

Trabajo Práctico 3

Materia: Ingeniería de Software 3

Alumno: Vietto Santiago

Docente: Fernando Bono

Institución: UCC

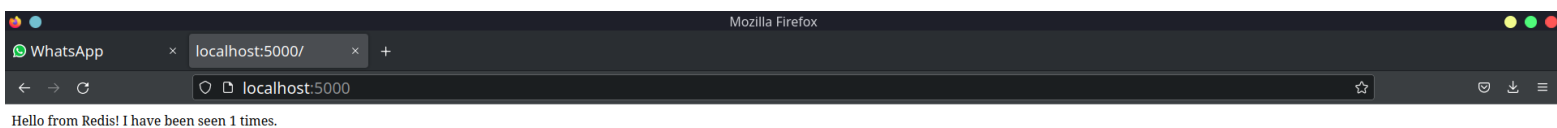
Año: 2022

Ejercicio 1:



```

[~] ~$ sudo systemctl start docker.service
[sudo] password for vietto:
[~] ~$ sudo docker network create -d bridge mybridge
d0d9970be5b3512e8b7a9b62f869bf5a7ee9845aacf0efc9d681a62ffe6e53c0
[~] ~$ sudo 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
734dbbafab8804d3afcf5a5773fa223d842e16be10f324a61fa8c5d08cb75367
[~] ~$ sudo 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: sha256:250221bea53e4e8f99a7ce79023c978ba0df69bdf620401756da46e34b7c80b
Status: Downloaded newer image for alexisfr/flask-app:latest
e9ca311d55add13514c5bbdce3065260935f82ed834171a1ac4a26d6b87c8be3

```





```

  ~ sudo docker network ls

```

NETWORK ID	NAME	DRIVER	SCOPE
a21dcd5d3a54	bridge	bridge	local
6f35b4a96aeb	host	host	local
d0d9970be5b3	mybridge	bridge	local
3ef75ccf0f1d	none	null	local

```

  ~ sudo docker network inspect mybridge

```

```

[
  {
    "Name": "mybridge",
    "Id": "d0d9970be5b3512e8b7a9b62f869bf5a7ee9845aacf0efc9d681a62ffe6e53c0",
    "Created": "2022-08-25T14:26:33.829496278-03:00",
    "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": {
      "734dbbafab8804d3afcfa5773fa223d842e16be10f324a61fa8c5d08cb75367": {
        "Name": "db",
        "EndpointID": "b2fb5df1c88d838b638a586cd3a10a4a6fbdae21820fc2f7e5f2facbc2420e4f",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      },
      "e9ca311d55add13514c5bbdce3065260935f82ed834171a1ac4a26d6b87c8be3": {
        "Name": "web",
        "EndpointID": "158871a1e702a406c1252b19a07fa579f9029c229a61fab7cc38bb00a4f5bfb7",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]

```

Ejercicio 2:

```
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'])

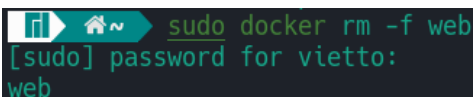
@app.route('/')

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

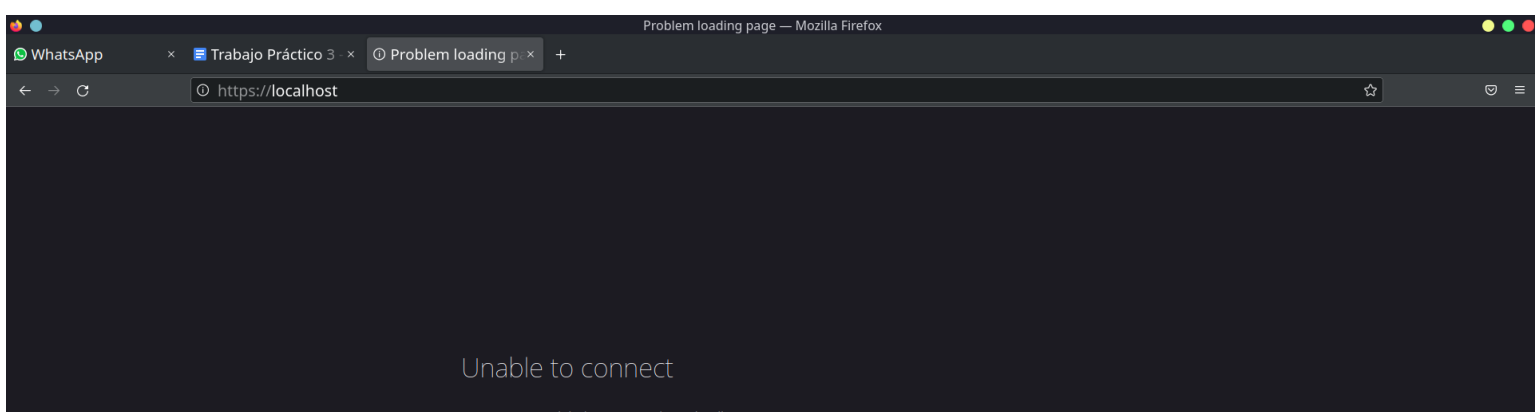
if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True, port=bind_port)
```

Explicación del código: analizando el mismo observamos que al comienzo se importa un framework llamado Flask que nos permite crear aplicaciones web de manera rápida y con un mínimo número de líneas de código, y por otro lado se importa Redis (Remote Dictionary Server) que es una base de datos con un motor de almacenamiento clave-valor. Se instancia una variable port a la cual se le va a asignar el puerto que tiene asignado Redis, es decir, el puerto que está reservado para la base de datos, y de igual manera con la variable bind_port pero en este caso se le asigna el puerto de la aplicación web. Mediante la anotación @app.route se aclara la ruta con la cual se va a acceder a los datos de la base de datos a través de la web en la URL. Siguiendo, se declara una función hello, mediante la cual, usando la base de datos Redis, se incrementa un contador hits cada vez que se carga la página y se almacenan en la variable total_hits. Luego devuelve un mensaje que se muestra en la página con la cantidad total de hits. Al final, tenemos el main en donde se declara que la aplicación web usando Flask corre en el host 0.0.0.0 y el puerto 5000.

Los parámetros `-e` en el segundo Docker run del ejercicio 1, se utilizan para setear variables de entorno en el contenedor, en donde seteas el nombre de la base de datos y el puerto, y desde un programa o el entorno se las puede buscar a dicha variables.



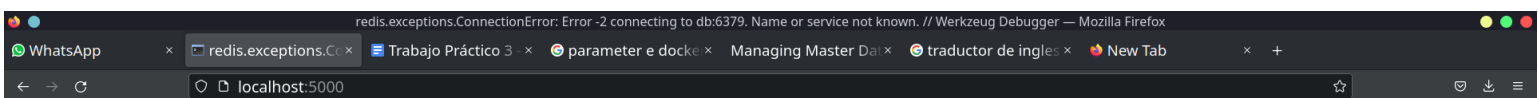
```
➜ sudo docker rm -f web
[sudo] password for vietto:
web
```



```
ef500f705ca94fa6617443c016308c5404d5d0d267342b6dc4488d6a5a960910
sudo docker run -d --net mybridge -e REDIS_HOST=db -e REDIS_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest
```

Cuando borramos el contenedor web, la imagen de db sigue corriendo pero no vamos a poder acceder a la misma. Al volver a levantar la imagen de web, vamos a seguir teniendo los datos, porque db nunca freno. Pero si borramos db y web, al levantar web vamos a empezar de cero sin datos en db.

```
db
sudo docker rm -f db
```



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 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/redis/client.py", line 667, in execute_command
    connection.send_command(*args)

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line 610, in send_command
    self.send_packed_command(self.pack_command(*args))

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line 585, in send_packed_command
    self.connect()

File "/usr/local/lib/python3.6/site-packages/redis/connection.py", line 489, in connect
    raise ConnectionError(self._error_message(e))
```

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/socket.py", line 745, in getaddrinfo
    for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
```

```
sudo docker run -d --net mybridge --name db redis:alpine
2a8c9c28fab530204a922acb86f2deb179505feaa8ba4718d6ed989d6a49bb90
```

Cuando borro el contenedor de Redis con `docker rm -f db`, lo que sucede es que este comando va a forzar su eliminación por más que esté corriendo. Se borra la imagen de la

base de datos, pero todavía podemos acceder, por lo que nos muestra un error de que la base de datos no existe. Si levanto nuevamente db con `docker run -d --net mybridge --name db redis:alpine`, podemos acceder a la página web ya que la imagen de web está corriendo y se levanta la imagen de la base de datos pero desde cero.

De esta manera podemos ver la independencia de las dos imágenes.

Para no perder la cuenta de las visitas se podría pensar en almacenar las mismas en otra base de datos de respaldo, para que en el caso de que sea borrada la original, no se pierdan los datos.

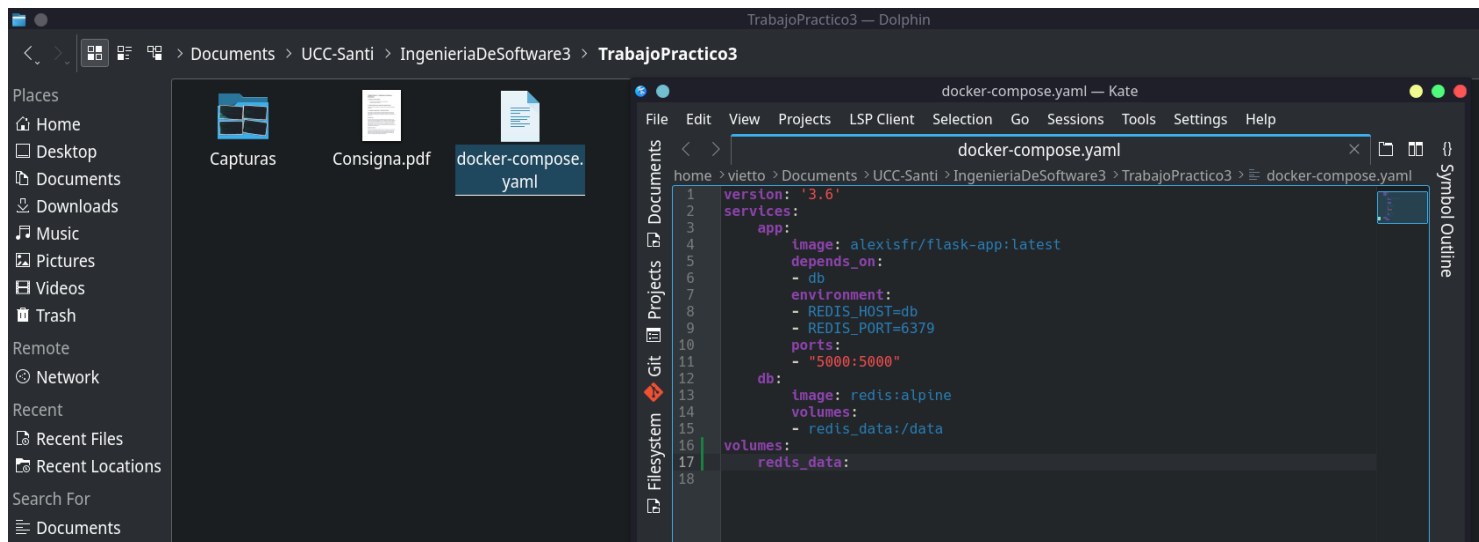
Eliminamos los contenedores:

```
❯ sudo docker rm -f db
[sudo] password for vietto:
db
❯ sudo docker rm -f web
web
❯ sudo docker network rm mybridge
mybridge
```

Ejercicio 3:

```
❯ sudo pip install docker-compose
Collecting docker-compose
  Downloading docker_compose-1.29.2-py2.py3-none-any.whl (114 kB)
    114.8/114.8 kB 1.2 MB/s eta 0:00:00
Collecting PyYAML<6,>=3.10
  Downloading PyYAML-5.4.1.tar.gz (175 kB)
    175.1/175.1 kB 1.2 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: distro<2,>=1.5.0 in /usr/lib/python3.10/site-packages (from docker-compose) (1.7.0)
Collecting docker[ssh]>=5
  Downloading docker-6.0.0-py3-none-any.whl (147 kB)
    147.2/147.2 kB 1.2 MB/s eta 0:00:00
Collecting dockerpty<1,>=0.4.1
  Downloading dockerpty-0.4.1.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: docopt<1,>=0.6.1 in /usr/lib/python3.10/site-packages (from docker-compose) (0.6.2)
Collecting jsonschema<4,>=2.5.1
  Downloading jsonschema-3.2.0-py2.py3-none-any.whl (56 kB)
    56.3/56.3 kB 1.3 MB/s eta 0:00:00
Collecting python-dotenv<1,>=0.13.0
  Downloading python_dotenv-0.20.0-py3-none-any.whl (17 kB)
Requirement already satisfied: requests<3,>=2.20.0 in /usr/lib/python3.10/site-packages (from docker-compose) (2.28.1)
Collecting texttable<2,>=0.9.0
  Downloading texttable-1.6.4-py2.py3-none-any.whl (10 kB)
Collecting websocket-client<1,>=0.32.0
  Downloading websocket_client-0.59.0-py2.py3-none-any.whl (67 kB)
```

```
❯ ls
Desktop Documents Downloads eclipse eclipse-workspace IngSoft3 IngSoft3Clone Music paginaWebHotel Pictures Public
❯ cd Documents
❯ ls
Commands.txt UCC-Santi
❯ cd UCC-Santi
❯ l
zsh: command not found: l
❯ ls
'Arquitectura de Software 2' 'Computacion Grafica' IngenieriaDeSoftware3 'Sistemas operativos'
❯ cd IngenieriaDeSoftware3
❯ LS
zsh: correct 'LS' to 'ls' [nyae]? n
zsh: command not found: LS
❯ ls
Trabajopractico1 TrabajoPractico2 TrabajoPractico3
```

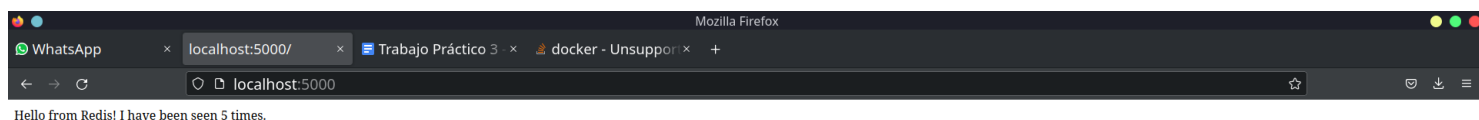


```
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 $ sudo docker-compose up -d
Creating network "trabajopractico3_default" with the default driver
Creating volume "trabajopractico3_redis_data" with default driver
Creating trabajopractico3_db_1 ... done
Creating trabajopractico3_app_1 ... done

~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 $ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
af925df64924   alexisfr/flask-app:latest           "python /app.py"        About a minute Up About a minute   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   trabajopractico3_app_1
93f4d2488928   redis:alpine                        "docker-entrypoint.s..." About a minute ago Up About a minute   6379/tcp                                     trabajopractico3_db_1

~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 $ sudo docker network ls
NETWORK ID     NAME                                  DRIVER  SCOPE
a21dc5d3a54    bridge                               bridge  local
6f35b4a96aeb   host                                 host    local
3ef75ccf0f1d   none                                 null    local
3d4c90823dc2   trabajopractico3_default             bridge  local

~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 $ sudo docker volume ls
DRIVER    VOLUME NAME
local     31a1e729ad7dbe7088c44bccf82136a3e9084a11cd419e880580aa7976c9ad45
local     97648a5e86271ba6c727a8cad4f98de67114d9b6e3ffa08fb96dc1c22995e81a
local     trabajopractico3_redis_data
```



De otra forma, decimos que Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de esta forma es más sencillo crear contenedores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc. Nos permite lanzar un solo comando para crear e iniciar todos los servicios desde su configuración (YAML), esto significa que puedes crear diferentes contenedores y al mismo tiempo diferentes servicios en cada contenedor, integrarlos a un volumen común e iniciarlos y/o apagarlos, etc.

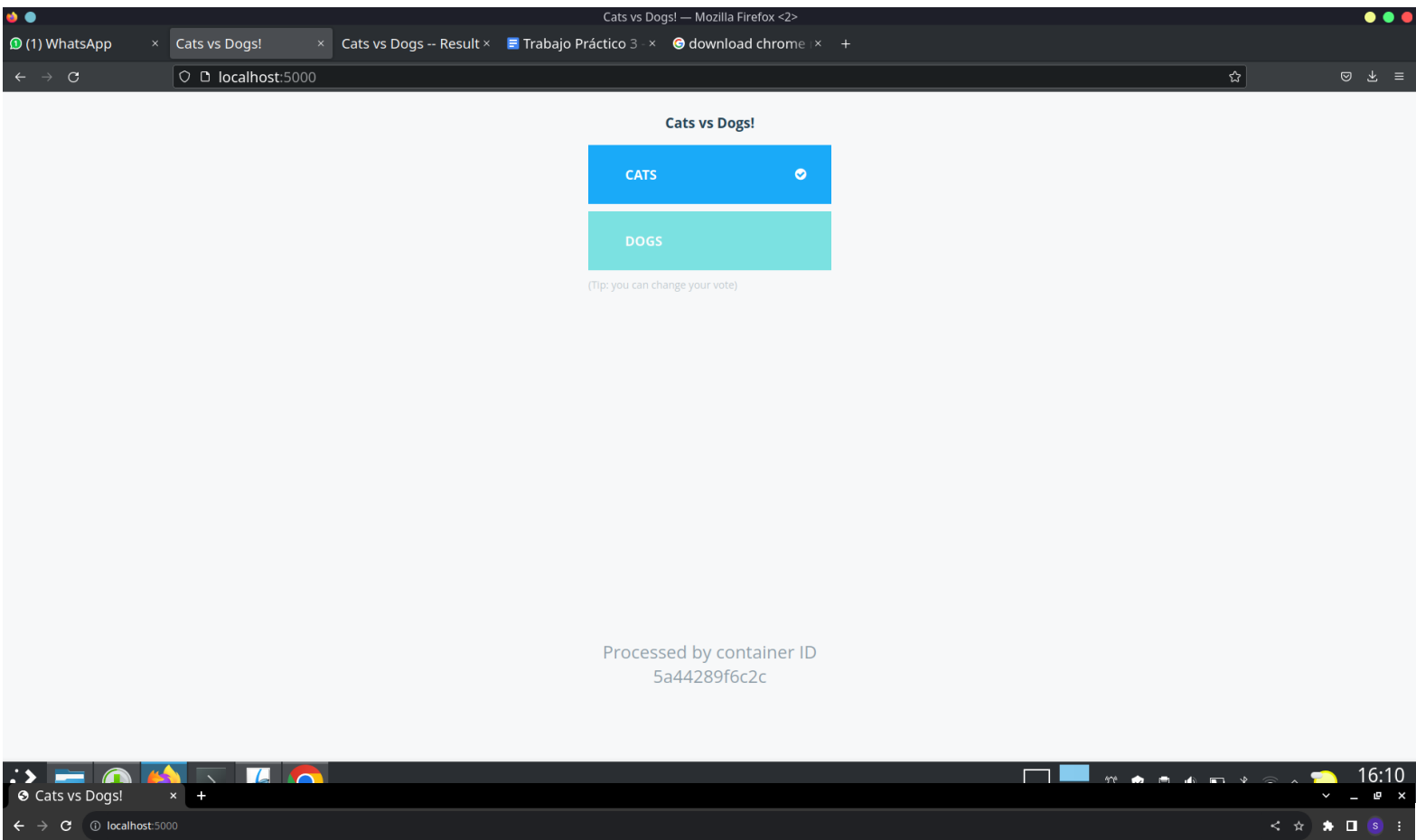
```
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 sudo docker-compose down
Stopping trabajopractico3_app_1 ... done
Stopping trabajopractico3_db_1 ... done
Removing trabajopractico3_app_1 ... done
Removing trabajopractico3_db_1 ... done
Removing network trabajopractico3_default
```

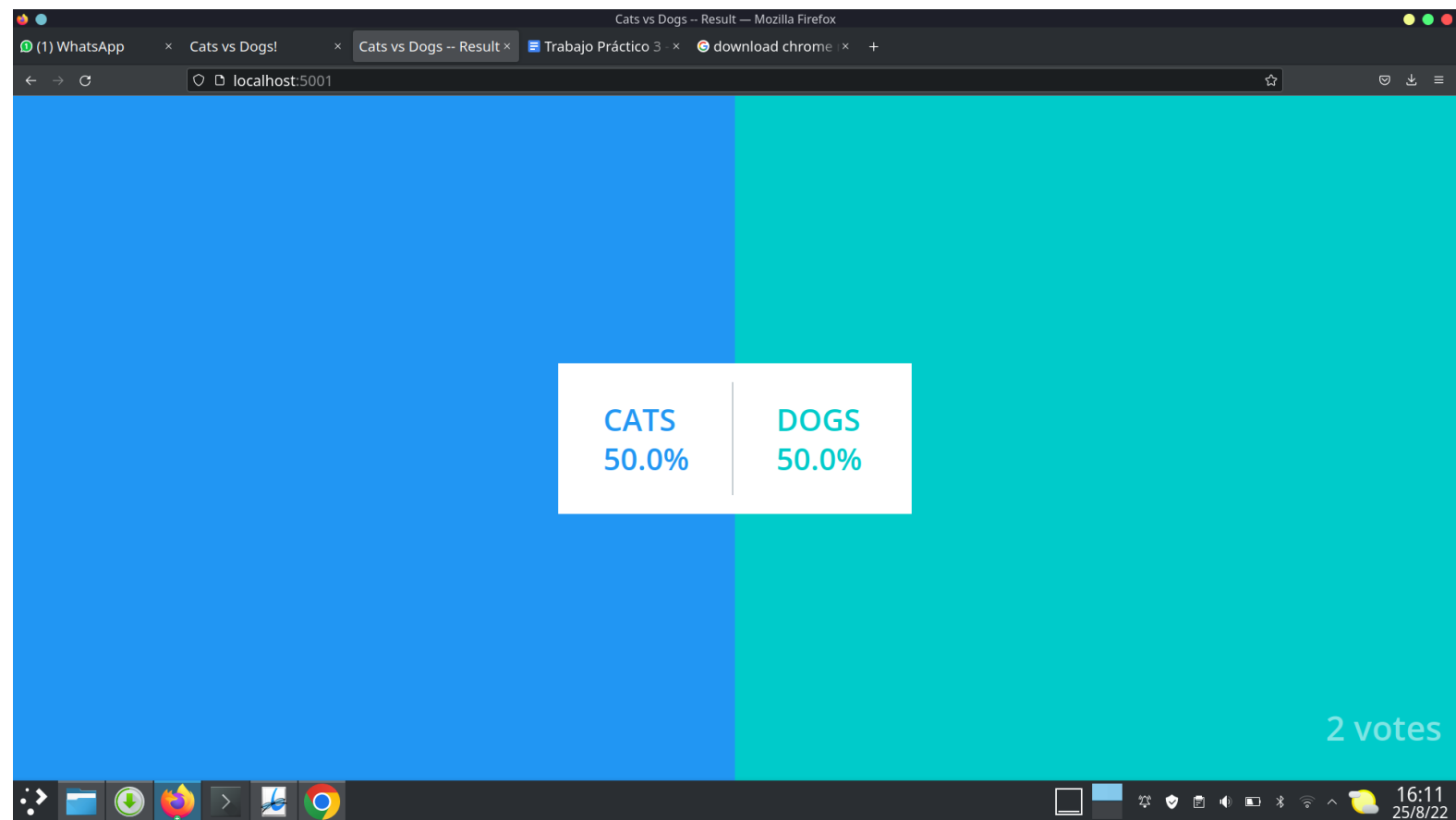
Ejercicio 4:

```
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 git config --global user.name "santi2019"
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 git config --global user.email "santiagovietto5@hotmail.com"
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 git clone https://github.com/dockerexamples/example-voting-app
Cloning into 'example-voting-app'...
remote: Enumerating objects: 985, done.
remote: Total 985 (delta 0), reused 0 (delta 0), pack-reused 985
Receiving objects: 100% (985/985), 1002.01 KiB | 813.00 KiB/s, done.
Resolving deltas: 100% (350/350), done.
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3 cd example-voting-app
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico3/example-voting-app sudo docker-compose -f docker-compose-javaworker.yml up -d
[sudo] password for vietto:
Creating network "example-voting-app_front-tier" with the default driver
Creating network "example-voting-app_back-tier" with the default driver
Creating volume "example-voting-app_db-data" with default driver
Building vote
Sending build context to Docker daemon 161.3kB
Step 1/8 : FROM python:3.9-slim
3.9-slim: Pulling from library/python
7a6db449b51b: Pull complete
e238bceb2957: Pull complete
b94fc7ac342a: Pull complete
aa1ba22295b5: Pull complete
76b791f9be0a: Pull complete
Digest: sha256:dcf2eafca55558d8b1aa73edd6aa41b7187c5bcb63e533a7b04a0673f81f37fe
Status: Downloaded newer image for python:3.9-slim
--> 5da6ce3c33c6
Step 2/8 : RUN apt-get update && apt-get install -y --no-install-recommends curl && rm -rf /var/lib/apt/lists/*
--> Running in 8746c8720403
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8182 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages [180 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [2596 B]
Fetched 8573 kB in 8s (1054 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libbrotli1 libcurl4 libldap-2.4-2 libnghttp2-14 libpsl5 librtmp1 libsasl2-2
  libsasl2-modules-db libssh2-1
Recommended packages:
```

```
[INFO] org/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] com/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/slf4j/ already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.slf4j/ already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.312 s
[INFO] Finished at: 2022-08-25T18:59:44Z
[INFO] -----
Removing intermediate container 2fac318cb33f
--> ec292c1627c6
```


Utilizando dos navegadores distintos en este caso Firefox y Chrome, realice dos votos diferentes y observamos el resultado:





Análisis del sistema:

Analizando el archivo docker-compose-javaworker.yml, vemos que el mismo cuenta con 5 contenedores, 2 networks, back-tier y front-tier, que permiten levantar el sistema y un volumen externo, db-data, para guardar los datos de la base de datos localmente. Vemos en más detalle cada contenedor:

- Vote: aplicación web escrita en python que permite emitir votos. Esta como vemos hace uso de flask y redis, en donde genera el volumen ./vote:/app, escucha en el puerto 5000 (abierto) y el contenedor hace uso del puerto 80, y usa las dos redes mencionadas.
- Result: aplicación web Node.js que permite visualizar los resultados de la votación en tiempo real. Esta genera el volumen ./result:/app, escucha en el puerto 5001 y en el 5858 (ambos abiertos) para comunicarse con la base de datos, y el contenedor hace uso del puerto 80. Por último, usa las dos redes mencionadas.
- Worker: aplicación escrita en java que mediante redis puede extraer los votos y los almacena en una base de datos de tipo postgres (db). Además vemos que solamente utiliza la red back-tier.
- Redis: es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes pero que opcionalmente puede ser usada como una base de datos durable o persistente. Acá vemos que se utiliza una cola de redis para recolectar los nuevos votos y así el worker pueda extraerlos y enviarlos a la base de datos. Tiene reservado el puerto 6379 tanto para el host como el contenedor y utiliza la red back-tier.
- db: es una base de datos postgres respaldada por un volumen de docker que se utiliza en este caso para almacenar los votos. Vemos que define las variables de entorno POSTGRES_USER: "postgres", estableciendo el usuario, y POSTGRES_PASSWORD: "postgres", definiendo la contraseña de la base de datos. Por otro lado vemos que tiene reservado el puerto 5432 (agregado al archivo), genera el volumen db-data:/var/lib/postgresql/data y utiliza la red back-tier.

Los volúmenes de vote, result y db están asociados al volumen externo db-data. Haciendo Docker ps -a, vemos que el motor de la base de datos db escucha en el puerto 49513.

```

~ / Doc / U / I / TrabajoPractico3 / example-voting-app P master ! sudo docker-compose down
Stopping example-voting-app_result_1 ... done
Stopping db ... done
Stopping redis ... done
Stopping example-voting-app_worker_1 ... done
Stopping example-voting-app_vote_1 ... done
Removing example-voting-app_result_1 ... done
Removing db ... done
Removing redis ... done
Removing example-voting-app_worker_1 ... done
Removing example-voting-app_vote_1 ... done
Removing network example-voting-app_back-tier
Removing network example-voting-app_front-tier

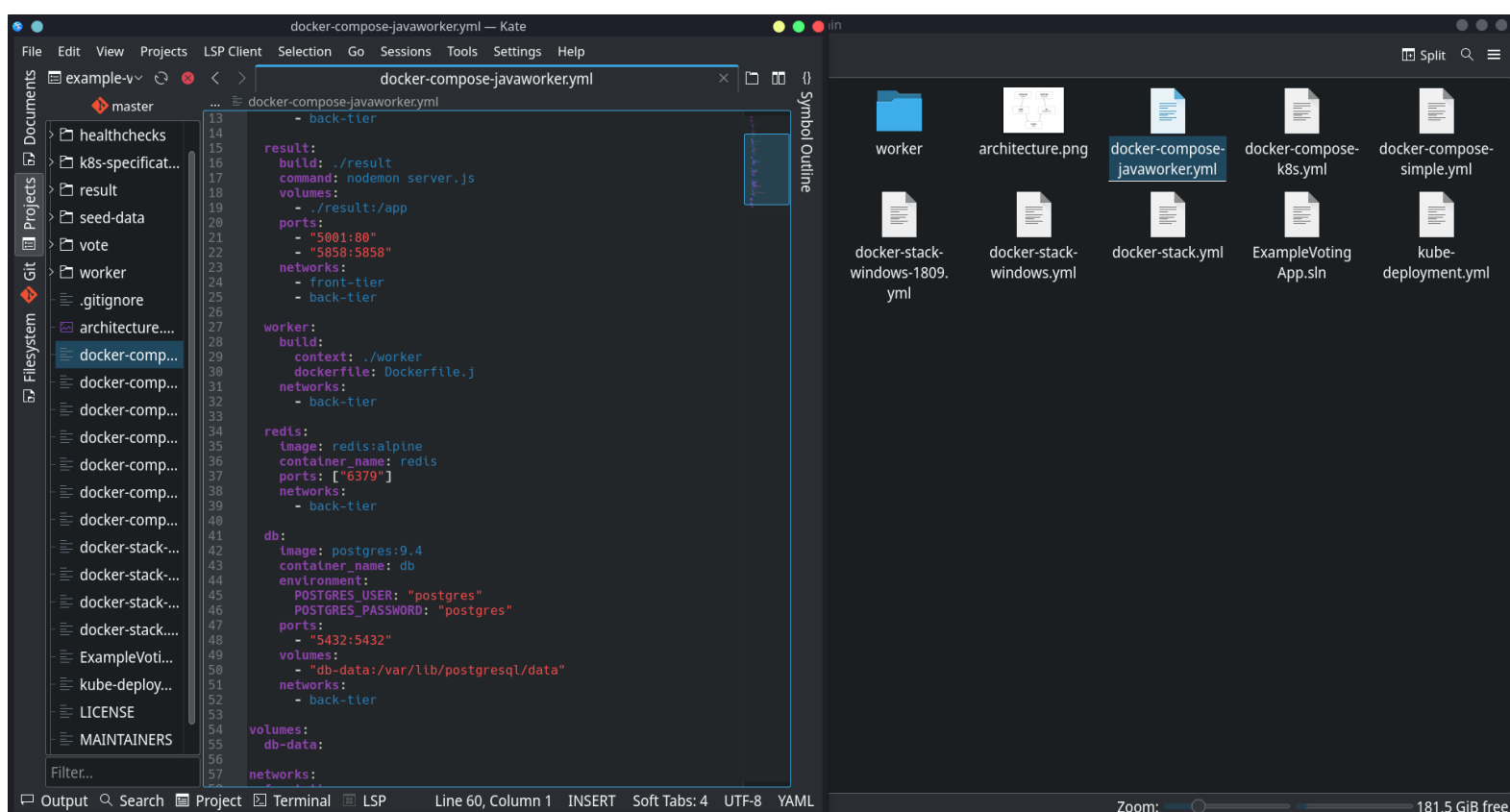
```

Ejercicio 5:

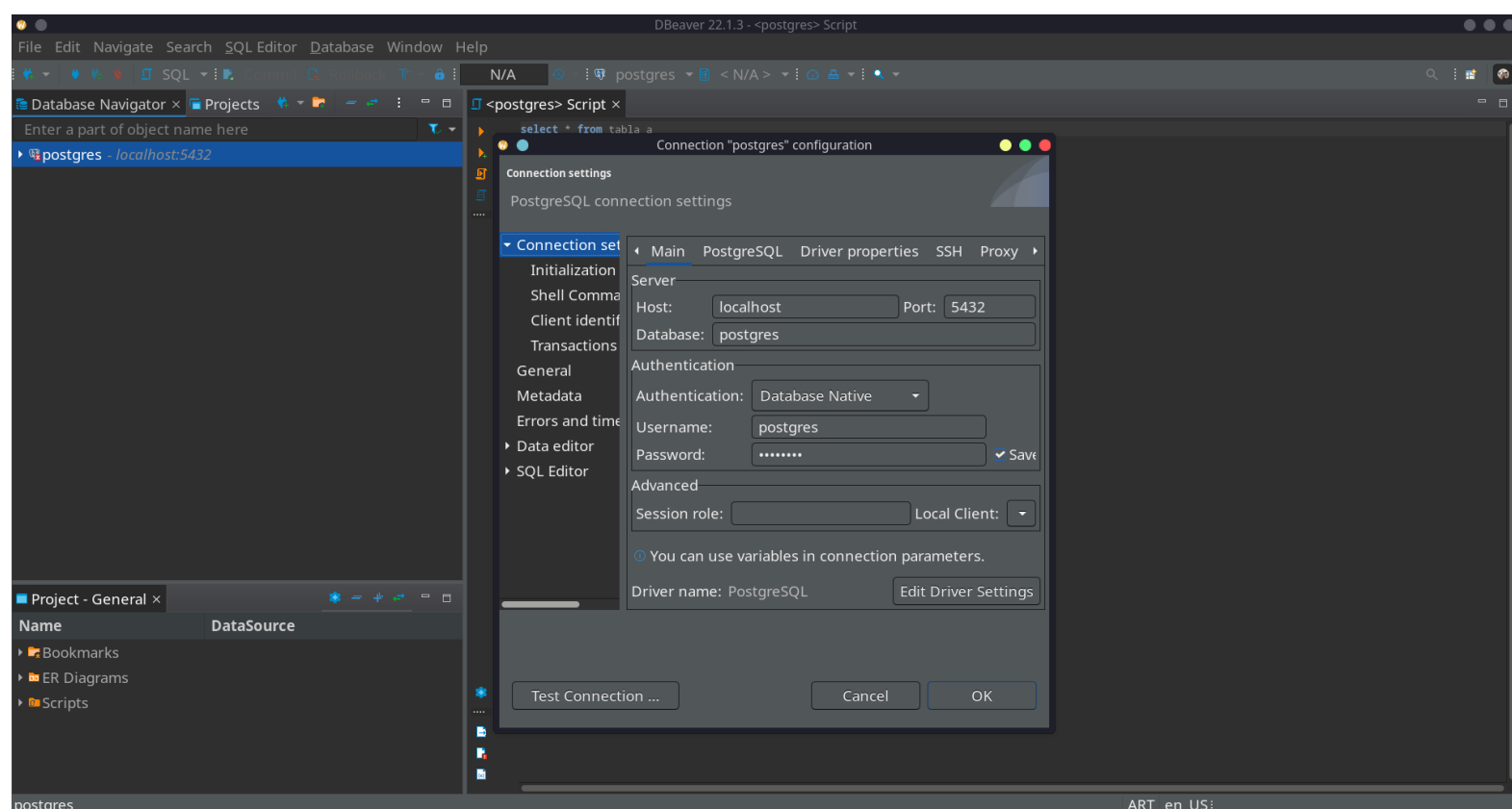
```

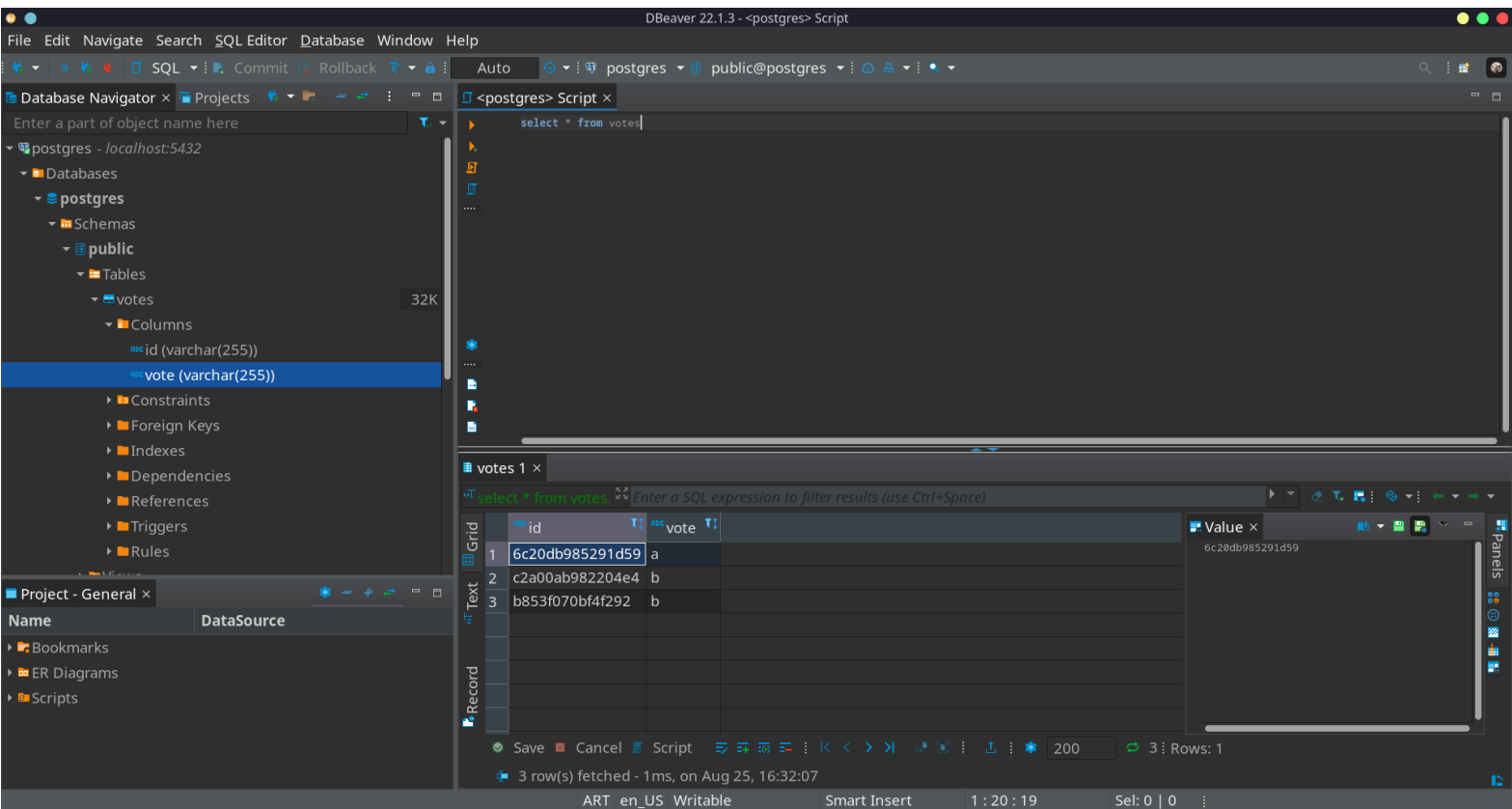
~ / Doc / U / I / TrabajoPractico3 / example-voting-app P master ! sudo docker-compose -f docker-compose-javaworker.yml up -d
Creating network "example-voting-app_front-tier" with the default driver
Creating network "example-voting-app_back-tier" with the default driver
Creating db ... done
Creating example-voting-app_vote_1 ... done
Creating example-voting-app_result_1 ... done
Creating redis ... done
Creating example-voting-app_worker_1 ... done
~ / Doc / U / I / TrabajoPractico3 / example-voting-app P master ! sudo docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
c0184332a390   example-voting-app_worker           "java -XX:+UnlockExp..." 9 seconds ago  Up 8 seconds
e2d2084aab5e   example-voting-app_vote             "python app.py"           9 seconds ago  Up 7 seconds  0.0.0.0:5000->80/tcp, :::5000->80/tcp
c5ae259b5a18   redis:alpine                        "docker-entrypoint.s..." 9 seconds ago  Up 7 seconds  0.0.0.0:49154->6379/tcp, :::49154->6379/tcp
2f09c1bc919   example-voting-app_result           "docker-entrypoint.s..." 9 seconds ago  Up 7 seconds  0.0.0.0:5858->5858/tcp, :::5858->5858/tcp, 0.0.0.0:5001->80/tcp, :::5001->80/tcp
934564fea57f   postgres:9.4                       "docker-entrypoint.s..." 9 seconds ago  Up 7 seconds  0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
8746c8720403   5da6ce3c33c6                       "/bin/sh -c 'apt-get..." 35 minutes ago  Exited (100) 34 minutes ago
7f0cc5ea4e3d   postgres:9.4                       "docker-entrypoint.s..." 7 days ago     Exited (0) 7 days ago
2a4ea39cd6ed   busybox                             "/bin/sh"               7 days ago     Exited (0) 7 days ago

```



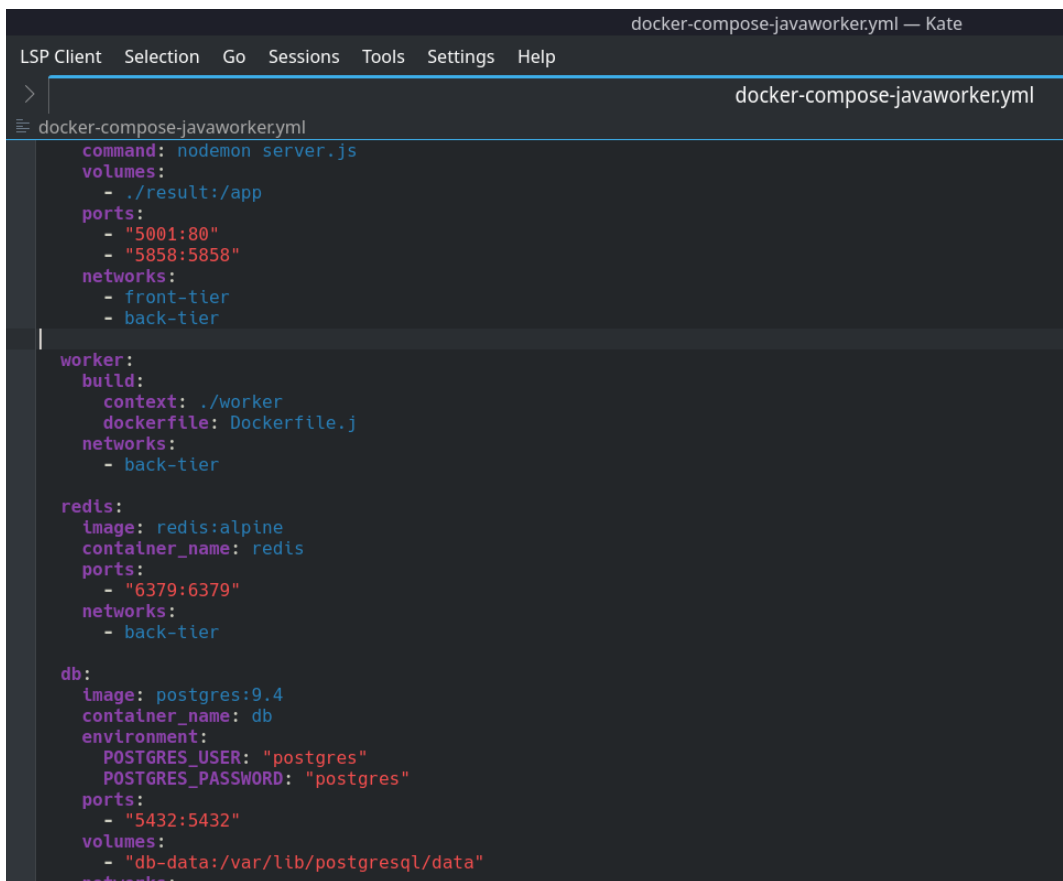
Exponemos las tablas de postgresQL con dbeaver:





Observamos que en la tabla de “vote” la letra “a” representan a cats y las letra “b” representa a dogs, cada uno con su respectivo ID.

Configuración de redis: en el archivo yaml seteamos el puerto 6379 como vemos a continuación:



Luego lo que hacemos es detener el proceso del worker ya que este está quitando permanentemente los votos del redis, por lo que si no lo hacemos, el cliente de redis que utilizamos, RESP.app, no muestra nada, entonces por eso lo frenamos, luego realizamos un voto (votamos cats, por eso el tipo de voto que vemos es “a”) y visualizamos el contenido en un formato Json.

```
~/Doc/U/I/TrabajoPractico3/example-voting-app master !1 sudo docker stop example-voting-app_worker_1 example-voting-app_worker_1
```

RESP.app - GUI for Redis® 2022.4.2+af3aee8

Connect To Redis Server

redis

- db0 (1)
- votes
- db1 (0)
- db2 (0)
- db3 (0)
- db4 (0)
- db5 (0)
- db6 (0)
- db7 (0)
- db8 (0)
- db9 (0)
- db10 (0)
- db11 (0)
- db12 (0)
- db13 (0)
- db14 (0)
- db15 (0)

LIST: votes

#	value	TTL
1	{\"voter_id\": \"1b18512ab129f4f\", \"vote\": \"a\"}	1

Order of elements: Default

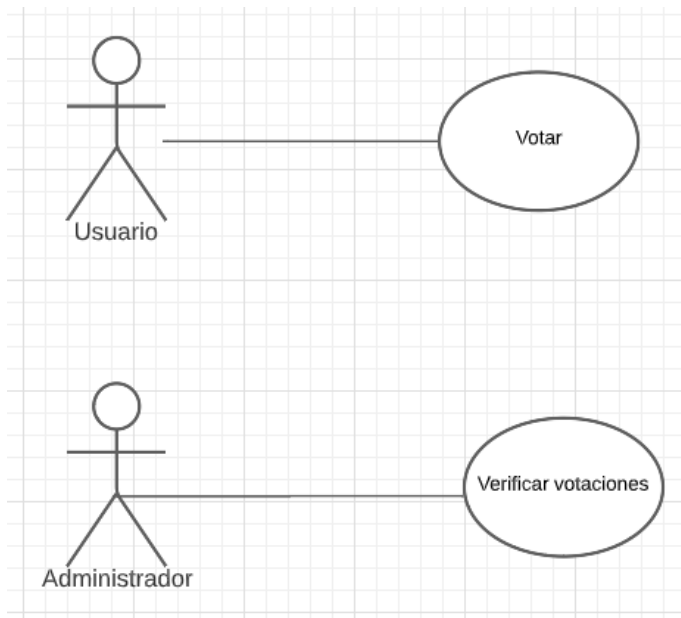
Value: Size: 0 bytes

View as: Plain Text

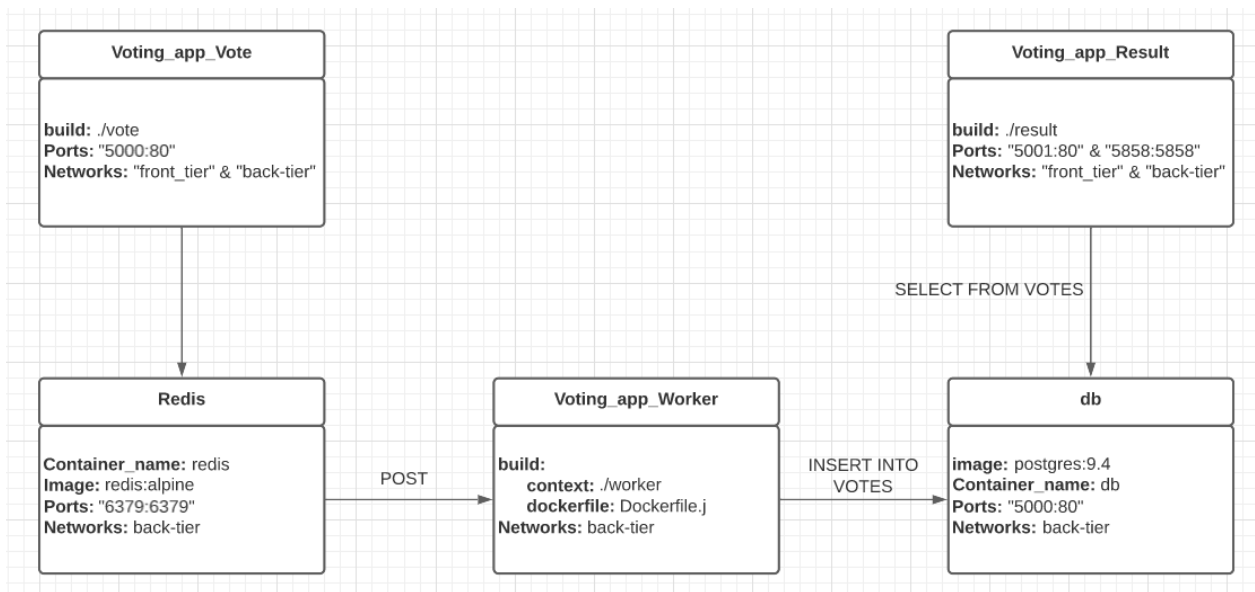
Log Extension Server Settings

Add Row Delete Row Reload Value Search on page... Full Search Page: 1 Size: 1

Diagrama de casos de uso:



Arquitectura:



Explicando el diagrama tenemos que:

- **Vote**: el usuario ingresa a la vista localhost:5000 y procede a realizar un voto seleccionando alguna de las opciones (cats o dogs). Al hacer click en una de las opciones se define un voter_id que guarda un registro de quien voto y cuál opción eligió, en donde posteriormente se hace un POST con la información del voto.
- **Redis**: hace de intermediario entre la webapp de vote y el worker.
- **Worker**: va a procesar la información que luego se va a almacenar en la base de datos. Este se conecta a redis y a db, verificando si la conexión de ambas está activa. En el caso de que ambas conexiones estén funcionando, actualiza la base de datos con un insert del id y el voto.
- **db**: como dijimos antes es una base de datos de tipo Postgres la cual en este caso lo que hace es recibir la información por parte del worker y la añade a una tabla de la misma base de datos. Por otro lado recibe las sentencias SELECT por parte de Results para mostrar la información.
- **Results**: esto hace referencia a la webapp Results en la vista localhost:5001 en donde se visualizan los resultados de la votación. Haciendo analogía con el diagrama de casos de uso, esto sería a lo que accedería el Administrador. Esta app, mediante una función, realiza un SELECT a la tabla "votes" y así trae todos los votos agrupados en el caso de ser cats o dogs, y la cantidad total de tuplas, es decir, la cantidad de votos (lo vemos en DBeaver). Luego, en el navegador se nos muestra los resultados de las votaciones con sus porcentajes y cantidad de votos.