

Trabajo Práctico 3 - Arquitectura de Sistemas Distribuidos

Desarrollo:

1- Sistema distribuido simple

```

pedrofernandez — -zsh — 80x24
Last login: Thu Aug 24 12:37:44 on ttys000
pedrofernandez@MacBook-Air-de-Pedro ~ % docker network create -d bridge mybridge

53f482afca6b4686c0c32d625a5d3c3390a1dc917cce75ae5db5ffefa8fdee2d
pedrofernandez@MacBook-Air-de-Pedro ~ %

[pedrofernandez@MacBook-Air-de-Pedro ~ % docker run -d --net mybridge --name db ]
redis:alpine
Unable to find image 'redis:alpine' locally
alpine: Pulling from library/redis
9fda8d8052c6: Pull complete
8808541ccd68: Pull complete
fe6a7a1ce6c8: Pull complete
9dd56d8e1ce8: Pull complete
85f432ac2c37: Pull complete
c251fd962134: Pull complete
Digest: sha256:fd5de2340bc46cbc2241975ab027797c350dec6fd86349e3ac384e3a41be6fee
Status: Downloaded newer image for redis:alpine
f7d599301ae569ede74d98c59974637177f5b17b9324f8e3e3351ead312ffd18
pedrofernandez@MacBook-Air-de-Pedro ~ %

[pedrofernandez@MacBook-Air-de-Pedro ~ % docker run -d --net mybridge -e REDIS_
HOST=db -e REDIS_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest
Unable to find image 'alexisfr/flask-app:latest' locally
latest: Pulling from alexisfr/flask-app
f49cf87b52c1: Pull complete
7b491c575b06: Pull complete
b313b08bab3b: Pull complete
51d6678c3f0e: Pull complete
09f35bd58db2: Pull complete
1bda3d37eead: Pull complete
9f47966d4de2: Pull complete
9fd775bfe531: Pull complete
2446eec18066: Pull complete
b98b851b2dad: Pull complete
e119cb75d84f: Pull complete
Digest: sha256:250221bea53e4e8f99a7ce79023c978ba0df69bdfe620401756da46e34b7c80b
Status: Downloaded newer image for alexisfr/flask-app:latest
WARNING: The requested image's platform (linux/amd64) does not match the detecte
d host platform (linux/arm64/v8) and no specific platform was requested
aa1506721e5be7a6f3a5aacefcf81213f6aece7e14c4eaab46bb9499ca294531
docker: Error response from daemon: Ports are not available: exposing port TCP 0
.0.0.0:5000 -> 0.0.0.0:0: listen tcp 0.0.0.0:5000: bind: address already in use.
pedrofernandez@MacBook-Air-de-Pedro ~ %
```

- Verificar el estado de los contenedores y redes en Docker, describir:
 - ¿Cuáles puertos están abiertos?

El contenedor de la aplicación web está escuchando en el puerto 5000.

El contenedor de la base de datos Redis está escuchando en su puerto por defecto, 6379.

- Mostrar detalles de la red `mybridge` con Docker.

```
pedrofernandez@MacBook-Air-de-Pedro ~ % docker network inspect mybridge
[
  {
    "Name": "mybridge",
    "Id": "53f482afca6b4686c0c32d625a5d3c3390a1dc917cce75ae5db5ffefa8fdee2d",
    "Created": "2023-08-24T16:02:10.033912334Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "f7d599301ae569ede74d98c59974637177f5b17b9324f8e3e3351ead312ffd18": {
        "Name": "db",
        "EndpointID": "c7913dbbe05b14bba06f9cd308d52e30e94f06ec247b9e256d0e81557ca5a682",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
pedrofernandez@MacBook-Air-de-Pedro ~ %
```

- ¿Qué comandos utilizó?

```
docker network inspect mybridge
```

2- Análisis del sistema

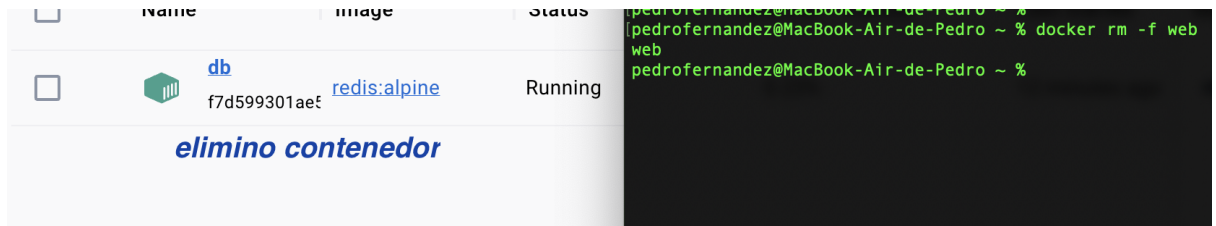
- Explicar cómo funciona el sistema:

Este sistema consta de tres componentes: una aplicación web escrita en Python usando Flask, una base de datos Redis y una interfaz para el usuario. La aplicación web incrementa un contador almacenado en Redis cada vez que se accede a través del navegador.

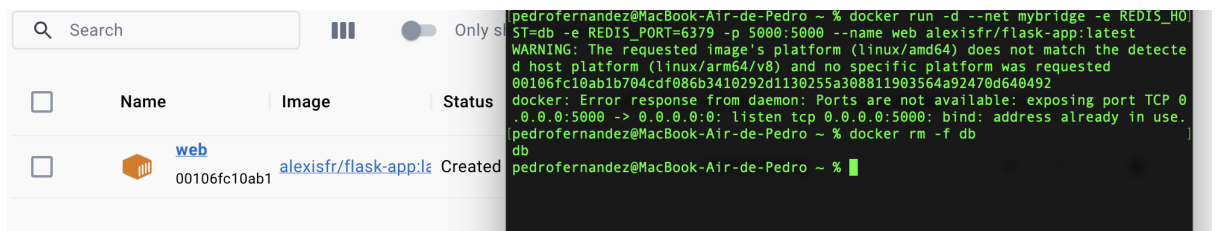
- ¿Para qué sirven y porque están los parámetros `-e` en el segundo Docker run del ejercicio 1?

Los parámetros `-e` en el segundo docker run del ejercicio 1 se utilizan para pasar variables de entorno a los contenedores.

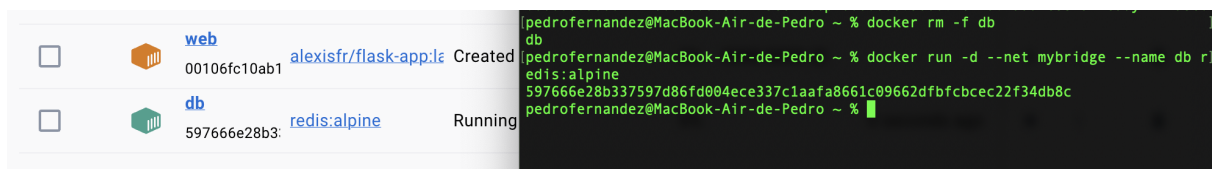
- ¿Qué pasa si ejecuta `docker rm -f web` y vuelve a correr `docker run -d --net mybridge -e REDIS_HOST=db -e REDIS_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest` ?



- ¿Qué ocurre en la página web cuando borro el contenedor de Redis con `docker rm -f db`?



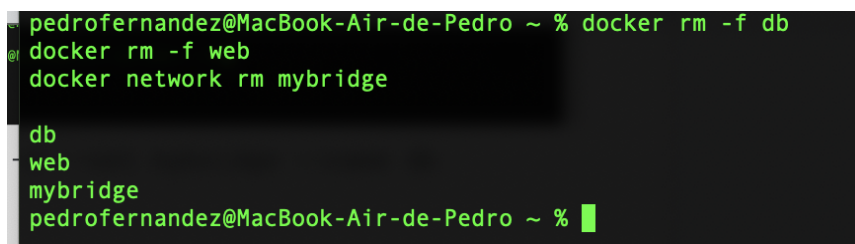
- Y si lo levanto nuevamente con `docker run -d --net mybridge --name db redis:alpine` ?



- ¿Qué considera usted que haría falta para no perder la cuenta de las visitas?

Para no perder la cuenta de las visitas, necesitarías una forma de persistir los datos del contador en Redis o en algún otro sistema de almacenamiento persistente, como una base de datos.

- Para eliminar los elementos creados corremos:



3- Utilizando docker compose

- Crear el siguiente archivo `docker-compose.yaml` en un directorio de trabajo:

```
[pedrofernandez@MacBook-Air-de-Pedro Desktop % cat > docker-compose.yaml
version: '3.6'
services:
  app:
    image: alexisfr/flask-app:latest
    depends_on:
      - db
    environment:
      - REDIS_HOST=db
      - REDIS_PORT=6379
    ports:
      - "5000:5000"
  db:
    image: redis:alpine
    volumes:
      - redis_data:/data
volumes:
  redis_data:
^C
pedrofernandez@MacBook-Air-de-Pedro Desktop %
```

- Ejecutar `docker-compose up -d`

```
pedrofernandez@MacBook-Air-de-Pedro Desktop % docker-compose up -d
[*] Running 4/1
[+] Network desktop_default Created0.0s
[+] Volume "desktop_redis_data" Created0.0s
[+] Container desktop-db-1 Starting0.0s [+] Running 4/5
[+] Network desktop_default Created0.0s
[+] Volume "desktop_redis_data" Created0.0s
[+] Container desktop-db-1 Starting0.0s [+] Running 4/5
[+] Network desktop_default Created0.0s
[+] Volume "desktop_redis_data" Created0.0s
[+] Container desktop-db-1 Starting0.0s [+] Running 4/5
[+] Network desktop_default Created0.0s
[+] Volume "desktop_redis_data" Created0.0s
[+] Container desktop-db-1 Starting0.0s
[+] Container desktop-app-1 Starting0.3s
```

- Ejecutar `docker ps`, `docker network ls` y `docker volume ls`

```
[pedrofernandez@MacBook-Air-de-Pedro ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS         NAMES
4fc6e45f0883   redis:alpine   "docker-entrypoint.s..." 2 minutes ago  Up 2 minu
tes           desktop-db-1
[pedrofernandez@MacBook-Air-de-Pedro ~ % docker network ls
NETWORK ID     NAME          DRIVER   SCOPE
0d02b3f32e94   bridge       bridge   local
9fcc6522e9d7   desktop_default bridge   local
a8bc27d6efb8   host         host     local
0ad1d0101a8f   none        null     local
[pedrofernandez@MacBook-Air-de-Pedro ~ % docker volume ls
DRIVER   VOLUME NAME
local    53735c670861a38e484776c6c0be565afaa160a073914fcedd937e6d9d4a0210
local    b115edaa9044824a8e3ac5d4f4a7d457a066abd50dbfbf899bd89105b1359c33
local    desktop_redis_data
pedrofernandez@MacBook-Air-de-Pedro ~ %
```

- ¿Qué hizo Docker Compose por nosotros? Explicar con detalle.

Docker Compose simplifica la administración de sistemas distribuidos al permitirte definir y ejecutar aplicaciones multicontenedor de manera declarativa. En el texto:

Se definen dos servicios: app y db.

El servicio app utiliza la imagen alexisfr/flask-app:latest y depende del servicio db.

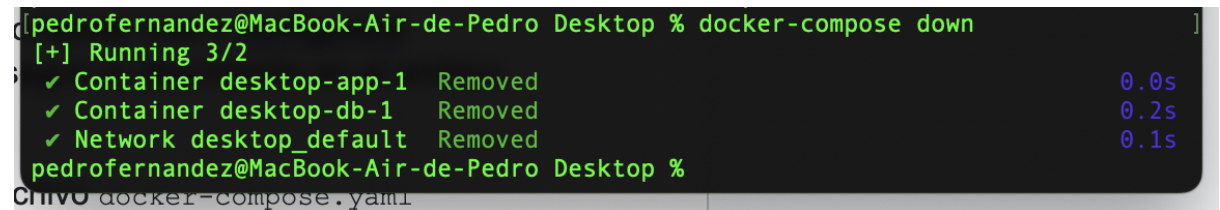
Se establecen variables de entorno para configurar la conexión a Redis.

Se mapea el puerto 5000 del contenedor al puerto 5000 del host para acceder a la aplicación web.

El servicio db utiliza la imagen Redis:alpine y crea un volumen llamado redis_data.

Al ejecutar `docker-compose up -d`, Docker Compose crea y configurará automáticamente los contenedores y la red según la descripción en el archivo YAML.

- Desde el directorio donde se encuentra el archivo `docker-compose.yaml` ejecutar:

A terminal window with a dark background and green text. The prompt is 'pedrofernandez@MacBook-Air-de-Pedro Desktop %'. The command 'docker-compose down' has been executed. The output shows a progress bar '[+] Running 3/2' followed by three lines: '✓ Container desktop-app-1 Removed 0.0s', '✓ Container desktop-db-1 Removed 0.2s', and '✓ Network desktop_default Removed 0.1s'. The prompt returns to 'pedrofernandez@MacBook-Air-de-Pedro Desktop %'. Below the terminal window, the text 'Archivo docker-compose.yaml' is visible.

```
pedrofernandez@MacBook-Air-de-Pedro Desktop % docker-compose down
[+] Running 3/2
✓ Container desktop-app-1 Removed 0.0s
✓ Container desktop-db-1 Removed 0.2s
✓ Network desktop_default Removed 0.1s
pedrofernandez@MacBook-Air-de-Pedro Desktop %
```

4- Aumentando la complejidad, análisis de otro sistema distribuido.

Pasos:

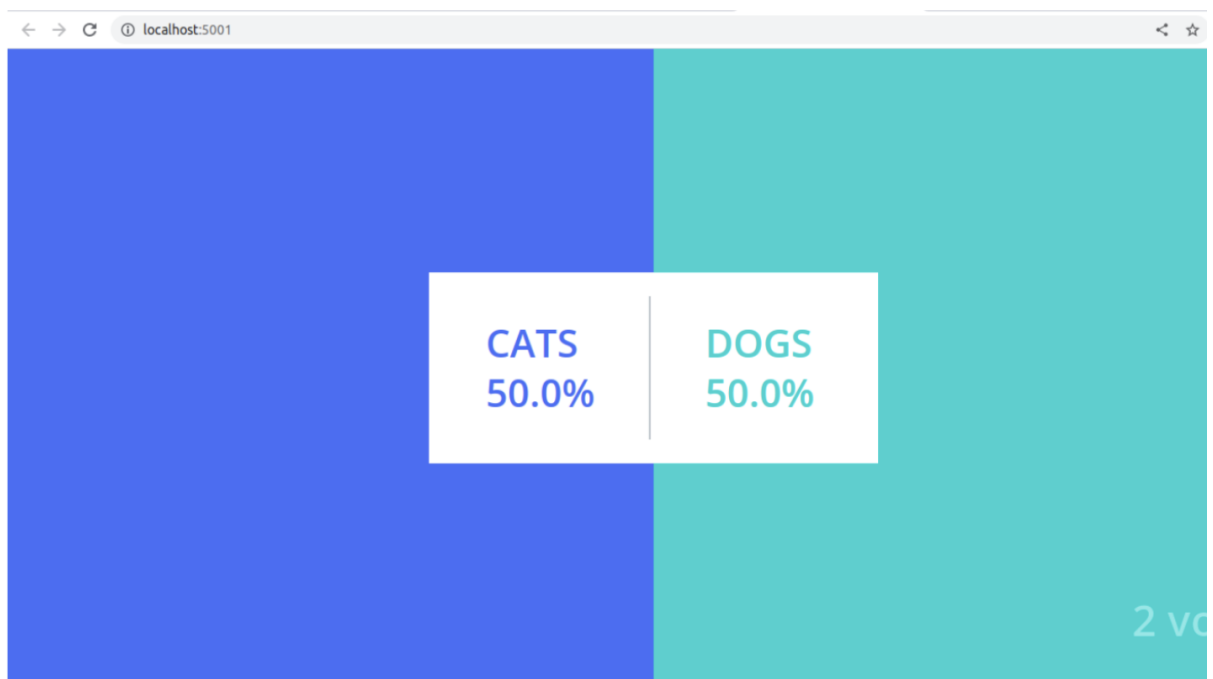
- Clonar el repositorio <https://github.com/dockersamples/example-voting-app>
- Abrir una línea de comandos y ejecutar

```

pedrofernandez@MacBook-Air-de-Pedro 03-arquitectura-sistemas-distribuidos % cd example-voting-app
pedrofernandez@MacBook-Air-de-Pedro example-voting-app % docker-compose -f docker-compose.yml up -d
[+] Running 9/9
  ✓ db 8 layers [#####] 0B/0B Pulled 81.7s
  ✓ 9fda8d8052c6 Already exists 0.0s
  ✓ b0d9bb38da5c Pull complete 0.9s
  ✓ a99f2e61e525 Pull complete 0.9s
  ✓ 675a64a52cf3 Pull complete 78.4s
  ✓ 0abcb1cdedc9 Pull complete 78.4s
  ✓ a86e63e9f268 Pull complete 78.4s
  ✓ dd648e2387c1 Pull complete 78.4s
  ✓ 248af3a055f3 Pull complete 78.5s
[+] Building 411.9s (30/38)
=> extracting sha256:4ee097f9a36616fddb52e45aba72142c4bc6f2e594f0a746e406acfd4f02ff51
=> extracting sha256:493e98a6d531d3e898241b29e0b8e182289a651c4f9f7d67bb1f50d9811a1b43
=> extracting sha256:08cedc5a91a9448070931d433f0965c0ee496f0900a2a7b0551a3590801174e2
=> extracting sha256:a1a4455ca20037ba3c8625bd6a88df6acf71c6b15c31f9303bc5e7cc3b011b00
=> extracting sha256:177e3dd16ed14f12cb1c710d5dc936b8c5609e1278aca102af3f5fc2c1877884
=> [vote internal] load build context
=> transferring context: 6.11kB
=> [worker build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:926b9337622ccc594ed051d3559f1c4544686c9fd5f4656f09d1fe37d9a5ace2
=> resolve mcr.microsoft.com/dotnet/sdk:7.0@sha256:926b9337622ccc594ed051d3559f1c4544686c9fd5f4656f09d1fe37d9a5ace2
=> sha256:ed434d8d0c6232b0d665022308bd7d5eb42f829871df394be44d9ad35dcf6a1d 7.23kB / 7.23kB
=> sha256:926b9337622ccc594ed051d3559f1c4544686c9fd5f4656f09d1fe37d9a5ace2 1.82kB / 1.82kB
=> sha256:1e6e5abd5185e937bd32667ce859aebad9841c9271b759140d91c78a5dd4503 2.01kB / 2.01kB
=> sha256:41f92d5a73b9bee296c7b4a3817b28098b22fb60112608b42bb03570ca296115 30.06MB / 30.06MB
=> sha256:5df5175ca8be3457791cd99e55bd8aa65365ae8ceeb35ef1600ff3b217fd08a 14.92MB / 14.92MB
=> sha256:a64d7b05f29270b0d104db8bac08feaf20c10160db51eab0f89ce701bbf5e2e3 30.72MB / 30.72MB
=> sha256:c9f99a6f21d7f96dfc15a577262d3d3e3ab3400734105f968aa708813b305554 156B / 156B
=> sha256:e904e1ee5fe4f9bd5502a2a7afd39a65b7f231fc90e9f754e030b2ddcd73ec00 9.81MB / 9.81MB
=> sha256:98a7bfe36790941bcc7824e3e9d070ad9bf6d1b4ae9ab396a0f67b6c791a7844 25.39MB / 25.39MB

```

- Una vez terminado acceder a <http://localhost:5000/> y <http://localhost:5001>
- Emitir un voto y ver el resultado en tiempo real.



- Explicar como está configurado el sistema, puertos, volúmenes, componentes involucrados, utilizar el Docker compose como guía.

Los componentes involucrados utilizando el archivo docker-compose.yml como guía son:

Servicio vote:

Depende del servicio redis y establece que se inicie una vez que el servicio redis esté en un estado saludable.

Servicio result:

Se construye utilizando el directorio ./result.

El entrypoint nodemon server.js se utiliza para iniciar la aplicación.

Depende del servicio db y establece que se inicie una vez que el servicio db esté en un estado saludable.

Servicio worker:

Depende de los servicios redis y db, y se inicia una vez que ambos servicios estén en un estado saludable.

Servicio redis:

Utiliza la imagen de Redis alpine.

Servicio db:

Utiliza la imagen de PostgreSQL 15 alpine.

Establece variables de entorno para el usuario y contraseña de PostgreSQL.

Servicio seed:

Está asociado al perfil seed, que permite ejecutarlo solo cuando se especifica el perfil seed al usar el comando docker-compose.

Depende del servicio vote y se inicia una vez que el servicio vote esté en un estado saludable.

Volúmenes:

db-data es un volumen utilizado por el servicio db para persistir los datos de PostgreSQL.

Redes:

front-tier es una red a la que pertenecen los servicios vote, result y seed. Permite la comunicación entre estos servicios.

back-tier es una red a la que pertenecen los servicios vote, result, worker, redis y db. Permite la comunicación entre estos servicios.

5- Análisis detallado

- Exponer más puertos para ver la configuración de Redis, y las tablas de PostgreSQL con alguna IDE como dbeaver.
- Revisar el código de la aplicación Python `example-voting-app\vote\app.py` para ver como envía votos a Redis.
 1. Importaciones y configuración de variables.
 2. Creación de la aplicación Flask.
 3. Configuración del registro.
 4. Función `get_redis()`.
 5. Ruta principal ("/") y función `hello()`.
 6. Inicio de la aplicación.
- Revisar el código del worker `example-voting-app\worker\program.cs` para entender como procesa los datos.
 1. Método Main: Punto de entrada del programa y configuración inicial.
 2. Bucle de procesamiento: Procesamiento continuo de votos almacenados en Redis.
 3. Función `OpenDbConnection`: Apertura y gestión de la conexión a la base de datos PostgreSQL.
 4. Función `OpenRedisConnection`: Apertura y gestión de la conexión a Redis.
 5. Función `GetIp`: Obtención de la dirección IP de un host utilizando DNS.
 6. Función `UpdateVote`: Actualización de la base de datos PostgreSQL con los votos.
- Revisar el código de la aplicación que muestra los resultados `example-voting-app\result\server.js` para entender como muestra los valores.

El código en `server.js` es una aplicación Node.js que utiliza Express y Socket.IO para mostrar los resultados de una votación en tiempo real. La aplicación se conecta a una base de datos PostgreSQL y realiza consultas periódicas para recopilar los resultados de los votos. Estos resultados se emiten a través de Socket.IO a los clientes conectados, lo que permite que los resultados se actualicen automáticamente en la página web sin necesidad de recargarla. El servidor Express también se configura para permitir el acceso desde cualquier origen y para servir un archivo `index.html` que muestra los resultados. En resumen, el código crea una experiencia interactiva donde los usuarios pueden ver los resultados de la votación en tiempo real mientras ocurren.