# DOCUMENTO DE ARQUITECTURA DE SW

## 1. Visión general

Tenemos una aplicación web para votación compuesta por 5 sistemas, como se detalla en la información de readme.md del repositorio https://github.com/dockersamples/example-voting-app:

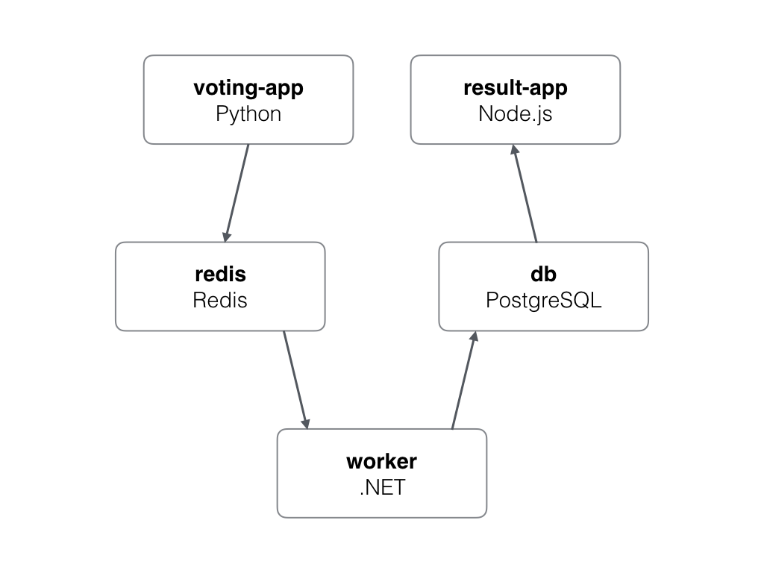
- Una aplicación web de Python que te permite votar entre dos opciones

- 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.



### Diagramas de secuencia del comportamiento del sistema

Diagrama

Descripción generada automáticamente

Al emitir un voto el usuario realiza un post de su voto que se envía a voting-app (quien extrae el id de las cookies) y responde mostrando el valor del voto. Luego hace un push de un objeto JSON que contiene id y vote a Redis quien le asigna una clave y lo almacena.

Diagrama

Descripción generada automáticamente

El worker consume los datos de Redis ejecutando un bucle infinito que vacía la cola Redis (de forma FIFO) obteniendo los JSON con id y vote e insertándolos de la base de datos (en el caso de que la inserción falle se interpreta como que puede ser que ese id ya haya emitido un voto por lo que se procede a hacer un update).

Diagrama

Descripción generada automáticamente

Result-app hace una consulta a bd que retorna la cantidad de votos de a y la cantidad de votos de b y los muestra al usuario.

La estructura de la base de datos es la siguiente:

Tabla

Descripción generada automáticamente

## 2. Forma en la que está configurado el sistema

Tenemos 5 contenedores, cada uno corresponde a un servicio y todos forman parte de la red example-voting-app\_back-tier, por otro lado solo dos de ellos (correspondientes a los servicios vote y result) pertenecen a la red example-voting-front-tier y están publicados en un puerto del host (5000 y 5001 respectivamente). Podemos verlo con el comando docker network inspect:

**> docker network inspect example-voting-app\_back-tier**

[

{

"Name": "example-voting-app\_back-tier",

"Id": "925167c781cbbea711fd1465b1b01b93912d796f5944ce01082af9368b45ef99",

"Created": "2021-09-04T21:59:53.3945838Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.21.0.0/16",

"Gateway": "172.21.0.1"

}

]

},

"Internal": false,

"Attachable": true,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"25d4f13da5842494f8e5fca2fa6ec1fb7b36310f9661624a7bc612700cd7774e": {

"Name": "example-voting-app\_vote\_1",

"EndpointID": "10d641d78bf080434c3bc53bebf504eff53f03146bae19d5ac731d4828422e5a",

"MacAddress": "02:42:ac:15:00:05",

"IPv4Address": "172.21.0.5/16",

"IPv6Address": ""

},

"4b6be63044aba33a036aabc5d66f7cd54b35fe25b11ff6532363cbbe6930f265": {

"Name": "example-voting-app\_worker\_1",

"EndpointID": "f0506e43d0a4709be6d18674ee32ed1ee7553ee3a5a913f9348d9afbd93f8dc5",

"MacAddress": "02:42:ac:15:00:03",

"IPv4Address": "172.21.0.3/16",

"IPv6Address": ""

},

"9f4c83d4430002d720a10929b3dc3abab0f9d6570a9fc7f1872740f143c26504": {

"Name": "redis",

"EndpointID": "f4556280cbdd0f8df7f52353f75ae7996e6cb597986aac56ab265772b29e9b07",

"MacAddress": "02:42:ac:15:00:02",

"IPv4Address": "172.21.0.2/16",

"IPv6Address": ""

},

"a232566b4ff70a4b6e3faf076f0b860b3dc7d14dc531a1388b50bc43a42bb5aa": {

"Name": "example-voting-app\_result\_1",

"EndpointID": "69d85e50eff845079c475e3e327ec2aef2b2623e116ec5ee4d9968cf7b8cd5c3",

"MacAddress": "02:42:ac:15:00:06",

"IPv4Address": "172.21.0.6/16",

"IPv6Address": ""

},

"d3da5893a6616e6cbcf6d4b6325e5362d1fb418fab0764b316c8ce6059e0af16": {

"Name": "db",

"EndpointID": "d2658486e3bc860acc11c08b933853ca312f96bb4b38c10d34abc09cf5495070",

"MacAddress": "02:42:ac:15:00:04",

"IPv4Address": "172.21.0.4/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {

"com.docker.compose.network": "back-tier",

"com.docker.compose.project": "example-voting-app",

"com.docker.compose.version": "1.29.2"

}

}

]

**> docker network inspect example-voting-app\_front-tier**

[

{

"Name": "example-voting-app\_front-tier",

"Id": "cb25075f9b4f4e74b1682c0e226b1520300fa25fee8641723105611e27d8d6ae",

"Created": "2021-09-04T21:59:51.7973397Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.20.0.0/16",

"Gateway": "172.20.0.1"

}

]

},

"Internal": false,

"Attachable": true,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"25d4f13da5842494f8e5fca2fa6ec1fb7b36310f9661624a7bc612700cd7774e": {

"Name": "example-voting-app\_vote\_1",

"EndpointID": "04bdc882064e670556075c95c84d98bafef587f55a6a222a8bb46c9ce6f1c292",

"MacAddress": "02:42:ac:14:00:02",

"IPv4Address": "172.20.0.2/16",

"IPv6Address": ""

},

"a232566b4ff70a4b6e3faf076f0b860b3dc7d14dc531a1388b50bc43a42bb5aa": {

"Name": "example-voting-app\_result\_1",

"EndpointID": "417fa95d44500b5cadc56d1f633c172219d812383c1570afd18f0d78f125b64b",

"MacAddress": "02:42:ac:14:00:03",

"IPv4Address": "172.20.0.3/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {

"com.docker.compose.network": "front-tier",

"com.docker.compose.project": "example-voting-app",

"com.docker.compose.version": "1.29.2"

}

}

]

Se utiliza un nuevo volumen denominado example-voting-app\_db-data por db en la ubicación /var/lib/postgresql/data.

Cuando se ingresa desde el host al puerto 5000 se ejecuta app.py

Cuando se ingresa desde el host al puerto 5001 se ejecuta server.js

## 3. Interacción app.py con Redis

Cuando se realiza un post se guarda en g (varible "global" que tiene el mismo tiempo de vida que el request y se genera con este) una instancia de redis (nombre de host, de la base de datos y socket\_timeout(cierra la conexión si está ociosa por más de 5 segundos)). Al realizarse un post, los resultados de la votación se almacenan en un Json llamado data conformado por las claves voter\_id y vote con sus respectivos valores, y se almacenan en votes de la instancia de redis mediante la función rpush.

from flask import Flask, render\_template, request, make\_response, g

#Application context, cuando una app Flask empieza a manipular una request, pide el contexto (este se crea y destruye cuantas veces sea necesario, pero generalmente tiene el mismo tiempo de vida que la request que le dio origen)

#g: tiene el proposito de almacenar datos en el application context durante una request. Estos datos son globales durante el contexto (termina con el fin de la request).

#make\_response: convierte un valor de retorno en una instancia de la clase response\_class

#render\_template: es una función que se usa para generar resultados a partir de un archivo de plantilla

from redis import Redis

import os

import socket

import random

import json

import logging

option\_a = os.getenv('OPTION\_A', "Cats")

option\_b = os.getenv('OPTION\_B', "Dogs")

hostname = socket.gethostname()

app = Flask(\_\_name\_\_)

gunicorn\_error\_logger = logging.getLogger('gunicorn.error')

app.logger.handlers.extend(gunicorn\_error\_logger.handlers)

app.logger.setLevel(logging.INFO)

def get\_redis():

if not hasattr(g, 'redis'): #devuelve !verdadero si el objeto g tiene un atributo con el nombre dado

g.redis = Redis(host="redis", db=0, socket\_timeout=5)

return g.redis

@app.route("/", methods=['POST','GET'])

def hello():

voter\_id = request.cookies.get('voter\_id')

#pide el valor de voter\_id en cookies

if not voter\_id:

voter\_id = hex(random.getrandbits(64))[2:-1]

#sino ahí no se generó se crea un nro random en hex de 64 bits y se asigna a esa variable, [] refiere a listas.

vote = None

if request.method == 'POST':

redis = get\_redis()

#llama a la función de arriba

vote = request.form['vote']

app.logger.info('Received vote for %s', vote)

data = json.dumps({'voter\_id': voter\_id, 'vote': vote})

#convierte un objeto python en un json

redis.rpush('votes', data)

#añade un valor al final de una lista redis

resp = make\_response(render\_template(

'index.html',

option\_a=option\_a,

option\_b=option\_b,

hostname=hostname,

vote=vote,

))

resp.set\_cookie('voter\_id', voter\_id)

return resp

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=80, debug=True, threaded=True)

## 4. Procesamiento de los datos por Worker.java

En example-voting-app\worker\src\main\java\worker\Worker.java se crea un objeto denominado redis que va a ser un cliente redis y un objeto denominado dbConn que va a ser el cliente de la base de datos postgres. Luego extrae de la conexión redis un par clave-valor en una variable del tipo string y la convierte en un objeto JSON para poder almacenar el valor de voter\_id y vote en dos variables independientes. Por útilmo inserta estos valores como una tupla en la base de datos mediante la función updateVote que ejecuta la consulta a postgres. En el caso de la inserción lance una excepción se hace un update.