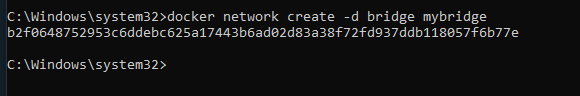
**Arquitectura de Sistemas Distribuidos**

1. **Sistema distribuido simple**

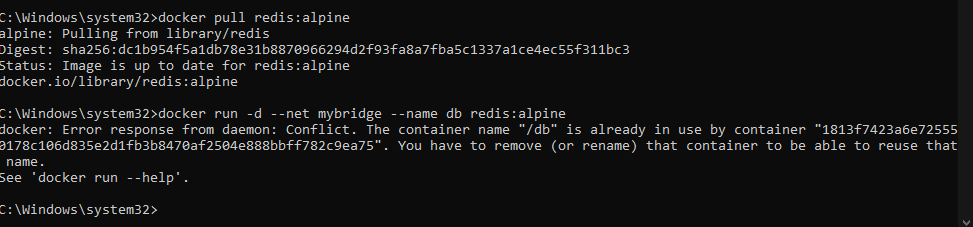
• Ejecutar el siguiente comando para crear una red en docker

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

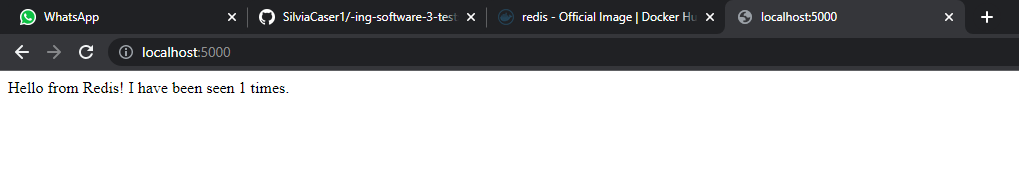


• Levantar una aplicacion web, que utilice esta base de datos

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

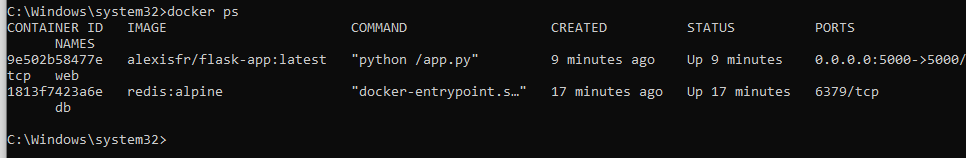


• Abrir un navegador y acceder a la URL: <http://localhost:5000/>



• Verificar el estado de los contenedores y redes en Docker, describir:

o ¿Cuáles puertos están abiertos?



o Mostrar detalles de la red mybridge con Docker.



o ¿Qué comandos utilizó?

Docker ps para ver el puerto

Docker network inspect mybridge

1. **Análisis del sistema**

• Siendo el código de la aplicación web el siguiente:

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)

• Explicar cómo funciona el sistema

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

Setea variables de entono para poder acceder a un contenedor con su puerto asignado

• ¿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 ?

La imagen de redis ya está descargada, no se volverá a descargar. Simplemente, vuelve a ejecutarse

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

No carga la página web porque le falta la db

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

Se vuelve a generar la DB pero desde cero, perdiendo datos que se podrían haber cargado antes.

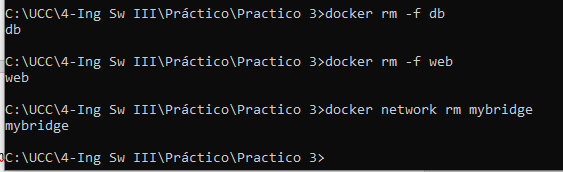
• ¿Qué considera usted que haría falta para no perder la cuenta de las visitas?

• Para eliminar los elementos creados corremos:

docker rm -f db

docker rm -f web

docker network rm mybridge



1. **Utilizando docker compose**

• Normalmente viene como parte de la solución cuando se instaló Docker

• De ser necesario instalarlo hay que ejecutar:

sudo pip install docker-compose

• Crear el siguiente 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

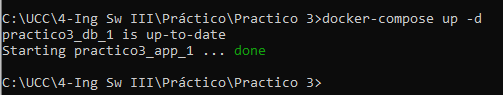
volumes:

- redis\_data:/data

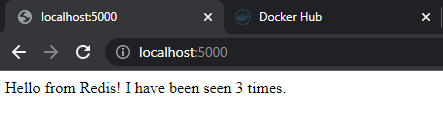
volumes:

redis\_data

• Ejecutar docker-compose up -d

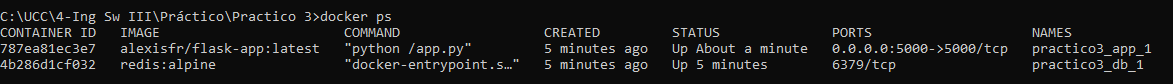


• Acceder a la url <http://localhost:5000/>

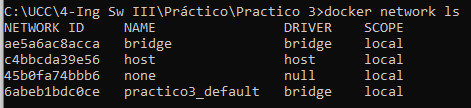


• Ejecutar

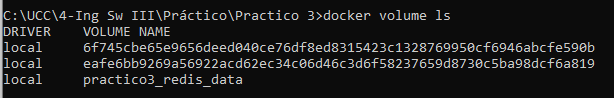
docker ps,



docker network ls



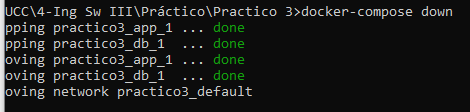
y docker volume ls



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

Creó nuestro contenedor web y la base de datos. Además del contenedor que hace de bridge entre los otros dos contenedores. Por último, nos creó el volumen.

• Desde el directorio donde se encuentra el archivo dockercompose.yaml ejecutar: docker-compose down



1. **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 resultados de la votación en tiempo real.

Pasos:

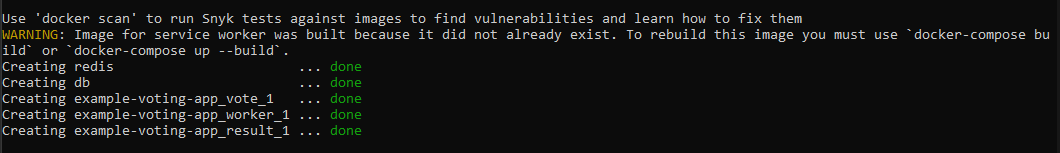
• Clonar el repositorio

https://github.com/dockersamples/example-voting-app

• Abrir una línea de comandos y ejecutar

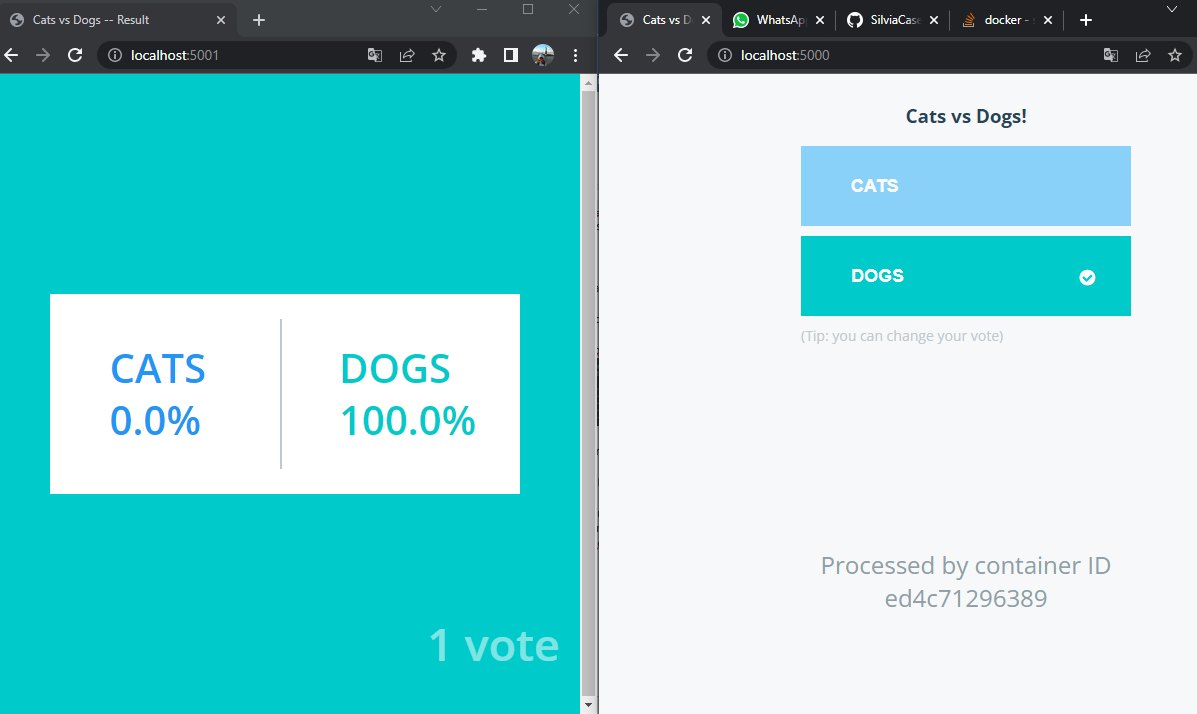
cd example-voting-app

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



• Una vez terminado acceder a http://localhost:5000/ y <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 cómo está configurado el sistema, puertos, volúmenes componentes involucrados, utilizar el Docker compose como guía.

Es una app hecha en java para contar votos. Se tienen dos hosts (pueden ser equivalentes a dos contenedores), uno que recibe los datos y otro, que los graba y los muestra. Para que cada sitio se “entere” de los votos, se usa un volumen que sirve para intercambiar información entre los contenedores.

Tiene 5 contenedores, 2 networks, back-tier y front-tier. Estos, permiten levantar el Sistema y volumen externo, db-data, para guarder los datos de la db localmente.

Contenedor Vote: app web escrita en python para emitir los votos. Usa flask y redis, en donde genera el volumen ./vote:/app, escucha en el puerto 5000 (abierto), y el contenedor, usa el puerto 80 y las dos redes.

Contenedor Result: app web Node.js para visualizar los rdos de la votación en tiempo real. Se genera el volumen ./result:/app, que escucha en el puerto 5001 y 5858 (abiertos) para comunicarse con la DB, y el contenedor usa el puerto 80. Mas las dos redes.

Contenedor Worker: motor de DB en memoria, basado en almacenamiento en tablas de hashes pero que, opcionalmente, puede ser usada como DB durable o persistente. Se usa una queue de Redis para recolectar los nuevos votos y el worker los extrae y los envía a la DB. Usa el puerto 6379 para el host y contenedor y usa la red back-tier.

Db: db postgres en un volumen docker para almacenar los votos. Variable de entorno definida: POSTGRES\_USER: “postgres” como usr y POSTGRES\_PASSWORD: “postgres” como pass de la db. Puerto 5432 (agregado al archivo), genera el volumen db-data:/var/lib/postgresql/data y usa la red back-tier.

Los volúmenes de vote, result y db están asociados al volumen externo db-data. Con el comando docker ps-a vemos el motor de db escucha en el puerto 49513.

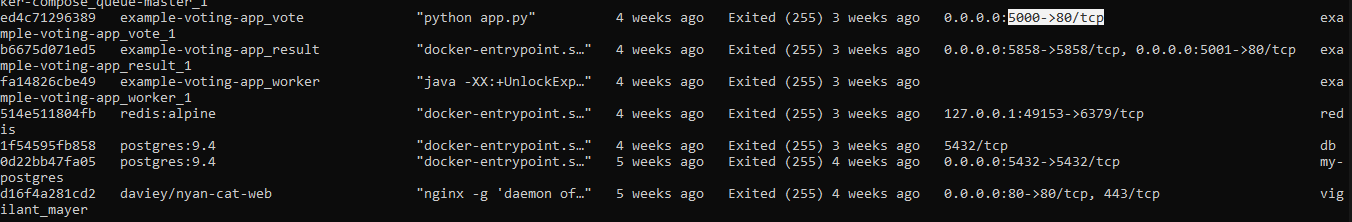
1. **Análisis detallado**

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

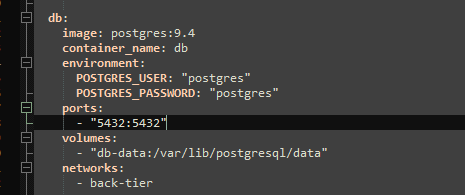
Escribir los puertos para poder acceder

Docker ps -a

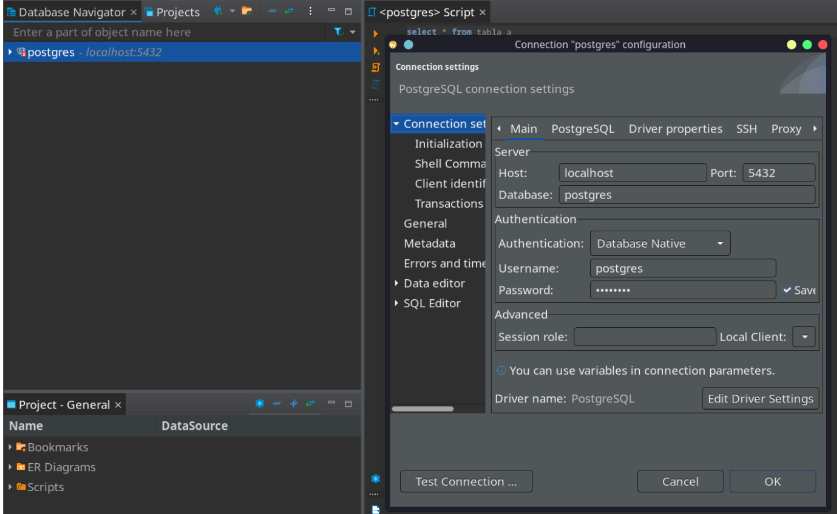
¿Qué puerto tiene postgres en docker? ¿Y el expuesto?

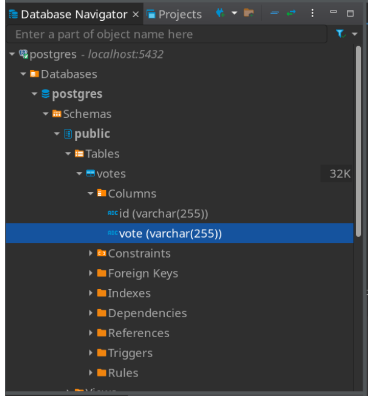


Abrir el archivo yaml y agregar otro puerto



Las tablas de postgres





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

Primero, verifica que el votante no haya votado ya. Luego, toma el voto ingresado. Devuelve los resultados en pantalla y envía los votos a redis.

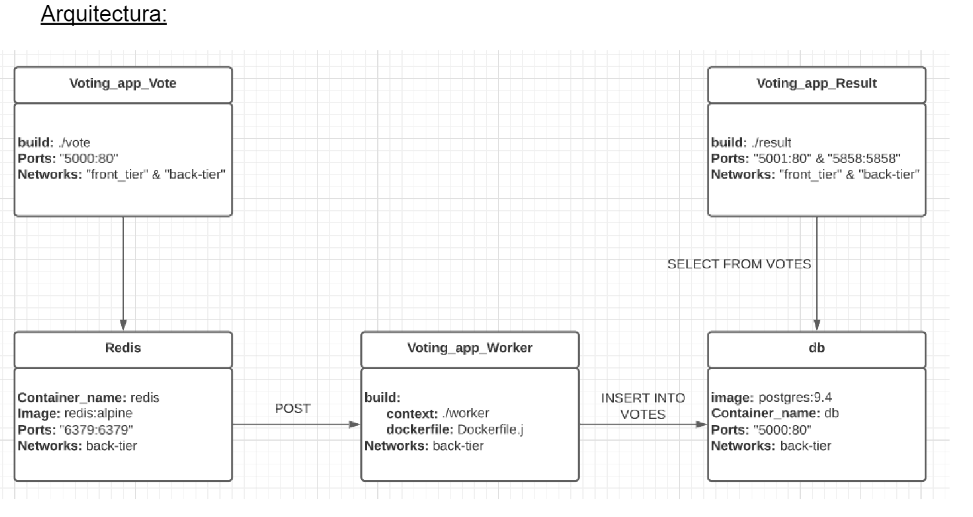
• Revisar el código del worker example-voting-app\worker\src\main\java\worker\Worker.java para entender como procesa los datos.

Se conecta a la db y procesa la info a almancenar en la db. Se conecta a redis y a la db, verifica la conexión de ambas y actualiza la db: id y voto

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

Se ven los rdos de la votación.

• Escribir un documento de arquitectura sencillo, pero con un nivel de detalle moderado, que incluya algunos diagramas de bloques, de secuencia, etc y descripciones de los distintos componentes involucrados es este sistema y como interactúan entre sí



Vote: usr ingresa a la vista localhost:5000 y vota. Una vez hecho esto, se registra un voter\_id que se guarda junto con el voto. POST

Redis: intermediario entre la webapp vote y worker

Worker: procesa la info a almancenar en la db. Se conecta a redis y a la db, verifica la conexión de ambas y actualiza la db: id y voto

DB: db tipo Postgres que recibe info de worker y graba en una tabla.

Result: webapp en la vista localhost:5001, en donde se ven los rdos de la votación.