Proyecto 1 - Data Migration Platform (DMP)

Tecnológico de Costa Rica Escuela de Ingeniería en Computación Bases de Datos II (IC 4302) Segundo Semestre 2022

> *Gabriela Gutiérrez Valverde - 2019024089 Brayan Marín Quirós - 2019180348 David Achoy Yakimova - 2020053336*

Guía de Instalación y uso:

Para iniciar es necesario que se tenga instalado Docker, con Kubernetes activo y debemos asegurarnos que tenga la configuración con Linux, esto es solamente seguir las instrucciones de instalación. Y Helm, ya que esto es necesario para poder ejecutar los comandos. Además, utilizamos Lens para trabajar con más facilidad Kubernetes.

Ya con esto debemos abrir la consola de comandos. En el caso de Windows trabajamos con el PowerShell, como el proyecto ya está creado, no es necesario volver a ejecutar los comandos de "Helm create …". Por lo tanto, el siguiente paso es ejecutar los comandos de instalación de las bases de datos, estos son los siguientes:

Para ingresar a la carpeta del proyecto

```
cd "C:\...\Proyectos\Proyecto1Nuevo\Proyecto01"
```

Debemos realizar la instalación de los motores de monitoreo, con el siguiente comando:

```
helm install monitoring monitoring
```

Y una vez que tenemos el sistema de monitoreo debemos instalar las bases de datos con el comando:

```
helm install databases databases
```

Ya con esto contamos con la instalación de Prometheus, Grafana, RabbitMQ, Elastic Cloud on Kubernetes y MariaDB.

Ahora se puede utilizar la aplicación de Docker Desktop para ver los contenedores que se han creado. Y con la aplicación de Lens es posible consultar los pods, secrets, services, entre otros.

Lo siguiente es instalar la aplicación, es decir, el conjunto de archivos Python que se ejecutan como imágenes de Docker. Para lo que se utiliza el comando:

```
helm install application application
```

Al ejecutar este comando comienza a correr la aplicación con los pods del Orchestrator, MySQL Connectors, SQL Processor, REGEX Processor, Elasticsearch Publisher. De forma inmediata el Orchestrator va a crear el índice de "**jobs**" y "**groups**" en Kibana, en caso de que no existan y espera a que se ingresen nuevos documentos.

Para el monitoreo:

Debemos ingresar a la interfaz de Grafana con estos credenciales:

USER	PASSWORD
admin	11111111

Al ingresar, en la configuración debemos presionar la opción **Add data source** y seleccionar la opción de **Prometheus**, debemos agregar el siguiente dato:

URL
http://monitoring-kube-prometheus-prometheus:9090/

Y presionar **Save & test**.

Para visualizar los datos se debe agregar Dashboards los cuales se pueden conseguir aquí. Se pueden agregar copiando el código y en el apartado de Dashboards en el Grafana local presionamos **New** y lo importamos, seleccionamos Prometheus como datasource, con esto podemos observar los datos deseados.

Recomendación de Dashboards

Herramienta	ID
MariaDB	13106
ElasticSearch	6483
RabbitMQ	10991
Pods de la aplicación	747
## Pruebas	

La prueba más completa se puede hacer ejecutando la aplicación completa con el archivo *json* de ejemplo. Sin embargo dentro del archivo del proyecto se incluyen pruebas que se pueden realizar a componentes individuales, como por ejemplo, solamente para el orchestrator.

Pasos:

Lo primero que se deb ehacer para realizar las pruebas es ingresar los valores de prueba en MariaDB, para esto utilizar el siguiente código:

Creación de la tabla de personas:

```
CREATE DATABASE people_db

CREATE TABLE persona (
     id int auto_increment,
    cedula INT,
    nombre VARCHAR(64),
    primary key(id)
    );
```

Creación de la tabla de carros:

```
CREATE DATABASE car_db
```

```
CREATE TABLE car (
     id int auto_increment,
    owner INT,
    description VARCHAR(64),
    primary key(id)
    );
```

Luego se debe ejecutar las siguientes instrucciones en *people_db* para crear datos:

```
DELIMITER |
BEGIN NOT ATOMIC
    DECLARE x INT DEFAULT 0;
    DECLARE ced INT DEFAULT 1000;
    WHILE x <= 400 DO

      INSERT INTO persona(cedula, Nombre)
        VALUES (ced+X, "Nombre");
        SET x = x + 1;
    END WHILE;
END|
DELIMITERS
```

Luego se debe ejecutar las siguientes instrucciones en *car_db* para crear datos:

```
DELIMITER |
BEGIN NOT ATOMIC
    DECLARE x INT DEFAULT 1;
    DECLARE z INT DEFAULT 0;
    DECLARE y INT DEFAULT 100;
    DECLARE ow INT DEFAULT 1000;
    WHILE z <= 400 DO
      INSERT INTO car(owner, description)
        VALUES (X, CONCAT("Toyota CVY-", y) );
        SET x = x + 1;
      SET y = y + 1;
      SET z = z + 1;
   END WHILE;
END|
DELIMITER
```

Una vez con esto, ya tenemos creados los datos, entonces debemos entrar a Kibana con los datos de ElasticSearch. Y copiar estas instrucciones en la opción de *dev tools* :

```
post jobs/_doc/
{
    "job_id": "job606",
    "status": "new",
    "msg": "",
    "data_sources": [
        {
            "type": "mysql",
            "name": "people_db",
            "url": "databases-mariadb-people",
```

```
            "port": "3306",
            "usuario": "XXXXXXX",
            "password": "XXXXXXX"
        },
        {
            "type": "mysql",
            "name": "car_db",
            "url": "databases-mariadb-people",
            "port": "3306",
            "usuario": "XXXXXX",
            "password": "XXXXXX"
        },
        {
            "type": "elasticsearch",
            "name": "destination_es",
            "url": "https://quickstart-es-http",
            "port": "9200",
            "usuario": "XXXXXXXXXX",
            "password": "XXXXXXXXX"
        }
    ],
    "control_data_source": "destination_es",
    "source": {
        "data_source": "people_db",
        "expression": "SELECT * FROM persona ORDER BY cedula",
        "grp_size": "100"
    },
    "stages" : [
        {
            "name": "extract",
            "source_queue": "extract",
            "destination_queue": "%{transform->transformation->add_car}%"
        },
        {
            "name": "transform",
            "transformation": [
                {
                    "name": "add_car",
                    "type": "sql_transform",
                    "table": "car",
                    "expression": "SELECT %{field_description}% FROM %{table}% WHERE %
{field_owner}% = %{doc_field}%",
                    "source_data_source": "car_db",
                    "destination_data_source": "destination_es",
                    "doc_field": "id",
                    "source_queue": "sql_queue",
                    "destination_queue": "%{transform->transformation->myregex}%",
                    "fields_mapping": {
                        "field_description": "description",
                        "field_owner": "owner"
                    }
                },
```

```
                {
                    "name": "myregex",
                    "type": "regex_transform",
                    "regex_config": {
                        "regex_expression": "^.* ([a-zA-z]{3}-[0-9]{3}) .*$",
                        "group": "1",
                        "field": "description"
                    },
                    "field_name": "placa",
                    "source_queue": "regex_queue",
                    "destination_queue": "%{load}%"
                }
            ]
        },
        {
            "name": "load",
            "source_queue": "ready",
            "destination_data_source": "destination_es",
            "index_name": "persona"
        }
    ]
}
```

Y con esto ya se estaría enviando la información a la aplicación por medio del Orchestrator, luego al SQL Connector, SQL Processor, Regex Processor y finalmente al Elastic Publisher. Si la prueba fue exitosa se publicarán los datos en el *root.stages[name=load]. index_name*, para corroborar los datos se puede utilizar la siguiente instrucción:

```
GET /_search
{
    "query": {
        "match_all": {}
    }
}
```

Y el índice se mostrará de la siguiente manera:

```
{
    "_index": "persona",
    "_id": "192",
    "_score": 1,
    "_routing": "jobs/_create/192",
    "_source": {
      "cedula": 1191,
      "description": "Toyota CVY-291",
      "id": 192,
      "nombre": "Nombre",
      "placa": "CVY-291"
    }
}
```

*Para la entrega de este proyecto no fue posible incluir pruebas de rendimiento principalmente por el factor de tiempo y completitud de la aplicación.

Recomendaciones y Conclusiones

Recomendaciones

1. Se recomienda leer toda la documentación antes de ejecutar el programa.
2. En caso de querer modificar la configuración de los pods se debe hacer desde el chart correspondiente, esto debido a que se utilizan variables de entorno para su configuración.
3. Se recomienda conocer la librería de ElasticSearch, la cual se puede instalar con el siguiente comando:

   ```
   python -m pip install elasticsearch
   ```

4. Se recomienda conocer la librería de MariaDB, la cual se puede instalar con el siguiente comando:

   ```
   python -m pip install mariadb
   ```

5. Se recomienda conocer la librería de RabbitMQ, la cual se puede instalar con el siguiente comando:

   ```
   python -m pip install pika
   ```

6. Para realizar pruebas se recomienda hacerlo con los pods de la imagen de python, no solamente con una descarga de python ya que puede existir una diferencia de versiones y algunas funciones podrían no estar disponibles en la versión.
7. En el caso de utilizar el sistema operativo Linux, algunos comandos presentados en las instrucciones de este documento podrían variar, por lo que se recomienda buscar los comandos adecuados para el sistema operativo.
8. Es recomendable contar con una computadora con suficiente Memoria y CPU para que al levantar la aplicación y utilizarla no se ponga lenta o incluso se quede pegada.
9. Se recomienda realizar una lectura al código fuente de la aplicación para familiarizarse con el uso de esta.
10. Se recomienda realizar la lectura y comprensión del documento *json* que se sube a kibana para conocer sus parámetros y las modificaciones que se pueden realizar.

Conclusiones

1. Se obtuvo un gran aprendizaje de las herramientas de Docker y Kubernetes, y la razón de su necesidad para el proyecto.
2. Se trabajó más con la herramienta de Helm para la automatización del proyecto, objetivo que no fue logrado para la entrega de Observability.
3. Fue muy importante comprender el documento *json* para el desarrollo de la aplicación ya que se tienen muchos datos que se están modificando en la aplicación.
4. Se logró toda la instalación y configuración de las Bases de Datos, ElasticSearch y MariaDB para el trabajo del proyecto.
5. Se logró crear todas las colas correspondientes con la ayuda de la herramienta RabbitMQ.

6. Se obtienen exitosamente métricas de MariaDB y RabbitMQ.
7. El proyecto se encuentra con una completitud del 90%, faltando el monitoreo de ElasticSearch y los tiempos de procesamiento de los grupos.
8. La curva de aprendizaje para completar los objetivos del proyecto fue bastante grande pero se logró la mayoría del propósito del proyecto.
9. Fue necesario contar con tiempo extra para poder lograr la completitud del proyecto.
10. Todos los aprendizajes fueron compartidos por el grupo de trabajo.

Referencias Bibliográficas

How to Connect Python Programs to MariaDB - referencia

Elastic Python Client - referencia

Configure ECK - referencia

Create Index API - referencia

MySQL: Get column name or alias from query - referencia

Python String Methods - referencia

Python Official Image - referencia