CLIENTE Y BASE DE DATOS INFLUX DB

Xabier Fernández Escabias

InfluxDB

¿Qué es InfluxDB?

Se trata de un **sistema de gestión de bases de datos** creada por los desarrolladores de InfluxData, Inc. Es un software gratuito y de código abierto, aunque posee una versión comercial llamada **InfluxDB Enterprise**, con la que los desarrolladores ofrecen mantenimiento y servidores dedicados a los clientes que lo adquieran.

InfluxDB ha sido creado en **Go**, el lenguaje de programación desarrollado por Google, y que hoy en día es muy popular en todo tipo de aplicaciones.

¿Por qué usar InfluxDB?

InfluxDB fue creado con el objetivo de almacenar **series temporales**. Las bases de datos que se gestionan son llamadas **Time Series Data Bases (TSDB)**, cuya principal finalidad es la de almacenar un flujo continuo de datos, extraídos comúnmente de instrumentos o herramientas de recolección de métricas (Termómetros, sensores, IoT…).

InfluxDB es capaz de procesar rápidamente todos esos datos y almacenarlos en una base de datos TSDB, que tienden a ser bastante simples, con dos o tres columnas.

Ejemplo de una TSDB:

Hora	**Sensor**	**Valor**
29/11/2021-19:09:56	Sensor_1	10,23
29/11/2021-19:09:57	Sensor_2	10,29
29/11/2021-19:10:43	Sensor_1	11,01

Cabe destacar que lo que caracteriza una **TSDB** es que almacena un campo **time** con la hora o fecha en la que se ha recogido la métrica, aunque no sea un campo obligatorio.

¿Cómo almacena los datos?

InfluxDB tiene una estructura que separa el contenido de las bases de datos en dos conceptos: **Tags** y **fields:**

Por una parte, los **tags o etiquetas** son usados como *metadatos* que ayudan a organizar las tablas, cumpliendo la función de índices. Un ejemplo sería la columa **Sensor** del ejemplo anterior, ya que no contiene ningún valor por sí mismo, pero viene emparejada con el **valor** que representa a esa **Hora**.

Por otra parte, tenemos los **fields o campos.** Almacenan los valores que querremos evaluar en un futuro, es decir, la **métrica** que nos interesa guardar.

Una opción de subir nuevas métricas a InfluxDB es en formato **JSON**, que será utilizado en esta práctica para actualizar la base de datos. Se envía el diccionario al cliente de InfluxDB indicando los tags y fields que queramos añadir.

Aplicación y cliente en Docker

Objetivo

El objetivo del sistema creado será disponer de una base de datos con InfluxDB y utilizar un pequeño cliente improvisado creado en Python para crear y leer datos de la base de datos.

El sistema simula una especie de "almacen de comentarios", en el que puedes añadir una **frase o comentario** junto a una palabra descriptiva sobre el **tema** del comentario.

Sistema creado

El sistema junto con la documentación ha sido subido al repositorio de GitHub público https://github.com/xafer337/influxdb_client.

Se lanzará un contenedor Docker con una Imagen de InfluxDB, a la que accederá nuestra aplicación cliente desde otro contenedor a través del puerto 8086. Para ejecutar el cliente se debe llamar a la aplicación encapsulada dentro del contenedor cliente:

docker exec -it client influx_client

Con ello ejecutaras la aplicación en modo interactivo de docker. Una vez en ella, tendrás dos opciones: **Escribir comentario**, con la que añadirás datos a la BBDD y **leer comentarios**, con lo que tendrás acceso a todos los comentarios almacenados.

Como extensión, se podrán guardar los comentarios leídos en un archivo de texto, que serán guardados en un volumen ubicado en **/tmp/influx_comentarios/** del host.

Tareas completadas

Para este proyecto se han completado las **tareas mínimas** requeridas para obtener un 5:

- **Descripción** de la aplicación asignada.
- Creación de un cliente con **1 operación básica** (lectura y escritura en BBDD en este caso) implementado en python.
- **Dockerización del cliente** y documentación del código generado.
- Imagen del cliente subida a **DockerHub**.
- Entorno **Docker-compose** para la ejecución del cliente y la aplicación designada.

Además, se han completado algunas de las **tareas optativas**:

- Creación de la aplicación en **Kubernetes**.
- Implementación de una operación básica que escriba en un fichero.
    - **Volumen** con el que se guardan búsquedas en la base de datos en un fichero, compartido con el host mediante un volumen. (Implementado en el mismo docker-compose que la entrega obligatoria).
    - **Volumen de kubernetes** para la persistencia de datos de la BBDD de influxDB.

Código generado

Docker

- Dockerfile

Por un lado, tenemos el **Dockerfile**, encargado de crear nuestro cliente a partir de una imagen de Python:3.

La imagen **xafer337/influx_client** será subida al repositorio en:

https://hub.docker.com/r/xafer337/influx_client.

```
FROM python:3                    # Imagen base de Python 3

WORKDIR /usr/local/bin/          # Cambiamos el directorio en el que trabajamos a uno en la variable de entorno $PATH

RUN pip install influxdb         # Instalamos el cliente Python para comunicarnos con InfluxDB

COPY influx_client .             # Copiamos el código del cliente Python que hemos creado en la imagen. Se adjunta el código del programa comentado

RUN chmod +x influx_client       # Otorgamos permisos de ejecución a la aplicación cliente. En el programa cliente especificamos en la primera línea que debe ejecutarse con Python.
```

- ## Docker-compose

En el **docker-compose** lanzaremos nuestro cliente y una instancia de InfluxDB, en contenedores separados.

```
version: '3'          # Versión de docker-compose
services:
 influxDB:            # Contenedor con InfluxDB
  image: influxdb:1.8.10            # Imagen provista de InfluxDB
  container_name: influxDB          # Nombre identificativo para el contenedor
  environment:  # Variables de entorno usadas por InfluxDB
   - INFLUXDB_ADMIN_USER=admin                # Nombre del usuario administrador
   - INFLUXDB_ADMIN_PASSWORD=admin            # Contraseña del usuario administrador
   - INFLUXDB_DB=influx_db                    # Base de datos base que se creará
  volumes:          # Almacenamiento persistente de la base de datos
   - data_influx:/var/lib/influxdb                  # Almacenamos el directorio de datos de InfluxDB

 client:              # Contenedor con la aplicación cliente
  image: xafer337/influx_client:v1.0.0    # Imagen creada por el Dockerfile
  container_name: client                   # Nombre identificativo para el contenedor
  tty: true          # Activar una pseudo terminal para evitar el cierre del contenedor
  volumes:           # Directorio en el que se almacenarán los comentarios leídos
   - /tmp/influx_comentarios/:/tmp/influx_comentarios/

volumes:
 data_influx:       # Volumen de almacenamiento persistente de InfluxDB
```

Kubernetes

Se harán uso de dos *deployments* para lanzar, por un lado, la **aplicación cliente** y por el otro la **base de datos de InfluxDB**. Para poder comunicar ambos componentes se hará uso de un clusterIP conectado al deployment de la base de datos, exponiendo el puerto 8086 al cliente.

Además, se incluirá un componente **persistentVolumeClaim** para la persistencia de datos de la BBDD.

Todos los objetos **Kubernetes** han sido subidos a github.

- ## Deployment cliente:

```
apiVersion: apps/v1                # Versión de la api
kind: Deployment                   # Especificamos que es un objeto de tipo Deployment
metadata:
 name: client-deployment           # Label para que sea fácil identificar este Deployment
spec:
 replicas: 1                       # Queremos una única réplica del contenedor
 selector:
  matchLabels:
   component: client               # Para que el Deployment encuentre los Pods. Buscará aquellos
                                   con esta etiqueta
 template:
  metadata:
   labels:
    component: client              # Label identificativa de los pods gestionados por el Deployment
  spec:
   containers:                     # Lo mismo que en el Docker-compose, pero con otro formato.
    - image: xafer337/influx_client:v1.0.0      # imagen del cliente
      name: client                 # nombre del contenedor
      tty: true                    # mantener abierto el contenedor
      volumeMounts:                # volumen para persistencia
       - mountPath: /tmp/influx_comentarios/
         name: client-vol          # nombre del volumen (definido más adelante)
   restartPolicy: Always           # restart en caso de fallo
   volumes:
```

```
    - name: client-vol          # definimos los volúmenes
      emptyDir: {}
```

- **Deployment InfluxDB:**

```
apiVersion: apps/v1             # Versión de la api
kind: Deployment                # Especificamos que es un objeto de tipo Deployment.
metadata:
 name: influxdb-deployment      # Label para que sea fácil identificar este Deployment
spec:
 replicas: 1                    # Queremos una única réplica del contenedor
 selector:
  matchLabels:
   component: influxdb          # Para que el Deployment encuentre los Pods.
 template:
  metadata:
   labels:
    component: influxdb         # Label identificativa de los pods gestionados por el Deployment.
  spec:
   containers:                  # Lo mismo que en el Docker-compose, pero con otro formato.
    - env:                      # Variables de entorno para InflucDB (usuario admin, psw…)
       - name: INFLUXDB_ADMIN_PASSWORD
         value: admin
       - name: INFLUXDB_ADMIN_USER
         value: admin
       - name: INFLUXDB_DB
         value: influx_db
      image: influxdb:1.8.10    # imagen del cliente
      name: influxdb            # nombre del contenedor
      volumeMounts:             # volumen para persistencia
       - mountPath: /var/lib/influxdb
         name: data-influx      # nombre del volumen (definido más adelante)
```

```
    restartPolicy: Always          # restart en caso de fallo
    volumes:
     - name: data-influx           # definimos los volúmenes
       persistentVolumeClaim:
         claimName: data-influx-claim    # Objeto volumen de kubernetes
```

- **Volumen InfluxDB:**

Tarea optativa: crear volumen en objeto kubernetes.

```
apiVersion: v1                     # Versión de la api
kind: PersistentVolumeClaim        # Tipo de objeto volumen claim (para persistencia de
                                   datos)
metadata:
 labels:
  component: data-influx-claim     # label identificativa de este componente.
 name: data-influx-claim           # label para que el Pod de InfluxDB pueda encontrarlo
spec:
 accessModes:
  - ReadWriteOnce                  # Tipo de acceso
 resources:
  requests:
   storage: 100Mi                  # Almacenaje solicitado al host
```

- ## ClusterIP InfluxDB

Expondrá el puerto 8086 **del deployment de InfluxDB a la aplicación cliente**.

```
apiVersion: v1                          # Versión de la api
kind: Service                           # Especificamos que es un objeto de tipo service
metadata:
  name: influxDB-cluster-ip-service     # Label para que sea fácil identificar el servicio
spec:
  type: ClusterIP                       # Servicio de tipo ClusterIP (solo necesitamos exponer
                                        puertos a objetos del mismo cluster, no al exterior)
  selector:
    component: influxdb                 # Para que el clusterIP encuentre los Pods de InfluxDB.
                                        Recuerdo que todos tenían la label de "component:
                                        influxdb"
  ports:                                # Mapeamos el puerto 8086 del Pod al 8086 del
                                        clusterIP
  - port: 8086
    targetPort: 8086
```