d
```

2.5 Ejecutar órdenes en los contenedores.

Con los contenedores en ejecución vamos a querer ejecutar ordenes en ellos. Querremos realizar operaciones como:

- Instalar paquetes.
- Modificar o ver el contenido de ciertos ficheros.
- Habilitar ciertos módulos de servicios
- etc...

Esto lo podemos hacer de **dos maneras** o bien **obteniendo un terminal** del contenedor y ejecutando las órdenes necesarias desde allí o bien **directamente** ejecutando una orden determinada "contra" el contenedor. Para ambos casos voy a necesitar la orden **docker exec** y es **NECESARIO QUE EL CONTENEDOR ESTÉ EN EJECUCIÓN**.

La sintaxis de esta orden es bastante sencilla y muy similar a la de docker run:

**docker exec [opciones] nombre_contenedor orden
[argumentos]**

Algunas de las opciones más importantes son:

- **-it** (-i y -t juntos) si vamos a querer tener interactividad con el contenedor ejecutando un shell (/bin/bash normamente). Una vez tenemos el terminal ya podremos trabajar desde dentro del propio sistema.
- **-u o --user** si quiero ejecutar la orden como si fuera un usuario distinto del de root.
- **-w o --workdir** si quiero ejecutar la orden desde un directorio concreto.

Lo vamos a ver mejor con algunos ejemplos:

```
# Obtener un terminal en un contenedor que ejecutar un servidor Apache (httpd) y que se llama web
```

```
> docker exec -it web /bin/bash
```

```
root@5d96ce1f7374:/usr/local/apache2#
```

```
# Mostrar el contenido de la carpeta /usr/local/apache2/htdocs del contenedor web. Como no hace falta interactividad no es necesario -it
```

```
> docker exec web ls /usr/local/apache2/htdocs
```

```
# Crear directamente un fichero "HOLA MUNDO" en el directorio raíz del servidor apache. Utilizo sh -c para ordenes compuestas o complejas
```

```
> docker exec -it web sh -c "echo 'HOLA MUNDO' > /usr/local/apache2/htdocs/index.html"
```

Adicionalmente existe otra orden que nos va a ser de mucha utilidad cuando trabajemos con contenedores. la orden **docker cp** que me permite mover ficheros desde mi sistema al contenedor y desde el contenedor a mi sistema. Su sintaxis es muy sencilla y la vamos a ilustrar con dos ejemplos, uno en cada sentido:

```
# Copiar mi fichero prueba.html al fichero
```

```
/usr/local/apache2/htdocs/index.html de mi contenedor llamado web que es
```

un servidor Apache (httpd)

```
> docker cp prueba.html web:/usr/local/apache2/htdocs/index.html
```

#Copiar el fichero index.html que se encuentra en /usr/local/apache2/htdocs/index.html de mi contenedor llamado web un fichero llamado test.html en mi directorio HOME

```
> docker cp web:/usr/local/apache2/htdocs/index.html $HOME/test.html
```

NOTA: LOS CONTENEDORES VIENEN CON SOLO LO IMPRESCINDIBLE INSTALADO. SI QUIERO INSTALAR ALGO DEBO NORMALMENTE HACER ANTES UN APT UPDATE (ya que la mayoría son basados en Debian).

2.6 Obtención de información de los contenedores

Conforme vayamos usando docker el número de contenedores que tenemos funcionando irá en aumento hasta que llegue un momento en que no sepamos con seguridad aspectos como si el contenedor tenía persistencia o no, si tenía redirección de puertos o no, si le había puesto algún nombre o estaba usando un nombre concedido por docker etc...

En ese contexto hay varios comandos docker que me van a ayudar a obtener información de un contenedor. En este curso vamos a usar los dos siguientes:

- La orden **docker ps**.
- La orden **docker inspect**.
- La orden **docker logs**.

DOCKER PS

La orden **docker ps** nos va a servir para obtener **información de los contenedores ya arrancados**. La información que nos proporciona va a ser **menos exhaustiva** que la que podemos obtener con **docker inspect** pero nos puede ayudar a determinar aspectos como:

- El **estado** del contenedor (Parado EXITED o Funcionado UP).
- La **imagen** de la que deriva el contenedor.
- El **tamaño** actual del contenedor.
- La orden que ejecuta el contenedor al arrancar, lo que se llama el **ENTRYPOINT** (hablaremos de ello en el capítulo 6).
- El **nombre** del contenedor, ya sea dado por nosotros o por docker.
- **Cuando fue creado** el contenedor.
- Las **redirecciones de puertos**, en caso de haberlas.

Como muchas de las órdenes de **docker ps** tiene **multitud de opciones** (flags) así que para ilustrar su uso mejor vamos a poner varios ejemplos de las más usadas.

```
# Mostrar los contenedores que están en ejecución
> docker ps
# Mostrar todos los contenedores, estén parados o en ejecución (-a o --all)
> docker ps -a
# Añadir la información del tamaño del contenedor a la información por defecto (-s o --size)
> docker ps -a -s
# Mostrar información del último contenedor que se ha creado (-l o --
```

latest). Da igual el estado

```
> docker ps -l
```

Filtrar los contenedores de acuerdo a algún criterio usando la opción (-f o --filter)

Filtrado por nombre

```
> docker ps --filter name=servidor_web
```

Filtrado por puerto. Contenedores que hacen público el puerto 8080

```
> docker ps --filter publish=8080
```

Hay mas opciones e incluso podemos formatear la salida usando el flag --format, pero con estas tenemos más que suficiente para poder empezar a trabajar con docker.

Una vez ejecutamos el comando elegido obtendremos una salida similar a la siguiente:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
5e5adf6815bc	jenkins/jenkins:lts	"/sbin/tini -- /usr/..."	4 days ago	Up About a minute	0.0.0.0:50000->50000/tcp, 0.0.0.0:9090->8080/tcp
49e4f95358ec	bitnami/laravel:7-debian-10	"/app-entrypoint.sh _..."	5 days ago	Exited (143) 3 days ago	
597f2966093b	bitnami/mariadb:10.1-debian-10	"/opt/bitnami/script..."	5 days ago	Exited (0) 3 days ago	
621fcf0ea66d	tomcat:9.0.39-jdk11	"catalina.sh run"	11 days ago	Exited (143) 37 hours ago	
6fd50086cbde	tomcat	"catalina.sh run"	12 days ago	Exited (143) 11 days ago	

Podemos ver que las filas nos muestran información de cada uno de los contenedores. El tipo de cada información vienen reflejado en las columnas y podemos apreciar que las redirecciones de puertos solo se muestran en aquellos contenedores que están en ejecución (UP).

Los contenedores parados tiene el estado(Status) **EXITED**.

DOCKER INSPECT

Si la información que hemos obtenido usando docker ps , que es una información general, no es suficiente para nuestro objetivo deberemos usar la **docker inspect** que nos va a dar una **información detallada** del contenedor que seleccione. Lo podemos hacer de las siguientes formas:

Por nombre. Por ejemplo: Mostrar información detallada del contenedor cuyo nombre es jenkins

```
> docker inspect jenkins
```

Por id. Por ejemplo: Mostrar información detallada del contenedor cuyo id es 5e5adf6815bc

```
> docker inspect 5e5adf6815bc
```

Al ejecutar esto obtendremos una imagen similar a la siguiente:

```

    "Name": "/jenkins",
    "RestartCount": 0,
    "Driver": "overlay2",
    "Platform": "linux",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "docker-default",
    "ExecIDs": null,
    "HostConfig": {
      "Binds": [
        "/home/pekechis/jenkins_home:/var/jenkins_home"
      ],
      "ContainerIDFile": "",
      "LogConfig": {
        "Type": "json-file",
        "Config": {}
      },
      "NetworkMode": "default",
      "PortBindings": {
        "50000/tcp": [
          {
            "HostIp": "",
            "HostPort": "50000"
          }
        ],
        "8080/tcp": [
          {
            "HostIp": "",
            "HostPort": "9090"
          }
        ]
      },
      "RestartPolicy": {
        "Name": "no",
        "MaximumRetryCount": 0
      },
      "AutoRemove": false,
      "VolumeDriver": "",
      "VolumesFrom": null,
      "CapAdd": null,
      "CapDrop": null,
      "Capabilities": null,
      "Dns": [],
      "DnsOptions": [],
      "DnsSearch": [],
      "ExtraHosts": null,
      "GroupAdd": null,
      "IpcMode": "private",
      "Cgroup": "",
      "Links": null,
      "OomScoreAdj": 0,
      "PidMode": "",
      "Privileged": false,

```

Esta imagen es una imagen parcial, porque se nos muestra mucha información, está en formato JSON (JavaScript Object Notation) y nos da datos sobre aspectos como:

- El **id** del contenedor.
- Los **puertos** abiertos y sus redirecciones
- Los **bind mounts** y **volúmenes** usados.
- El **tamaño del contenedor**
- La **configuración de red** del contenedor.
- El **ENTRYPOINT** que es lo que se ejecuta al hacer docker run.
- El valor de las **variables de entorno**.

- Y muchas más cosas....

Adicionalmente podemos formatear la salida usando [Go Templates](#) y el flag `--format/-f`. Una descripción detallada queda fuera de los objetivos de este curso pero vamos a poner varios ejemplos:

```
# Mostrar la ip del contenedor
> docker inspect --format 'La ip es
{{.NetworkSettings.Networks.bridge.IPAddress}}' jenkins
La ip es 172.17.0.2
# Mostrar las redirecciones de puertos del contenedor
> docker inspect --format 'Las redirecciones de puertos son
{{.NetworkSettings.Ports}}' jenkins
Las redirecciones de puertos son map[50000/tcp:[{0.0.0.0 50000}]
8080/tcp:[{0.0.0.0 9090}]]
```

NOTA: Para poder este formateo debemos conocer en profundidad la estructura del JSON que nos devuelve.

DOCKER LOGS

Los dos comandos que hemos visto anteriormente nos dan información relativa al contenedor pero no nos dan **información de lo que está pasando en el contenedor**. Para determinar este tipo de cosas siempre hemos tenido los **logs** y siguen estando disponibles aunque estemos en docker mediante el uso de la orden **docker logs**, que me va a servir tanto para contenedores que estén **parados** como para contenedores en **ejecución**.

Los podemos hacer de las siguientes formas:

```
# Por nombre. Por ejemplo: Mostrar los logs del contenedor cuyo nombre
es jenkins
> docker logs jenkins
# Por id. Por ejemplo: Mostrar los logs cuyo id es 5e5adf6815bc
> docker logs 5e5adf6815bc
```

Evidentemente los logs de una contenedor varían mucho de un contenedor a otro. No es lo mismo un log de un contenedor de base de datos que los logs de un servidor web. No obstante una posible salida si es un contenedor con jenkins seria esta:

```

Trying to load models from /var/jenkins_home/plugins/cucumber/WEB-INF/classes/models
Loading /var/jenkins_home/plugins/cucumber/WEB-INF/classes/models/cucumber-builder.rb
2020-12-06 11:02:59.132+0000 [id=117] INFO jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2020-12-06 11:02:59.146+0000 [id=106] INFO jenkins.InitReactorRunner$1#onAttained: Started all plugins
2020-12-06 11:03:00.162+0000 [id=95] INFO jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2020-12-06 11:03:00.222+0000 [id=136] INFO jenkins.InitReactorRunner$1#onAttained: System config loaded
2020-12-06 11:03:00.223+0000 [id=136] INFO jenkins.InitReactorRunner$1#onAttained: System config adapted
2020-12-06 11:03:00.315+0000 [id=62] INFO jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2020-12-06 11:03:00.315+0000 [id=68] INFO jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs updated
2020-12-06 11:03:00.401+0000 [id=160] INFO hudson.model.AsyncPeriodicWork#lambda$doRun$0: Started Download metadata
2020-12-06 11:03:00.431+0000 [id=160] INFO hudson.util.Retrier#start: Attempt #1 to do the action check updates server
2020-12-06 11:03:01.001+0000 [id=74] INFO o.s.c.s.AbstractApplicationContext#prepareRefresh: Refreshing org.springframework.web.context.support.StaticWebApplication
ext@56b25954: display name [Root WebApplicationContext]; startup date [Sun Dec 06 11:03:01 UTC 2020]; root of context hierarchy
2020-12-06 11:03:01.082+0000 [id=74] INFO o.s.c.s.AbstractApplicationContext#obtainFreshBeanFactory: Bean factory for application context [org.springframework
xt.support.StaticWebApplicationContext@56b25954]: org.springframework.beans.factory.support.DefaultListableBeanFactory@16810917
2020-12-06 11:03:01.095+0000 [id=74] INFO o.s.b.f.s.DefaultListableBeanFactory#preInstantiateSingletons: Pre-instantiating singletons in org.springframework
ry.support.DefaultListableBeanFactory@16810917: defining beans [authenticationManager]; root of factory hierarchy
2020-12-06 11:03:01.354+0000 [id=74] INFO o.s.c.s.AbstractApplicationContext#prepareRefresh: Refreshing org.springframework.web.context.support.StaticWebA
ext@24a37c95: display name [Root WebApplicationContext]; startup date [Sun Dec 06 11:03:01 UTC 2020]; root of context hierarchy
2020-12-06 11:03:01.354+0000 [id=74] INFO o.s.c.s.AbstractApplicationContext#obtainFreshBeanFactory: Bean factory for application context [org.springframework
xt.support.StaticWebApplicationContext@24a37c95]: org.springframework.beans.factory.support.DefaultListableBeanFactory@52e028b4
2020-12-06 11:03:01.355+0000 [id=74] INFO o.s.b.f.s.DefaultListableBeanFactory#preInstantiateSingletons: Pre-instantiating singletons in org.springframework
ry.support.DefaultListableBeanFactory@52e028b4: defining beans [filter,legacy]; root of factory hierarchy
2020-12-06 11:03:01.437+0000 [id=74] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
2020-12-06 11:03:01.704+0000 [id=32] INFO hudson.WebAppMain$3#run: Jenkins is fully up and running
2020-12-06 11:03:07.939+0000 [id=160] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
2020-12-06 11:03:08.073+0000 [id=160] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Ant.AntInstaller
2020-12-06 11:03:08.355+0000 [id=160] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.plugins.gradle.GradleInstaller
2020-12-06 11:03:09.323+0000 [id=160] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tools.JDKInstaller
2020-12-06 11:03:09.323+0000 [id=160] INFO hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1
2020-12-06 11:03:09.328+0000 [id=160] INFO hudson.model.AsyncPeriodicWork#lambda$doRun$0: Finished Download metadata. 8,925 ms
2020-12-06 11:32:08.269+0000 [id=180] INFO hudson.model.AsyncPeriodicWork#lambda$doRun$0: Started Periodic background build discarder
2020-12-06 11:32:08.279+0000 [id=180] INFO hudson.model.AsyncPeriodicWork#lambda$doRun$0: Finished Periodic background build discarder. 8 ms

```

Como todas las órdenes docker logs tiene más opciones más cuyo uso vamos a ilustrar con ejemplos:

Opción -f o --follow . Sigue escuchando la salida que pueden dar los logs del contenedor

> docker logs -f jenkins

Opción --tail 5. Muestra las 5 últimas líneas de los logs del contenedor en cuestión

> docker logs --tail 5 jenkins

CONSULTAR LOS LOGS ES DE ESPECIAL UTILIDAD CUANDO NUESTROS CONTENEDORES CON SERVICIOS ESTÁN FALLANDO

2.7 Gestión de contenedores

Con el paso del tiempo iremos ejecutando muchos contenedores y llegará un momento en que tengamos la necesidad de realizar operaciones como las siguientes:

- **Parar un contenedor** que no estamos necesitando o que , puede ser, esté ejecutando un servicio que ocupe un puerto que queremos ocupar con otro servicio o contenedor.
- **Eliminar un contenedor** que instalamos y que ya no necesitamos. Puede ser que ya ni nos acordemos del motivo por el cual teníamos "eso" en nuestro sistema (a mí al menos me pasa).
- Queremos **iniciar un contenedor** que estaba parado pero que vamos a volver a necesitar.
- Queremos **reiniciar un contenedor** para que nuevas opciones de configuración sean aplicadas.

Para operaciones de ese tipo tenemos las siguientes órdenes docker:

- **docker stop** para detener el contenedor, ya sea por nombre o por ID.
- **docker rm** para borrar el contenedor, ya sea por nombre o por ID.
- **docker start** iniciar un contenedor que estaba parado previamente, ya sea por nombre o por ID.
- **docker restart** para reiniciar un contenedor que previamente ya estaba en ejecución.

Cada una de ellas tiene diferentes flags u opciones. Vamos a ver las más importantes mediante ejemplos:

```
# Para un contenedor en ejecución que se llame servidorWeb
> docker stop servidorWeb
# Para un contenedor en ejecución cuyo ID es ea9b922190d8 pero
esperando 10 segundo (-t o --time)
> docker stop -t 10 ea9b922190d8
# Borrar un contenedor que se llama servidorBD
> docker rm servidorBD
# Borrado un contenedor que se llame jenkins aunque esté en ejecución (--
force o -f)
> docker rm -f jenkins
# Inicio de un contenedor con nombre jenkins
> docker start jenkins
# Inicio de un contenedor con nombre jenkins pero haciendo el attach de la
entrada estándar para poder interactuar con él (-i o --interactive)
> docker start -i jenkins
# Reinicio de un contenedor con ID ea9b922190d8
> docker restart ea9b922190d8
# Reinicio de un contenedor con ID ea9b922190d8 pero esperando 10
segundo (-t o --time)
```

```
> docker restart -t 10 ea9b922190d8
```

Hay que tener en cuenta varias cosas que si pensamos un poco ya nos indica los efectos el propio sentido común:

- Si hago **docker start** y el contenedor ya está iniciado, no pasa nada.
- Si hago **docker stop** y el contenedor ya está parado, no pasa nada.
- Si hago **docker restart** y el contenedor ya está parado, es lo mismo que si ejecutar un **docker start**.
- Pero es importante destacar que **SI UN CONTENEDOR ESTÁ EN EJECUCIÓN NO PODEMOS BORRARLO** salvo que usemos la opción **-f**. Si lo intentamos obtendremos un error similar al siguiente:

```
Error response from daemon: You cannot remove a running container d6f7b624fa8e5e8a4bab03339f05525c49659cce612953d211f55112442bb4e4. Stop the container before attempting removal or force remove
```

2.3 Ejecución de servicios. Puertos y variables de entorno

Una de las cosas que **más me gusta de usar docker** no es ya el hecho de que puedo **probar todas versiones de los distintos sistemas** que van apareciendo, es el hecho de que **PARA PROBAR Y USAR CUALQUIER SERVICIO Y CUALQUIER APLICACIÓN NO TENGO QUE INSTALAR NADA EN MI SISTEMA**, sea cual sea el servicio o la aplicación que se me ocurra, siempre la tengo en Docker Hub. Solo tengo que buscarla, averiguar cuál es la versión que quiero y lanzar el contenedor o contenedores necesarios. Y si por casualidad no está lo que yo quiero siempre podré construirlo desde cero, cosa que veremos en el módulo 6.

No es ya que al evitar la instalación evite la sobrecarga que eso supone y la cantidad de ficheros y "basurilla" que van dejando este proceso, es que el uso de contenedores de este tipo impacta de manera muy favorable en mi práctica docente en al menos dos aspectos:

- Me va a permitir **utilizar servicios y aplicaciones** que a veces, dadas la **limitada potencia de los equipos de aula** que tenemos, es difícil que podamos usar con soltura. Recordad que una de las características de los contenedores es que eran entornos de ejecución aislados (no dejan basura ninguna si los elimino) y que la sobrecarga sobre el sistema era mínima si lo comparábamos con una máquina virtual arrancando, consiguientemente, de una manera mucho más rápida.
- **Elimino** de una vez el problema que tantos quebraderos de cabeza nos da: **"EN MI EQUIPO NO FUNCIONA"**. En cualquier equipo donde se haya instalado docker de manera previa el contenedor que quiero que los demás usen va a funcionar con total seguridad.

De momento en este apartado **nos vamos a centrar en servicios de un solo contenedor**, estamos hablando de **servidores de bases de datos, servidores web, servidores de aplicaciones etc...** servicios que por otra parte son de uso casi diario en nuestras aulas. Veremos en módulos posteriores aplicaciones que requieren la interacción de más de un contenedor.

Para la ejecución de contenedores de este tipo vamos a tener que en cuenta varias cosas:

- Usar el **flag -d** para que el servicio se ejecute en **modo background o detach**. Si no lo hacemos se bloqueará el terminal mostrando el log del servicio (en ciertas ocasiones puede interesarnos) y tendremos que salir del mismo con Ctrl+C. Esto para el contenedor aunque podremos arrancarlo posteriormente.
- Si queremos que el servicio que vamos a lanzar sea accesible desde el exterior tendremos que añadir el **flag -p** de la siguiente manera **-p PUERTO_EN_HOST:PUERTO_EN_CONTENEDOR** que normalmente sería el puerto por defecto del servicio. Esto es una **REDIRECCIÓN DE PUERTOS**. Podemos tener varias reglas -p al arrancar (dependiendo del servicio será necesario) y es muy importante recordar que no podemos tener dos servicios escuchando en el mismo puerto. Si lo intentamos se nos mostrará un mensaje de error.
- **Comprobar y definir** si es necesario las **variables de entorno** que puede tener el contenedor. Las variables de entorno importantes se describen **en la página de las imágenes en DockerHub** y para usarlas tenemos que usar el **flag -e**

NOMBRE_VARIABLE=VALOR.

Para ilustrar todo esto vamos a poner varios ejemplos:

```
# Ejecuto un servidor Apache sin el flag -d ni redirección de puertos. Se
bloquea el terminal mostrando los logs y tendré que salir con Ctrl+C
> docker run httpd
AH00558: httpd: Could not reliably determine the server's fully qualified
domain name, using 172.17.0.22. Set the 'ServerName' directive globally to
suppress this message

AH00558: httpd: Could not reliably determine the server's fully qualified
domain name, using 172.17.0.22. Set the 'ServerName' directive globally to
suppress this message

[Mon Dec 07 18:27:28.561909 2020] [mpm_event:notice] [pid 1:tid
140253864719488] AH00489: Apache/2.4.46 (Unix) configured -- resuming
normal operations

[Mon Dec 07 18:27:28.562072 2020] [core:notice] [pid 1:tid
140253864719488] AH00094: Command line: 'httpd -D FOREGROUND'
# Ejecuto un servidor Apache en background y accediendo desde el
exterior a través del puerto 8888 de mi máquina.
> docker run -d -p 8888:80 httpd
# Creación de un servidor de base de datos mariadb accediendo desde el
exterior a través del puerto 3306 y estableciendo una contraseña de root
mediante una variable de
# entorno
> docker run -it -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root
mariadb
```

2.4 Asignando nombre a los contenedores

Hasta ahora cuando hemos puesto en ejecución los contenedores la propia aplicación docker ha sido la que nos ha dado un **nombre por defecto**. Estos nombres **creados aleatoriamente** por docker constan de **dos nombres aleatorios unidos por un guión bajo _**, por ejemplo: *happy_golick, magical_mclean etc..*

Evidentemente esto no es operativo. Son nombre **difíciles de recordar**, que no tienen **nada que ver con los contenedores** que queremos lanzar e **imposible de gestionar y memorizar** cuando empezamos a tener muchos contenedores en nuestro sistema.

Por este motivo es conveniente que hagamos **obligatorio el uso del flag --name** cuando usamos la orden docker run. De esta manera, si usamos **nombre elegidos por nosotros** serán **más fáciles de recordar** que los asignados por defecto. Además podemos elegir nombres que tenga **relación con la función que va a desempeñar** dicho contenedor.

Pondremos varios ejemplos:

```
# Damos el nombre de servidorBD a un contenedor de la imagen
mysql:8.0.22
> docker run -d --name servidorBD -p 3306:3306 mysql:8.0.22
# Damos el nombre de servidorWeb a un contenedor de la imagen
httpd:latest (Apache)
> docker run -d --name servidorWeb -p 80:80 httpd
```