

1)

```
C:\Users\gusta>docker network create -d bridge mybridge
390142a48a77a8d577e3de448278287ca4f5864fb59a6775b4dd47b92364a6e0

C:\Users\gusta> docker run -d --net mybridge --name db redis:alpine
Unable to find image 'redis:alpine' locally
alpine: Pulling from library/redis
96526aa774ef: Pull complete
6adfacd3b74c: Pull complete
8f2d8ff49f68: Pull complete
473eef84775a: Pull complete
a8fc03039a58: Pull complete
4f4fb700ef54: Pull complete
3c27b2421cc2: Pull complete
Digest: sha256:343e6546f35877801de0b8580274a5e3a8e8464cabe545a2dd9f3c78df77542a
Status: Downloaded newer image for redis:alpine
f9581e313e900741b2d1b4cca048710c26d9a92c8efba789de651c929c51416f
```

db redis:alpine Running 0.17% 32

```
C:\Users\gusta> 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
9fd775bf531: Pull complete
2446eec18066: Pull complete
b98b851b2dad: Pull complete
e119cb75d84f: Pull complete
Digest: sha256:250221bea53e4e8f99a7ce79023c978ba0df69bdf620401756da46e34b7c80b
Status: Downloaded newer image for alexisfr/flask-app:latest
f204089c442e258adb88e5b50ba92701d7f70dbad2e6d73b1e38a9f552c303a8
```

localhost:5000

localhost:5000

Hello from Redis! I have been seen 2 times.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
f204089c442e	alexisfr/flask-app:latest	"python /app.py"	3 minutes ago	Up 3 minutes	0.0.0.0:5000->5000/tcp
p_web					
f9581e313e90	redis:alpine	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes	6379/tcp
db					

Están abiertos los puertos 5000 en el contenedor de la aplicación web y 6379 en el contenedor de la base de datos Redis.

```
C:\Users\gusta>docker network inspect mybridge
[
  {
    "Name": "mybridge",
    "Id": "390142a48a77a8d577e3de448278287ca4f5864fb59a6775b4dd47b92364a6e0",
    "Created": "2023-10-16T05:21:16.177131311Z",
    "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": {
      "f204089c442e258adb88e5b50ba92701d7f70dbad2e6d73b1e38a9f552c303a8": {
        "Name": "web",
        "EndpointID": "add6301e5333574ffdebb7820818b01551366aff7374f18de9ee726e3b92a19d",
        "MacAddress": "02:42:ac:14:00:03",
        "IPv4Address": "172.20.0.3/16",
        "IPv6Address": ""
      },
      "f9581e313e900741b2d1b4cca048710c26d9a92c8efba789de651c929c51416f": {
        "Name": "db",
        "EndpointID": "1d69be10dbf15e5e958859709cbe60eb1dd805be522706e1e215a37a1e02f718",
        "MacAddress": "02:42:ac:14:00:02",
        "IPv4Address": "172.20.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
```

Detalles de la red mybridge:

Información sobre los contenedores que están conectados a la red, subredes, rangos de direcciones IP.

- 2) El sistema consiste en una aplicación web Flask que utiliza una base de datos Redis para mantener un contador de visitas. Se ejecutan en contenedores Docker y se comunican a través de una red Docker llamada "mybridge". Los usuarios pueden acceder a la aplicación web a través de un navegador web y ver cuántas veces ha sido visitada la página.

Los parámetros -e sirven para pasar variables de entorno al contenedor. En este caso, se utilizan para proporcionar la dirección del host de Redis y el puerto de Redis a la aplicación web.

Si ejecuto "docker rm -f web" y vuelvo a correr docker run, se creará una nueva instancia de la aplicación web con un contador hits reiniciado y se pierde el estado anterior del contador.

Cuando borro el contenedor de Redis, si un usuario accede a la página web el contador 'hits' no podrá incrementarse ni recuperarse desde Redis, y mostrará un error.

Si lo levanto nuevamente, Redis se reiniciará con el contador hits en cero. La aplicación web podrá comunicarse con la nueva instancia de Redis, pero los datos anteriores se pierden.

Para no perder la cuenta de las visitas necesito almacenar el estado del contador hits en una ubicación persistente.

```
C:\Users\gusta>docker rm -f db
db

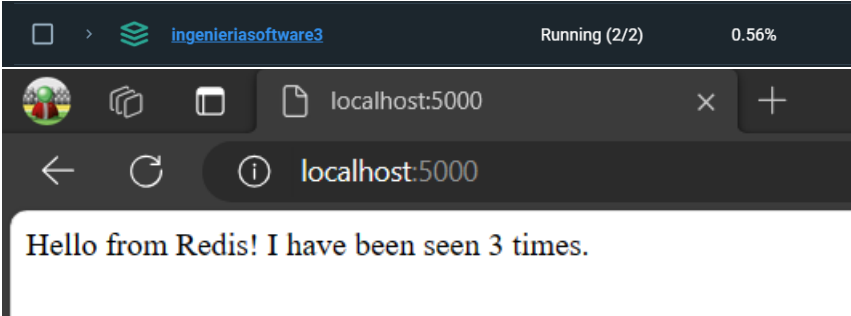
C:\Users\gusta>docker rm -f web
web

C:\Users\gusta>docker network rm mybridge
mybridge
```

3)

```
C:\Users\gusta>docker-compose --version
Docker Compose version v2.20.2-desktop.1
```

```
PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> docker-compose up -d
[+] Running 4/4
  ✓ Network ingenieriasoftware3_default      Created           0.7s
  ✓ Volume "ingenieriasoftware3_redis_data" Created           0.0s
  ✓ Container ingenieriasoftware3-db-1       Started           1.1s
  ✓ Container ingenieriasoftware3-app-1      Started           2.1s
```



```
PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS                               NAMES
25d7a039eb71  alexisfr/flask-app:latest           "python /app.py"        About a minute ago Up About a minute 0.0.0.0:5000->5000/tcp             ingenieriasoftware3-ap
p-1
8002563281e5  redis:alpine                        "docker-entrypoint.s..." About a minute ago Up About a minute 6379/tcp                           ingenieriasoftware3-db
-1

PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> docker network ls
NETWORK ID     NAME                                DRIVER  SCOPE
7f091a2a23d6  api_blockchain_wrapper_default     bridge  local
106e646453f7  bridge                             bridge  local
56efdfefeda2  fabric_debug                       bridge  local
9e2236d77391  host                               host    local
b5897c9809b9  ingenieriasoftware3_default        bridge  local
db1269f125ca  none                               null    local

PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> docker volume ls
DRIVER  VOLUME NAME
local  8e92037d6f82343568abcadd7d8a9ae2e007892adcfa46ae1dd5a31069666fbc
local  43658b8aff48b27493f9b09a04ecd3a7f862065ab38b5173ee52241233ea7dfc
local  33212029bf1ba64b618e9795b781616dbbd34fad0081c4d2e0db04e28a06c660
local  135508751e1d6eeb1a43b58e1928900073d132a4b5d7f9c2872bd161f8cbebf
local  api_blockchain_wrapper_mongo_data
local  b39508530fbf75d7e24a484274887c100e2930ed6b6b7f78a8978746f6f83d5
local  ingenieriasoftware3_redis_data
```

Docker Compose realiza varias acciones que configuran y ejecutan los servicios y contenedores definidos en el archivo.

Creó dos contenedores, uno para la aplicación web y otro para la base de datos Redis.

Configuró los contenedores con las variables de entorno REDIS_HOST y REDIS_PORT en el contenedor app para que pueda comunicarse con la base de datos db.

Definió un volumen redis_data utilizado para persistir los datos de Redis en el contenedor db.

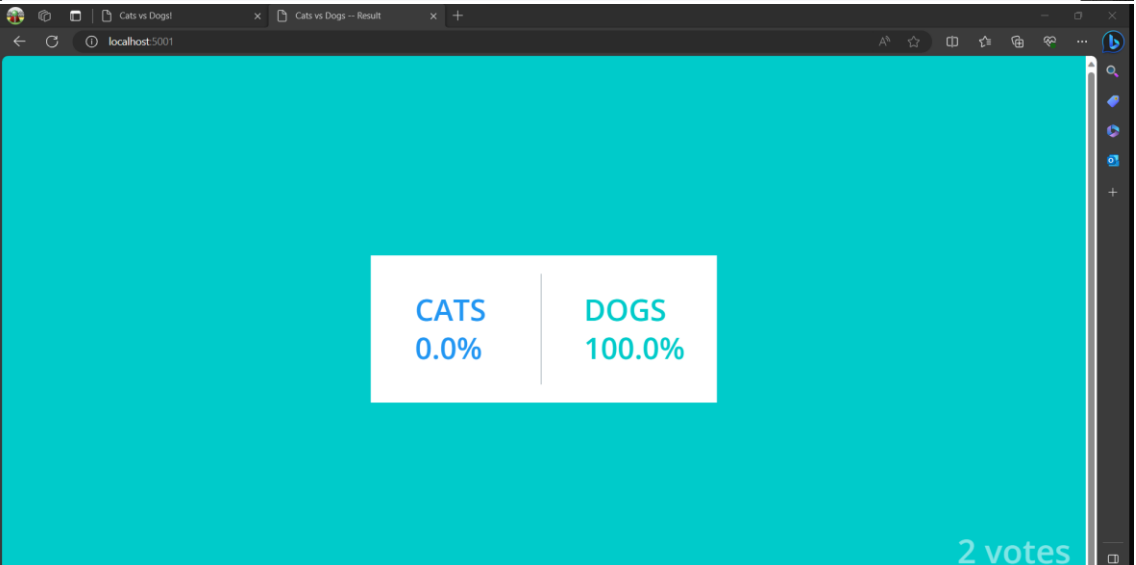
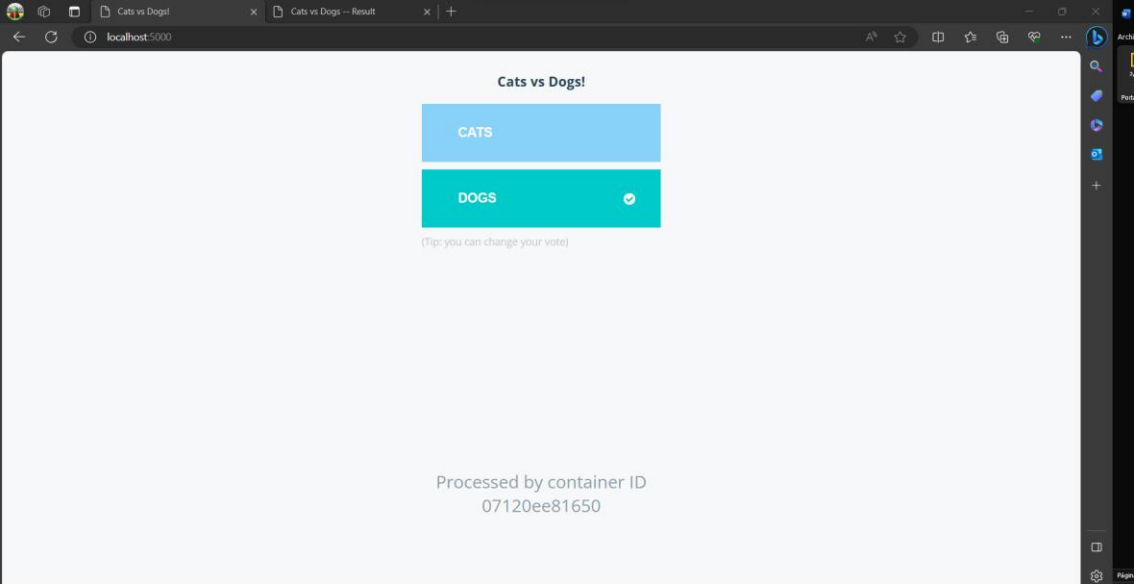
docker-compose up -d hace que los contenedores se ejecuten en segundo plano, permitiendo que los servicios se mantengan en ejecución en el fondo.

```
PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> docker-compose down
[+] Running 3/3
  ✓ Container ingenieriasoftware3-app-1      Removed           0.5s
  ✓ Container ingenieriasoftware3-db-1       Removed           0.4s
  ✓ Network ingenieriasoftware3_default      Removed           0.6s
```

4)

```
PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3> git clone https://github.com/docker-samples/example-voting-app.git
Cloning into 'example-voting-app'...
remote: Enumerating objects: 1099, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 1099 (delta 0), reused 1 (delta 0), pack-reused 1091
Receiving objects: 100% (1099/1099), 1.16 MiB | 4.39 MiB/s, done.
Resolving deltas: 100% (411/411), done.

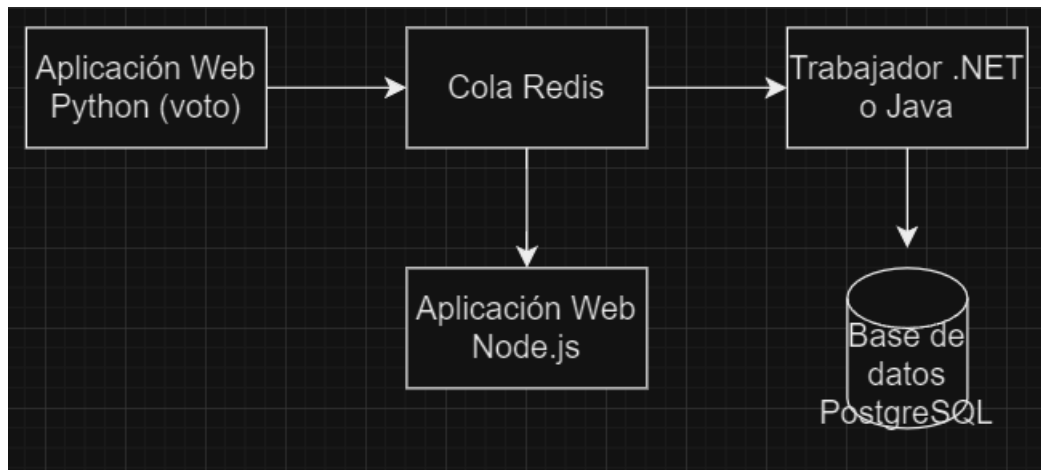
PS C:\Users\gusta\OneDrive\Escritorio\UCC\Cuarto Año\IngenieriaSoftware 3\example-voting-app> docker-compose -f docker-c
ompose.yml up -d
[+] Running 7/7
✓ Network example-voting-app_front-tier Created 0.7s
✓ Network example-voting-app_back-tier Created 0.7s
✓ Container example-voting-app-db-1 Healthy 7.7s
✓ Container example-voting-app-redis-1 Healthy 7.7s
✓ Container example-voting-app-worker-1 Started 9.2s
✓ Container example-voting-app-vote-1 Started 10.8s
✓ Container example-voting-app-result-1 Started 11.8s
```



- Docker Compose:
El sistema utiliza Docker Compose para definir y gestionar los servicios que componen la aplicación. El archivo docker-compose.yml especifica la configuración de los servicios, sus dependencias y cómo se deben comunicar entre sí.
- Aplicación Web de Python (vote):
Esta aplicación permite a los usuarios emitir votos entre dos opciones. Está configurada para escuchar en el puerto 80 y se mapea al puerto 5000 en el host local.

- Cola de Redis:
Redis se utiliza como una cola para recolectar nuevos votos de la aplicación de votación.
- Trabajador .NET o Java:
Consume los votos de la cola de Redis y los almacena en la base de datos de PostgreSQL.
- Base de Datos PostgreSQL:
La base de datos de PostgreSQL almacena los votos y está respaldada por un volumen Docker para persistencia de datos.
- Aplicación Web Node.js:
Esta aplicación muestra los resultados de la votación en tiempo real. Está configurada para escuchar en el puerto 80 y se mapea al puerto 5001 en el host local.
- Volúmenes:
Se utilizan volúmenes Docker para asegurar la persistencia de datos de Redis y PostgreSQL. Esto garantiza que los datos se mantengan incluso después de detener o eliminar los contenedores.
- Comunicación entre Servicios:
Docker Compose se encarga de conectar todos los servicios en una red interna. Esto permite que la aplicación de votación envíe votos a Redis, que luego son consumidos por el trabajador .NET o Java para ser almacenados en PostgreSQL. La aplicación web Node.js muestra los resultados actualizados en tiempo real.
- Escalabilidad:
Se pueden emitir más votos al abrir varios navegadores o instancias de la aplicación de votación. Los componentes del sistema distribuido están diseñados para funcionar juntos y escalar horizontalmente según sea necesario.

5)



Descripción de la Interacción:

Los usuarios acceden a la Aplicación Web de Python (voto) para emitir sus votos por "Cats" o "Dogs". Los votos se envían a través de una API REST a la cola de Redis.

La Cola de Redis almacena los votos entrantes en colas separadas para cada opción de voto. Los trabajadores consumen estos votos y los procesan.

Los Trabajadores (implementados en .NET o Java) se encargan de tomar los votos de la cola de Redis, procesarlos y guardarlos en la Base de Datos PostgreSQL.

La Aplicación Web Node.js obtiene los resultados de la votación desde la base de datos PostgreSQL y muestra los resultados en tiempo real a los usuarios a través de una conexión WebSocket.

Este sistema distribuido permite a los usuarios votar y ver los resultados en tiempo real, con una arquitectura que separa las preocupaciones de la aplicación web, la gestión de votos y la presentación de resultados. La combinación de Redis, PostgreSQL y Node.js permite una experiencia de votación y visualización eficiente y escalable.