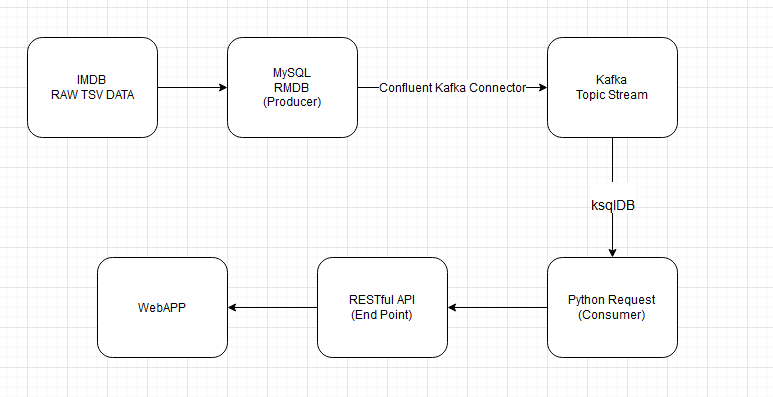
Massive Movie Data - A RESTful API Implementation

# Project Description

In this project, IMDB database (about 50M entries) were used as the source of mass-data collection. The data was imported into MySQL RMDB first, and fed into Kafka topic stream using Kafka Connector. Ksqldb web server was used to build a RESTful API that could be used access the database based on Kafka topic stream.

Kafka as a high network throughput, high performance persistent messaging platform is used to cache producer data (from MySQL). It provides O(1) access time for millions of data entries. This project is developed on a single machine but could be scaled into Kafka clusters, providing ms-scale response time.

# Project Diagram



# Project Setup

This project consists of three parts, which is described below as the direction of data flow:

First, raw TSV data downloaded from <https://www.imdb.com/interfaces/> were translated into MySQL RMDB, with the script given in attachment. Please refer to the SQL script in Attachment 1 for schema of each table.

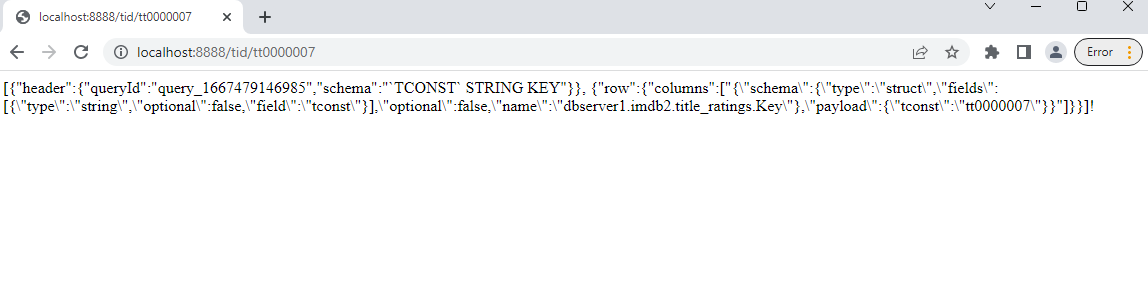
Then, we use docker-compose to build up Kafka micro-services, including zookeeper, kafka, confluent kafka connector, ksqldb-server and ksqldb-cli. Kafka Connector API was used to use MySQL DB as the source of producer. Refer Attachement 2 docker-compose.yaml file for setting up Kafka micro services.

After Kafka micro services were started up, we set up the Confluent Connector to connect MySQL DB as producer of Kafka stream. See Attachment 3 register-mysql.json to connect MySQL DB and Kafka stream. This step is translation from RMDB to high-performance Kafka stream.

After this, we use Python requests and ksqldb API to provide query into IMDB database stored in Kafka stream. Python Flask API is used to build RESTful APIs for database queries. Additionally, a web-based can be set up based upon this. Sett Attachment 4 for details.

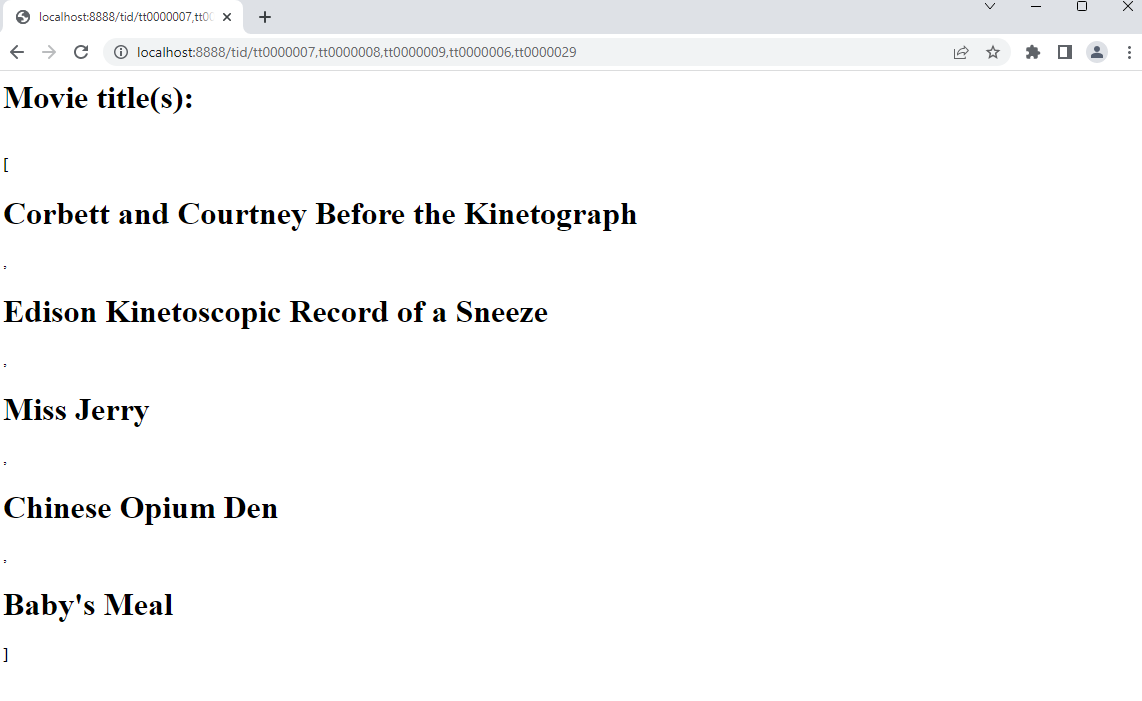
# Demostration

A RESTful call to query the Kafka stream would be like:



Additionally, we can use a single URL, where each title ids are separated by comma, to query multiple movie titles simultaneously, example:

localhost:8888/tid/tt0000007,tt0000008,tt0000009,tt0000006,tt0000029



Implementation of this depends on Python Flask and Requests libraries.

# Attachement 1: SQL Script For Setting Up MySQL DB

CREATE DATABASE imdb;

USE imdb;

# ========================

SHOW VARIABLES LIKE "secure\_file\_priv";

# ========================

show global variables like 'local\_infile';

set global local\_infile=true;

# ==========name\_basics==============

DROP TABLE name\_basics;

/\* CREATE TABLE \*/

CREATE TABLE name\_basics(

`nconst` VARCHAR(100),

`primaryName` VARCHAR(100),

`birthYear` INT,

`deathYear` INT,

`primaryProfession` VARCHAR(100),

`knownForTitles` VARCHAR(100)

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/name.basics.tsv'

INTO TABLE name\_basics

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM name\_basics LIMIT 10;

select \* from name\_basics WHERE primaryName LIKE "Fred Astaire%";

# two lines without comments, thus missing comments column

# 12029558 row(s) affected, 2 warning(s):

# 1265 Data truncated for column 'primaryName' at row 7706341

# 1265 Data truncated for column 'primaryName' at row 7941053

# Records: 12029558 Deleted: 0 Skipped: 0 Warnings: 2

SELECT \* FROM name\_basics WHERE nconst = "nm4900920" OR nconst = "nm4635165";

# ========title\_akas================

DROP TABLE title\_akas;

/\* CREATE TABLE \*/

CREATE TABLE title\_akas(

`titleId` VARCHAR(100),

`ordering` INT,

`title` VARCHAR(100),

`region` VARCHAR(10),

`language` VARCHAR(20),

`types` VARCHAR(50),

`attributes` VARCHAR(20),

`isOriginalTitle` INT

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.akas.tsv'

INTO TABLE title\_akas

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_akas LIMIT 10;

# ========title\_basics================

DROP TABLE title\_basics;

/\* CREATE TABLE \*/

CREATE TABLE title\_basics(

`tconst` VARCHAR(10),

`titleType` VARCHAR(100),

`primaryTitle` VARCHAR(100),

`originalTitle` VARCHAR(100),

`isAdult` INT,

`startYear` INT,

`endYear` INT,

`runtimeMinutes` INT,

`genres` VARCHAR(50)

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.basics.tsv'

INTO TABLE title\_basics

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_basics LIMIT 10;

# ========title\_crew================

DROP TABLE title\_crew;

/\* CREATE TABLE \*/

CREATE TABLE title\_crew(

`tconst` VARCHAR(10),

`directors` VARCHAR(100),

`writers` VARCHAR(100)

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.crew.tsv'

INTO TABLE title\_crew

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_crew LIMIT 10;

# ========title\_episode================

DROP TABLE title\_episode;

/\* CREATE TABLE \*/

CREATE TABLE title\_episode(

`tconst` VARCHAR(10),

`parentTconst` VARCHAR(100),

`seasonNumber` INT,

`episodeNumber` INT

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.episode.tsv'

INTO TABLE title\_episode

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_episode LIMIT 10;

# ========title\_principals================

DROP TABLE title\_principals;

/\* CREATE TABLE \*/

CREATE TABLE title\_principals(

`tconst` VARCHAR(10),

`ordering` INT,

`nconst` VARCHAR(100),

`category` VARCHAR(100),

`job` VARCHAR(100),

`characters` VARCHAR(100)

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.episode.tsv'

INTO TABLE title\_principals

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_principals LIMIT 10;

# ========title\_ratings================

DROP TABLE title\_ratings;

/\* CREATE TABLE \*/

CREATE TABLE title\_ratings(

`tconst` VARCHAR(10),

`averageRating` DOUBLE,

`numVotes` INT,

PRIMARY KEY (tconst)

);

LOAD DATA LOCAL INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/title.ratings.tsv'

INTO TABLE title\_ratings

FIELDS TERMINATED BY '\t'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

SELECT \* FROM title\_ratings LIMIT 10;

# Attachement 2: docker-compose.yaml for producer data

version: '2'

services:

zookeeper:

image: quay.io/debezium/zookeeper:2.0

ports:

- 21813:2181

- 2888:2888

- 3888:3888

kafka:

image: quay.io/debezium/kafka:2.0

ports:

- 9092:9092

links:

- zookeeper

environment:

- ZOOKEEPER\_CONNECT=zookeeper:2181

# mysql:

# image: quay.io/debezium/example-mysql:2.0

# ports:

# - 3307:3306

# environment:

# - MYSQL\_ROOT\_PASSWORD=debezium

# - MYSQL\_USER=mysqluser

# - MYSQL\_PASSWORD=mysqlpw

connect:

image: quay.io/debezium/connect:2.0

ports:

- 8083:8083

links:

- kafka

# - mysql

- ksqldb-server

- ksqldb-cli

environment:

- BOOTSTRAP\_SERVERS=kafka:9092

- GROUP\_ID=1

- CONFIG\_STORAGE\_TOPIC=my\_connect\_configs

- OFFSET\_STORAGE\_TOPIC=my\_connect\_offsets

- STATUS\_STORAGE\_TOPIC=my\_connect\_statuses

ksqldb-server:

image: confluentinc/ksqldb-server:0.28.2

hostname: ksqldb-server

container\_name: ksqldb-server

ports:

- "8088:8088"

links:

- kafka

environment:

KSQL\_LISTENERS: http://0.0.0.0:8088

KSQL\_BOOTSTRAP\_SERVERS: kafka:9092

KSQL\_KSQL\_CONNECT\_URL: http://connect:8083

KSQL\_KSQL\_LOGGING\_PROCESSING\_STREAM\_AUTO\_CREATE: "true"

KSQL\_KSQL\_LOGGING\_PROCESSING\_TOPIC\_AUTO\_CREATE: "true"

ksqldb-cli:

image: confluentinc/ksqldb-cli:0.28.2

container\_name: ksqldb-cli

depends\_on:

- ksqldb-server

links:

- kafka

- ksqldb-server

entrypoint: /bin/sh

tty: true

# Attachment 3: register-mysql.json

{

"name": "imdb-connector",

"config": {

"connector.class": "io.debezium.connector.mysql.MySqlConnector",

"tasks.max": "1",

"database.hostname": "host.docker.internal",

"database.port": "3306",

"database.user": "debezium",

"database.password": "dbz",

"database.server.id": "184054",

"topic.prefix": "dbserver1",

"database.include.list": "imdb",

"table.cinlude.list": "title\_ratings",

"schema.history.internal.kafka.bootstrap.servers": "kafka:9092",

"schema.history.internal.kafka.topic": "schema-changes.imdb"

}

}

# Attachment 4: imdb.py (RESTful API)

import json  
import requests  
from flask import Flask  
  
app = Flask(\_\_name\_\_)  
url = "http://localhost:8088/query/"  
headers = {'Content-type': 'application/json', 'Accept': 'application/vnd.ksql.v1+json'}  
  
  
@app.route('/tid/<tConst>')  
def title\_query(tConst):  
 sql = f"SELECT tconst FROM FF WHERE tconst LIKE '%{tConst}%';"  
 print(sql)  
 data = {"ksql": sql, "streamsProperties": {}}  
 r = requests.post(url, data=json.dumps(data), headers=headers)  
 return f'{r.text}!'  
  
  
@app.route('/actor/<actorName>')  
def title\_query(actorName):  
 sql = f"SELECT tconst FROM EE WHERE primaryName LIKE '%{actorName}%';"  
 print(sql)  
 data = {"ksql": sql, "streamsProperties": {}}  
 r = requests.post(url, data=json.dumps(data), headers=headers)  
 return f'{r.text}!'  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(port=8888, debug=True)