Data Engineering Technical Challenge

Introduction

You will work within a local Kubernetes environment (Kind) to integrate data from multiple sources into a central analytics store.

Your primary focus should be on the reliability and correctness of the data pipeline. We value solutions that are not just functional but also resilient and maintainable. Pay close attention to data integrity, idempotency, and performance.

Architecture Overview

The target architecture involves capturing data from two distinct sources and processing it through a streaming pipeline into an analytical database. The core components are:

- **Kind Cluster**: A local Kubernetes cluster for all services.
- **PostgreSQL**: A transactional database serving as the system of record for core business entities.
- **MongoDB**: A NoSQL database used for storing semi-structured event data.
- **Kafka**: The central nervous system for our real-time data streams.
- **Debezium**: The Change Data Capture (CDC) platform to stream changes from our source databases into Kafka.
- **ClickHouse (Altinity Operator)**: A high-performance, column-oriented SQL database for real-time analytics.
- **Airflow**: The orchestration platform for scheduling, monitoring, and executing data workflows.

The Challenge

The challenge is divided into a few parts. While a basic implementation might seem straightforward, we encourage you to think about the edge cases and potential failure modes that can occur in a production system.

Part 1: Environment Setup

Set up a clean, local development environment using Kind. All subsequent components will be deployed into this cluster.

Tasks:

1. Create a new Kind cluster.

```shell
kind create cluster --name data-engineering-challenge
```

2. Install the Strimzi operator to manage Kafka clusters. (Kraft)
3. Install the Altinity ClickHouse operator to manage ClickHouse clusters.
4. Deploy a PostgreSQL instance. You will need to configure it to enable logical replication (`wal_level = logical`).
5. Deploy a MongoDB instance.

Part 2: Multi-Source CDC Pipeline

Establish a real-time data stream from both PostgreSQL and MongoDB into Kafka. The goal is to capture every change with high fidelity.

Tasks:

1. In **PostgreSQL**, create a `users` table (`user_id`, `full_name`, `email`, `created_at`, `updated_at`). Populate it with some sample data. Perform some updates and deletes on this data.
2. In **MongoDB**, create an `events` collection. Populate it with documents representing user actions (e.g., `login`, `page_view`, `add_to_cart`). These events should reference a `user_id`.
3. Deploy a Kafka cluster.
4. Deploy Kafka Connect with the Debezium connectors for both PostgreSQL and MongoDB.
5. Configure the PostgreSQL connector
6. Configure the MongoDB connector

Part 3: Real-time Ingestion and Transformation in ClickHouse

Ingest the raw CDC data from Kafka into ClickHouse, ensuring that the state in ClickHouse accurately reflects the state in the source databases, including updates and deletes.

Tasks:

1. Deploy a ClickHouse cluster (CHI & CHK).
2. Create the necessary Kafka engine tables to consume from the `postgres.public.users` and `mongo.commerce.events` topics.
3. Design and implement a `silver_users` table in ClickHouse that accurately reflects the current state of the `users` table in PostgreSQL. It should correctly handle inserts, updates, and deletes from the CDC stream, deletes are soft-only.
4. Create a `silver_events` table to store the raw event data from MongoDB.

Part 4: Batch Orchestration with Airflow

Use Airflow to build a reliable, daily batch process that transforms and aggregates the data into a final reporting model.

Tasks:

1. Deploy Airflow on your Kind cluster.
2. Create an Airflow DAG that runs once daily. This DAG must be idempotent and capable of being backfilled without creating data duplication.
3. The DAG should perform the following logic:
    - Join the `silver_users` and `silver_events` tables to create a `gold_user_activity` table.
    - This `gold_user_activity` table should summarize, for each user, their total number of events and the timestamp of their last event for the previous day.

You can use docker compose in-case kind cluster deployment is complex. k8s kind is always preferred.

Deliverables

- A Git repository containing all the necessary YAML files, Dockerfiles, and Airflow DAGs. Make sure it is a public repo.
- A `README.md` file in the repository with clear instructions on how to set up the environment and run your solution.
- Send us the solution link (your repo) through linkedin before the below due date.

Due Date: 1 March 2026 at 11:59 pm

Good luck!