



# 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))

Jorge Veiga ([jorge.veiga@udc.es](mailto:jorge.veiga@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 denominados **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>



# Consideraciones iniciales

4

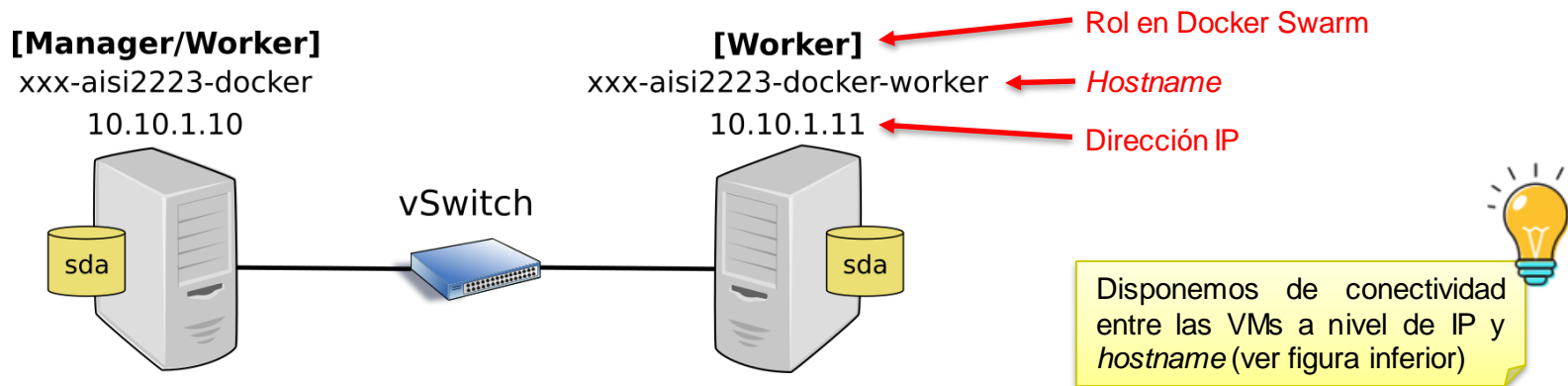
- Clona el [repositorio de la práctica 2](#) para obtener los ficheros necesarios para realizar los ejercicios propuestos
  - **Recuerda:** sin espacios, acentos, eñes o caracteres “raros” en la ruta 
- **Debes usar el Vagrant box** creado en la práctica previa con Packer junto con el *Vagrantfile* disponible en el repositorio de esta práctica 2
  - Este *Vagrantfile* despliega **dos VMs** (*manager/worker*) interconectadas formando un mini clúster que usaremos con Docker Swarm (ejercicios 4 y 5)
    - Ver siguiente transparencia
  - **Modifica el *Vagrantfile* para cambiar el box y el hostname de las VMs** 
    - **Debes sustituir "xxx" por tus iniciales correspondientes (líneas 13-15)**
- Recuerda que la carpeta del proyecto Vagrant (donde reside el fichero *Vagrantfile*) se comparte con la ruta **/vagrant** en las VMs
  - Resulta muy útil para poder editar ficheros desde el *host* y acceder a ellos desde las VMs



# Consideraciones iniciales

5

- Esquema gráfico del despliegue con Vagrant:



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

```
vagrant@rre-aisi2223-docker:~$ ping -c 2 rre-aisi2223-docker-worker
PING rre-aisi2223-docker-worker (10.10.1.11) 56(84) bytes of data.
64 bytes from rre-aisi2223-docker-worker (10.10.1.11): icmp_seq=1 ttl=64 time=0.575 ms
64 bytes from rre-aisi2223-docker-worker (10.10.1.11): icmp_seq=2 ttl=64 time=0.335 ms

--- rre-aisi2223-docker-worker ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.335/0.455/0.575/0.120 ms
vagrant@rre-aisi2223-docker:~$
```



# Justificación de la práctica

6

- La realización de esta práctica se justificará de la siguiente forma:
  - Fichero *Dockerfile* resultado del **ejercicio 1**
  - Documento en formato PDF que incluya las capturas de pantalla indicadas para demostrar la realización de la **parte principal de cada ejercicio**
    - **Debes incluir capturas similares a las mostradas en las transparencias:**
      - **9, 11 (EJ1); 17, 18 (EJ2); 28 (EJ3); 32, 34 (EJ4); 36, 38 (EJ5)**



Busca este icono en la parte superior derecha



**IMPORTANTE**



- **ENTREGA** a través de Moodle: **06/03 (15:30)**
- **ES OBLIGATORIO** usar la nomenclatura que se propone para nombrar los recursos y debe apreciarse sin confusión en las capturas aportadas
  - **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 esta práctica



# 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)
- Usando la plantilla *Dockerfile* disponible en el repositorio:
  - Como imagen de base (instrucción FROM) usa **debian:buster-slim**
  - Actualiza los repositorios (*apt-get update*) e instala el paquete **wget** (*apt-get install*)
  - Descarga y compila el código fuente de Redis
    - Descarga: `wget -O redis.tar.gz https://download.redis.io/releases/redis-7.0.7.tar.gz`
    - Descomprime los ficheros de código fuente 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 fuente e instala los binarios: `make && make install`
  - Minimiza el tamaño de la imagen resultante, eliminando:
    - El fichero descargado (`redis.tar.gz`) y el directorio con el código fuente de Redis
    - Los paquetes instalados y sus dependencias (`apt-get purge -y --autoremove wget gcc make`)
    - La caché local del gestor de paquetes (`apt-get clean all`)



# Ejercicio 1

8

- Usando la plantilla *Dockerfile* disponible en el repositorio (cont.):
  - Establece el directorio de trabajo por defecto (WORKDIR): `/data`
    - Dicho directorio debe pertenecer al usuario y grupo `redis`
    - Debes crear previamente el usuario y grupo `redis`:
      - `groupadd -r redis && useradd -r -g redis redis`
  - Establece el usuario que ejecutará el contenedor como `redis`
  - Expón el puerto TCP 6379 (puerto por defecto donde escucha el servidor Redis)
  - Establece el punto de entrada usando formato `exec: redis-server`
    - Añade `--protected-mode no` como parámetro al binario anterior
- Crea la imagen (*docker build*) usando tu fichero *Dockerfile*
  - **Debes nombrarla siguiendo el formato: xxx-aisi2223-redis** 
  - Sustituye **xxx** por las iniciales de tu nombre y apellidos
- Una vez creada, lista las imágenes Docker disponibles
  - `docker image ls`
- Determina su número de capas consultando su "historial"
  - `docker history`





# Ejercicio 1



¿Cuál es el tamaño resultante de tu imagen? Fíjate en el tamaño de las capas en la salida del comando *docker history*. No te preocupes si tu imagen "pesa" más, sigue adelante con la práctica

```
vagrant@rre-aisi2223-docker:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rre-aisi2223-redis	latest	a8570ba636a0	24 seconds ago	124MB
debian	buster-slim	2e397d78b126	2 days ago	69.3MB

```
vagrant@rre-aisi2223-docker:~$ docker history rre-aisi2223-redis
```

IMAGE	CREATED	CREATED BY	ENTRYPOINT	SIZE
a8570ba636a0	48 seconds ago	/bin/sh -c #(nop)	ENTRYPOINT ["redis-server...]	0B
2d075c5efa35	48 seconds ago	/bin/sh -c #(nop)	EXPOSE 6379/tcp	0B
850aad47d805	48 seconds ago	/bin/sh -c #(nop)	USER redis	0B
ac9548d56890	48 seconds ago	/bin/sh -c #(nop)	WORKDIR /data	0B
f8df1ebb8d3f	48 seconds ago	/bin/sh -c mkdir \$REDIS_DATA && chown redis:...		0B
daaf5d3a9fbe	49 seconds ago	/bin/sh -c #(nop)	ENV REDIS_DATA=/data	0B
b0a7a72f087c	50 seconds ago	/bin/sh -c apt-get update && apt-get instal...		54.4MB
2e397d78b126	2 days ago	/bin/sh -c #(nop)	CMD ["bash"]	0B
<missing>	2 days ago	/bin/sh -c #(nop)	ADD file:67ceb946e54c26c50...	69.3MB

```
vagrant@rre-aisi2223-docker:~$
```

¿Por qué coinciden los dos identificadores de imagen resaltados en la figura?





# 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 xxx-aisi2223-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@rre-aisi2223-docker:~$ docker run -d --rm --name redis-server rre-aisi2223-redis
95f99ed3b8482225a78373d6e70871b407abdddef2a5164f815dbff4cee0f9912
vagrant@rre-aisi2223-docker:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
95f99ed3b848	rre-aisi2223-redis	"redis-server '--pro..."	6 seconds ago	Up 5 seconds	6379/tcp	redis-server

```
vagrant@rre-aisi2223-docker:~$ docker logs redis-server
1:C 13 Jan 2023 15:49:33.842 # o000o000o000o Redis is starting o000o000o000o
1:C 13 Jan 2023 15:49:33.842 # Redis version=7.0.7 bits=64, commit=00000000, modified=0, pid=1, just started
1:C 13 Jan 2023 15:49:33.842 # Configuration loaded
1:M 13 Jan 2023 15:49:33.842 * monotonic clock: POSIX clock_gettime
1:M 13 Jan 2023 15:49:33.843 * Running mode=standalone, port=6379.
1:M 13 Jan 2023 15:49:33.843 # Server initialized
1:M 13 Jan 2023 15:49:33.843 # WARNING Memory overcommit must be enabled! Without it, a background save or replicat
t can can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. T
sctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 13 Jan 2023 15:49:33.843 * Ready to accept connections
vagrant@rre-aisi2223-docker:~$ docker inspect --format "{{.NetworkSettings.IPAddress}}" redis-server
172.17.0.2
vagrant@rre-aisi2223-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 debes **redefinir el punto de entrada (`--entrypoint`)** de la imagen para ejecutar el binario `redis-cli` (que es el cliente Redis) en vez del binario `redis-server` (que es el servidor)
    - Ver [parámetro `--entrypoint`](#) del comando `docker run`
    - Sintaxis del cliente Redis: `redis-cli -h <SERVER_IP> ping`
  - Debes saber la IP del contenedor que ejecuta el servidor Redis y usar el comando `ping` desde el contenedor cliente para comprobar la conectividad entre ambos
    - Recibirás un PONG como respuesta desde el servidor Redis
  - Muestra información sobre todos los contenedores (`docker ps -a`)
    - Razona sobre el **estado actual** de los contenedores servidor y cliente de Redis

```
vagrant@rre-aisi2223-docker:~$ docker run --name redis-client --entrypoint
```

```
PONG
```

```
vagrant@rre-aisi2223-docker:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bff88086e39d	rre-aisi2223-redis	"redis-cli -h 172.17..."	5 seconds ago	Exited (0) 4 seconds ago		redis-client
95f99ed3b848	rre-aisi2223-redis	"redis-server '--pro..."	3 minutes ago	Up 3 minutes	6379/tcp	redis-server

```
vagrant@rre-aisi2223-docker:~$
```

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





# Ejercicio 1

12

- Fíjate que en este ejercicio no hemos proporcionado ningún parámetro a los contenedores relacionado con su configuración de red
  - **Es decir, ningún parámetro de *docker run* hace referencia a la configuración de red**
- Sin embargo, hemos comprobado que el contenedor cliente puede comunicarse con el contenedor servidor sin ningún problema a nivel de dirección IP
  - Por tanto, ambos contenedores deberían estar en la misma red
- Con el servidor Redis en ejecución, **inspecciona su contenedor** (sin filtrar la salida) para determinar cuál es 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
  - Lista las redes disponibles para determinar su tipo (*name, driver y scope*)
    - *docker network ls*
- Para finalizar, elimina los contenedores
  - *docker rm -f redis-server redis-client*



NETWORK ID	NAME	DRIVER	SCOPE
f01e1b695404	bridge	bridge	local
26665acf4d55	host	host	local
086e0e004b3e	none	null	local

```
vagrant@rre-aisi2223-docker:~$
```

```

vagrant@rrre-aisi2223-docker:~$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS              PORTS          NAMES
bff88086e39d   rre-aisi2223-redis   "redis-cli -h 172.17..." 11 minutes ago Exited (0) 11 minutes ago      6379/tcp      redis-client
95f99ed3b848   rre-aisi2223-redis   "redis-server '--pro..." 14 minutes ago Up 14 minutes          6379/tcp      redis-server
vagrant@rrre-aisi2223-docker:~$ docker rm -f redis-server redis-client
redis-server
redis-client
vagrant@rrre-aisi2223-docker:~$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS              PORTS          NAMES
vagrant@rrre-aisi2223-docker:~$

```

Eliminando los contenedores

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

## Eliminando los contenedores



# Ejercicio 2

14

- **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
- **Guía orientativa:**
  - Entre los ficheros de este ejercicio tienes un *Dockerfile*, el cual (ver figura inferior):
    - Usa una imagen base basada en [Alpine Linux](#) con soporte para Python
    - Instala el framework Flask y el cliente Python para acceder a Redis usando k
    - Luego instala el comando *curl*, copia el código de la aplicación Python a ejecutar (fichero *counter.py* disponible en el directorio *src*) y expone el puerto 5000
    - Añade un HEALTHCHECK para reportar la "salud" (estado) del contenedor mediante *curl*
    - Finalmente, establece el punto de entrada para ejecutar la aplicación web con Flask

```
FROM python:3.11.1-alpine3.17
ENV FLASK_APP counter.py
ENV FLASK_RUN_HOST 0.0.0.0
ENV FLASK_DEBUG True
RUN pip install flask && pip install redis
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?



# Ejercicio 2

15

- Guía orientativa (cont.):
  - Por otro lado, dispones del código Python de la aplicación web Flask: **counter.py**
    - **Debes modificar el mensaje que devuelve la aplicación web (pon tu nombre y apellidos)**

```
import time
import redis
from flask import Flask
```

```
app = Flask(__name__)
```

```
cache = redis.Redis(host='redis-server', port=6379)
```

Datos de conexión a la base de datos. Fíjate en el nombre del *host* y puerto al que se conecta

```
def get_hit_count():
```

```
    retries = 3
```

```
    while True:
```

```
        try:
```

```
            return cache.incr('hits')
```

```
        except redis.exceptions.ConnectionError as exc:
```

```
            if retries == 0:
```

```
                raise exc
```

```
            retries -= 1
```

```
            time.sleep(0.5)
```

```
@app.route('/')
def hello():
```

```
    count = get_hit_count()
```

```
    return 'GEI AISI 2022/2023: counter for [alumno/a] ({} times)\n'.format(count)
```

Sustituye [alumno/a] por tu nombre y apellidos





# Ejercicio 2

16

- Guía orientativa (cont.):
  - Crea una imagen Docker usando el *Dockerfile* y lista las imágenes disponibles
    - **Debes nombrarla siguiendo el formato: xxx-aisi2223-webcounter**
    - `docker build -t xxx-aisi2223-webcounter /vagrant/ej2`

```
vagrant@rre-aisi2223-docker:~$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rre-aisi2223-webcounter	latest	e53c3e0e54b1	56 seconds ago	76.3MB
rre-aisi2223-redis	latest	a8570ba636a0	53 minutes ago	124MB
debian	buster-slim	2e397d78b126	2 days ago	69.3MB
python	3.11.1-alpine3.17	f1f4a546ac91	3 days ago	52.4MB

```
vagrant@rre-aisi2223-docker:~$
```

- Inicia un contenedor con nombre **redis-server** para ejecutar el servidor Redis **en segundo plano** usando la imagen creada en el Ejercicio 1

```
vagrant@rre-aisi2223-docker:~$ docker run --rm -d --name redis-server rre-aisi2223-redis
d564a444789ef24bc96645473dc3104513d6801bed690e9c1b1e55dd9d323a5d
vagrant@rre-aisi2223-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d564a444789e	rre-aisi2223-redis	"redis-server '--pro..."	4 seconds ago	Up 3 seconds	6379/tcp	redis-server

```
vagrant@rre-aisi2223-docker:~$
```



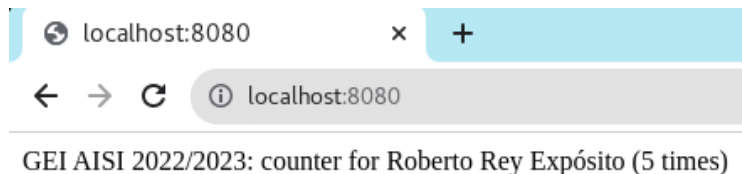


# Ejercicio 2

17



- Guía orientativa (cont.):
  - 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 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 su dirección IP
      - Fíjate cómo se conecta la aplicación web a la base de datos (línea 6 de *counter.py*)
- Comprueba el acceso a la aplicación web desde de 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



Peticiones GET recibidas en el contenedor web-app →



¿Por qué hay accesos desde dos IPs distintas?

```
vagrant@rre-aisi2223-docker:~$ docker run --rm --name webapp --link redis-server
* Serving Flask app 'counter.py'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
* 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: 130-139-053
10.0.2.2 - - [13/Jan/2023 17:06:53] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Jan/2023 17:06:56] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Jan/2023 17:06:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2023 17:07:21] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [13/Jan/2023 17:07:26] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2023 17:07:51] "GET / HTTP/1.1" 200 -
```



## Ejercicio 2

18



- Detén el contenedor webapp (CTRL+C) y ejecútalo ahora en **segundo plano**
  - Comprueba que ambos contenedores están en ejecución y que el contenedor **webapp** reporta **"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
  - Comprueba ahora el acceso **desde la propia VM** usando el comando: **`curl localhost:X`**
    - ¿A qué puerto X deberás hacer la petición? ¿8080? ¿80? ¿5000? ¿Otro?
    - Haz pruebas y trata de razonar el/los correcto/s

```
vagrant@rre-aisi2223-docker:~$ docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED        STATUS                        PORTS
77235d112b76   rre-aisi2223-webcounter   "flask run"             51 seconds ago Up 50 seconds (healthy)    0.0.0.0:80->5000/tcp
a03a43fe7221   rre-aisi2223-redis        "redis-server '--pro..." 54 seconds ago Up 53 seconds              6379/tcp

vagrant@rre-aisi2223-docker:~$
vagrant@rre-aisi2223-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: 110-283-267
10.0.2.2 - - [16/Jan/2023 11:41:55] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [16/Jan/2023 11:41:56] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [16/Jan/2023 11:41:56] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [16/Jan/2023 11:41:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Jan/2023 11:42:21] "GET / HTTP/1.1" 200 -
vagrant@rre-aisi2223-docker:~$
```



## Ejercicio 2

19

- 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
    - ¿Están conectados a la misma red? ¿Por qué no funciona sin definir el *link*?
  - Modifica el código fuente para que la conexión a la base de datos se realice usando la IP del contenedor en vez de usar el nombre 'redis-server' (línea 6 de *counter.py*)
    - Al modificar el código, hay que volver a generar la imagen xxx-aisi2223-webcounter
  - Ejecuta un nuevo contenedor webapp **sin definir el *link* y comprueba que sí funciona**
    - **Cambia de nuevo *counter.py* para que la conexión a la base de datos se realice usando 'redis-server' y no la dirección IP del contenedor y vuelve a crear la imagen**
- Desde webapp, comprueba la conectividad con redis-server mediante ***ping***
  - 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 (ver siguientes transparencias)



# Ejercicio 2

20

```
vagrant@rre-aisi2223-docker:~$ docker run --rm -d --name webapp --link redis-server -p 3d2c117a6fda61c9f29f9048cd394f3de3e58039ad70d6fe61e49f91636db093
```

```
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 2 172.17.0.2
```

```
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.179 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.045 ms
```

Ejecutamos el comando *ping* dentro del contenedor webapp

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

```
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.045/0.112/0.179 ms
```

```
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 2 redis-server
```

```
PING redis-server (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.045 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.042 ms
```

Pruebas con *ping* definiendo el *link*



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

```
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.042/0.043/0.045 ms
```

```
vagrant@rre-aisi2223-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     redis-server c5a83a3dc7fe
```

```
172.17.0.3     3d2c117a6fda
```

```
vagrant@rre-aisi2223-docker:~$ docker stop webapp
webapp
```

```
vagrant@rre-aisi2223-docker:~$
```

Mostramos el fichero de *hosts* del contenedor webapp



# Ejercicio 2

21

```
vagrant@rre-aisi2223-docker:~$ docker run --rm -d --name webapp -p 80:5000 rre-aisi2223-webcounter
fe4c99e5856a996ac08f04dc38af93220be6928ea23dc41090f4769a8a13ffde
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 3 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.102 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.041 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.041 ms

--- 172.17.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.041/0.061/0.102 ms
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 2 redis-server
ping: bad address 'redis-server'
vagrant@rre-aisi2223-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      fe4c99e5856a
vagrant@rre-aisi2223-docker:~$ docker stop webapp
webapp
vagrant@rre-aisi2223-docker:~$
```



Pruebas con *ping* sin definir el *link*



# Ejercicio 2

22

```
vagrant@rre-aisi2223-docker:~$ docker run --rm -d --name webapp --link redis-server:aisi -p
0558c96ec71ecef7b245d774deb4072e41d4a98de3620a7f92602653c52701f
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 3 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.095 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.045 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.044 ms

--- 172.17.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.044/0.061/0.095 ms
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 2 redis-server
PING redis-server (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.044 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.044 ms

--- redis-server ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.044/0.044/0.044 ms
vagrant@rre-aisi2223-docker:~$ docker exec webapp ping -c 2 aisi
PING aisi (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.043 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.040 ms

--- aisi ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.040/0.041/0.043 ms
vagrant@rre-aisi2223-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 c5a83a3dc7fe redis-server
172.17.0.3       0558c96ec71e
vagrant@rre-aisi2223-docker:~$ docker stop webapp
webapp
vagrant@rre-aisi2223-docker:~$
```



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



# Ejercicio 3

23

- **Despliegue de una aplicación multicontenedor usando Docker Compose**
  - La aplicación a desplegar será la misma que la del ejercicio 2
- **Guía orientativa:**
  - **Para este ejercicio y siguientes, elimina la instrucción HEALTHCHECK del *Dockerfile* del ejercicio 2 y vuelve a crear la imagen **xxx-aisi2223-webcounter****
    - Definiremos el *healthcheck* en el propio fichero *compose.yml*
  - Instala [Docker Compose](#) en la VM:
    - El repositorio de la práctica contiene un *script bash* en el directorio *provisioning* con los comandos necesarios para descargar Docker Compose y realizar su instalación
    - Modifica el *Vagrantfile* y habilita la fase de aprovisionamiento de tipo *shell* que ejecuta el *script* para instalar Docker Compose
    - Desde tu *host*, fuerza el aprovisionamiento
      - `vagrant up --provision`
    - Conéctate a la VM y comprueba la instalación ejecutando
      - `docker compose version`

```
vagrant@rre-aisi2223-docker:~$ docker compose version
Docker Compose version v2.15.1
vagrant@rre-aisi2223-docker:~$
```



# Ejercicio 3

24

- Guía orientativa (cont.):
  - Usando la plantilla YAML **compose.yml**, completa la definición de los **dos servicios** que componen la aplicación multicontenedor a desplegar
    - Para Redis, define un **servicio** llamado **redis** usando la imagen del ejercicio 1
    - Para la aplicación web, define un **servicio** llamado **webapp** usando la imagen del ejercicio 2
      - 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 ese nombre
        - **Asegúrate que el código de la aplicación (counter.py) utiliza 'redis-server' como hostname de conexión a la base de datos**
      - 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 ver los cambios "al vuelo"





# Ejercicio 3

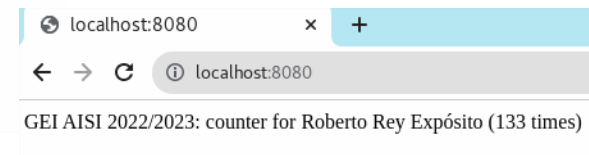
25

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

```
vagrant@rre-aisi2223-docker:~$ docker compose -f /vagrant/ej3/compose.yml up
[+] Running 3/3
  # Network ej3_default      Created
  # Container ej3-redis-1    Created
  # Container ej3-webapp-1    Created
Attaching to ej3-redis-1, ej3-webapp-1
ej3-redis-1 | 1:C 16 Jan 2023 09:48:25.502 # o000o000o000o Redis is starting o000o000o000o
ej3-redis-1 | 1:C 16 Jan 2023 09:48:25.502 # Redis version=7.0.7, bits=64, commit=00000000,
ej3-redis-1 | 1:C 16 Jan 2023 09:48:25.502 # Configuration loaded
ej3-redis-1 | 1:M 16 Jan 2023 09:48:25.503 * monotonic clock: POSIX clock_gettime
ej3-redis-1 | 1:M 16 Jan 2023 09:48:25.503 * Running mode=standalone, port=6379.
ej3-redis-1 | 1:M 16 Jan 2023 09:48:25.503 # Server initialized
ej3-redis-1 | 1:M 16 Jan 2023 09:48:25.503 # WARNING Memory overcommit must be enabled! With
ej3-redis-1 | eing disabled, it can can also cause failures without low memory condition, see https://github
ej3-redis-1 | = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1'
ej3-redis-1 | 1:M 16 Jan 2023 09:48:25.528 * Ready to accept connections
ej3-webapp-1 | * Serving Flask app 'counter.py'
ej3-webapp-1 | * Debug mode: on
ej3-webapp-1 | WARNING: This is a development server. Do not use it in a production deployment!
ej3-webapp-1 | * Running on all addresses (0.0.0.0)
ej3-webapp-1 | * Running on http://127.0.0.1:5000
ej3-webapp-1 | * Running on http://172.18.0.3:5000
ej3-webapp-1 | Press CTRL+C to quit
ej3-webapp-1 | * Restarting with stat
ej3-webapp-1 | * Debugger is active!
ej3-webapp-1 | * Debugger PIN: 111-185-917
ej3-webapp-1 | 10.0.2.2 - - [16/Jan/2023 09:48:35] "GET / HTTP/1.1" 200 -
ej3-webapp-1 | 10.0.2.2 - - [16/Jan/2023 09:48:36] "GET / HTTP/1.1" 200 -
ej3-webapp-1 | 10.0.2.2 - - [16/Jan/2023 09:48:37] "GET / HTTP/1.1" 200 -
ej3-webapp-1 | 10.0.2.2 - - [16/Jan/2023 09:48:38] "GET / HTTP/1.1" 200 -
ej3-webapp-1 | 127.0.0.1 - - [16/Jan/2023 09:48:40] "GET / HTTP/1.1" 200 -
ej3-webapp-1 | 127.0.0.1 - - [16/Jan/2023 09:48:55] "GET / HTTP/1.1" 200 -
```

Peticiones GET recibidas  
en el contenedor webapp

Acceso desde el host



- Detén la aplicación (CTRL+C) y desplégala de nuevo en **segundo plano**
- Comprueba los servicios y contenedores en ejecución
  - `docker compose ps`
  - `docker ps`



# Ejercicio 3

26

- Comprueba que gracias al montaje *bind* puedes modificar **desde tu host** el código fuente de la aplicación web "al vuelo" (sin reiniciar el servicio)
  - Desde la VM, accede a la aplicación web usando *curl*
  - A continuación, cambia el mensaje a mostrar modificando el fichero *counter.py*
  - Sin detener el servicio, accede de nuevo usando *curl* para obtener el nuevo mensaje

```
vagrant@rre-aisi2223-docker:~$ docker compose -f /vagrant/ej3/compose.yml ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
ej3-redis-1	rre-aisi2223-redis	"redis-server '--pro..."	redis	8 minutes ago	Up 5 minutes	6379/tcp
ej3-webapp-1	rre-aisi2223-webcounter	"flask run"	webapp	8 minutes ago	Up 5 minutes (healthy)	0.0.0.0:80->5000/tcp, :::80->5000/tcp

```
vagrant@rre-aisi2223-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
993c1b086dae	rre-aisi2223-webcounter	"flask run"	8 minutes ago	Up 5 minutes (healthy)	0.0.0.0:80->5000/tcp, :::80->5000/tcp	ej3-webapp-1
56643e3742e4	rre-aisi2223-redis	"redis-server '--pro..."	8 minutes ago	Up 5 minutes	6379/tcp	ej3-redis-1

```
vagrant@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: counter for Roberto Rey Expósito (49 times)
vagrant@rre-aisi2223-docker:~$ sed -i 's/counter for/my new counter for/g' /vagrant/ej2/src/counter.py
vagrant@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (50 times)
vagrant@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (51 times)
vagrant@rre-aisi2223-docker:~$
```

← Modificación del mensaje a mostrar usando el comando *sed*. También puedes modificar el fichero *counter.py* directamente desde tu equipo

- Detén la aplicación y elimina todos sus recursos
  - *docker compose down*



# Ejercicio 3

27

- ¿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 bien en los recursos que crea Docker Compose al desplegar la aplicación

```
vagrant@rre-aisi2223-docker:~$ docker inspect ej3-webapp-1 | grep NetworkID
"NetworkID": "ad4ff47493bc714847b8b6869bc04209f4a474276d7dde805fd17d5719730bca",
vagrant@rre-aisi2223-docker:~$ docker inspect ej3-redis-1 | grep NetworkID
"NetworkID": "ad4ff47493bc714847b8b6869bc04209f4a474276d7dde805fd17d5719730bca",
vagrant@rre-aisi2223-docker:~$ docker network ls
NETWORK ID        NAME          DRIVER        SCOPE
f01e1b695404      bridge       bridge        local
ad4ff47493bc      ej3_default  bridge        local
26665acf4d55      host         host          local
086e0e004b3e      none         null          local
vagrant@rre-aisi2223-docker:~$
```



# Ejercicio 3

28



- Acceso a la aplicación web (**sin link** definido en el fichero YAML)
  - **Para justificar este ejercicio**, ejecuta estos mismos comandos (y en el mismo orden) y adjunta una única captura tal y como se muestra en el ejemplo

```
vagrant@rre-aisi2223-docker:~$ docker compose -f /vagrant/ej3/compose.yml ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
ej3-redis-1	rre-aisi2223-redis	"redis-server '--pro..."	redis	18 minutes ago	Up 18 minutes	6379/tcp
ej3-webapp-1	rre-aisi2223-webcounter	"flask run"	webapp	18 minutes ago	Up 18 minutes (healthy)	0.0.0.0:80->5000/tcp

```
vagrant@rre-aisi2223-docker:~$ cat /vagrant/ej3/compose.yml | grep links
vagrant@rre-aisi2223-docker:~$ cat /vagrant/ej2/src/counter.py | grep "GEI AISI"
return 'GEI AISI 2022/2023: my new counter for Roberto Rey Expósito ({} times)\n'.format(count)
vagrant@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (75 times)
vagrant@rre-aisi2223-docker:~$
```

Link no definido

Mensaje a mostrar por la aplicación web

Acceso a la aplicación web desde la VM

- Para terminar este ejercicio, detén la aplicación y elimina todos sus recursos



# Ejercicio 4

29

- **Despliegue de una aplicación multicontenedor usando Docker Swarm**
  - La aplicación a desplegar es la misma del ejercicio 3
- **Guía orientativa:**
  - Desde la *VM manager*, crea un clúster de contenedores Docker en Swarm Mode
    - `docker swarm init --advertise-addr 10.10.1.10`
    - Esta VM actuará como *manager* y *worker* (configuración por defecto)
  - Comprueba que tu clúster dispone de un nodo activo
    - `docker node ls`

```
vagrant@rre-aisi2223-docker:~$ docker swarm init --advertise-addr 10.10.1.10
Swarm initialized: current node (iv6srcb857b2ukx6muxx9xjum) is now a manager.
```

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

```
docker swarm join --token SWMTKN-1-08qjedw6fjg7wfjhq2corvzr9mdvf5ksg2wz2tih8aj0a9sk2t-cvr43rybsgxb8t93zoqd8q0t 10.10.1.10:2377
```

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

```
vagrant@rre-aisi2223-docker:~$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
iv6srcb857b2ukx6muxx9xjum *	rre-aisi2223-docker	Ready	Active	Leader	20.10.22

Comando para añadir nodos *worker* al enjambre. **Cópialo**

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

Estamos conectados  
a la VM *worker*

```
vagrant@rre-aisi2223-docker-worker:~$ docker swarm join --token SWMTKN-1-08qjedw6fjg7wfjhq2corvzr9mdvf5ksg2wz2tih8aj0a9sk2t-cvr43rybsgxb8t93zoqd8q0t 10.10.1.10:2377
This node joined a swarm as a worker.
vagrant@rre-aisi2223-docker-worker:~$
```

Token de acceso  
recortado



# Ejercicio 4

30

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

```
vagrant@rre-aisi2223-docker:~$ docker node ls
ID                                HOSTNAME                STATUS    AVAILABILITY    MANAGER STATUS
iv6srcb857b2ukx6muxx9xjum *    rre-aisi2223-docker     Ready    Active           Leader
xn47rq52jxrcdfwqix1qvb87y      rre-aisi2223-docker-worker Ready    Active
```

- 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 la *VM manager*, ejecuta:
  - `docker service create --name registry -p 5000:5000 registry:2` ← Desplegamos el *Docker Registry* como un servicio que se ejecuta en el clúster Swarm
- Etiquetamos las imágenes con el *hostname* y puerto del repositorio:
  - `docker tag rre-aisi2223-redis localhost:5000/rre-aisi2223-redis`
  - `docker tag rre-aisi2223-webcounter localhost:5000/rre-aisi2223-webcounter`
- Subimos las imágenes que necesitamos al repositorio:
  - `docker push localhost:5000/rre-aisi2223-redis`
  - `docker push localhost:5000/rre-aisi2223-webcounter`



# Ejercicio 4

31

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

```
vagrant@rre-aisi2223-docker:~$ curl localhost:5000/v2/_catalog
{"repositories":["rre-aisi2223-redis","rre-aisi2223-webcounter"]}
vagrant@rre-aisi2223-docker:~$
```

- Usando la plantilla YAML ***stack.yml***, completa la definición de los **servicios**:
  - Usa las imágenes creadas en ejercicios previos y alojadas en el *Docker Registry* local
  - 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
  - Modifica **webapp** para definirlo como un servicio replicado usando **tres** instancias
- Despliega el *stack* en el clúster Swarm
  - `docker stack deploy -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 4

32



```
vagrant@rre-aisi2223-docker:~$ docker stack deploy -c /vagrant/ej4/stack.yml mystack
```

```
Creating network mystack_default
```

```
Creating service mystack_redis
```

```
Creating service mystack_webapp
```

```
vagrant@rre-aisi2223-docker:~$ docker stack ls
```

```
NAME      SERVICES  ORCHESTRATOR
mystack   2          Swarm
```

```
vagrant@rre-aisi2223-docker:~$
```

```
vagrant@rre-aisi2223-docker:~$ docker stack services mystack
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
rutyo9h5kq7i	mystack_redis	replicated	1/1	localhost:5000/rre-aisi2223-redis:latest	
awm02vjsqlxr	mystack_webapp	replicated	3/3	localhost:5000/rre-aisi2223-webcounter:latest	*:80->5000/tcp

```
vagrant@rre-aisi2223-docker:~$ docker stack ps mystack
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
9iwbwlwmczsp	mystack_redis.1	localhost:5000/rre-aisi2223-redis:latest	rre-aisi2223-docker	Running	Running 4 minutes ago
fvfoh97req16	mystack_webapp.1	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker-worker	Running	Running 4 minutes ago
rcms55yfoz34	mystack_webapp.2	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker	Running	Running 4 minutes ago
kr3eekuv4veq	mystack_webapp.3	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker-worker	Running	Running 4 minutes ago

```
vagrant@rre-aisi2223-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
472ec1311579	localhost:5000/rre-aisi2223-redis:latest	"redis-server '--pro..."	4 minutes ago	Up 4 minutes	6379/tcp	mystack
a75f3c8de76f	localhost:5000/rre-aisi2223-webcounter:latest	"flask run"	4 minutes ago	Up 4 minutes (healthy)	5000/tcp	mystack
5eeb69277e71	registry:latest	"/entrypoint.sh /etc..."	25 minutes ago	Up 25 minutes	5000/tcp	registr

```
vagrant@rre-aisi2223-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@rre-aisi2223-docker-worker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
c8236eb63b65	localhost:5000/rre-aisi2223-webcounter:latest	"flask run"	13 minutes ago	Up 12 minutes (healthy)	5000/tcp
2656cd4d74c8	localhost:5000/rre-aisi2223-webcounter:latest	"flask run"	13 minutes ago	Up 12 minutes (healthy)	5000/tcp



Podría ocurrir que obtengas una distribución diferente de los contenedores webapp y redis que se ejecutan en cada VM (p.e. contenedor de redis ejecutándose en la VM worker), ya que eso dependerá de cómo Docker Swarm planifica las réplicas





# Ejercicio 4

33

- Comprueba el acceso desde tu **host**: <http://localhost:8080>
  - Realiza múltiples accesos (F5) y revisa los logs de los tres contenedores del servicio webapp para comprobar cómo se distribuyen las peticiones entre los contenedores
- Comprueba el acceso desde la **VM manager** usando el comando **curl**

```
vagrant@rre-aisi2223-docker-worker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (287 times)
vagrant@rre-aisi2223-docker-worker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (289 times)
vagrant@rre-aisi2223-docker-worker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (290 times)
vagrant@rre-aisi2223-docker-worker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (291 times)
```

- 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 4



34

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

```
vagrant@rre-aisi2223-docker:~$ docker stack services mystack
ID            NAME          MODE          REPLICAS  IMAGE                                  PORTS
ruty09h5kq7i  mystack_redis replicated    1/1        localhost:5000/rre-aisi2223-redis:latest
awm02vjsqlxr  mystack_webapp replicated    3/3        localhost:5000/rre-aisi2223-webcounter:latest  *:80->5000/tcp

vagrant@rre-aisi2223-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 converged
vagrant@rre-aisi2223-docker:~$ docker stack services mystack
ID            NAME          MODE          REPLICAS  IMAGE                                  PORTS
ruty09h5kq7i  mystack_redis replicated    1/1        localhost:5000/rre-aisi2223-redis:latest
awm02vjsqlxr  mystack_webapp replicated    4/4        localhost:5000/rre-aisi2223-webcounter:latest  *:80->5000/tcp

vagrant@rre-aisi2223-docker:~$ docker stack ps mystack
ID            NAME          IMAGE
9iwbwlwmczsp  mystack_redis.1 localhost:5000/rre-aisi2223-redis:latest
jvf0h97req16  mystack_webapp.1 localhost:5000/rre-aisi2223-webcounter:latest
rcms55yfoz34  mystack_webapp.2 localhost:5000/rre-aisi2223-webcounter:latest
kr3seekuv4veq mystack_webapp.3 localhost:5000/rre-aisi2223-webcounter:latest
nt1dhl0qww5   mystack_webapp.4 localhost:5000/rre-aisi2223-webcounter:latest

vagrant@rre-aisi2223-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 converged
vagrant@rre-aisi2223-docker:~$ docker stack services mystack
ID            NAME          MODE          REPLICAS  IMAGE                                  PORTS
ruty09h5kq7i  mystack_redis replicated    1/1        localhost:5000/rre-aisi2223-redis:latest
awm02vjsqlxr  mystack_webapp replicated    2/2        localhost:5000/rre-aisi2223-webcounter:latest

vagrant@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (478 times)
vagrant@rre-aisi2223-docker:~$
```

NODE	DESIRED STATE	CURRENT STATE
rre-aisi2223-docker	Running	Running 35 min
rre-aisi2223-docker-worker	Running	Running 35 min
rre-aisi2223-docker	Running	Running 35 min
rre-aisi2223-docker-worker	Running	Running 35 min
rre-aisi2223-docker	Running	Running 30 sec

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



# Ejercicio 5

35

- **Despliegue de servicios Docker en un clúster Swarm**
  - Alternativa no declarativa (sin usar un fichero YAML) de desplegar servicios
  - Ya la hemos usado al desplegar el *Docker Registry* anteriormente
  - Desplegaremos los mismos servicios que en el ejercicio 4
- Guía orientativa:
  - Crea dos servicios Swarm usando el comando: ***docker service create***
    - Servicio con nombre **redis** con una única instancia
    - 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 5

36



```
vagrant@rre-aisi2223-docker:~$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
r6as0t0q3skk	redis	replicated	1/1	localhost:5000/rre-aisi2223-redis:latest	
70sia330t3f3	registry	replicated	1/1	registry:latest	*:5000->5000/tcp
nr3nvt0vhf5	webapp	replicated	5/5	localhost:5000/rre-aisi2223-webcounter:latest	*:80->5000/tcp

```
vagrant@rre-aisi2223-docker:~$ docker service ps redis webapp
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
mn8om3only36	redis.1	localhost:5000/rre-aisi2223-redis:latest	rre-aisi2223-docker-worker	Running	Running 3 minutes a
n3jpalbps831	webapp.1	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker-worker	Running	Running about a min
m1lxz9zezpmw	webapp.2	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker-worker	Running	Running about a min
9gu7pv9pmxg6	webapp.3	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker	Running	Running about a min
vvi22rjbtisb	webapp.4	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker-worker	Running	Running about a min
v477ksz07763	webapp.5	localhost:5000/rre-aisi2223-webcounter:latest	rre-aisi2223-docker	Running	Running about a min

```
vagrant@rre-aisi2223-docker:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
f121a97b0608	localhost:5000/rre-aisi2223-webcounter:latest	"flask run"	About a minute ago	Up About a minute	5000/tcp
54a3d2e5bf2b	localhost:5000/rre-aisi2223-webcounter:latest	"flask run"	About a minute ago	Up About a minute	5000/tcp
Seeb69277e71	registry:latest	"/entrypoint.sh /etc..."	About an hour ago	Up About an hour	5000/tcp

```
vagrant@rre-aisi2223-docker:~$
```

- Comprueba el acceso desde tu **host**: <http://localhost:8080>
  - ¿Por qué no funciona?
- 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 --pretty*)
- Elimina ambos servicios
  - *docker service rm redis webapp*



# Ejercicio 5

37

- Crea una red *multihost* usando el driver *overlay* y a continuación lista las redes
  - **Debes nombrar tu red siguiendo el formato: xxx-aisi2223-network**
  - Como estamos en un entorno distribuido (un clúster Swarm) debemos crear una red *multihost* (driver *overlay*) a la que se conecten nuestros servicios

```
vagrant@rre-aisi2223-docker:~$ docker network create --driver overlay rre-aisi2223-network
b07jdhk96hu2e48dm5dcutxsj
vagrant@rre-aisi2223-docker:~$ docker network ls
NETWORK ID          NAME                DRIVER             SCOPE
095834760b37        bridge              bridge             local
1122d56fd44b        docker_gwbridge     bridge             local
26665acf4d55        host                host               local
wjsutnakqtuk        ingress             overlay            swarm
086e0e004b3e        none                null               local
b07jdhk96hu2        rre-aisi2223-network overlay            swarm
vagrant@rre-aisi2223-docker:~$
```



Se crea únicamente en la VM manager. Veremos que Docker Swarm se encarga 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 4 funcionó sin necesidad de crear ninguna red adicional?
  - Puedes comprobar también que la red *multihost* existe en la VM *worker*



# Ejercicio 5

38



- Inspecciona el servicio webapp (*docker service inspect --pretty*) y comprueba el acceso desde la **VM** usando el comando **curl**

```
vagrant@rre-aisi2223-docker:~$ docker service inspect --pretty webapp
```

```
ID:          6534flt7vw3exrci9itbm7iw1
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/rre-aisi2223-webcounter:latest@sha256:7
  Init:         false
Mounts:
  Target:       /src
  Source:       /vagrant/ej2/src
  ReadOnly:     false
  Type:         bind
Resources:
  Networks: rre-aisi2223-network
  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@rre-aisi2223-docker:~$ curl localhost
GEI AISI 2022/2023: my new counter for Roberto Rey Expósito (6 times)
vagrant@rre-aisi2223-docker:~$
```



# Ejercicio 5

39

- **Conéctate ahora a la VM worker y ejecuta estos comandos:**

- `docker ps`
- `curl xxx-aisi2223-docker`
- `curl xxx-aisi2223-docker-worker`
- `curl localhost`



¿Funciona desde la VM worker el acceso cuando se usa *localhost*?  
¿Qué deduces de estas pruebas?

- **Para terminar la práctica, elimina los servicios (incluido el Docker Registry)**

- `docker service rm redis webapp registry`