Despliegue de la app Guestboock

25/01/26
CURSO:2ºDAW

Despliegue
Celia Caravaca Vega

índice

Parte 1: Despliegue básico de Guestbook

Tarea 1.1: Preparación del entorno

1. **Crea una red Docker personalizada llamada red_guestbook que permita la comunicación entre los contenedores.**

```
docker network create red_guestbook .
```

```
PS C:\Users\rando> docker network create red_guestbook
d8c854321a402418f1722a0fd85aff51ae3c767188cdd42861d135dfeffddff3
PS C:\Users\rando> docker network ls
NETWORK ID     NAME            DRIVER    SCOPE
d75fdec3586a   bridge          bridge    local
31c629633a1c   host            host      local
74097351060e   none            null      local
d8c854321a40   red_guestbook   bridge    local
PS C:\Users\rando>
```

2. **Investiga en Docker Hub la documentación de la imagen redis para comprender:**
   - **Qué puerto utiliza por defecto.**
     6379/tcp.
   - **Cómo habilitar la persistencia de datos (modo append-only).**
     –appendonly yes
   - **Qué directorio utiliza para almacenar los datos.**
     /data

Tarea 1.2: Despliegue del servicio de base de datos

1. **Crea un contenedor con la base de datos Redis con las siguientes características:**
   - **Nombre del contenedor: redis**
   - **Volumen montado desde /opt/redis_data del host (crea este directorio previamente)**

```
PS C:\Users\rando> mkdir -p /opt/redis_data


    Directorio: C:\opt


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        25/01/2026     13:47                redis_data
```

   - **Ejecutando en modo daemon (segundo plano)**
   - **Conectado a la red red_guestbook**
   - **Configurado para persistencia de datos en el directorio /data del contenedor**

```
docker run -d \
  --name redis \
  --network red_guestbook \
  -v /opt/redis_data:/data \
  redis:latest \
  redis-server --appendonly yes
```

```
PS C:\Users\rando> docker run -d --name redis --network red_guestbook -v /opt/redis_data:/data redis:latest redis-server --appendonly yes
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
9c6dc2f051d0: Pull complete
202d17997ce3: Pull complete
87cb049a6d7d: Pull complete
4e5d87291b59: Pull complete
4f4fb700ef54: Pull complete
e5a35462b9a7: Pull complete
b308a2348d28: Pull complete
92752f2200e8: Download complete
222c7cf21fcb: Download complete
Digest: sha256:73dad4271642c5966db88db7a7585fae7cf10b685d1e48006f31e0294c29fdd7
Status: Downloaded newer image for redis:latest
7f1cdc85d8915256dbe8eefc0e94e38da5f5e5c9c7c3edb80620be518511018a
PS C:\Users\rando>
```

2. Verifica que el contenedor esté en ejecución correctamente.

ps -a

```
PS C:\Users\rando> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS      NAMES
7f1cdc85d891   redis:latest   "docker-entrypoint.s…"   37 seconds ago   Up 33 seconds   6379/tcp   redis
```

logs

```
PS C:\Users\rando> docker logs redis
Starting Redis Server
1:C 25 Jan 2026 12:52:15.968 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 25 Jan 2026 12:52:15.968 * Redis version=8.4.0, bits=64, commit=00000000,
1:C 25 Jan 2026 12:52:15.968 * Configuration loaded
1:M 25 Jan 2026 12:52:15.969 * monotonic clock: POSIX clock_gettime
1:M 25 Jan 2026 12:52:15.972 * Running mode=standalone, port=6379.
1:M 25 Jan 2026 12:52:15.973 * <bf> RedisBloom version 8.4.0 (Git=unknown)
1:M 25 Jan 2026 12:52:15.973 * <bf> Registering configuration options: [
1:M 25 Jan 2026 12:52:15.973 * <bf>      { bf-error-rate       :      0.01 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { bf-initial-size     :       100 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { bf-expansion-factor :         2 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { cf-bucket-size      :         2 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { cf-initial-size     :      1024 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { cf-max-iterations   :        20 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { cf-expansion-factor :         1 }
1:M 25 Jan 2026 12:52:15.973 * <bf>      { cf-max-expansions   :        32 }
1:M 25 Jan 2026 12:52:15.973 * <bf> ]
1:M 25 Jan 2026 12:52:15.974 * Module 'bf' loaded from /usr/local/lib/redis/m
1:M 25 Jan 2026 12:52:15.979 * <search> Redis version found by RedisSearch :
1:M 25 Jan 2026 12:52:15.980 * <search> RediSearch version 8.4.2 (Git=9e2b676
1:M 25 Jan 2026 12:52:15.980 * <search> Low level api version 1 initialized s
1:M 25 Jan 2026 12:52:15.980 * <search> gc: ON, prefix min length: 2, min wor
olicy: return, cursor read size: 1000, cursor max idle (ms): 300000, max doct
1:M 25 Jan 2026 12:52:15.980 * <search> Initialized thread pools!
1:M 25 Jan 2026 12:52:15.980 * <search> Disabled workers threadpool of size 0
1:M 25 Jan 2026 12:52:15.980 * <search> Subscribe to config changes
1:M 25 Jan 2026 12:52:15.980 * <search> Subscribe to cluster slot migration e
1:M 25 Jan 2026 12:52:15.980 * <search> Enabled role change notification
1:M 25 Jan 2026 12:52:15.981 * <search> Cluster configuration: AUTO partition
1:M 25 Jan 2026 12:52:15.981 * <search> Register write commands
1:M 25 Jan 2026 12:52:15.981 * Module 'search' loaded from /usr/local/lib/red
1:M 25 Jan 2026 12:52:15.982 * <timeseries> RedisTimeSeries version 80400, gi
1:M 25 Jan 2026 12:52:15.982 * <timeseries> Redis version found by RedisTimeS
1:M 25 Jan 2026 12:52:15.986 * Ready to accept connections tcp
```

Probar volumen:

Entrar dentro del cliente para hablar con la base de datos de redis desde la terminal

docker exec -it redis redis-cli

```
PS C:\Users\rando> docker exec -it  redis redis-cli
127.0.0.1:6379> SET usuario "Celia"
OK
127.0.0.1:6379> GET usuario
"Celia"
127.0.0.1:6379> SET saludo "Hola de nuevo"
OK
127.0.0.1:6379> GET saludo
"Hola de nuevo"
127.0.0.1:6379>
```

Ahora lo reiniciamos (Funciona)

```
PS C:\Users\rando> docker stop redis
redis
PS C:\Users\rando> docker start redis
redis
PS C:\Users\rando> docker exec -it  redis redis-cli
127.0.0.1:6379> GET usuario
"Celia"
127.0.0.1:6379> GET saludo
"Hola de nuevo"
127.0.0.1:6379>
```

Tarea 1.3: Despliegue del servicio web

1. **Despliega la aplicación Guestbook con las siguientes características:**
   - **Nombre del contenedor: guestbook**
   - **Conectado a la red red_guestbook**
   - **Puerto 80 del host mapeado al puerto 5000 del contenedor**
   - **Ejecutando en modo daemon**

   Primero voy a descargar la imagen tipica de goestbook que se conecta a Redis.
   docker pull ibmcom/guestbook:v1

```
PS C:\Users\rando> docker pull ibmcom/guestbook:v1
v1: Pulling from ibmcom/guestbook
ee94e6efc695: Pull complete
a3ed95caeb02: Pull complete
4d9ec8896d5a: Pull complete
2dcdf06a49b8: Pull complete
71cd7fd8c569: Pull complete
300273678d06: Pull complete
Digest: sha256:8e99727133b5c3b40c05bb313dd4591774f0a23415e3d44eac0bc883051de75d
Status: Downloaded newer image for ibmcom/guestbook:v1
docker.io/ibmcom/guestbook:v1
PS C:\Users\rando>
```

   Ahora creo el contenedor
   docker run -d \
   --name guestbook \
   --network red_guestbook \
   -p 80:5000 \
   ibmcom/guestbook:v1

```
PS C:\Users\rando> docker run -d --name guestbook --network red_guestbook -p 80:5000 ibmcom/guestbook:v1
25ed59406a867c221d091524facd46bd78dfa2f25204ca3ef6b84be258bcb327
PS C:\Users\rando>
```

2. **Accede a la aplicación desde tu navegador web (http://localhost).**

Incidendia: El contenedor no se muestra.

Comprobar contenedor up:

STATUS	PORTS	NAMES
Up 30 minutes	0.0.0.0:80->5000/tcp, [::]:80->5000/tcp	guestbook
Up 2 hours	6379/tcp	redis

Los logs de guestbook:

[negroni] listening on :3000 **(Escucha por el puerto 3000)**

Solución:Mapearlo al puerto correcto

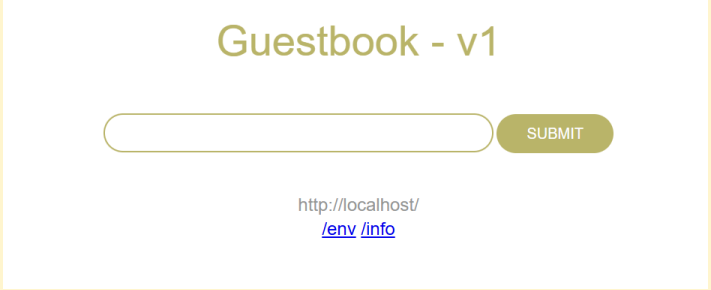
docker rm guestbook (borrar)

docker run -d --name guestbook --network red_guestbook -p 80:3000 ibmcom/guestbook:v1

Hacemos el ps.

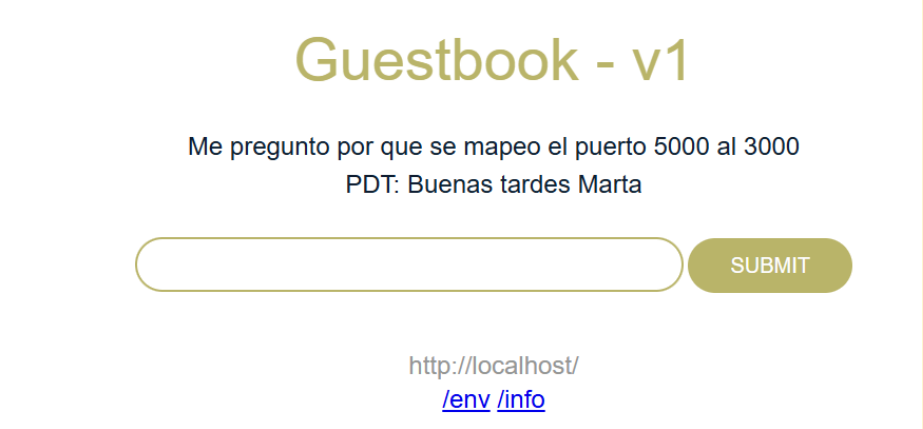
```
PS C:\Users\rando> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                                   NAMES
373a48452dba   ibmcom/guestbook:v1  "./guestbook"            25 seconds ago   Up 23 seconds   0.0.0.0:80->3000/tcp, [::]:80->3000/tcp guestbook
7f1cdc85d891   redis:latest         "docker-entrypoint.s…"   3 hours ago      Up 2 hours      6379/tcp                                redis
PS C:\Users\rando>
```

Y nos vamos a localhost.



(ˋ ω ´)✧

3. **Añade varios mensajes en el libro de visitas.**



Tarea 1.4: Verificación de la persistencia

1. Detén y elimina el contenedor redis.

```
PS C:\Users\rando> docker stop redis
redis
PS C:\Users\rando> docker container rm redis
redis
PS C:\Users\rando>
```

2. Verifica que los datos persisten en el directorio del host /opt/redis_data.

```
PS C:\Users\rando> docker exec -it redis redis-cli
127.0.0.1:6379> SET prueba "esto persiste"
OK
127.0.0.1:6379> GET usuario
(nil)
127.0.0.1:6379> SET usuario "Rando"
OK
127.0.0.1:6379> exit
```

```
PS C:\Users\rando> docker rm redis
redis
PS C:\Users\rando> docker volume ls
DRIVER    VOLUME NAME
local     5f5fb06c112cb61825147c14e32
local     82960a6f3982f386899c730f1a6
local     7752198ddad3c3284790e6fc373
local     c83b3332ba179f52da2762ab09d
local     e5c0e827b31d6f24f948be25034
local     redis_data
```

3. Vuelve a crear el contenedor redis con la misma configuración.

```
PS C:\Users\rando> docker run -d --name redis --network red_guestbook -v redis_data:/data redis redis-server --appendonly yes
b3e783e27da66ee3106dd5a1b1ba85ec3579c2b84d4e9f113260488e57b3a111
```

4. Accede nuevamente a la aplicación Guestbook y comprueba que los mensajes anteriores siguen presentes.

```
PS C:\Users\rando> docker exec -it redis redis-cli GET prueba
"esto persiste"
```

Los mensajes introducidos en la aplicación Guestbook se almacenan en Redis. Al eliminar y recrear el contenedor Redis utilizando el mismo volumen de datos, los mensajes siguen estando disponibles en la aplicación, lo que confirma la correcta persistencia de la información.

Parte 2: Configuración personalizada

Tarea 2.1: Cambio del nombre del contenedor de base de datos

La aplicación Guestbook utiliza una variable de entorno llamada REDIS_SERVER para saber a qué servidor de base de datos debe conectarse. Por defecto, esta variable tiene el valor redis.

1. **Elimina los contenedores anteriores (mantén la red y el volumen).**

```
PS C:\Users\rando> docker stop redis
redis
PS C:\Users\rando> docker ps
CONTAINER ID   IMAGE                  COMMAND         CREATED       STATUS       PORTS                                     NAMES
373a48452dba   ibmcom/guestbook:v1    "./guestbook"   2 hours ago   Up 2 hours   0.0.0.0:80->3000/tcp, [::]:80->3000/tcp   guestbook
PS C:\Users\rando> docker stop 37
37
PS C:\Users\rando> docker container rm 37
37
PS C:\Users\rando> docker container rm redis
redis
```

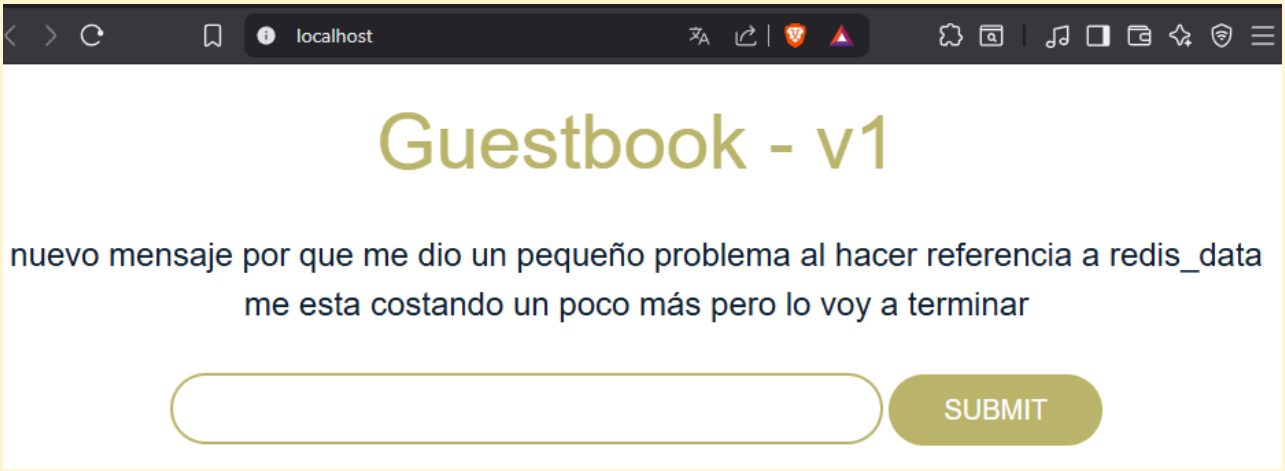
2. **Crea un nuevo contenedor de base de datos con el nombre bd_guestbook (en lugar de redis).**

```
PS C:\Users\rando> docker run -d --name  bd_guestbook --network red_guestbook -v redis_data/data redis redis-server --appendonly yes
5a44bda54a52bd6cf9f8b472cdba175e05667c841193c49fa43d77473719d488
PS C:\Users\rando>
```

3. **Despliega el contenedor de la aplicación Guestbook configurando la variable de entorno REDIS_SERVER para que apunte al nuevo nombre del contenedor de base de datos.**
   Durante las primeras pruebas, los mensajes no persistieron debido a una configuración incorrecta del volumen de Redis. Posteriormente, al corregir el montaje del volumen y recrear el contenedor Redis utilizando el mismo volumen de datos, la persistencia funcionó correctamente

```
PS C:\Users\rando> docker rm -f bd_guestbook
bd_guestbook
PS C:\Users\rando> docker run -d --name bd_guestbook --network red_guestbook -v redis_data:/data redis redis-server --appendonly yes
bef9846bc0043052651bc7ea4fb5d1214713305b9ff1eaf6e5261c0e6fadb7ae
```

4. **Verifica que la aplicación funciona correctamente con esta nueva configuración.**



Parte 3: Análisis y documentación

Tarea 3.1: Análisis de la arquitectura

Responde a las siguientes preguntas en tu documentación:

1. **Redes Docker:**
    - **¿Por qué es necesario crear una red personalizada en lugar de usar la red por defecto de Docker?**
      Para poder localizarla y asignarla de manera controlada.
      Pero la razon principal es el DNS interno para que los contenedores se encuentren por sus nombres.
    - **¿Cómo se comunican los contenedores dentro de la red red_guestbook?**
      Se comunican a través de sus nombres de contenedor por el DNS.
    - **¿Qué ventaja proporciona la resolución DNS automática de Docker?**
      Fácil identificación de los nombres de los docker(abstracción).
2. **Volúmenes:**
    - **¿Por qué es importante usar volúmenes para la base de datos?**
      Porque almacenan y organizan los datos de nuestros contenedores para que en caso de actualización o error podamos recuperar la información.
    - **¿Qué pasaría si no usáramos un volumen para Redis?**
      Pues que en caso de pararlo borrarlo o un error se perderán todos los datos.
    - **¿Qué diferencias existen entre un volumen y un bind mount?**
      El volumen gestiona el docker totalmente en una parte privada del disco.
      Con bind mount eliges tú donde se guardan.
3. **Variables de entorno:**
    - **¿Qué función cumple la variable de entorno REDIS_SERVER?**
      Sirve para indicar a la aplicación donde está la base de datos para que el contenedor sepa a quién llamar.
    - **¿Cómo se pasa una variable de entorno a un contenedor en Docker?**
      Con la etiqueta -e la variable = valor
4. **Puertos:**
    - **¿Por qué no es necesario exponer el puerto de Redis al host?**
      Por seguridad, para que nadie pueda tocarlo desde afuera.
    - **¿En qué casos sería necesario exponer el puerto de la base de datos?**
      Principalmente para mantenimiento o desarrollo.