Data Engineer Take Home Test 2026

Introduction

In this test, you will design and implement an end-to-end data pipeline from source to destination. We have set up **PostgreSQL** and **Kafka** for you to use in this assignment.

Your main objectives are:

1. Use PostgreSQL as both source and data warehouse
2. Design a **batch pipeline** for daily data processing
3. Design a **streaming pipeline** for real-time data processing
4. Demonstrate your understanding of ETL principles and containerization

Assessment Criteria: We will evaluate your understanding of Data Engineering fundamentals, code quality, and ability to work with modern data tools.

Files Included

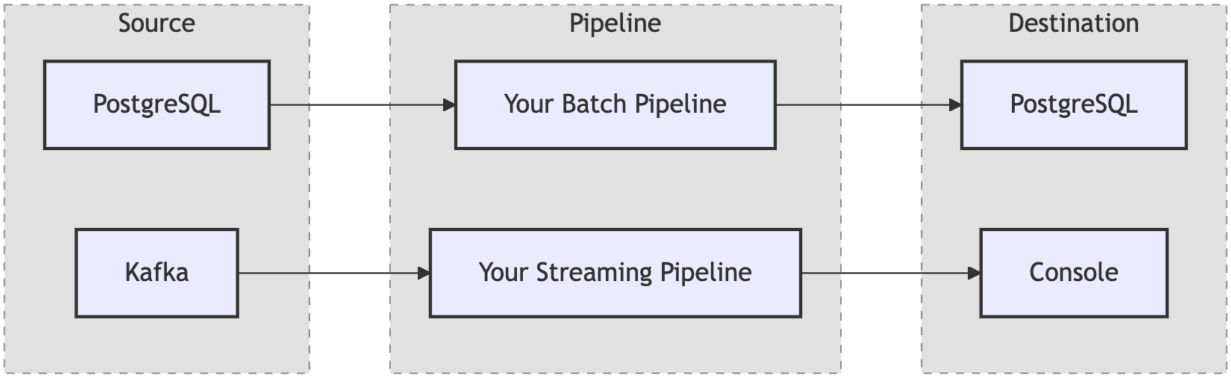
Shell

```
assignment/
├── docker-compose.yaml     # Container orchestration
├── kafka.py                # Script to load data to Kafka
├── pg.py                   # Script to load data to PostgreSQL
├── requirements.txt        # Python dependencies
├── products.csv            # Product master data
├── orders.csv              # Order transaction data
├── problem-0/              # Setup folder
│   ├── products.sql        # (Fill this) Products table
│   └── orders.sql          # (Fill this) Orders table schema
├── problem-1/              # Batch pipeline folder
│   └── answer.md           # (Fill this) SQL answers
└── problem-2/              # Streaming pipeline folder
```

Data Overview

Architecture

Source (PostgreSQL/Kafka) → Pipeline (Your Code) → Destination (PostgreSQL)



Products Data (products.csv)

Sample product catalog:

product_id	product_name	price	category
PROD-001	Coffee	45	Beverage
PROD-002	Green Tea	40	Beverage
PROD-003	Sandwich	85	Food

Orders Data (orders.csv)

Sample order transactions:

order_id	order_timestamp	user_id	product_id	quantity	status
ORD-00001	2024-01-01 11:00:00	USER-001	PROD-001	2	COMPLETE
ORD-00002	2024-01-01 12:00:00	USER-002	PROD-003	1	COMPLETE
ORD-00003	2024-01-02 08:00:00	USER-001	PROD-002	1	COMPLETE

Note: Only orders with status = 'COMPLETE' should be included in revenue calculations.

Problem 0 - Environment Setup

Tasks

1. **Install Docker** on your local machine (if not already installed)
2. **Start services** using docker-compose up -d
3. **Verify Kafka UI** is accessible at http://localhost:8080
4. **Define table schemas** in the following files:
    a. problem-0/products.sql
    b. problem-0/orders.sql
5. **Load data to Kafka** by running python kafka.py
6. **Load data to PostgreSQL** by running python pg.py
7. **Verify data** is correctly loaded in both Kafka and PostgreSQL

Kafka UI Verification

After running kafka.py, you should see:

- Topic: orders (with messages)
- Topic: products (with messages)

PostgreSQL Verification

After running pg.py, verify with:

```
SELECT COUNT(*) FROM orders;   – Result: 1000000 records
SELECT COUNT(*) FROM products;  – Result: 10 records
SELECT * FROM orders LIMIT 5;
SELECT * FROM products LIMIT 5;
```

Deliverables

Two SQL files with CREATE TABLE statements:

- assignment/problem-0/sql/products.sql
- assignment/problem-0/sql/orders.sql

Hints:

- Look at the CSV files to understand column types
- Consider appropriate data types (VARCHAR, INTEGER, DATE, etc.)
- Think about which columns should be primary keys

Problem 1 - Batch Pipeline

Business Requirements

Your analytics team needs daily reports to monitor business performance. Design a batch ETL pipeline with these requirements:

1. **Schedule:** Data must be ready for querying every morning at 8 AM
2. **Key Metrics to Calculate:**
   a. Daily total revenue (only COMPLETE orders)
   b. Daily number of new customers (first-time buyers)
   c. Daily order count by product category
3. **Incremental Processing:** Pipeline should process only new data each day (not reprocess everything)

Technical Requirements

- **Source:** Read from PostgreSQL tables (orders, products)
- **Processing: Apache Spark is mandatory** for batch processing. Use PySpark to build your batch pipeline.
- **Destination:** Write aggregated results to PostgreSQL (you design the schema)
- **Orchestration:** Airflow (should include in docker)

SQL Questions

Question 1: Find the total revenue for user USER-001 across all completed orders.

Question 2: Find how many completed orders were placed on 2024-01-03.

Question 3: Find the average daily order count for the 'Beverage' product category in January 2024.
Question 4: Find the date with the highest number of new customers acquired.

Deliverables

The deliverables are broken down into their respective directories:

1. Batch Pipeline Code (src/ **and** dags/**):**

- **Core PySpark application logic**, including data extraction, transformation (joins, aggregations), loading to destination table(s), and incremental processing logic, placed under assignment/problem-1/src/.
- **Airflow DAGs** for orchestration placed under assignment/problem-1/dags/.
- **Containerization:** assignment/problem-1/Dockerfile and updated assignment/problem-1/docker-compose.yml for the pipeline.
- **Documentation:** assignment/problem-1/README.md explaining how to run the code.

2. Data Model (doc/ **and** sql/**):**

- **Design Document:** assignment/problem-1/doc/data_model.png or data_model.md showing your destination table schema(s), explanation of design choices, and an ER diagram or data model diagram.
- **Destination Table DDL:** assignment/problem-1/sql/problem-1-<table_name>.sql (DDL for your new table(s)).
- **Solution Explanation:** assignment/problem-1/doc/solution.md explaining the overall batch solution.

3. SQL Queries + Answers (sql/ **and** results/**):**

- **SQL Query Files:** Dedicated SQL files for the required questions, placed under assignment/problem-1/sql/:
    - problem-1-ddl-<table_name> ()
    - problem-1-dml-1.sql (Question 1)
    - problem-1-dml-2.sql (Question 2)
    - problem-1-dml-3.sql (Question 3)
    - problem-1-dml-4.sql (Question 4)
- **Results:** Screen captures of the SQL query results placed under assignment/problem-1/results/:
    - problem-1-question-1.png
    - problem-1-question-2.png
        - ...and so on for all questions.

Problem 2 - Streaming Pipeline

Business Requirements

The business team wants **real-time monitoring** of order activity. They need to see:

- **Order count per minute** for each product
- **Update frequency:** Within 1 minute of order placement

This will power a real-time dashboard showing live order activity.

Technical Requirements

- **Source:** Kafka topic orders
- **Processing:** Streaming framework (e.g., Flink, or Spark Streaming)
- **Output:** Can be to console output, or file
- **Aggregation Window:** 1-minute tumbling window

Example Output

```
product_id  | window_start        | window_end          | order_count
------------|---------------------|---------------------|------------
PROD-001    | 2024-01-01 10:00:00 | 2024-01-01 10:01:00 | 5
PROD-002    | 2024-01-01 10:00:00 | 2024-01-01 10:01:00 | 3
PROD-001    | 2024-01-01 10:01:00 | 2024-01-01 10:02:00 | 2
```

Deliverables

The deliverables are organized under the problem-2/ directory:

1. **Streaming Pipeline Code:**
   - assignment/problem-2/streaming_pipeline.py (or your preferred language) containing the Kafka consumer setup, window aggregation logic, and output mechanism.
2. **Documentation and Results:**
   - **Solution Explanation:** assignment/problem-2/doc/solution.md explaining your streaming solution.
   - **Results:** Screen capture of the example output in assignment/problem-2/results/problem-2-question-1.png.
   - **Documentation:** assignment/problem-2/README.md explaining how to run the streaming pipeline code.

Hints

- Start with a simple Kafka consumer that prints messages
- Add windowing logic incrementally
- Test with the existing data first, then consider how it would work with real-time data
- You can simulate real-time data by producing messages slowly to Kafka

Submission Guidelines

File Structure

Shell

```
assignment/
├── problem-0/
│   └── sql
│       ├── products.sql
│       └── orders.sql
├── problem-1/
│   ├── results # screen capture
│   │   ├── problem-1-question-1.png  # result for question 1
│   │   ├── problem-1-question-2.png  # result for question 2
│   │   └── ...
│   ├── dags/
│   ├── plugins/
│   ├── src/    # application code
│   ├── tests/
│   ├── Dockerfile
│   ├── docker-compose.yml
│   ├── doc/
│   │   ├── solution.md # explain your solution
│   │   └── data_model.png # data model image (or .md)
│   ├── sql     # required SQL files
│   │   ├── problem-1-ddl-<table_name>.sql   # ddl for your new table
│   │   ├── problem-1-dml-1.sql   # sql for question 1
│   │   ├── problem-1-dml-2.sql   # sql for question 2
│   │   └── ...sql
│   ├── config/
│   └── README.md # how to run your code
├── problem-2/
│   ├── results # screen capture
│   │   └── problem-2-question-1.png
│   ├── doc/
│   │   └── solution.md # explain your solution
│   ├── streaming_pipeline.py # your streaming pipeline code
│   └── README.md # how to run your code
└── (all original files)
```

Submission Method

1. Ensure all files are in the assignment/ folder
2. Compress the folder: [firstname]-[lastname]-lmwn-de.zip
3. Submit via the provided link/email

Example: john-smith-lmwn-de.zip

Questions?

If you have questions about the assignment requirements or encounter technical issues with the provided setup, please contact us.

Good luck!