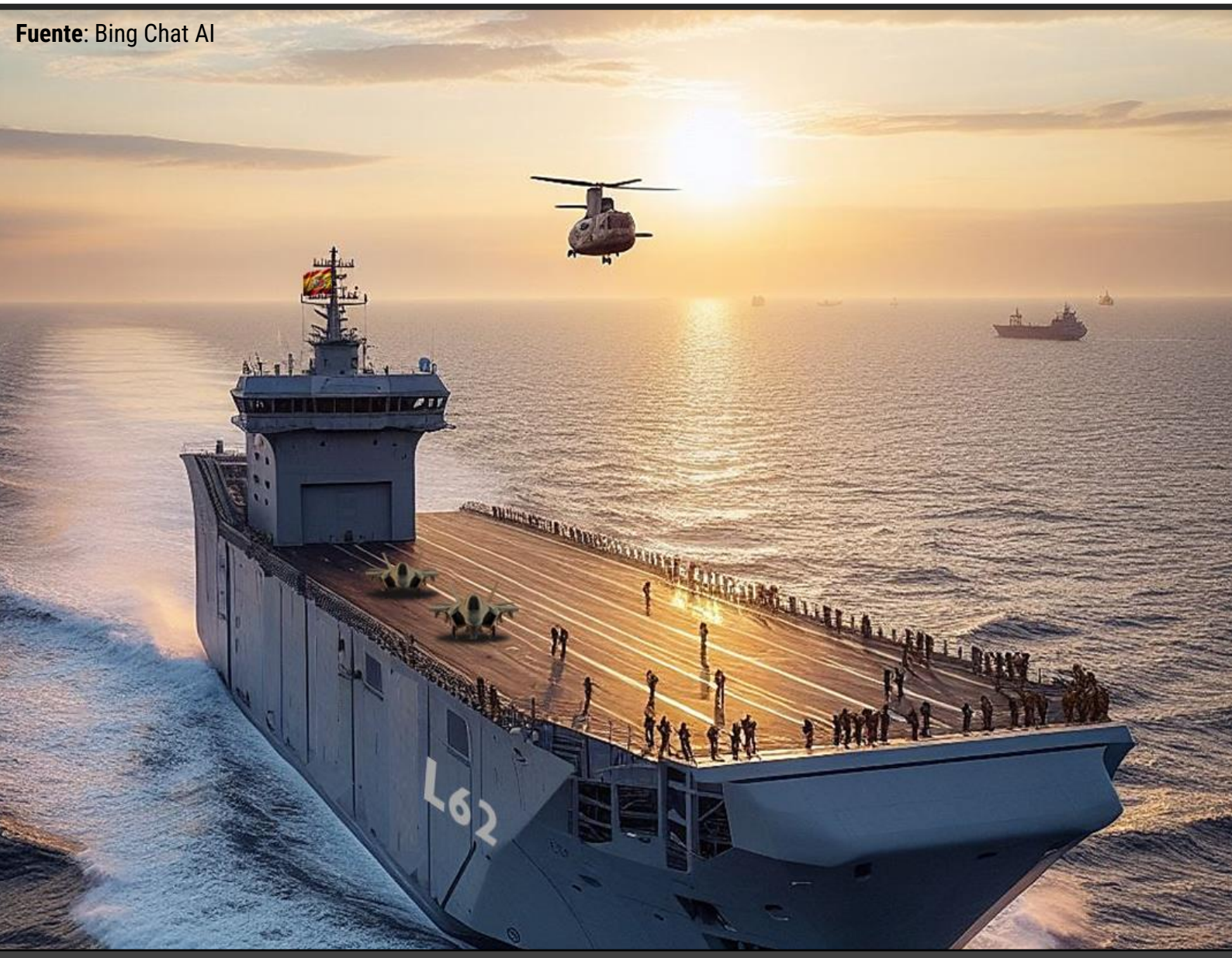
Fuente: Bing Chat AI



Bloque 3.2. Contenedores de Aplicaciones

MÁSTER EN INGENIERÍA WEB 2023 – 2024

Administración de Sistemas Operativos v1.2 *("L-62 Princesa de Asturias")*

© José Manuel Redondo López. Universidad de Oviedo

Contenido

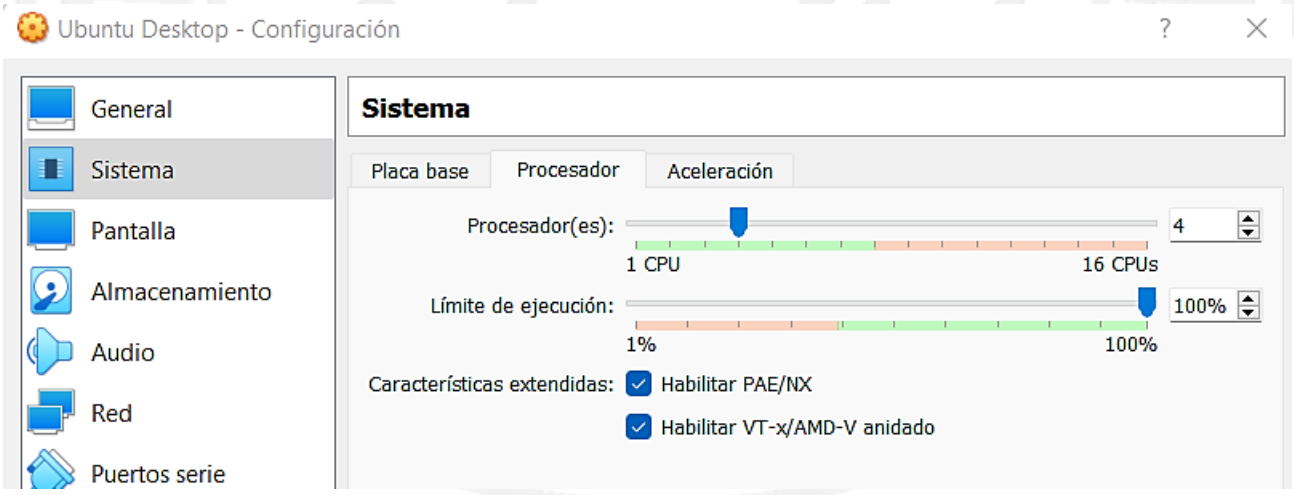
Nivel 1 (Cabo): Instalación y uso básico

¿Qué necesito para hacer este bloque de actividades?

Para realizar este bloque de actividades necesitas una máquina física o virtual, **como por ejemplo las que se puedan haber creado con el módulo de *Vagrant***. Dado que *Docker* es compatible con *Windows*, *Linux* (varias distribuciones) y *MacOS*, no debería haber ningún problema al respecto.

Si no deseas instalar ningún software en ninguno de tus PCs, este bloque puede hacerse dentro de una máquina virtual *Linux* o *Windows* sin problemas, siendo quizá la opción más recomendable usar una máquina virtual *Linux* para ello (se ha probado sobre *Ubuntu 18.04*). Si eliges este camino, necesitas:

- **Un software de virtualización instalado**: en principio sirve cualquiera que tenga la opción que se describirá en el último punto. Se recomienda *VirtualBox*, como en el caso de *Vagrant*.
- **Núcleos de CPU**: Con 2 se puede trabajar perfectamente, un extra siempre es bienvenido
- **RAM**: Mínimo 2, mejor 4 o más
- **SO**: Un *Linux* da mejor resultado que *Windows*, se ha probado sobre *Ubuntu 18.04* con éxito.
- **(Opcional, sólo se haces el ejercicio de *Docker Desktop*). Otra máquina virtual *Ubuntu 22.04* donde se active el soporte de virtualización hardware** (*VT-X/AMD-V*) anidada tal y como establezca tu programa de virtualización. En *VirtualBox* se trata de esta opción. **OJO**: No todas las CPUs soportan esta característica, por lo que, si te sale desactivada, **no podrás usar esta opción**.



Se recomienda la opción de usar una máquina virtual, si bien usar una física que admita la instalación de *Docker* y *Docker Compose* no debería dar problemas.

N1DOCKER_INSTALAR: Instalación del software para crear contenedores de aplicaciones *(0,1 puntos)*

Descripción de la actividad

Consiste en tener una **instalación operativa** de *Docker*.

Resultados esperados

El comando `docker` está instalado en el *host* y es usable

Información necesaria para su realización

Esta actividad consiste en **seguir la documentación oficial de instalación de *Docker*** para el sistema operativo base elegido. Se recomienda usar un *Ubuntu* como el mencionado en el módulo de *Vagrant*. Si esa es tu elección, hay **varias formas de instalar *Docker* en *Ubuntu*** que se contemplan aquí: https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce.

Se recomienda usar `apt` para ello, haciendo los pasos para instalar desde los repositorios (pasos 1-4) y posteriormente instalar el paquete `docker-ce`. Tras esta fase de instalación, es necesario hacer unas tareas de post-instalación:

- Siempre debemos **comprobar primero que *Docker* funciona** `sudo docker run hello-world`
- Para no ejecutar cada orden de *Docker* con `sudo`, podemos seguir los pasos para **añadir a nuestro usuario actual al grupo de usuarios autorizados a usar *Docker***. **Esto es opcional, pero altamente recomendable**.
- Si lo vamos a usar constantemente, conviene hacer el paso de **iniciarlo en el arranque**. Probablemente esto ya se haga por defecto en tu SO.
- Si no lo vamos a usar localmente, conviene hacer los pasos indicados para **habilitar el acceso remoto al mismo**, configurando también el *firewall* de forma acorde. Para esta asignatura se recomienda una instalación y uso local y, por tanto, **saltarse este paso**.
- Configurar **límites de *swap*** de la forma indicada si nos sale el error relacionado indicado en enlace anterior (normalmente no sale).
- **Configurar el DNS** usado (no es nuestro caso, al ser una instalación local).

N1DOCKER_ARRANCARCONT: Arranque y creación de un contenedor a partir de una imagen preconstruida: modos interactivo y *detached* *(0,3 puntos)*

Descripción de la actividad

Consiste en poder **arrancar un contenedor *Docker*** de una imagen predefinida y estudiar los **modos de interactuar con él**.

Resultados esperados

Eres capaz de crear y entrar en sesión de un contenedor interactivo que has creado, y también entiendes la **diferencia entre lanzarlo en modo *detached* y lo que esto implica**. Para terminar esta tarea debes lanzar un contenedor de tipo `nginx`, otro de tipo `alpine` y un tercero `distroless.dev/nginx` y comprobar las diferencias de tamaño entre los tres. Puedes contestar a estas preguntas:

- *¿Qué posible riesgo de seguridad crees que puedes tener si te equivocas al teclear el nombre de una imagen Docker?*
- *¿Cuál usarías para hostear una web a la vista de estos resultados?*

Información necesaria para su realización

El objetivo de esta tarea es **crear nuestro primer contenedor**. Para hacer esto, necesitamos **una imagen base** de la que partir. *Docker Hub* tiene, por ejemplo, una imagen con la última versión de *Ubuntu* siempre disponible, aunque **sirve cualquier *Linux*** con el que te encuentres más cómodo trabajando. Para localizarla, se puede ir a la web del *Docker Hub* (https://hub.docker.com/search?q=) o bien usar el comando `docker search ubuntu` para buscar en *Docker Hub* imágenes base con ese nombre.

Localizada la imagen que necesitamos (`ubuntu`), y **asegurándonos de que está marcada como imagen oficial (esto es extremadamente importante para no instalarnos *malware* sin querer)**, podemos hacer `docker pull ubuntu` para traerla a nuestro *host* local. No obstante, hacer `pull` tampoco es estrictamente necesario, puesto que si intentamos arrancar un contenedor con una imagen que no esté en el *host* local, ***Docker* la descargará automáticamente del *Docker Hub*** si existe una con el nombre que hemos puesto.

En cualquier caso podemos arrancar en **modo interactivo** o **modo *foreground*** con: `docker run -i -t ubuntu /bin/bash`

- `-t`: Crear una terminal
- `-i`: Mantener la terminal creada abierta
- `/bin/bash`: Proceso a ejecutar al crear en contenedor

Esta combinación de opciones es la típica usada para tener una terminal abierta sobre la que ejecutar comandos en el contenedor, y se suele abreviar con `-ti`. **Más información**: https://docs.docker.com/engine/reference/run/

Otra forma de arrancar un contenedor es el **modo *detached*** (opción `-d`), que es excluyente de las opciones `-i` y `-t`. Este modo **no es interactivo**: **ejecuta el proceso que se indique y finaliza automáticamente una**

vez dicho proceso termine. Su tiempo de vida es por tanto el del proceso arrancado dentro del contenedor. Esto lo hace adecuado para hacer una tarea automatizada *batch* (con un principio y fin determinado), pero **NO** para ejecutar servidores web, *daemons* o procesos en segundo plano: **si un proceso no se ejecuta en primer plano *Docker* considerará que no está "vivo"** y por tanto el contenedor lanzado "morirá" inmediatamente después de lanzarlo.

Para comprobar este hecho, lanza este contenedor: `docker run -d ubuntu` y haz luego `docker ps`. *¿Está el contenedor lanzado operativo?* Cabe decir que **ciertos procesos te permiten elegir si se ejecutan como *daemon* o en primer plano**, lo cual es muy útil en estos casos. Por ejemplo, *Nginx* y *Apache* se ejecutan por defecto como *daemon*, pero tienen formas de arrancarse en primer plano que podrían usarse junto al modo *detached* si estuvieran instalados en el contenedor que queremos arrancar (no es el caso):

- ***Nginx***: `docker run -d <nombre de la imagen> nginx -g 'daemon off;'`
- ***Apache***: `docker run –d <nombre de la imagen> /usr/sbin/apache2ctl –D FOREGROUND`

N1DOCKER_INTERAC: Conexión e interacción con un contenedor

(0,2 puntos)

Descripción de la actividad

Consiste en aprender a cómo **interactuar con el *daemon* de *Docker*** usando las opciones más comunes para ello.

Resultados esperados

Eres capaz de crear un contenedor con nombre y puedes ver que se ha creado, sus **características principales** (incluida una IP para acceder a él desde el host), la **imagen** a partir de la cual se creó y **reiniciarlo** correctamente, así como **entrar o salir de él** sin "matarlo" necesariamente.

Información necesaria para su realización

A la hora de interactuar con un contenedor **podemos usar su ID o su nombre para referirnos a él**. Podemos ver ambos al ejecutar `docker ps`. No obstante, el identificador es aleatorio y, por defecto, ***Docker* asigna un nombre también aleatorio** compuesto de dos palabras separadas por `_` (Ej.: `focused_shockley`) a cada nuevo contenedor.

Por ello, muchas veces localizar el contenedor que queremos es una labor más difícil de lo que debería y **es mejor asignarles nombre** a los contenedores con el parámetro `--name <nombre sin espacios>` al hacer `docker run`. Con él podemos identificar de forma clara cuál es el propósito de cada contenedor (Ej.: `frontend1`, `backend2`). También podemos usar el nombre de una máquina en cualquier lugar en vez de su ID: **ambos son intercambiables** para todos los comandos de *Docker*.

Dada esta información, esta actividad consiste en arrancar un contenedor **con el nombre que desees** y probar los siguientes comandos, comprobando que se obtienen resultados correctos:

- `docker ps`: Lista los contenedores arrancados

- `docker inspect`: Ofrece un listado de la configuración del contenedor cuya ID o nombre de le pasa, incluyendo la IP que se puede usar para conectarse a él desde el *host*
- `docker images`: Lista las imágenes disponibles localmente para arrancar contenedores
- `docker restart`: Reinicia un contenedor, con temporización opcional

En esta actividad también se trata de probar **formas de interaccionar con un contenedor *foreground*** arrancado una vez estamos dentro del mismo, que no necesariamente tienen que destruirlo:

- Si tecleamos `exit` o Ctrl+D, lo finalizaremos (apagado definitivo)
- Si hacemos Ctrl+C saldremos del contenedor, pero este no finalizará (como salir de sesión)
- Podemos "reconectarnos" a dicho contenedor con `docker attach <id o nombre del contenedor>`

N1DOCKER_DETACH: Crear un contenedor *detached* con un servidor web *(0,2 puntos)*

Descripción de la actividad

Consiste en **crear un contenedor *detached* correctamente** y que quede operativo con el servicio indicado.

Resultados esperados

Eres capaz de crear un contenedor con nombre *detached* que tiene un servidor *Apache 2* y otro con un servidor *Nginx* a partir de las imágenes oficiales del *Docker Hub* de estos y acceder de forma interactiva al mismo. Además puedes contestar a la siguiente pregunta *¿te parece esta opción más "económica" en recursos que usar una máquina virtual?*

Información necesaria para su realización

Hasta ahora hemos visto que los contenedores que hemos creado son básicamente terminales interactivas para hacer pruebas, pero que, a pesar de que hemos dicho en la teoría que son contenedores de aplicaciones, no se ha lanzado ninguna aplicación dentro de ellos que no sea el propio *bash*.

Para ver más opciones, vamos a lanzar como ejemplo dos contenedores oficiales que se ofrecen en el *Docker Hub* con los servidores ***Apache*** (también llamado HTTPD) (https://hub.docker.com/_/httpd) y ***Nginx*** (https://hub.docker.com/_/nginx). Para ello se deben usar las instrucciones vistas y hacerlo de forma *detached*. Posteriormente, accede a la web por defecto que ofrecen desde el *host* averiguando primero la IP que se puede usar para ello.

Finalmente, aunque hayas lanzado un contenedor ***detached*** no significa que no puedas acceder de forma interactiva si es necesario. Para **obtener una terminal** que nos permita interactuar con él se puede hacer: `docker exec -it <id o nombre del contenedor> /bin/bash`. Para salir de este terminal creado con `exec` se puede pulsar Ctrl + p + Ctrl + q

⚒️ N1DOCKER_PARAR: Parada y limpieza de contenedores e imágenes *(0,1 puntos)*

🥷 Descripción de la actividad

Consiste en aprender a hacer una **parada y borrado ordenado** y adecuado de contenedores *Docker*.

🏅 Resultados esperados

Eres capaz de destruir un contenedor activo y también todas las imágenes y contenedores creados en un sistema en un momento dado. Puedes contestar a esta pregunta *¿Qué es más adecuado, borrar primero los contenedores y luego las imágenes o al revés?*

📖 Información necesaria para su realización

El manejo de contenedores puede dejar en nuestro sistema una "huella" muy grande, en el sentido de que **podemos ocupar muchos Mb de espacio en contendores** que quedan funcionando de pruebas anteriores y nos olvidamos de apagar cuando ya no son útiles. En esta actividad se trata de usar alguno de los contenedores anteriores para probar estos comandos:

- `docker kill`: Envía la señal `SIGKILL` a un contenedor (o la especificada como parámetro) para finalizarlo. No olvides que los contenedores son realmente procesos.
- `docker stop`: Para un contenedor mandándole las señales `SIGTERM` y posteriormente `SIGKILL`

Sin embargo, el problema a veces es que tenemos una gran cantidad de contenedores o imágenes inservibles y **borrarlas una a una no es operativo**. Por tanto en esta actividad también se trata de probar estos comandos que **hacen desaparecer todos los contenedores e imágenes de un sistema dado** (hacer "limpieza", aunque de forma bastante drástica 😊). Comprueba luego la diferencia de espacio libre en la máquina tras hacer esta operación.

- Borrar todas las imágenes *Docker*: `docker rmi $(docker images -q)`
- Borrar todos los contenedores *Docker*: `docker rm -f $(docker ps -a -q)`

Nivel 2 (Sargento): Resolución de problemas sencillos frecuentes

N2DOCKER_CUSTOMCONT: Creación de un contenedor personalizado *(0,2 puntos)*

Descripción de la actividad

Consiste en **crear contenedores con el conjunto de *software* que quieras** a partir de una imagen que crees con dicho software instalado para los mismos.

Resultados esperados

Eres capaz de crear una imagen personalizada a tu gusto para la función que quieras, a base de **instalar el *software* necesario en un contenedor existente** y "empaquetarlo" luego en una imagen que puedes usar para crear contenedores, a la que **le des un nombre personalizado**.

Información necesaria para su realización

En esta actividad vamos a ilustrar el uso de *Docker* de forma práctica mediante la creación de una imagen *Ubuntu* personalizada de la que luego crearemos contenedores. Para practicar, en lugar de hacer la operación habitual de instalar una sola aplicación en un contenedor vamos a instalar varias, tratándolo como si fuera algo similar a una máquina virtual *Linux*. Ten en cuenta que **esta NO es la forma habitual de usar contenedores de aplicaciones**, ya que esto es más habitual en contenedores de sistemas operativos. No obstante, es una forma válida de practicar con los mismos. Para ello, instala en un contenedor existente todo esto:

- Dejar el *Ubuntu* actualizado (`apt update`, `apt full-upgrade`).
- Instalar *Apache 2* (`apt install apache2`).
- El paquete `net-tools` (para `ifconfig`).
- Los paquetes `nano`, `curl` y `nmap` (para usar las órdenes correspondientes).
- Instalar un servidor `ssh` para poder conectarse a él por esa vía.
- Crear un usuario que se llame como tu UO y el paquete `sudo`. Hacer posteriormente que dicho usuario sea ***sudoer*** (pertenezca al grupo adecuado).

Hecho esto ya tenemos el contenedor configurado como queremos y podemos convertirlo a una imagen de la que luego podremos hacer más contenedores. Para ello:

- **Cambiamos de terminal en el *host*** (Ctrl + Alt + F1…F7) o simplemente abrimos una nueva ventana de terminal si estamos usando un GUI.

- Ejecutamos `docker ps` y **apuntamos el id del contendor que estamos configurando** (algo como `01d0d773e3e8`) o su **nombre** (algo como "`elated_gould`")
- Ejecutamos `docker commit -m "Apache en Ubuntu" -a "Redondo" <id o nombre del contenedor del paso anterior> miw/apache_ubuntu:v1`. El significado de los parámetros es el siguiente:
  - `commit`: Crea una imagen a partir de un contenedor en marcha, considerando todos los cambios que se le hayan hecho respecto a su imagen base
  - `-m`: Es un simple mensaje a mostrar.
  - `-a`: Nombre del autor de la imagen.
  - `miw/apache_ubuntu:v1`: Es el **nombre de la nueva imagen**, compuesto de varias partes:
    - Ruta dentro del repositorio, opcional (`miw` en el ejemplo)
    - Nombre de la imagen (en este caso `apache_ubuntu`)
    - Versión de la imagen, opcional (en este caso `v1`)

  Obviamente tanto la descripción (`-m`) como el autor (`-a`) como el nombre de la imagen final (`miw/apache_ubuntu:v1`) podéis ponerlo a vuestro gusto. **Más información**: https://docs.docker.com/engine/reference/commandline/commit/
- **Ejecutamos finalmente** `docker images` desde la terminal del *host* para comprobar que hay una nueva imagen disponible con ese nombre. Podemos entonces salir del contenedor en marcha (`exit` o Ctrl+D).
- Ahora solo tenemos ahora que usar los conocimientos vistos para arrancar un nuevo contenedor a partir de la misma y completar la operación.

N2DOCKER_SSH: Conexión vía SSH a un contenedor *(0,1 puntos)*

Descripción de la actividad

Consiste en **conectarse a un contenedor *Docker* vía SSH** y entender cómo funcionan los servicios que pudiera tener arrancados.

Resultados esperados

Eres capaz de entender cómo se comportan los servicios instalados dentro de un contenedor y lo que tienes que hacer para usarlos. Puedes contestar a esta pregunta: *¿Por qué crees que en los contenedores* `apache` *y* `nginx` *de antes no hizo falta hacer nada de esto?*

Información necesaria para su realización

En el contenedor anterior hemos instalado un servidor `ssh`, por lo que al convertirlo a imagen y luego crear contenedores a partir de ella, lo esperable es que podamos conectarnos al mismo vía `ssh`. Aunque *Docker*

proporciona una forma propia de tener una terminal en un contenedor, conectarse vía `ssh` es necesario para otros servicios (`scp`, el *Ansible* del bloque 4…), por lo que no es raro tener que instalarlo. Sin embargo, si usamos la IP que vimos en el nivel anterior para intentar conectarnos a él desde cualquier cliente SSH **comprobaremos que no es posible**. ¿Qué ha pasado?

Lo que ha pasado es **que este contenedor lo tenemos que arrancar en modo interactivo** para que quede "vivo", porque hacerlo del otro modo hará que "muera" enseguida al no tener ningún servicio en primer plano preparado para funcionar en el arranque (salvo que lo indiquemos). Es decir, ninguno de los servicios que hemos instalado arrancará automáticamente.

En otras palabras, tenemos un servidor `ssh` y un *Apache* instalados listos para funcionar, pero no están funcionando en este momento porque **nadie les ha dicho que lo hagan**. Comprueba esto entrando en el contenedor de forma interactiva y ejecutando el comando `service -status-all` (los servicios arrancados aparecen con `[+]` y los apagados con `[-]`).

Por tanto, para que el `ssh` (y cualquier otro servicio) funcione hay que arrancarlos manualmente como lo haríamos en un *Linux* normal (Ej.: `service ssh start`). Hazlo y comprueba que ahora la conexión es posible vía `ssh`. Reinicia el contenedor y vuelve a intentarlo. *¿Qué ha pasado? ¿Qué ocurre ahora si vuelves a arrancar el `ssh`?*

⚒ N2DOCKER_SERVICIOS: Exposición de servicios de un contenedor al exterior *(0,1 puntos)*

Descripción de la actividad

Consiste en aprender a **exportar un servicio que se ejecuta dentro de un contenedor** para que pueda usarse desde el exterior de este.

Resultados esperados

Eres capaz de exportar un servicio que se encuentra arrancado dentro de un contenedor para que lo usen máquinas que se encuentren fuera del *host*

Información necesaria para su realización

Hasta ahora hemos visto que los contenedores son accesibles desde el propio *host* y las particularidades de sus servicios, pero estaréis de acuerdo en que esta forma de funcionar no es precisamente la óptima porque:

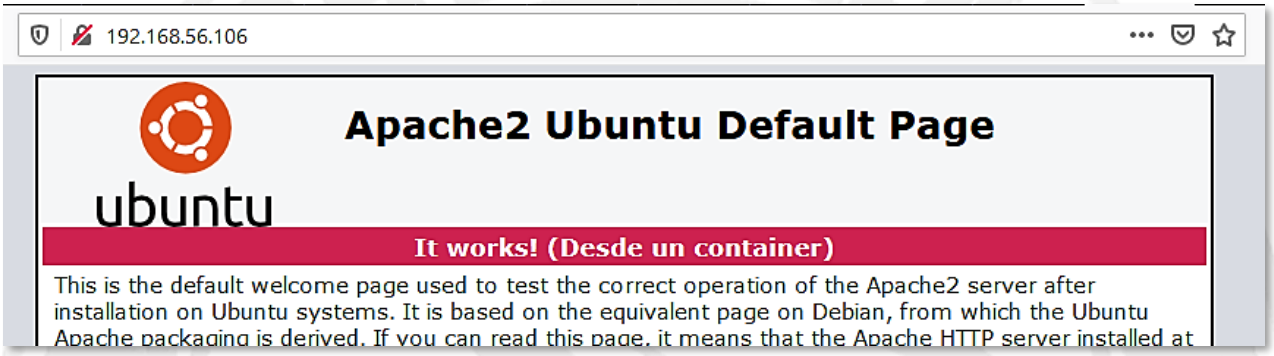
1. Arrancar algo dentro de un contenedor debería dar menos problemas.
2. **Nadie fuera del host podrá acceder al servicio ofrecido por el contenedor**.

Vamos a tratar de solventar ambos problemas en una sola operación usando y entendiendo esta orden: `docker run -d -p 192.168.56.106:80:80 miw/apache_ubuntu:v1 /usr/sbin/apache2ctl -D FOREGROUND`. El comando consta de los siguientes elementos:

- `-p <IP del host>:<puerto del host>:<puerto del contenedor al que se mapea el del host>`. Aquí la IP de nuestra máquina es la `192.168.56.106` y estamos redirigiendo las conexiones

al puerto `80` de la máquina al puerto `80` del contenedor: El acceso HTTP (puerto `80`) sobre la IP `192.168.56.106` del *host* realmente accederá al puerto `80` del contenedor que acabamos de arrancar.

- Podemos usar tantos `-p` con combinaciones de IPs y puertos como queramos.
- La máquina del ejemplo tiene una interfaz *host-only* de *VirtualBox* en la red `192.168.56.X`. Arrancado este contenedor, el `apache2` instalado en él será accesible desde el *host* (*Windows 10* en las pruebas realizadas para este curso) si se accede a esa IP dese un navegador. Podemos así arrancar tantos contenedores *Apache 2* como queramos y de esta forma crear un "clúster" de manera sencilla y mucho más económica en recursos. Si todo va bien, deberíamos ver algo como esto:



N2DOCKER_WEB: Infraestructura para web y base de datos con contenedores *(0,3 puntos)*

Descripción de la actividad

Consiste en **crear una infraestructura básica para hospedar una web** y su BBDD con contenedores Docker.

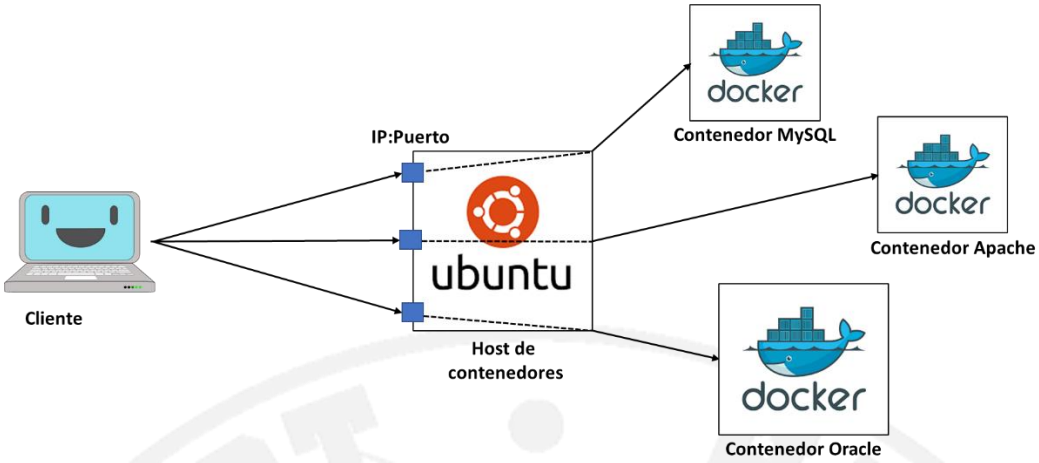
Resultados esperados

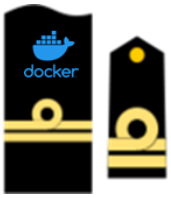
Eres capaz de arrancar varios contenedores distintos, configurarlos y exportar algunos de sus servicios a través de puertos del *host*

Información necesaria para su realización

Ahora que sabemos arrancar servicios y exportarlos a distintos puertos, se pide **reproducir la infraestructura de la imagen** cumpliendo con los siguientes criterios:

- Contenedor oficial `mysql` exportado en el puerto `3306` (puerto por defecto de *MySQL*)
- Contenedor oficial `apache` exportando el puerto `8080` (a través del que se accedería al puerto `80` del contenedor)
- Contenedor oficial `oraclelinux` en el que se instale un servidor `ssh` que se exponga al exterior a través del puerto `10022` del host


docker
Contenedor MySQL
IP:Puerto
docker
Contenedor Apache
ubuntu
Cliente
Host de
contenedores
docker
Contenedor Oracle



Nivel 3 (Teniente de navío): Resolución de situaciones más avanzadas

N3DOCKER_DOCKERFILES: Uso básico de *Dockerfiles* *(0,1 puntos)*

Descripción de la actividad

Consiste en **aprender a usar *Dockerfiles*** con las opciones más comunes para crear imágenes *Docker* nuevas.

Resultados esperados

Eres capaz de crear una imagen de forma automatizada a partir de una `Dockerfile` que contenga instrucciones que instalen el *software* que necesitas en la misma

Información necesaria para su realización

Esta actividad pretende que te familiarices con el uso práctico de *Dockerfiles* para la **construcción de imágenes personalizadas** a partir de ficheros de texto de manera automatizada. Para probarlo, creamos un directorio y dentro del mismo creamos fichero llamado `Dockerfile` con un equivalente a la creación de la imagen personalizada anterior. Para facilitar su creación, se puede hacer completando el ejemplo de esta imagen, que **también se da como fichero adjunto de la teoría**.

```
FROM ubuntu
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update
RUN apt -y dist-upgrade
RUN apt -y install apache2
RUN apt -y install net-tools
RUN apt -y install nano
RUN apt -y install curl
RUN apt -y install nmap
RUN echo 'Imagen construida con una Dockerfile' \
        > /var/www/html/index.html
EXPOSE 80
```

Por tanto, debes analiza las instrucciones que se usan y completar lo que falta para que la imagen creada sea equivalente a la creada “a mano” anteriormente.

- `FROM`: La **imagen base**, a partir de la cual trabajar (normalmente una del *Docker Hub*, pero por favor asegúrate de que sea una oficial).
- `RUN`: Cada acción a ejecutar para construir la imagen final.
- `EXPOSE`: Indica el puerto a partir del cual va a estar disponible un servicio de esta imagen.
- **Ninguna instrucción debe hacer preguntas al usuario**: En caso contrario, el proceso se interrumpirá (de ahí que se haya usado la opción `-y`, y la variable de entorno `DEBIAN_FRONTEND=noninteractive`). **Acostúmbrate a incluirlas** en las *Dockerfiles* que crees por si acaso.

Una vez creado el `Dockerfile`, este puede procesarse con `docker build`. Además, debe especificarse con `-t` cómo se llamará la imagen a construir. Por defecto se busca un fichero llamado `Dockerfile` en la ruta actual (aunque puede especificarse otro fichero). Si la construcción termina correctamente, **veremos una imagen nueva en el listado de imágenes de *Docker***, que podemos usar igual que las que hemos usado anteriormente. Para finalizar la actividad se pide **arrancar un contenedor de la nueva imagen** creada para comprobar que funciona.

N3DOCKER_COPIARFICH: Copia de ficheros a contenedores *(0,1 puntos)*

Descripción de la actividad

Consiste en **aprender a incorporar ficheros desde el *host*** a cualquiera de los contenedores que se vayan a arrancar.

Resultados esperados

Eres capaz de incorporar ficheros a un contenedor, por ejemplo una web a servir en un contenedor que tenga un servidor web de los que hemos visto anteriormente.

Información necesaria para su realización

Para configurar un contenedor no solo es necesario instalar programas, sino también **incorporarle ficheros con distintos contenidos** (configuraciones, datos, una web, etc.). Esto puede hacerse usando la instrucción de las *Dockerfiles* `COPY` (mejor opción normalmente que `ADD`, por recomendación del propio manual de *Docker*, mira la sección "*ADD or COPY*" de https://docs.docker.com/develop/develop-images/dockerfile_best-practices/).

Por ejemplo, si en la propia carpeta donde tenemos la `Dockerfile` tenemos los ficheros `tmux.conf` y `nanorc`, estas dos instrucciones copiarán el primero a la carpeta `/etc/` del contenedor y el segundo a la carpeta del usuario `root`.

```
COPY tmux.conf /etc/tmux.conf
COPY nanorc /root/.nanorc
```

La instrucción `COPY` también se puede usar **para copiar carpetas completas** si la ruta apunta a una. Si no queremos copiar todo el contenido de una carpeta, se puede usar un fichero `.dockerignore` dentro de ella para decir qué no se quiere copiar.

Con toda esta información, se pide copiar una web de ejemplo que tengamos en el *host* a un contenedor que tenga dentro un servidor web **de forma que pueda servirla de manera inmediata**, considerando que los servidores web necesitan las webs que sirven en la ruta `/var/www/html`.

Finalmente, aunque `COPY` es preferible a `ADD`, esta última aún tiene sus usos, sobre todo con ficheros `.tar` comprimidos, que podemos consultar aquí si tenemos curiosidad: https://phoenixnap.com/kb/docker-add-vs-copy

N3DOCKER_VOLUMEN: Creación de volúmenes en contenedores

(0,2 puntos)

Descripción de la actividad

Consiste en aprender a **crear e incorporar volúmenes de almacenamiento** a contenedores *Docker*.

Resultados esperados

Eres capaz de arrancar un contenedor que tiene dentro un servidor web y hacer que sirva una web sacada de una carpeta del *host* **haciendo uso de la creación de volúmenes de los contenedores *Docker***. Hecho esto, se pide:

1. Entrar en sesión interactiva en el contenedor y escribir algo en la carpeta a la que apunta el volumen
2. Parar y arrancar luego el contenedor para comprobar que queda guardado en el host y permanece ahí en sucesivas ejecuciones
3. Apagar ese contenedor y borrar el volumen creado.

Información necesaria para su realización

Hemos visto que los volúmenes de *Docker* son la forma de tener un **espacio de almacenamiento persistente**, de manera que la información que podamos modificar dentro de un contenedor *Docker* pueda "sobrevivir" a rearranques de este si se guarda en uno de estos volúmenes.

Para añadir un volumen de datos a un contenedor *Docker* hay que usar el *flag* `-v`, admitiéndolo tanto el comando `docker create` como el `docker run`. También se pueden usar múltiples `-v` para montar varios volúmenes.

Un ejemplo de este comando sería: `docker run -d -P --name web -v /src/web:/var/www/html miw/apache2 /usr/sbin/apache2ctl -D FOREGROUND`. Con esto montamos el directorio del *host* `/src/web` en la ruta por defecto para webs del *Apache 2* del contenedor, consiguiendo así que el *Apache* sirviese esa web desde el contenedor aunque sus ficheros estén en el host. Hay una serie de diferencias respecto al `COPY` visto antes:

- **No hay copia física de ficheros**: si cambiamos los ficheros en el *host* el contenedor verá esos cambios inmediatamente. Esto es bueno o malo, según el contexto 😊
- Si ya existiese el contenido en la ruta del contenedor **no se borra**, el montado **lo reemplaza**. Si el volumen se desmonta, el contenido existente vuelve a aparecer. Se comporta pues como el `mount` de *Linux*.

- `docker inspect <nombre de contenedor>` muestra también los volúmenes que tiene asociados un contenedor concreto.

Finalmente, al igual que vimos antes con la creación de contenedores e imágenes, es posible que nuestras pruebas nos dejen una gran cantidad de volúmenes sin usar en nuestro sistema que luego queramos borrar. Además de usar `docker volume rm <id de volumen>` para borrar un volumen concreto, podemos también **borrarlos todos** con `docker volume rm $(docker volume ls -q)` o `sudo docker volume prune`

N3DOCKER_DCOMPOSE: Uso básico de *Docker Compose* *(0,3 puntos)*

Descripción de la actividad

Consiste en **aprender a hacer un uso básico de *Docker Compose*** para aprender a arrancar contenedores ya creados de manera coordinada.

Resultados esperados

Eres capaz de arrancar varios contenedores distintos, configurarlos y exportar algunos de sus servicios a través de puertos del host usando *Docker Compose*

Información necesaria para su realización

Como vimos en teoría, *Docker Compose* es una herramienta ideal para **arrancar infraestructuras de aplicaciones** que estén en varios contenedores de forma coordinada. Un fichero YAML de *Docker Compose* está compuesto por los siguientes elementos vistos en teoría, que aparecen en la siguiente imagen:

```yaml
version: "2"
services:
    ubuntu_crypto:
        image: ssi/ubuntu_ssh_crypto
        container_name: ubuntu_crypto
        restart: on-failure
        volumes:
            - ./volume_data/ubuntu:/home/ssiuser/shared
    kali_crypto:
        image: ssi/kali_for_crypto
        container_name: kali_crypto
        restart: on-failure
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        volumes:
            - ./volume_data/kali:/root/shared
```

- La **versión** (`version`) de *Docker Compose* que admite ese fichero. Podemos dejarlo siempre en **"2"**
- Una lista de **servicios** (`services`, **contenedores creados a partir de imágenes**) a arrancar
- **Para cada entrada** de la lista de servicios necesitamos además:
    - Un **nombre** que identifique **de forma única** a la entrada (por ejemplo, `ubuntu_crypto` en la imagen)
    - La **imagen** a partir de la cual se construirá un contenedor (`image`). La imagen **DEBE** estar ya construida previamente, puesto que por defecto *Docker Compose* no lo hará **salvo que usemos la opción** `build` (no en este caso).
    - Un **nombre para el contenedor** que se arrancará (`container_name`): En el ejemplo coincide con el nombre de la entrada, **pero no es obligatorio.**
    - **Si se reinician los contenedores en caso de fallo** (`restart: on-failure`), lo cual es muy útil para evitar caídas accidentales de un servicio.
    - Si se admite o no **consola interactiva en el contenedor** (añadiendo `stdin_open` y `tty` y poniéndolos a `true`, ya que, si no se añaden, son `false` por defecto).
    - Los **volúmenes** añadidos a cada contenedor, si los hubiera. En el ejemplo se mapea una carpeta del host (subdirectorio `/volume_data/<nombre del SO>` de la carpeta actual) a una carpeta del contenedor. Pueden crearse tantos como sea necesario
    - Si un contenedor **expone o no puertos** en el *host*, añadiendo en su entrada una sección `ports` como esta:

```
    ports:
      - "8000:8000"
```

Con esta información y lo visto anteriormente, se pide **que se cree con *Docker Compose* una infraestructura que despliegue un equivalente a la que se vio al final del Nivel 2** (la del dibujo), con el añadido de que la máquina *Apache 2* sirve una web que se encuentra en una carpeta del *host*.

Nivel 4 (Capitán de navío): Explotación de Contenedores de Aplicaciones

N4DOCKER_DOCKERFILEADV: Uso avanzado de *Dockerfiles* *(0,4 puntos)*

Descripción de la actividad

Consiste en aprender a **crear *Dockerfiles* que usen imágenes ya construidas de forma incremental**, para crear así nuevas imágenes que solo se diferencien de su "padre" en el nuevo software que instalan, reutilizando el resto.

Resultados esperados

Esta actividad se terminará cuando seas capaz de crear nuevas imágenes "incrementales" a partir de otras existentes, de manera que **minimices la cantidad de operaciones comunes que se hagan entre ellas** y las ya existentes, y por tanto **saques el mayor partido posible de la caché de *Docker*** mientras creas *Dockerfiles* más fácilmente entendibles. Puedes contestar a estas preguntas

- *¿Son las imágenes "intermedias" también usables?*
- *¿Cuáles son para ti las principales ventajas que tiene trabajar así frente a crear las imágenes con todo el software necesario a partir de una con solo el SO del Docker Hub?*

Información necesaria para su realización

Hasta ahora hemos hecho contenedores con *Dockerfiles* de forma que en la `Dockerfile` se instalaba todo lo necesario para que el contenedor arranque. Esto es correcto, pero muchas veces **necesitamos crear varios tipos de contenedores que tienen una buena parte de su *software* y configuración en común**: actualizaciones, instalación de programas, etc.

Si seguimos una aproximación "tradicional" a la hora de construirlas, esto nos obliga a **repetir muchas veces las mismas instrucciones en distintas *Dockerfiles***, cosa que nos lo podemos ahorrar si construimos contenedores de forma escalonada o **incremental**.

Esto significa que **construiremos una imagen "base"** con una serie de *software*, configuración y operaciones comunes, y el resto de las imágenes que construiremos dependerán de dicha imagen base. De esta forma, no se repetirán el trabajo realizado al construirla, porque ya se habrá hecho al construir la base. **Es algo parecido a la refactorización *Extract Superclass*** que podemos ver en lenguajes de programación. Para practicar con esta idea, esta actividad plantea la implementación de lo dicho en el apartado de "***Imágenes enlazadas***" de la teoría haciendo lo siguiente.

Crear la imagen "base"

Para crear una `Dockerfile` que aglutine el software, configuración, etc. que luego tendrán todas las imágenes siguientes, sigue estos pasos:

- Usar la técnica para **evitar que la instalación de un paquete pregunte algo al usuario** añadiendo esto al principio de la *Dockerfile*: `ENV DEBIAN_FRONTEND=noninteractive`
- Vamos a permitir **usar la consola de forma interactiva**, por lo que es más oportuno permitir usar colores, y para ello debemos añadir esta otra variable de entorno al principio de la *Dockerfile*: `ENV TERM=xterm-color`
- **Actualizar** la lista de paquetes (`apt-get update`)
- **Instalar los siguientes paquetes** comunes para interaccionar con la consola: `apt-utils curl iputils-ping nano net-tools wget less tmux gpm mc`
- **Copiar los ficheros** `tmux.conf` y `nanorc` proporcionados en `/etc/` y en el directorio `root` del contenedor respectivamente (escribiéndola esta última en un fichero `.nanorc`) para introducir una configuración avanzada para el editor `nano` y el uso de varias terminales simultaneas (`tmux`)
- Puedes además **etiquetar la imagen construida** poniendo un nombre de autor, una versión y una descripción de esta forma, con la instrucción `LABEL`:

```
LABEL  maintainer="<tu nombre>" \
       version="1.0" \
       description="<algún texto descriptivo>"
```

- **Construimos ahora la imagen** como ya hemos visto y le damos un nombre, por ejemplo `ubuntu_base`.

Construir una imagen derivada de la base

Ahora que ya tenemos la imagen base, podemos usarla para construir la siguiente imagen de la secuencia en otra `Dockerfile` distinta, a la que llamaremos `ubuntu_ssh`, con las siguientes características:

- **Se crea a partir de la** `ubuntu_base` anterior, por lo que tenemos que poner su nombre en el `FROM`
- **No se repite nada** que hayamos puesto ya en la imagen base (¡es la gracia de esta técnica!)
- Hay que **actualizar la lista de paquetes de nuevo** (`apt-get update`) para asegurarnos de que no hay problemas a la hora de resolver las rutas de estos
    - La imagen base, que fue la que hizo la última actualización de paquetes, puede haberse construido hace tiempo y esas rutas, cacheadas de la última vez que se hizo `apt-get update`, pueden haber cambiado, dando errores de instalación.
- **Se instalan los paquetes** `openssh-server` y `openssl`
- Se creará un usuario **no sudoer** nuevo para practicar con la creación de variables en una *Dockerfile* con `ARG`. Usar este ejemplo como base para hacer esta operación:

```
ARG SSH_USER=ssiuser
ARG SSH_USER_PASS=test123...
RUN useradd -m -s /bin/bash -p $(openssl passwd -1 $SSH_USER_PASS) $SSH_USER
```

- Se **expone el puerto** SSH (`22`) con `EXPOSE`
- Puedes **volver a etiquetar la imagen de nuevo** con `LABEL` para adaptarla al nuevo contenido.
- Prueba a construir esta imagen y mira como ahora **sólo se construye la parte nueva** y no lo correspondiente a la imagen base, maximizando el uso de la caché de *Docker*.

Construir una imagen derivada de la derivada 😉

Ahora que tenemos ya dos imágenes creadas ("padre" e "hijo"), finalmente podemos construir una tercera imagen `ubuntu_ssh_apache` **a partir de esta segunda**, en una tercera `Dockerfile` distinta a las anteriores y con las siguientes características:

- **Vuelve a actualizar la lista de paquetes** (`apt-get update`) por la razón mencionada anteriormente
- Al ser una **imagen final** a partir de la cual se van a construir contenedores en nuestro caso, conviene hacer una actualización completa de paquetes en ella (`apt-get full-upgrade`). De esta manera no se ejecuta este proceso tan costoso en tiempo en cada nivel a construir y **no se ejecuta *software* sin actualizar** (crítico por temas de seguridad).
- **Instala** el paquete `apache2`
- **Expón** el puerto `80` (`EXPOSE`)
- **Vuelve a describir** la imagen construida con `LABEL`

Una vez hecho esto, se pide entrar en un contenedor creado de la última imagen de manera interactiva y comprobar que funciona adecuadamente, comprobando que tienes todos los servicios esperados (SSH y *Apache 2*).

⚒️ N4DOCKER_MULTISERV: Arranque de varios servicios *(0,1 puntos)*

🕵️ Descripción de la actividad

Consiste en aprender a **arrancar varios servicios en un mismo contenedor *Docker***.

🏅 Resultados esperados

Eres capaz de crear y ejecutar un *script* que permita inicializar cualquier servicio dentro del contenedor, incluso más de uno, y que el contenedor no muera tras hacerlo.

📖 Información necesaria para su realización

Al final del ejercicio anterior hemos visto que creábamos una máquina con dos servicios instalados, SSH y *Apache*. Sin embargo, **no se habían arrancado y teníamos que hacerlo a mano**. Esta actividad te enseña una forma de arrancar varios servicios en la inicialización de un contenedor, que da muy pocos problemas y permite que el contenedor **no se "muera" prematuramente** aunque dichos servicios funcionen en segundo plano por defecto (SSH y *Apache 2* lo hacen).

Para ello, podemos crear un *script* de *bash* ejecutable, que podemos llamar `container_init.sh`, en la misma carpeta donde está la `Dockerfile` de la imagen a construir, por ejemplo, para no tener que poner rutas. Dentro se escriben las **órdenes de arranque de los servicios** que queramos como lo haríamos en

cualquier sistema *Linux*, tantas como servicios haya. Por ejemplo, para SSH y *Apache 2* el fichero sería este, pero pueden arrancarse todos los servicios necesarios que queramos:

```
#!/bin/bash
service ssh start
service apache2 start
tail -f /dev/null
```

El "truco" para arrancar los servicios que queramos tal y como lo haríamos en *Linux* es el `tail -f` que vemos al final. Esta instrucción le dice a *Linux* que imprima el final del dispositivo `/dev/null`, que está vacío, y como consecuencia de eso se para la ejecución del *script* indefinidamente. Como `tail` **es un proceso en primer plano**, el contenedor no muere y los servicios que queramos quedan arrancados.

Hecho esto, basta con copiar el *script* dentro del contenedor a arrancar (`COPY`) y ejecutarlo (`CMD`), como vemos en la siguiente imagen. Puedes probar la técnica con las imágenes incrementales del ejercicio anterior si quieres.

```
# Copy and run an initialization script to
# start the appropriate servers in the containers
COPY container_init.sh container_init.sh
RUN chmod +x container_init.sh
CMD ["./container_init.sh"]
```

N4DOCKER_REDESADV: Configuración avanzada de redes *(0,2 puntos)*

Descripción de la actividad

Consiste en aprender a crear redes distintas que **comuniquen contenedores *Docker*** entre sí.

Resultados esperados

Eres capaz de arrancar varios contenedores en redes *bridge* diferentes y **comprobar que están aislados entre ellos**, siendo además capaz de eliminar este aislamiento conectando redes cuando sea necesario. Puedes contestar a esta pregunta: *dado que un mismo contenedor puede estar conectado a todas las redes que quiera ¿Crees que es viable aislar una aplicación en varias capas como vimos en el SNI del primer tema de teoría, usando proxies para controlar el tráfico entre ellas?*

Información necesaria para su realización

En esta actividad se trata de poner en práctica la **creación de redes** vista en teoría para tener varios contenedores que se comuniquen entre ellos, pero que los demás no tengan acceso a su contenido. Para ello crearemos un nueva red con `docker network create` como hemos visto en teoría, y haremos las siguientes operaciones:

- **Arrancar un contenedor** de los creados en alguna actividad anterior conectado a una red tipo `bridge` nueva que creemos previamente, y comprobar que tiene conectividad a internet (`apt-get update`)

- **Crear otra red** `bridge` en un rango de IPs distinto a la anterior, y arrancar otro contenedor conectado a la misma. Comprobar que desde el primer contenedor no es posible acceder al otro
- **Conectar el primer contenedor a la red del segundo** con `docker network connect`
- **Comprobar que ahora sí pueden verse** ambos contenedores (puede usarse `ping` o acceder a cualquier servicio que los contenedores arrancados tengan)
- **Desconectar el primer contenedor de la red del segundo** y comprobar que ahora vuelve a no ser posible que se vean entre ellos
- Parar ambos contenedores y borrar todas las redes creadas con `docker network prune`

N4DOCKER_GUIAPP: Aplicaciones con GUIs dentro de contenedores de aplicaciones *(0,1 puntos)*

Descripción de la actividad

Consiste en aprender a **arrancar aplicaciones que tienen GUIs** dentro de contenedores *Docker*.

Resultados esperados

Esta actividad se dará por finalizada cuando consigas arrancar una aplicación gráfica desde un contenedor que la tenga instalada y pueda usarse desde el *host*. Puedes contestar a esta pregunta: *¿Por qué crees que es más seguro ejecutar una aplicación así?*

Información necesaria para su realización

Es habitual asumir erróneamente que *Docker* solo funciona con aplicaciones sin GUI, pero esto no es cierto y **podemos ejecutar aplicaciones con GUI de manera aislada en un contenedor**. Para esto necesitamos hacer una **redirección del sistema gráfico *XServer* de *Linux*** que haga que la aplicación del contenedor se ejecute sobre el sistema gráfico del *host*. El objetivo de esta actividad es implementar precisamente esto.

Para hacer esto, creamos una *Dockerfile* que instale la aplicación gráfica deseada y la ejecute en el contenedor. Como ejemplo, vamos a habilitar la navegación por internet en un *Firefox* dentro de un contenedor, y lograr así un mayor nivel de aislamiento en caso de problemas (más seguridad). La `Dockerfile` puede ser esta:

```
FROM ubuntu
RUN apt update
RUN apt install -y firefox
CMD ["/usr/bin/firefox"]
```

Y con ella construimos la imagen correspondiente, a la que llamaremos `browserdocker`: `sudo docker build -t browserdocker .`. Hecho esto, redirigimos posteriormente el *display* local al usuario `root` con `xhost local:root` y arrancamos la aplicación con: `docker run --net=host --env="DISPLAY" --volume="$HOME/.Xauthority:/root/.Xauthority:rw" browserdocker`.

NOTA: Es posible que, si no funciona directamente como se ha explicado, haya que hacer `xhost +` desde el host para eliminar problemas con los gráficos.

N4DOCKER_PLATMIW: Instalación de plataformas especializadas de otras asignaturas *(0,5 puntos)*

Descripción de la actividad

Consiste en aprender a **desplegar plataformas para trabajar en otras asignaturas del máster** dentro de contenedores *Docker* adaptados para ellas.

Resultados esperados

Esta actividad se completará cuando puedas crear satisfactoriamente los contenedores enumerados que preparen automáticamente *software* que se usa en otras asignaturas del máster. Puedes contestar a estas preguntas:

- ¿Cuál crees que es la ventaja principal de hacer infraestructuras en esta clase de contenedores?
- ¿Ves además alguna desventaja de hacerlo así?

Información necesaria para su realización

En esta actividad se usarán todos los conocimientos anteriores para preparar contenedores de **la última versión estable** de estas herramientas. Estos contenedores corresponderán a entornos de desarrollo que se van a usar en otras asignaturas del máster, por lo que se ruega que **se instalen a partir de imágenes oficiales que se encuentren en el *Docker Hub*.**

- **Administración de Sistemas de Persistencia de Objetos**: *Oracle Enterprise Linux 7*
- **Gestores de Contenidos Web**: Un *Ubuntu 18* o superior con XAMPP
- **Programación Orientada a Objetos:** Un *Ubuntu 18* con *Python 3* y `virtualenv`
- **Sistemas de Persistencia de Objetos**: Un *Ubuntu 18* con *Hadoop* (*https://hub.docker.com/r/apache/hadoop*)
- **Sistemas de Seguridad en la Web**: Una máquina *Kali Linux* (versión *Rolling Release*)

N4DOCKER_WEB3CAPAS: Creación de infraestructuras de red en tres capas y *reverse proxy* con *Docker Compose* *(0,3 puntos)*

Descripción de la actividad

Consiste en aprender a **crear una arquitectura de contenedores *Docker* que permita desplegar una web en tres capas** aisladas.

Resultados esperados

Eres capaz de arrancar varios contenedores distintos, configurarlos y exportar algunos de sus servicios a través de puertos del *host* imitando una infraestructura de tres capas similar a la SNI que vimos al principio de la asignatura

Información necesaria para su realización

La última actividad de este bloque consiste en aplicar todos los conocimientos vistos y los siguientes nuevos para **crear una infraestructura en capas que imite a la SNI vista al principio de esta asignatura**. Se trataría de hacer una infraestructura en tres capas de la siguiente forma:

- ***Public DMZ***: un contenedor oficial `apache2`, que debe exponer su puerto 80 en el *host* y servir una web de ejemplo desde un volumen
- ***Private DMZ***: un contenedor oficial `tomcat`. No es necesario añadir nada al mismo.
- ***Intranet***: un contenedor oficial `mysql` o `mariadb`. No es necesario añadir nada al mismo.

Dado que **no es posible hacer contenedores que dentro tengan un *firewall***, ya que la tecnología de contenedores de aplicaciones no permite hacer modificaciones sobre sus reglas, omitiremos estas máquinas intermedias que separan cada capa.

Lo que si podemos hacer en sustitución de dichos *firewalls* es **crear redes privadas individuales** que conecten la máquina `apache2` con la `tomcat` y otra independiente que conecte la `tomcat` con el SGBD, valiéndonos de la sintaxis que aparece en los ejemplos de teoría, tanto para crear redes como para situarlas en cada contenedor, y que se reproduce a continuación. Se puede además hacer uso de la opción `hostname` para que el nombre de cada contenedor se distinga de otros.

```yaml
version: "2"
services:
    eii:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab10_web_eii
        hostname: lab10_web_eii
        restart: on-failure
        volumes:
            - ../../webs/web_eii:/var/www/html
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.3
    epi:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab10_web_epi
        hostname: lab10_web_epi
        restart: on-failure
        volumes:
            - ../../webs/web_epi:/var/www/html
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.4
    proxy:
        image: ssi/ubuntu_apache2_ssh_proxy
        container_name: lab10_reverse_proxy
        hostname: lab10_reverse_proxy
        restart: on-failure
        volumes:
            - ./volume_data/ssh:/shared
            - ./volume_data/rules:/rules/
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.2
            lab10_front_net:
                ipv4_address: 198.102.0.3
```

```yaml
    kali:
        image: ssi/kali_nids
        container_name: lab10_kali
        hostname: lab10_kali
        restart: on-failure
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        volumes:
            - ./volume_data/kali:/shared
        networks:
            default:
            lab10_front_net:
                ipv4_address: 198.102.0.2

networks:
    lab10_back_net:
        driver: bridge
        ipam:
            config:
                - subnet: 172.10.0.0/16
    lab10_front_net:
        driver: bridge
        ipam:
            config:
                - subnet: 198.102.0.0/16
```

Nivel 5 (Almirante): Operaciones avanzadas con contenedores de aplicaciones

N5DOCKER_MANAGERGUI: Instalación de un GUI para la gestión de contenedores *(0,2 puntos)*

Descripción de la actividad

Consiste en poner en marcha una **aplicación gráfica que te permita gestionar correctamente tus contenedores** *Docker*.

Resultados esperados

Eres capaz de instalar y arrancar *Docker Desktop* o *Portainer* para *Linux* (elige **sólo una de las opciones**) y manejar contenedores haciendo las operaciones indicadas sobre ellos

Información necesaria para su realización

Opción 1: Docker Desktop

Docker Desktop es probablemente el software más popular para **manejar imágenes y contenedores que estén en el sistema de forma gráfica**, lo cual puede facilitar mucho trabajar con ellos. Es una herramienta oficial que podemos instalar siguiendo estas instrucciones: https://docs.docker.com/desktop/linux/install/ubuntu/. No obstante, en la actualidad usarla necesita que cumpláis con dos requisitos, porque en caso contrario **NO FUNCIONARÁ**:

- Usar una máquina física o una virtual **con soporte de virtualización por *hardware*** (en el caso de ser una máquina virtual, debe tener activa la **virtualización por *hardware* anidada**).
- Usar una versión de sistema operativo que soporte este *software*. **En caso de *Ubuntu* sería de la 22.04 en adelante** (no la 18.04 con la que se hacen el resto de los ejercicios).

Si cumples con estas condiciones, esta actividad consiste en instalar y arrancar esta herramienta y explorar todas sus opciones de acuerdo con su **manual de usuario** (https://docs.docker.com/desktop/linux/), interactuando con su ***Dashboard*** (https://docs.docker.com/desktop/dashboard/) para arrancar y parar un contenedor y acceder a su consola.

Para esta actividad deben probarse opciones básicas como importar una imagen de *Docker Hub*, crear un contenedor de esta, acceder a la consola del contenedor, crear una red y crear un volumen, por lo menos.

Opción 2: Portainer

Esta opción para hacer este ejercicio está especialmente indicada para aquellos de vosotros **que no tengáis la posibilidad de tener una máquina virtual con extensiones de virtualización anidadas activas**, o simplemente que prefiráis tener un GUI en el mismo sistema *Docker CE* que tenéis instalado, y no en uno que se ejecuta aislado en paralelo, como *Docker Desktop*.

La instalación de *Portainer* sobre *Ubuntu 18.04* es muy sencilla siguiendo documentación oficial: https://www.portainer.io/blog/portainer-your-docker-gui-for-your-ubuntu-linux-desktop y este ejercicio requiere que manejéis los contenedores con la GUI de *Portainer* probando **las mismas operaciones** que se describieron para *Docker Desktop*.

N5DOCKER_MINSIZE: Minimización del tamaño de contenedores *(0,2 puntos)*

Descripción de la actividad

Consiste en aprender a **optimizar el tamaño de los contenedores *Docker*** que creas.

Resultados esperados

Eres capaz de transformar una imagen que se crea con una `Dockerfile` en una equivalente pero que ocupe menos espacio gracias a las técnicas vistas en teoría. Puedes contestar a esta pregunta: *¿Por qué crees que minimizar el tamaño de los contenedores Docker es muy importante en producción?*

Información necesaria para su realización

En esta actividad se trata de que, a partir de la imagen creada en la actividad "*Uso avanzado de Dockerfiles*", se hagan los cambios necesarios para generar una nueva imagen que haga lo mismo, pero con todas las medidas que hemos visto en teoría para **minimizar el tamaño de la imagen creada**, comparando el tamaño de la imagen original con la nueva y, por tanto, cuánto se ha reducido el tamaño de la imagen con esas medidas.

N5DOCKER_AISLAR: Aislamiento de conexiones y vinculación de contenedores *(0,2 puntos)*

Descripción de la actividad

Consiste en aprender a configurar de manera detallada y avanzada **qué contenedores *Docker* pueden ver a otros**.

Resultados esperados

Puedes anular la comunicación por defecto entre contenedores en la misma red, y usar el enlazado de contenedores para que **sólo aquellos que estén expresamente autorizados** puedan hacerlo.

Información necesaria para su realización

En esta actividad se trata de activar la comunicación por defecto entre contenedores como se mencionó en la transparencia "*Prohibir la comunicación*" de la teoría y comprobar que **efectivamente ya no es posible contactar desde un contenedor a otro donde antes sí era posible**.

Una vez hecho esto, y usando contenedores de imágenes creados en actividades anteriores, se trata de probar a **enlazar dos de ellos** y ver cómo esos dos **sí pueden comunicarse** mientras el resto no. También se requiere **ver cómo se han creado variables de entorno** en los contenedores enlazados que apuntan a recursos del otro contenedor (comando `env`).

N5DOCKER_PRIVREG: Registros privados de contenedores *(0,3 puntos)*

Descripción de la actividad

Consiste en **aprender a crear tu propio registro de contenedores *Docker***.

Resultados esperados

Eres capaz de crear tu propio registro de imágenes local, subir una imagen al mismo y descargarla de nuevo de allí. Puedes contestar a las siguientes preguntas:

- *¿Puedes enumerar las ventajas de seguridad que tiene usar tu propio registro en lugar de Docker Hub?*
- *¿Puedes hacer lo mismo teniendo en cuenta la privacidad de los datos de tu organización?*

Información necesaria para su realización

Hasta ahora hemos trabajado con el registro general en Internet *Docker Hub*, pero **en esta actividad vamos a aprender a crear el nuestro propio**. Esto es típico de muchas empresas que tienen sus contenedores preparados y configurados para sus propias necesidades, y no quieren compartirlos con el mundo por distintos motivos, así que **no tiene sentido para ellos usar el *Docker Hub***.

Para obtener la imagen que implementa el *software* de un registro de imágenes *Docker* (`registry`) y arrancar un contenedor con ella necesitamos ejecutar una orden como: `docker run -d -p 5000:5000 --restart=always --name miregistro registry:2`

Esto hace que el registro esté disponible en el puerto `5000` del *host*, aunque podemos asignar cualquier combinación de IP y puerto. Debido a lo crítico de la función de este contenedor, es muy importante especificar la opción `--restart=always` para que, si el contenedor se apaga, **se reinicie automáticamente**. Esto permite que esté operativo todo el tiempo posible. **Más información**:

https://docs.docker.com/engine/reference/run/#restart-policies-restart. El objetivo es tener una salida como esta:

```
redondo@miw:~$ sudo docker run -d -p 5000:5000 --restart=always --name miregistr
o registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
cbdbe7a5bc2a: Pull complete
47112e65547d: Pull complete
46bcb632e506: Pull complete
c1cc712bcecd: Pull complete
3db6272dcbfa: Pull complete
Digest: sha256:8be26f81ffea54106bae012c6f349df70f4d5e7e2ec01b143c46e2c03b9e551d
Status: Downloaded newer image for registry:2
09d3ffc99dcb08337f7cef7432622f0a147112138c8b730d48f4447267769f97
```

Una vez tenemos nuestro registro de imágenes local operativo, para sacar de *Docker Hub* la imagen llamada `ubuntu` y **meterla con el mismo nombre en nuestro registro privado** hacemos lo siguiente

- **Renombramos la imagen** `ubuntu` local con un nombre que indique que se va a guardar en nuestro registro local: `docker pull ubuntu && docker tag ubuntu localhost:5000/ubuntu`
  - Así creamos una copia de la imagen cuyo nombre indica que se va a subir a nuestro registro, que reside en la misma máquina en el puerto `5000`. Debemos llamar a las imágenes así si vamos a subirlas a un registro particular
  - Si da un problema de permisos es necesario cambia los permisos de `/var/run/docker.sock` a `666`
- **La subimos a nuestro registro**: `docker push localhost:5000/ubuntu`
- Ahora podríamos **borrarla del almacenamiento local** y descargarla de nuestro registro para asegurarnos de que es esa la que se va a usar en adelante, usando el nombre que le dimos en el primer paso: `docker pull localhost:5000/ubuntu`

```
redondo@miw:~$ sudo chmod 666 /var/run/docker.sock
redondo@miw:~$ sudo docker pull ubuntu && docker tag ubuntu 127.0.0.1:5000/ubunt
u
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:55cd38b70425947db71112eb5dddfa3aa3e3ce307754a3df2269069d2278ce47
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
```

- Si ya no queremos tener un registro local, podemos eliminarlo así: `docker stop registry && docker rm -v registry`. **Más información**: https://docs.docker.com/registry/deploying/

N5DOCKER_VAGRANTDOCKER: *Vagrant y Docker* *(0,2 puntos)*

Descripción de la actividad

Consiste en **aprender a desplegar contenedores *Docker* usando *Vagrant***.

(Esta actividad requiere hacer hecho el módulo de Vagrant hasta el nivel 2 o superior)

Resultados esperados

Eres capaz de crear una imagen con el *software* que desees instalado usando las funcionalidades combinadas de *Vagrant* y *Docker*, uniendo así ambos módulos de la asignatura para conseguir un fin

Información necesaria para su realización

En esta actividad consiste en aplicar tus conocimientos de *Vagrant* para **usar un proveedor de virtualización que sea *Docker*** en lugar de los que vimos en el módulo anterior. Para ello, además de cambiar el proveedor por `"docker"` en la `Vagrantfile`, especificamos que la `Dockerfile` a usar está en el mismo directorio (`"."`) con la propiedad `build_dir` de dicho proveedor y, además, avisamos a *Vagrant* de que el contenedor seguirá ejecutándose (`remains_running = true`) tras arrancarlo.

```
  GNU nano 2.5.3                    Archivo: Vagrantfile

Vagrant.configure("2") do |config|
        config.vm.provider "docker" do |d|
                d.build_dir = "."
                d.remains_running = true
        end
end
```

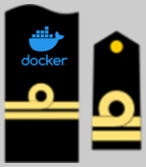
Con este proveedor podemos construir imágenes de manera automatizada gracias a una *Dockerfile* que **se tratará tal y como hemos visto en niveles anteriores**.

Para completar la actividad, se pide **ser capaz de construir una imagen cualquiera** (nueva o de las ya mencionadas) usando *Vagrant*. Hay que recordar que la restricción de que si un contenedor no ejecuta un proceso en primer plano muere inmediatamente **sigue presente**, por lo que tendremos que colocar al final de la *Dockerfile* una instrucción `RUN` como la de la imagen para arrancar el servicio deseado en el contenedor, o bien recurrir a las técnicas de arranque de servicios que se vieron en niveles anteriores.

```
RUN /usr/sbin/apache2ctl -D FOREGROUND
```

Insignias y Autoevaluación

Esta asignatura hace uso de insignias. Esto quiere decir que, cuando vayas desbloqueando niveles de maestría en la tecnología, podrás obtener **una insignia pública** que luego podrás usar en tu currículum gracias al soporte de estas del campus virtual. Este enlace te enseña a usar esta característica para poder unirlas a tu currículum: https://docs.moodle.org/all/es/Insignias. Cada insignia pública **lleva adjuntos los textos de las habilidades que desbloquea** que puedes consultar en la siguiente tabla, para que cualquiera los vea si decides usarlas y sepa lo que has conseguido hacer, validado por mí. De esta manera, mi intención es ayudarte a aumentar tus posibilidades de contratación en un futuro, **exponiendo claramente lo que has podido hacer en esta asignatura** referente a cada tecnología sobre la que has decidido evaluarte.

Rango de Maestría en la Tecnología y Habilidades Demostradas		Insignia Pública Obtenida
Nivel 1: Cabo de Contenedores de Aplicaciones		
	Es capaz de instalar el *software* necesario para arrancar contenedores de aplicaciones desde un *host*	
	Puede crear un contenedor de aplicaciones simple a partir de una imagen preconstruida y conectarse interactivamente para interactuar con él ***(2 ejercicios)***	
	Puede crear un contenedor con un servidor web y conectarse a él desde un *host*	
	Puede parar y eliminar un contenedor cualquiera y también eliminar todos los contenedores e imágenes de un sistema	
Nivel 2: Sargento de Contenedores de Aplicaciones		
	Puede crear un contenedor personalizado con cualquier *software* que necesite	
	Puede habilitar la conexión a un contenedor vía SSH	
	Puede exponer cualquier servicio que se ejecute en un contenedor al exterior de un *host*, incluso con varios contenedores arrancados al mismo tiempo ***(2 ejercicios)***	
Nivel 3: Teniente de Navío de Contenedores de Aplicaciones		
	Es capaz de construir una imagen que tenga todo el *software* que sea necesario y todos los ficheros que se requieran para cumplir con una función usando *Dockerfiles* ***(2 ejercicios)***	
	Es capaz de usar volúmenes de un contenedor de aplicaciones para guardar de forma persistente información o importar ficheros de un *host*	
	Puede construir infraestructuras simples de contenedores con *Docker Compose*	
Nivel 4: Capitán de Navío de Contenedores de Aplicaciones		

	Puede construir *Dockerfiles* por capas, de manera que no se repitan operaciones idénticas en contenedores distintos y se refactoricen las *Dockerfiles* creadas	
	Es capaz de arrancar múltiples servicios en un contenedor si fuera necesario	
	Puede crear redes para ciertos contenedores y conectarlos y desconectarlos según sea necesario	
	Sabe cómo arrancar una aplicación con GUI dentro de un contenedor	
	Es capaz de instalar plataformas de desarrollo que sean necesarias para otros contextos con *Docker*, además de un sistema para desplegar una web en tres capas ***(2 ejercicios)***	
Nivel 5: Almirante de Contenedores de Aplicaciones		
	Puede instalar y usar *Docker Desktop* o *Portainer* para la gestión de contenedores	
	Es capaz de reducir el tamaño de una imagen *Docker* aplicando una serie de técnicas avanzadas	
	Puede prohibir la comunicación entre contenedores y enlazar aquellos que desees expresamente que se comuniquen	
	Es capaz de crear y usar tu propio registro de imágenes local	
	Puede coordinar *Vagrant* y *Docker* para desplegar contenedores y crear imágenes	
	**Cruz con el distintivo azul en Contenedores de Aplicaciones**: Ha completado con éxito **4 puntos** (aproximadamente el 80%) de las tareas correspondientes de los 5 niveles de la parte de *Contenedores de Aplicaciones*, demostrando así su maestría en esta tecnología y estar preparado para usarla en tu futuro trabajo de forma efectiva.