eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

TRABAJO INDIVIDUAL

ADMINISTRACIÓN DE SISTEMAS

Autor: *Peio Llano*

6 de diciembre de 2022

Índice

1 Introducción

El objetivo de este trabajo es poner en práctica los conocimientos sobre técnicas de gestión y orquestación de contenedores vistas durante el cuatrimestre en la asignatura de Administración de sistemas. Para ello se asigno una imagen de una aplicación "base"de *Docker* a cada uno de los alumno, en mi caso esa imagen fue *BlobCity*. Respecto al grueso de la tarea, teníamos que escribir un cliente para la aplicación asignada que realizase, al menos, 1 operación básica que en mi caso ha sido crear una colección, escribir un dato en dicha colección.

A partir de esa premisa, teníamos que ir desarrollando el entorno con intención de enriquecerlo y practicar lo aprendido.

2 BlobCity

BlobCity DB es un Data-Lake NoSQL diseñado para el procesamiento de datos a escala organizativa. Recoge datos estructurados y no estructurados en BlobCity DB e impulsa decisiones de negocio utilizando sus avanzadas capacidades de ciencia de datos.

BlobCity está diseñada para satisfacer una gran variedad de requisitos. Dice ser la única base de datos que ofrece dos motores completos de almacenamiento de datos. Uno en memoria y el segundo en disco. El almacenamiento dual le permite dividir sus datos entre la memoria y el disco, sin dividirlos realmente, ni perder la capacidad de consulta colectiva entre los almacenes de datos en disco y en memoria. Los enfoques tradicionales exigían que esos datos se almacenaran en productos diferentes. Esto limitaba enormemente las capacidades de consulta cruzada, al tiempo que añadía una latencia significativa a las ejecuciones de consulta. Estas arquitecturas tradicionales ya no son adecuadas para los requisitos de análisis en tiempo real y de baja latencia.

Por otro lado, BlobCity es una base de datos totalmente compatible con ACID que entra en la categoría de bases de datos de procesamiento analítico/transaccional híbrido (HTAP). Ofrece todas las capacidades de los sistemas de bases de datos relacionales tradicionales, con la velocidad añadida necesaria para la analítica de la nueva era. BlobCity ofrece la potencia de una NoSQL, sin comprometer ninguna de las características de las bases de datos relacionales. HTAP es un diseño nuevo y en constante evolución para los sistemas de almacenamiento de datos, y es una de las primeras bases de datos en atender los requisitos de HTAP.

Por ultimo, no es necesario tener los datos en un formato uniforme. BlobCity almacena y procesa de forma nativa datos en cualquiera de estos 17 formatos: JSON, XML, CSV, SQL, Plaintext, PDF, Excel, Word, RTF, ZIP, Log, Powerpoint, syslog, archivos de audio, archivos de vídeo, archivos de imagen, GIS. Esto permite que la aplicación recopile y procese datos de diversas fuentes sin la complejidad añadida de convertir los datos a un formato uniforme antes de la ingestión.

3 Aplicación Cliente

Para la aplicación cliente he creado un archivo muy básico de *Shell* que encarga de introducir unos datos ya predefinidos, el archivo se vale de *Netcat* para la introducción de los datos. Para ello además del propio fichero de *Shell* hace falta otro fichero que contiene las sentencias que deberían de escribirse para introducir dichos datos una vez dentro de la terminal de BlobCity. Asimismo al ejecutarse podremos ver si los datos se han añadido, pues tendremos la salida de la linea de comandos de BlobCity.

En primer lugar esperaremos unos segundo hasta que el contenedor de *Blobcity* arranque por completo y no nos de errores ni nos devuelva que la conexión ha sido fallida. Después, tendremos que conseguir las credenciales para conectar con el usuario *root* a la base de datos. En otras bases de datos, como por ejemplo *ArangoDB* u otros, se puede definir mediante una variable de entorno cual quieres que sea la contaseña por defecto del usuario *root*, pero en este caso no se puede. Para poder acceder a la *root-pass* he tenido que enlazar mediante volúmenes la carpeta de datos de *Blobcity* con una carpeta dentro del cliente, teniendo accesible de esta forma la contraseña en el lugar donde vamos a necesitar la contraseña para poder acceder a la base de datos desde el cliente. Una vez conseguidas las credenciales se sustituye la contraseña real y se ejecutan todos los comandos de dentro del fichero de comandos.

```
1  #!/bin/bash
2  sleep 2
3
4  echo "Consiguiendo credenciales..."
5  pass=$(cat /data/root-pass.txt)
6
7  echo "Actualizando fichero de comandos..."
8  sed -i 's/passwordHere/'$pass'/g' comandos.txt
9
10 echo "Configuracion finalizada"
11
12 echo "Se procede a crear una colección y escribir un dato en la
   ↪  colección."
13 echo " "
14 nc blobcity 10113 < comandos.txt
15 echo " "
16 echo "Creación de colección y escritura de dato en la colección
   ↪  finalizada."
```

En cuanto a la imagen de *Docker* de la aplicación cliente, utilizaremos como base la imagen de *Ubuntu*. En primer lugar crearemos la carpeta que vamos a definir como directorio de trabajo mediante el uso del comando *WORKDIR*. Una vez configurado el directorio de trabajo, valiéndonos del comando *RUN* vamos a instalas las aplicaciones necesarias para ejecutar nuestro archivo después. Además tendremos que pasar los dos ficheros que hemos nombrado antes al directorio donde nos hemos cambiado y cambiarle los permisos al ejecutable. Por ultimo utilizaremos el comando *CMD* para que cuando se ejecute un contenedor de esta imagen se ejecute el fichero que hemos preparado para la introducción de los datos.

```
1  FROM ubuntu
2
3  RUN mkdir -p /dir
4
5  WORKDIR /dir
6
7  RUN apt-get update && apt-get install -y netcat
8
9  COPY cliente.sh /dir
10 COPY comandos.txt /dir
11
12 RUN chmod +x cliente.sh
13
14 CMD ./cliente.sh
```

4 Docker-compose

Para poder ejecutar localmente la aplicación asignada y la aplicación cliente se debe crear un fichero *docker-compose.yml* que cree un entorno Docker Compose.

En el archivo de docker compose tenemos dos servicios distintos. El primero de ellos, es el servicio de *BlobCity* (la imagen que se me asigno) cuya imagen es *blobcity/db*. Habrá dos puertos abierto de la maquina local al contenedor *Docker*. El puerto 10111 es utilizado por el adaptador y otros servicios web *REST*, mientras que el puerto 10113 se utiliza para la conectividad *CLI/Telnet/Netcat* para la gestión de la base de datos. Los datos almacenados en la base de datos se almacenan dentro del propio contenedor. Esto puede ser un problema, por lo que para asignar un sistema de almacenamiento externo a *Docker*, en este caso con los volúmenes de *Docker*. Todos los datos almacenados por *BlobCity DB* dentro del contenedor se almacenan en la ubicación /data". Cuando se monta una carpeta externa en la misma ubicación del contenedor, todos los datos se guardan en la carpeta externa.

En cuanto al segundo servicio, ejecuta la aplicación cliente. Podemos ver que el volumen de ambos servicio conecta con la misma carpeta de la maquina local esto es para tener accesible la contraseña del usuarios *root* para poder entrar a la consola de la base de datos e introducir datos.

```
services:
  blobcity:
    container_name: blobcity
    image: blobcity/db
    ports:
      - 10111:10111
      - 10113:10113
    volumes:
      - ./blobcity_data:/data

  cliente_blobcity:
    container_name: clienteBlobcity
    image: peiollano/as-cliente-blobcity
    restart: on-failure
    links:
      - blobcity
    volumes:
      - ./blobcity_data:/data

```

```
20 volumes:
21   blobcity_data:
```

5 Kubernetes

Para ejecutar la aplicación asignada y la aplicación cliente he tenido que crear una aplicación Kubernetes. Para ello, he tenido que crear varios ficheros de configuración y crear una nueva imagen que se adecue a las necesidades de este apartados. La estructura resultado de dicha carpeta es la siguiente.

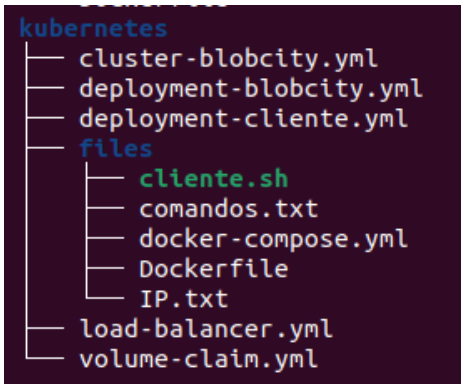

Figura 1: Estructura de la carpeta kubernetes

- **Deployment Blobcity**: Este fichero lanzara la imagen de *Blobcity* y configurara tanto los puertos como los volumenes que deseemos.
- **Volume Claim**: Gracias a este fichero el *deployment* anterior podrá guardar de forma persistente los datos. Al no estar conectado con un volumen en concreto lo creara.
- **Load Balancer**: Este archivo expondrá un puerto del *Pod* creado por el *Deployment* al exterior del cluster. De esta manera podremos acceder a la consola de la base de datos.
- **Deployment Cliente**: Este fichero lanzara la imagen creada por nosotros e introducirá los datos.
- **Carpeta "files"**: Esta carpeta busca solucionas los problemas que presenta la complejidad de acceso de la imagen que se me ha asignado.

  Por un lado, como ya se había comentado previamente, para conseguir la contraseña de l usuario root"había que conectar los volúmenes de las dos imágenes utilizando como puente la maquina en la que se ejecutaba. Esto cambia para la ejecución en *kubernetes*. Reproducir el mismo proceso en este caso se volvía muy engorroso, por lo que se ha decidido quitar la

parte de código donde buscaba la *root-pass* y poner la contraseña como root". Mas adelante se explicara como hacer para que esto sea posible.

Por otro lado, la forma de conectarse a *Netcat* también cambia pues tenemos que utilizar la IP resultante de la ejecución del *load-balancer*. De esta manera se crea una archivo que contenga dicha dirección IP. Mas adelante se explicara como hacer para que esto sea posible.

- **Cluster Blobcity**: Este archivo permite conectar diferentes objetos a nivel de cluster, util para conectar el deployment del cliente a el deployment de la base de datos. Por necesidades especiales de la imagen no se ha podido hacer de esta manera y no se utiliza.

6 Pautas para la ejecución

6.1 Docker

Para la ejecución de la aplicación usando *Docker* se recomienda seguir los siguientes pasos. Se sobre-entiende que el usuario a ejecutar el programa debe estar dentro de la carpeta "docker".

```
#1. Paso
sudo docker-compose rm
#2. Paso
sudo docker-compose up --force-recreate
```

Nos debería de salir un mensaje entre el que podremos leer Ïnserted". En cualquier caso podremos ver si se ha creado accediendo a la carpeta "blobcity_data".

6.2 Kubernetes

Para la ejecución de la aplicación usando *Kubernetes* se recomienda seguir los siguientes pasos. Se sobre-entiende que el usuario a ejecutar el programa debe estar dentro de la carpeta "kubernetes". Antes de comenzar la ejecución desde el apartado "Direccion IP"dentro de Red de VCP.[en] "Google Cloud Console"se debera de reservar una IP estática. Una vez tengamos el número IP tendremos que cambiarlo tanto en el fichero ÏP.txt"de la carpeta files como en el valor de la clave "loadBalancerIP"del archivo "load-balancer.yml". Una vez hecho esto debemos de crear de nuevo la imagen de Docker (en local o en *DockerHub*, como se prefiera) y actualizarla en el valor de la clave ïmage.[en] el archivo "deployment-cliente.yml".

```
#1. Paso
kubectl apply -f volume-claim.yml
#2. Paso
kubectl apply -f deployment-blobcity.yml
#3. Paso
kubectl apply -f load-balancer.yml
```

Llegados a este punto podremos conectarnos a la base de datos mediante la IP estática que hemos reservado previamente. Ejecutaremos los siguientes comando para poner la contraseña como root".

```
#1. Paso
kubectl get pods
  #Cogeremos el nombre del pod creado
#2. Paso
kubectl exec -it <nombre-del-pod-creado> cat /data/root-pass.txt
  #Cogeremos la root-pass
#3. Paso
nc <direccion-del-load-balancer> 10113
#4. Paso
  #Acceder a la base de datos con las creadenciales que hemos
  ↪  obtenido en el paso 2
set-user-password root <root-pass> root
```

En este punto ya tendremos todo listo para ejecutar la aplicación cliente. Como a veces con *kubernetes* suele fallar se explicaran dos formas de hacerlo; vía *kubernetes* y vía *docker*.

```
#Via kubernetes
kubectl apply -f deployment-cliente.yml

#Via Docker
  #Estando dentro de la carpeta "files"
  #1. Paso
  sudo docker-compose rm
  #2. Paso
  sudo docker-compose up --force-recreate

```

7 Código y más

Código de la aplicación: `https://hub.docker.com/repository/docker/peiollano/as-cliente-blobcity`

Imagen de la aplicación cliente en *Docker*: `https://hub.docker.com/repository/docker/peiollano/as-cliente-blobcity`

Imagen de la aplicación cliente en *Kubernetes* (no va a ser útil para la gente que quiera utilizarlo que no sea yo): `https://hub.docker.com/repository/docker/peiollano/as-cliente-blobcity-kub`