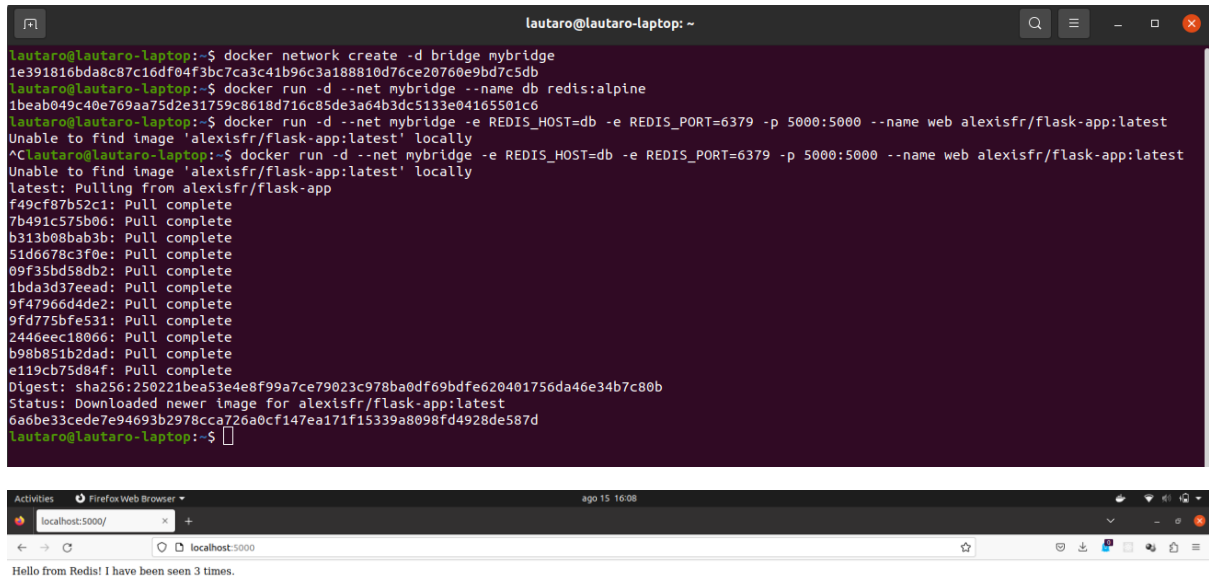


1)



The image shows a terminal window and a web browser. The terminal window, titled 'lautaro@lautaro-laptop: ~', displays the following commands and output:

```
lautaro@lautaro-laptop:~$ docker network create -d bridge mybridge
1e391816bda8c87c16df04f3bc7ca3c41b96c3a188810d76ce20760e9bd7c5db
lautaro@lautaro-laptop:~$ docker run -d --net mybridge --name db redis:alpine
1beab049c40e769aa75d2e31759c8618d716c85de3a64b3dc5133e04165501c6
lautaro@lautaro-laptop:~$ 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
^C
lautaro@lautaro-laptop:~$ docker run -d --net mybridge -e REDIS_HOST=db -e REDIS_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest
latest: Pulling from alexisfr/flask-app
f49cf87b52c1: Pull complete
7b491c575b06: Pull complete
b313b08bab3b: Pull complete
51d6678c3f0e: Pull complete
09f35bd58db2: Pull complete
1bda3d37ead: Pull complete
9f47966d4de2: Pull complete
9fd775bfe531: Pull complete
2446eec18066: Pull complete
b98b851b2dad: Pull complete
e119cb75d84f: Pull complete
Digest: sha256:250221bea53e4e8f99a7ce79023c978ba0df69bdf620401756da46e34b7c80b
Status: Downloaded newer image for alexisfr/flask-app:latest
6a6be33cede7e94693b2978cca726a0cf147ea171f15339a8098fd4928de587d
lautaro@lautaro-laptop:~$
```

The web browser, titled 'Firefox Web Browser', shows the address 'localhost:5000/' and the page content: 'Hello from Redis! I have been seen 3 times.'

- Verificar el estado de los contenedores y redes en Docker, describir:
 - ¿Cuáles puertos están abiertos?
 - Mostrar detalles de la red mybridge con Docker.
 - ¿Qué comandos utilizó?

Los puertos abiertos son el 5000 y el 6379.

```
ago 15 16:11
lautaro@lautaro-laptop: ~
lautaro@lautaro-laptop:~$ docker network inspect mybridge
[
  {
    "Name": "mybridge",
    "Id": "1e391818da8c87c16df0ef3bc7ca3c41b96c3a188810d76ce2070e9bd7c5db",
    "Created": "2023-08-15T18:52:41.715985862Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.28.0.0/16",
          "Gateway": "172.28.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "1beab849c40e769aa75d2e31759c8618d716c85de3a64b3dc5133e04165501c6": {
        "Name": "db",
        "EndpointID": "5416dd133d973b9e1189c126fd14bdf0b2903149c8d8633ca0ee34d881b3e7",
        "MacAddress": "02:42:ac:1c:08:02",
        "IPv4Address": "172.28.0.2/16",
        "IPv6Address": ""
      },
      "6a8be33cede7e94693b2978cca726a0cf147ea171f15339a8098fd4928de587d": {
        "Name": "web",
        "EndpointID": "41b000be77bd7a37ac33572679be0ffa4fabda4145871941c8febbdae54c95a",
        "MacAddress": "02:42:ac:1c:08:03",
        "IPv4Address": "172.28.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
```

Los comandos utilizados fueron:

- `docker ps -a`
- `docker network inspect mybridge`

2)

1. Explicar cómo funciona el sistema.
2. ¿Para qué sirven y porque están los parámetros `-e` en el segundo Docker run del ejercicio 1?
3. ¿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`?
4. ¿Qué ocurre en la página web cuando borro el contenedor de Redis con `docker rm -f db`?
5. Y si lo levanto nuevamente con `docker run -d --net mybridge --name db redis:alpine`?
6. ¿Qué considera usted que haría falta para no perder la cuenta de las visitas?

1. Este código representa una aplicación web simple escrita en Python utilizando el framework Flask y Redis como una base de datos en memoria. La aplicación web registra el número de veces que se accede a ella y muestra esa información en el navegador cuando se visita la ruta raíz ("/").

Aquí hay una explicación de cómo funciona el sistema:

Importando módulos:

- `os`: Módulo para interactuar con funcionalidades del sistema operativo, como acceder a variables de entorno.

- `Flask`: Framework web en Python que se utiliza para crear aplicaciones web.
- `Redis`: Cliente para Redis, una base de datos en memoria que se utiliza como almacén para mantener el recuento de las visitas a la aplicación.

Creación de la aplicación Flask y configuración:

- Se crea una instancia de la aplicación Flask llamada `app`.
- Se crea una instancia de Redis que se conecta al host y puerto proporcionados a través de variables de entorno (`REDIS_HOST` y `REDIS_PORT`).

Definición de una ruta y una función de manejo:

- Se define una ruta raíz ("/") utilizando el decorador `@app.route('/')`.
- La función `hello()` se ejecutará cuando alguien acceda a la ruta raíz.
- Dentro de esta función:
 - Incrementa el contador de visitas en Redis usando `redis.incr('hits')`. Redis almacena los datos en una estructura de clave-valor.
 - Recupera el valor actual del contador de visitas con `redis.get('hits')` y lo convierte a una cadena decodificada.
 - Retorna un mensaje de saludo junto con el número total de visitas.

Bloque de inicio de la aplicación:

- Se verifica si el script se está ejecutando directamente (`if __name__ == "__main__":`).
- Si es así, la aplicación Flask se ejecutará en modo de depuración (`debug=True`) en la dirección IP `0.0.0.0` y en el puerto especificado en la variable `bind_port`.

En resumen, esta aplicación web en Flask utiliza Redis para rastrear el número de visitas a la página y muestra esta información en el navegador cuando se accede a la ruta raíz. Cada vez que alguien visita la página, el contador de visitas en Redis se incrementa y se muestra el recuento actualizado en la respuesta. La aplicación se ejecutará en la dirección IP `0.0.0.0`, lo que significa que estará disponible para acceder desde cualquier dirección IP.

2. Los `-e` indican que se va a pasar una variable de entorno, estos sirven para que la app ponga dinámicamente los nombres de los puertos y la base de datos.
3. Al hacer esto vemos que el conteo de ingreso a la página continúa en donde se quedó.
4. Cuando borro el contenedor, la página web tira un error de conexión a la base de datos.
5. Al levantar nuevamente la base de datos, el conteo comienza nuevamente desde cero.
6. Necesitaría crear un volumen para que persistan los datos.

3)

1. Ejecutar `docker-compose up -d`
2. Acceder a la url `http://localhost:5000/`
3. Ejecutar `docker ps`, `docker network ls` y `docker volume ls`

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

1.

```
lautaro@lautaro-laptop: ~/Documents/Facultad/Ingenieria de Software III/Github/TP3$ ls
docker-compose.yaml  example-voting-app
lautaro@lautaro-laptop:~/Documents/Facultad/Ingenieria de Software III/Github/TP3$ docker compose up
[+] Building 0.0s (0/0)
[+] Running 4/4
✓ Network tp3_default      Created      0.0s
✓ Volume "tp3_redis_data" Created      0.0s
✓ Container tp3-db-1       Created      0.0s
✓ Container tp3-app-1      Created      0.0s
Attaching to tp3-app-1, tp3-db-1
tp3-db-1 | 1:C 15 Aug 2023 19:26:05.514 # o000o000o000o Redis is starting o000o000o000o
tp3-db-1 | 1:C 15 Aug 2023 19:26:05.514 # Redis version=7.0.12, bits=64, commit=00000000, modified=0,
tp3-db-1 | pid=1, just started
tp3-db-1 | 1:C 15 Aug 2023 19:26:05.514 # Warning: no config file specified, using the default config
tp3-db-1 | . In order to specify a config file use redis-server /path/to/redis.conf
tp3-db-1 | 1:M 15 Aug 2023 19:26:05.514 * monotonic clock: POSIX clock_gettime
tp3-db-1 | 1:M 15 Aug 2023 19:26:05.515 * Running mode=standalone, port=6379.
tp3-db-1 | 1:M 15 Aug 2023 19:26:05.515 # Server initialized
tp3-db-1 | 1:M 15 Aug 2023 19:26:05.515 * Ready to accept connections
tp3-app-1 | * Serving Flask app "app" (lazy loading)
tp3-app-1 | * Environment: production
tp3-app-1 | WARNING: Do not use the development server in a production environment.
tp3-app-1 | Use a production WSGI server instead.
tp3-app-1 | * Debug mode: on
tp3-app-1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
tp3-app-1 | * Restarting with stat
tp3-app-1 | * Debugger is active!
tp3-app-1 | * Debugger PIN: 155-482-719
```

2. Al ingresar a localhost:5000 ingresamos a la misma app a la cual ingresamos anteriormente.

3.

```
lautaro@lautaro-laptop:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
f61057c9470b   alexisfr/flask-app:latest          "python /app.py"        3 minutes ago Up About a minute 0.0.0.0:5000->5000/tcp   tp3-app-1
b2551f939107   redis:alpine                      "docker-entrypoint.s..." 3 minutes ago Up About a minute 6379/tcp            tp3-db-1

lautaro@lautaro-laptop:~$ docker network ls
NETWORK ID     NAME                                DRIVER    SCOPE
55df0586b485   bridge                             bridge    local
a57f3421b8b0   example-voting-app_back-tier       bridge    local
c64fe6a059f2   example-voting-app_front-tier       bridge    local
aded22bc1883   host                               host      local
ad0c814c3f00   none                               null      local
4b83fc7d2d1f   tp3_default                         bridge    local

lautaro@lautaro-laptop:~$ docker volume ls
DRIVER    VOLUME NAME
local     45fae38523f656c2c4cd051e330aadb3c6c7a88ca62ff699d4fc1f9a5a1ab075
local     9439cebab57201f7f50e18096726f0e108c2c34fe24f641a9cd6ddb9fdd4cf39
local     e3bcd716f6ce7808b7a4854b0e50be3ccbbb437fe5242f45f42230387b07b69
local     example-voting-app_db-data
local     tp3_redis_data
```

4.

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multi-contenedor en Docker de una manera más conveniente. En el caso del archivo `docker-compose.yaml` que proporcionaste, Docker Compose se encarga de gestionar y orquestar los contenedores necesarios para tu aplicación, así como de establecer las conexiones y

configuraciones entre ellos. Aquí está una explicación detallada de lo que Docker Compose hizo por nosotros en este caso:

Definición de servicios:

- La sección ``services`` define los servicios (contenedores) que componen tu aplicación. En este caso, defines dos servicios: ``app`` y ``db``.

Servicio ``app`` (Aplicación Flask):

- Utiliza la imagen ``alexisfr/flask-app:latest`` para crear el contenedor de la aplicación Flask.

- ``depends_on`` especifica que el servicio ``app`` depende del servicio ``db``. Esto significa que Docker Compose garantizará que el servicio de Redis (``db``) se inicie antes de iniciar el servicio de la aplicación (``app``).

- ``environment`` establece las variables de entorno ``REDIS_HOST`` y ``REDIS_PORT`` para la aplicación. Esto permite que la aplicación Flask sepa dónde encontrar la instancia de Redis.

- ``ports`` establece un mapeo de puertos para el servicio ``app``, lo que significa que el puerto 5000 del contenedor se mapeará al puerto 5000 del host (tu máquina local).

Servicio ``db`` (Redis):

- Utiliza la imagen ``redis:alpine`` para crear el contenedor de Redis.

- ``volumes`` define un volumen llamado ``redis_data`` y lo monta en el directorio ``/data`` dentro del contenedor de Redis. Esto permite persistir los datos de Redis incluso si el contenedor se detiene o se elimina.

Definición de volúmenes:

- La sección ``volumes`` define los volúmenes que se utilizarán en los contenedores. En este caso, solo se define un volumen llamado ``redis_data``.

En resumen, Docker Compose se encarga de varios aspectos clave para facilitar la ejecución de tu aplicación:

- Gestiona la creación y el inicio de los contenedores de la aplicación Flask y Redis.
- Establece la dependencia entre los servicios (``app`` depende de ``db``) para asegurarse de que se inicie en el orden correcto.
- Configura las variables de entorno necesarias para que la aplicación Flask se conecte a la instancia de Redis.
- Mapea el puerto 5000 del contenedor de la aplicación al puerto 5000 de tu máquina local, lo que te permite acceder a la aplicación desde tu navegador.
- Crea un volumen llamado ``redis_data`` y lo monta en el contenedor de Redis, lo que permite persistir los datos de Redis más allá del ciclo de vida del contenedor.

En esencia, Docker Compose simplifica la administración de múltiples contenedores y su interconexión, lo que hace que el proceso de desarrollo y despliegue sea mucho más manejable y reproducible.

4)

```
lautaro@lautaro-laptop: ~/Documents/Facultad/Ingenieria d...
lautaro@lautaro-laptop:~/Documents/Facultad/Ingenieria de Software III/Github/TP3/example-voting-app$ docker compose -f docker-compose.yml up -d
[+] Building 0.0s (0/0)
[+] Running 5/5
✓ Container example-voting-app-db-1      Healthy      5.8s
✓ Container example-voting-app-result-1   Started      6.2s
✓ Container example-voting-app-redis-1    Healthy      5.8s
✓ Container example-voting-app-worker-1   Started      6.1s
✓ Container example-voting-app-vote-1     Started      6.2s
lautaro@lautaro-laptop:~/Documents/Facultad/Ingenieria de Software III/Github/TP3/example-voting-app$
```

Este archivo de Docker Compose describe un conjunto de servicios interconectados que forman una aplicación compuesta por varios componentes. Cada servicio representa un componente diferente de la aplicación y se ejecuta en su propio contenedor. A continuación, se explica cada sección del archivo:

Servicio "vote" (votación):

- Se construye a partir de la carpeta `./vote`.
- Usa el comando `python app.py` para ejecutar la aplicación en lugar de usar Gunicorn para el desarrollo local.
- Depende del servicio "redis" y espera a que esté en un estado saludable antes de iniciar.
- Realiza un chequeo de salud usando `curl` para verificar si la aplicación está disponible.
- Comparte la carpeta `./vote` con el contenedor (volumen).
- Mapea el puerto `5000` del host al puerto `80` del contenedor.
- Se conecta a las redes `front-tier` y `back-tier`.

Servicio "result" (resultado):

- Se construye a partir de la carpeta `./result`.
- Utiliza `nodemon server.js` como punto de entrada en lugar de solo `node` para el desarrollo local.
- Depende de los servicios "db" y "redis" y espera a que ambos estén en un estado saludable antes de iniciar.
- Comparte la carpeta `./result` con el contenedor (volumen).
- Mapea los puertos `5001` y `5858` del host a los puertos `80` y `5858` del contenedor.
- Se conecta a las redes `front-tier` y `back-tier`.

Servicio "worker" (trabajador):

- Se construye a partir de la carpeta `./worker``.
- Depende de los servicios "redis" y "db" y espera a que ambos estén en un estado saludable antes de iniciar.
- Se conecta a la red `back-tier``.

Servicio "redis":

- Utiliza la imagen predefinida `redis:alpine``.
- Comparte la carpeta `./healthchecks`` con el contenedor (volumen).
- Realiza un chequeo de salud ejecutando el script `/healthchecks/redis.sh`` cada 5 segundos.
- Se conecta a la red `back-tier``.

Servicio "db" (base de datos PostgreSQL):

- Utiliza la imagen `postgres:15-alpine``.
- Configura las variables de entorno `POSTGRES_USER`` y `POSTGRES_PASSWORD``.
- Comparte la carpeta `./healthchecks`` con el contenedor (volumen).
- Realiza un chequeo de salud ejecutando el script `/healthchecks/postgres.sh`` cada 5 segundos.
- Se conecta a la red `back-tier``.

Servicio "seed" (semilla de la base de datos):

- Se construye a partir de la carpeta `./seed-data``.
- Se ejecuta solo cuando se especifica el perfil "seed" al utilizar `docker-compose --profile seed up -d``.
- Depende del servicio "vote" y espera a que esté en un estado saludable antes de iniciar.
- Se conecta a la red `front-tier``.

Definición de volúmenes:

- Define un volumen llamado `db-data`` que se utiliza para almacenar los datos de PostgreSQL y persistirlos por si el contenedor se elimina.

Definición de redes:

- Define dos redes: `front-tier`` y `back-tier``, que se utilizan para conectar los servicios de la aplicación.

En conjunto, este archivo de Docker Compose describe una aplicación distribuida con múltiples servicios que se ejecutan en contenedores independientes y se comunican entre sí a través de redes definidas. La configuración también incluye chequeos de salud para garantizar que los servicios estén funcionando correctamente antes de que la aplicación se considere en un estado saludable.