Project Documentation: Data Pipeline Setup with Change Data Capture (CDC) Concept

Introduction

This document outlines the steps involved in setting up a data pipeline with Change Data Capture (CDC) concept, utilizing MySQL as a transaction database and MySQL as a data warehouse. The pipeline involves generating fake data, configuring MySQL for CDC, connecting MySQL with Kafka using Debezium, and finally loading data changes into the data warehouse.

Project Overview

The project aims to create a robust data pipeline that captures real-time data changes from a transactional MySQL database and loads them into a PostgreSQL data warehouse using CDC principles.

Step 1: Creating MySQL Database and Defining Schema

1. Access MySQL Shell:
    - Open a terminal or command prompt.
    - Log in to MySQL shell using the appropriate credentials:

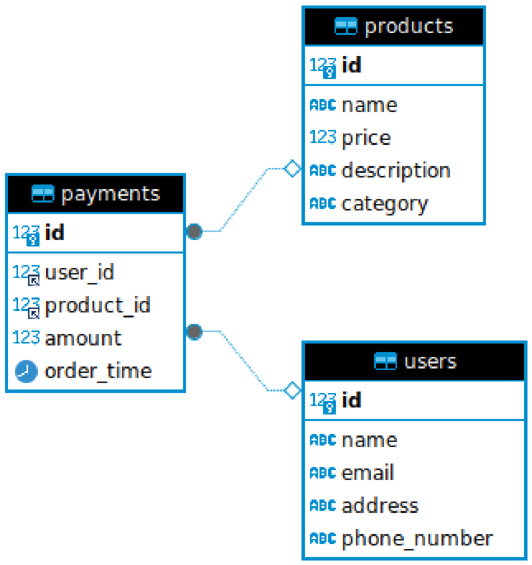
    ```
    mysql -u your_username -p
    ```

    - Enter your password when prompted.

2. Create MySQL Database:
    - Execute the following SQL command to create a new database named 'transactions' within the MySQL shell:

    ```
    CREATE DATABASE transactions;
    ```

3. Create tables:



```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255),
    address VARCHAR(255),
    phone_number VARCHAR(15)
);
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    price DECIMAL(10, 2),
    description TEXT,
    category VARCHAR(50)
);

CREATE TABLE payments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    product_id INT,
    amount DECIMAL(10, 2),
    order_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Step 2: Generating Fake Data

1. Utilize the `faker` library within the `data/generate_fake_data.py` script to generate synthetic data for the "users," "products," and "payments" tables in the transaction database.

Step 3: Setting up MySQL Server for CDC

1. Configure MySQL for CDC by enabling binary logging and setting appropriate configurations in the `mysqld.cnf` file. Add the following line, to conf file.

```
server-id           = 1
log_bin             = /var/log/mysql/mysql-bin.log
binlog-format       = row
```

2. Restart MySQL service to apply the configuration changes.

```
sudo systemctl restart mysql
```

Step 4: Installing Kafka

1. Install Java 8 on the designated server for Debezium.
2. Download and extract Kafka binaries.
3. Move Debezium MySQL Connector plugin to Kafka's plugins directory.

```
cd kafka-3.6.1-src
wget
https://repo1.maven.org/maven2/io/debezium/debezium-connector-mysql/
1.8.0.Final/debezium-connector-mysql-1.8.0.Final-plugin.tar.gz
tar -xvzf debezium-connector-mysql-1.8.0.Final-plugin.tar.gz

mv debezium-connector-mysql-1.8.0.Final-plugin plugins
```

Step 5: Configuring Debezium MySQL Connector

1. Create a properties file (`connect-debezium-mysql.properties`) with the necessary configurations for Debezium MySQL Connector.
2. Specify connector name, database details, Kafka bootstrap servers, and other relevant settings.

```
name=mysql-connector
connector.class=io.debezium.connector.mysql.MySqlConnector
tasks.max=1
database.hostname=localhost
database.port=3306
database.user=root
database.password=dwhPassWprd
database.server.id=1
database.history.kafka.topic=db_history_topic
database.server.name=mysql-connector
database.dbname=dwh
database.history.kafka.bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
```

schemas.enable=false

3. Create a Kafka topic (db_history_topic) for Debezium's database history.
4. Start Zookeeper, Kafka, and Kafka Connect services.

After completing all the tasks, I encountered an issue: I could only observe schema change transactions in the database logs, while the topic lacked row-level changes such as inserts, updates, and deletes. Despite investing several hours in troubleshooting, I couldn't find a solution. Consequently, I turned to `pymysqlreplication` in Python to read the binary log files and transform the changes into our data warehouse.

Step 6: Create MySQL Data Warehouse

Access MySQL Shell:

- Open a terminal or command prompt.
- Log in to MySQL shell using the appropriate credentials:

```
mysql -u your_username -p
```

- Enter your password when prompted.

Create MySQL Database:

- Execute the following SQL command to create a new database named 'transactions' within the MySQL shell:

```
CREATE DATABASE dwh;
```

Create tables:

```
CREATE TABLE dwh.order_fact (
 `id` int NOT NULL AUTO_INCREMENT,
 `user_id` int DEFAULT NULL,
 `product_id` int DEFAULT NULL,
 `amount` decimal(10,2) DEFAULT NULL,
 `order_time` datetime DEFAULT NULL,
 PRIMARY KEY (`id`),
 KEY `user_id` (`user_id`),
 KEY `product_id` (`product_id`)
) ENGINE=InnoDB AUTO_INCREMENT=2001 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

```sql
-- transactions.products definition
 CREATE TABLE dwh.products_dim (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `description` text,
  `category` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
 ) ENGINE=InnoDB AUTO_INCREMENT=401 DEFAULT CHARSET=utf8mb4
 COLLATE=utf8mb4_0900_ai_ci;

 -- transactions.users definition
 CREATE TABLE dwh.users_dim (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  `address` text,
  `phone_number` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
 ) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb4
 COLLATE=utf8mb4_0900_ai_ci;
```

Step 7: Loading Data into MySQL Data Warehouse

1. Implement a Python script (`dag_etl_cdc.py`) using `pymysqlreplication` to access MySQL binlog files.
2. Create a data warehouse (DWH) in PostgreSQL.
3. Write ETL logic to transform and load data changes from MySQL binlog files into the PostgreSQL DWH.
4. Schedule and orchestrate the ETL process using Apache Airflow or similar workflow management tools.

Challenges:

For scheduling Apache Airflow tasks, I aim to execute a Directed Acyclic Graph (DAG) every 15 minutes. During each DAG run, I intend to set the start_date as the current DAG date and the end_date as 15 minutes later. Ideally, I'd filter logs within this time window into bin files. However, due to time constraints, I opted to run the DAG only once.

Additionally, I plan to utilize MySqlHook to establish a MySQL connection within Airflow. Unfortunately, I encountered difficulties installing this connector. Consequently, I choose to postpone this aspect as well.

Conclusion

The data pipeline setup successfully establishes a mechanism for capturing real-time data changes from a MySQL transaction database using Debezium and Kafka. By leveraging CDC principles, the pipeline ensures efficient and reliable data synchronization with a MySQL data warehouse, enabling timely analytics and reporting.