



# ADMINISTRACIÓN DE INFRAESTRUCTURAS Y SISTEMAS INFORMÁTICOS (AISI)

Grado en Ingeniería Informática

Grado en Ingeniería Informática

Roberto R. Expósito ([roberto.rey.exposito@udc.es](mailto:roberto.rey.exposito@udc.es))

# PRÁCTICA 2

## Docker



# Objetivo

3

- El propósito de esta práctica es aprender los conceptos básicos de **Docker**, una tecnología de virtualización por compartición de *kernel* basada en entornos virtuales de ejecución llamados **contenedores**
  - Se verán conceptos relacionados con la creación de imágenes Docker, así como la ejecución, configuración y gestión de contenedores
  - Se propondrán ejemplos sencillos de despliegue de aplicaciones usando contenedores mediante Docker Compose y Docker Swarm



<https://www.docker.com>



# Justificación de la práctica

4

- La **entrega** de la práctica consistirá en un **breve documento** en formato **PDF** que incluya las **todas capturas de pantalla** mostradas en las transparencias:
  - 9, 11 (EJ1); 15 (EJ2); 18 (EJ3); 23, 24 (EJ4); 31, 32 (EJ5); 37, 40 (EJ6); 42, 45, 46 (EJ7)**



Para ayudar a identificarlas, estas transparencias incluyen el icono de un monitor en la parte superior derecha



**IMPORTANTE**





- En ocasiones, durante la práctica, se pide crear recursos con un nombre que empieza por un **prefijo** que contiene información del estudiante y del curso
- ES OBLIGATORIO** usar la siguiente nomenclatura para nombrar los recursos:  
*<iniciales del nombre y apellidos><curso>-<nombre del recurso>*
  - Ejemplo: El alumno Roberto Rey Expósito, que hace la práctica en el curso 2024/2025, utilizará el siguiente prefijo: **rre2425**
- NO RECORTES** las capturas de pantalla, **debe verse toda la información** que sea relevante para comprobar el trabajo realizado
- NO** seguir estas normas **IMPLICA UNA CALIFICACIÓN “C”** en la práctica



# Consideraciones iniciales

5

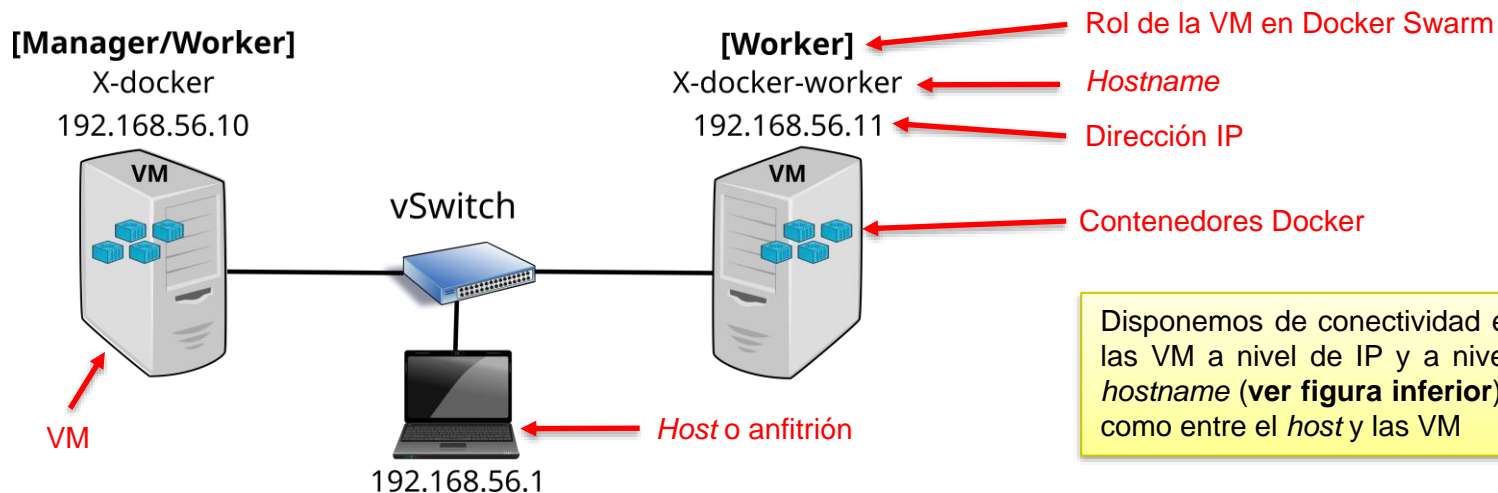
- Clona el [repositorio de la práctica 2](#) para obtener los ficheros necesarios para realizar los ejercicios propuestos
  - **Recuerda:** sin espacios, acentos, eñes o caracteres “raros” en la ruta 
- **Debes usar tu Vagrant box** creado con Packer en la práctica previa junto con el *Vagrantfile* disponible en el repositorio de esta práctica 2
  - Este *Vagrantfile* despliega **dos VM** (*manager/worker*) interconectadas formando un mini clúster que usaremos con Docker Swarm (ejercicios 6 y 7)
    - La siguiente transparencia muestra el esquema del despliegue
  - **Modifica el *Vagrantfile* para cambiar el box y el hostname de las VM**
    - **Modifica la variable *STUDENT\_PREFIX* (línea 7)** 
    - Debes sustituir **X** por tu **prefijo**
- Recuerda que la carpeta del proyecto Vagrant (donde reside el fichero *Vagrantfile*) se comparte con la ruta **/vagrant** en las VM
  - Resulta muy útil para poder editar ficheros desde el *host* y acceder a ellos desde las VM



# Consideraciones iniciales

6

- Esquema gráfico del despliegue con Vagrant:



- En los cuatro primeros ejercicios se usará únicamente la *VM manager*
  - Para conectarte a ella ejecuta: *vagrant ssh* (o *vagrant ssh manager*)

Ping desde la  
VM manager a  
la VM worker

```
vagrant@rre2425-docker:~$ ping -c 2 rre2425-docker-worker
PING rre2425-docker-worker (192.168.56.11) 56(84) bytes of data.
64 bytes from rre2425-docker-worker (192.168.56.11): icmp_seq=1 ttl=64 time=1.59 ms
64 bytes from rre2425-docker-worker (192.168.56.11): icmp_seq=2 ttl=64 time=0.914 ms

--- rre2425-docker-worker ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.914/1.252/1.590/0.338 ms
vagrant@rre2425-docker:~$
```



# Ejercicio 1

7

- **Creación de un fichero *Dockerfile* para generar una imagen que ejecute un servidor Redis en un contenedor Docker**
  - [Redis](#) es un motor de base de datos NoSQL en memoria implementado en C basado en el almacenamiento en tablas de *hashes* (pares de tipo clave-valor)
- **Completa la plantilla *Dockerfile* proporcionada en los ficheros del ejercicio:**
  - Como imagen base (instrucción FROM) usa **debian:bookworm-slim**
  - Descarga y compila el código fuente de Redis
    - Establece el directorio de trabajo por defecto (WORKDIR) a la ruta `/tmp`
    - Descarga el código fuente de Redis usando: `ADD <URL> redis.tar.gz`
      - URL de descarga: `https://download.redis.io/releases/redis-7.4.2.tar.gz`
    - Descomprime el código fuente descargado en un directorio llamado *redis*
      - `mkdir redis && tar -xzf redis.tar.gz -C redis --strip-components=1`
    - Instala las dependencias necesarias para su compilación (paquetes: **gcc, make**)
    - Compila el código e instala los binarios: `make -C redis && make -C redis install`
  - Minimiza el tamaño de la imagen resultante, eliminando:
    - El fichero descargado (*redis.tar.gz*) y el directorio *redis* con el código fuente
    - Los paquetes instalados y sus dependencias: `apt-get purge -y --autoremove gcc make`
    - La caché local del gestor de paquetes: `apt-get clean all`



# Ejercicio 1

8

- Completa la plantilla *Dockerfile* disponible en el repositorio (cont.):
  - Usando ENV, define la variable de entorno *REDIS\_DATA* con el valor */data*
  - Cambia el directorio de trabajo (WORKDIR) a *\$REDIS\_DATA*
    - Dicho directorio debe pertenecer al usuario y grupo redis: *chown redis:redis \$REDIS\_DATA*
    - Debes crear previamente el usuario y el grupo redis:
      - *groupadd -r redis && useradd -r -g redis redis*
  - Establece el usuario que ejecutará el contenedor (instrucción USER) como redis
  - Expón el puerto TCP 6379 (puerto por defecto donde escucha el servidor Redis)
  - Establece el **punto de entrada** usando formato exec como: *redis-server*
    - Añade al binario *redis-server* los parámetros “*--protected-mode no*” y “*--save 5 1*”
- Crea la imagen (*docker build*) usando tu fichero *Dockerfile*
  - **Debes nombrar la imagen como: X-redis**
  - Sustituye **X** por tu **prefijo**
- Una vez creada, lista las imágenes Docker disponibles: *docker image ls*
- Consulta el “historial” de tu imagen: *docker history*







# Ejercicio 1

9



```
vagrant@rre2425-docker:~$ docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
rre2425-redis       latest         58fc63a23666   2 minutes ago  130MB

vagrant@rre2425-docker:~$ docker history rre2425-redis
IMAGE               CREATED             CREATED BY          SIZE      COMMENT
58fc63a23666       2 minutes ago      ENTRYPOINT ["redis-server" "--protected-mode... 0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      EXPOSE map[6379/tcp:{}] 0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      USER redis           0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      RUN /bin/sh -c chown redis:redis $REDIS_DATA... 0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      WORKDIR /data         0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      ENV REDIS_DATA=/data   0B        buildkit.dockerfile.v0
<missing>          2 minutes ago      RUN /bin/sh -c apt-get update && apt-get in... 51.7MB    buildkit.dockerfile.v0
<missing>          4 minutes ago      ADD https://download.redis.io/releases/redis... 3.53MB    buildkit.dockerfile.v0
<missing>          4 minutes ago      WORKDIR /tmp           0B        buildkit.dockerfile.v0
<missing>          2 weeks ago       # debian.sh --arch 'amd64' out/ 'bookworm' '... 74.8MB    debuerreotype 0.15

vagrant@rre2425-docker:~$
```



¿Por qué hay diferencia de tiempo entre la primera y la segunda capa?



¿De dónde salen los 74.8 MB?



# Ejercicio 1

10

- Ejecuta un contenedor **en primer plano** llamado **redis-server** para comprobar el correcto funcionamiento del servidor Redis (observa la salida obtenida)
  - `docker run --rm --name redis-server X-redis`
- Detén el contenedor anterior (CTRL+C) y ahora ejecútalo **en segundo plano**
  - Comprueba su estado y usa el comando `docker logs` para ver su salida
- **Inspecciona el contenedor** en ejecución para obtener su **dirección IP**
  - Comando `docker inspect` (filtra su salida como se muestra en la figura)

```
vagrant@rre2425-docker:~$ docker run -d --rm --name redis-server rre2425-redis
f3c486c5b3bb9e442dd12832cc8a76809506b9b50641b4f4692a30e56fb0e27f
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS          NAMES
f3c486c5b3bb   rre2425-redis  "redis-server '--pro..." 6 seconds ago  Up 5 seconds        6379/tcp       redis-server
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker logs redis-server
1:C 28 Jan 2025 15:57:19.765 # WARNING Memory overcommit must be enabled! Without it, a background save or repl
. Being disabled, it can also cause failures without low memory condition, see https://github.com/jemalloc/jema
.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1'
1:C 28 Jan 2025 15:57:19.765 * 000000000000 Redis is starting 000000000000
1:C 28 Jan 2025 15:57:19.765 * Redis version=7.4.2 bits=64, commit=00000000, modified=0, pid=1, just started
1:C 28 Jan 2025 15:57:19.765 * Configuration loaded
1:M 28 Jan 2025 15:57:19.765 * monotonic clock: POSIX clock_gettime
1:M 28 Jan 2025 15:57:19.765 * Running mode=standalone, port=6379.
1:M 28 Jan 2025 15:57:19.765 * Server initialized
1:M 28 Jan 2025 15:57:19.765 * Ready to accept connections tcp
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker inspect --format "{{.NetworkSettings.IPAddress}}" redis-server
172.17.0.2
vagrant@rre2425-docker:~$
```



# Ejercicio 1



11

- Inicia un segundo contenedor **en primer plano** llamado **redis-client** para ejecutar el **cliente Redis** usando la **misma imagen** que acabas de crear
  - Es decir, **sin modificar** el *Dockerfile* ni la imagen creada previamente debes **redefinir el punto de entrada (--entrypoint)** de la imagen para ejecutar el binario *redis-cli* (el cliente Redis) en vez del binario *redis-server* (el servidor)
    - Ver [parámetro --entrypoint](#) del comando *docker run*
      - Sintaxis: *docker run [options] --entrypoint new\_command image [args]*
    - Argumentos que debes especificar al cliente Redis (*redis-cli*): *-h <SERVER\_IP> ping*
      - El comando *ping* del cliente Redis permite comprobar la conectividad con el servidor, recibiendo un PONG como respuesta en caso de éxito
  - Muestra información sobre todos los contenedores (*docker ps -a*)
    - Razona sobre el **estado actual** de ambos contenedores

```
vagrant@rre2425-docker:~$ docker run --name redis-client --entrypoint
```

```
PONG
```

```
vagrant@rre2425-docker:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1b6f6cedff66	rre2425-redis	"redis-cli -h 172.17..."	7 seconds ago	Exited (0) 6 seconds ago		redis-client
f3c486c5b3bb	rre2425-redis	"redis-server '--pro..."	3 minutes ago	Up 3 minutes	6379/tcp	redis-server

```
vagrant@rre2425-docker:~$
```

Fíjate en el estado de los contenedores ¿Por qué el contenedor cliente ya no está en ejecución?






# Ejercicio 1

12

- Hasta ahora no hemos proporcionado ningún parámetro relacionado con la red cuando ejecutamos los contenedores
  - **Es decir, ningún parámetro de `docker run` hace referencia a la configuración de red**
- Sin embargo, hemos comprobado que el contenedor cliente de Redis puede comunicarse con el contenedor servidor sin problemas usando su dirección IP
  - Por tanto, ambos contenedores deberían estar en la misma red
- Con el servidor Redis en ejecución, **inspecciona el contenedor** sin filtrar la salida para determinar su **configuración de red**
  - Fíjate en el sub-apartado *Networks* dentro del apartado *NetworksSettings*
  - Determina el **nombre** e **identificador** de las redes (*NetworkID*) a las que se encuentra conectado el contenedor redis-server, y fíjate en la dirección IP del Gateway
  - Lista las redes Docker disponibles para determinar su tipo (*name*, *driver* y *scope*)
    - `docker network ls`
- Comprueba la dirección IP de la interfaz `docker0`

¿Qué deduces?


  - `ip address show <interface>`
- Para finalizar, elimina los contenedores: `docker rm -f redis-server redis-client`



# Ejercicio 1

13

```
"Networks": {  
  "bridge": {  
    "IPAMConfig": null,  
    "Links": null,  
    "Aliases": null,  
    "MacAddress": "02:42:ac:11:00:02",  
    "DriverOpts": null,  
    "NetworkID": "8af9e960b942f87c2567f5fe885f9f8dd00e5204a62b82bcf4e446e0422d28c",  
    "EndpointID": "c7c013a597b65a2219f0f953a8fc169785aee0c8c550c81bcffd4e4a88030900",  
    "Gateway": "172.17.0.1",  
    "IPAddress": "172.17.0.2",  
    "IPPrefixLen": 16,  
    "IPv6Gateway": "",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "DNSNames": null  
  }  
}
```

Parte de la salida del comando *docker inspect* del contenedor *redis-server* donde se puede ver la configuración de red

```
]  
vagrant@rre2425-docker:~$ docker network ls  
NETWORK ID      NAME      DRIVER      SCOPE  
8af9e960b94     bridge    bridge       local  
941b952803d8     host      host         local  
3c66a128cc88     none      null         local  
vagrant@rre2425-docker:~$
```

Listando las redes disponibles


```
vagrant@rre2425-docker:~$ docker ps -a  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS          NAMES  
1b6f6cedff66   rre2425-redis  "redis-cli -h 172.17..." 25 minutes ago Exited (0) 25 minutes ago 6379/tcp       redis-client  
f3c486c5b3bb   rre2425-redis  "redis-server '--pro..." 28 minutes ago Up 28 minutes    6379/tcp       redis-server  
vagrant@rre2425-docker:~$  
vagrant@rre2425-docker:~$ docker rm -f redis-server redis-client  
redis-server  
redis-client  
vagrant@rre2425-docker:~$ docker ps -a  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS          NAMES  
vagrant@rre2425-docker:~$
```

Eliminando los contenedores



## Ejercicio 2

14

- Creación de un fichero *Dockerfile* para generar una imagen *multi-stage* optimizada para ejecutar el servidor Redis del ejercicio previo
- Completa la plantilla *Dockerfile* proporcionada en los ficheros del ejercicio:
  - En la **primera etapa** (denominada **builder**) simplemente es necesario compilar el código fuente de Redis usando la **misma imagen base** que antes
    - Además, completa la URL de descarga, el comando *tar* e instala las dependencias
    - Fíjate que ahora no hace falta hacer ninguna "operación de limpieza" en esta etapa
  - Completa la **segunda etapa** de la siguiente forma:
    - Como imagen base usa **frolvlad/alpine-glibc** ← Imagen Alpine Linux con glibc ya instalado
    - Para crear el usuario y grupo ejecuta: *addgroup -S redis && adduser -S redis -G redis*
    - Usando la instrucción COPY y su opción **--from**, copia los binarios del servidor y del cliente Redis **creados en la etapa builder** en la ruta */usr/local/bin* de la etapa actual
      - Revisa las transparencias 64 y 67 del tutorial de Docker
- Crea la nueva imagen y nómbrala como: **X-redis-multi** 
- Lista las imágenes Docker disponibles: *docker image ls*
- Consulta el "historial" de la nueva imagen: *docker history*
- Prueba la imagen creando un contenedor en **primer plano**



# Ejercicio 2

15



```
agrant@rre2425-docker:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rre2425-redis-multi	latest	7f642ede6930	57 seconds ago	39.4MB
rre2425-redis	latest	58fc63a23666	16 minutes ago	130MB

```
agrant@rre2425-docker:~$
```

```
agrant@rre2425-docker:~$ docker history rre2425-redis-multi
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
7f642ede6930	About a minute ago	ENTRYPOINT ["redis-server" "--protected-mode...	0B	buildkit.dockerfile.v0
<missing>	About a minute ago	COPY /usr/local/bin/redis-cli /usr/local/bin...	7.22MB	buildkit.dockerfile.v0
<missing>	About a minute ago	COPY /usr/local/bin/redis-server /usr/local/...	16.6MB	buildkit.dockerfile.v0
<missing>	2 days ago	EXPOSE map[6379/tcp:{}]	0B	buildkit.dockerfile.v0
<missing>	2 days ago	USER redis	0B	buildkit.dockerfile.v0
<missing>	2 days ago	RUN /bin/sh -c addgroup -S redis -g 990 && ...	3.06kB	buildkit.dockerfile.v0
<missing>	2 days ago	WORKDIR /data	0B	buildkit.dockerfile.v0
<missing>	2 days ago	ENV REDIS_DATA=/data	0B	buildkit.dockerfile.v0
<missing>	3 weeks ago	RUN /bin/sh -c ALPINE_GLIBC_BASE_URL="https:...	7.75MB	buildkit.dockerfile.v0
<missing>	3 weeks ago	ENV LANG=C.UTF-8	0B	buildkit.dockerfile.v0
<missing>	3 weeks ago	CMD ["/bin/sh"]	0B	buildkit.dockerfile.v0
<missing>	3 weeks ago	ADD alpine-minirootfs-3.20.5-x86_64.tar.gz /...	7.8MB	buildkit.dockerfile.v0

```
agrant@rre2425-docker:~$
```

```
agrant@rre2425-docker:~$ docker run --rm --name redis-server rre2425-redis-multi
```

```
1:C 31 Jan 2025 11:57:33.014 # WARNING Memory overcommit must be enabled! Without it, a background save or replicatio
. Being disabled, it can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/i
.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for th
1:C 31 Jan 2025 11:57:33.014 * o000o000o000o Redis is starting o000o000o000o
1:C 31 Jan 2025 11:57:33.014 * Redis version=7.4.2, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 31 Jan 2025 11:57:33.014 * Configuration loaded
1:M 31 Jan 2025 11:57:33.014 * monotonic clock: POSIX clock_gettime
1:M 31 Jan 2025 11:57:33.014 * Running mode=standalone, port=6379.
1:M 31 Jan 2025 11:57:33.015 * Server initialized
1:M 31 Jan 2025 11:57:33.015 * Ready to accept connections tcp
```



# Ejercicio 3

16

- Inicia un contenedor para ejecutar el servidor Redis en **segundo plano**
  - Usa la imagen *multi-stage*
- Ejecuta **varios** clientes Redis en **primer plano** para **incrementar** el contador
  - `docker run --rm --name redis-client --entrypoint redis-cli X-redis-multi <args>`
  - Argumentos del cliente Redis (ver imagen inferior): `-h <SERVER_IP> incr hits`
- Ejecuta otro cliente Redis en **primer plano** para **obtener** el valor del contador
  - Argumentos del cliente Redis: `-h <SERVER_IP> get hits`
- Elimina el servidor Redis y vuelve a ejecutarlo en **segundo plano**
- Vuelve a obtener el valor del contador desde un cliente Redis

¿Qué ocurre?



el contador se reinicia, ya que al eliminar el servidor se eliminan con él todos los archivos

```
vagrant@rre2425-docker:~$ docker run -d --rm --name redis-server rre2425-redis-multi
6084ae5d3b496321a35813918bd20acf45d7b6279274fd235223b0625224e2ad
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 incr hits
1
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 incr hits
2
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 get hits
2
vagrant@rre2425-docker:~$ docker rm -f redis-server
redis-server
vagrant@rre2425-docker:~$ docker run -d --rm --name redis-server rre2425-redis-multi
88b19ab26d8eca0c443605f7b64b097c2ed04413d5db66ad022596e435279f6
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 get hits
```

para hacerlo persistente  
debemos hacer uso de un  
volumen





# Ejercicio 3

17

- Vamos a proporcionarle **persistencia** al contenedor del servidor Redis
  - Elimina el contenedor del servidor Redis (si está en ejecución)
- Crea un volumen “con nombre” (es decir, no anónimo):
  - `docker volume create redis-vol`
- Lista los volúmenes disponibles e inspecciona el nuevo volumen creado
  - `docker volume ls`
  - `docker inspect redis-vol`
- Inicia en segundo plano un nuevo contenedor que ejecute el servidor Redis **montando el volumen** creado previamente en la ruta `/data` del contenedor
  - Puedes usar las opciones `-v` o `--mount` del comando `docker run` (**repasa el tutorial**)
- Vuelve a ejecutar varios clientes Redis para **incrementar** el valor del contador
- Elimina el servidor Redis y vuelve a ejecutarlo en **segundo plano** con el volumen
  - Obtén el valor del contador desde un cliente Redis y luego increméntalo una vez más
  - Lista el contenido del volumen
    - La ruta del volumen la puedes obtener de la salida del comando `docker inspect`



# Ejercicio 3

18



```
vagrant@rre2425-docker:~$ docker volume create redis-vol
redis-vol
```

```
vagrant@rre2425-docker:~$ docker volume ls
```

```
DRIVER    VOLUME NAME
local     redis-vol
```

```
vagrant@rre2425-docker:~$ docker inspect redis-vol
```

```
[
  {
    "CreatedAt": "2025-01-29T14:22:22Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/redis-vol/_data",
    "Name": "redis-vol",
    "Options": null,
    "Scope": "local"
  }
]
```



Para crear el contenedor del servidor Redis podrías usar cualquiera de las dos versiones que tenemos de la imagen. Sin embargo, la versión que uses la primera vez que montes el nuevo volumen es la que deberás usar siempre a partir de ese momento para poder reutilizarlo. En caso contrario, podrías tener problemas de permisos debido a que las dos versiones de la imagen que tenemos usan un SO diferente (Debian y Alpine). Para crear el contenedor cliente es indiferente qué versión de la imagen uses

```
vagrant@rre2425-docker:~$ docker run -d --rm --name redis-server -v redis-vol:/data rre2425-redis-multi
```

```
88b7fd4b4a40d8a4d3eed6c5f992ad957963323187ddecc8fdd6b11729a442eb
```

```
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 incr hits
1
```

```
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 incr hits
2
```

```
vagrant@rre2425-docker:~$ docker rm -f redis-server
redis-server
```

```
vagrant@rre2425-docker:~$ docker run -d --rm --name redis-server -v redis-vol:/data rre2425-redis-multi
```

```
a2767bbdf73e76a00edd077d07cb11bb5b5cde161f85914653215aa1e796c86
```

```
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 get hits
2
```

```
vagrant@rre2425-docker:~$ docker run --rm --name redis-client --entrypoint redis-cli rre2425-redis-multi -h 172.17.0.2 incr hits
3
```

```
vagrant@rre2425-docker:~$ sudo ls -lh /var/lib/docker/volumes/redis-vol/_data
```

```
total 4.0K
```

```
-rw-r--r-- 1 990 990 101 Jan 29 14:24 dump.rdb
```

```
vagrant@rre2425-docker:~$
```

Periódicamente el servidor Redis persiste la base de datos en disco, representada por este fichero



# Ejercicio 4

19

- **Despliegue de una aplicación web Flask que cuenta el número de accesos y los almacena en una base de datos Redis**
  - [Flask](#) es un microframework ligero escrito en Python que permite crear aplicaciones web bajo el patrón MVC de forma sencilla y rápida
- Entre los ficheros del ejercicio tienes un *Dockerfile* (ver figura inferior), el cual:
  - Usa una imagen basada en Alpine Linux con soporte para Python
  - Instala Flask y el cliente Python para acceder a Redis (entre otros) usando *pip*
  - Instala *curl* usando el gestor de paquetes de Alpine Linux (*apk*) y copia el código de la aplicación web Python disponible en el directorio *src*
  - Expone el puerto TCP 5000 (el servidor que incorpora Flask escucha en dicho puerto)
  - Configura un HEALTHCHECK para reportar la "salud" del contenedor mediante *curl*
  - Finalmente, establece el punto de entrada para ejecutar la aplicación con Flask

```
FROM python:3.13.1-alpine
ENV FLASK_APP=counter.py
ENV FLASK_RUN_HOST=0.0.0.0
ENV FLASK_RUN_PORT=5000
ENV FLASK_DEBUG=True
RUN pip install flask redis MarkupSafe
RUN apk --no-cache add curl
WORKDIR /src
COPY src /src
EXPOSE 5000/tcp
HEALTHCHECK CMD curl --fail http://127.0.0.1:5000 || exit 1
ENTRYPOINT ["flask", "run"]
```




Fíjate en el HEALTHCHECK y razona qué hace. Repasa el tutorial de Docker si no recuerdas para que sirve esta instrucción. ¿Por qué se hace una petición GET a esa dirección IP y puerto en concreto?



# Ejercicio 4

20

- Por otro lado, en el directorio `src` dispones del código Python de la aplicación web Flask (**`counter.py`**) y la plantilla HTML (**`index.html`**) en *templates*
  - Abre el fichero `index.html` con un **editor de texto en tu equipo** para incluir tu nombre y apellidos
    - **Debes modificar únicamente la variable `name` (línea 7)** 

```
app = Flask(__name__)
redis_host = 'redis-server'
cache = redis.Redis(host=redis_host, port=6379)
error = None

def get_hit_count():
    global error
    retries = 2
    redis_result = True

    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                error = str(exc)
                return -1
            retries -= 1
            time.sleep(0.5)
```

← **Extracto del fichero `counter.py` donde se configuran los datos de conexión a la base de datos Redis. Fíjate en el nombre del `host` y el puerto al que se conecta**



# Ejercicio 4

21

- Crea una imagen usando el *Dockerfile* proporcionado
  - **Debes nombrar la imagen como: X-web**
- Lista las imágenes disponibles
- Inicia un contenedor con nombre **redis-server** para ejecutar el servidor Redis en **segundo plano** usando la **imagen y el volumen** creados previamente



```
vagrant@rre2425-docker:~$ docker image ls
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
rre2425-web          latest      50e364fade83  5 seconds ago  66.3MB
rre2425-redis-multi  latest      7f642ede6930  4 minutes ago  39.4MB
rre2425-redis        latest      58fc63a23666  19 minutes ago 130MB
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker run --rm -d --name redis-server -v redis-vol:/data rre2425-redis-multi
726a9ab01001f50da70595111711b907d1a0665169190704e6b145820a3ebc06
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED        STATUS        PORTS        NAMES
726a9ab01001   rre2425-redis-multi "redis-server '--pro..." 5 seconds ago  Up 5 seconds  6379/tcp     redis-server
vagrant@rre2425-docker:~$
```



# Ejercicio 4

22

- Inicia un contenedor con nombre **webapp** para ejecutar la aplicación Flask en **primer plano** usando la imagen creada en este ejercicio
  - El comando *docker run* debe:
    - Redirigir el puerto 80 de la VM al puerto 5000 del contenedor (razona el por qué)
    - Definir un **link** con nombre **redis-server** para que el contenedor webapp pueda conectarse a la base de datos Redis usando dicho nombre en vez de usar su dirección IP
      - Repasa cómo se conecta la aplicación web a la base de datos (ver transparencia 20)
- Comprueba el acceso a la aplicación web desde tu **host**: <http://localhost:8080>
  - Si funciona, presiona F5 varias veces para comprobar que el contador se incrementa
  - En la salida del contenedor **webapp** deberías ver las peticiones GET que recibe

para que al buscar el puerto 8080 en el localhost (que es el que está forwardado con el puerto 80 de la VM) se conecte al servidor web del contenedor directamente

```
vagrant@rre2425-docker:~$ docker run --rm --name webapp --link redis-server -p 80:5000 rre2425-web
```

```
* Serving Flask app 'counter.py'
```

```
* Debug mode: on
```

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
```

```
* Running on all addresses (0.0.0.0)
```

```
* Running on http://127.0.0.1:5000
```

```
* Running on http://172.17.0.3:5000
```

```
Press CTRL+C to quit
```

```
* Restarting with stat
```

```
* Debugger is active!
```

```
* Debugger PIN: 262-959-879
```

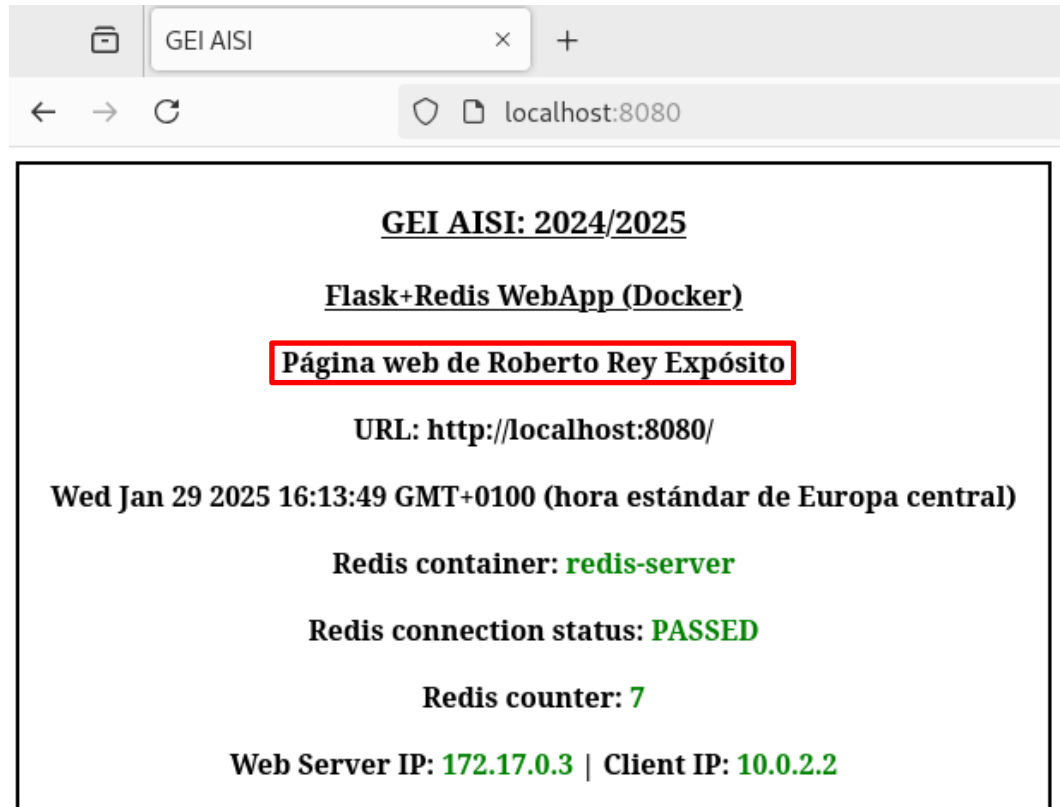
```
10.0.2.2 - - [29/Jan/2025 15:13:18] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [29/Jan/2025 15:13:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2025 15:13:44] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [29/Jan/2025 15:13:49] "GET / HTTP/1.1" 200 -
█
```

Las conexiones desde localhost se realizan debido a la instrucción: `HEALTHCHECK CMD curl --fail http://127.0.0.1:5000 || exit 1`



¿Por qué hay accesos desde dos IPs distintas?

Peticiones GET recibidas en el contenedor webapp →





# Ejercicio 4

24



- Detén el contenedor webapp (CTRL+C) y ejecútalo ahora **en segundo plano**
  - Comprueba que ambos contenedores están en ejecución (*docker ps*) y **espera** a que el contenedor **webapp** reporte **"healthy"** en su estado
  - Comprueba de nuevo que funciona el acceso desde el navegador de tu **host**
    - Usa el comando *docker logs* para ver las peticiones GET que recibe el contenedor webapp

```
vagrant@rre2425-docker:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                    PORTS
c52081d638fd   rre2425-web    "flask run"              42 seconds ago Up 42 seconds (healthy)  0.0.0.0:80->5000/tcp, [::]:80->5000/tcp
e28fab7003ca   rre2425-redis-multi "redis-server '--pro..." 18 minutes ago Up 18 minutes           6379/tcp

vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker logs webapp
* Serving Flask app 'counter.py'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.3:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 262-959-879
127.0.0.1 - - [29/Jan/2025 15:19:09] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [29/Jan/2025 15:19:23] "GET / HTTP/1.1" 200 -
vagrant@rre2425-docker:~$
```

- Comprueba ahora el acceso **desde la VM** con el comando: ***curl http://localhost:X***
  - ¿A qué puerto X deberás hacer la petición? ¿8080? **¿80?** ¿5000? ¿Otro?
  - Haz pruebas y trata de razonar el/los puerto/s correcto/s





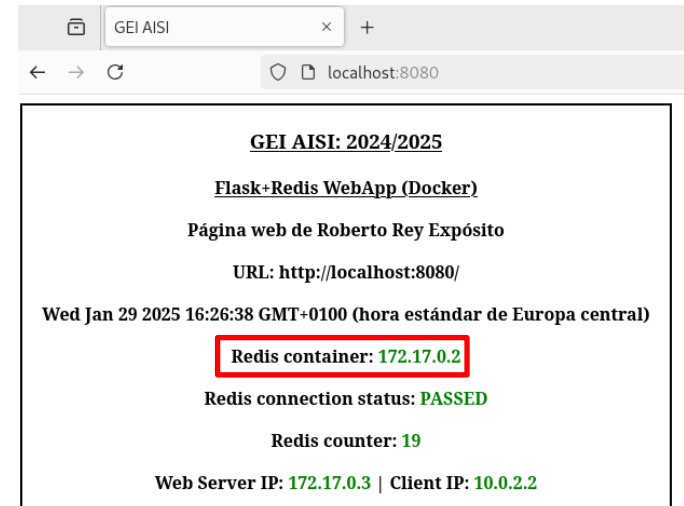
# Ejercicio 4

25

- Detén el contenedor webapp y prueba a ejecutarlo de nuevo, **pero ahora sin definir el link** en el comando *docker run*
  - ¿Qué error obtienes al acceder desde tu *host*? Fíjate en el motivo concreto
- Inspecciona ambos contenedores para obtener su configuración de red
  - Comprueba a qué redes están conectados y anota sus IPs *redis-server: 172.17.0.2*
    - ¿Están conectados a la misma red? ¿Por qué no funciona sin el *link*?
- Modifica el código fuente para que la conexión a la base de datos se realice usando la IP del contenedor Redis en vez de usar el nombre 'redis-server'
  - **Al modificar el código fuente, tienes que volver a crear la imagen X-web**
- Ejecuta de nuevo el contenedor webapp



Después de estas pruebas, modifica de nuevo *counter.py* para que la conexión a la base de datos se realice usando el nombre 'redis-server' (como estaba originalmente) y no la dirección IP del contenedor y **vuelve a crear la imagen**





# Ejercicio 4

26



Asegúrate que la conexión a la base de datos usa el nombre 'redis-server' y no la dirección IP

- Desde webapp, comprueba la conectividad con redis-server mediante **ping**
  - Para estas pruebas usa **docker exec** para estas pruebas, el cual permite ejecutar un nuevo comando dentro de un contenedor ya en ejecución
  - Prueba haciendo **ping** a la IP y al nombre del contenedor, tanto en el caso en el que ejecutas webapp definiendo el **link** como sin definirlo

```
vagrant@rre2425-docker:~$ docker run --rm -d --name webapp --link redis-server -p 80:5000 rre2425-web  
ed031aaa159374f4b85650fd66a6389b07386f2c3b0d1b85fed8625acd1c5c0
```

```
vagrant@rre2425-docker:~$ docker inspect --format "{{.NetworkSettings.IPAddress}}" redis-server  
172.17.0.2
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 172.17.0.2  
PING 172.17.0.2 (172.17.0.2): 56 data bytes  
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.185 ms
```

Ejecutamos el comando **ping** dentro del contenedor webapp usando su IP y su nombre

```
--- 172.17.0.2 ping statistics ---  
1 packets transmitted, 1 packets received, 0% packet loss  
round-trip min/avg/max = 0.185/0.185/0.185 ms
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 redis-server  
PING redis-server (172.17.0.2): 56 data bytes  
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.030 ms
```

Pruebas con **ping** definiendo el **link**



```
--- redis-server ping statistics ---  
1 packets transmitted, 1 packets received, 0% packet loss  
round-trip min/avg/max = 0.030/0.030/0.030 ms
```

```
vagrant@rre2425-docker:~$ docker exec webapp cat /etc/hosts
```

Mostramos el fichero de **hosts** del contenedor webapp

```
127.0.0.1    localhost  
::1         localhost ip6-localhost ip6-loopback  
fe00::0     ip6-localnet  
ff00::0     ip6-mcastprefix  
ff02::1     ip6-allnodes  
ff02::2     ip6-allrouters
```

```
172.17.0.2   redis-server e28fab7003ca  
172.17.0.3   ed031aaa1593
```

```
vagrant@rre2425-docker:~$ docker stop webapp  
webapp  
vagrant@rre2425-docker:~$
```



# Ejercicio 4

27

```
vagrant@rre2425-docker:~$ docker run --rm -d --name webapp -p 80:5000 rre2425-web  
f2758c26bb81c2ac8723e9effe54c3c20db442be6ab1d04807d878d8be928a58
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 172.17.0.2
```

```
PING 172.17.0.2 (172.17.0.2): 56 data bytes
```

```
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.139 ms
```

```
--- 172.17.0.2 ping statistics ---
```

```
1 packets transmitted, 1 packets received, 0% packet loss
```

```
round-trip min/avg/max = 0.139/0.139/0.139 ms
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 redis-server
```

```
ping: bad address 'redis-server'
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp cat /etc/hosts
```

```
127.0.0.1      localhost
```

```
::1           localhost ip6-localhost ip6-loopback
```

```
fe00::0 ip6-localnet
```

```
ff00::0 ip6-mcastprefix
```

```
ff02::1 ip6-allnodes
```

```
ff02::2 ip6-allrouters
```

```
172.17.0.3      f2758c26bb81
```

```
vagrant@rre2425-docker:~$ docker stop webapp
```

```
webapp
```

```
vagrant@rre2425-docker:~$
```



Pruebas con *ping* sin definir el *link*



# Ejercicio 4

28

```
vagrant@rre2425-docker:~$ docker run --rm -d --name webapp --link redis-server:aisi -p 80:5000 rre2425-web  
72cd892721567707d747594100a8846901f3e33836aa1c84f6c1881cd3c502b2
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 172.17.0.2
```

```
PING 172.17.0.2 (172.17.0.2): 56 data bytes  
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.169 ms
```

```
--- 172.17.0.2 ping statistics ---
```

```
1 packets transmitted, 1 packets received, 0% packet loss  
round-trip min/avg/max = 0.169/0.169/0.169 ms
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 redis-server
```

```
PING redis-server (172.17.0.2): 56 data bytes  
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.037 ms
```

```
--- redis-server ping statistics ---
```

```
1 packets transmitted, 1 packets received, 0% packet loss  
round-trip min/avg/max = 0.037/0.037/0.037 ms
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp ping -c 1 aisi
```

```
PING aisi (172.17.0.2): 56 data bytes  
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.028 ms
```

```
--- aisi ping statistics ---
```

```
1 packets transmitted, 1 packets received, 0% packet loss  
round-trip min/avg/max = 0.028/0.028/0.028 ms
```

```
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker:~$ docker exec webapp cat /etc/hosts
```

```
127.0.0.1      localhost  
::1          localhost ip6-localhost ip6-loopback  
fe00::0      ip6-localnet  
ff00::0      ip6-mcastprefix  
ff02::1      ip6-allnodes  
ff02::2      ip6-allrouters
```

```
172.17.0.2     aisi e28fab7003ca redis-server
```

```
172.17.0.3     72cd89272156
```

```
vagrant@rre2425-docker:~$ docker stop webapp
```

```
webapp
```

```
vagrant@rre2425-docker:~$
```



Pruebas con *ping* definiendo un *link* diferente. **Fíjate bien en este ejemplo**



# Ejercicio 5

29

- **Despliegue de una aplicación multicontenedor usando Docker Compose**
  - La aplicación a desplegar será la misma que la del ejercicio 4
- Para este ejercicio (y siguientes), **elimina** del *Dockerfile* de la aplicación web:
  - La instrucción **HEALTHCHECK**, ya que lo vamos a definir en el propio fichero *compose.yml*
  - La instrucción **COPY**, ya que haremos un montaje *bind* en el fichero *compose.yml* para que el contenedor acceda directamente al código fuente de la aplicación
  - Tras modificar el *Dockerfile*, debes volver a crear la imagen **X-web**
- Asegúrate también de que el código fuente de la aplicación web (*counter.py*) utiliza 'redis-server' como *hostname* de conexión a la BBDD y no la IP
- Finalmente, asegúrate de que no tienes ningún contenedor en ejecución
  - Elimínalos en caso contrario



# Ejercicio 5

30

- Usando la plantilla YAML **compose.yml**, completa la definición de los **dos servicios** que componen la aplicación multicontenedor a desplegar
  - Para Redis, define el servicio **redis** usando la imagen del ejercicio 2
    - Monta el volumen con nombre **redis-vol** en la ruta */data* del contenedor
      - Fíjate que el volumen ya definido en el fichero usa la opción [external](#)
  - Para la aplicación web, define el servicio **webapp** usando la imagen del ejercicio 4
    - Mapea el puerto 80 de la VM al puerto 5000 del contenedor
    - Define un **link** con el nombre **redis-server** para que la aplicación web pueda conectarse a la base de datos Redis usando dicho nombre
    - Configura el *healthcheck* con *curl* para que se ejecute cada 15 segundos
    - Añade una dependencia con el servicio **redis**
    - Define un montaje *bind* para montar el directorio de la VM que contiene el código fuente de la aplicación (*counter.py*) en la ruta */src* del contenedor
      - Esto te permitirá modificar el código fuente desde tu *host* y que la aplicación web pueda detectar los cambios "al vuelo"



# Ejercicio 5

31



- Despliega la aplicación en **primer plano**: `docker compose -f <yaml> up`
- Comprueba el acceso desde un navegador de tu **host**: <http://localhost:8080>

```
vagrant@vxe2425-docker:~$ docker compose -f /vagrant/ej5/compose.yml up
[+] Running 3/3
  ✓ Network ej5_default      Created
  ✓ Container ej5-redis-1    Created
  ✓ Container ej5-webapp-1   Created
Attaching to redis-1, webapp-1
redis-1 | 1:C 29 Jan 2025 16:07:28.083 # WARNING Memory overcommit must be enabled! Without it, a
redis-1 | ry condition. Being disabled, it can also cause failures without low memory condition, see https://
redis-1 | ssue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl
redis-1 | 1:C 29 Jan 2025 16:07:28.083 * o000o000o000o Redis is starting o000o000o000o
redis-1 | 1:C 29 Jan 2025 16:07:28.083 * Redis version=7.4.2, bits=64, commit=00000000, modified=
redis-1 | 1:C 29 Jan 2025 16:07:28.083 * Configuration loaded
redis-1 | 1:M 29 Jan 2025 16:07:28.085 * monotonic clock: POSIX clock_gettime
redis-1 | 1:M 29 Jan 2025 16:07:28.085 * Running mode=standalone, port=6379.
redis-1 | 1:M 29 Jan 2025 16:07:28.088 * Server initialized
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * Loading RDB produced by version 7.4.2
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * RDB age 405 seconds
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * RDB memory usage when created 0.95 Mb
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * Done loading RDB, keys loaded: 1, keys expired: 0.
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * DB loaded from disk: 0.002 seconds
redis-1 | 1:M 29 Jan 2025 16:07:28.090 * Ready to accept connections tcp
webapp-1 | * Serving Flask app 'counter.py'
webapp-1 | * Debug mode: on
webapp-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a
webapp-1 | * Running on all addresses (0.0.0.0)
webapp-1 | * Running on http://127.0.0.1:5000
webapp-1 | * Running on http://172.18.0.3:5000
webapp-1 | Press CTRL+C to quit
webapp-1 | * Restarting with stat
webapp-1 | * Debugger is active!
webapp-1 | * Debugger PIN: 342-982-782
webapp-1 | 10.0.2.2 - - [29/Jan/2025 16:07:38] "GET / HTTP/1.1" 200 -
redis-1 | 1:M 29 Jan 2025 16:07:38.485 * 1 changes in 5 seconds. Saving...
redis-1 | 1:M 29 Jan 2025 16:07:38.489 * Background saving started by pid 11
redis-1 | 11:C 29 Jan 2025 16:07:38.495 * DB saved on disk
redis-1 | 11:C 29 Jan 2025 16:07:38.496 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
redis-1 | 1:M 29 Jan 2025 16:07:38.592 * Background saving terminated with success
webapp-1 | 10.0.2.2 - - [29/Jan/2025 16:07:41] "GET / HTTP/1.1" 200 -
webapp-1 | 127.0.0.1 - - [29/Jan/2025 16:07:43] "GET / HTTP/1.1" 200 -
redis-1 | 1:M 29 Jan 2025 16:07:44.046 * 1 changes in 5 seconds. Saving...
redis-1 | 1:M 29 Jan 2025 16:07:44.046 * Background saving started by pid 12
redis-1 | 12:C 29 Jan 2025 16:07:44.050 * DB saved on disk
```

Peticiones GET recibidas  
en el contenedor webapp.  
También se aprecia como  
el Redis persiste la BBDD



# Ejercicio 5



32

- Detén la aplicación (CTRL+C) y desplégala de nuevo en **segundo plano**
  - Comprueba los servicios y contenedores en ejecución (*docker compose ps* | *docker ps*)
- Comprueba que gracias al montaje *bind* puedes modificar el código fuente de la aplicación web "al vuelo" (**sin detener el servicio**)
  - Desde la VM, accede a la aplicación usando *curl -s* (**filtra con *grep* como en la figura**)
  - A continuación, cambia el mensaje a mostrar modificando el fichero *index.html*
  - Accede de nuevo usando *curl -s* para obtener el nuevo mensaje

```
vagrant@rre2425-docker:~$ docker compose -f /vagrant/ej5/compose.yml ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
ej5-redis-1	rre2425-redis-multi	"redis-server '--pro..."	redis	About a minute ago	Up About a minute	6379/tcp
ej5-webapp-1	rre2425-web	"flask run"	webapp	About a minute ago	Up About a minute (healthy)	0.0.0.0:80->5000/tcp, [::]:80->5000/tcp

```
vagrant@rre2425-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7b4f92b54c27	rre2425-web	"flask run"	About a minute ago	Up About a minute (healthy)	0.0.0.0:80->5000/tcp, [::]:80->5000/tcp	ej5-webapp-1
e1478fff9638	rre2425-redis-multi	"redis-server '--pro..."	About a minute ago	Up About a minute	6379/tcp	ej5-redis-1

```
vagrant@rre2425-docker:~$ curl -s http://localhost | grep "Redis counter"
```

<p>Redis counter: <span style="color: green;">77</span></p>

```
vagrant@rre2425-docker:~$ sed -i 's/Redis counter/New Redis counter/g' /vagrant/ej4/src/templates/index.html
```

```
vagrant@rre2425-docker:~$ curl -s http://localhost | grep "Redis counter"
```

<p>New Redis counter: <span style="color: green;">79</span></p>

Modificación del mensaje a mostrar usando el comando *sed*

- Detén la aplicación y elimina todos sus recursos:
  - *docker compose -f <yaml> down*





# Ejercicio 5

33

- ¿Podría la aplicación web conectarse a la base de datos Redis sin necesidad de definir el *link* en el fichero *compose.yml*? Vamos a comprobarlo
  - **Elimina el *link*** del fichero YAML, vuelve a desplegar los servicios en segundo plano y comprueba si puedes acceder desde el navegador de tu *host*
    - ¿Qué error obtienes y por qué?
    - Sin detener los servicios, modifica el código de la aplicación (*counter.py*) para usar "redis" como *hostname* de conexión a la base de datos y accede de nuevo desde el navegador
    - **¿Por qué ahora sí funciona?**
  - Comprueba (inspecciona) a qué red(es) están conectados los contenedores en ejecución, su tipo y qué dirección IP tienen
    - ¿Qué diferencias observas con el ejercicio anterior?
    - Lista también las redes disponibles (*docker network ls*)
    - Fíjate en los recursos que crea Docker Compose al desplegar y terminar la aplicación

```
vagrant@rre2425-docker:~$ docker inspect ej5-webapp-1 | grep NetworkID
    "NetworkID": "baaab3aeb2e9c4d3b37a457e79757f4dfc96c73bbe049434a94ed3166a1ee852",
vagrant@rre2425-docker:~$ docker inspect ej5-redis-1 | grep NetworkID
    "NetworkID": "baaab3aeb2e9c4d3b37a457e79757f4dfc96c73bbe049434a94ed3166a1ee852",
vagrant@rre2425-docker:~$
vagrant@rre2425-docker:~$ docker network ls
NETWORK ID        NAME              DRIVER           SCOPE
c5a6dc81fdf4     bridge           bridge          local
baaab3aeb2e9     ej5_default      bridge          local
941b952803d8     host             host            local
3c66a128cc88     none            null            local
vagrant@rre2425-docker:~$
```



## Ejercicio 6

34

- **Despliegue de una aplicación multicontenedor usando Docker Swarm**
  - La aplicación a desplegar es la misma del ejercicio previo
- Desde la *VM manager*, crea un clúster Docker Swarm
  - `docker swarm init --advertise-addr enp0s8`
  - Esta VM actuará como *manager* y como *worker* (configuración por defecto)
- Comprueba que tu clúster dispone de un nodo activo: `docker node ls`

```
vagrant@rre2425-docker:~$ docker swarm init --advertise-addr enp0s8
Swarm initialized: current node (eg8z2zbkjpgtehtl3wmx5l4k) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-2faofnkslklynxqprer61kdw33ud5rojv1xwjz315ne4z4pz10-0sa0c19zg78c0y2etdnwifplk 192.168.56.10:2377
```

Comando para añadir nodos *worker* al enjambre. **Copia el tuyo**

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
vagrant@rre2425-docker:~$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
eg8z2zbkjpgtehtl3wmx5l4k *	rre2425-docker	Ready	Active	Leader	27.5.0

- **Conéctate a la VM *worker* y ejecuta el comando para unirse al enjambre**
  - Para conectarte a la VM, ejecuta desde tu *host*: `vagrant ssh worker`

Estamos conectados  
a la VM *worker*

```
vagrant@rre2425-docker-worker:~$ docker swarm join --token SWMTKN-1-2
This node joined a swarm as a worker.
vagrant@rre2425-docker-worker:~$
```

Token de acceso  
recortado



# Ejercicio 6

35

- Conéctate de nuevo a la *VM manager* y lista los nodos

```
vagrant@rre2425-docker:~$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
eg8z2zbkjgptehgtl3wmx514k *	rre2425-docker	Ready	Active	Leader	27.5.0
v1tud2zmqtffolj9mmz3gryew	rre2425-docker-worker	Ready	Active		27.5.0

```
vagrant@rre2425-docker:~$
```

- Como usaremos imágenes Docker creadas en local (no las hemos subido al *Docker Hub*), necesitamos un *Docker Registry* privado para alojarlas, de forma que **todos los nodos** del enjambre tengan acceso a las imágenes
  - Una alternativa (no recomendable) sería crear las imágenes también en la *VM worker*
- Para desplegar un *Docker Registry* en el clúster, ejecuta:
  - `docker service create --name registry -p 5000:5000 registry:2.8.3` ←
- Etiquetamos las imágenes con el *hostname* y puerto del *Registry*:
  - `docker tag X-redis-multi localhost:5000/X-redis-multi`
  - `docker tag X-web localhost:5000/X-web`
- Subimos las imágenes que necesitamos al *Registry*:
  - `docker push localhost:5000/X-redis-multi`
  - `docker push localhost:5000/X-web`

Ejecutado como un **servicio** en el clúster (veremos este comando en el siguiente ejercicio)



Como ya es habitual, sustituye X por tu prefijo



## Ejercicio 6

36

- Obtenemos el listado de imágenes disponibles en nuestro *Docker Registry*
  - `curl localhost:5000/v2/_catalog`

```
vagrant@rre2425-docker:~$ curl localhost:5000/v2/_catalog
{"repositories":["rre2425-redis-multi","rre2425-web"]}
```

`vagrant@rre2425-docker:~$`
- Copia el fichero **compose.yml** del ejercicio previo (**sin el link**) en el directorio del ejercicio 6 y modifica la definición de los **servicios** de la siguiente forma:
  - Usa las imágenes alojadas en nuestro *Docker Registry* local
  - Define **webapp** como un servicio **replicado** con **tres** instancias usando **deploy**
- Despliega el *stack* en el clúster Swarm
  - `docker stack deploy -d=false -c <yaml> <stack_name>`
- Comprueba el *stack*, los servicios, las tareas y los contenedores en ejecución
  - `docker stack ls`
  - `docker stack services`
  - `docker stack ps`
  - `docker ps`



# Ejercicio 6

37



El despliegue de todas las instancias puede tardar varios segundos, paciencia. **Repite los comandos hasta ver todas las instancias en ejecución**

```
vagrant@rre2425-docker:~$ docker stack ls
```

```
NAME      SERVICES
```

```
mystack  2
```

```
vagrant@rre2425-docker:~$ docker stack services mystack
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
d3zol17195jy	mystack_redis	replicated	1/1	localhost:5000/rre2425-redis-multi:latest	
m89kxbfowl3	mystack_webapp	replicated	3/3	localhost:5000/rre2425-web:latest	*:80->5000/tcp

```
vagrant@rre2425-docker:~$ docker stack ps mystack
```

ID	NAME	IMAGE	MODE	DESIRED STATE	CURRENT STATE
mkh1xlox6wj	mystack_redis.1	localhost:5000/rre2425-redis-multi:latest	rre2425-docker	Running	Running 54 seconds ago
iqywohz1q7o4	mystack_webapp.1	localhost:5000/rre2425-web:latest	rre2425-docker-worker	Running	Running 39 seconds ago
ccji0arkh5xq	mystack_webapp.2	localhost:5000/rre2425-web:latest	rre2425-docker	Running	Running 39 seconds ago
susnl340xx7f	mystack_webapp.3	localhost:5000/rre2425-web:latest	rre2425-docker-worker	Running	Running 39 seconds ago

```
vagrant@rre2425-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
2b131545cab2	localhost:5000/rre2425-web:latest	"flask run"	About a minute ago	Up About a minute (healthy)	5000/tcp
7776ba39cdd6	localhost:5000/rre2425-redis-multi:latest	"redis-server '--pro..."	About a minute ago	Up About a minute	6379/tcp
cc93b8c11af8	registry:2.8.3	"/entrypoint.sh /etc..."	12 minutes ago	Up 12 minutes	5000/tcp

```
vagrant@rre2425-docker:~$
```

- Conéctate a la VM worker y ejecuta: *docker stack ls*
- A continuación, ejecuta en la VM worker: *docker ps*

¿Qué error obtienes y por qué?



Estamos  
conectados a la  
VM worker

```
vagrant@rre2425-docker-worker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
07a132279c7a	localhost:5000/rre2425-web:latest	"flask run"	3 minutes ago	Up 3 minutes (healthy)	5000/tcp	mystack_webapp.1.
33567c000af9	localhost:5000/rre2425-web:latest	"flask run"	3 minutes ago	Up 3 minutes (healthy)	5000/tcp	mystack_webapp.3.

```
vagrant@rre2425-docker-worker:~$
```



Puede ocurrir que obtengas una **distribución diferente** de los contenedores que se ejecutan en cada VM (p.e. contenedor redis ejecutándose en la VM worker), ya que eso dependerá de cómo Docker Swarm planifica las tareas en tu caso



# Ejercicio 6

38

- Comprueba el acceso **desde tu host**: <http://localhost:8080>
  - Realiza múltiples accesos (F5) y comprueba cómo se distribuyen las peticiones entre los contenedores del servicio webapp
  - Fíjate en la IP del servidor web que responde (ver siguiente transparencia)
- Comprueba el acceso **desde las VM manager/worker** usando **curl -s**

```
vagrant@rre2425-docker:~$ curl -s http://localhost | grep "Redis counter"
<p>New Redis counter: <span style="color: green;">397</span></p>
vagrant@rre2425-docker:~$
```

```
vagrant@rre2425-docker-worker:~$ curl -s http://localhost | grep "Redis counter"
<p>New Redis counter: <span style="color: green;">323</span></p>
vagrant@rre2425-docker-worker:~$
```

- Comprueba (inspecciona) a qué red(es) están conectados todos los contenedores en ejecución, su tipo y qué dirección IP tienen
  - ¿Qué diferencias observas con el ejercicio anterior?
  - Lista también las redes disponibles (*docker network ls*)
  - Fíjate bien en los recursos que crea Docker Swarm al desplegar la aplicación
- Inspecciona también ambos servicios y fíjate en "*VirtualIPs*"
  - *docker service inspect <SERVICE/ID>*



# Ejercicio 6

39

GEI AISI

localhost:8080

**GEI AISI: 2024/2025**

**Flask+Redis WebApp (Docker)**

Página web de Roberto Rey Expósito

URL: <http://localhost:8080/>

Wed Jan 29 2025 17:45:21 GMT+0100 (hora estándar de Europa central)

Redis container: **redis**

Redis connection status: **PASSED**

New Redis counter: **241**

**Web Server IP: 10.0.0.23 | Client IP: 10.0.0.2**

GEI AISI

localhost:8080

**GEI AISI: 2024/2025**

**Flask+Redis WebApp (Docker)**

Página web de Roberto Rey Expósito

URL: <http://localhost:8080/>

Wed Jan 29 2025 17:45:48 GMT+0100 (hora estándar de Europa central)

Redis container: **redis**

Redis connection status: **PASSED**

New Redis counter: **248**

**Web Server IP: 10.0.0.24 | Client IP: 10.0.0.2**

GEI AISI

localhost:8080

**GEI AISI: 2024/2025**

**Flask+Redis WebApp (Docker)**

Página web de Roberto Rey Expósito

URL: <http://localhost:8080/>

Wed Jan 29 2025 17:45:51 GMT+0100 (hora estándar de Europa central)

Redis container: **redis**

Redis connection status: **PASSED**

New Redis counter: **249**

**Web Server IP: 10.0.0.25 | Client IP: 10.0.0.2**



Es normal que tus direcciones IP puedan ser distintas



# Ejercicio 6

40



- Elimina uno de los contenedores del servicio **webapp**: `docker rm -f <NAME/ID>`
  - Tras unos segundos comprueba los contenedores en ejecución, ¿qué observas?
- Prueba a escalar el servicio **webapp** hacia arriba y abajo: `docker service scale`

```
vagrant@rre2425-docker:~$ docker stack services mystack
ID            NAME            MODE            REPLICAS  IMAGE                                  PORTS
d3zol17l95jy  mystack_redis    replicated       1/1        localhost:5000/rre2425-redis-multi:latest
m89kxbxfowl3  mystack_webapp    replicated       3/3        localhost:5000/rre2425-web:latest      *:80->5000/tcp

vagrant@rre2425-docker:~$ docker service scale mystack_webapp=4
mystack_webapp scaled to 4
overall progress: 4 out of 4 tasks
1/4: running [=====]
2/4: running [=====]
3/4: running [=====]
4/4: running [=====]
verify: Service mystack_webapp converged
vagrant@rre2425-docker:~$ docker stack services mystack
ID            NAME            MODE            REPLICAS  IMAGE                                  PORTS
d3zol17l95jy  mystack_redis    replicated       1/1        localhost:5000/rre2425-redis-multi:latest
m89kxbxfowl3  mystack_webapp    replicated       4/4        localhost:5000/rre2425-web:latest      *:80->5000/tcp

vagrant@rre2425-docker:~$ docker stack ps mystack
ID            NAME            IMAGE                                  NODE            DESIRED STATE  CURRENT STATE  ERROR
mkh1xlox6wd  mystack_redis.1  localhost:5000/rre2425-redis-multi:latest  rre2425-docker  Running         Running 25 minutes ago
iqywzhz1q7o4  mystack_webapp.1  localhost:5000/rre2425-web:latest          rre2425-docker-worker  Running         Running 24 minutes ago
t3vxf9sel5v3  mystack_webapp.2  localhost:5000/rre2425-web:latest          rre2425-docker-worker  Running         Running about a minute ago
ccji0arkh5xq  \_ mystack_webapp.2  localhost:5000/rre2425-web:latest          rre2425-docker-worker  Shutdown        Failed 2 minutes ago  "task: non-zero exit (137)"
susn1340xx7f  mystack_webapp.3  localhost:5000/rre2425-web:latest          rre2425-docker-worker  Running         Running 24 minutes ago
89b1pkvtd9c5  mystack_webapp.4  localhost:5000/rre2425-web:latest          rre2425-docker-worker  Running         Running 17 seconds ago

vagrant@rre2425-docker:~$ docker service scale mystack_webapp=2
mystack_webapp scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====]
2/2: running [=====]
verify: Service mystack_webapp converged
vagrant@rre2425-docker:~$ docker stack services mystack
ID            NAME            MODE            REPLICAS  IMAGE                                  PORTS
d3zol17l95jy  mystack_redis    replicated       1/1        localhost:5000/rre2425-redis-multi:latest
m89kxbxfowl3  mystack_webapp    replicated       2/2        localhost:5000/rre2425-web:latest      *:80->5000/tcp

vagrant@rre2425-docker:~$
```



¿Por qué sale este error?

- Para terminar, detén el stack y elimina todos los recursos
  - `docker stack rm/down <stack_name>`





# Ejercicio 7

41

- **Despliegue de servicios Docker en un clúster Swarm**
  - Alternativa "no declarativa" (sin usar fichero YAML) para desplegar servicios
  - Desplegaremos los mismos servicios que en el ejercicio 6
- Crea dos servicios Swarm usando el comando: ***docker service create***
  - Servicio con nombre **redis** con una única instancia
    - Monta el volumen con nombre **redis-vol** en la ruta */data* del contenedor
  - Servicio replicado con **cinco instancias** con nombre **webapp**
    - Mapea el puerto 80 de la VM al puerto 5000 del servicio
    - Define un montaje *bind* para montar el directorio de la VM que contiene el código fuente de la aplicación (*counter.py*) en la ruta */src* del contenedor
- Comprueba los servicios, tareas y contenedores en ejecución
  - *docker service ls*
  - *docker service ps redis webapp*
  - *docker ps*



# Ejercicio 7

42



```
vagrant@rre2425-docker:~$ docker service ls
ID                NAME      MODE          REPLICAS  IMAGE                                  PORTS
drdhgjrlwsp      redis     replicated    1/1        localhost:5000/rre2425-redis-multi:latest
hndr3xvcx88b     webapp    replicated    5/5        localhost:5000/rre2425-web:latest     *:80->5000/tcp

vagrant@rre2425-docker:~$ docker service ps redis webapp
ID                NAME      IMAGE                                  NODE                DESIRED STATE  CURRENT STATE  ERROR  PORTS
htfb7tco50db     redis.1   localhost:5000/rre2425-redis-multi:latest  rre2425-docker     Running        Running 2 minutes ago
w5xx9tu1co88     webapp.1  localhost:5000/rre2425-web:latest          rre2425-docker-worker Running        Running 2 minutes ago
qrt903l7cae9     webapp.2  localhost:5000/rre2425-web:latest          rre2425-docker-worker Running        Running 2 minutes ago
zfvyo65asrlu     webapp.3  localhost:5000/rre2425-web:latest          rre2425-docker     Running        Running 2 minutes ago
xya8z7wi8bnq     webapp.4  localhost:5000/rre2425-web:latest          rre2425-docker-worker Running        Running 2 minutes ago
jfi89vsvzhntu    webapp.5  localhost:5000/rre2425-web:latest          rre2425-docker     Running        Running 2 minutes ago

vagrant@rre2425-docker:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED   STATUS    PORTS
db5457c1f7a8   localhost:5000/rre2425-web:latest   "flask run"             2 minutes ago  Up 2 minutes  5000/tcp
5d93145b9c87   localhost:5000/rre2425-web:latest   "flask run"             2 minutes ago  Up 2 minutes  5000/tcp
9056d623d406   localhost:5000/rre2425-redis-multi:latest "redis-server '--pro..." 2 minutes ago  Up 2 minutes  6379/tcp
cecc02373e6a   registry:2.8.3                      "/entrypoint.sh /etc..." 3 minutes ago  Up 3 minutes  0.0.0.0:5000->5000/tcp, :::5000->5000/tcp

vagrant@rre2425-docker:~$
```

- Comprueba el acceso desde tu **host**: <http://localhost:8080>
  - ¿Funciona? ¿Por qué no?
- Inspecciona a qué red(es) están conectados todos los contenedores en ejecución, su tipo y qué dirección IP tienen
  - Lista también las redes disponibles (*docker network ls*)
  - ¿Qué diferencias observas con ejercicios anteriores?
  - Inspecciona también ambos servicios (*docker service inspect*)
- Elimina ambos servicios
  - *docker service rm redis webapp*



# Ejercicio 7

43

- Crea una red *multihost* usando el driver *overlay* y a continuación lista las redes
  - Como estamos en un entorno distribuido (un clúster Swarm) debemos crear una red *multihost* (driver *overlay*) a la que se conecten nuestros servicios y contenedores

```
vagrant@rre2425-docker:~$ docker network create --driver overlay mynetwork
yz5cwgf0g5a1u8j0rt53vmj36
vagrant@rre2425-docker:~$ docker network ls
NETWORK ID          NAME                DRIVER             SCOPE
e9ff53b14721        bridge              bridge              local
520a91eb672f        docker_gwbridge     bridge              local
941b952803d8        host                host                local
sz0qdxolztr7        ingress             overlay             swarm
yz5cwgf0g5a1        mynetwork            overlay             swarm
3c66a128cc88        none                null                local
vagrant@rre2425-docker:~$
```



Crea la red únicamente en la VM manager. Docker Swarm se encargará de replicar la red en los nodos *worker*

- Crea ambos servicios usando el parámetro *--network* para conectarlos a tu red
  - También sería posible "actualizar" un servicio ya en ejecución y conectarlo a la red con el comando: *docker service update --network-add*
- Comprueba de nuevo el acceso desde tu **host**: <http://localhost:8080>
  - ¿Por qué el ejercicio 6 funcionó sin necesidad de crear ninguna red adicional?
  - Puedes comprobar también que la red *multihost* existe en la VM *worker*



# Ejercicio 7

44

- Inspecciona el servicio redis (*docker service inspect --pretty redis*)

```
vagrant@rre2425-docker:~$ docker service inspect --pretty redis
```

```
ID:                oa6sbq1pblkdctq26da9lbgp2
Name:              redis
Service Mode:      Replicated
Replicas:          1
Placement:
UpdateConfig:
  Parallelism:      1
  On failure:       pause
  Monitoring Period: 5s
  Max failure ratio: 0
  Update order:     stop-first
RollbackConfig:
  Parallelism:      1
  On failure:       pause
  Monitoring Period: 5s
  Max failure ratio: 0
  Rollback order:   stop-first
ContainerSpec:
  Image:            localhost:5000/rre2425-redis-multi:latest@sha256
  Init:             false
Mounts:
  Target:           /data
  Source:           redis-vol
  ReadOnly:         false
  Type:            volume
Resources:
  Networks: mynetwork
  Endpoint Mode:  vip

vagrant@rre2425-docker:~$
```

Comprueba el montaje del  
volumen y la configuración  
de red del servicio redis





# Ejercicio 7

45



- Inspecciona el servicio webapp y comprueba el acceso a la aplicación web desde la **VM manager** usando el comando **curl -s**

```
vagrant@rre2425-docker:~$ docker service inspect --pretty webapp
```

```
ID:                ruousdbsfkqkpgj4awnxwlqbo
Name:              webapp
Service Mode:      Replicated
Replicas:          5
Placement:
UpdateConfig:
Parallelism:       1
On failure:        pause
Monitoring Period: 5s
Max failure ratio: 0
Update order:      stop-first
RollbackConfig:
Parallelism:       1
On failure:        pause
Monitoring Period: 5s
Max failure ratio: 0
Rollback order:    stop-first
ContainerSpec:
Image:             localhost:5000/rre2425-web:latest@sha256:bb381c9b4b0f45e9e87f96
Init:              false
Mounts:
Target:            /src
Source:            /vagrant/ej4/src
ReadOnly:          false
Type:              bind
Resources:
Networks: mynetwork
Endpoint Mode: vip
Ports:
PublishedPort = 80
Protocol = tcp
TargetPort = 5000
PublishMode = ingress
```

Comprueba el montaje  
*bind* y la configuración  
de red del servicio  
webapp



Acceso funcionando  
desde la VM manager



```
vagrant@rre2425-docker:~$ curl -s localhost | grep "Redis counter"
<p>New Redis counter: <span style="color: green;">4</span></p>
vagrant@rre2425-docker:~$
```



# Ejercicio 7

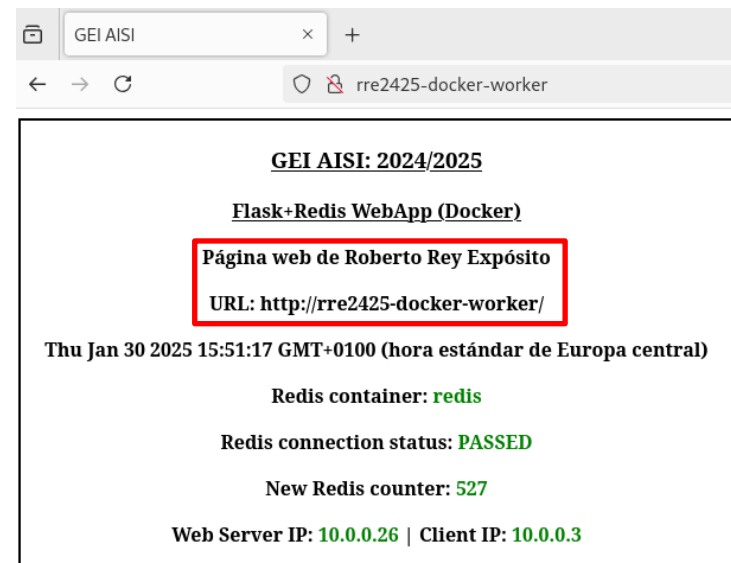
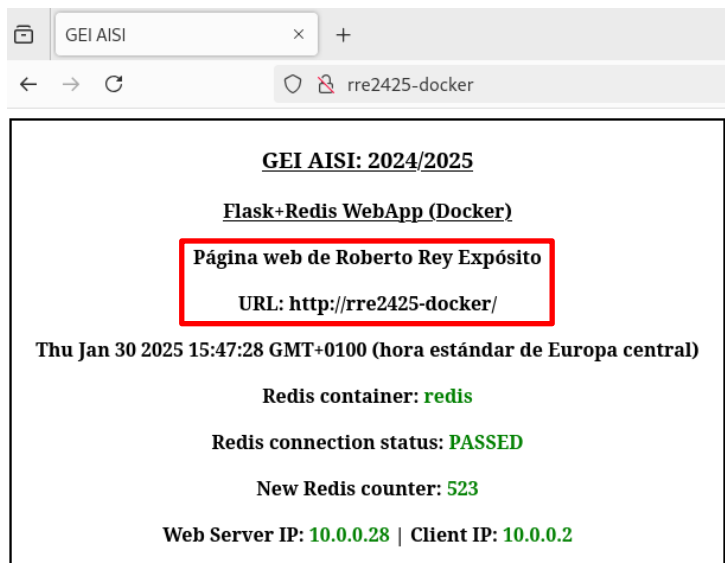
46



- Comprueba el acceso desde tu **host** mediante estas URLs y tu prefijo:

- <http://X-docker>
- <http://X-docker-worker>

¿Funcionaría si accedes a los puertos redireccionados en el *host* (8080 y 9090, revisa el *Vagrantfile*)?



- Para terminar la práctica, elimina los servicios desde la **VM manager** (elimina también el contenedor que ejecuta el *Docker Registry*)
  - `docker service rm redis webapp`
  - `docker rm -f registry`