



Centros de Datos

P6 - Contenedores con Docker

David Ruano Ordás
Departamento de Informática

✉ drordas@uvigo.es

📍 Despacho 409

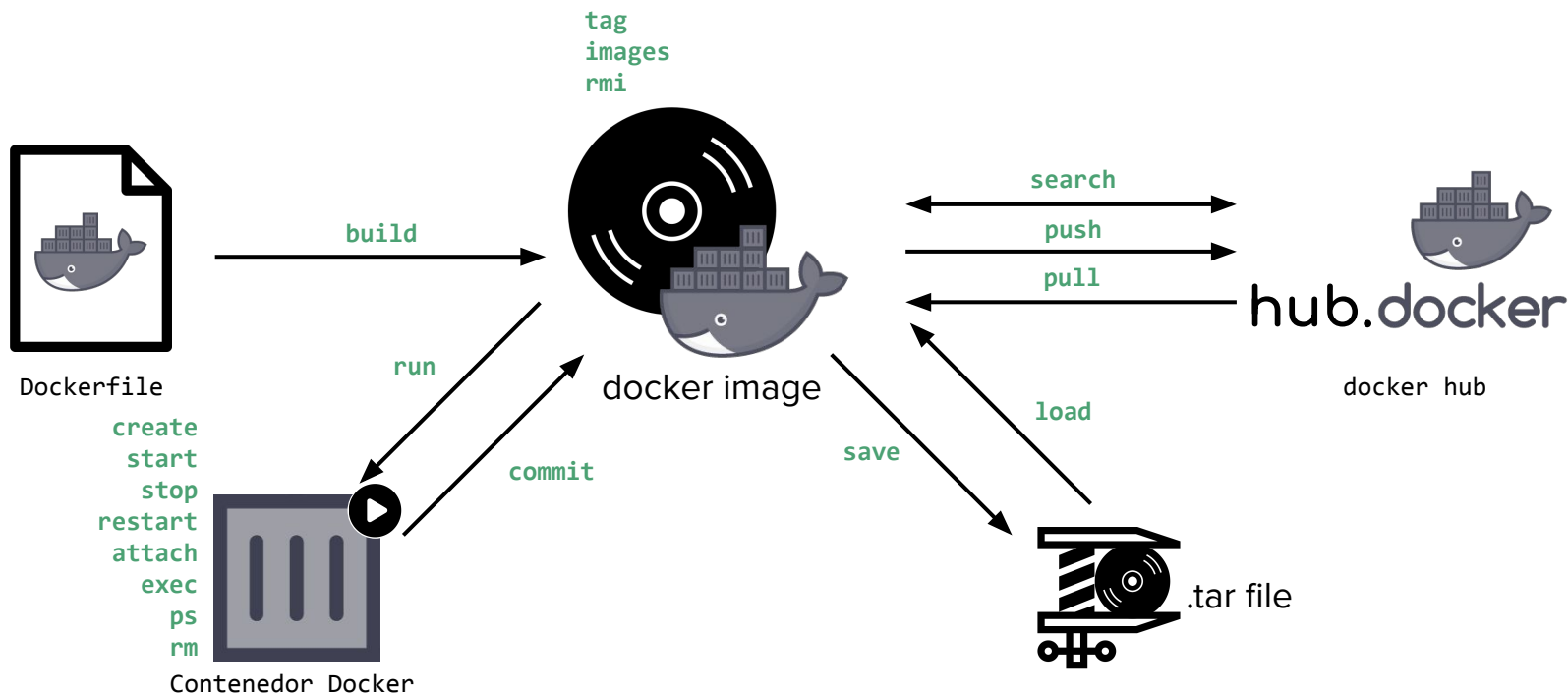
Contenidos

- Estructura y comandos básicos
 - Esquema general
 - Comandos con Docker
 - Creación de contenedores.
 - Gestión de contenedores.
 - “*Mapeo*” de puertos.
 - “*Mapeo*” de volúmenes y mount bind
 - Linking
 - Redes
- Entregable

Docker

- Estructura y comandos básicos:

- Esquema general:



Docker

- **Estructura y comandos básicos:**

- **Comandos con Docker:**

- **Gestión de imágenes (I):**

- docker **search**: **busca imágenes** existentes **en un registro de imágenes** (Docker Hub o privado).

```
~#docker search debian
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	16590	[OK]	
debian	Debian is a Linux distribution that's compos...	4847	[OK]	
.....				

- docker **pull**: **descarga imágenes** de Docker **desde un registro de imágenes** (Docker Hub o privado).

```
~#docker pull debian:latest
latest: Pulling from library/debian
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

- docker **push**: **carga imágenes** Docker a un **registro de imágenes** (Docker Hub o privado).
 - docker **images**: **lista las imágenes** Docker **existentes** en el **equipo**.

```
~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian	latest	e4c58958181a	6 weeks ago	77.8MB
owncloud/server	10.11	304d2aa03b61	12 months ago	1.06GB
wordpress	6.1.0-fpm-alpine	39d6a0c052ff	12 months ago	302MB
...	...	...	...	...

Docker

- Estructura y comandos básicos:

- Comandos con Docker:

- Gestión de imágenes (II):

- docker **tag**: renombra una imagen de Docker existente (mantiene la imagen original).

```
~#docker tag debian:latest renamed:latest
~#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
renamed             latest             e4c58958181a       6 weeks ago        77.8MB
debian              latest             e4c58958181a       6 weeks ago        77.8MB
.....             .....             .....             ....              .....
```

- docker **rmi**: elimina una imagen de Docker

```
~#docker rmi renamed:latest
~#docker images
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest             e4c58958181a       6 weeks ago        77.8MB
```

- docker **save**: guarda una (o varias) imágenes Docker en un **fichero ".tar"**

```
~#docker save debian:latest -o mi_debian.tar
~#ls -lah mi_debian.tar
-rw----- 1 root root 116M nov 16 18:34 mi_debian.tar
```

- docker **load**: carga una (o varias) imágenes Docker a **partir** de un **fichero ".tar"**

```
~#docker load -i mi_debian.tar
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest             fd7391105260       2 weeks ago        117MB
```

Docker

- Estructura y comandos básicos:

- Comandos con Docker:

- Gestión de contenedores (I):

- docker **create**: crea un nuevo contenedor basado en una imagen sin iniciarlo (estado **detenido**).

The diagram shows the command `~#docker create --name debian9 -it debian:9 /bin/bash` with several annotations:

- A blue arrow points from `debian9` to the text "nombre del contenedor".
- A brown arrow points from `debian:9` to the text "imagen".
- A purple arrow points from `/bin/bash` to the text "comando a ejecutar dentro del contenedor (intérprete de comandos)".
- A red arrow points from `-it` to a red text box containing: "Ejecutar comandos y ver la salida en tiempo real", "-i (modo interactivo): Habilita entrada estándar (stdin).", and "-t (tty): asigna una terminal al contenedor."

- docker **start**: ejecuta un contenedor que ha sido **previamente creado** (estado **detenido**).

```
~#docker start debian9
```

- docker **attach**: accede al contenedor que está en ejecución.

```
~#docker attach debian9
```

- docker **run**: crea e inicia un contenedor basado en una **imagen**.

```
~#docker run --name debian9 -it debian:9 /bin/bash
```

docker run =
docker create
docker start
docker attach

- docker **stop**: para la ejecución de un contenedor (sin eliminarlo)

```
~#docker stop debian9
```

Docker

- **Estructura y comandos básicos:**

- **Comandos con Docker:**

- **Gestión de contenedores (II):**

- docker **ps**: lista los **contenedores** en **ejecución**

```
~#docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
c3f315b033fa   debian:9  "/bin/bash"             16 minutes ago  Up 4 seconds          debian9
```

```
~#docker ps -a → LISTAR TODOS LOS CONTENEDORES EXISTENTES
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
c3f315b033fa   debian:9  "/bin/bash"             17 minutes ago  Up 4 seconds          debian9
91a0fb229864   hello-world  "/hello"                3 years ago     Exited (0)          hello
```

- docker **rm**: **elimina** un **contenedor** existente.

```
~#docker rm hello
hello
```

- docker **restart**: **reinicia** un **contenedor** en **ejecución**.

```
~#docker restart debian9
debian9
```

- docker **exec**: **ejecuta comandos** dentro de un **contenedor** en **ejecución**

```
~#docker exec ls debian9
bin
boot
dev
...
```

Docker

- “Mapeo” de puertos

```
docker run --name ssh_server -it -p 22:22 debian:9 /bin/bash
root@d2b318dfef64:/# apt update && apt install ssh → INSTALAR SERVIDOR SSH
root@d2b318dfef64:/# adduser user → CREAR UN USUARIO
root@d2b318dfef64:/# service ssh start → INICIALIZAR EL SERVIDOR SSH
^pq → SALIR DEL CONTENEDOR
```

Diagram illustrating port mapping for an SSH server container. Red arrows point from the host ports (22) to the container ports (22). The container is named `ssh_server` and runs `debian:9`. The commands inside the container are: `apt update && apt install ssh` (INSTALLAR SERVIDOR SSH), `adduser user` (CREAR UN USUARIO), and `service ssh start` (INICIALIZAR EL SERVIDOR SSH). The prompt `^pq` indicates exiting the container.

- “Mapeo” de volúmenes

```
docker run --name apache_server -it -v /home/usuario/html_page:/var/www/html -p 80:80 debian:9 /bin/bash
root@f84175afeff9:/# apt update && apt install -y apache2 → INSTALAR SERVIDOR APACHE2
root@f84175afeff9:/# service apache2 start && ps aux | grep apache2 → INICIALIZAR EL SERVIDOR APACHE2 Y VERIFICAR SU ESTADO
^pq → SALIR DEL CONTENEDOR
links 127.0.0.1:80 → ACCEDER A LA WEB
```

Diagram illustrating volume and port mapping for an Apache server container. Red arrows point from the host volume (`/home/usuario/html_page`) to the container volume (`/var/www/html`). Purple arrows point from the host port (80) to the container port (80). The container is named `apache_server` and runs `debian:9`. The commands inside the container are: `apt update && apt install -y apache2` (INSTALAR SERVIDOR APACHE2), `service apache2 start && ps aux | grep apache2` (INICIALIZAR EL SERVIDOR APACHE2 Y VERIFICAR SU ESTADO), and `links 127.0.0.1:80` (ACCEDER A LA WEB). The prompt `^pq` indicates exiting the container.

- Bind mount

```
mkdir /home/usuario/html_page
echo "<html><body><h1>Hola, Docker!</h1></body></html>" > /home/usuario/html_page → HTML SE COPIA EN PATH A MONTAR
docker run -d --name apache_server -p 8080:80 --mount type=bind,source=/home/usuario/html_page,target=/var/www/html httpd:latest
links 127.0.0.1:8080 → ACCEDER A LA WEB
```

Diagram illustrating a bind mount for an Apache server container. A purple arrow points from the host path (`/home/usuario/html_page`) to the container path (`/var/www/html`). The container is named `apache_server` and runs `httpd:latest`. The command inside the container is: `echo "<html><body><h1>Hola, Docker!</h1></body></html>" > /home/usuario/html_page` (HTML SE COPIA EN PATH A MONTAR). The prompt `links 127.0.0.1:8080` indicates accessing the web.

Docker

- **Linking**

VARIABLE DE ENTORNO: CONTRASEÑA BD

PUERTO HOST

PUERTO CONTENEDOR

```
~#docker run --name mysql_server -e MYSQL_ROOT_PASSWORD=cda -p 3306:3306 mysql:latest
~#docker run --name web_server -it -p 80:80 --link mysql_server:db httpd:latest
```

CONTENEDOR AL QUE ASOCIARSE

AÑADIR UN ALIAS

ENTRADA CON LA IP DEL CONTENEDOR MYSQL PARA DB.
PETICIONES A DB SERÁN RESUELTAS A ESA IP.

ENTRADA CON LA IP DEL PROPIO CONTENEDOR

root@d43235drfet5:/# cat /etc/hosts
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
....
172.17.0.2 db 7962654f2564 db
172.17.0.4 d43235drfet5

- **Redes**

- docker **network create**: crea una red Docker

RED **cda_network** de tipo birge (por defecto) se le asocia un ID determinado

```
~#docker network create cda_network
493dc12fe6e83772a66f4f8db61e10857be7328133eb9bf5edd24af0438efd4d

~#docker network ls
NETWORK ID        NAME          DRIVER  SCOPE
4a92d3856bb5     bridge       bridge  local
493dc12fe6e8     cda_network  bridge  local
....             ....         ...     ...
```

Especificar el tipo de red:

- driver brigde
- driver host

Docker

- **Redes**

- docker **network inspect**: muestra **información** sobre la **configuración** de una **red**.

```
~#docker network inspect cda_network
[
  {
    "Name": "cda_network",
    "Id": "dbb534d5ae88f7b05891cd17ec1d87663dd33cef8da7e490df4d3f0f6b8b0dba",
    "Created": "2023-11-17T14:37:35.699732058+01:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.20.0.0/16",
          "Gateway": "172.20.0.1"
        }
      ]
    },
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

Docker

- **Redes**

- docker **network connect**: **conecta** un **contenedor** a una **red** previamente creada

```
~# docker run -d nginx_server nginx:latest
2e76aa9443c85be38027689575f442df9df956556c81bafdd58ce5707c2bac95

~#docker network connect cda_network nginx_server
~#docker network inspect cda_network
.....
"Containers": {
  "2e76aa9443c85be38027689575f442df9df956556c81bafdd58ce5707c2bac95": {
    "Name": "nginx_server",
    "EndpointID": "75320cc1d90d9d215556732bd5bbc84e31d599d6f3f2bcf651310a5bd79f4599",
    "MacAddress": "02:42:ac:14:00:03",
    "IPv4Address": "172.20.0.3/16",
    "IPv6Address": ""
  }
},
```

- docker **network disconnect**: **desconecta** un **contenedor** de una **red**.

```
~#docker network disconnect cda_network nginx_server
.....
"Containers": {},
```

Docker

- **Redes**

- docker **network rm:** elimina una determinada red.

```
~#docker network rm cda_network
cda_network
```

```
~#docker network ls
NETWORK ID      NAME                DRIVER  SCOPE
4a92d3856bb5    bridge             bridge  local
....           ....              ...    ...
```

- docker **network purge:** elimina todas las **redes** que no estén siendo **usadas** por **algún contenedor**.
- docker **network ls:** lista todas las redes disponibles.

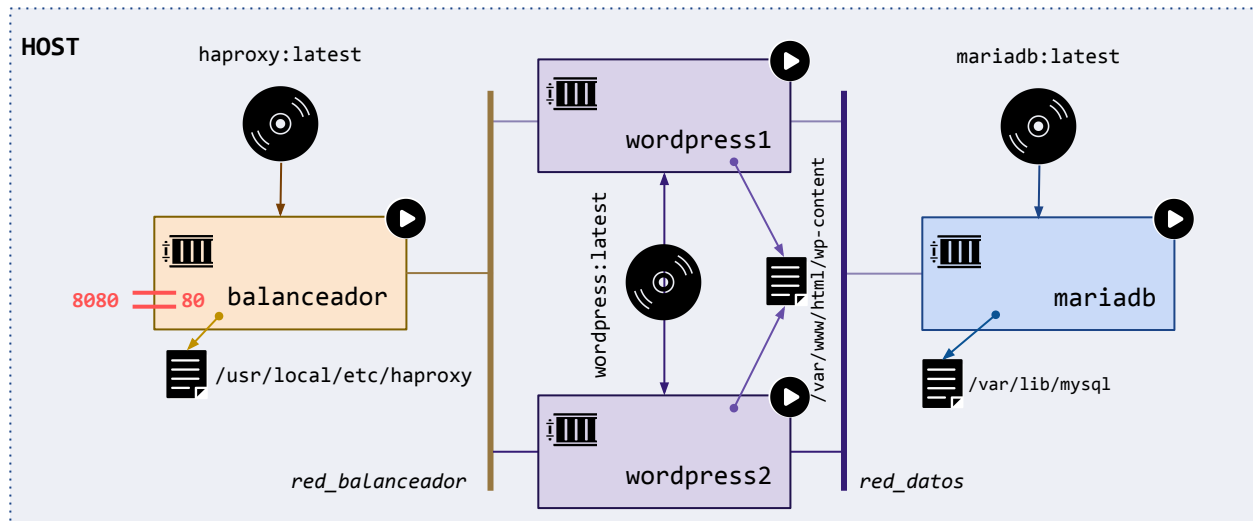
```
~#docker network ls
NETWORK ID      NAME                DRIVER  SCOPE
4a92d3856bb5    bridge             bridge  local
8ab4f29183a8    host               host    local
f49d7f068d35    none               null    local
```

Despliegue contenedores Docker



Entregable: Despliegue de balanceador de Carga con servidores Wordpress usando contenedores Docker

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



Despliegue contenedores Docker



Entregable: Despliegue de balanceador de Carga con servidores Wordpress usando contenedores Docker

4. Pasos:
 - a. Crear en el anfitrión los directorios a usar con -v (volume mapping)
 - 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
 - b. 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
 - c. Crear las redes necesarias
 - d. Crear los contenedores necesarios, ajustando las variables de entorno (con -e) según corresponda
 - e. Establecer las conexiones de red adicionales
5. Documentación a entregar:
 - a. 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.
 - b. Comentar los aspectos más relevantes de su configuración (redirecciones, mounts, etc).
 - c. Aportar los comandos utilizados, la salida que dan (si es relevante) y una breve descripción de lo que hace cada uno.
 - d. Añadir capturas del resultado final una vez se acceda mediante un navegador al anfitrión con la URL `http://localhost:8080`
 - e. NOTA: No es necesario completar la instalación de los Wordpress basta con constatar que se muestra la página de configuración.

FECHA DE ENTREGA: 24/12/2023 [23:59]!!



Centros de Datos

P6 - Contenedores con Docker

David Ruano Ordás
Departamento de Informática

✉ drordas@uvigo.es

📍 Despacho 409