**MySQL to S3 Data Migration using Docker, Debezium, Kafka, and Pyspark**

**Description** : Seamless solution for migrating data from a MySQL database to Amazon S3 using modern data processing technologies. By leveraging Docker containers, Debezium connectors, Apache Kafka, and Pyspark, this system ensures a smooth and efficient data flow, enabling real-time synchronization and analysis.

**HIGH LEVEL DESIGN:**

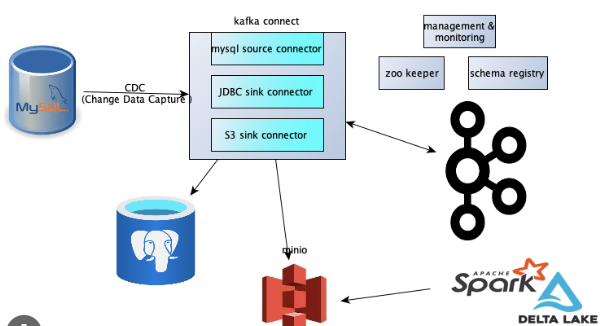
|  |
| --- |
|  |

**Prerequisites :**

* **Docker Installation**: Ensure Docker is installed locally.
* **Debezium Installation:** Install Debezium for change data capture (CDC).
* **Docker Compose File:** Create a Docker Compose file with configurations for MySQL, Zookeeper, and Kafka services.

**Steps for Real-Time Processing:**

**Pipeline Snippet:**

****

**1.Docker Compose:**

* Create a docker-compose.yml file with service configurations.
* Run docker-compose up to create and start the containers.
* MySQL Python Script (mysql.py)
* Implement a Python script (mysql.py) to insert data into MySQL.

**2.Debezium Connector:**

* Launch Debezium connector to capture changes from MySQL.
* Use the connector to publish data to Kafka topics in JSON format.

**3.Create Kafka Topic:**

* Use curl command to launch the connector and create Kafka topics.

|  |
| --- |
| **curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json"** [**http://localhost:8083/connectors/**](http://localhost:8083/connectors/) **-d '{**  **"name": "mysql-connector",**  **"config": {**  **"connector.class": "io.debezium.connector.mysql.MySqlConnector",**  **}**  **}'** |

**4.Consume Kafka Topics:**

* Develop CDC (Change Data Capture) and snapshot scripts to consume Kafka topics.
* Implement logic to write consumed data into Apache Iceberg format.

**A diagram of a software application

Description automatically generated**

**Execution:**

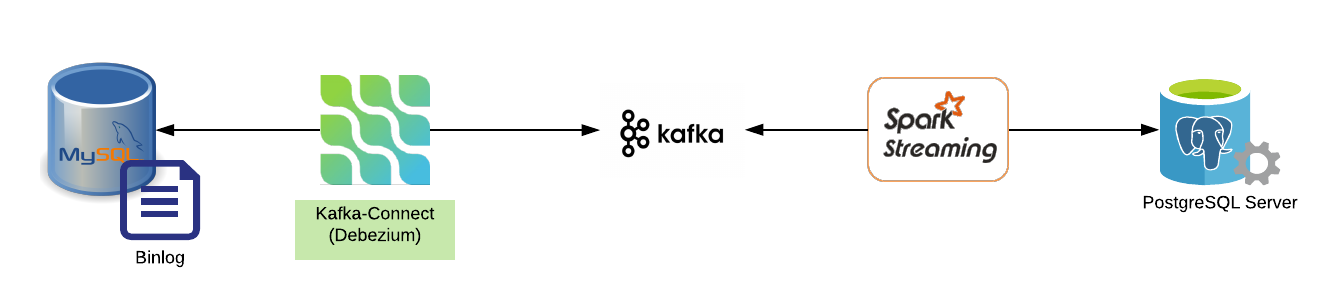
* Execute docker-compose up to start MySQL, Zookeeper, and Kafka services.
* Run python mysql.py to insert data into MySQL.
* Launch Debezium connector to capture changes.
* Use curl command to create Kafka topics.
* Run scripts for consuming Kafka topics, processing CDC, and snapshot data.
* Processed data will be written into Apache Iceberg.
* Ensure proper configurations and dependencies are set for each step to achieve real-time processing of Kafka topics from MySQL.

**Snapshot (History Load from MySQL to Iceberg):**

* **AWS Cluster Setup:**Ensure the AWS cluster is up and running with required configurations.
* **Open Workspace:**Access the AWS workspace named "xyz" for data processing.
* **Jupyter Notebook Setup:**Open Jupyter Notebook using the script named "xyz" in the workspace.
* **Snapshot Execution:**Execute the Jupyter Notebook script to perform a snapshot.

**Note : The script should include logic to connect to MySQL, retrieve historical data, and write it to Iceberg.**

**CDC (Incremental Load from MySQL to Iceberg):**



File store in iceberg format

**Mysql Debezium kafka consumer iceberg-format**

* **AWS Cluster Setup:**Ensure the AWS cluster is up and running with necessary configurations.
* **Open Workspace:**Access the AWS workspace named "xyz" for data processing.
* **Jupyter Notebook Setup:**Open Jupyter Notebook using the script named "xyz" in the workspace.
* **CDC Execution:**Execute the Jupyter Notebook script to perform incremental CDC.

The script should utilize change data capture mechanisms to fetch and process only the incremental changes.

Write the incremental changes to the Iceberg table.

|  |
| --- |
| Note:  Ensure that AWS cluster configurations align with the requirements of the snapshot and CDC processes.  Jupyter Notebook scripts should have appropriate error handling, logging, and AWS SDK integration for seamless execution.  Verify that the necessary IAM roles and permissions are assigned for data access and processing.  These steps outline the low-level design for performing a snapshot (history load) and CDC (incremental load) from MySQL to Iceberg using AWS infrastructure and Jupyter Notebooks. |

**How It Works :**

**1. \*\*Debezium Captures Changes\*\***: Debezium, a CDC (Change Data Capture) platform, monitors the MySQL database in real-time. It captures any changes made to the database, such as inserts, updates, and deletes.

**2. \*\*Data Flows to Kafka\*\***: Debezium feeds these changes into Apache Kafka topics, where they are organized and made available for consumption. Kafka acts as the messaging backbone, ensuring fault tolerance and scalability.

**3. \*\*PySpark Processes Data\*\***: A PySpark job is responsible for processing the data received from Kafka topics. PySpark provides a powerful and scalable processing framework that can handle large datasets efficiently. Here, you can implement transformations, aggregations, or any data manipulations as needed.

**4. \*\*Data Lands in S3\*\***: After processing, the PySpark job transfers the data to Amazon S3, a robust and scalable cloud storage service. S3 ensures durability, availability, and security for your data, making it an ideal choice for storing valuable business information.

**Key Components**

**1.\*\*Docker Containers\*\***: Docker simplifies the setup by encapsulating each component within isolated containers, ensuring consistency across different environments.

2.**\*\*Debezium Connectors\*\***: Debezium connectors capture changes at the database level, enabling real-time data streaming without the need for complex ETL processes.

**3.\*\*Apache Kafka\*\***: Kafka acts as the message broker, efficiently handling data streaming between Debezium and PySpark components.

**4.\*\*PySpark\*\***: PySpark, the Python library for Apache Spark, provides a high-level API for distributed data processing. Its flexibility and scalability make it suitable for handling various data processing tasks.

**-5.\*\*Amazon S3\*\***: Amazon S3 offers a reliable and cost-effective storage solution for your processed data. Its integration capabilities and global availability make it a preferred choice for data storage and archival.