Desarrollo paso a paso de el trabajo practico numero 3, de ingenieria de software 3, referido a sistemas distribuidos, introduciendo un poco sobre como funciona docker network con cargas de trabajo que no sean de docker, bridge network driver y docker compose.

Luego de la introducción que nos da de los conceptos teóricos, procedemos al desarrollo practico del trabajo practico

Desarrollo:
1-Sistema distribuido simple

- Ejecuto el siguiente comando para crear la red en docker
  docker network create -d bridge mybridge.

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker network create -d bridge my
bridge
14c470e8b01dc3f9c12271b9f424648c63e60538aa4fa463a3cdca313e5cc96d
```

- Instanciar una base de datos Redis conectada a esa Red.
  docker run -d --net mybridge --name db redis:alpine

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$  docker run -d --net mybridge --na
me db redis:alpine
Unable to find image 'redis:alpine' locally
alpine: Pulling from library/redis
213ec9aee27d: Pull complete
c99be1b28c7f: Pull complete
8ff0bb7e55e3: Pull complete
6d80de393db7: Pull complete
8dbffc478db1: Pull complete
7402bc4c98a0: Pull complete
Digest: sha256:dc1b954f5a1db78e31b8870966294d2f93fa8a7fba5c1337a1ce4ec55f311bc3
Status: Downloaded newer image for redis:alpine
5036f32d3853cc63953315b578833c50d2b37829d3f53b2fc9f37e3c21706cae
```

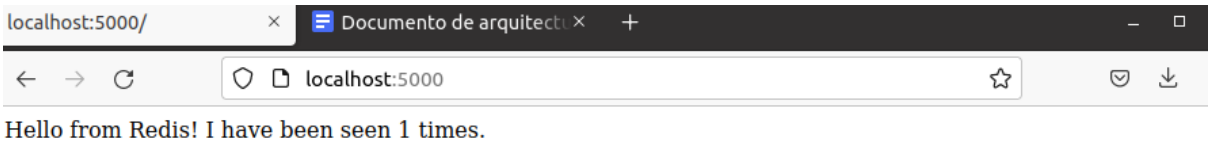
- **Levantar una web app, que utilice esta base de datos**

Generalmente los que se hace es dentro del entorno de codigo se pasa el contexto de la db y de la configuración se pasa la instancia de la base de datos que se este utilizando, en este caso desde el docker se instancio esta base de datos redis, donde realizamos esta instancia y luego con el siguiente comando la levantamos:

docker run -d --net mybridge -e REDIS_HOST=db -e REDIS_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker run -d --net mybridge -e RE
DIS_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
6f500da1ea0355467ea2a22696ea4240786d72950565335a71f5a6b12a1c975a
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$
```

- **Luego abrimos el navegador en local donde se levanto esa aplicacion, en el puerto especificado en este caso el 5000**



Abrir un navegador y acceder a la URL: http://localhost:5000/

- Averiguar que puertos tengo abiertos: utilizo el comando tambien visto en el practico anterior, docker ps, para ver los puertos abiertos

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                                       NAMES
6f500da1ea03   alexisfr/flask-app:latest   "python /app.py"         4 minutes ago   Up 4 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   web
5036f32d3853   redis:alpine             "docker-entrypoint.s…"   11 minutes ago  Up 11 minutes  6379/tcp                                    db
```

- Mostrar los detalles de la red mybridge con Docker:

Se ahondo en la documentación de docker, mas precisamente en docker network, donde buscamos el comando: **docker network inspect mybridge ,** el cual nos retorna la información sobre una o mas redes, por defectos el response de la misma es apartir de un JSON.

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker network inspect mybridge
[
    {
        "Name": "mybridge",
        "Id": "14c470e8b01dc3f9c12271b9f424648c63e60538aa4fa463a3cdca313e5cc96d",
        "Created": "2022-08-24T16:58:45.980398263-03:00",
        "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": {
            "5036f32d3853cc63953315b578833c50d2b37829d3f53b2fc9f37e3c21706cae": {
                "Name": "db",
                "EndpointID": "bae7890e4502d367b185eaf920ad7e0ee1785138637097d33218d3180e253a84",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "6f500da1ea0355467ea2a22696ea4240786d72950565335a71f5a6b12a1c975a": {
                "Name": "web",
                "EndpointID": "a2791e77353e99aedbfe03e12954c8c97f8ca0ecd2fda10be1abaec2300660f7",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

- Comandos utilizados: docker ps, docker network inspect mybridge

2-Analisis del sistema

```python
import os

from flask import Flask #importo la libreria de flask que me ayuda a crear apis de forma rapida
from redis import Redis #importo la libreria de redis que me ayuda a crear una conexion con redis


app = Flask(__name__) #creo una instancia de la libreria flask
redis = Redis(host=os.environ['REDIS_HOST'], port=os.environ['REDIS_PORT']) #se conecta con redis y especifica el host y puerto, que se encontraron en el contenedor cuando se creo.
bind_port = int(os.environ['BIND_PORT'])  #Especifica el puerto donde estara escuchando el servidor, el bind_port es el que esta en el contenedor que se creo.


@app.route('/') #creo una ruta para la raiz de la api, como si fuera el route de .NET, para el controlador
def hello(): #creo una funcionque imparcta en la ruta raiz
    redis.incr('hits') #incremento el valor de la variable hits en redis, osea cuando se carga o le pego a la url +=1 en hits
    total_hits = redis.get('hits').decode() #obtengo el valor de la variable hits en redis y lo decodifico para que sea un string
    return f'Hello from Redis! I have been seen {total_hits} times.' #retorno un mensaje con el valor de la variable hits


if __name__ == "__main__": #si el archivo es el principal
    app.run(host="0.0.0.0", debug=True, port=bind_port) #corro la aplicacion en el puerto indicado en el archivo de entorno o servidor localhost:5000
    #host: es el nombre del host o ip del servidor para escuchar, que por default es localhost
    #debug: es un parametro que permite ver errores en la aplicacion
    #port: puerto donde corre la aplicacion, que por defecto es el 5000
```

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

- Lo que hace ese parametro es establecer variables de entorno del contenedor

- ¿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 ?

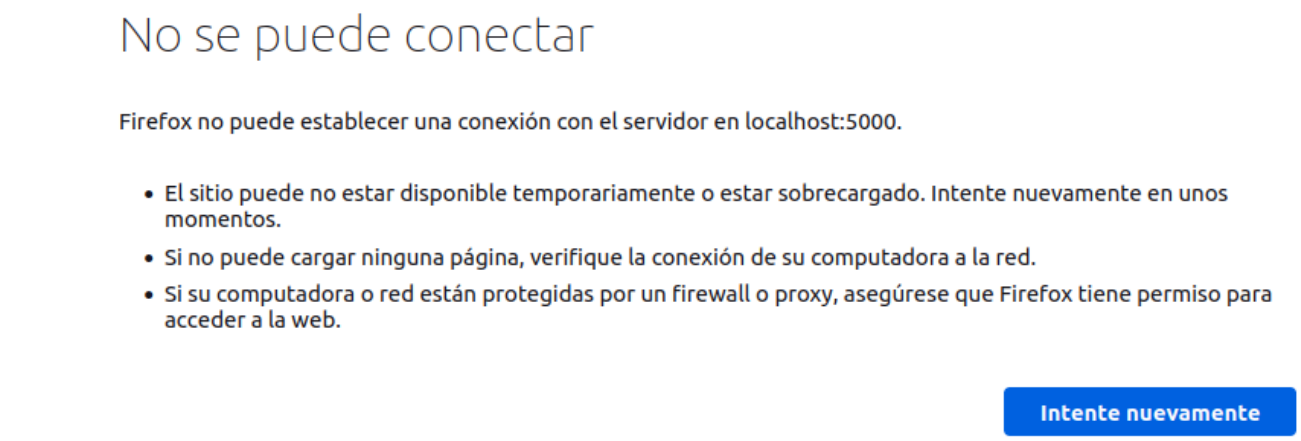
```
rm            Remove one or more containers
```

el rm -f lo que hace es eliminar el contenedor, el -f es force y elemina un contenedor por mas que el mismo se este ejecutando y despues no se podria conectar a la web, algo evidente que me lleva a pensar esto es que como uno de los comandos que se utilizaron al

principio, que instanciaba la base de datos y configuraba la conexion directamente al contenedor, pero si se levanta de nuevo el contenedor, la web tambien se levanta, pero al

levantarlo de vuelta, los datos no se perderían porque ya la conexión a la base de datos ya se hizo.

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker rm -f web
web
```




```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker run -d --net mybridge -e REDIS_HOST=db -
fff2f3c76e9acfd539d9724f43434f513780ba7716b498ba7e748301f48d28ed
```

Ahi lo vuelvo a levantar

Hello from Redis! I have been seen 2 times.

y el codigo por detras funciona perfecto, vemos que el servicio simple incremento el {**hit**}

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

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker rm -f db
db
```

Por lo que se ve, estoy tirando abajo la instancia a la base de datos.

redis.exceptions.ConnectionError

```
redis.exceptions.ConnectionError: Error -2 connecting to db:6379. Name or service not known.
```

Traceback (most recent call last)

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *177*, in _read_from_socket

```
raise socket.error(SERVER_CLOSED_CONNECTION_ERROR)
```

During handling of the above exception, another exception occurred:

File "/usr/local/lib/python3.6/site-packages/redis/client.py", line *668*, in execute_command

```
return self.parse_response(connection, command_name, **options)
```

File "/usr/local/lib/python3.6/site-packages/redis/client.py", line *680*, in parse_response

```
response = connection.read_response()
```

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *624*, in read_response

```
response = self._parser.read_response()
```

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *284*, in read_response

```
response = self._buffer.readline()
```

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *216*, in readline

```
self._read_from_socket()
```

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *191*, in _read_from_socket

```
(e.args,))
```

During handling of the above exception, another exception occurred:

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *484*, in connect

```
sock = self._connect()
```

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line *511*, in _connect

```
socket.SOCK_STREAM):
```

File "/usr/local/lib/python3.6/socket.py", line *745*, in getaddrinfo

```
for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
```

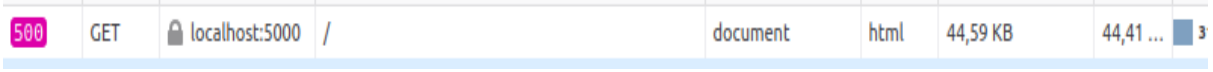
During handling of the above exception, another exception occurred:

File "/usr/local/lib/python3.6/site-packages/flask/app.py", line *2309*, in __call__

```
return self.wsgi_app(environ, start_response)
```

File "/usr/local/lib/python3.6/site-packages/flask/app.py", line *2295*, in wsgi_app

```
response = self.handle_exception(e)
```



veamos como me tira el internal server error, que me dice que no se conoce el connection string hacia la base de datos.

Lo vuelvo a levantar:

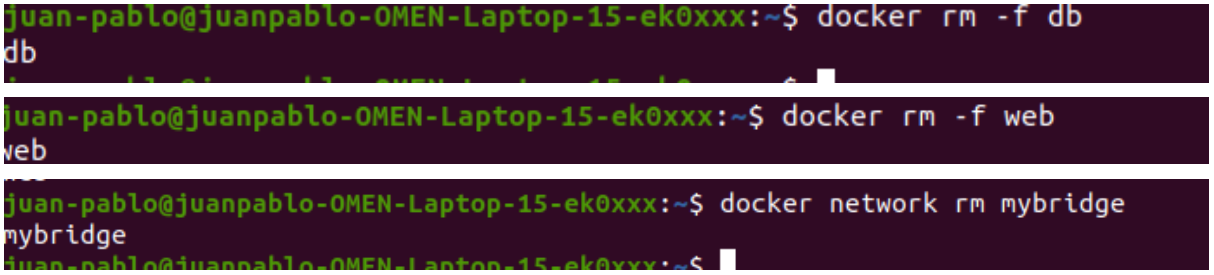
```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~$ docker run -d --net mybridge --name db redis:alpine
367508ef62e821c6873b6ac20eb17ef2f2268353fe79e1b23395263a3bc27a65
```

- Lo que ocurre cuando la volvi a levantar es lo siguiente

  Hello from Redis! I have been seen 1 times.

La base de datos se elimino, por eso cuando la volví a levantar las tablas se vaciaban.

- ¿Qué considera usted que haría falta para no perder la cuenta de las visitas? Buscar alguna forma de backup o respaldo con respecto al volumen impactado en la base de datos con el objetivo de que si se elimina el contenedor y la base de datos, de alguna forma si se levanta de nuevo se recuperen los datos de la persona que la levanto.



3-**Utilizando docker compose**

- Normalmente viene como parte de la solucion cuando se instaló Docker
- De ser necesario instalarlo hay que ejecutar:

  sudo pip install docker-compose

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

  Creo una carpeta en el escritorio, accedo por consola y hago un touch con el nombre y le pego lo siguiente

```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:
```

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

  después de estar un buen rato viendo porque me daba este error:

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker-compose u
p -d
ERROR: In file './docker-compose.yaml', volume must be a mapping, not a NoneType.
```

  sabia que era algo de la indentación, pero no lograba solucionarlo, el error era en la ultima redis_data, no debía estar al nivel de volume.

  resultado

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker-compose u
p -d
Creating network "dockercompose_default" with the default driver
Creating volume "dockercompose_redis_data" with default driver
Creating dockercompose_db_1 ... done
Creating dockercompose_app_1 ... done
```

  docker ps

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                                            NAMES
907e4365f2d4   alexisfr/flask-app:latest   "python /app.py"         2 minutes ago   Up 2 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   dockercompose_app_1
8b616eca0b1c   redis:alpine             "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   6379/tcp                                         dockercompose_db_1
```

  docker network ls

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker network ls
NETWORK ID     NAME                   DRIVER    SCOPE
844a28f98c75   bridge                 bridge    local
30124b2e32ec   dockercompose_default  bridge    local
fbd15f7a938f   host                   host      local
e5d095130ec4   none                   null      local
```

docker volume ls

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker volume ls
DRIVER    VOLUME NAME
local     8df053da82a4e3c1fa653311ccb9fea5b1bf73ff61002ebc04b4a22c4d7f5607
local     16f77eafd33da292ab526d1815a50a9dbf37fc4da7bd4e249f5e7e8083c1bf19
local     dockercompose_redis_data
local     fc68c4f59e97a21e251019c4dbe640064bec3fe7e72bc198b3405882c238d700
```

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

- Tomando la intro teorica de que hace docker compose, adjunto para repaso

Si nuestro sistem distribuido está compuesto de varios componentes corriendo con Docker, al momento de compilar, ejecutar y conectar los contenedores desde Dockerfiles separados definitivamente requiere mucho tiempo. Entonces, como solución a este problema, Docker Compose nos permite usar un archivo YAML para definir aplicaciones de múltiples contenedores. Es posible configurar tantos contenedores como queramos, cómo se deben construir y conectar, y dónde se deben almacenar los datos. Podemos ejecutar un solo comando para compilar, ejecutar y configurar todos los contenedores cuando el archivo YAML esté completo

Volvio a levantar el sistema web anteriormente descargado del contenedor, donde se le dieron las debidas especificaciones en el archivo .yaml, como ya los habiamos tirado abajo, si nos fijamos si existia un contenedor antes de realizar el paso anterior, no había nada, entonces podemos ver que este yaml puso en ejecución la app, que seria la web y tenia la referencia a la base de datos donde le pasamos el host y puerto, una vez configurado los n contenedores, terminamos de determinar el yaml y cuando este completo lo levanto con el up y se crea una red privada apartir de lo visto en el bridge network driver, que crea esta para que los contenedores puedan comunicarse, entonces lo mismo ocurre aqui, se crea esta red, donde crea un bridge del directorio de la web app con el contexto de la base de datos, que es lo denominado al final en el volumen redis_data.

- Tiramos abajo este yaml que se configuro, en el directorio del docker-compose
- con el comando docker-compose down

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/DockerCompose$ docker-compose down
Stopping dockercompose_app_1 ... done
Stopping dockercompose_db_1  ... done
Removing dockercompose_app_1 ... done
Removing dockercompose_db_1  ... done
Removing network dockercompose_default
```

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

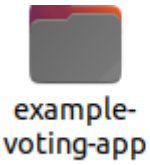
Este es un sistema compuesto por:

- Una aplicación web de Python que te permite votar entre dos opciones
- Una cola de Redis que recolecta nuevos votos
- Un trabajador .NET o Java que consume votos y los almacena en...
- Una base de datos de Postgres respaldada por un volumen de Docker
- Una aplicación web Node.js que muestra los dados de la votación en tiempo real.

Pasos:

- Clonar el repositorio https://github.com/dockersamples/example-voting-app

  git clone (url)

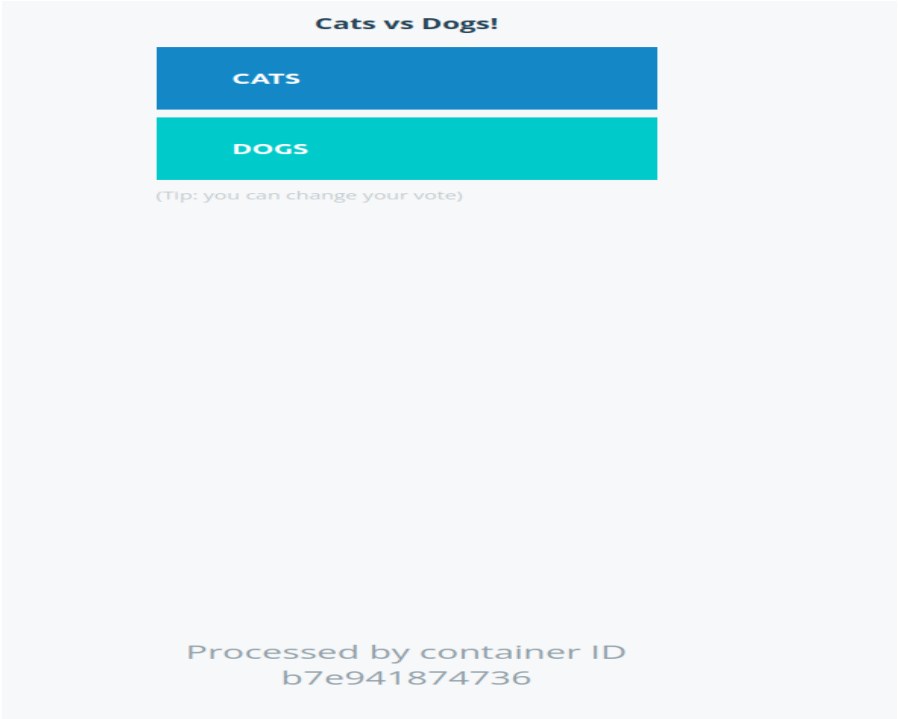


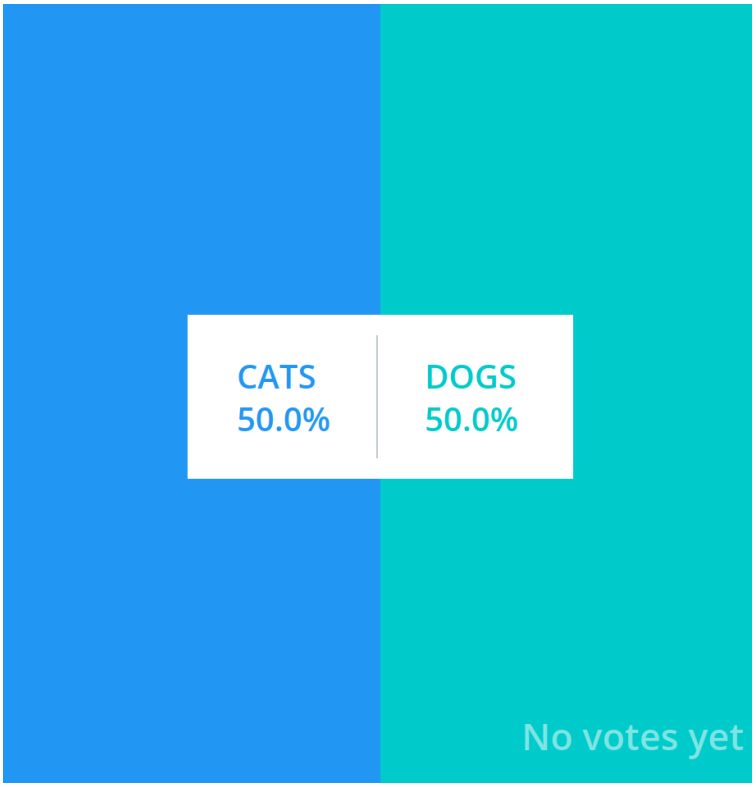
- Abrir una línea de comandos y ejecutar
- cd example-voting-app
- docker-compose -f docker-compose-javaworker.yml up -d

el ultimo comando, tenia un output largo, porque esta levantando algun contenedor de 0, que tiene muchos yaml configurados, por lo que es mas amplio que el anterior.

ultimo output

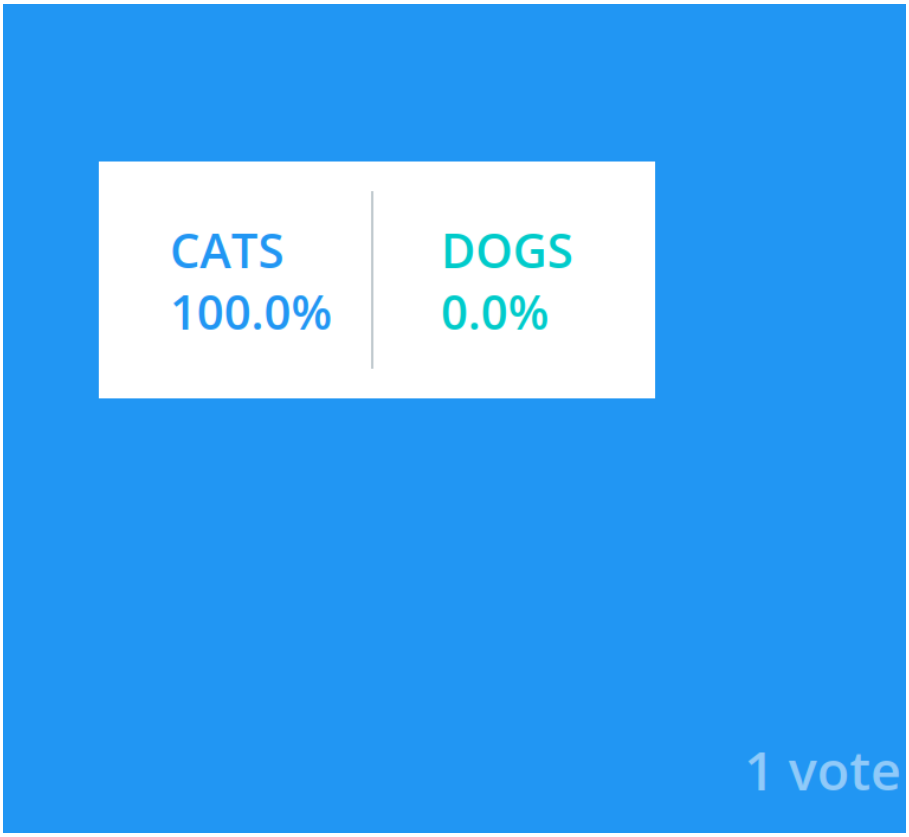
```
Digest: sha256:f362b165b870ef129cbe730f29065ff37399c0aa8bcab3e44b51c302938c9193
Status: Downloaded newer image for openjdk:8-jre-alpine
 ---> f7a292bbb70c
Step 9/10 : COPY --from=build /code/target/worker-jar-with-dependencies.jar /
 ---> 34f0ee118508
Step 10/10 : CMD ["java", "-XX:+UnlockExperimentalVMOptions", "-XX:+UseCGroupMem
oryLimitForHeap", "-jar", "/worker-jar-with-dependencies.jar"]
 ---> Running in f878773791fc
Removing intermediate container f878773791fc
 ---> 215ed4e38928
Successfully built 215ed4e38928
Successfully tagged example-voting-app_worker:latest
WARNING: Image for service worker was built because it did not already exist. To
 rebuild this image you must use `docker-compose build` or `docker-compose up --
build`.
Creating redis                      ... done
Creating example-voting-app_result_1 ... done
Creating example-voting-app_worker_1 ... done
Creating db                         ... done
Creating example-voting-app_vote_1   ... done
```

- Una vez terminado acceder a http://localhost:5000/ y http://localhost:5001
- 



emito un voto para mostrar el cambio en rt

- Emitir un voto y ver el resultado en tiempo real.



- Para emitir más votos, abrir varios navegadores diferentes para poder hacerlo
- Explicar como está configurado el sistema, puertos, volumenes componentes involucrados, utilizar el Docker compose como guía.

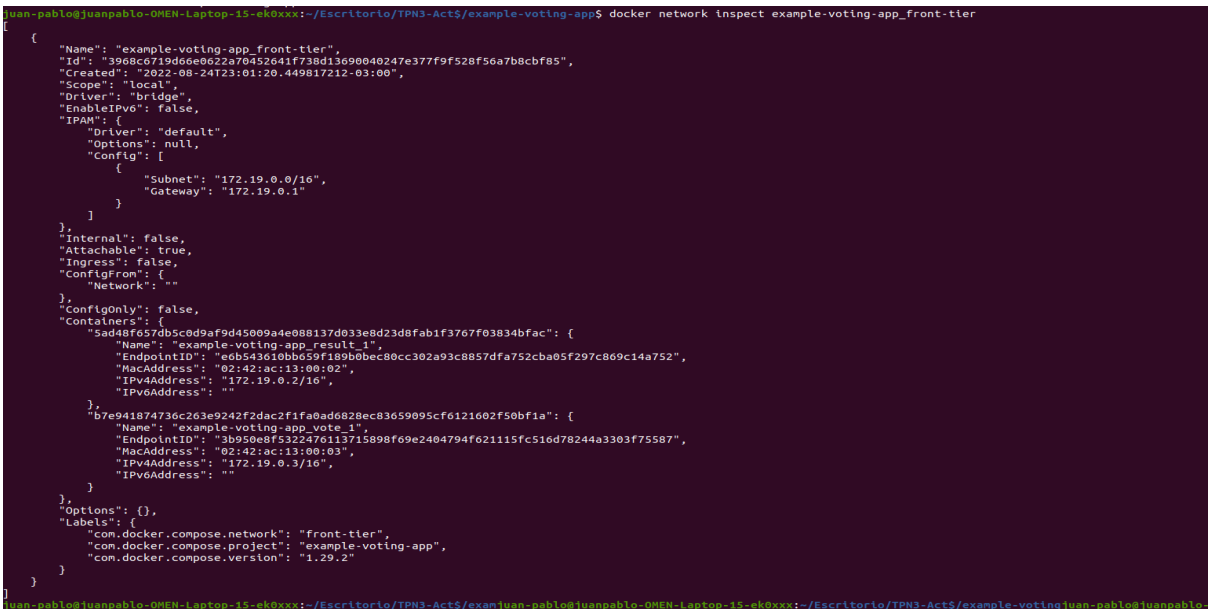
  Como habiamos visto en uno de los puntos anteriores, podemos traernos información de una o mas redes que por defecto el response es en json, entonces podemos hacer eso para poder ver la información de la red de esos contenedores,

entonces, ejecutamos el comando docket network inspect, le pasamos el nombre de la app en este caso example-voting-app concatenado _(network), que seria o front-tier o back-tier entonces el comando queda **docker network inspect example-voting-app_(back || front)-tier.**

red especificada en docker-compose: networks:back-tier

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/TPN3-Act$/example-voting-app$ docker network inspect example-voting-app_back-tier
[
    {
        "Name": "example-voting-app_back-tier",
        "Id": "5ba4fafc408d645a0f25ed18f2303562c4406af07f9b8636bf13c44c6a5be3b5",
        "Created": "2022-08-24T23:01:20.479548258-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.20.0.0/16",
                    "Gateway": "172.20.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "5ad48f657db5c0d9af9d45009a4e088137d033e8d23d8fab1f3767f03834bfac": {
                "Name": "example-voting-app_result_1",
                "EndpointID": "2742cefaeeb6011f88bb649f7946a8d9b1c3b57d2564ba4a0943c9255f04a8c6",
                "MacAddress": "02:42:ac:14:00:04",
                "IPv4Address": "172.20.0.4/16",
                "IPv6Address": ""
            },
            "66f232311bd27b6d8bcea82b3fca18b51fdeb5203e62abdde3f201d64f6f4b65": {
                "Name": "example-voting-app_worker_1",
                "EndpointID": "50b60e3bb6e7b598b7b619028b71111c91f2566bbc34e7ba5485e4b7f14d614e",
                "MacAddress": "02:42:ac:14:00:02",
                "IPv4Address": "172.20.0.2/16",
                "IPv6Address": ""
            },
            "8c9405eb9117ce124f1a18de9b3f19990485753bd79825accaecc2a5372bef1a": {
                "Name": "db",
                "EndpointID": "8afbf768c37f9629e9f3f671d451597dc0a77e26a49c380fd8e3ba0aab4b6c01",
                "MacAddress": "02:42:ac:14:00:06",
                "IPv4Address": "172.20.0.6/16",
                "IPv6Address": ""
            },
            "b7e941874736c263e9242f2dac2f1fa0ad6828ec83659095cf6121602f50bf1a": {
                "Name": "example-voting-app_vote_1",
                "EndpointID": "70b6d467e6f1da416a05e17e5f2977538cc4d361a9e2096093d0037a3e198abe",
                "MacAddress": "02:42:ac:14:00:05",
                "IPv4Address": "172.20.0.5/16",
                "IPv6Address": ""
            },
            "ff11f3fb6bfc2dec5a5b9e2cac65ec4c2fecd6658d7d50cf98a7e266d8f6381e": {
                "Name": "redis",
                "EndpointID": "e9d057f3dab65b2cd90c7f29b00ea32320b195aacfedcd43e326d23ecd13cfa6",
                "MacAddress": "02:42:ac:14:00:03",
                "IPv4Address": "172.20.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "back-tier",
            "com.docker.compose.project": "example-voting-app",
            "com.docker.compose.version": "1.29.2"
        }
    }
```

red especificada en docker-compose: networks:front-tier



- Con respecto a los volumenes

teniendo en cuenta algunos aspectos de los volumenes de docker como:

Un volumen de contenedor permite conservar los datos, aunque se elimine el Docker container. Los volúmenes también permiten un intercambio práctico de datos entre el host y el container.

Crear un volumen de Docker es una buena solución para poder:

- **Transferir datos a un contenedor de Docker**
- **Guardar los datos de un contenedor de Docker**
- **Intercambiar datos entre contenedores de Docker**

aca podemos ver los volumenes de datos de docker

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/TPN3-Act$/example-voting-app$ docker volume ls
DRIVER    VOLUME NAME
local     8df053da82a4e3c1fa653311ccb9fea5b1bf73ff61002ebc04b4a22c4d7f5607
local     16f77eafd33da292ab526d1815a50a9dbf37fc4da7bd4e249f5e7e8083c1bf19
local     478d8a8fbfef0f225bb3c98b504813a2a8b759cf520be6eb0971a11f45ba7162
local     dockercompose_redis_data
local     example-voting-app_db-data
local     fc68c4f59e97a21e251019c4dbe640064bec3fe7e72bc198b3405882c238d700
```

al igual que con las networks, lo que hice con el volumen es el inspect como muestra la siguiente imagen:

```
juan-pablo@juanpablo-OMEN-Laptop-15-ek0xxx:~/Escritorio/TPN3-Act$/example-voting-app$ sudo docker volume inspect example-voting-app_db-data
[sudo] contraseña para juan-pablo:
[
    {
        "CreatedAt": "2022-08-24T23:05:52-03:00",
        "Driver": "local",
        "Labels": {
            "com.docker.compose.project": "example-voting-app",
            "com.docker.compose.version": "1.29.2",
            "com.docker.compose.volume": "db-data"
        },
        "Mountpoint": "/var/lib/docker/volumes/example-voting-app_db-data/_data",
        "Name": "example-voting-app_db-data",
        "Options": null,
        "Scope": "local"
    }
]
```

donde nos muestra el mount point ubicado en ese path, que seria la db en teoría.

Con respecto a los puertos, además de verlo en el yaml y demas de forma manual, podemos tirar un docker ps si esta levantado para ver en cuales puertis esta actuando, podemos ver que hay dos imagenes o contenedores levantados, en el puerto 5001 esta el app_result de la aplicacion y en el 5000 esta el app.py que es el app_vote de la app.

```
CONTAINER ID   IMAGE                      COMMAND                  CREATED         STATUS         PORTS                                                                                NAMES
b7e941874736   example-voting-app_vote    "python app.py"          29 minutes ago  Up 29 minutes  0.0.0.0:5000->80/tcp, :::5000->80/tcp                                                example-voting-app_vote_1
8c9405eb9117   postgres:9.4               "docker-entrypoint.s…"   29 minutes ago  Up 29 minutes  5432/tcp                                                                             db
66f232311bd2   example-voting-app_worker  "java -XX:+UnlockExp…"   29 minutes ago  Up 29 minutes                                                                                       example-voting-app_worker_1
5ad48f657db5   example-voting-app_result  "docker-entrypoint.s…"   29 minutes ago  Up 29 minutes  0.0.0.0:5858->5858/tcp, :::5858->5858/tcp, 0.0.0.0:5001->80/tcp, :::5001->80/tcp     example-voting-app_result_1
ff11f3fb6bfc   redis:alpine               "docker-entrypoint.s…"   29 minutes ago  Up 29 minutes  0.0.0.0:49153->6379/tcp, :::49153->6379/tcp                                           redis
```

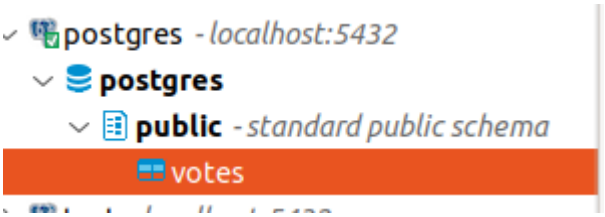
5- Análisis detallado

Exponer más puertos puertos para ver la configuración de Redis, y las tablas de PostgreSQL con alguna IDE como dbeaver. **Solucion** a este problema, el puerto escuchaba en local pero no le pegaba, entonces para que las dos partes escuchen el mismo puerto, el postgres ya esta en el 5432, en el javaworker.yaml tambien hay que agregar port:

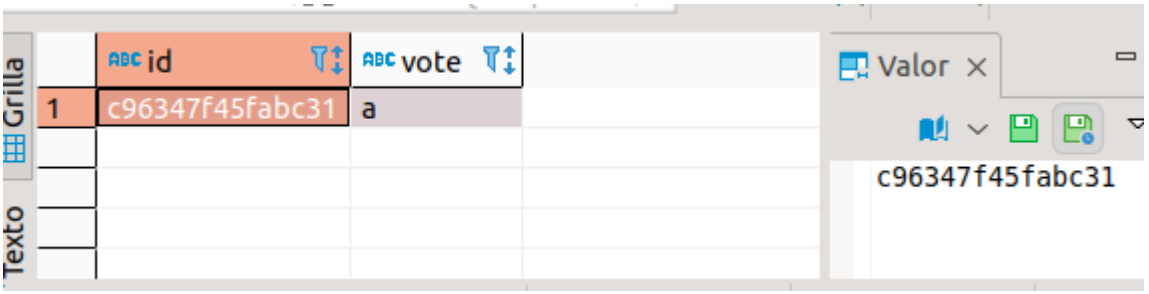
Para poder ingresar a la base de datos de la aplicacion, abro el debeaver y voy en busca de la base de datos que muestra en el output de la la terminal anterior, en el entrypoint, el puerto es el 5432. como aca

```yaml
redis:
  image: redis:alpine
  container_name: redis
  ports: ["6379"]
  networks:
    - back-tier

db:
  image: postgres:9.4
  container_name: db
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
  volumes:
    - "db-data:/var/lib/postgresql/data"
  ports:
    - "5432:5432"
  networks:
    - back-tier
```

ahora me conecto a la base de datos desde el dbeaver:

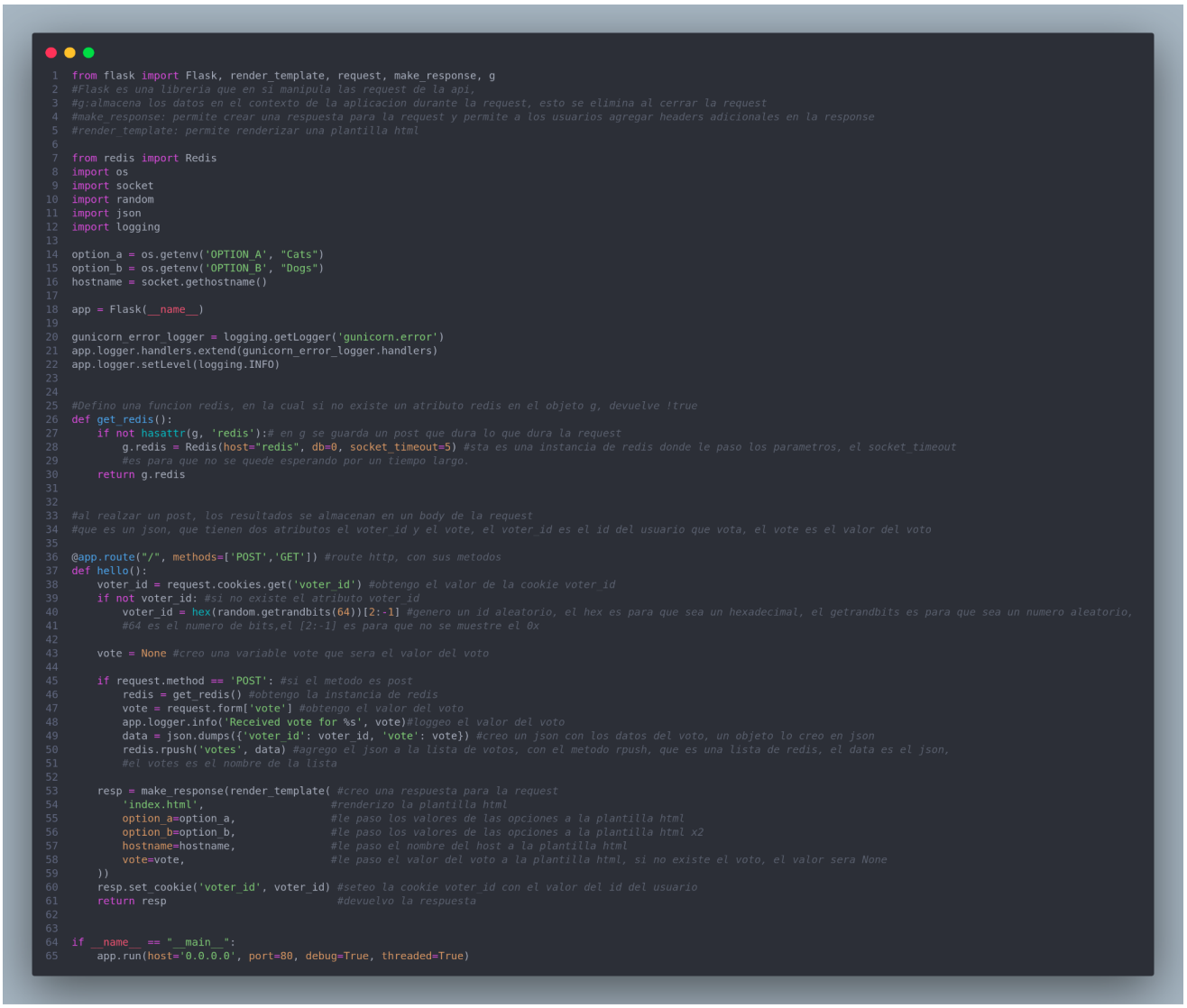


los datos de la db los veo en el yaml, en worker.yaml veo todos estos datos



- Revisar con el profe

- Revisar el código de la aplicación Python **example-voting-app\vote\app.py** para ver como envía votos a Redis.



```python
from flask import Flask, render_template, request, make_response, g
#Flask es una libreria que en si manipula las request de la api,
#g:almacena los datos en el contexto de la aplicacion durante la request, esto se elimina al cerrar la request
#make_response: permite crear una respuesta para la request y permite a los usuarios agregar headers adicionales en la response
#render_template: permite renderizar una plantilla html

from redis import Redis
import os
import socket
import random
import json
import logging

option_a = os.getenv('OPTION_A', "Cats")
option_b = os.getenv('OPTION_B', "Dogs")
hostname = socket.gethostname()

app = Flask(__name__)

gunicorn_error_logger = logging.getLogger('gunicorn.error')
app.logger.handlers.extend(gunicorn_error_logger.handlers)
app.logger.setLevel(logging.INFO)


#Defino una funcion redis, en la cual si no existe un atributo redis en el objeto g, devuelve !true
def get_redis():
    if not hasattr(g, 'redis'):# en g se guarda un post que dura lo que dura la request
        g.redis = Redis(host="redis", db=0, socket_timeout=5) #sta es una instancia de redis donde le paso los parametros, el socket_timeout
        #es para que no se quede esperando por un tiempo largo.
    return g.redis


#al realzar un post, los resultados se almacenan en un body de la request
#que es un json, que tienen dos atributos el voter_id y el vote, el voter_id es el id del usuario que vota, el vote es el valor del voto

@app.route("/", methods=['POST','GET']) #route http, con sus metodos
def hello():
    voter_id = request.cookies.get('voter_id') #obtengo el valor de la cookie voter_id
    if not voter_id: #si no existe el atributo voter_id
        voter_id = hex(random.getrandbits(64))[2:-1] #genero un id aleatorio, el hex es para que sea un hexadecimal, el getrandbits es para que sea un numero aleatorio,
        #64 es el numero de bits,el [2:-1] es para que no se muestre el 0x

    vote = None #creo una variable vote que sera el valor del voto

    if request.method == 'POST': #si el metodo es post
        redis = get_redis() #obtengo la instancia de redis
        vote = request.form['vote'] #obtengo el valor del voto
        app.logger.info('Received vote for %s', vote)#loggeo el valor del voto
        data = json.dumps({'voter_id': voter_id, 'vote': vote}) #creo un json con los datos del voto, un objeto lo creo en json
        redis.rpush('votes', data) #agrego el json a la lista de votos, con el metodo rpush, que es una lista de redis, el data es el json,
        #el votes es el nombre de la lista

    resp = make_response(render_template( #creo una respuesta para la request
        'index.html',                     #renderizo la plantilla html
        option_a=option_a,                #le paso los valores de las opciones a la plantilla html
        option_b=option_b,                #le paso los valores de las opciones a la plantilla html x2
        hostname=hostname,                #le paso el nombre del host a la plantilla html
        vote=vote,                        #le paso el valor del voto a la plantilla html, si no existe el voto, el valor sera None
    ))
    resp.set_cookie('voter_id', voter_id) #seteo la cookie voter_id con el valor del id del usuario
    return resp                           #devuelvo la respuesta


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True, threaded=True)
```

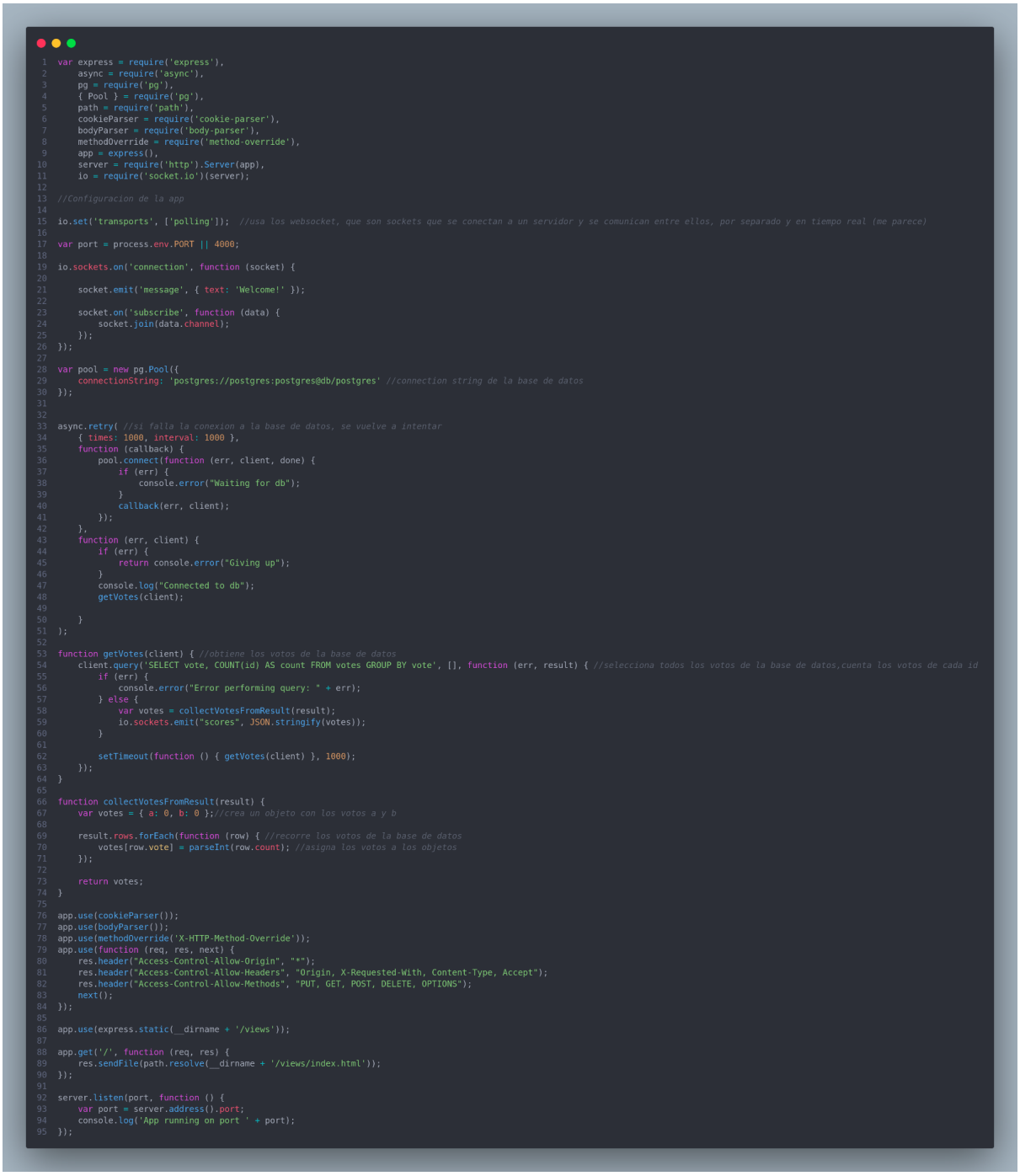
Revisar el código del worker **example-voting-app\worker\src\main\java\worker\Worker.java** para entender como procesa los datos.

En este script de java , lo que hace primero es crear un objeto redis de tipo Jedis donde se conecta al cliente de redis, luego el dbconn es otro objeto o variable de tipo Connection, donde le pasamos el nombre de la base de datos o host que seria db, seria el cliente de la base de datos postgres, despues trae los votos de la cola de votos y los guarda en una variable que la convierte en json, y lo almacenamos en tipo clave valor al voter_id y vote, en

dos variables separadas, sabemos que en redis las db en memoria trabajan con clave valor, despues se insertan estos votos en una tupla de la base de datos, como podemos ver en el metodo updateVote(...), donde le pasamos los parametros y hace la consulta a postgres.
y mas abajo estan los métodos para las conexiones con las excepciones tanto para el cliente de redis como el de postgres

```java
package worker;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.exceptions.JedisConnectionException;
import java.sql.*;
import org.json.JSONObject;

class Worker {
    public static void main(String[] args) {
        try {
            Jedis redis = connectToRedis("redis"); // connect to redis database
            Connection dbConn = connectToDB("db"); // connect to database

            System.err.println("Watching vote queue"); // print message to console

            while (true) {
                String voteJSON = redis.blpop(0, "votes").get(1); // trae los votos de la cola de votos, y los guarda en la variable voteJSON
                JSONObject voteData = new JSONObject(voteJSON); // convierte el string a un objeto JSON
                String voterID = voteData.getString("voter_id"); // obtiene el id del votante
                String vote = voteData.getString("vote"); // obtiene el voto

                System.err.printf("Processing vote for '%s' by '%s'\n", vote, voterID);
                updateVote(dbConn, voterID, vote);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    static void updateVote(Connection dbConn, String voterID, String vote) throws SQLException { // actualiza el voto en la base de datos
        PreparedStatement insert = dbConn.prepareStatement(
                "INSERT INTO votes (id, vote) VALUES (?, ?)");
        insert.setString(1, voterID);
        insert.setString(2, vote);

        try {
            insert.executeUpdate();
        } catch (SQLException e) {
            PreparedStatement update = dbConn.prepareStatement(
                    "UPDATE votes SET vote = ? WHERE id = ?");
            update.setString(1, vote);
            update.setString(2, voterID);
            update.executeUpdate();
        }
    }

    static Jedis connectToRedis(String host) { // conecta a la base de datos redis , le pasa el nombre del host
        Jedis conn = new Jedis(host);

        while (true) {
            try {
                conn.keys("*");
                break;
            } catch (JedisConnectionException e) {
                System.err.println("Waiting for redis");
                sleep(1000);
            }
        }

        System.err.println("Connected to redis");
        return conn;
    }

    static Connection connectToDB(String host) throws SQLException { //conecta al a base de datos postgres , le pasa el nombre del host
        Connection conn = null;

        try {

            Class.forName("org.postgresql.Driver");
            String url = "jdbc:postgresql://" + host + "/postgres";

            while (conn == null) {
                try {
                    conn = DriverManager.getConnection(url, "postgres", "postgres");
                } catch (SQLException e) {
                    System.err.println("Waiting for db");
                    sleep(1000);
                }
            }

            PreparedStatement st = conn.prepareStatement(
                    "CREATE TABLE IF NOT EXISTS votes (id VARCHAR(255) NOT NULL UNIQUE, vote VARCHAR(255) NOT NULL)");
            st.executeUpdate();

        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        System.err.println("Connected to db");
        return conn;
    }

    static void sleep(long duration) {
        try {
            Thread.sleep(duration);
        } catch (InterruptedException e) {
            System.exit(1);
        }
    }
}
```

Revisar el código de la aplicacion que muestra los resultados **example-voting-app\result\server.js** para entender como muestra los valores.

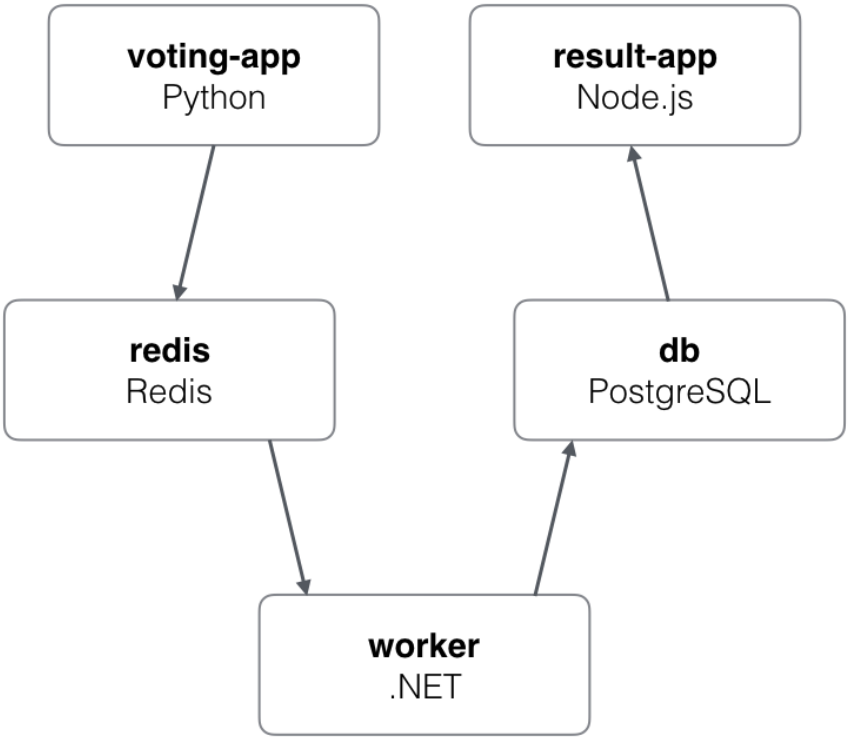


```
var express = require('express'),
    async = require('async'),
    pg = require('pg'),
    { Pool } = require('pg'),
    path = require('path'),
    cookieParser = require('cookie-parser'),
    bodyParser = require('body-parser'),
    methodOverride = require('method-override'),
    app = express(),
    server = require('http').Server(app),
    io = require('socket.io')(server);

//Configuracion de la app

io.set('transports', ['polling']);  //usa los websocket, que son sockets que se conectan a un servidor y se comunican entre ellos, por separado y en tiempo real (me parece)

var port = process.env.PORT || 4000;

io.sockets.on('connection', function (socket) {

    socket.emit('message', { text: 'Welcome!' });

    socket.on('subscribe', function (data) {
        socket.join(data.channel);
    });
});

var pool = new pg.Pool({
    connectionString: 'postgres://postgres:postgres@db/postgres' //connection string de la base de datos
});


async.retry( //si falla la conexion a la base de datos, se vuelve a intentar
    { times: 1000, interval: 1000 },
    function (callback) {
        pool.connect(function (err, client, done) {
            if (err) {
                console.error("Waiting for db");
            }
            callback(err, client);
        });
    },
    function (err, client) {
        if (err) {
            return console.error("Giving up");
        }
        console.log("Connected to db");
        getVotes(client);

    }
);

function getVotes(client) { //obtiene los votos de la base de datos
    client.query('SELECT vote, COUNT(id) AS count FROM votes GROUP BY vote', [], function (err, result) { //selecciona todos los votos de la base de datos,cuenta los votos de cada id
        if (err) {
            console.error("Error performing query: " + err);
        } else {
            var votes = collectVotesFromResult(result);
            io.sockets.emit("scores", JSON.stringify(votes));
        }

        setTimeout(function () { getVotes(client) }, 1000);
    });
}

function collectVotesFromResult(result) {
    var votes = { a: 0, b: 0 };//crea un objeto con los votos a y b

    result.rows.forEach(function (row) { //recorre los votos de la base de datos
        votes[row.vote] = parseInt(row.count); //asigna los votos a los objetos
    });

    return votes;
}

app.use(cookieParser());
app.use(bodyParser());
app.use(methodOverride('X-HTTP-Method-Override'));
app.use(function (req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
    res.header("Access-Control-Allow-Methods", "PUT, GET, POST, DELETE, OPTIONS");
    next();
});

app.use(express.static(__dirname + '/views'));

app.get('/', function (req, res) {
    res.sendFile(path.resolve(__dirname + '/views/index.html'));
});

server.listen(port, function () {
    var port = server.address().port;
    console.log('App running on port ' + port);
});
```

- **Escribir un documento de arquitectura sencillo, pero con un nivel de detalle moderado, que incluya algunos diagramas de bloques, de sequencia, etc y descripciones de los distintos componentes involucrados es este sistema y como interactuan entre sí.**

En la ultima consigna del trabajo practico nos brindaron una web app que consistia en realizar votos y el mismo tenia 5 sistemas o contenedores que usan python,node.js, .NET Core, con redis y una base de datos postgres.

- Una aplicación web de Python que te permite votar entre dos opciones
- Una cola de Redis que recolecta nuevos votos
- Un trabajador .NET o Java que consume votos y los almacena en...
- Una base de datos de Postgres respaldada por un volumen de Docker
- Una aplicación web Node.js que muestra los dados de la votación en tiempo real.



Aca se pueden ver un diagrama de la aplicación, donde tenemos las dos aplicaciones, el usuario emite un voto en la voting-app , es decir hace un post que se envia a la app, que dentro del mismo se extrae la información del usuario y desde el voting app, despues del post, me trae una response hace un push a redis de un json que tiene como veiamos en el codigo el id y el vote, este como dijimos lo manda a redis que se encarga de asignarle una clave y almacenarlo, luego el **worker** lo que hace es consumir esos datos de redis, es decir vacia la cola de redis, como una cola de estructura de datos, obtiene el json con los atributos de id y vote e una vez obtenidos todos los datos de la cola de redis, los inserta en la base de datos **db,** de postgres, despues result- app lo que hace es un get o una consulta a la base de datos postgres donde muestra la cantidad de votos ya sean **a cats b ,dogs** y los muestra al usuario.

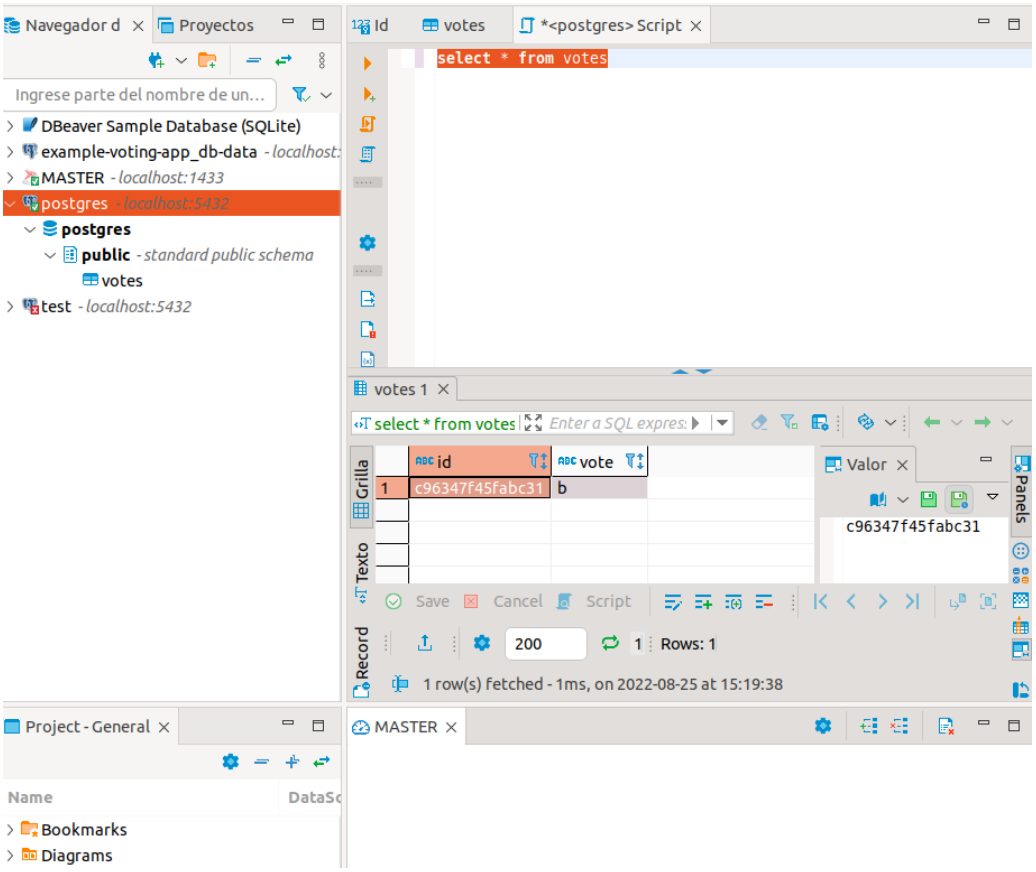


Diagrama de secuencia de la arquitectura completa