Proyecto #1: Data Migration Platform (DMP)

> Integrantes:

- Max Richard Lee Chung - 2019185076
- Miguel Ku Liang - 2019061913
- Sebastian Campos Zuñiga - 2016140230
- Dylan Stef Torres Walker - 2018135751

Guía de instalación

El programa necesita de la aplicación "Docker" y habilitar el servicio de "Kubernetes" para tener un clúster inicial y básico, llamado "docker-desktop". La aplicación para controlar y monitorear el comportamiento del clúster de Docker se denomina "Lens". Además, se utilizarán tres carpetas para cada uno de los servicios automatizados del proyecto tales como helm_charts, consumer y producer.

Instalación de helm_charts

Para el desarrollo de este proyecto, se van a implementar dos helm charts principales (databases y monitoring) y una para las aplicaciones implementadas en Docker (application, el cual será mencionado posteriormente). Dentro del helm chart databases, se utilizó la base de datos MariaDB y Elasticsearch. Dentro del helm chart monitoring, se utilizó Grafana y Prometheus junto con el operador de Elasticsearch (eck-operator). Antes de instalar todos componentes, se debe de ubicar o crear una carpeta cualquiera para luego descargar las dependencias necesarias. Ya con la carpeta seleccionada, se ingresa a la consola de comando y realizar un "cd" a dicha carpeta y ejecutar los siguientes comandos:

```
helm create batabases
helm create monitoring
helm create application           # Se comentará más adelante su uso
```

Luego de ejecutar los comandos, se debe de eliminar el contenido de la carpeta "templates" de todos los archvios generados para evitar algún problema a la hora de instalar los recursos. En este punto, se debe de gestionar las dependencias de los helm charts de databases y monitoring dentro del archivo Chart.yaml al final del códgio ya proporcionado, el cual se debe de conocer las versión, enlace del repositorio y nombre para poder extraer y descargar como un archivo comprimido los recursos de cada uno. Los repositorios se deben de buscar en Google buscando, como por ejemplo, el repositorio de Bitnami o Elasticsearch. Al conocer el enlace del repositorio, se añade a la libería de helm charts y buscar el nombre del recurso al que se quiera utilizar u actualizar como se muestra en el siguiente bloque de código.

```
helm add repo <enlace proporcionado>    # Agregar a la librería de helm charts
helm update repo <enlace proporcionado> # Actualizar el repositorio
helm search repo <recurso a buscar>     # Buscar la información del recurso
```

Al conocer los datos específicos de los recuros a utilizar, se agregan las dependencias como se observa en el siguiente apartado. Además, se agregaron variables para habilitar o deshabilitar el recurso en específico, el cual se comentará más adelante sobre cómo configurarlo.

```
# Chart.yaml de monitoring
dependencies:
- name: kube-prometheus
  version: "8.1.11"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enablePrometheus
- name: grafana
  version: "8.2.11"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableGrafana
- name: eck-operator
  version: "2.4.0"
  repository: "https://helm.elastic.co"
  condition: enableOperator
```

```
# Chart.yaml de databases
dependencies:
- name: rabbitmq
  version: "11.0.3"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableRabbitMQ
- name: mariadb
  version: "11.3.3"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableMariaDB
- name: elasticsearch
  version: "19.4.4"
  repository: "https://charts.bitnami.com/bitnami"
  condition: enableElasticsearch
```

Al tener listas las dependencias con los recursos deseados, se debe de hacer un "cd" a cada una de las carpetas generadas para poder ejecutar el siguiente comando, con el fin de "actualizar" las dependencias y que se descarguen los archivos comprimidos como se mencionó anteriormente e instalarlas.

```
cd databases                        # Ingresa a la carpeta
helm dependency update              # Actualiza las dependencias
cd ..                               # Se sale de la carpeta
helm install databases databases    # Se instala el helm chart
```

En caso contrario

```
```

```
cd monitoring
helm dependency update
cd ..
helm install monitoring monitoring
```

Sin embargo, si se instalan directamente, las bases de datos no tendrán las configuraciones que se desea, por lo que se modificó el archivo values.yaml para dicho aspecto. Para ello, se agregaron variables para habilitar o deshabilitar dicho recurso y habilitar las métricas del servicio de monitoreo para que Prometheus pueda observarlos y notificar en Grafana.

```
enableRabbitMQ: true
enableMariaDB: true
```

La primera configuración es la de RabbitMQ, en donde únicamente se le agregó la contraseña y habilitar el servicio de monitoreo.

```
rabbitmq:
  auth:
    # Se agrega un password fija para installar/desinstalar
    password: "rabbitmqpass"
  # Habilitar el servicio de monitoreo
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
```

La segunda configuración es la de MariaDB, en donde se tuvo que crear la instancia estática de una contraseña principal y un usuario con su respecta contraseña para evitar conflictos con los datos persistentes almacenados en Kubernetes. Además, se le habilitó el servicio de monitoreo.

```
mariadb:
  auth:
    # Se agrega un password fija para installar/desinstalar
    rootPassword: "mariadbpass"
    username: "user"
    password: "user"
  # Habilitar el servicio de monitoreo
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
```

Por último, la base de datos Elascticsearch se configuró los recursos del ordenador a utilizar por cada réplica (un máster, tres nodos de datos, un coordinador y un procesador de datos), en este caso, se optó por

configurarlo a un consumo de 1Mb excepto el máster, el cual usa 2 Mb.

```
  elasticsearch:
    # Configuracion del nodo master
    master:
      # Numero de replicas definidas para el master
      replicaCount: 1
      resources:
        requests:
          memory: "2Mi"
          cpu: "125m"
    # Configuracion del nodo de datos
    data:
      # Numero de replicas definidas para almacenar los datos
      replicaCount: 3
      resources:
        requests:
          memory: "1Mi"
          cpu: "125m"
    # Configuracion del nodo procesador de datos
    ingest:
      # Numero de replicas definidas para el procesador de datos
      replicaCount: 1
      resources:
        requests:
          memory: "1Mi"
          cpu: "125m"
    # Configuracion del nodo coordinador
    coordinating:
      # Numero de replicas definidas para el coordinador
      replicaCount: 1
      resources:
        requests:
          memory: "1Mi"
          cpu: "125m"
    metrics:  # Habilitar el servicio de monitoreo
      enabled: true
      serviceMonitor:
        enabled: true
```

Además, Elasticsearch necesita de forma básica un clúster y una instancia de Kibana para comunicar con la base de datos como se muestra a continuación, en el cual son aplicados como un modelo básico predefinido (template) dentro de la carpeta de databases en un archivo único.

```
## elasticsearch.yaml
# Deploy an Elasticsearch cluster
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
  name: quickstart
```

```
spec:
  version: 8.4.3
  nodeSets:
  - name: default
    count: 1
    config:
      node.store.allow_mmap: false
---
# Deploy a Kibana instance
apiVersion: kibana.k8s.elastic.co/v1
kind: Kibana
metadata:
  name: quickstart
spec:
  version: 8.4.3
  count: 1
  elasticsearchRef:
    name: quickstart
```

También, es necesario correr los siguientes comandos en la terminal para tener localmente las imágenes a utilizar del clúster y la instancia de Kibana.

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:8.4.3
docker pull docker.elastic.co/kibana/kibana:8.4.3
```

Instalación del consumer y producer

Dentro de esta sección, es necesario crear una cuenta en Docker Hub para poder guardar las imágenes de las aplicaciones desarrolladas para el proyecto, el cual se usarán para las aplicaciones de "consumer" y "producer", utilizando lenguaje Python para otorgarles la lógca de recibir mensajes de la cola de RabbitMQ y procesarlas.

Luego de tener la cuenta habilitada, dentro de la carpeta general, se crearon dos carpetas para las aplicaciones respectivas, en donde cada uno tiene otra carpeta con los datos únicos de configuración de la aplicación de Python y un archivo de configuración "Docker file" para poder publicarlo al Docker Hub (ambos iguales).

Con respecto al archivo de configuración, se buscó la versión de Python a utilizar dentro de la aplicación, el cual se selecionó una versión comprimida (slim). Además, se siguieron las recomendaciones dadas por el profesor con la configuración necesaria para poder ejecutar la aplicación.

```
FROM python:3.10.7-slim-bullseye

WORKDIR /app

COPY app/. .

RUN pip install --no-cache-dir -r requirements.txt
```

```
CMD [ "python", "-u", "./app.py"]
```

Por consiguiente, dentro de la carpeta de configuración de Python, tiene un archivo de texto (requirements.txt) con las librerías necesarias para instalarlas de forma automática luego de publicar la aplicación a Docker Hub. Dicha librarías utilizadas son la de pika (para poder enviar o recibir mensajes del servidor a cliente y viceversa) y elasticsearch.

Luego de crear el archivo de texto, es necesario definir el código fuente de Python para poder conectarlo con la cola de RabbitMQ por medio de variables de entorno, los cuales se comentarán más adelante. También, definir el comportamiento lógico para procesar la información.

```
DATA = os.getenv('DATAFROMK8S')
RABBIT_MQ = os.getenv('RABBITMQ')
RABBIT_MQ_PASSWORD = os.getenv('RABBITPASS')
QUEUE_NAME = os.getenv('RABBITQUEUE')
```

Además, como se mencionó en la anterior sección sobre la carpeta de "application", es necesario crear modelos predefinidos (al igual que el archivo quickstart de Elasticsearch) del consumer y producer con sus respectivas variables de entorno y sus contraseñas (adquiridas por referencias a los secrets). Las variables de entorno es información guardada en variables que son visibles en todo momento y almacenadas en el sistema.

```
# consumer.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      containers:
      - name: consumer
        image: <user>/consumer
        env:
          - name: DATAFROMK8S
            value: "Hey"
          - name: RABBITMQ
            value: "databases-rabbitmq"
```

```
          - name: RABBITQUEUE
            value: "queue"
          - name: RABBITPASS
            valueFrom:
              secretKeyRef:
                name: databases-rabbitmq
                key: rabbitmq-password
                optional: false
          - name: ESENDPOINT
            value: quickstart-es-default
          - name: ESPASSWORD
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
                key: elastic
                optional: false
          - name: ESINDEX
            value: docidx
```

```
# producer.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: producer
  labels:
    app: producer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: producer
  template:
    metadata:
      labels:
        app: producer
    spec:
      containers:
      - name: producer
        image: <user>/producer
        env:
          - name: DATAFROMK8S
            value: "Hey"
          - name: RABBITMQ
            value: "databases-rabbitmq"
          - name: RABBITQUEUE
            value: "queue"
          - name: RABBITPASS
            valueFrom:
              secretKeyRef:
                name: databases-rabbitmq
                key: rabbitmq-password
```

```
                optional: false
          - name: ESENDPOINT
            value: quickstart-es-default
          - name: ESPASSWORD
            valueFrom:
              secretKeyRef:
                name: quickstart-es-elastic-user
                key: elastic
                optional: false
```

Por último, luego de tener los archivos listos para poder publicarlos a la nube, se siguen los siguientes comandos para poder realizarlo desde la terminal, donde el user es el nombre de usuario de Docker. Además, si la ejecución de comandos no sirve en la aplicación Lens, se puede utilizar el Command Prompt para ejecutarlos.

```
docker login                        # Inicio de sesión al repositorio de Docker
Hub
docker build -t <user>/producer .   # Construye la imagen
docker build -t <user>/consummer .  # Construye la imagen
docker images                       # Permite revisar la lista de imágenes
docker push <user>/producer         # Publica la imagen al repositorio en la nube
docker push <user>/consumer         # Publica la imagen al repositorio en la nube
helm install application application
```

Configuración de herramientas no automatizadas

Grafana

Configurar Grafana primero es necesario haber instalado el helm chart monitoring para tener un Pod funcional de grafana, en donde se selecciona para poder acceder al enlace con el puerto de Grafana. Al estar en la página, se le solicita un usuario, en donde el usuario predeterminado es "admin" y la contraseña se adquiere dentro del Secret "monitoring-grafana-admin", el cual es generado automáticamente.

Una vez dentro de la aplicación, es necesario agregar a Prometheus para monitorear y alertar eventos. Dentro de la esquina inferior izquierda, se encuentra el botón de configuracíón con forma de engranaje, al darle click, se despliega la página y pestaña de "Data sources", en donde se da la opción de crear uno nuevo. La nueva ventana despliega todas las opciones de "Data Source", el cual se le da a la opción de Prometheus. Ya en la opción de ingresar los datos del nuevo recurso, es necesario conocer el enlace del Prometheus creado por el helm chart cuyo enlace es el nombre del "Service" (servicio) "monitoring-kube-prometheus-prometheus" seguido de :9090 (puerto predeterminado) y se agrega.

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

Por último, es necesario buscar un tablero "dashboard" para poder ver todas las métricas que Prometheus monitorea en Grafana, cuyo tablero se busca uno apropiado con los datos requeridos. Dicho tablero es preferible de la misma página de Grafana para poder importarlo con un ID en la sección de "Dashboards".

Kibana

El acceso al servicio de Kibana se realiza a partir del puerto expuesto dado por el pod "quickstart-kb-...". Ya dentro de la página redirigida, el sistema solicita una cuenta, en donde el usuario predefinido es "elastic" y la contraseña se obtiene mediante la búsqueda del secreto (secret) "quickstart-es-elastic-user".

MariaDB

MariaDB no tiene una forma de accesar fácilmente a diferencia de otros servicios, por lo que hay que habilitar o exponer el puerto para poder acceder a la base de datos por medio de MySQL Workbench, de esta manera, el siguiente comando habilita dicho espacio.

```
kubectl port-forward databases-mariadb-0 3305:3306
```

Dentro del MySQL Workbench, se ingresan los datos del servidor, tales como nombre del servidor, nombre de host (127.0.0.1), puerto (3305), nombre de usuario (root) y contraseña (mariadbpass). Además, se debe de deshabilitar el uso de SSL por medio de la pestaña "Advanced/Others:" y se le agrega useSSL=1. Luego se puede probar la conexión e ingresar a la base de datos.

Luego de habilitar el acceso, se crearon dos tablas simples, una de persona y otra de carro. La tabla de persona tiene las siguientes columnas: identificador primario, nombre y cédula. La tabla de carro tiene las siguientes columnas: identificador primario, color, placa y llave foráneo a la tabla persona.

Conclusión

La migración de datos entre plataformas es un sistema muy versátil de mensajería que puede ser aplicado a diferentes campos para obtener resultados temporizados, transición de datos y manipulación de la misma. Además, se pueden observar el rendimiento que conlleva cada procesamiento de los datos en la cola por medio de las aplicaciones de monitoreo.

Referencias

- Repositorio
- Bitnami - MariaDB (2022) Github. Recuperdo de MariaDB
- Bitnami - RabbitMQ (2022) Github. Recuperado de Elasticsearch
- Docker (2022). Docker Hub. Recuperado de Docker Hub
- Elastic (2022). Elastic Cloud on Kubernetes [2.4]. Recuperado de Elasticsearch-operator
- Elastic - elasticsearch-docker (2017). Can't pull official images. Recuperado de elasticsearch-docker
- Elasticsearch (2022). Python Elasticsearch Client. Recuperado de Python Elasticsearch Client
- RabbitMQ (2022). RabbitMQ Tutorials. Recuperado de RabbitMQ Tutorials
- Regolith (2021). How do I disable the SSL requirement in MySQL Workbench? Answer. Recuperado de Regolith
- Pika (2022). Introduction to Pika. Recuperado de Pika