

데이터 관계 기술 중간고사 대체 과제 데모 및 발표

소프트웨어학과 201835503 이지민

Requirements

Overlook

목적

CDC란?

Debezium

Kafka

목차

1. SourceDB → Connector → Kafka
2. Kafka → JDBC connector → SinkDB1
- 2-1. Kafka → JDBC connector → SinkDB2
3. Spring → MySQL 연동
4. React → Spring 연동
5. TroubleShooting

1. sourceDB → connector → Kafka

- A. docker-compose.yml 작성 및 컨테이너 설치
docker-compose 이해

B. DB 설정

데이터베이스 및 테스트용 테이블 생성

mysql 사용자 추가 및 권한 확인

C. Debezium Connector for MySQL 플러그인 설치

Debezium Connector 설치

plugin 경로 설정

D. kafka connect 실행

Distributed Mode로 kafka connect 실행

E. Source Connector 생성하기

Kafka Connect 클러스터 확인

Rest API로 connector 생성

Connector 설정 이해

connector 목록/상세 정보 확인

Topic 목록 확인

F. 레코드 확인

테스트 데이터 입력

콘솔 컨슈머 확인

2. Kafka → JDBC connector → SinkDB

- A. SinkDB 컨테이너 생성 (docker-compose.yml 작성)

B. DB 설정

데이터베이스 및 테스트용 테이블 생성

mysql 사용자 추가 및 권한 확인

C. 4. Kafka JDBC Connector (Source and Sink) 설치

JDBC Connector 설치

plugin 경로 확인

D. Kafka connect 실행

E. Sink Connector 생성하기

MySQL 커넥터 플러그인 확인

Rest API로 connector 생성

상세 Configuration 확인

Kafka Connect REST API

F. MySQL Sink 확인

ADD

UPDATE

DELETE ← Trouble Shooting #5

2.1 TargetDB2 : SinkDB2 및 custom table 생성

- A. MYSQL 테이블 생성 및 권한설정

B. SinkDB2 Connector 생성

C. Insert 수행, Kafka Consumer로 Topic 저장 확인

성공!!

3. Spring → MySQL 연동

Spring Server 1 → SourceDB

Spring Server 2 → SinkDB1

Spring Server 3 → SinkDB2

4. React → Spring 연동

React 서버에서 Spring 서버 1, 2, 3를 각각 연동하였다.

TroubleShooting

#1 Docker-compose m1침 에러

#2 Docker-compose Volume error

#3 Linux vim

#4 JDBC connector 설치시 No suitable driver found

#5 DELETE의 경우 CDC 반영이 안된다

#5-1 __deleted column 생성으로 인한 TOPIC 오류

#6 (SINK) 새로운 TABLE 생성시 전체TABLE 업데이트 안됨

8. (REACT, SPRING) Axios 사용시 CORS 에러

9. (SPRING) DB TABLE명 지정 에러

후기

▼ Requirements

1. (필수) 카카오엔터프라이즈 CDC Tutorial 단계별 데모 및 발표
2. (필수) 자신이 만든 custom table을 Source에 생성하고, target table에 데이터 변경사항을 Capture하는 부분을 데모하고 발표
3. (필수) 카카오엔터프라이즈 인프라 사용을 local PC나 다른 인프라로 변경하고 CDC 관련 프로그램이 동작하도록 수정한다.
4. (Advantage) 필수사항 외 사항 추가점수 부여
 - a. TargetDB 1개 이상인 경우
 - b. 파일럿 프로그램 외에 Kafka Connect API를 통한 CDC 기능 구현
 - c. 프로그램 언어 : Any 가능

MySQL → Kafka → SinkDB

CDC Pilot Demo

Overlook

CDC Demo

Show Sink 1 Show Sink 2

Create New Data In

SourceDB.CUSTOM_TABLE

id :
title :
name :

Create Data

Delete Data In

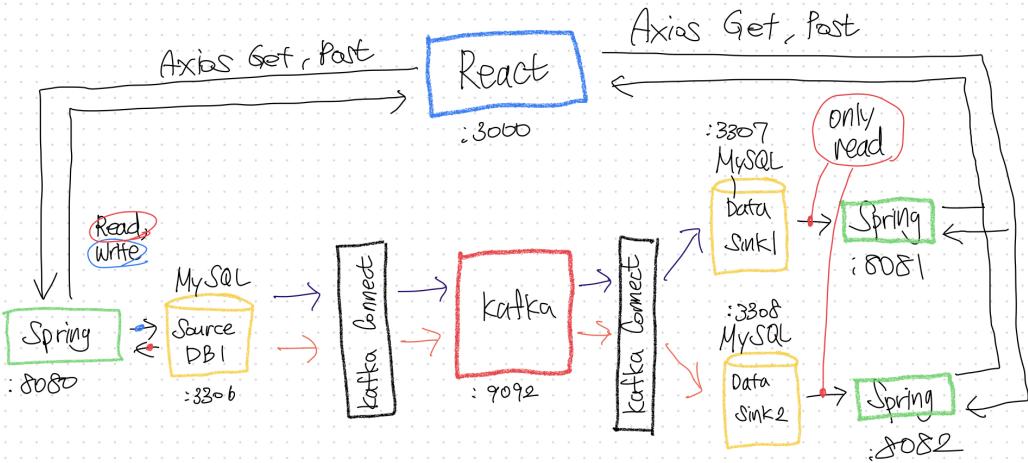
SourceDB.CUSTOM_TABLE

Delete Row

SinkDB2 CUSTOM_TABLE :

```
{"id":2,"title":"test","name":"adfsfsfn"}  
{"id":4,"title":"del","name":"asffffn"}  
{"id":6,"title":"test","name":"adfssfn"}
```

Load sink data



목적

1. Docker, Kafka, Debezium Engine을 기반으로 CDC 구현.

기반 CDC 구조를 구축한 뒤 Source Database작과 변동사항이 적용된 Sink Database 확인의 접근이 가능하도록

2. Front & Back End 연결 : React → Spring → DB 구조를 구축하는 것.

CDC :

MySQL → kafka Connect(Source Connector, Debezium) → Kafka → kafka Connect(JDBC Sink Connector) → Mysql

Front & Back :

React(Axios) → Spring1, Spring2, Spring3(JPA) → MySQL1(SourceDB), MySQL2(SinkDB1), MySQL3(SinkDB2)

CDC란?

Change Data Capture. 데이터베이스에서 데이터 변경 발생 시 변경된 데이터를 사용하여 동작을 취할 수 있도록 지원하는 기능

<https://morphus.github.io/mysql/cdc/debezium/2020/05/23/mysql-cdc-with-debezium-1.html>

// CDC가 필요한 이유

Debezium

Apache Kafka를 기반으로 구축되었으며,

특정 DBMS를 모니터링하는 Kafka Connect 호환 커넥터를 제공하기 위해 시작된 프로젝트

다양한 DBMS의 변경 사항을 캡쳐하고 유사한 구조의 변경 이벤트를 produce 하는 커넥터 라이브러리를 구축

Kafka

- 아파치 카프카는 빠르고 확장 가능한 작업을 위해 데이터 피드의 분산 스트리밍, 파이프 라이닝 및 재생을 위한 실시간 스트리밍 데이터를 처리하기 위한 목적으로 설계 된 오픈 소스 분산형 게시-구독 메시징 플랫폼이다

Kafka 용어 정리

- **Broker** : 카프카 애플리케이션이 설치되어 있는 서버 또는 노드
- **Topic** : 프로듀서와 컨슈머들이 카프카로 보낸 자신들의 메세지를 구분하기 위한 네임으로 사용
- **Partition** : 병렬처리가 가능하도록 토픽을 나눌 수 있고, 많은 양의 메세지 처리를 위해 파티션의 수를 늘려줄 수 있다.
- **Producer** : 메세지를 생산하여 브로커의 토픽 이름으로 보내는 서버 또는 애플리케이션 등을 말한다.
- **Consumer** : 브로커의 토픽 이름으로 저장된 메세지를 가져가는 서버 또는 애플리케이션 등을 말한다.
- kafka 이해 : <https://velog.io/@shinmj1207/Apache-Kafka-메세징-시스템과-Kafka의-작동-방식>

목차

1. SourceDB → Connector → Kafka
2. Kafka → JDBC connector → SinkDB1
- 2-1. Kafka → JDBC connector → SinkDB2
3. Spring → MySQL 연동
4. React → Spring 연동
5. TroubleShooting

1. sourceDB → connector → Kafka

A. docker-compose.yml 작성 및 컨테이너 설치

```

version: '3'
services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: sasd
      MYSQL_USER: mysqluser
      MYSQL_PASSWORD: mysqlpw
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    volumes:
      - /Users/jiminlee/TempCodes/CDC/mysql/data:/var/lib/mysql

  zookeeper:
    platform: linux/amd64/v8 //platform: linux/amd64/
    container_name: zookeeper
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"

  kafka:
    container_name: kafka
    image: wurstmeister/kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 127.0.0.1
      KAFKA_ADVERTISED_PORT: 9092
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
  
```

▼ docker-compose 이해

- **version**: docker-compose 버전을 지정한다. 여기서는 2라고 기술했다.
- **services**: docker-compose의 경우 docker 컨테이너로 수행될 서비스들은 services 하위에 기술한다.
- **hostname**: container에 이름을 지정해주는 것이라고 생각할 수 있다.
- **mysql**: 서비스 이름을 mysql로 작성했다. service 하위에 작성하면 서비스 이름으로 동작한다.
- **image**: 사용하는 서비스의 버전을 명시할 수 있는 부분이다.
 - 참고로 실전에서 사용하려면 latest라는 태그를 사용하지 말고, 정확히 원하는 버전을 기술해서 사용하길 추천한다.
 - latest라고 태그를 지정하면, 맵 컨테이너를 실행할 때마다 최신버전을 다운받아 실행하므로 변경된 버전으로 인해 원하지 않는 결과를 볼 수 있다. (주의 !!!)
- **volume**: 마운트할 볼륨을 지정한다. 로컬에 있는 directory를 container 안에 마운트한다고 생각하면 된다.
- **platform**: 이미지가 어떤 플랫폼 대상인지 지정할 수 있다.
- **environment**: 환경 변수를 설정할 수 있다. environment 하위에 필요한 환경을 작성하자.
 - **mysqlpassword**: DB root 계정의 비밀번호를 설정한다.
- **ports**: kafka 브로커의 포트를 의미한다. 외부포트:컨테이너내부포트 형식으로 지정한다.
- **volumes**: 컨테이너와 호스트 서버 디렉토리 연결. 여러개 설정 가능.: 기준 왼쪽이 호스트 서버, 오른쪽이 컨테이너 서버
- **kafka**: kafka 브로커 이름을 지정한다.
- **depends_on**: docker-compose에서는 서비스들의 우선순위를 지정해 주기 위해서 depends_on 을 이용한다.
 - zookeeper 라고 지정하였으므로, kafka는 zookeeper가 먼저 실행되어 있어야 컨테이너가 올라오게 된다.
- **KAFKA_ZOOKEEPER_CONNECT**: kafka가 zookeeper에 커넥션하기 위한 대상을 지정한다. 여기서는 zookeeper(서비스이름):2181(컨테이너내부포트)로 대상을 지정했다.

docker-compose 파일 실행, 컨테이너 실행

```
docker-compose -f docker-compose.yml up -d
```

```
Started 0.9s          Started 0.9s
  #: zookeeper The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested  0.0s
  #: Container kafka                                         Started 1.1s
  #: Container kafka                                         Started 1.1s
  #: /TempCodes/CDC                                         09:3
  #: /TempCodes/CDC                                         09:30:1
  #: /TempCodes/CDC                                         09:30:1
  #: docker-compose -f docker-compose.yml up -d
  #: [+] Running 3/3
  #:   #: Container mysql      Running                         0.0s
  #:   #: Container zookeeper  Started                         21.2s
  #:   #: Container kafka     Started                         11.0s
  #: /TempCodes/CDC                                         22s  09:40:26 PM
  #:
```

진행하다 위와 같이 에러가 나서 compose 파일을 수정해줬다 - #1 확인

```
docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
4059b947b0fa        wurstmeister/kafka    "start-kafka.sh"    6 minutes ago     Exited (1) 6 minutes ago
3c6e96f52e0         wurstmeister/zookeeper  "/bin/sh -c '/usr/sb..."  6 minutes ago     Up 6 minutes       22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp
8f86d04dd3         mysql:8.0              "docker-entrypoint.s..." 16 minutes ago    Up 16 minutes      0.0.0.0:3306->3306/tcp, 33060/tcp
ad24e122598f        mariadb:latest        "docker-entrypoint.s..." 2 weeks ago       Exited (0) 7 days ago
NAMES                kafka
zookeeper
mysql
mariadb
```

설치된 컨테이너 확인: `docker ps -a`

B. DB 설정

데이터베이스 및 테스트용 테이블 생성

```
mysql -u root -p
create database testdb;
use testdb;

CREATE TABLE accounts (
    account_id VARCHAR(255),
```

```

    role_id VARCHAR(255),
    user_name VARCHAR(255),
    user_description VARCHAR(255),
    update_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (account_id)
);

```

MySQL Container 접속 후 - `docker exec -it mysql /bin/bash`

- `-it` 는 표준입출력을 열고 tty를 통해 접속하겠다는 의미
- 컨테이너명 뒤에는 접속할 때 어떤쉘을 사용할지 지정할 수 있다. `bash` 가 표준이기에 bash 를 사용

SQL문으로 TABLE을 생성해준다.

```

$ docker exec -it mysql /bin/bash
bash-4.4# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 

```

```

mysql> create database testdb;
Query OK, 1 row affected (0.02 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| testdb |
+-----+
5 rows in set (0.00 sec)

mysql> use testdb;
Database changed
mysql> CREATE TABLE accounts (
    >     account_id VARCHAR(255),
    >     role_id VARCHAR(255),
    >     user_name VARCHAR(255),
    >     user_description VARCHAR(255),
    >     update_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    >     PRIMARY KEY (account_id)
    > );
Query OK, 0 rows affected (0.04 sec)

```

mysql 사용자 추가 및 권한 확인

mysqluser에게 모든 권한을 부여하는 과정이다.

```

use mysql;

// mysqluser 가 추가 되어 있는지 확인
select host, user from user;

// mysqluser 없으면 생성
CREATE USER 'mysqluser'@'%' IDENTIFIED BY 'mysqlpw';
// mysqluser 에게 권한 부여
GRANT ALL PRIVILEGES ON *.* TO 'mysqluser'@'%';

FLUSH PRIVILEGES;

```

```

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select host, user from user;
+-----+-----+
| host | user |
+-----+-----+
| %    | mysqluser |
| %    | root      |
| localhost | mysql.infoschema |
| localhost | mysql.session |
| localhost | mysql.sys   |
| localhost | root      |
+-----+-----+
6 rows in set (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'mysqluser'@'%';
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

```

C. Debezium Connector for MySQL 플러그인 설치

Debezium Connector 설치

<https://debezium.io/releases/1.5/> → `debezium-connector-mysql-1.5.4.Final-plugin.tar.gz` 을 다운로드 받는다.

`/opt/kafka_2.13-2.8.1/` 경로에 미리 `connectors` 폴더를 만들자.

```

> docker exec -it kafka /bin/bash
root@4059b947b0fa:/# cd /opt/kafka_2.13-2.8.1/
root@4059b947b0fa:/opt/kafka_2.13-2.8.1# mkdir connectors
root@4059b947b0fa:/opt/kafka_2.13-2.8.1#

```

로컬 컴퓨터에서 kafka 컨테이너로 `debezium-connector-mysql-1.5.4.Final-plugin.tar.gz` 를 업로드한다.

```
docker cp debezium-connector-mysql-1.5.4.Final-plugin.tar.gz kafka:/opt/kafka_2.13-2.8.1/connectors/debezium-connector-mysql-2.1.4.Fin
```

파일 압축을 푸다

```

cd /opt/kafka_2.13-2.8.1/connectors
tar -zvxf debezium-connector-mysql-1.5.4.Final-plugin.tar.gz

```

```

root@4059b947b0fa:/# cd /opt/kafka_2.13-2.8.1/connectors
tar -zvxf debezium-connector-mysql-1.5.4.Final-plugin.tar.gz
debezium-connector-mysql/CHANGELOG.md
debezium-connector-mysql/CONTRIBUTE.md
debezium-connector-mysql/COPYRIGHT.txt
debezium-connector-mysql/LICENSE-3rd-PARTIES.txt
debezium-connector-mysql/LICENSE.txt
debezium-connector-mysql/README.md
debezium-connector-mysql/README_ZH.md
debezium-connector-mysql/debezium-core-1.5.4.Final.jar
debezium-connector-mysql/debezium-api-1.5.4.Final.jar
debezium-connector-mysql/guava-30.0-jre.jar
debezium-connector-mysql/failureaccess-1.0.1.jar
debezium-connector-mysql/debezium-ddl-parser-1.5.4.Final.jar
debezium-connector-mysql/antlr4-runtime-4.7.2.jar
debezium-connector-mysql/mysql-binlog-connector-java-0.25.1.jar
debezium-connector-mysql/mysql-connector-java-8.0.21.jar
debezium-connector-mysql/debezium-connector-mysql-1.5.4.Final.jar
root@4059b947b0fa:/opt/kafka_2.13-2.8.1/connectors#

```

plugin 경로 설정

카프카 컨테이너에 접속하여 `/opt/kafka/config/connect-distributed.properties` 파일을 수정한다. 파일을 수정한 뒤에는 카프카 컨테이너를 재시작해야 플러그인 경로가 정상 반영된다.

```

// 원래 경로
#plugin.path=

// 수정 경로
plugin.path=/opt/kafka_2.13-2.8.1/connectors

```

```

# Set to a list of filesystem paths separated by commas (,) to enable class loading isolation for plugins
# (connectors, converters, transformations). The list should consist of top level directories that include
# any combination of:
# a) directories immediately containing jars with plugins and their dependencies
# b) uber-jars with plugins and their dependencies
# c) directories immediately containing the package directory structure of classes of plugins and their dependencies
# Examples:
# plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
plugin.path=/opt/kafka_2.13-2.8.1/connectors
-- INSERT --

```

86,45 Bot

D. kafka connect 실행

Distributed Mode로 kafka connect 실행

분산모드(distributed) 카프카 커넥트를 실행한다. 분산모드는 2개 이상의 커넥트를 한 개의 클러스터를 묶어서 운영한다.

```

connect-distributed.sh /opt/kafka/config/connect-distributed.properties

// 정상 실행 시 INFO Kafka Connect started (org.apache.kafka.connect.runtime.Connect:57) 확인 가능

```

E. Source Connector 생성하기

Kafka Connect 클러스터 확인

다른 탭을 새로 열어 카프카에 접속 후 플러그인 목록을 조회한다. `io.debezium.connector.mysql.MySqlConnector` 가 있어야 한다.

```
curl --location --request GET 'localhost:8083/connector-plugins'
```

```

root@4059b947b0fa:/# curl --location --request GET 'localhost:8083/connector-plugins'
[{"class": "io.debezium.connector.mysql.MySqlConnector", "type": "source", "version": "1.5.4.Final"}, {"class": "org.apache.kafka.connect.file.FileStreamSinkConnector", "type": "sink", "version": "2.8.1"}, {"class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "type": "source", "version": "2.8.1"}, {"class": "org.apache.kafka.connect.mirror.MirrorCheckpointConnector", "type": "source", "version": "1"}, {"class": "org.apache.kafka.connect.mirror.MirrorHeartbeatConnector", "type": "source", "version": "1"}, {"class": "org.apache.kafka.connect.mirror.MirrorSourceConnector", "type": "source", "version": "1"}]root@4059b947b0fa:#

```

Rest API로 connector 생성

rest api 를 호출하여 connector 를 생성하자.

```

curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "source-test-connector",
  "config": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "mysql",
    "database.port": "3306",
    "database.user": "mysqluser",
    "database.password": "mysqlpw",
    "database.server.id": "184054",
    "database.server.name": "dbserver1",
    "database.allowPublicKeyRetrieval": "true",
    "database.include.list": "testdb",
    "database.history.kafka.bootstrap.servers": "kafka:9092",
    "database.history.kafka.topic": "dbhistory.testdb",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap,addTopicPrefix",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.addTopicPrefix.regex": "(.*",
    "transforms.addTopicPrefix.replacement": "$1"
  }
}'

```

```

curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "source-test-connector",
  "config": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "mysql",

```

```

    "database.port": "3306",
    "database.user": "mysqluser",
    "database.password": "mysqlpw",
    "database.server.id": "184054",
    "database.server.name": "observer1",
    "database.allowPublicKeyRetrieval": "true",
    "database.include.list": "testdb",
    "database.history.kafka.bootstrap.servers": "kafka:9092",
    "database.history.kafka.topic": "dbhistory.testdb",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap,addTopicPrefix",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.addTopicPrefix.regex": "(.*",
    "transforms.addTopicPrefix.replacement": "$1",
    "topic.prefix": "testdb",
    "schema.history.internal.kafka.topic": "schemahistory.fullfillment",
    "schema.history.internal.kafka.bootstrap.servers": "kafka:9092",
    "include.schema.changes": "true"
}
}

```

▼ Connector 설정 이해

이러한 설정 중 일부는 일반적인 설정으로서, 모든 커넥터에 대해 지정해야 합니다. 예를 들면 다음과 같습니다.

- `connector.class` 는 커넥터의 Java 클래스입니다.
- `tasks.max` 는 이 커넥터에 대해 생성되어야 할 태스크의 최대 수입니다.

다른 설정은 Debezium MySQL 커넥터에만 해당됩니다.

- `database.hostname` 은 Aurora 데이터베이스의 작성자 인스턴스 엔드포인트를 포함합니다.
- `database.server.name` 은 데이터베이스 서버의 논리적 이름입니다. 이 설정은 이름은 Debezium에서 생성한 Kafka 주제의 이름에 사용됩니다.
- `database.include.list` 는 지정한 서버에서 호스팅하는 데이터베이스의 목록을 포함합니다.
- `database.history.kafka.topic` 은 데이터베이스 스키마 변경을 추적하기 위해 Debezium에서 내부적으로 사용하는 Kafka 주제입니다.
- `database.history.kafka.bootstrap.servers` 는 MSK 클러스터의 부트스트랩 서버를 포함합니다.
- 마지막의 8개 줄(`database.history.consumer.*` 및 `database.history.producer.*`)은 데이터베이스 기록 주제를 액세스하기 위한 IAM 인증을 활성화합니다.

Amazon MSK Connect – Apache Kafka 클러스터로 데이터 전달 서비스 출시 | Amazon Web Services
 Apache Kafka는 실시간 스트리밍 데이터 파이프라인 및 애플리케이션 구축을 위한 오픈 소스 플랫폼입니다. re:Invent 2018에서 AWS는 스트리밍 데이터의 프로세싱을 위해 Apache Kafka를 사용하는 애플리케이션을 쉽게 구축 및 실행할 수 있게 해 주는 완전관리형 서비스인 Amazon Managed Streaming for Apache Kafka를 발표했습니다. Apache  <https://aws.amazon.com/ko/blogs/korea/introducing-amazon-msk-connect-stream-data-to-and-from-your-apache-kafka-clusters-using-managed-connectors/>



```

Connector","type":"source","version":"1"}]root@4059b947b0fa:/#
root@4059b947b0fa:/# curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "source-test-connector",
  "config": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "mysql",
    "database.port": "3306",
    "database.user": "mysqluser",
    "database.password": "mysqlpw",
    "database.server.id": "184054",
    "database.server.name": "dbserver1",
    "database.allowPublicKeyRetrieval": "true",
    "database.include.list": "testdb",
    "database.history.kafka.bootstrap.servers": "kafka:9092",
    "database.history.kafka.topic": "dbhistory.testdb",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap,addTopicPrefix",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.addTopicPrefix.regex": "(.*",
    "transforms.addTopicPrefix.replacement": "$1"
  }
}
{"name": "source-test-connector", "config": {"connector.class": "io.debezium.connector.mysql.MySqlConnector", "tasks.max": "1", "database.hostname": "mysql", "database.port": "3306", "database.user": "mysqluser", "database.password": "mysqlpw", "database.server.id": "184054", "database.server.name": "dbserver1", "database.allowPublicKeyRetrieval": "true", "database.include.list": "testdb", "database.history.kafka.bootstrap.servers": "kafka:9092", "database.history.kafka.topic": "dbhistory.testdb", "key.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter.schemas.enable": "true", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "value.converter.schemas.enable": "true", "transforms": "unwrap,addTopicPrefix", "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState", "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter", "transforms.addTopicPrefix.regex": "(.*", "transforms.addTopicPrefix.replacement": "$1", "name": "source-test-connector"}, "tasks": [], "type": "source"}]root@4059b947b0fa:/# 

```

connector 목록/상세 정보 확인

```

# 목록
curl --location --request GET 'http://localhost:8083/connectors'

# 상세정보
curl --location --request GET 'http://localhost:8083/connectors/{connector-name}/config' \
--header 'Content-Type: application/json'

curl --location --request GET 'http://localhost:8083/connectors/source-test-connector/config' \
--header 'Content-Type: application/json'

#삭제
curl --location --request DELETE 'http://localhost:8083/connectors/source-test-connector'

```

```

root@4059b947b0fa:/# curl --location --request GET 'http://localhost:8083/connectors' \
["source-test-connector"]root@4059b947b0fa:/# curl --location --request GET 'http://localhost:8083/connectors/source-test-connector/config' \
--header 'Content-Type: application/json'
{"connector.class": "io.debezium.connector.mysql.MySqlConnector", "database.allowPublicKeyRetrieval": "true", "database.user": "mysqluser", "database.server.id": "184054", "tasks.max": "1", "database.history.kafka.bootstrap.servers": "kafka:9092", "database.history.kafka.topic": "dbhistory.testdb", "transforms": "unwrap,addTopicPrefix", "database.server.name": "dbserver1", "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState", "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RegexRouter", "database.port": "3306", "key.converter.schemas.enable": "true", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter.schemas.enable": "true", "value.converter.schemas.enable": "true", "value.converter.name": "source-test-connector", "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState", "value.converter.regex": "(.*", "value.converter.replacement": "$1", "name": "source-test-connector"}, "tasks": [], "type": "source"}]root@4059b947b0fa:/# 

```

Topic 목록 확인

```
kafka-topics.sh --list --bootstrap-server localhost:9092
```

```
"data":[]_consumer_offsetsconnect-configsconnect-offsetsconnect-statusdbhistory.testdbdbserver1root@4059b947b0fa:/# 
```

F. 레코드 확인

테스트 데이터 입력

```

INSERT INTO accounts VALUES ("123456", "111", "Susan Cooper", "God", "2021-08-16 10:11:12");
INSERT INTO accounts VALUES ("123457", "111", "Rick Ford", "mistakes", "2021-08-16 11:12:13");
INSERT INTO accounts VALUES ("123458", "999", "Bradley Fine", "face", "2021-08-16 12:13:14");

```

```

mysql> use testDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> INSERT INTO accounts VALUES ("123456", "111", "Susan Cooper", "God", "2021-08-16 10:11:12");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO accounts VALUES ("123457", "111", "Rick Ford", "mistakes", "2021-08-16 11:12:13");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO accounts VALUES ("123458", "999", "Bradley Fine", "face", "2021-08-16 12:13:14");
Query OK, 1 row affected (0.00 sec)

mysql> show * from accounts;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '* from accounts' at line 1
mysql> select * from accounts;
+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description |
+-----+-----+-----+-----+
| 123456 | 111 | Susan Cooper | God |
| 123457 | 111 | Rick Ford | mistakes |
| 123458 | 999 | Bradley Fine | face |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

콘솔 컨슈머 확인

```
kafka-console-consumer.sh --topic dbserver1.testdb.accounts --bootstrap-server localhost:9092 --from-beginning
```

```

root@4059b947b0fa:/# kafka-console-consumer.sh --topic dbserver1.testdb.accounts --bootstrap-server localhost:9092 --from-beginning
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": false, "field": "account_id"}, {"type": "string", "optional": true, "field": "role_id"}, {"type": "string", "optional": true, "field": "user_name"}, {"type": "string", "optional": true, "field": "user_description"}], "type": "io.debezium.time.Timestamp", "version": 1, "default": 0, "field": "update_date"}, {"optional": false, "name": "dbserver1.testdb.accounts.Value"}, "payload": {"account_id": "123456", "role_id": "111", "user_name": "Susan Cooper", "user_description": "God", "update_date": "1629108672000"}}
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": false, "field": "account_id"}, {"type": "string", "optional": true, "field": "role_id"}, {"type": "string", "optional": true, "field": "user_name"}, {"type": "string", "optional": true, "field": "user_description"}], "type": "io.debezium.time.Timestamp", "version": 1, "default": 0, "field": "update_date"}, {"optional": false, "name": "dbserver1.testdb.accounts.Value"}, "payload": {"account_id": "123457", "role_id": "111", "user_name": "Rick Ford", "user_description": "mistakes", "update_date": "1629112333000"}}
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": false, "field": "account_id"}, {"type": "string", "optional": true, "field": "role_id"}, {"type": "string", "optional": true, "field": "user_name"}, {"type": "string", "optional": true, "field": "user_description"}], "type": "io.debezium.time.Timestamp", "version": 1, "default": 0, "field": "update_date"}, {"optional": false, "name": "dbserver1.testdb.accounts.Value"}, "payload": {"account_id": "123458", "role_id": "999", "user_name": "Bradley Fine", "user_description": "face", "update_date": "1629115994000"}}

```

위 처럼 MySQL에서 변화가 생기는 것을 Kafka가 topic의 형태로 잘 감지하고 저장하고 있음을 확인 할 수 있다.

2. Kafka → JDBC connector → SinkDB

A. SinkDB 컨테이너 생성 (docker-compose.yml 작성)

```

mysql-sink:
  image: mysql:8.0
  container_name: mysql-sink
  ports:
    - 3307:3306
  environment:
    MYSQL_ROOT_PASSWORD: sasd
    MYSQL_USER: mysqluser
    MYSQL_PASSWORD: mysqlpw
  command:
    - --character-set-server=utf8mb4
    - --collation-server=utf8mb4_unicode_ci
  volumes:
    - /Users/jiminlee/TempCodes/CDC/mysql-sink/data:/var/lib/mysql

```

docker-compose파일을 수정하여 쉽게 mysql sink용 컨테이너를 생성해 주었다.

B. DB 설정

데이터베이스 및 테스트용 테이블 생성

```

mysql -u root -p
create database sinkdb;
use sinkdb;

CREATE TABLE accounts (
  account_id VARCHAR(255),
  role_id VARCHAR(255),
  user_name VARCHAR(255),

```

```

    user_description VARCHAR(255),
    update_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (account_id)
);

```

mysql 사용자 추가 및 권한 확인

```

use mysql;

// mysqluser 가 추가 되어 있는지 확인
select host, user from user;

// mysqluser 없으면 생성
CREATE USER 'mysqluser'@'%' IDENTIFIED BY 'mysqlpw';
// mysqluser 에게 권한 부여
GRANT ALL PRIVILEGES ON *.* TO 'mysqluser'@'%';

FLUSH PRIVILEGES;

```

C. 4. Kafka JDBC Connector (Source and Sink) 설치

JDBC Connector 설치

<https://www.confluent.io/hub/confluentinc/kafka-connect-jdbc>

위 링크에서 다운로드후, Kafka 컨테이너로 업로드

```
#파일 업로드
docker cp confluentinc-kafka-connect-jdbc-10.7.0.zip kafka:/opt/kafka_2.13-2.8.1/connectors/
```

```
cd /opt/kafka_2.13-2.8.1/connectors
unzip confluentinc-kafka-connect-jdbc-10.7.0.zip
```

plugin 경로 확인

source connector를 설치할 때 이미 `/opt/kafka/config/connect-distributed.properties` 파일의 plugin 경로를 수정해두었다.

D. Kafka connect 실행

```
connect-distributed.sh /opt/kafka/config/connect-distributed.properties
```

E. Sink Connector 생성하기

worker, version, commit 및 Kafka 클러스터 ID에 대한 kafka Connect 클러스터 정보를 확인해보자.

```
curl http://localhost:8083/
```

```
curl http://localhost:8083/
root@4059b947b0fa:~# curl http://localhost:8083/
{"version":"2.8.1","commit":"839b886f9b73b15","kafka_cluster_id":"Y46X_iycSE-5lSb5ZC_u9A"}root@4059b947b0fa:~#
```

MySQL 커넥터 플러그인 확인

```
curl --location --request GET 'localhost:8083/connector-plugins'
```

```
root@4059b947b0fa:~# curl http://localhost:8083/
{"version":"2.8.1","curl --location --request GET 'localhost:8083/connector-plugins'": "curl --location --request GET 'localhost:8083/connector-plugins'"}root@4059b947b0fa:~# curl --location --request GET 'localhost:8083/connector-plugins'
[{"class":"io.confluent.connect.jdbc.JdbcSinkConnector","type":"sink","version":"10.7.0"}, {"class":"io.confluent.connect.jdbc.JdbcSourceConnector","type":"source","version":"10.7.0"}, {"class":"io.debezium.connector.mysql.MySqlConnector","type":"source","version":"1.5.4.Final"}, {"class":"org.apache.kafka.connect.file.FileStreamSinkConnector","type":"sink","version":"2.8.1"}, {"class":"org.apache.kafka.connect.file.FileStreamSourceConnector","type":"source","version":"2.8.1"}, {"class":"org.apache.kafka.connect.mirror.MirrorCheckpointConnector","type":"source","version":"1"}, {"class":"org.apache.kafka.connect.mirror.MirrorHeartbeatConnector","type":"source","version":"1"}, {"class":"org.apache.kafka.connect.mirror.MirrorSourceConnector","type":"source","version":"1"}]root@4059b947b0fa:~#
```

`io.confluent.connect.jdbc.JdbcSinkConnector`, `io.confluent.connect.jdbc.JdbcSourceConnector` 가 있어야 한다.

Rest API 로 connector 생성

```
curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "sink-test-connector",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://mysql-sink:3306/sinkdb?user=mysqluser&password=mysqlpw",
    "auto.create": "false",
    "auto.evolve": "false",
    "delete.enabled": "true",
    "insert.mode": "upsert",
    "pk.mode": "record_key",
    "table.name.format": "${topic}",
    "tombstones.on.delete": "true",
    "connection.user": "mysqluser",
    "connection.password": "mysqlpw",
    "topics.regex": "dbserver1.testdb.(.*)",
    // "topics": "dbserver1.testdb.accounts", <- 하나만 명시
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap, route, TimestampConverter",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.unwrap.drop.tombstones": "true",
    // "transforms.unwrap.drop.tombstones": "false",
    // "transforms.unwrap.delete.handling.mode": "rewrite",
    "transforms.route.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.route.regex": "[^.]+\\.(^.+)\\.(^.+)",
    "transforms.route.replacement": "$3",
    "transforms.TimestampConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.TimestampConverter.format": "yyyy-MM-dd HH:mm:ss",
    "transforms.TimestampConverter.target.type": "Timestamp",
    "transforms.TimestampConverter.field": "update_date"
  }
}'
```

상세 Configuration 확인

https://docs.confluent.io/kafka-connectors/jdbc/current/sink-connector/sink_config_options.html#connection

Kafka Connect REST API

```
# 목록
curl --location --request GET 'http://localhost:8083/connectors'

# 상세정보
curl --location --request GET 'http://localhost:8083/connectors/sink-test-connector/config' \
--header 'Content-Type: application/json'

# 삭제
curl --location --request DELETE 'http://localhost:8083/connectors/sink-test-connector'
```

F. MySQL Sink 확인

```
mysql> use sinkdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from accounts
-> ;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

이미 카프카 상에 생성되어있던 토픽으로 sinkDB가 업데이트 되어있는 것을 확인할 수 있다.

ADD

```
INSERT INTO accounts VALUES ("111111", "111", "Jimin", "ADD", "2021-08-16 10:11:12");
INSERT INTO accounts VALUES ("222222", "222", "Lee", "FROM", "2021-08-16 11:12:13");
INSERT INTO accounts VALUES ("333333", "333", "Test", "SOURCE", "2021-08-16 12:13:14");
```

```
mysql> INSERT INTO accounts VALUES ("111111", "111", "Jimin", "ADD", "2021-08-16 10:11:12");
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO accounts VALUES ("222222", "222", "Lee", "FROM", "2021-08-16 11:12:13");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO accounts VALUES ("333333", "333", "Test", "SOURCE", "2021-08-16 12:13:14");
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM accounts;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 111111 | 111 | Jimin | ADD | 2021-08-16 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
5 rows in set (0.03 sec)
```

Source DB

```
mysql> select * from accounts
-> ;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 111111 | 111 | Jimin | ADD | 2021-08-16 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

SinkDB

UPDATE

```
UPDATE accounts SET user_name = 'UPDATE!!!!' WHERE account_id = 111111
```

```
mysql> UPDATE accounts SET user_name = 'UPDATE!!!!' WHERE account_id = 111111;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM accounts;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 111111 | 111 | UPDATE!!!! | ADD | 2021-08-16 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Source DB

```
mysql> select * from accounts
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 111111 | 111 | UPDATE!!!! | ADD | 2021-08-16 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from accounts;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 111111 | 111 | UPDATE!!!! | ADD | 2021-08-16 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Sink DB

DELETE ← Trouble Shooting #5

```
delete from accounts where account_id = 0;
```

```
mysql> select * from accounts;
+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date |
+-----+-----+-----+-----+-----+
| 0 | Insert | From | React | 1111-11-11 10:11:12 |
| 123 | Insert | From | React | 1111-11-11 10:11:12 |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 |
| 222222 | 222 | Insert | From | React |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 |
| 345 | afff | From | React | 1111-11-11 10:11:12 |
| 44 | 44 | JI | MIN | 2021-08-16 12:13:14 |
| 444 | 444 | 444 | 444 | 2021-08-16 12:13:14 |
| 4444 | 444 | 444 | 444 | 2021-08-16 12:13:14 |
| 5555 | 555 | 555 | 555 | 2021-08-16 12:13:14 |
| 5678 | Insert | From | React | 1111-11-11 10:11:12 |
| 666 | 66 | 665 | 65 | 2021-08-16 12:13:14 |
| 8880 | 8880 | From Source | Do you see? | 2021-08-16 10:11:12 |
| 9999 | 9999 | Spring Post | first post | 2021-08-16 10:11:12 |
+-----+-----+-----+-----+-----+
17 rows in set (0.00 sec)

mysql> delete from accounts where account_id = 0;
Query OK, 1 row affected (0.01 sec)
```

SOURCE DB

```
mysql> select * from accounts;
+-----+-----+-----+-----+-----+-----+
| account_id | role_id | user_name | user_description | update_date | _deleted |
+-----+-----+-----+-----+-----+-----+
| 111111 | 111 | UPDATE!!! | ADD | 2021-08-16 10:11:12 | NULL |
| 123 | Insert | From | React | 1111-11-04 10:11:12 | NULL |
| 123456 | 111 | Susan Cooper | God | 2021-08-16 10:11:12 | NULL |
| 123457 | 111 | Rick Ford | mistakes | 2021-08-16 11:12:13 | NULL |
| 123458 | 999 | Bradley Fine | face | 2021-08-16 12:13:14 | NULL |
| 222222 | 222 | Lee | FROM | 2021-08-16 11:12:13 | NULL |
| 333333 | 333 | Test | SOURCE | 2021-08-16 12:13:14 | NULL |
| 345 | afff | From | React | 1111-11-04 10:11:12 | NULL |
| 44 | 44 | JI | MIN | 2021-08-16 12:13:14 | NULL |
| 444 | 444 | 444 | 444 | 2021-08-16 12:13:14 | NULL |
| 5555 | 555 | 555 | 555 | 2021-08-16 12:13:14 | NULL |
| 5678 | Insert | From | React | 1111-11-04 10:11:12 | NULL |
| 666 | 66 | 665 | 65 | 2021-08-16 12:13:14 | NULL |
| 8880 | 8880 | From Source | Do you see? | 2021-08-16 10:11:12 | NULL |
| 9999 | 9999 | Spring Post | first post | 2021-08-16 10:11:12 | NULL |
+-----+-----+-----+-----+-----+-----+
17 rows in set (0.00 sec)
```

SINK DB

2.1 TargetDB2 : SinkDB2 및 custom table 생성

A. MYSQL 테이블 생성 및 권한설정

```
mysql -u root -p
create database testdb;
use testdb;
CREATE TABLE CUSTOM_TABLE (
```

```
mysql -u root -p
// 권한설정
use mysql;

// mysqluser 가 추가 되어 있는지 확인
select host, user from user;
```

```

title VARCHAR(255),
name VARCHAR(255),
id VARCHAR(255),
PRIMARY KEY (id)
};

// mysqluser 없으면 생성
CREATE USER 'mysqluser'@'%' IDENTIFIED BY 'mysqlpw';
// mysqluser 에게 권한 부여
GRANT ALL PRIVILEGES ON *.* TO 'mysqluser'@'%';

FLUSH PRIVILEGES;

//데이터 베이스 테이블 생성
create database sinkdb2;

use sinkdb2;

CREATE TABLE CUSTOM_TABLE (
    title VARCHAR(255),
    name VARCHAR(255),
    id VARCHAR(255),
    PRIMARY KEY (id)
);

```

```

mysql> desc custom_table;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| title | varchar(255) | YES  |     | NULL    |       |
| name  | varchar(255) | YES  |     | NULL    |       |
| id    | varchar(255) | NO   | PRI | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

B. SinkDB2 Connector 생성

중요한 것은 topics에서 CUSTOM_TABLE만 받아오도록 설정하는 것.

```

curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "sink-test2-connector",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://mysql-sink2:3306/sinkdb2?user=mysqluser&password=mysqlpw",
    "auto.create": "false",
    "auto.evolve": "true",
    "delete.enabled": "true",
    "insert.mode": "upsert",
    "pk.mode": "record_key",
    "table.name.format": "${topic}",
    "tombstones.on.delete": "true",
    "connection.user": "mysqluser",
    "connection.password": "mysqlpw",
    "topics": "dbserver1.testdb.CUSTOM_TABLE",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap, route, TimestampConverter",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.unwrap.drop.tombstones": "true",
    "transforms.unwrap.drop.tombstones": "false",
    "transforms.unwrap.delete.handling.mode": "rewrite",
    "transforms.route.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.route.regex": "(^.+)\.\.([^.+])\.\.([^.+])",
    "transforms.route.replacement": "$3",
    "transforms.TimestampConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.TimestampConverter.format": "yyyy-MM-dd HH:mm:ss",
    "transforms.TimestampConverter.target.type": "Timestamp",
    "transforms.TimestampConverter.field": "update_date"
  }
}'

```

C. Insert 수행, Kafka Consumer로 Topic 저장 확인

```

mysql> select * from CUSTOM_TABLE;
+-----+-----+-----+
| title | name   | id   |
+-----+-----+-----+
| test  | jimin  | 1   |
| test  | adfssfn | 2   |
| test  | adfssfn | 6   |
+-----+-----+-----+
3 rows in set (0.00 sec)

kafka-console-consumer.sh --topic dbserver1.testdb.CUSTOM_TABLE --bootstrap-server localhost:9092
--from-beginning
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": true, "field": "title"}, {"type": "string", "optional": true, "field": "name"}, {"type": "string", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "__deleted"}]}, "payload": {"name": "dbserver1.testdb.CUSTOM_TABLE.Value"}, "payload": {"title": "test", "name": "jimin", "id": "1", "__deleted": "false"}}
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": true, "field": "title"}, {"type": "string", "optional": true, "field": "name"}, {"type": "string", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "__deleted"}]}, "payload": {"name": "dbserver1.testdb.CUSTOM_TABLE.Value"}, "payload": {"title": "test", "name": "adfssfn", "id": "2", "__deleted": "false"}}
{"schema": {"type": "struct", "fields": [{"type": "string", "optional": true, "field": "title"}, {"type": "string", "optional": true, "field": "name"}, {"type": "string", "optional": false, "field": "id"}, {"type": "string", "optional": true, "field": "__deleted"}]}, "payload": {"name": "dbserver1.testdb.CUSTOM_TABLE.Value"}, "payload": {"title": "test", "name": "adfssfn", "id": "6", "__deleted": "false"}}

kafka-console-consumer.sh --topic dbserver1.testdb.CUSTOM_TABLE --bootstrap-server localhost:9092 --from-beginning

```

성공!!

```

mysql> select * from CUSTOM_TABLE;
+-----+-----+-----+-----+
| title | name   | id   | __deleted |
+-----+-----+-----+-----+
| test  | jimin  | 1   | false    |
| test  | adfssfn | 2   | false    |
| test  | adfssfn | 6   | false    |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)

```

SINKDB2.CUSTOM_TABLE

3. Spring → MySQL 연동

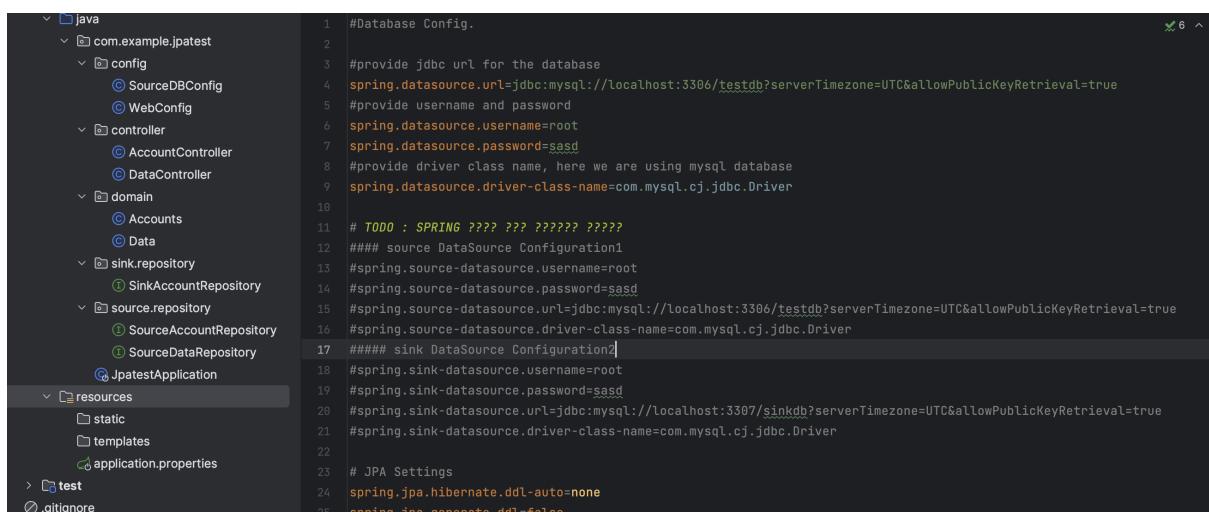
<https://doozi0316.tistory.com/entry/Spring-Boot-MyBatis-MySQL-연동-방법>

mybatis, JDBC, JPA 세 가지 방법이 있다.

두가지에 실패해서 JPA에 도전

총 3개의 Spring Server를 돌린다

Spring Server 1 → SourceDB



```

1 #Database Config.
2
3 #provide jdbc url for the database
4 spring.datasource.url=jdbc:mysql://localhost:3306/testdb?serverTimezone=UTC&allowPublicKeyRetrieval=true
5 #provide username and password
6 spring.datasource.username=root
7 spring.datasource.password=sasd
8 #provide driver class name, here we are using mysql database
9 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
10
11 # TODO : SPRING ??????????????
12 ##### source DataSource Configuration1
13 #spring.source-datasource.username=root
14 #spring.source-datasource.password=sasd
15 #spring.source-datasource.url=jdbc:mysql://localhost:3306/testdb?serverTimezone=UTC&allowPublicKeyRetrieval=true
16 #spring.source-datasource.driver-class-name=com.mysql.cj.jdbc.Driver
17 ##### sink DataSource Configuration2
18 #spring.sink-datasource.username=root
19 #spring.sink-datasource.password=sasd
20 #spring.sink-datasource.url=jdbc:mysql://localhost:3307/sinkdb?serverTimezone=UTC&allowPublicKeyRetrieval=true
21 #spring.sink-datasource.driver-class-name=com.mysql.cj.jdbc.Driver
22
23 # JPA Settings
24 spring.jpa.hibernate.ddl-auto=None
25 spring.jpa.generate-ddl=false

```

- Accounts, Data 두개의 도메인을 가지고 있다. → SoureDB에 두개의 TABLE이 있기 때문
- 그에 따른 Controller, datarepository도 각각 있음

- Port : **8080**
- Controller에서는 데이터 **INSERT, DELETE, SELECT**에 관련된 API를 제공한다.

```

AccountController.java

```

@RestController
@RequestMapping(path = "/api")
public class AccountController {
 @Autowired
 SourceAccountRepository sourceAccountRepository;

 @PostMapping("/new-account")
 public String add(@RequestBody Accounts account) {
 sourceAccountRepository.save(account);
 return "update OK";
 }

 @DeleteMapping("/delete-account")
 public String deleteById(@RequestBody Accounts account) {
 sourceAccountRepository.deleteById(account.getAccount_id());
 return "delete OK";
 }

 @PostMapping("/test")
 public String test() {
 return "test OK";
 }

 @GetMapping("/accounts")
 public List<Accounts> getAccounts() {
 return sourceAccountRepository.findAll();
 }
}

```

DataController.java

```

no usages
@RestController
@RequestMapping(path = "/api")
public class DataController {
 @Autowired
 SourceDataRepository sourceDataRepository;

 @PostMapping("/new-data")
 public String add(@RequestBody Data data) {
 sourceDataRepository.save(data);
 return "update OK";
 }

 @DeleteMapping("/delete-data")
 public String deleteById(@RequestBody Data data) {
 sourceDataRepository.deleteById(data.getId());
 return "delete OK";
 }

 @GetMapping("/getdata")
 public List<Data> getData() {
 return sourceDataRepository.findAll();
 }
}

```


```

Spring Server 2 → SinkDB1

```

application.properties

```

#Database Config.
spring.datasource.url=jdbc:mysql://localhost:3307/sinkDB?serverTimezone=UTC&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=sasd
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

JPA Settings
spring.jpa.hibernate.ddl-auto=None
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database=mysql
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.properties.hibernate.format_sql=true

Hibernate Logging
logging.level.org.hibernate=info
server.port=8081

```


```

- Accounts topic만 저장하는 SinkDB1이다.
- Accounts에 관련된 java파일들이 있다.
- Port : **8081**
- Controller에서는 데이터 **SELECT**에 관련된 API만 제공한다. → 데이터 조작은 SourceDB에서만 할 것이다.

```

no usages
@RestController
@RequestMapping(path=@"/api")
public class AccountController {
    1 usage
    @Autowired
    SinkAccountRepository sinkAccountRepository;

    /**
     * @PostMapping("/new-account")
     */
    public String add(@RequestBody Accounts account) {
        // sinkAccountRepository.save(account);
        // return "update OK";
    }

    no usages
    @GetMapping(@"/accounts")
    public List<Accounts> getAccounts() { return sinkAccountRepository.findAll(); }
}

```

Spring Server 3 → SinkDB2

The left pane shows the project structure:

- gradle
- src
 - main
 - com.example.jpasink
 - config
 - SecurityConfig
 - WebConfig
 - controller
 - AccountController
 - domain
 - Accounts
 - repository
 - SinkAccountRepository
 - JpasinkApplication
 - resources
 - static
 - templates
 - application.properties
- test
- .gitignore
- build.gradle
- gradlew
- gradlew.bat

The right pane shows the content of application.properties:

```

#Database Config.
1 #provide jdbc url for the database
2
3 spring.datasource.url=jdbc:mysql://localhost:3307/sinkdb?serverTimezone=UTC&allowPublicKeyRetrieval=true
4
5 #provide username and password
6 spring.datasource.username=root
7 spring.datasource.password=sasd
8 #provide driver class name, here we are using mysql database
9 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
10
11
12 # JPA Settings
13 spring.jpa.hibernate.ddl-auto=none
14 spring.jpa.generate-ddl=false
15 spring.jpa.show-sql=true
16 spring.jpa.database=mysql
17 #spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
18 spring.jpa.properties.hibernate.format_sql=true
19
20
21 # Hibernate Logging
22 logging.level.org.hibernate=info
23
24 server.port=8081

```

- Data(CUSTOM_TABLE) topic만 저장하는 SinkDB2이다.
- Data에 관련된 java 파일들이 있다.
- Port : **8082**
- Controller에서는 데이터 **SELECT**에 관련된 API만 제공한다. → 데이터 조작은 SourceDB에서만 할 것이다.

```

6 import java.util.List;
7
8 no usages
9
10 @RestController
11 @RequestMapping(path=@"/api")
12 public class DataController {
13     1 usage
14     @Autowired
15     SinkDataRepository sinkDataRepository;
16
17     // @PostMapping("/new-data")
18     // public String add(@RequestBody Data data){
19     //     sinkDataRepository.save(data);
20     //     return "update OK";
21     // }
22
23     no usages
24     @GetMapping(@"/getdata")
25     public List<Data> getData() { return sinkDataRepository.findAll(); }
26 }

```

4. React → Spring 연동

React 서버에서 Spring 서버 1, 2, 3를 각각 연동하였다.

- Spring Server1 (SourceDB)로는 Post Api를 통해 새로운 데이터를 생성, 삭제가 가능하도록 하였다
- Spring Server2, 3 (SinkDB)로는 Get Api를 통해 sourceDB → Kafka → SinkDB로 CDC되어 업데이트된 SinkDB의 데이터를 확인하도록 하였다.
- Sink1, Sink2에 접근하는 페이지를 각각 나눠 버튼을 눌러 확인이 가능하도록 하였다.

CDC Demo

Show Sink 1
Show Sink 2

Create New Data In
SourceDB.CUSTOM_TABLE

Create Data

Delete Data In
SourceDB.CUSTOM_TABLE

Delete Row

SinkDB2 CUSTOM_TABLE :

```

{"id":2,"title":"test","name":"adfssfn"}
{"id":4,"title":"del","name":"asfffn"}
{"id":6,"title":"test","name":"adfssfn"}

```

load sink data

```

import React, { useState, useEffect } from "react";
import Axios from "axios";
import { Button } from "react-bootstrap";

export const DelSourceDB2 = () => {
  const [id, setId] = useState("0");
  const deletePostHandler = () => {
    Axios.post("http://localhost:8080/api/delete-data", {
      id: parseInt(id),
    })
      .then(function (response) {
        alert(response.data);
        console.log(response);
      })
      .catch(function (error) {
        console.log(error);
      });
  };
  return (
    <div className="SourceDB">
      <h3>Delete Data In</h3>
      <h3>SourceDB.CUSTOM_TABLE</h3>
      <div className="inputs">
        <div className="inputWrapper">
          id :
          <input value={id} onChange={(e) => setId(e.target.value)} />
        </div>
        <Button
          onClick={deletePostHandler}
          className="deleteBtn"
        >Delete</Button>
      </div>
    </div>
  );
}

```

TroubleShooting

#1 Docker-compose m1칩 에러

`WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8).`

Image가 맥에 호환되지 않는 문제. Docker-Compose 파일에서 `platform: linux/amd64/v8`로 따로 명시 해주어 해결

<https://collabnix.com/warning-the-requested-images-platform-linux-amd64-does-not-match-the-detected-host-platform-linux-arm64-v8/>

#2 Docker-compose Volume error

`Error response from daemon: invalid volume specification: 'D:/mysql-sink/data:/var/lib/mysql:rw'`

Volume에 없는 경로를 지정해서 발생한 에러. Docker-Compose 파일을 전혀 이해하지 않고 복사 붙여넣기 하다가 발생한 에러이다.

: 전 경로가 실제 내 디렉토리에 위치하도록 다시 확인해주어야 한다.

#3 Linux vim

`bash: vim: command not found`

```
apt update
apt install vim
```

#4 JDBC connector 설치시 No suitable driver found

```
Caused by: java.sql.SQLException: No suitable driver found for jdbc:mysql://mysql-sink:3306/sinkdb?user=mysqluser&password=mysqlpw
at java.sql/java.sql.DriverManager.getConnection(Unknown Source)
at java.sql/java.sql.DriverManager.getConnection(Unknown Source)
at io.confluent.connect.jdbc.dialect.GenericDatabaseDialect.getConnection(GenericDatabaseDialect.java:250)
at io.confluent.connect.jdbc.util.CachedConnectionProvider.newConnection(CachedConnectionProvider.java:81)
at io.confluent.connect.jdbc.util.CachedConnectionProvider.getConnection(CachedConnectionProvider.java:53)
... 13 more
```

kafka connect

<https://wecandev.tistory.com/111>

Connect/J JDBC driver for MySQL를 다운받는다.

해당 jar 파일을 Confluent의 connect 플러그인이 설치된 디렉토리에 넣는다.

파일명 : mysql-connector-java-8.0.27.jar

경로 : {KAFKA_HOME}/connectors/confluentinc-kafka-connect-jdbc-10.2.5/lib/

```
docker cp mysql-connector-java-8.0.27.jar kafka:/opt/kafka_2.13-2.8.1/connectors/confluentinc-kafka-connect-jdbc-10.7.0/lib/
```

#5 DELETE의 경우 CDC 반영이 안된다

Sink Connector, Source Connector에 각각 config를 새로 추가해준다.

```
curl --location --request DELETE 'http://localhost:8083/connectors/source-test-connector'  
curl --location --request DELETE 'http://localhost:8083/connectors/sink-test-connector'
```

```
curl --location --request POST 'http://localhost:8083/connectors' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
  "name": "source-test-connector",  
  "config": {  
    "connector.class": "io.debezium.connector.mysql.MySQLConnector",  
    "tasks.max": "1",  
    "database.hostname": "mysql",  
    "database.port": "3306",  
    "database.user": "mysqluser",  
    "database.password": "mysqlpw",  
    "database.server.id": "184054",  
    "database.server.name": "dbserver1",  
    "database.allowPublicKeyRetrieval": "true",  
    "database.include.list": "testdb",  
    "database.history.kafka.bootstrap.servers": "kafka:9092",  
    "database.history.kafka.topic": "dbhistory.testdb",  
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",  
    "key.converter.schemas.enable": "true",  
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",  
    "value.converter.schemas.enable": "true",  
    "transforms": "unwrap,addTopicPrefix",  
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",  
    "transforms.addTopicPrefix.type": "org.apache.kafka.connect.transforms.RewriteTopic",  
    "transforms.addTopicPrefix.regex": "(.*)",  
    "transforms.unwrap.drop.tombstones": "false",  
    "transforms.unwrap.delete.handling.mode": "rewrite",  
    "transforms.addTopicPrefix.replacement": "$1"  
  }  
}'
```

```
# Sink  
curl --location --request POST 'http://localhost:8083/connectors' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
  "name": "sink-test-connector",  
  "config": {  
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",  
    "tasks.max": "1",  
    "connection.url": "jdbc:mysql://mysql-sink:3306/sinkdb?user=mysqluser&password=mysqlpw",  
    "auto.create": "false",  
    "auto.evolve": "true",  
    "delete.enabled": "true",  
    "insert.mode": "upsert",  
    "pk.mode": "record_key",  
    "table.name.format": "${topic}",  
    "tombstones.on.delete": "true",  
    "connection.user": "mysqluser",  
    "connection.password": "mysqlpw",  
    "topics": "dbserver1.testdb.accounts",  
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",  
    "key.converter.schemas.enable": "true",  
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",  
    "value.converter.schemas.enable": "true",  
    "transforms": "unwrap,route,TimestampConverter",  
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",  
    "transforms.unwrap.drop.tombstones": "false",  
    "transforms.unwrap.delete.handling.mode": "rewrite",  
    "transforms.route.type": "org.apache.kafka.connect.transforms.RewriteTopic",  
    "transforms.route.regex": "[^.]+\\.(^.+)\\.(^.+)",  
    "transforms.route.replacement": "$3",  
    "transforms.TimestampConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter",  
    "transforms.TimestampConverter.format": "yyyy-MM-dd HH:mm:ss",  
    "transforms.TimestampConverter.target.type": "Timestamp",  
    "transforms.TimestampConverter.field": "update_date"  
  }  
}'
```

#5-1 deleted column 생성으로 인한 TOPIC 오류

위 #5의 솔루션대로 따라가면 sinkDB에는 새로운 __deleted 컬럼이 생긴다.

```
[2023-04-25 12:47:04,881] [ERROR] WorkerSinkTask{id=sink-test2-connector-0} Task threw an uncaught and unrecoverable exception. Task is being killed and will not recover until manually restarted. Error: Table "CUSTOM_TABLE" is missing fields ([SinkRecordField{schema=Schema<STRING>, name='__deleted', isPrimaryKey=false}]) and auto-evolution is disabled (org.apache.kafka.connect.runtime.WorkerSinkTask:608)  
io.confluent.connect.jdbc.sink.TableAlterOrCreateException: Table "CUSTOM_TABLE" is missing fields ([SinkRecordField{schema=Schema<STRING>, name='__deleted', isPrimaryKey=false}]) and auto-evolution is disabled  
at io.confluent.connect.jdbc.sink.DbStructure.amendIfNecessary(DbStructure.java:193)  
at io.confluent.connect.jdbc.sink.DbStructure.createOrAmendIfNecessary(DbStructure.java:83)  
at io.confluent.connect.jdbc.sink.BufferedRecords.add(BufferedRecords.java:122)  
at io.confluent.connect.jdbc.sink.JdbcDbWriter.write(JdbcDbWriter.java:74)  
at io.confluent.connect.jdbc.sink.JdbcSinkTask.put(JdbcSinkTask.java:88)  
at org.apache.kafka.connect.runtime.WorkerSinkTask.deliverMessages(WorkerSinkTask.java:582)  
at org.apache.kafka.connect.runtime.WorkerSinkTask.poll(WorkerSinkTask.java:330)  
at org.apache.kafka.connect.runtime.WorkerSinkTask.iteration(WorkerSinkTask.java:232)  
at org.apache.kafka.connect.runtime.WorkerSinkTask.execute(WorkerSinkTask.java:201)  
at org.apache.kafka.connect.runtime.WorkerTask.doRun(WorkerTask.java:188)  
at org.apache.kafka.connect.runtime.WorkerTask.run(WorkerTask.java:237)  
at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Unknown Source)  
at java.base/java.util.concurrent.FutureTask.run(Unknown Source)
```

TOPIC에는 정상적으로 저장이 되나, SINKDB에서 이를 받아오지 못하는 오류

새로 생긴 컬럼을 생성하지 못하게 막아놓았기 때문.

SINK CONNECTOR config를 수정해서 다시 생성해주면 정상적으로 업데이트가 된다.

```

curl --location --request POST 'http://localhost:8083/connectors' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "sink-test2-connector",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://mysql-sink2:3306/sinkdb?user=mysqluser&password=mysqlpw",
    "auto.create": "false",
    "auto.evolve": "true", // 새로운 column이 생기면 이를 sinkdb에도 동일하게 생성해준다.
    "delete.enabled": "true",
    "insert.mode": "upsert",
    "pk.mode": "record_key",
    "table.name.format": "${topic}",
    "tombstones.on.delete": "true",
    "connection.user": "mysqluser",
    "connection.password": "mysqlpw",
    "topics": "dbserver1.testdb.CUSTOM_TABLE",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable": "true",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "true",
    "transforms": "unwrap, route, TimestampConverter",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
    "transforms.unwrap.drop.tombstones": "true",
    "transforms.unwrap.drop.tombstones": "false",
    "transforms.unwrap.delete.handling.mode": "rewrite",
    "transforms.route.type": "org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.route.regex": "([^.]+)\\.([^.]+)\\.([^.]+)",
    "transforms.route.replacement": "$3",
    "transforms.TimestampConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.TimestampConverter.format": "yyyy-MM-dd HH:mm:ss",
    "transforms.TimestampConverter.target.type": "Timestamp",
    "transforms.TimestampConverter.field": "update_date"
  }
}'

```

#6 (SINK) 새로운 TABLE 생성시 전체TABLE 업데이트 안됨

```

Caused by: org.apache.kafka.connect.errors.ConnectException: Sink connector 'sink-test-connector' is configured with 'delete.enabled=true' and 'pk.mode=record key' and therefore requires records with a non-null key and non-null Struct or primitive key schema, but found record at (topic='DB_TEST',partition=0,offset=0,timestamp=1601953279066) with a null key and null key schema.
        at io.confluent.connect.jdbc.sink.RecordValidator.lambda$requireKey$3(RecordValidator.java:116)
        at io.confluent.connect.jdbc.sink.BufferedRecords.add(BufferedRecords.java:81)
        at io.confluent.connect.jdbc.sink.JdbcDbWriter.write(JdbcDbWriter.java:74)
        at io.confluent.connect.jdbc.sink.JdbcSinkTask.put(JdbcSinkTask.java:88)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.deliverMessages(WorkerSinkTask.java:582)
        ... 10 more
[2023-04-28 02:59:15,025] INFO Stopping task (io.confluent.connect.jdbc.sink.JdbcSinkTask:170)
[2023-04-28 02:59:15,025] INFO Stopped task (io.confluent.connect.jdbc.sink.JdbcSinkTask:170)

```

kafka connect 에러메세지

Source db에서 새로운 테이블 DB_TEST를 생성했다. 이를 까먹은채로 accounts 테이블에 insert했을때 카프카 topic으로 생성은 되지만 sink DB에는 연동이 안되던 문제

DB_TEST는 무언가 만족하지 않는 조건이 있는 것 같았다.

Debezium Connector Config를 보면

```

"database.include.list": "testdb",
"database.history.kafka.bootstrap.servers": "kafka:9092",
"database.history.kafka.topic": "dbhistory.testdb",

```

데이터베이스 전체를 보내게 되어있다.

Sink Connector Config를 보면

```

"topics.regex": "dbserver1.testdb.(.*)",
// "topics": "dbserver1.testdb.accounts", <- 하나만 명시
"key.converter": "org.apache.kafka.connect.json.JsonConverter",

```

위처럼 전부 받게 되어있다. 일단 Sink DB에는 accounts table만 받으면 되어서 config에 하나만 받도록 수정했더니 정상적으로 accounts table을 받게 되었다.

8. (REACT, SPRING) Axios 사용시 CORS 에러

```

const getSinkAccounts = () => {
  Axios.get("http://localhost:8081/api/accounts").then((response) => {
    if (response.data) {
      console.log(response.data);
      setAccounts(response.data);
    } else {
      alert("failed to");
    }
  });
};

```

- ✖ Access to XMLHttpRequest at '<http://localhost:8081/api/accounts>' from origin '<http://localhost:3001>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
- ✖ ▶ GET <http://localhost:8081/api/accounts> xhr.js:233 ↗
net::ERR_FAILED 200
- ✖ ▶ Uncaught (in promise) xhr.js:142
AxiosError {message: 'Network Error', name: 'AxiosError', code: 'ERR_NETWORK', config: {...}, request: XMLHttpRequest, ...}

Cross Origin Resource Sharing의 약자로,

현재 웹페이지 도메인에서 다른 웹페이지 도메인으로 리소스가 요청되는 경우를 말한다.

예를 들면, 웹페이지인 <http://web.com>에서 API 서버 URL인 <http://api.com> 도메인으로 API를 요청하면

http 형태로 요청이 되므로 브라우저 자체에서 보안 상 이유로 CORS를 제한하게 되는 현상을 말합니다.

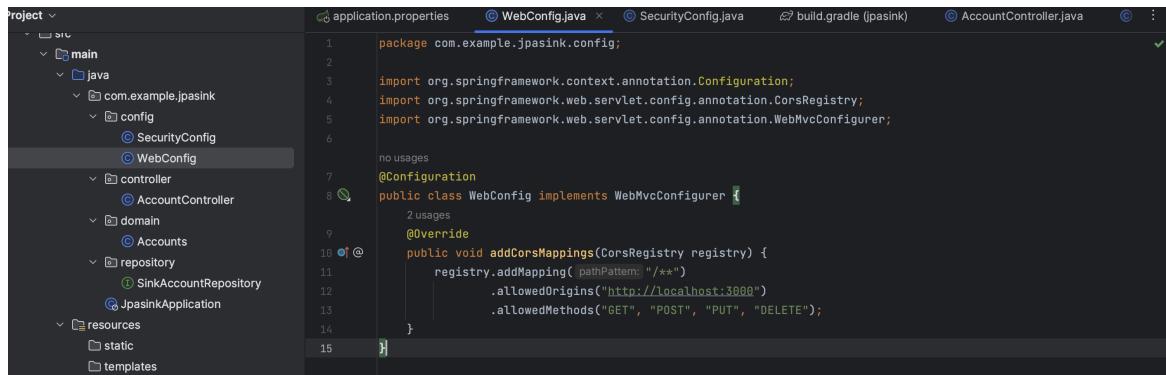
▼ Client측에서 해결해보려한 시도 (실패)

<https://woochan-dev.tistory.com/94>

<https://www.datoybi.com/http-proxy-middleware/>

▼ Server측에서 해결해보려한 시도 (성공)

WebConfig 파일을 만들어줘서 로컬 주소로부터 오는 API호출을 허용해줬다.



```

Project : 
  - src
    - main
      - java
        - com.example.jpasink
          - config
            - SecurityConfig
            - WebConfig
          - controller
            - AccountController
          - domain
            - Accounts
          - repository
            - SinkAccountRepository
            - JpasinkApplication
          - resources
            - static
            - templates
  - application.properties
  - WebConfig.java
  - SecurityConfig.java
  - build.gradle (jpasink)
  - AccountController.java

WebConfig.java content:
package com.example.jpasink.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
no usages
@Configuration
public class WebConfig implements WebMvcConfigurer {
  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping(pathPattern: "/*")
      .allowedOrigins("http://localhost:3000")
      .allowedMethods("GET", "POST", "PUT", "DELETE");
  }
}

```

9. (SPRING) DB TABLE명 지정 예러

React에서 AXIOS로 호출 시, SPRING에서 데이터베이스 테이블 명 이름을 찾지 못한다고 나올 때이다.

Domain의 Class이름이 default로 테이블명을 지정해주는 것 같았다.

@Entity 아래에 @Table(name="테이블") 어노테이션을 달아 테이블 이름을 제대로 명시해주었다.

```

9 import java.io.Serializable;
10
11     4 usages
12     @Getter
13     @NoArgsConstructor(access = AccessLevel.PUBLIC)
14     @Entity
15     @Table(name="CUSTOM_TABLE")
16     public class Data implements Serializable {
17
18         1 usage
19         @Id
20         @Column
21         // @GeneratedValue(strategy = GenerationType.IDENTITY)
22         private Long id;
23         1 usage
24         @Column
25         private String title;
26         1 usage
27         @Column

```

후기

CDC, 카프카, 데베이스, JDBC등 개념이 확실치 않은 상태에서 실습을 진행해보니 한줄 짊어넣을 때 마다 에러가 생겼다

로컬(mac m1), VM(Ubuntu 20.04)등 환경을 옮겨가며 진행해봤는데 계속 에러가 나오고 고쳐지지 않아 4번정도 엎은 것 같다.

무지성으로 따라가기보다 천천히 해보자는 생각으로 공식문서를 읽으며 공식문서에서 제공하는 실습을 해본 뒤, OCI 환경에서 성공을 해냈다.

문서화를 위해 마지막으로 Local에서 천천히 다시 시도하는데 귀신같이 에러가 나지 않았고 CDC 구축을 성공적으로 마무리 해냈다.

기본적인 것 인데, 공식문서를 잘 읽어보고 차근차근 모르는 부분은 시간을 들여 검색해가며 이해하고 실행하는것을 잊어먹고 있었다.

파일럿 프로그램을 작성하면서 React, Spring, 이 결합되어 어떻게 데이터베이스와 통신하는지 조금이나마 이해한 것 같아 기분이 좋았다.

CDC의 의미와 카프카가 필요한 이유에 대해서도 얕게나마 이해할 수 있었다.