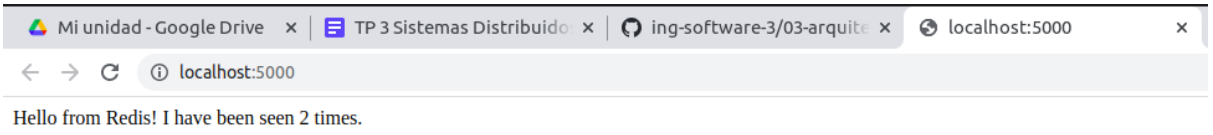
1- Sistema distribuido simple

```
manuel@manuel-Aspire-A315-54K:~$  docker run -d --net mybridge --name 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
e3959d419d4e56e50fdc0fd3d6f0c6910917148d4f06488ef3fae2f2adee794e
```

```
manuel@manuel-Aspire-A315-54K:~$   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
9e06e87dba40b2f4ccb83fad1544c0f03a31d2f43303010b3cc7c5dd884d3c10
```




```
manuel@manuel-Aspire-A315-54K:~$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED              STATUS              PORTS                                       NAMES
9e06e87dba40   alexisfr/flask-app:latest   "python /app.py"         About a minute ago   Up About a minute   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   web
e3959d419d4e   redis:alpine             "docker-entrypoint.s…"   6 minutes ago        Up 6 minutes        6379/tcp                                    db
```

el puerto que se encuentra expuesto es el 5000

```
manuel@manuel-Aspire-A315-54K:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
e67a9ae8f932   bridge              bridge    local
165a24285a92   ejcompose_default   bridge    local
225fc4a33d53   ejcompose_mired     bridge    local
ee659bda9877   host                host      local
87f25b928f43   mybridge            bridge    local
6e354087e153   none                null      local
```

```
manuel@manuel-Aspire-A315-54K:~$ docker network inspect mybridge
[
    {
        "Name": "mybridge",
        "Id": "87f25b928f43303254702971fb9238c6ce0afc12ae67a5157f4ecb5c6bc0a9ea",
        "Created": "2022-08-24T14:11:34.199457876-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.20.0.0/16",
                    "Gateway": "172.20.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9e06e87dba40b2f4ccb83fad1544c0f03a31d2f43303010b3cc7c5dd884d3c10": {
                "Name": "web",
                "EndpointID": "4843cf967c3cbea4bd9d52f3baf7aa0be43bcccd5545b4a939440c12c9b4c411",
                "MacAddress": "02:42:ac:14:00:03",
                "IPv4Address": "172.20.0.3/16",
                "IPv6Address": ""
            },
            "e3959d419d4e56e50fdc0fd3d6f0c6910917148d4f06488ef3fae2f2adee794e": {
                "Name": "db",
                "EndpointID": "0e98eb85f9dccd5996ff457c87d604fd2891e8d8097872de4de47723a4c6e03b",
                "MacAddress": "02:42:ac:14:00:02",
                "IPv4Address": "172.20.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

para poder revisar los detalles de *mybridge* se utilizó el comando *docker network inspect <red>*

2- Análisis del sistema

```
import os

from flask import Flask
from redis import Redis


app = Flask(__name__)
redis                =                Redis(host=os.environ['REDIS_HOST'],
port=os.environ['REDIS_PORT'])
bind_port = int(os.environ['BIND_PORT'])


@app.route('/')
def hello():
    redis.incr('hits')
    total_hits = redis.get('hits').decode()
    return f'Hello from Redis! I have been seen {total_hits} times.'


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

El código utiliza las variables de entorno generadas con la creación de los contenedores para poder realizar la correcta configuración de las instancias creadas, para esto se utiliza el elemento -e.

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?

No se percibe ningún cambio

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

Al borrar la base de datos, la página no es capaz de levantarse, debido a que falla la conección con Redis

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

El contador de la página se reinicia

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

Generar una variable de entorno que almacene la cantidad de visitas y cargar la cantidad desde allí

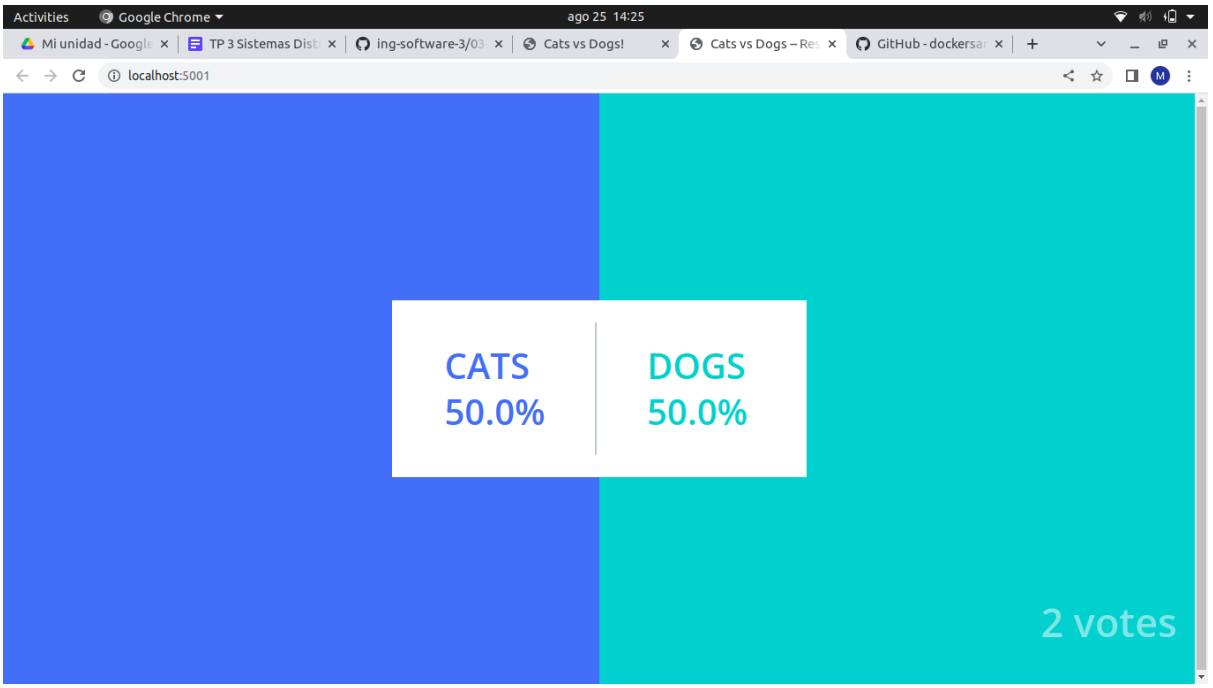
3- Utilizando docker compose

```
manuel@manuel-Aspire-A315-54K:~/Documents$ docker-compose up -d
Creating network "documents_default" with the default driver
Creating volume "documents_redis_data" with default driver
Creating documents_db_1 ... done
Creating documents_app_1 ... done
```

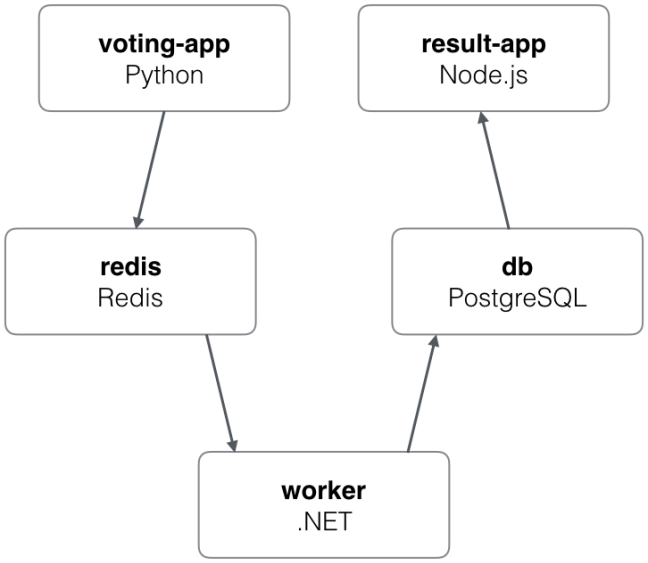
Con la utilización del archivo .yaml y docker compose fuimos capaces de realizar la configuración de los diversos contenedores, para así utilizar el comando up y poder correr todos los contenedores de manera correcta, en el documento .yaml se realizó la configuración de los puertos, las variables de entorno, las imágenes sobre las cuales se deben generar los contenedores, la creación de volúmenes, y también definir dependencias entre contenedores (en este caso, el contenedor app depende de db para poder crearse

```
manuel@manuel-Aspire-A315-54K:~/Documents$ docker-compose down
Stopping documents_app_1 ... done
Stopping documents_db_1  ... done
Removing documents_app_1 ... done
Removing documents_db_1  ... done
Removing network documents_default
```

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



La arquitectura lograda por el docker compose es la siguiente



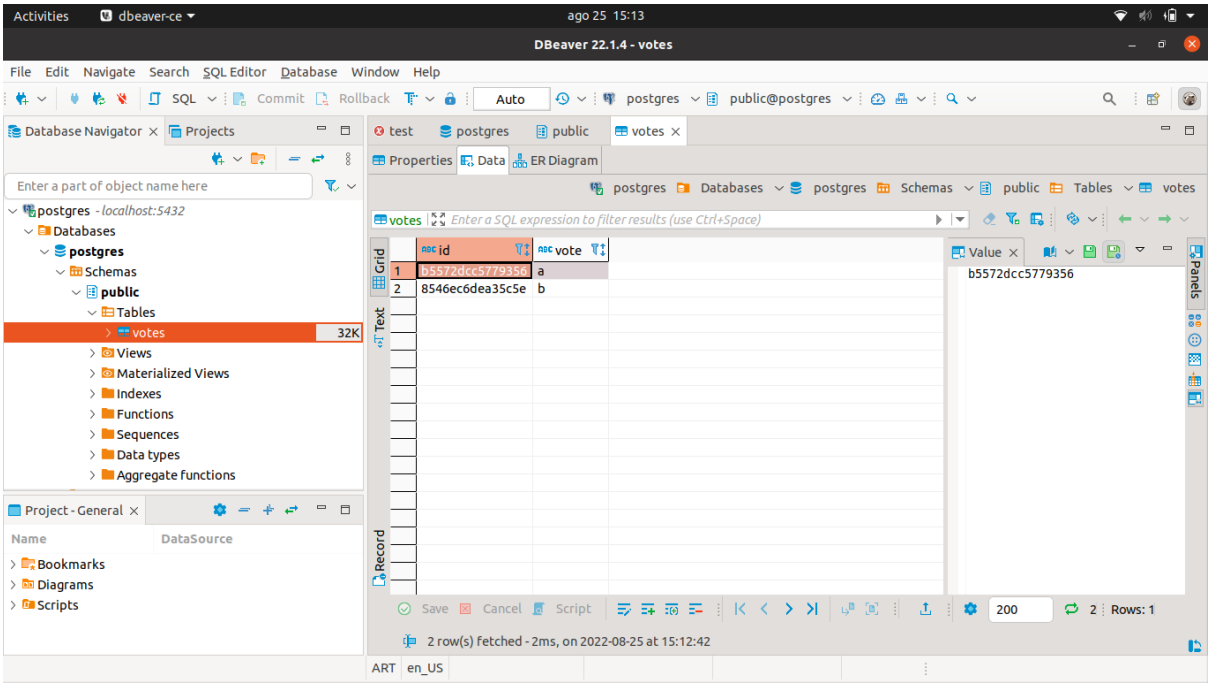
Leyendo el archivo docker-compose.yaml se observa que del contenedor

- **vote** se linkean los puertos 80 del contenedor con el 5000 del host, utiliza las redes front-tier y back-tier y genera el volumen ./vote:/app
- **result** se linkean los puertos 80 y 5858 del contenedor con los puerto 5001 y 5858 del host, utiliza las redes front-tier y back-tier y genera el volumen ./result:/app
- **worker** utiliza la red back-tier
- **redis** genera el volumen ./healthchecks:/healthchecks y expone el puerto 6379, utiliza back-tier

- **db** define las siguientes variables de entorno POSTGRES_USER: "postgres" POSTGRES_PASSWORD: "postgres", crea los volúmenes db-data:/var/lib/postgresql/data y ./healthchecks:/healthchecks, y utiliza la red back-tier

5- Análisis detallado

Se agregaron a la configuración del archivo del docker compose los puertos 5432:5432 en el contenedor db



[to do]