Práctica 7: Uso básico de Docker

Introducción a Docker

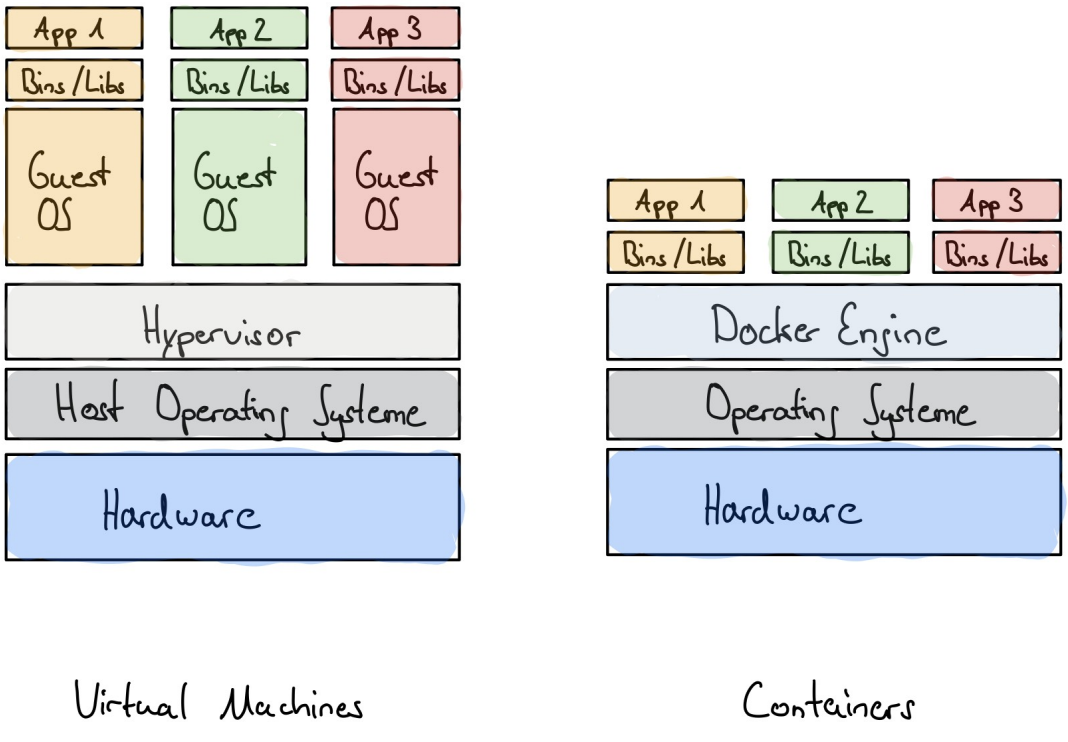
Funcionamiento

Docker es una plataforma que ofrece soporte para la ejecución de aplicaciones y servicios en *contedores*.

- Un *contenedor* ofrece un entorno de ejecución aislado en lo que respecta a ficheros, procesos, redes y demás recursos del sistema, funcionando de forma similar a una máquina virtual, aunque con una menor sobrecarga, ofreciendo una virtualización a nivel de SO.
- Los *contenedores* se crean a partir de *imágenes* Docker (propias o disponibles en repositorios como Docker Hub), definidas en *Dockerfiles*, que funcionan como plantillas para la creación de los *contenedores* que se ejecutarán en un entorno de ejecución aislado.
- Docker y otras tecnologías de contenedores como LXC, ofrecen este tipo de entornos de ejecución aislados haciendo uso de una serie de elementos y servicios de los que dispone sistema operativo para proporcionar aislamiento de recursos:
  - *chroot*: permite la ejecución de un proceso en una jerarquía de ficheros propia (`/`, `/etc`, `/dev`, `/usr`, etc), separada de la del sistema subyacente
  - *Linux namespaces*: característica del kernel de Linux que permite "separar" los recursos del kernel que son "vistos" por un conjunto de procesos.
    - UTS(Unix Time Sharing) namespace: Aisla hostname y dominio.
    - PID namespace: Aisla identificadores de proceso.
    - Mounts namespace: Aisla puntos de montaje.
    - IPC namespace: Aisla recursos de comunicación entre procesos.

        - Network namespace: Aisla recursos de red.
        - User namespace: Aisla identificadores de usuario y de grupos.

      Como resultado se consigue que los procesos dentro de un espacio de nombres tengan la percepción de tener su propia instancia aislada de los recursos del sistema.
    - *cgroups*: característica del kernel de Linux que permite definir y acotar la limitación, priorización, contabilidad y control de recursos (CPU, memoria, entrada/salida, ...).



Elementos principales

Imágenes

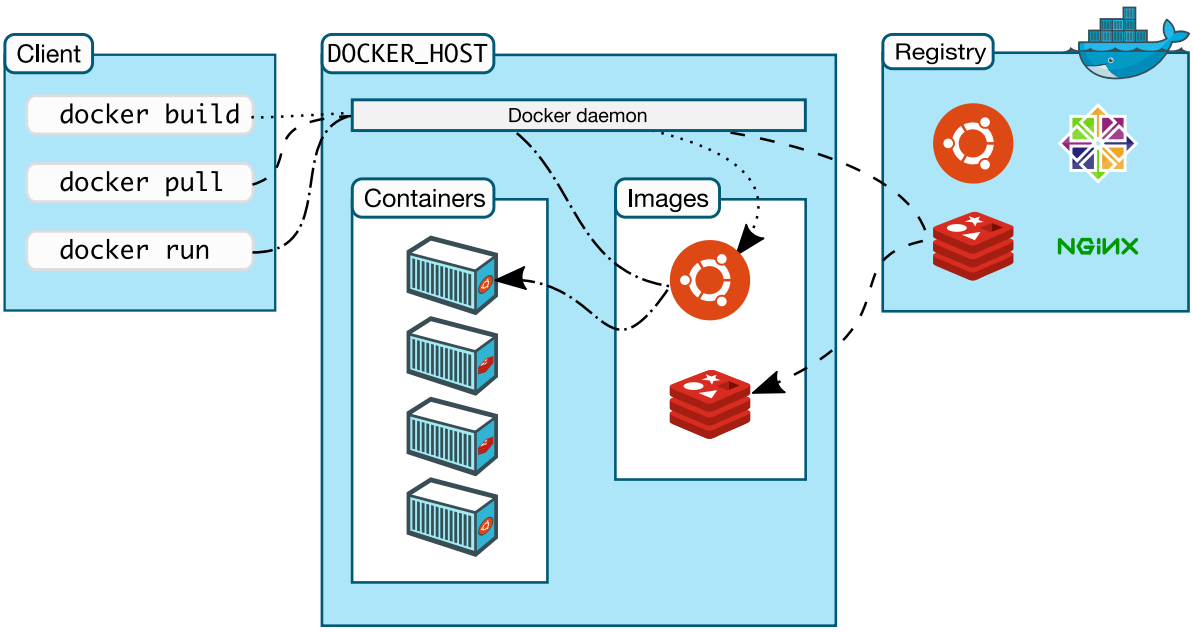
Plantillas de sólo lectura que sirven de base para la creación del contenedor.

- Para crear un contenedor es necesario que la correspondiente imagen esté disponible en el "registro local" de imágenes
- Docker ofrece en Docker Hub (https://hub.docker.com/) un repositorio público de imágenes aportadas por la comunidad que pueden ser desacargadas y usadas en local
- Definidas en *Dockerfiles* que son archivos de configuración donde se describe la composición de la imagen a partir de una *imagen base* previa (~ soporta herencia), los comandos y ficheros a incluir para construir la nueva imagen, junto con información adicional como puertos redirigidos, volúmenes a usar o el comando a lanzar una vez creado un contenedor a partir de esa imágen.
- Imagenes se organizan en "capas" (*layers*) como consecuencia de la jerarquía/herencia definida en la cadena de *Dockerfiles*, cada una con los cambios a aplicar sobre el sistema de ficheros de la capa anterior, según indique el correspondinte *Dockerfile*. La imagen final a partir de la que se crea el contenedor el resultado de acumular esos cambios.

Principales comandos para la gestión de imágenes:

- `docker search`: Busca imágenes en Docker Hub
- `docker images` (alias de `docker image ls`): Muestra las imágenes disponibles en el registro local.
- `docker pull` (alias de `docker image pull`): Descargar la última versión de la imagen indicada

- `docker rmi` (alias de `docker image rm`) : Permite eliminar imágenes (no es posible eliminar imágenes con contenedores creados)
- `docker inspect` (alias de `docker image inspect`): Muestra la información de la imágen indicada
- `docker commit` (alias de `docker container comit`): Crea una imagen Docker a partir del estado actual del contenedor indicado
- Otros: ver `docker image --help`



Contenedores

Los contenedores son instancia en ejecución una imagen.

- El motor de Docker crea el sistema de ficheros donde estarán confinados los procesos del contenedor a partir de la información de la imagen base (capas de sólo lectura), creando una capa superior de lectura-escritura. volátil (se elimina al finalizar la ejecución del contenedor)
- Una vez creado el contenedor, Docker ejecutará el comando indicado en el Dockerfile de la imagen o el comando especificado al crear el contenedor.
- Mediante el mecanismo de *namespaces* y *cgroups* Docker aisla y limita los recursos del sistema a los que tienen acceso los procesos del contenendor.

Principales comandos para la gestión de contenedores:

- `docker ps` (alias de `docker container ls`): Muestra los contenendores en ejecución (con `-a` los muestra todos y con `-s` informa del espacio ocupado)
- `docker create` (alias de `docker container create`): Crea un nuevo contenedor a partir de la imagen indicada y el proceso a ejecutar indicado, pero no lo pone en marcha.
- `docker start|stop|restart` (alias de `docker container start|stop|restart`): Pone en marca, para o reinicia un contenedor previamente existente.
- `docker attach` (alias `docker container attach`): Conecta el terminal del anfitrión a la entrada y salida estándar (STDIN, STDOUT y STDERR) del comando en ejecución del contenedor.
    - *Importante:* Si el comando en ejecución es un *shell* (`/bin/bash`), un `exit` terminará la ejecución de ese proceso y del contenedor. Para retornar al terminal del anfitrión sin parar el contendor se puede salir con `[Control] + pq`.

- `docker run` (alias de `docker container run`): Crea y pone en ejecución un contenedor con la imagen indicada y el procesos a ejecutar indicado (combina `docker create` y `docker start`).
- `docker exec` (alias de `docker container exec`): Ejecuta el comando indicado sobre un contenedor en ejecución.
- `docker inspect` (alias de `docker container inspect`): Muestra la información del contenedore indicado.
- `docker rm` (alias de `docker container inspect`): Elimina un contenedor (no debe estar en ejecución)
- Otros: ver `docker container --help`.

Parámetros relevantes de `docker create` y `docker run`:

- `--name`: da nombre al contenedor (alternativa a usar el UUID que asigna Docker)
- `-i|--interactive`: modo interactivo (conecta a STDIN del proceso del contenedor), suele ir acompañador de `-t|--tty` que crea un pseudo terminal
- `-p|--publish <puerto anfitrion>:<puerto contenedor>`: configura la redirección de puertos entre anfitrión y contenedor
- `-e|--env <variable>=<valor>`: establece variables de entorno en el contenedor
- `--network <nombre>`: conecta el contenedor a la red indicada (si no se indica por defecto se conecta a la red `bridge` creada por Docker)
- `-v|--volume <volumen|ruta>:<punto de montaje>`: conecta un *volumen Docker* (o un fichero/directorio del anfitrión) en el punto de montaje indicado en el contenedor
- `-m|--mount type=volume|bind,src=<volumen|ruta>;dst=<punto de montaje>`: conecta un *volumen Docker* (o un fichero/directorio del anfitrión) en el punto de montaje indicado en el contenedor (casi equivalente a `-v`)
- `--ip <direccion>` ,`-h|--hostname <nombre>`, `--add-host`, `--dns`, etc: configuran aspectos de la red del contenedor
- `--cpus`, `-m|--memory`, etc: limita uso de recursos (CPU, memoria, etc) por parte del contenedor
- `--rm`: elimina el contenedor automáticamente cuando termine su ejecución
- `-d|--dettach` (sólo con `docker run`): ejecuta el contenedor en *background*
- `--link <nombre contenedor>:<alias>`: establece una conexión de red con un contenedor existente (*deprecated*, se recomienda usar redes y conexiones explícitas)

Volúmenes y *bind mount*

Los ficheros manejados por los contenedores son efímeros y no se mantienen entre ejecuciones.

Docker ofrece dos mecanismos (volúmenes Docker y *bind mount*) para mantener datos persitentes entre ejecuciones de un mismo contenedor o para compartir datos entre varios contenedores o entre contenedor y anfitrión.

1. Volúmenes Docker

Almacenamiento de datos gestionado por Docker (almacenado en `/var/lib/docker/volumes`, no editable directamente).

- Pueden tener nombre o ser anónimos.
- Utilidad:
    - almacenamiento persistente que debe mantenerse entre ejecuciones de un mismo contenedor

    - compartición de datos entre diferentes contenedores

Modo de uso: Al crear el contenedor (`docker create` ó `docker run`) los volúmenes (con nombre o anónimos) son vinculados a los contenendores con `-v|--volume` ó `-m|--mount type=volume`, indicando en ambos casos el *punto de montaje* en el contenedor donde se ubicará el contenido del volumen indicado.

Comandos principales:

- `docker volume create`: Crea un volumen con el nombre indicado
- `docker volume rm`: Elimina el volumen indicado (`docker volumen prune` elimina los volúmenes que no están siendo usados)
- `docker volume ls`: Listado de los volúmenes creados.
- `docker volume inspect`: Información detallada del volumen indicado

2. Bind mount

Docker permite vincular(montar) directorios o ficheros del sistema anfitrión en la jearquía de directorios del anfitrión.

- Utilidad:

- mismos que los volúmenes Docker
- compartición de datos con el anfitrión

Modo de uso: Al crear el contenedor (`docker create` ó `docker run`) son los *bind mount* son vinculados a los contenendores con `-v|--volume` ó `-m|--mount type=bind`, indicando en ambos casos la ruta del directorio/fichero del anfitrión y el *punto de montaje* en el contenedor donde se ubicará el contenido del directorio/fichero montado.

Redes

Por defecto en Docker hay 3 redes predefinidas (`bridge`, `host` y `none`), pero es posible crear redes definidas por el usuario.

- Las redes de tipo *bridge* (incluida la red `bridge` por defecto) permiten la conectividad entre contenedores conectados a la misa red *bridge* (intermanente Docker establce reglas de filtrado que impiden tráfico entre contenedores en redes distintas). Este tipo de redes permiten:
    - aislar los contenedores en subredes distintas y asiladas entre sí según sea necesario para nuestra aplicación
    - aislar los contendores del acceso exterior, salvo los accesos a puertos redireccionados explícitamente con la opción de redirección `-p`, que Docker traducirá en las correspondientes reglas `iptables` en el anfitrión.
- En las redes de tipo *host* la red del contenedor no está aislada de la red del anfitrión (el contenedor comparte el *namespace* de red del host). En este caso, la redirección de puertos con `-p` no tiene efecto, porque no es necesaria.

Ver (https://docs.docker.com/network/)

Salvo que se indique lo contrario con la opción `--network`, los contenedores se conectan por defecto a la red de tipo *bridge* llamada `bridge` con direcciones 172.17.0.0/16 (siendo la IP `172.17.0.1` el propio anfitrión que actúa como *gateway*). Los contenedores conectados a esta red que quieren exponer algún puerto al exterior tienen que usar la opción `-p` para mapear puertos.

En general las redes *bridge* definidas por el usuario son más flexibles que la red `bridge` por defecto: tienes resulución DNS implícita, permiten conexiones "en caliente" de nuevos contenendores (la conexión a la red `bridge` por defecto sólo puede hacerse en el momento de la creación del contenedor) y ofrecen mayor control y aislamiento.

Comandos relevantes para redes definidas por el usuario:

- `docker network ls`: Listado de las redes disponibles
- `docker network create`: Crea una red con el nombre indicado, del tipo indicado (`-d|--driver`) y con los parámetros indicados (`--subnet`, `--gateway`, etc)
- `docker network rm`: Elimina la red indicada (`docker network prune` elimina todas las redes). Sólo se permite borrar redes que no tengan contenedores conectados.
- `docker network connect <red> <contenedor>`: Conecta el contenedor indicado a la red indicada (puede parametrizarse con `--ip`) (con `--link` estable un enlace con el contenedor indicado a través de esa red)
- `docker network disconnect <red> <contenedor>`: Desconecta un contenedor de la red indicada.
- `docker network inspect`: Información detallada de la red indicada
- Otros: `docker network --help`.

PREVIO: Instalación de Docker

Equipos físicos del laboratorio

Ya tienen Docker instalado, con los privilegios adecuados, por lo que se puede usar con cualquier usuario (con el `alumno` genérico o con el propio de cada alumno)

Equipos propios

Están disponibles las instrucciones de instalación para MS Windows, iOS y diversas distribuciones GNU/Linux en la web de Docker: Docker Engine Instalation

Máquina virtual de prácticas

Scripts de instalación:

- GNU/Linux: (http://ccia.esei.uvigo.es/docencia/CDA/2223/practicas/ejercicio-docker.sh)
- MS Windows: (http://ccia.esei.uvigo.es/docencia/CDA/2223/practicas/ejercicio-docker.ps1)

1. Acceder con el usuario `root` y la contraseña `purple` y arrancar el entorno gráfico con `startx`.
2. Desde un terminal, instalar con `apt` los paquetes Docker de la distribución Debian (paquete `docker.io`) .

   ```
   root@docker:~# apt update
   root@docker:~# apt install docker.io docker-compose
   ```

   - **Nota:** La versión de los mantenedores del paquete .deb es un poco más antigua que la disponible en los repositior oficiales de Docker.

3. Opcionalmente, se puede añadir al usuario sin privilegios `usuario` al grupo `docker` para que se le permita ejcutar los comandos de Docker sin necesidad de usar `sudo.`

```
root@docker:~# usermod -aG docker usuario
```

Ejemplos

Ejemplo 1: Imágenes y contenendores

- Lanzar Docker `hello-world` y comprobar las imágenes y contenedores disponibles.

```
root@docker:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:faa03e786c97f07ef34423fcceeeec2398ec8a5759259f94d99078f264e9d7af
Status: Downloaded newer image for hello-world:latest

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

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

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

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

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

```
root@docker:~# docker images -a     # tambien docker image ls -a
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   13 months ago   13.3kB

root@docker:~# docker ps -a         # tambien docker container ls -a
CONTAINER ID   IMAGE         COMMAND       CREATED             STATUS
            PORTS     NAMES
6e0d3a8c896c   hello-world   "/hello"      2 minutes ago       Exited (0) 2
minutes ago                 wizardly_nobel
```

- Crear un contenedor Debian y ver la diferencias entre `docker create`, `docker start`, `docker run`, `docker exec`

1. Busca y recuperar la imagen `debian`

```
root@docker:~# docker search debian
NAME                                      DESCRIPTION
        STARS     OFFICIAL   AUTOMATED
ubuntu                                    Ubuntu is a Debian-based Linux
operating sys…   15223     [OK]
debian                                    Debian is a Linux distribution that's
compos…   4495      [OK]
...
```

```
root@docker:~# docker pull debian
Using default tag: latest
latest: Pulling from library/debian
a8ca11554fce: Pull complete
Digest:
sha256:3066ef83131c678999ce82e8473e8d017345a30f5573ad3e44f62e5c9c46442b
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

2. Ejecutar comando `echo` en contenedor con imagen `debian`

- reemplaza al comando por defecto (`bash`) de la imagen `debian`
- diferencias `docker run` *vs* `docker create + docker start`

```
root@docker:~# docker run --name prueba1 debian echo "Hola CDA"
Hola CDA

root@docker:~# docker create --name prueba2 debian echo "Hola CDA"
6d236b2c49b945348c7b028be0c2c6517548a42369914d62fd0dc547107a1bfc

root@docker:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND           CREATED          STATUS
           PORTS     NAMES
6d236b2c49b9   debian    "echo Hola CDA"   5 seconds ago    Created
                    prueba2
91bd10b4c528   debian    "echo Hola CDA"   19 seconds ago   Exited (0) 18
seconds ago             prueba1

root@docker:~# docker start -a prueba2
Hola CDA

root@docker:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND           CREATED              STATUS
                   PORTS     NAMES
6d236b2c49b9   debian    "echo Hola CDA"   49 seconds ago       Exited (0) 5
seconds ago                 prueba2
91bd10b4c528   debian    "echo Hola CDA"   About a minute ago   Exited (0)
About a minute ago            prueba1

root@docker:~# docker start -a prueba2
Hola CDA

root@docker:~# docker start -a prueba2
Hola CDA
```

```
root@docker:~# docker container prune        # elimina contenedores parados
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
6d236b2c49b945348c7b028be0c2c6517548a42369914d62fd0dc547107a1bfc
91bd10b4c5287453cead93c6fcf0c27934b7642b32132add5a4569d1591244d7

Total reclaimed space: 0B
```

- Acceso interactivo (`-it`) a un contenedor Debian (se ejecuta el comando por defecto `bash`)
  - Comprobar con `uname -a` que coincide el kernel de anfitrión y contenedores
  - Acceso interactivo (con `-it`) en `docker run`: diferencia entre salir con `exit` (terminal el comando por defecto `bash` y finaliza el contenedor) y salir con "secuencia de escape" `[Contro]+pq`

```
root@docker:~# uname -a
Linux datos.cda.net 5.10.0-16-amd64 #1 SMP Debian 5.10.127-1 (2022-06-
30) x86_64 GNU/Linux

root@docker:~# docker run -it --name debian1 --hostname debian1 debian
root@debian1:/# hostname
debian1
root@debian1:/# uname -a
Linux debian1 5.10.0-16-amd64 #1 SMP Debian 5.10.127-1 (2022-06-30)
x86_64 GNU/Linux
root@debian1:/#                # para salir pulsar [Control]+pq
root@debian2:/# read escape sequence

root@docker:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS
PORTS     NAMES
bcf594966076   debian    "bash"    34 seconds ago   Up 32 seconds
      debian1

root@docker:~# docker run -it -d --name debian2 --hostname debian2
debian
669769bc4478c41e8270ad8f53d54df3f85ea94acba655784f0e482ec3a912c8

root@docker:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED              STATUS
   PORTS     NAMES
669769bc4478   debian    "bash"    3 seconds ago        Up 2 seconds
             debian2
bcf594966076   debian    "bash"    About a minute ago   Up About a
minute             debian1

root@docker:~# docker attach debian2
root@debian2:/# hostname
debian2
root@debian2:/# uname -a
Linux debian2 5.10.0-16-amd64 #1 SMP Debian 5.10.127-1 (2022-06-30)
x86_64 GNU/Linux
root@debian2:/#                    # para salir pulsar [Control]+pq
root@debian2:/# read escape sequence

root@docker:~# docker exec debian2 echo "Hola otra vez"
```

```
Hola otra vez

root@docker:~# docker exec -it debian2 /bin/bash
root@debian2:/# hostname
debian2
root@debian2:/# uname -a
Linux debian2 5.10.0-16-amd64 #1 SMP Debian 5.10.127-1 (2022-06-30)
x86_64 GNU/Linux
root@debian2:/# exit
exit
root@docker:~#
```

- Comprobación de las conexiones a la red por defecto `bridge` (172.17.0.0/16) en anfitrion (interface `docker0` con IP 172.17.0.1) y contenedores (requiere instalar los comandos `ip` y `ping` no incluidos en la imagen Docker)

```
root@docker:~# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP group default qlen 1000
    link/ether 08:00:27:11:11:11 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 scope global enp0s3
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:f1:42:3c:d6 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
...
```

```
root@docker:~# docker exec -it debian1 /bin/bash

root@debian1:/# apt update
root@debian1:/# apt install iproute2 iputils-ping
root@debian1:/# ip address
root@debian1:/# ping 172.17.0.1     # ping al anfitrion
root@debian1:/# ping 172.17.0.3     # ping a debian2
root@debian1:/# exit
exit
```

```
root@docker:~# docker exec -it debian2 /bin/bash
root@debian2:/# apt update
root@debian2:/# apt install iproute2 iputils-ping
root@debian2:/# ip address
root@debian2:/# ping 172.17.0.1
root@debian2:/# ping 172.17.0.2
root@debian2:/# exit
exit
```

Parar contenedores y eliminarlos

```
root@docker:~# docker stop debian1 debian2
root@docker:~# docker rm debian1 debian2
root@docker:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Ejemplo 2: Redirección de puertos (`-p`) y compartición con *mount bind* (`--mount`)

- Crear contenedores con redirecciones de puertos y un directorio compartido mediante *mount bind* (debe existir antes de configurar el contenedor)

```
root@docker:~# mkdir comun_html

root@docker:~# docker run -it -d --name apache1 \
                          --hostname apache1 -d -p 8081:80 \
                          --mount
type=bind,src=$PWD/comun_html,dst=/var/www/html debian

root@docker:~# docker run -it -d --name apache2 \
                           --hostname apache2 -d -p 8082:80 \
                           --mount
type=bind,src=$PWD/comun_html,dst=/var/www/html debian
```

- Acceso e instalación de herramientas (Apache y PHP)

```
root@docker:~# docker exec -it apache1 /bin/bash
root@apache1:~# apt update
root@apache1:~# apt install apache2 libapache2-mod-php php iproute2 iputils-
ping
root@apache1:~# /etc/init.d/apache2 start
```

```
root@docker:~# docker exec -it apache2 /bin/bash
root@apache2:~# apt update
root@apache2:~# apt install apache2 libapache2-mod-php php iproute2 iputils-
ping
root@apache2:~# /etc/init.d/apache2 start
```

- Contenido PHP compartido (en directorio `comun_html` del anfitrión)

```
root@docker:~# cd comun_html/
root@docker:~/comun_html# rm index.html
root@docker:~/comun_html# echo "Servido por: <?php echo gethostname(); ?>" >
index.php
```

- Prueba de funcionamiento
  Desde el anfitrión abrir en un navegador la URL http://localhost:8081 (responderá `apache1`) y a continuación la URL http://localhost:8082 (responderá `apache2`).
- Ejemplo de redes definidas por el usuario (nueva red `red_pruebas` con `apache2` conectado a 2 redes)

```
root@docker:~# docker network create --subnet 10.10.10.0/24 --gateway
10.10.10.1 red_pruebas
2a8b5c99419429b2bb7ab0c75ee8cc64d3e9e0163fc8a1061e84745f981c0166
root@docker:~# docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
4f513bbffc81   bridge        bridge    local
e032c20d4df3   host          host      local
36879c11aa27   none          null      local
2a8b5c994194   red_pruebas   bridge    local
```

```
root@docker:~# docker run -it -d -rm --name tercero \
                         --network red_pruebas --hostname pruebas debian

root@docker:~# docker network connect red_pruebas apache2
root@docker:~# docker network inspect red_pruebas
```

```
[  {    "Name": "red_pruebas",
        "Id": "2a8b5c99419429b2b...66",
        ...
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [ {
                    "Subnet": "10.10.10.0/24",
                    "Gateway": "10.10.10.1"
                     } ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9bf9f9aa77a028...64": {
                "Name": "apache2",
                "EndpointID": "c6e887f546ff82c..88f5",
                "MacAddress": "02:42:0a:0a:0a:03",
                "IPv4Address": "10.10.10.3/24",
                "IPv6Address": ""
            },
            "dff0633b46408a...6e": {
                "Name": "tercero",
                "EndpointID": "b439f5cb6954f5a...8c4",
                "MacAddress": "02:42:0a:0a:0a:02",
                "IPv4Address": "10.10.10.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }  ]
```

```
root@docker:~# docker exec -it apache2 /bin/bash
```

```
root@apache2:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
50: eth0@if51: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
55: eth1@if56: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default
    link/ether 02:42:0a:0a:0a:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.10.10.3/24 brd 10.10.10.255 scope global eth1
       valid_lft forever preferred_lft forever
root@apache2:/# ping 10.10.10.1
root@apache2:/# ping 10.10.10.2
root@apache2:/# exit
exit
```

Parar y eliminar contenedores y redes creadas

```
root@docker:~# docker stop apache1 apache2 tercero
root@docker:~# docker rm apache1 apache2 tercero
root@docker:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

root@docker:~# docker network rm red_pruebas
root@docker:~# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
4f513bbffc81   bridge    bridge    local
e032c20d4df3   host      host      local
36879c11aa27   none      null      local
```

Ejemplo 3: Uso conjunto de contenedores y redes (MariaDB + PHPMyAdmin)

Imagen oficial de MariaDB: https://hub.docker.com/_/mariadb

- La imagen de MariaDB asume que el directorio con los datos (`/var/lib/mysql`) sea proporcionado desde el anfitrión, bien con un volumen Docker (opción `-v`) o con *bind mount* (opción `--mount`). De esa forma, los datos almacenados no se perderán a detener el contenedor y estarán disponibles si se vuelve a arrancar.

Imagen oficial de PHPMyAdmin: https://hub.docker.com/_/phpmyadmin

```
root@docker:~# mkdir datos_mariadb

root@docker:~# docker network create red_mysql
395f4f3599e14935b2efdf0f89be0eb382827e66cbf0a04f96e53ec480974c5d

root@docker:~# docker run -d --name servidor_bd  \
```

```
                          --network red_mysql \
                          --mount
type=bind,src=$PWD/datos_mariadb,dst=/var/lib/mysql \
                          -e MARIADB_ROOT_PASSWORD=purple mariadb:latest

root@docker:~# docker run -d --name phpmyadmin \
                          --network red_mysql \
                          -e PMA_HOST=servidor_bd \
                          -p 8080:80   phpmyadmin:latest
Unable to find image 'phpmyadmin:latest' locally
latest: Pulling from library/phpmyadmin
a603fa5e3b41: Pull complete
...
Digest: sha256:0f0e89603ad8072e5e21bb684dc6a7b47cc31fc1542736b15c97c627e4c5b841
Status: Downloaded newer image for phpmyadmin:latest
df00e4c6743651d82243d76a3e07c8d056560c5ea2b735d4463805437145b543
```

- Desde el anfitrión acceder con un navegador a la URL http://localhost:8080.
- Conectarse a *PHPMyAdmin* con usuario `root` y contraseña `purple`.
- Al terminar, parar y eliminar contenedores y redes utilizados

```
root@docker:~# docker stop servidor_bd phpmyadmin
root@docker:~# docker rm servidor_bd phpmyadmin
root@docker:~# docker ps -a

root@docker:~# docker network rm red_mysql
root@docker:~# docker network ls
```

Ejercicio entregable.

Hacer manualmente un despliegue de contenedores Docker con las siguientes caracterísiticas:

- un contenendor HAProxy haciendo de frontal (con el puerto 80 de ese contenedor redirigido al puerto 8080 del anfitrión para verificar el acceso)
- dos contenedores con el gestor de contenidos *Wordpress* instalado sobre los que repartirá las peticiones el balanceador HAProxy
- un contenedor con una base de datos MariaDB compartida por los dos *Wordpress*.

Documentación de las imágenes a utilizar (disponibles en Docker Hub):

- Imagen oficial de MariaDB: https://hub.docker.com/_/mariadb
  - La imagen espera que el directorio con los datos (`/var/lib/mysql`) sea proporcionado desde el anfitrión, bien con un volumen Docker (opción `-v`) o con *bind mount* (opción `--mount`)
  - En este caso, además de usar la variable de entorno `MARIADB_ROOT_PASSWORD` como en el ejemplo anterior es necesario usar las variables `MARIADB_USER`, `MARIADB_PASSWORD` y `MARIADB_DATABASE` (ver sección *Environment Variables*) para crear la BD que usará *Wordpress*, el usuario y la contraseña con la que accede a la Base de Datos.
- Imagen oficial de Wordpress: https://hub.docker.com/_/wordpress
  - La imagen espera que el directorio donde *Wordpress* almacena los contenidos subidos por los usuarios (`/var/www/html/wp-content`) sea proporcionado desde el anfitrión,

bien con un volumen Docker (opción `-v`) o con *bind mount* (opción `--mount`). De esta forma, además de que los datos permanezcan entre paradas de los contenedores, los dos contenedores *Worddpress* compartirán el mismo contenido.
    - Es necesario proporcionar mediante las variables de entorno `WORDPRESS_DB_HOST`, `WORDPRESS_DB_USER`, `WORDPRESS_DB_PASSWORD` y `WORDPRESS_DB_NAME` los parámetros para conectar con la BD (que deberán ser consistentes con los indicados en el servidor MariaDB). (ver sección *How to use this image* en la documentación de la imagen)
- Imagen oficial de HAProxy: https://hub.docker.com/_/haproxy
    - Esta imagen apenas tiene configuración externa.
        - IMPORTANTE 1: El HAProxy de esta imagen espera la configuración en `/usr/local/etc/haproxy` no en `/etc/haproxy` como en la Práctica 5.
        - IMPORTANTE 2: El HAProxy de esta imagen se ejecuta bajo un usuario `haproxy` sin privilegios, para poder poner el balanceador en escucha en el puerto 80, debe añadirse al contenedeor la opción `--sysctl net.ipv4.ip_unprivileged_port_start=0` (ver explicación en sección *Run container* en la documentación de la imagen)
    - La forma más sencilla de aportar el fichero de configuración `haproxy.cfg` es mediante un volumen Docker o un *bind mount* que proporcione el contenido del directorio `/usr/local/etc/haproxy` (como se indica en la sección *Directly via bind mount*).
        - IMPORTANTE 3: El fichero de configuración `haproxy.cfg` debe existir antes de crear y arrancar el contenedor con `docker run`
        - Respecto al contenido del fichero `haproxy.cfg` se puede reutilizar el de la Práctica 5:
            - indicado en el parámetro `bind` del `frontend` la dirección `0.0.0.0:80` (escuchará en todas las IPs del contenedor en lugar de en una concreta como en el ejemplo de la Práctica 5)
            - ajustando las IPs de los `server` del `backend`, usando en su lugar los nombres que se se hayan puesto a los dos contenedores *Wordpress*.
    - Para verificar desde el anfitrión el funcionamiento del balanceador y los demás contenedores se debará habilitar una redirección del puerto 8080 del anfitrión al puerto 80 del contenendor HAProxy.
        - basta con verificar que se accede a *Wordpress* (mostrando una captura del resultado), no es necesario configurar la herramienta.

NUEVO: Pasos/pistas (el orden es relevante)

1. Crear en el anfitrión los directorios a usar con `--mount`
    - para `/var/lib/mysql` del contenedor *MariaDB*
    - para `/var/www/html/wp-content` de los contenedores *Wordpress*
    - para `/usr/local/etc/haproxy` del contenedor *HAProxy*
2. Crear el fichero de configuración de *HAProxy* (`haproxy.cfg`)
    - en `frontend` ajustar `bind` a `0.0.0.0:80` ó `*:80`
    - en `backend` ajustar la dirección de los `server` usando los nombres de los contenedores *Wordpress*
3. Crear las redes necesarias
4. Crear los contenedores necesarios, ajustando las variables de entorno (con `-e`) según corresponda
5. Establecer las conexiones de red adicionales

Nota: La instalación final de *Wordpress* a partir de la imagen Docker oficial necesitaría configuración adicional para que funcione correctamente con la redirección de puertos utilizada (`-p 8080:80`)(ver https://www.cloudytuts.com/tutorials/docker/how-to-solve-wordpress-redirects-to-localhost-8080/). Para este ejemplo basta con verificar que se tiene acceso.

- Se puede completar el asistente de instalación que aparece en el primer acceso a `localhost:80080` con valores arbitrarios y completar la instalación:
  - `Site Title: pruebaCDA`
  - `Username: prueba`
  - `Password: $PurPle2022$`
  - `Your Email: prueba@cda.net`
  - Pulsar en `[Install Wordpress]`
- Debido al uso de la redirección al puerto `8080` el proceso de *login* no funcionará adecuadamente.
- Si se vuelve a acceder a `localhost:8080` se verá la portada del blog por defecto.

Entregable

- Entrega individual, en MOOVI hasta 23/12/2022
- Incluir una explicación de los elementos del entorno que se han creado (redes y contenedores, redirecciones, volúmenes o *bind mounts*, etc), bien de forma textual, con esquemas o combinando ambas formas. Comenar los aspectos más relevantes de su configuración (redirecciones, *mounts*, etc).
- Aportar los comandos utilizados, la salida que dan (si es relevante) y una breve descripción de lo que hace cada uno.
- Añadir capturas del resultado final una vez se acceda mediante un navegador al anfitrión con la URL http://localhost:8080
  - **NOTA:** No es necesario completar la instalación de los *Wordpress* basta con constatar que se muestra la página de configuración.