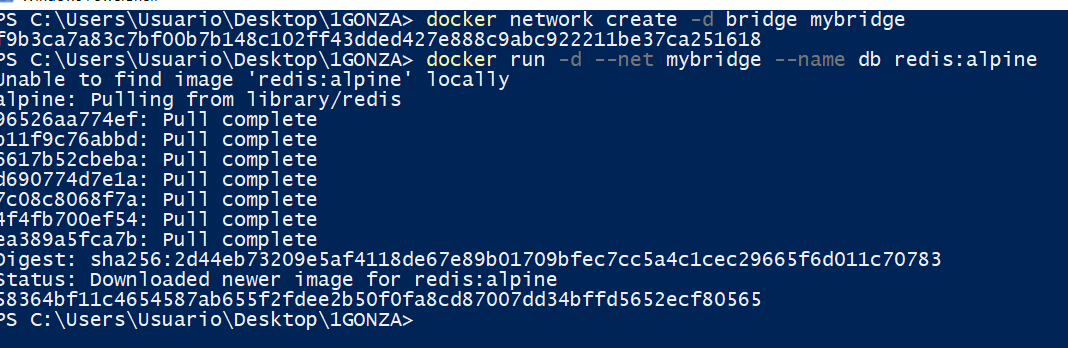
**[1- Sistema distribuido simple](https://github.com/ingsoft3ucc/TPs/blob/master/trabajos/03-arquitectura-sistemas-distribuidos.md" \l "1--sistema-distribuido-simple)**

* Ejecutar el siguiente comando para crear una red en docker

docker network create -d bridge mybridge

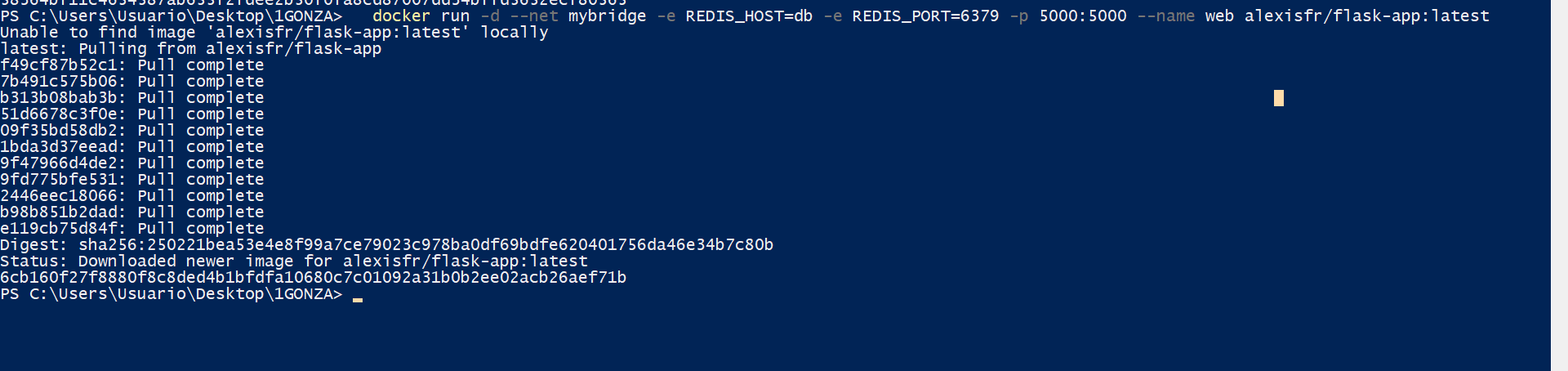
* Instanciar una base de datos Redis conectada a esa Red.

docker run -d --net mybridge --name db redis:alpine



* Levantar una aplicacion web, que utilice esta base de datos

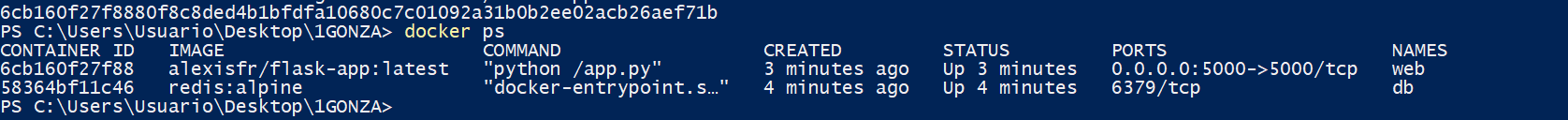
docker run -d --net mybridge -e REDIS\_HOST=db -e REDIS\_PORT=6379 -p 5000:5000 --name web alexisfr/flask-app:latest



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

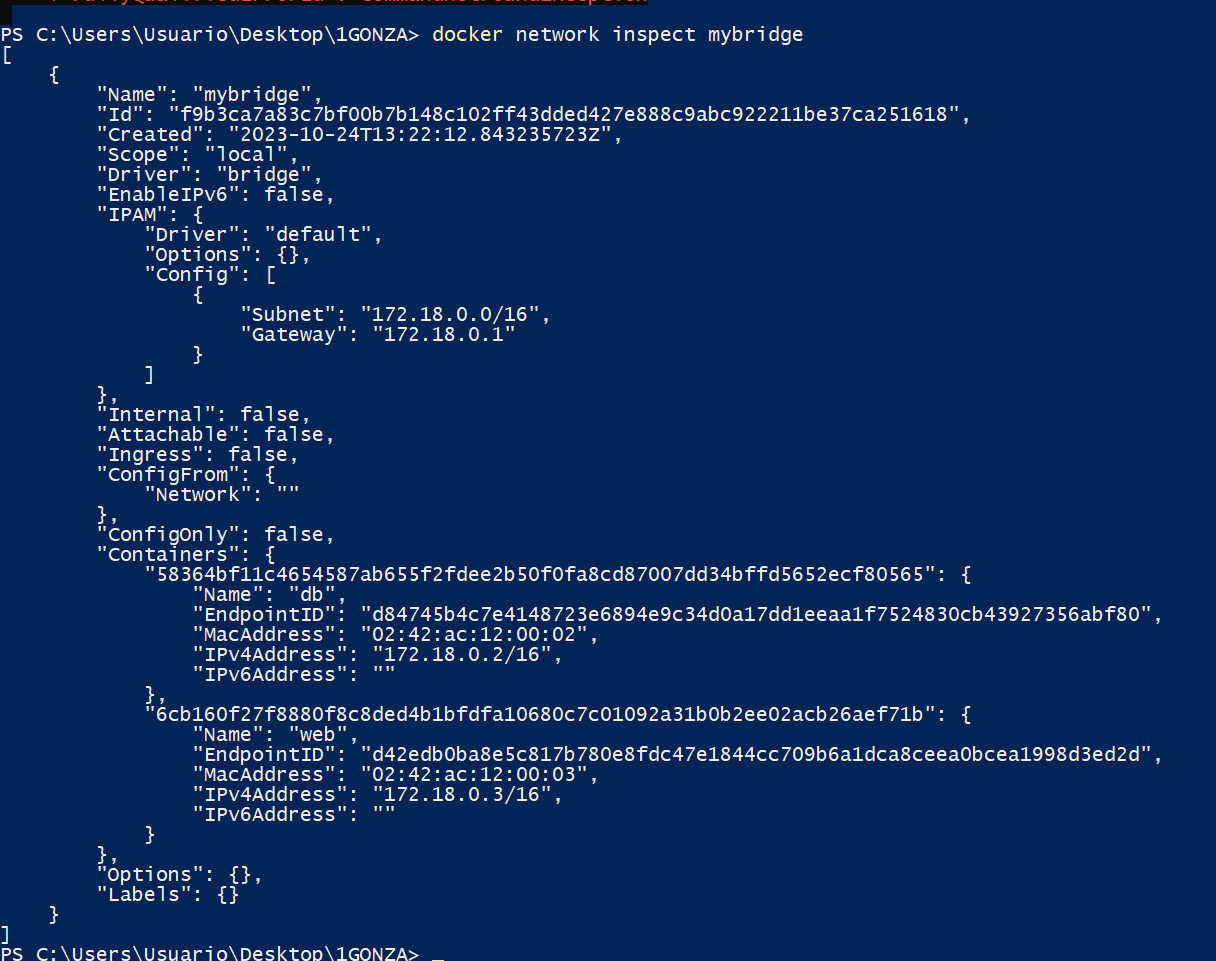


* Verificar el estado de los contenedores y redes en Docker, describir:
  + ¿Cuáles puertos están abiertos?



Los puertos abiertos son el 5000 y el 6379

* + Mostrar detalles de la red mybridge con Docker.



* + ¿Qué comandos utilizó?
* Docker ps
* Docker network inspect “nombre”

[**2- Análisis del sistema**](https://github.com/ingsoft3ucc/TPs/blob/master/trabajos/03-arquitectura-sistemas-distribuidos.md#2--an%C3%A1lisis-del-sistema)

* Siendo el código de la aplicación web el siguiente:

import os #Importa la biblioteca **os** para acceder a las variables de entorno y otras funcionalidades relacionadas con el sistema operativo

from flask import Flask #Importa la clase **Flask** del framework Flask para crear la aplicación web

from redis import Redis #Importa la clase **Redis** para interactuar con la base de datos Redis

app = Flask(\_\_name\_\_) # Crea una instancia de la aplicación Flask. Esto inicializa la aplicación web.

redis = Redis(host=os.environ['REDIS\_HOST'], port=os.environ['REDIS\_PORT']) #Crea una instancia de la clase **Redis** para interactuar con Redis. Los valores de host y puerto se obtienen de las variables de entorno **REDIS\_HOST** y **REDIS\_PORT**.

bind\_port = int(os.environ['BIND\_PORT'])

@app.route('/') # Define una ruta en la aplicación Flask. En este caso, la ruta raíz ("/") activará la función asociada cuando se acceda a esa URL.

def hello(): # Define una función llamada **hello** que se ejecutará cuando alguien acceda a la ruta raíz.

redis.incr('hits') # Incrementa el valor de la clave 'hits' en la base de datos Redis. Esta clave se utiliza para contar cuántas veces se ha accedido a la página.

total\_hits = redis.get('hits').decode() # Obtiene el valor actual de la clave 'hits' y lo decodifica. Esto representa el número de veces que la página ha sido visitada.

return f'Hello from Redis! I have been seen {total\_hits} times.' # Devuelve una respuesta al cliente que muestra un mensaje junto con el número de visitas.

if \_\_name\_\_ == "\_\_main\_\_": # Es una condición que verifica si este script se está ejecutando directamente como un programa independiente.

app.run(host="0.0.0.0", debug=True, port=bind\_port) # Inicia el servidor web Flask para escuchar en todas las direcciones (host="0.0.0.0"), habilita el modo de depuración (debug=True) y utiliza el puerto especificado en la variable **bind\_port**.

* Explicar cómo funciona el sistema

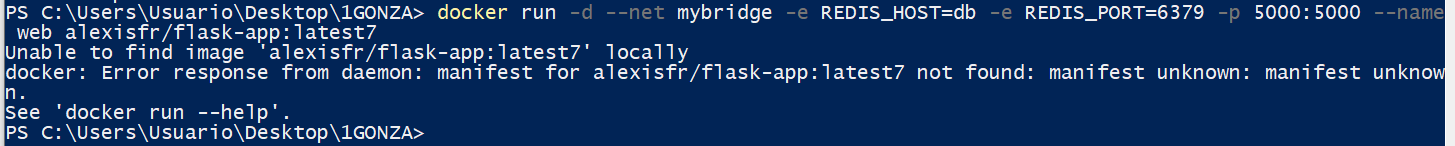
Este sistema es un ejemplo simple de una aplicación web que utiliza el framework Flask para crear un servicio web básico que interactúa con una base de datos Redis. Aquí tienes una explicación paso a paso de cómo funciona:

cuando ejecutas esta aplicación, Flask inicia un servidor web que escucha en un puerto específico. Cuando alguien accede a la ruta raíz, la función hello se ejecuta, incrementa el contador de visitas en la base de datos Redis y muestra un mensaje que incluye el número de visitas. La aplicación web y Redis interactúan a través de variables de entorno para establecer la conexión y compartir datos.

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

Cuando volvamos a ejecutar docker run, estaremos creando un nuevo contenedor llamado "web" desde cero. Este nuevo contenedor no tiene relación con el contenedor anterior. Es como crear una nueva máquina virtual independiente con la misma configuración.

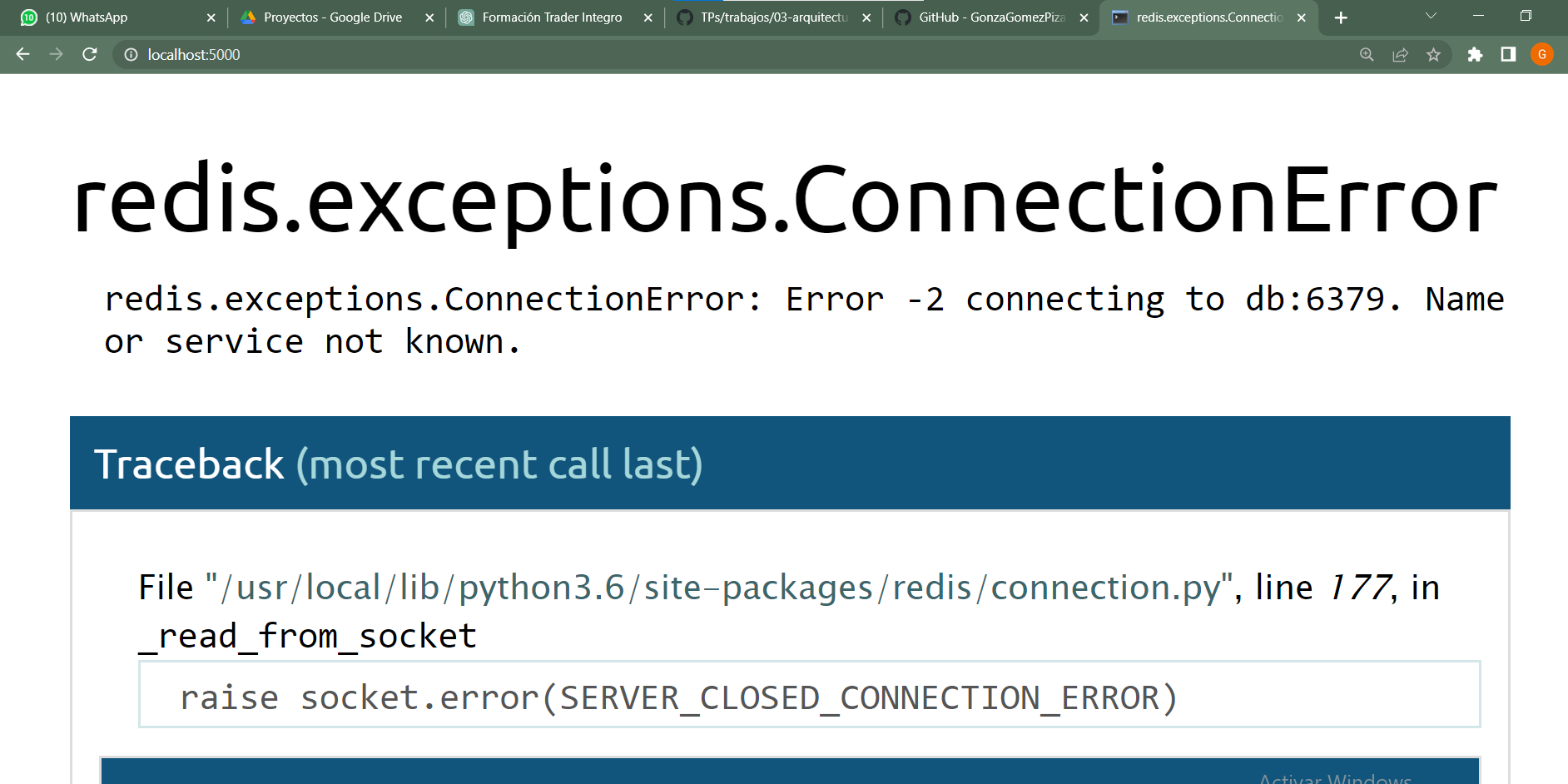




Auquen en realidad me da error

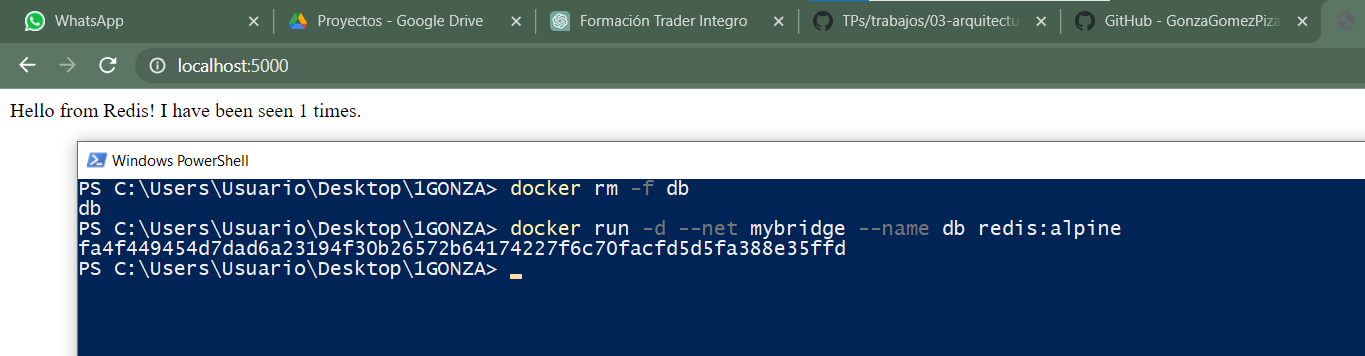
No se por que

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



Anda pero no se conecta a la base de datos

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



Todo ok

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

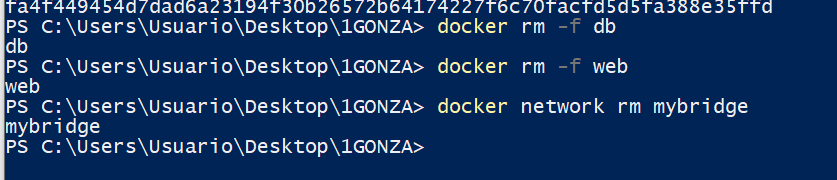
Definir un volumen para que se guarden los datos ¿??

* Para eliminar los elementos creados corremos:

docker rm -f db

docker rm -f web

docker network rm mybridge



[**3- Utilizando docker compose**](https://github.com/ingsoft3ucc/TPs/blob/master/trabajos/03-arquitectura-sistemas-distribuidos.md#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:

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

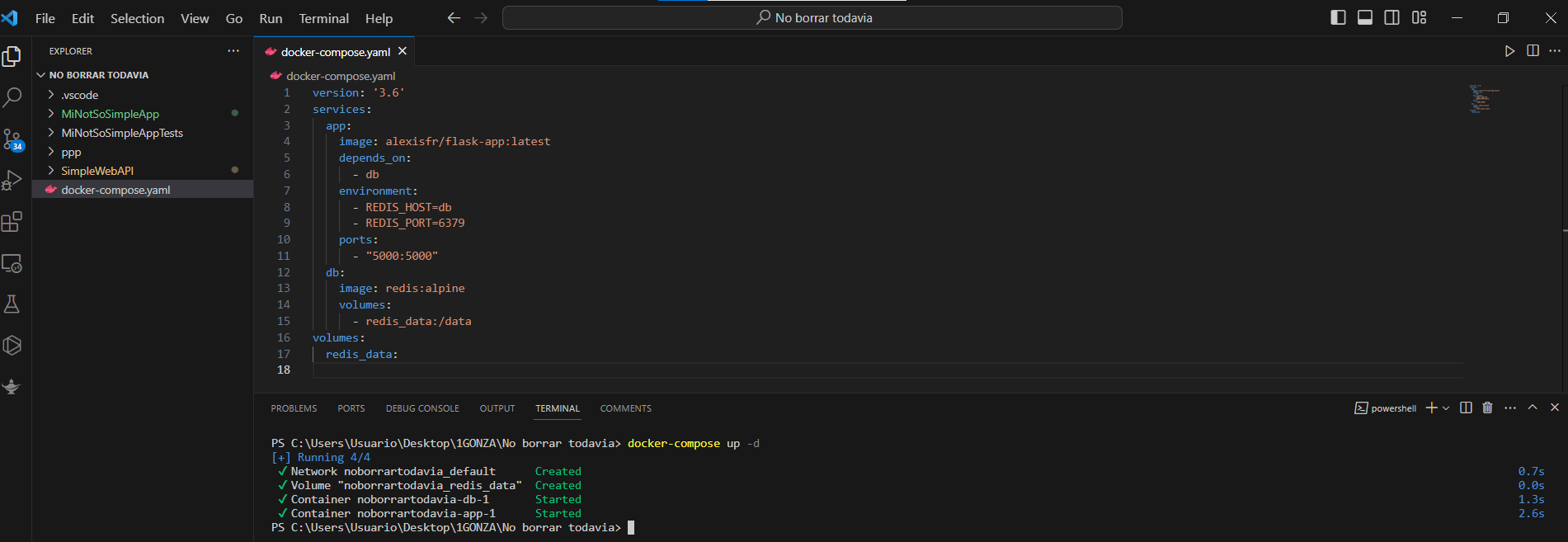
volumes:

- redis\_data:/data

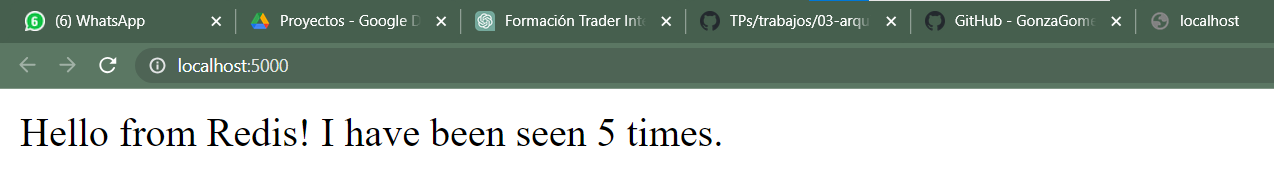
volumes:

redis\_data:

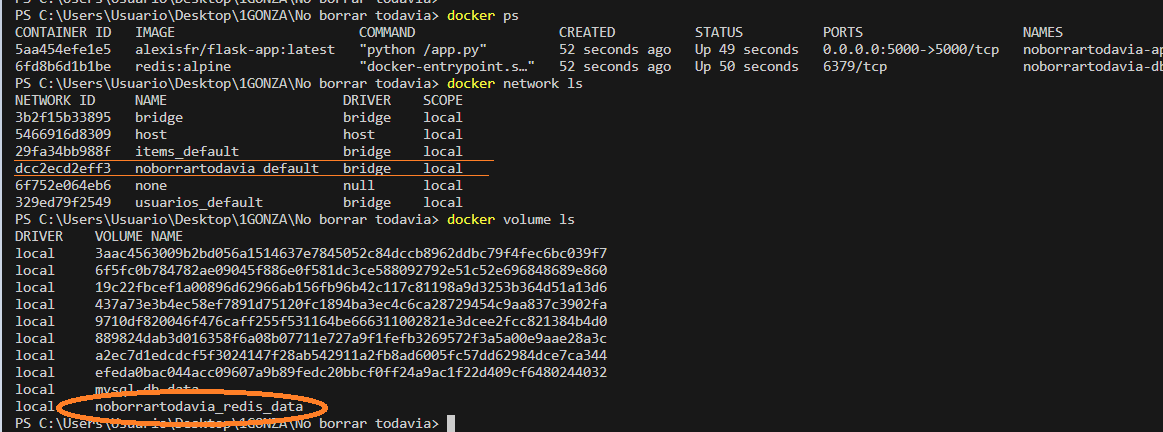
* Ejecutar docker-compose up –d



* Acceder a la url <http://localhost:5000/>



* Ejecutar docker ps, docker network ls y docker volume ls



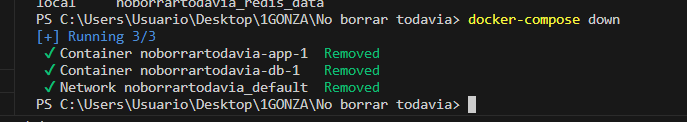
* ¿Qué hizo **Docker Compose** por nosotros? Explicar con detalle.

Docker compose se encargo de crear los contenedores usasndo las imágenes correspondientes, crear la red y crear el volumen para la persistencia de datosss

En Detalle… Chat GPT nos explica mejor:  
  
Este archivo es un archivo de configuración de Docker Compose, utilizado para definir y ejecutar múltiples contenedores como una aplicación única. En este caso, el archivo define dos servicios: "app" y "db", junto con sus configuraciones.

* El servicio "app":
  + Utiliza la imagen **alexisfr/flask-app:latest**, que contiene una aplicación Flask.
  + Depende del servicio "db", lo que significa que Docker Compose asegurará que el servicio "db" esté en funcionamiento antes de iniciar el servicio "app".
  + Configura variables de entorno que la aplicación Flask utilizará para conectarse a la base de datos Redis. Estas variables son **REDIS\_HOST** y **REDIS\_PORT**.
  + Mapea el puerto 5000 del contenedor al puerto 5000 del sistema host, permitiendo el acceso a la aplicación Flask a través del puerto 5000 en el host.
* El servicio "db":
  + Utiliza la imagen **redis:alpine**, que contiene una instancia de Redis.
  + Define un volumen llamado "redis\_data" que se utiliza para almacenar los datos de Redis. Esto asegura que los datos de Redis sean persistentes incluso si el contenedor se detiene o se elimina.
* Desde el directorio donde se encuentra el archivo docker-compose.yaml ejecutar:

docker-compose down



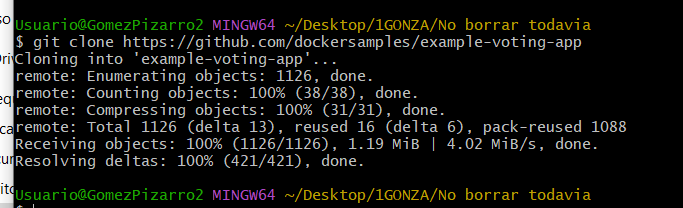
[**4- Aumentando la complejidad, análisis de otro sistema distribuido.**](https://github.com/ingsoft3ucc/TPs/blob/master/trabajos/03-arquitectura-sistemas-distribuidos.md#4--aumentando-la-complejidad-an%C3%A1lisis-de-otro-sistema-distribuido)

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 resultados de la votación en tiempo real.

Pasos:

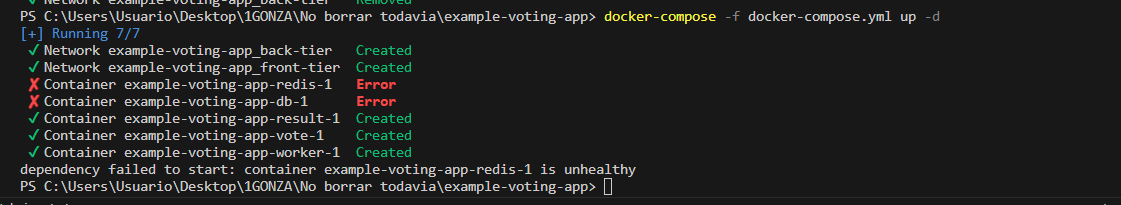
* Clonar el repositorio <https://github.com/dockersamples/example-voting-app>



* Abrir una línea de comandos y ejecutar

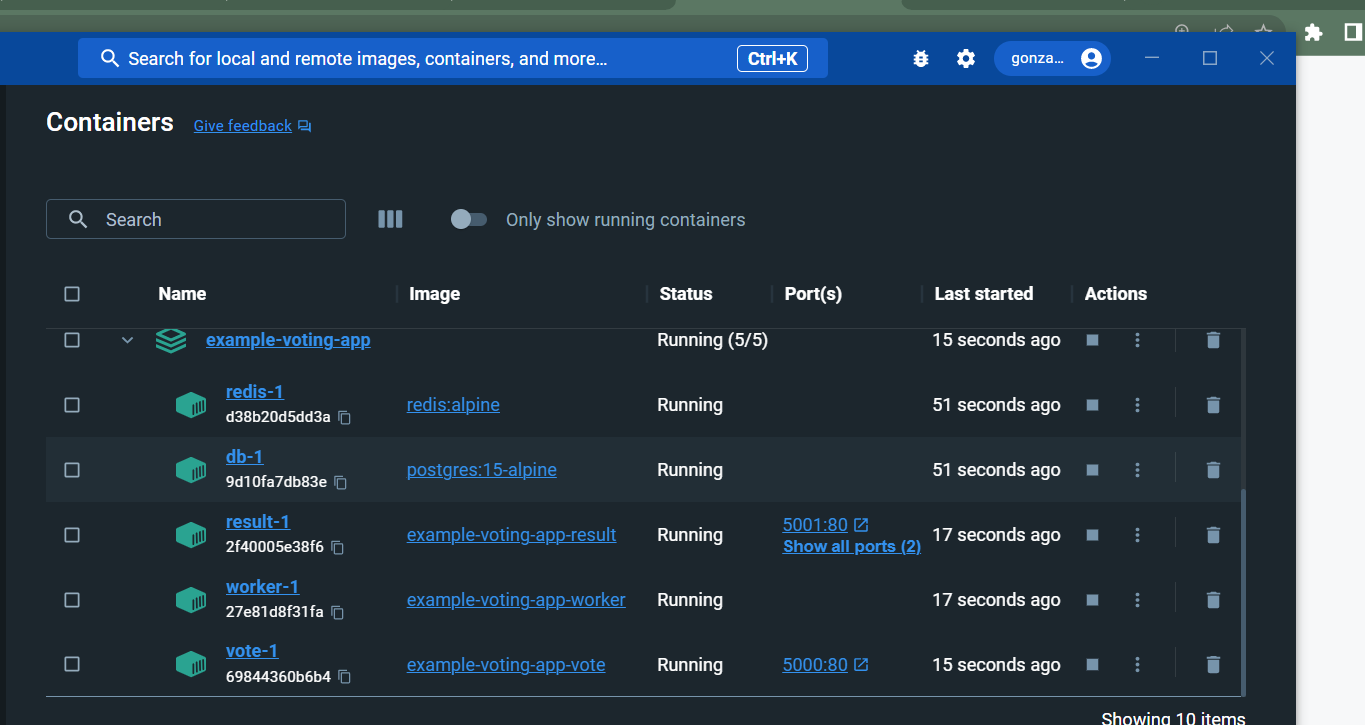
cd example-voting-app

docker-compose -f docker-compose.yml up –d

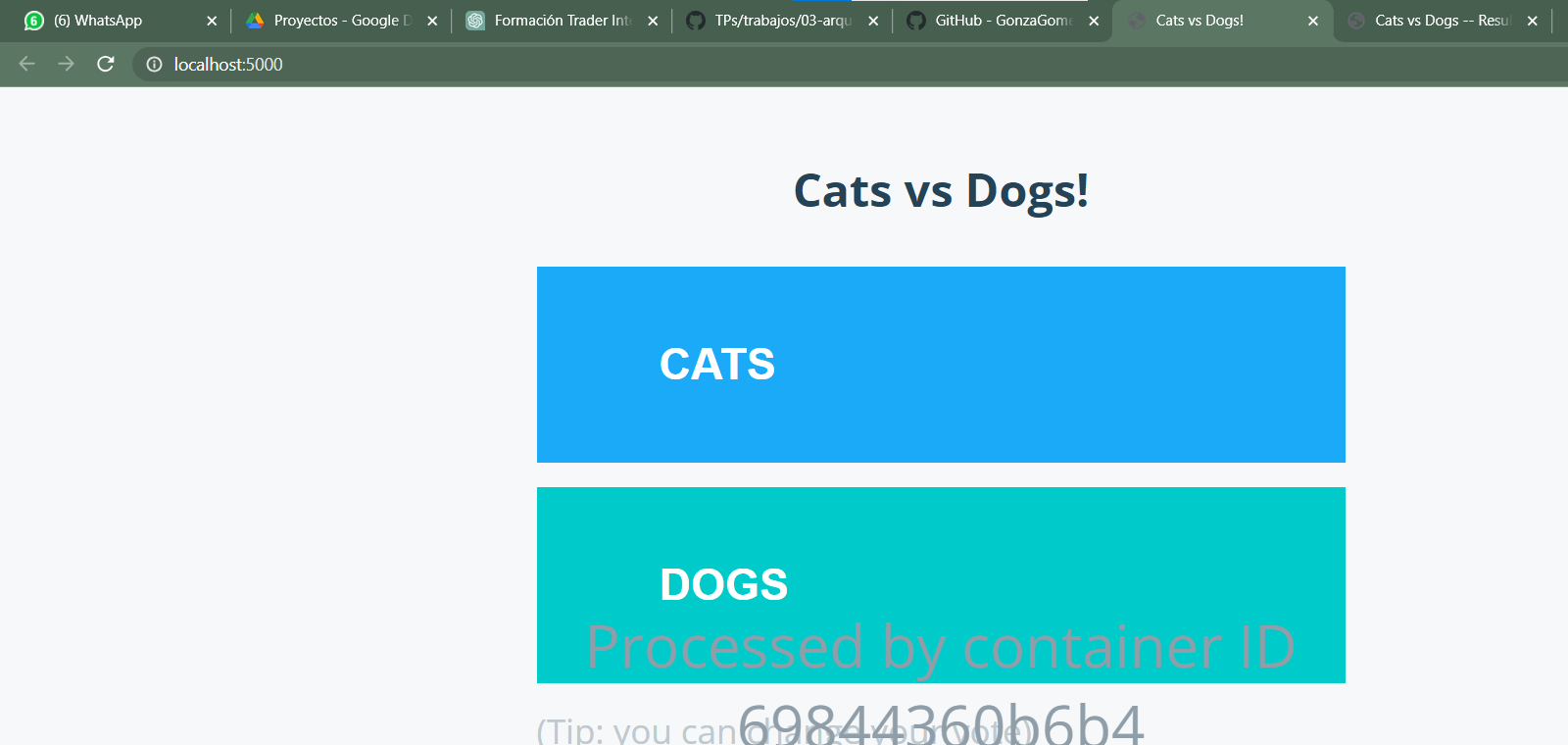


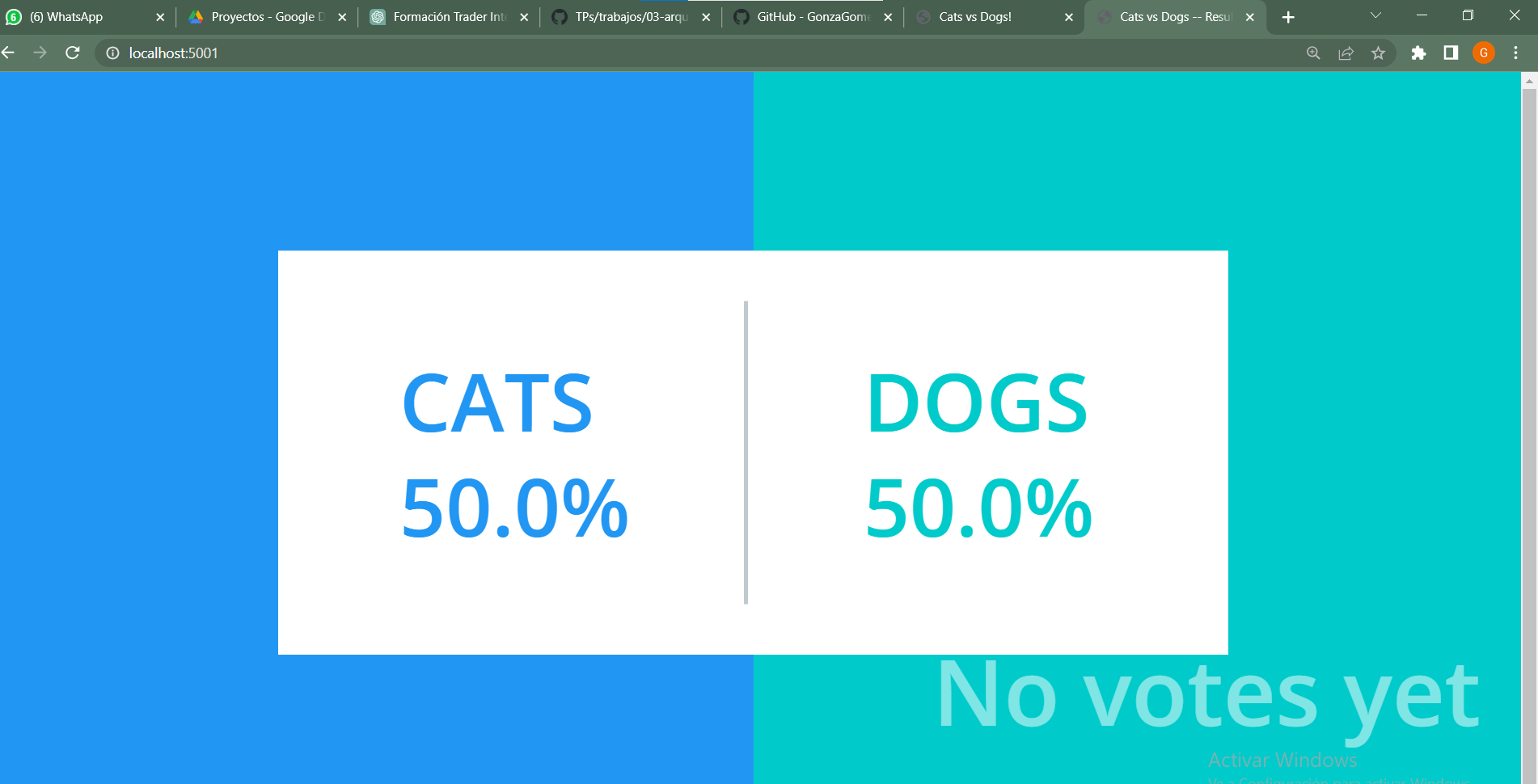
=( ¿¿¿???

Lo pudimos correr desde docker =)

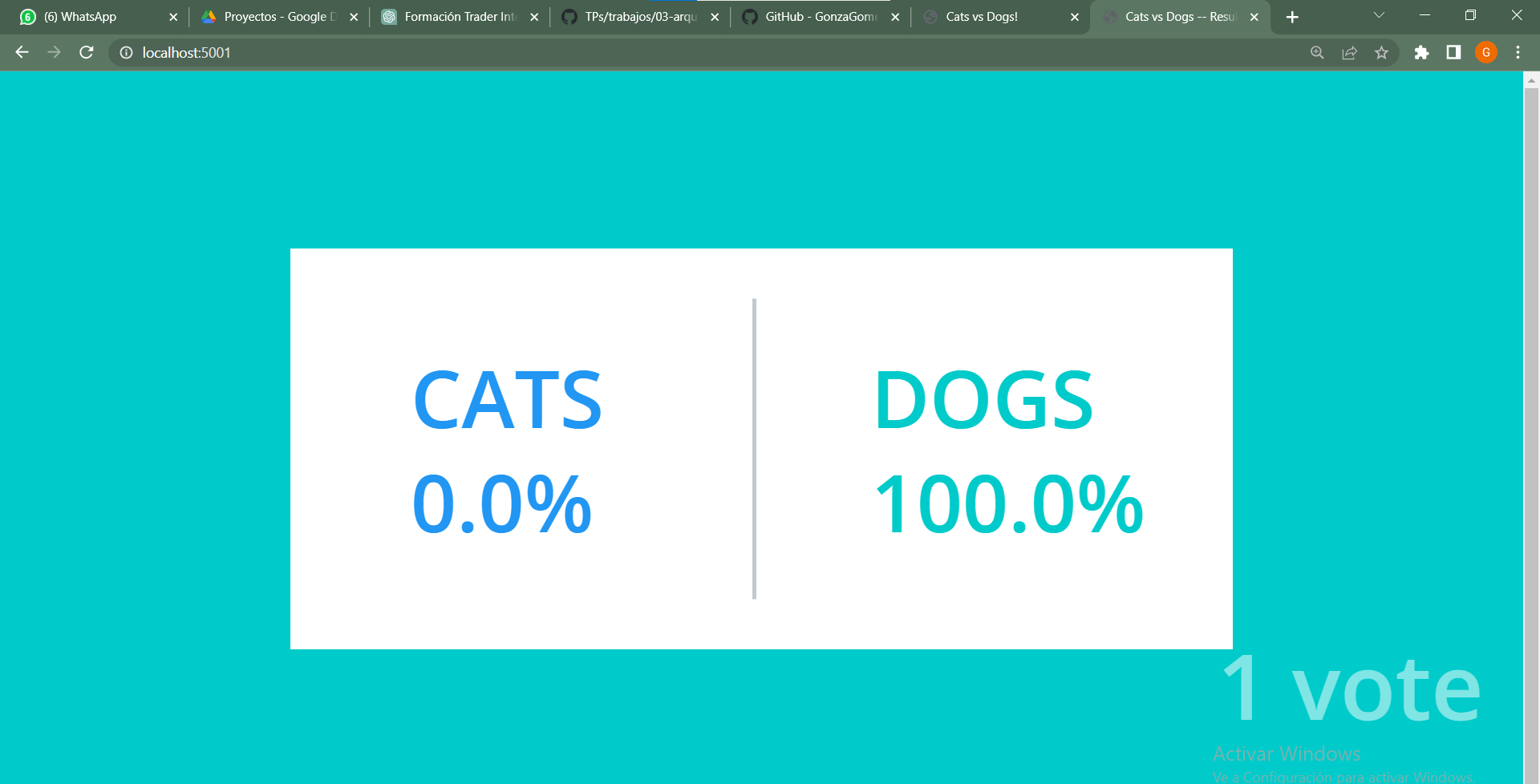


* Una vez terminado acceder a <http://localhost:5000/> y [http://localhost:5001](http://localhost:5001/)





* 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 componenetes involucrados, utilizar el Docker compose como guía.

**Servicio "vote":**

* Es una aplicación construida a partir del directorio local "./vote".
* Se ejecuta en el puerto 80 en el contenedor, que se mapea al puerto 5000 del sistema host.
* Depende del servicio "redis" y verifica su salud con una prueba de "curl".

**Servicio "result":**

* Es una aplicación construida a partir del directorio local "./result".
* Se ejecuta en el puerto 80 en el contenedor, que se mapea al puerto 5001 del sistema host.
* También se ejecuta en el puerto 9229 en el contenedor, que se mapea al puerto 9229 del sistema host.
* Depende de los servicios "db" y "redis" y verifica su salud con comprobaciones personalizadas.

**Servicio "worker":**

* Se construye a partir del directorio local "./worker".
* Se utiliza para tareas en segundo plano y se conecta a la red "back-tier".

**Servicio "redis":**

* Utiliza la imagen de Redis y se conecta a la red "back-tier".
* Realiza comprobaciones de salud utilizando un script personalizado.

**Servicio "db":**

* Utiliza la imagen de PostgreSQL y se conecta a la red "back-tier".
* Configura variables de entorno para PostgreSQL y realiza comprobaciones de salud utilizando un script personalizado.
* Almacena datos de PostgreSQL en un volumen llamado "db-data".

**Servicio "seed":**

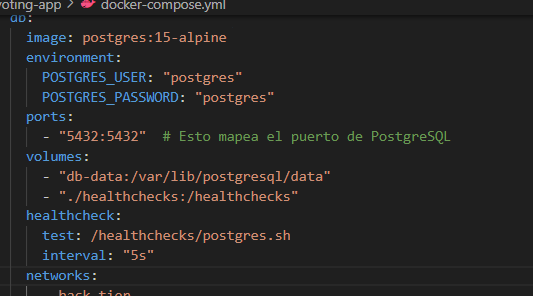
* Se utiliza para cargar datos en la base de datos y se ejecuta una sola vez (usando el perfil "seed").

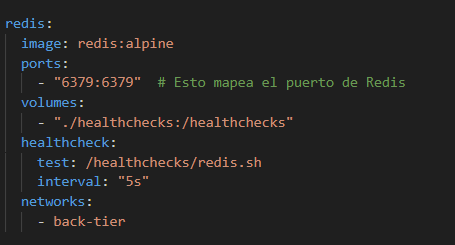
Este archivo también define dos redes: "front-tier" y "back-tier" para permitir la comunicación entre los servicios.

En resumen, el archivo de Docker Compose configura una aplicación distribuida compuesta por múltiples servicios que se comunican entre sí y utilizan comprobaciones de salud personalizadas. También se asegura de que los datos de PostgreSQL sean persistentes utilizando un volumen llamado "db-data".

[**5- Análisis detallado**](https://github.com/ingsoft3ucc/TPs/blob/master/trabajos/03-arquitectura-sistemas-distribuidos.md#5--an%C3%A1lisis-detallado)

* Exponer más puertos para ver la configuración de Redis, y las tablas de PostgreSQL con alguna IDE como dbeaver.





Con esas modificaciones mapeamos los puertos

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

1. Importa los módulos necesarios.
2. Define las opciones de votación y obtiene los valores de las variables de entorno (por defecto, "Cats" y "Dogs").
3. Obtiene el nombre del host de la máquina.
4. Crea una instancia de la aplicación Flask.
5. Configura el registro de eventos y errores de la aplicación.
6. Define una función para obtener una conexión Redis.
7. Define una ruta principal ("/") que maneja solicitudes GET y POST:
   * En solicitudes GET, verifica si el usuario tiene un "voter\_id" almacenado en una cookie. Si no, genera uno nuevo y lo almacena en una cookie en la respuesta.
   * En solicitudes POST (cuando el usuario emite un voto), registra el voto y el "voter\_id" en un servidor Redis y muestra el resultado en una plantilla HTML.
8. La aplicación se ejecuta directamente y escucha en todas las interfaces en el puerto 80.

* Revisar el código del worker example-voting-app\worker\program.cs para entender como procesa los datos.

1. La aplicación se conecta a la base de datos PostgreSQL (Postgres) y al servidor Redis.
2. Luego, utiliza un bucle infinito para procesar votos continuamente.
3. En cada iteración del bucle: a. Verifica si hay un voto disponible en la cola de Redis. Si hay un voto, lo retira de la cola. b. Procesa el voto, registrando el voto y el votante en la base de datos PostgreSQL.
4. La aplicación se asegura de manejar las reconexiones a la base de datos y al servidor Redis, lo que garantiza que pueda continuar procesando los votos incluso si hay interrupciones en la conexión.

Esta aplicación procesa votos desde una cola en Redis y los almacena en una base de datos PostgreSQL, asegurándose de que la operación sea continua y resistente a fallos en la conexión a ambas bases de datos.

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

El script utiliza el módulo Express.js para configurar un servidor web que atiende a la raíz (**'/'**) y envía un archivo HTML cuando se accede a esa ruta.

1. El servidor WebSocket de Socket.io se inicia para permitir la comunicación en tiempo real con los clientes conectados.
2. Luego, el script establece una conexión a la base de datos PostgreSQL. Si no puede conectarse de inmediato, se realizarán varios intentos (hasta 1000 intentos, cada 1 segundo) antes de darse por vencido.
3. Una vez que se establece la conexión a la base de datos, la función **getVotes** se encarga de realizar una consulta SQL que recopila los votos y la cantidad de votos para cada opción ('a' y 'b'). Luego, esta información se envía a través del socket WebSocket a los clientes conectados.
4. La aplicación web sirve una página HTML que incluye una interfaz para mostrar las votaciones en tiempo real. Los datos de votación se actualizan en la página a medida que cambian.
5. El servidor web se inicia y escucha en un puerto determinado (por defecto, el puerto 4000).

En resumen, este script Node.js establece un servidor web y una conexión WebSocket para mostrar en tiempo real los resultados de las votaciones a medida que se almacenan en la base de datos PostgreSQL. Los clientes web pueden conectarse y recibir actualizaciones en tiempo real de los resultados de las votaciones.

* 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.