Configuración de una canalización de Kafka Connect en Kubernetes: paso a paso

Sumant Rana Seguir
17 de abril de 2020 · 6 min de lectura ★

Recientemente instalé y configuré Kafka Connect en GKE (usando gráficos de Helm) y creé una canalización de extremo a extremo para transferir datos desde una base de datos MySQL a un archivo de texto usando el conector JDBC y el conector de archivos.

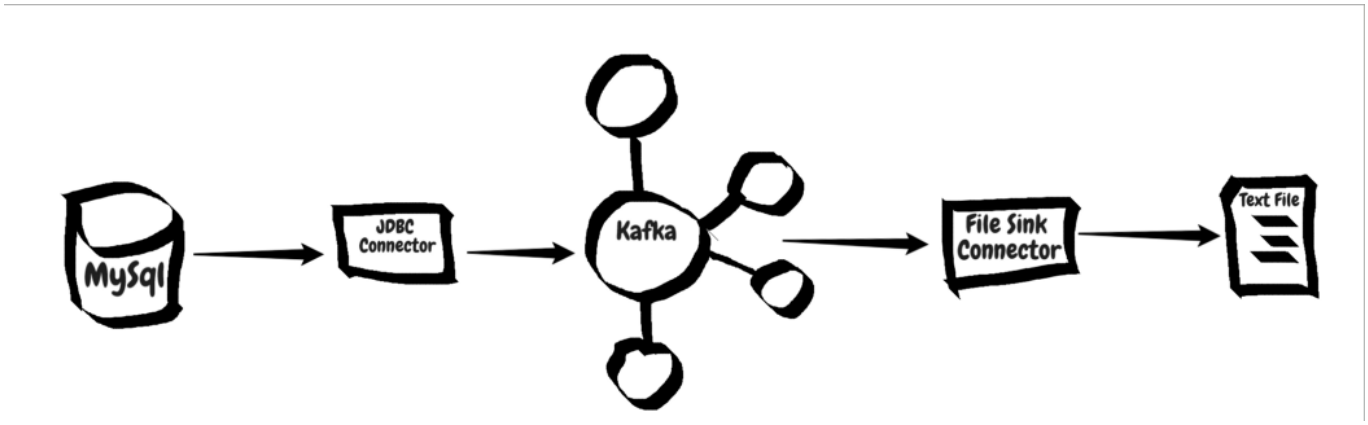
En este artículo, lo guiaré a través de los pasos y también señalaré los cambios de configuración que tuve que hacer para que funcione correctamente en Kubernetes. Los archivos están disponibles en el repositorio https://github.com/sumantrana/kafka-connect-pipeline .

Kafka Connect

Kafka Connect es un marco para mover datos hacia y desde otros sistemas utilizando Kafka como intermediario. Kafka Connect viene con una variedad de conectores incorporados que ayudan a conectarse a los sistemas existentes para transferir datos hacia y desde Kafka. Kafka Connect se puede escalar según los requisitos. Tiene una versión independiente y una versión distribuida.

Este artículo no pretende explicar en detalle Kafka Connect, su arquitectura y sus ventajas, sino que se centra más en su instalación y configuración en Kubernetes y hace que funcione de un extremo a otro en un entorno independiente. Para conocer en profundidad Kafka Connect, visite el sitio oficial aquí .

En este artículo, crearemos una canalización de extremo a extremo con MySQL como fuente de datos y un archivo de texto como receptor. El siguiente diagrama muestra el flujo de datos:



Prerrequisitos:

- Clúster de Kubernetes (GKE o equivalente)

Inicialmente comencé con un clúster de 3 nodos ("n1-estándar-1"), pero finalmente se quedó sin CPU. Así que tuve que aumentar el tamaño del grupo de nodos a 4.

- Administrador de paquetes Helm (siga las instrucciones en https://docs.helm.sh/using_helm/#installing-helm .

Poder instalar Kafka Connect y MySQL usando gráficos de timón.

- Cliente MySQL

Poder conectarse al servidor MySQL y crear tablas e insertar datos.

Instalar Kafka Connect

Clonar el repositorio de gráficos de timón de Kafka Connect

```
clon de git https://github.com/confluentinc/cp-helm-charts.git
```

Cambiar los valores predeterminados

El archivo values.yaml predeterminado está configurado para instalar un clúster Kafka de 3 nodos. Esto requeriría mucha más CPU y un grupo de nodos más grande o mejor. Por lo tanto, por el bien de este ejercicio, reduciremos Kafka a un solo clúster de nodos. Se requieren los siguientes cambios de configuración:

- cp-helm-charts / charts / cp-control-center / values.yaml

```
servidores: 1
corredores: 1
```

- cp-helm-charts / charts / cp-kafka / values.yaml

```
"offsets.topic.replication.factor": "1"
```

- cp-helm-charts / charts / cp-kafka-connect / values.yaml

```
"config.storage.replication.factor": "1"
"offset.storage.replication.factor": "1"
"status.storage.replication.factor": "1"
```

- cp-helm-charts / charts / cp-control-center / values.yaml

```
"replication.factor": "1"
```

Instalar la tabla de timones Kafka Connect

```
helm install <connect-release-name> cp-helm-chart
```

<connect-release-name> puede ser cualquier nombre dado a la instalación. Esto tendrá como prefijo todos los artefactos de Kubernetes implementados como parte de la instalación.

Espere hasta que se desplieguen todos los artefactos. Esto muestra el resultado de muestra del comando `kubectl get pods` cuando se instaló el gráfico de timón con `kafkaconnect` el <connect-release-name>:

```
NOMBRE ESTADO LISTO
kafkaconnect-cp-control-center-784846dd89-v88zs 1/1 Ejecutando
kafkaconnect-cp-kafka-0 2/2 Ejecutando
kafkaconnect-cp-kafka-connect-6bcbd5cbbf-zjz4q 2/2 Ejecutando
kafkaconnect-cp-kafka- rest-864cc8c67f-9g7fc 2/2 Ejecutando
kafkaconnect-cp-ksql-server-7594f6d6b7-4bpxx 2/2 Ejecutando
kafkaconnect-cp-schema-registry-59f5887595-xdz8p 2/2 Ejecutando
kafkaconnect-cp-zookeeper-0 2/2 Ejecutando
```

Instalar y configurar MySQL

Agregar repositorio de helm (repositorio de Google para MySQL)

```
helm repo agregar estable https://kubernetes-charts.storage.googleapis.com/
```

Instalar el gráfico MySQL

```
helm install <mysql-release-name> estable / mysql
```

De forma similar a Kafka Connect, <mysql-release-name> puede ser cualquier cadena para identificar de forma exclusiva esta instalación. Lo he utilizado `mysql` como nombre de versión para esta instalación.

Conectarse a MySQL

- Cree una regla de reenvío de puerto para el servicio MySQL que escucha en el puerto 3306

```
kubectl port-forward svc / <mysql-release-name> 3306
```

- Exportar variables de entorno requeridas

```
MYSQL_HOST = 127.0.0.1
MYSQL_PORT = 3306
MYSQL_ROOT_PASSWORD = $ (kubectl obtiene secreto - espacio de nombres
predeterminado mysql -o jsonpath = "{. Data.mysql-root-password}" |
base64 --decode; echo)
```

- Conectarse a MySQL

```
mysql -h $ {MYSQL_HOST} -P $ {MYSQL_PORT} -u root -p $
{MYSQL_ROOT_PASSWORD}
```

- Usando el símbolo del sistema de MySQL, cree una base de datos y una tabla

```
crear prueba de base de datos;
prueba de uso;

create table`students` (
  `name` varchar (50) DEFAULT NULL,
  ` id` int (11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
);
```

- Inserte algunos datos predeterminados en la tabla

```
insertar en los estudiantes (nombre) valores ('Aaren');
insertar en los estudiantes (nombre) valores ('Aarika');
insertar en los estudiantes (nombre) valores ('Abagael');
```

Configurar la conexión independiente

- Conéctese al servidor de conexión de Kafka

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- Opcional: instale vi o nano editor y descomprímalo (no está presente por defecto). Alternativamente, los archivos se pueden copiar al pod después de editarlos en el sistema local

```
apt-get update
apt install nano
apt install descomprime
```

- Edite el archivo de propiedades independiente

```
vi /etc/schema-registry/connect-avro-standalone.properties
```

- Hacer cambios de acuerdo al medio ambiente

```
bootstrap.servers = <nombre de lanzamiento de conexión> -cp-kafka:
9092

key.converter.schema.registry.url = http: // <connect-release-name> -
cp-schema-registry: 8081

value.converter.schema.registry.url = http: // <connect-release-name>
-cp-schema-registry: 8081

rest.host.name = localhost

rest.port = 9083
(Se queja de que el puerto original 8083 ya está en uso)
```

Configurar el conector de origen JDBC

- Conéctese al servidor de conexión de Kafka (si aún no está conectado)

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- Cree el archivo de propiedades del conector de origen JDBC:

```
vi jdbc-connect.properties
```

- Agregue las siguientes propiedades

```
nombre = prueba-jdbc

connector.class = io.confluent.connect.jdbc.JdbcSourceConnector

tasks.max = 1

connection.url = jdbc: mysql: // <mysql-release-name>: 3306 / test?
user = root & password = <mysql-root-password>

incrementing.column.name = id

modo = incrementando

topic.prefix = test-
```

Ejecute y verifique el conector de origen JDBC

- Conéctese al servidor de conexión de Kafka (si aún no está conectado)

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- Descargue y copie el jar del conector java de MySQL

```
wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-
java-5.1.48.zip

descomprimir mysql-connector-java-5.1.48.zip
```

- Copie el contenedor del conector en la ubicación correcta

```
cp conector-mysql-java-5.1.48 / conector-mysql-java-5.1.48 -bin.jar /
usr / share / java / kafka-connect-jdbc /
```

- Ejecute el conector independiente para cargar datos de MySQL a Kafka mediante el conector JDBC

```
connect-standalone /etc/schema-registry/connect-avro-
standalone.properties jdbc-connect.properties
```

- Configurar kafka-avro-console-consumer

```
vi / usr / bin / kafka-avro-console-consumer

DEFAULT_SCHEMA_REGISTRY_URL = ”- propiedad schema.registry.url =
http: // localhost : 8081 "
```

Reemplazar **`localhost`** con **`<connect-release-name>-cp-schema-registry`**

- Verifique que los datos se hayan cargado en el tema de Kafka

```
kafka-avro-console-consumer --bootstrap-server = <connect-release-
name> -cp-kafka: 9092 --topic = test-students --desde-principio
```

La salida debería verse así:

```
{"nombre": {"cadena": "Aaren"}, "id": 1}
{"nombre": {"cadena": "Aarika"}, "id": 2}
{"nombre": {"cadena ":" Abagael "}," id ": 3}
```

Esto asegura que el conector de origen jdbc se ejecutó correctamente y que los datos se cargan en el tema de kafka "estudiantes de prueba".

> *Si usamos el kafka-console-consumer genérico, aún podremos ver el resultado. Pero no estará estructurado / formateado y también puede contener caracteres adicionales. Un ejemplo:*

```
Adara y
Adda (
Addi *
```

Configurar el conector del receptor de archivos

- Conéctese al servidor de conexión de Kafka (si aún no está conectado)

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- Crear un archivo de propiedades para el conector File Sink

```
vi file-sink-connector.properties
```

- Agregue las siguientes propiedades al archivo

```
name = test-file-sink
connector.class = FileStreamSink
tasks.max = 1
file = / tmp / test.sink.txt
topics = test-estudiantes
```

Ejecutar y verificar el conector del receptor de archivos

- Conéctese al servidor de conexión de Kafka (si aún no está conectado)

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- Ejecute el conector independiente para recuperar datos del tema de Kafka en un archivo de texto mediante File SinkConnector

```
connect-standalone /etc/schema-registry/connect-avro-
standalone.properties file-sink-connector.properties
```

> *Si el proceso se queja del puerto REST "Dirección ya en uso", actualice el archivo "* ***/etc/schema-registry/connect-avro-standalone.properties*** *", cambie* ***rest.port*** *a* ***9084*** *y vuelva a ejecutar.*

- Verifica los datos

```
cat /tmp/test.sink.txt
```

La salida debe mostrar los registros presentes en la tabla de la base de datos.

```
Estructura {nombre = Aaren, id = 1}
Estructura {nombre = Aarika, id = 2}
Estructura {nombre = Abagael, id = 3}
```

Verificar actualizaciones en vivo

Conéctese al servidor de conexión de Kafka utilizando tres shells diferentes.

```
kubectl exec -c cp-kafka-connect-server -it <pod de conexión kafka> -
/ bin / bash
```

- En el primero, ejecute el conector jdbc.

```
connect-standalone /etc/schema-registry/connect-avro-
standalone.properties jdbc-connect.properties
```

- En el segundo, ejecute el conector del receptor de archivos.

```
connect-standalone /etc/schema-registry/connect-avro-
standalone.properties file-sink-connector.properties
```

- En el tercero, **siga** el archivo **/tmp/test.sink.txt**

```
tail -f /tmp/test.sink.txt
```

Agregar nuevos datos

- Abra una nueva terminal y conéctese a MySQL

```
mysql -h $ {MYSQL_HOST} -P $ {MYSQL_PORT} -u root -p $
{MYSQL_ROOT_PASSWORD}
```

- Inserte una nueva fila en la tabla de la base de datos MySQL.

```
insertar en los estudiantes (nombre) valores ('LiveTest');
```

Verificar resultados

- Verifique la terminal que se encuentra en el archivo **/tmp/test.sink.txt.**
- Los datos recién insertados deberían estar visibles.

Kafka Connect MySQL Kubernetes Gráfico de timón


App Store


Google Play