Data-Engineer-Assignment 2

Data Engineering Assignment 2: Delta Lake, Spark, ScyllaDB, and Airflow Pipeline

Overview

This assignment assesses your practical skills in building a modern data engineering pipeline using Delta Lake, Apache Spark, ScyllaDB, and Apache Airflow. You are required to run the complete solution locally using a containerized setup with either Docker Compose or Kubernetes (minikube/kind/k3d).

Objectives

- Work with Delta Lake for data storage and versioning
- Implement data transformations using Apache Spark
- Load processed data into ScyllaDB database
- Orchestrate the pipeline using Apache Airflow
- Demonstrate proficiency with containerized environments

Assignment Tasks

Task 1: Environment Setup

Set up a containerized environment with the following services:
- Delta Lake (with Spark)
- Apache Spark
- ScyllaDB database
- Apache Airflow

Requirements:
- Create a `docker-compose.yml` file to orchestrate all services OR provide Kubernetes manifests
- Ensure proper inter-container networking between all services

- Include persistent data volumes for data persistence
- Configure necessary drivers and connectors for ScyllaDB connectivity

Delta Lake Setup:

- Create Delta Lake storage locally on your host machine (e.g., `./data/delta-lake/`)
- Mount the local Delta Lake directory as a volume in your Spark container
- This allows for persistent Delta table storage with versioning capabilities
- Ensure proper read/write permissions for the mounted volume

Task 2: Sample Data Generation and Upload

Create a sample dataset matching the customer transactions format with at least 1000 records.

Input Data Format:

```
transaction_id,customer_id,amount,timestamp,merchant
```

Column Name	Type	Description
transaction_id	String	Unique transaction identifier
customer_id	String	Customer’s unique identifier
amount	Float	Transaction amount
timestamp	Timestamp	Transaction timestamp (UTC)
merchant	String	Merchant name

Sample Data Example:

```
{
  "transaction_id": "d74b5cda-f254-4ac0-b053-9493fa15ac8a",
  "customer_id": "C12345",
  "amount": 150.36,
  "timestamp": "2025-11-30T15:37:49Z",
  "merchant": "STORE_23"
}
```

Data Generation Requirements:

- Create a Python script using Faker library or similar to generate sample data

- Include various data quality scenarios (duplicates, zero/negative amounts, etc.)
- Create the Delta Lake table `customer_transactions` locally
- Mount the local Delta Lake directory (e.g., `./data/delta-lake/`) as a volume in containers
- Upload/import your generated data into the Delta Lake table
- Verify Delta table versioning is working (show table history)

Task 3: Data Extraction and Transformation

Write a Spark job that performs the following operations:

3.1 Data Extraction

- Extract data from the Delta Lake table `customer_transactions`

3.2 Data Quality and Transformations

- **Deduplication**: Remove duplicate transactions based on `transaction_id`
- **Data Validation**: Filter out transactions where `amount <= 0`
- **Date Processing**: Add a new column `transaction_date` (extract date part from `timestamp`)
- **Aggregation**: Calculate daily totals for each `customer_id`

3.3 Data Processing Output

The transformed data should include:
- `customer_id`: Customer identifier
- `transaction_date`: Date of transactions
- `daily_total`: Sum of all valid transactions for that customer on that date

Task 4: ScyllaDB Database Design and Loading

4.1 Database Schema

Design and create ScyllaDB table:

Table: **`daily_customer_totals`**

```
CREATE TABLE daily_customer_totals (
    customer_id TEXT,
    transaction_date DATE,
    daily_total FLOAT,
    PRIMARY KEY (customer_id, transaction_date)
);
```

Schema Details:
- `customer_id`: Partition key
- `transaction_date`: Clustering key
- `daily_total`: Aggregated transaction amount

4.2 Data Loading

- Upsert the daily totals into the `daily_customer_totals` table in ScyllaDB
- Implement proper error handling and connection management
- Ensure data consistency and proper partitioning

Task 5: Airflow Orchestration

5.1 DAG Development

Create an Airflow DAG that automates the complete ETL pipeline:

DAG Requirements:
- **Task 1**: Data extraction from Delta Lake
- **Task 2**: Data transformation using Spark
- **Task 3**: Data loading to ScyllaDB
- Proper task dependencies and error handling
- Appropriate retry mechanisms and alerting

5.2 DAG Configuration

- Schedule the DAG appropriately (daily recommended)
- Include proper logging and monitoring
- Implement data quality checks between tasks

Task 6: Containerization and Deployment

6.1 Container Setup (Choose one approach)

Option A: Docker Compose

- Provide `docker-compose.yml` with all required services
- Include persistent volume configurations
- Ensure proper networking between containers
- Document port mappings and access methods

Option B: Kubernetes

- Supply deployment manifests for all services
- Include service definitions and persistent volume claims
- Use minikube/kind/k3d for local environment
- Provide clear setup and deployment instructions

6.2 Service Configuration

Ensure the following services are properly configured:

- **Spark**: Master and worker nodes with Delta Lake support and mounted volume access
- **ScyllaDB**: Proper keyspace and table initialization
- **Airflow**: Webserver, scheduler, and executor configuration with volume mounts
- **Delta Lake**: Local storage mounted as volume (e.g., `./data/delta-lake:/opt/spark/delta-lake`)

6.3 Volume Mounting Best Practices

Local Delta Lake Setup:

```
# Create local directory structure
mkdir -p ./data/delta-lake/customer_transactions
```

Docker Compose Volume Example:

```
volumes:  - ./data/delta-lake:/opt/spark/delta-lake
```

Task 7: Documentation and Code Organization

Provide the following deliverables:

1. **README.md** with comprehensive setup and execution instructions
2. **docker-compose.yml** or **Kubernetes manifests** for environment setup
3. **Data generation script** (Python/Scala) with Faker or similar
4. **Spark ETL job code** (well-commented Python/Scala)
5. **Airflow DAG definition** with proper task dependencies
6. **ScyllaDB initialization scripts** for schema creation
7. **Delta Lake table creation and versioning examples**
8. **Sample input and output data**

Evaluation Criteria

Technical Implementation (35%)

- Correct Spark transformations and Delta Lake integration
- Proper ScyllaDB connectivity and data modeling
- Airflow DAG design and task orchestration
- Code organization and best practices

Containerization & Infrastructure (30%)

- Proper containerization of all services
- Service orchestration and networking
- Resource management and persistence
- Local environment setup and documentation

Data Quality and Pipeline Design (20%)

- Comprehensive data processing logic
- Appropriate error handling and data validation
- Pipeline reliability and monitoring
- Performance considerations

Documentation and Presentation (15%)

- Clear setup and execution instructions
- Code comments and architectural documentation
- Data pipeline flow explanation
- Troubleshooting guide and logs

Submission Guidelines

1. Create a Git repository with all code, configurations, and documentation
2. Include a demo video (5 minutes or so) showing:
    - Environment setup and service startup
    - Data generation and upload to Delta Lake (show local directory structure)
    - Delta Lake versioning demonstration (table history, time travel queries)
    - Airflow DAG execution
    - Data verification in ScyllaDB
3. Provide sample input data and expected output examples
4. Include troubleshooting section for common issues.

Architecture Overview

The expected pipeline architecture:

```
[Data Generation] → [Delta Lake (Local Mount)] → [Spark ETL] → [ScyllaDB]
                        ↑
                   [Airflow DAG Orchestration]
```

Component Responsibilities:

- **Delta Lake**: Source data storage with ACID transactions and versioning (locally mounted)
- **Spark**: Data processing and transformation engine (accessing mounted Delta Lake)

- **ScyllaDB**: High-performance target database for aggregated results
- **Airflow**: Workflow orchestration and scheduling

Data Flow and Persistence:

```
Host Machine (./data/delta-lake/) ←→ Container (/opt/spark/delta-lake/)
                 ↓
           [Delta Table Versioning]
                 ↓
            [Spark Processing]
                 ↓
            [ScyllaDB Storage]
```

Sample Data Structure

Here's the expected structure for your customer transactions data:

```
transaction_id,customer_id,amount,timestamp,merchant
d74b5cda-f254-4ac0-b053-9493fa15ac8a,C12345,150.36,2025-11-30T15:37:49Z,STORE_23
e85c6deb-a365-5bd1-c164-0594gb26bd9b,C67890,-25.00,2025-11-30T10:22:15Z,ONLINE_SHOP
d74b5cda-f254-4ac0-b053-9493fa15ac8a,C12345,150.36,2025-11-30T15:37:49Z,STORE_23
a92d7fec-b476-6ce2-d275-1605hc37ce0c,C11111,89.99,2025-11-29T14:18:32Z,GAS_STATION
```

Note: This sample shows duplicate transactions and negative amounts that should be handled during processing.

Questions?

If you have any questions about the assignment requirements, technology stack, or implementation approach, please don't hesitate to ask for clarification.

Good luck!