Data Engineering - Module 00

-Nexus-

ETL, Streaming, Warehousing, and Orchestration

Batch pipelines, Kafka streams, Airflow DAGs, and Docker

Summary:
This document contains the exercises for the Data Engineering
It simulates a realistic Global E-Commerce Analytics Platform.

Version: 1.3

Contents

1 Introduction **2**
1.1 Architecture Overview 2

2 General rules **3**
2.1 Environment 3
2.2 Testing and Validation 3
2.3 Allowed/Forbidden 3

3 Exercise 00: The Foundation **4**

4 Exercise 01: The Lake **5**

5 Exercise 02: The Refinery **6**

6 Exercise 03: The Conductor **7**

7 Bonus part **8**
7.1 Submission and peer-evaluation 9
7.2 Peer-Evaluation Checklist 9

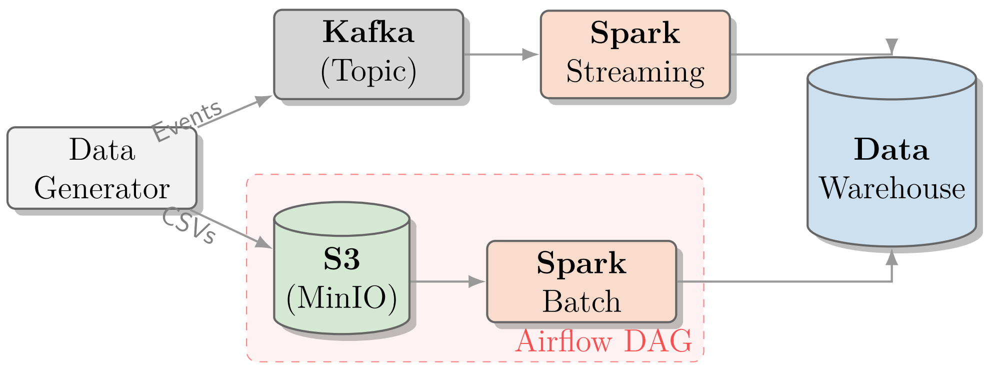
Chapter I
Introduction

Data Engineering is the backbone of modern analytics. It is not just about moving data; it is about building reliable, scalable, and maintainable systems.

The goal of this module is to introduce you to an **End-to-End Data Engineering Project**. You will build a simulated **Global E-Commerce Analytics Platform**.

Architecture Overview

Below is the high-level architecture you will implement.



We are aware that modern Data Stacks evolve quickly. So if you want to become a proficient Data Engineer, it's up to you to go further after the 42 Common Core!

Chapter II
General rules

Environment

- Your code must be containerized. Docker is your friend.
- All services (Airflow, Kafka, Spark) should be orchestratable via `docker-compose`.
- Compile/Run your code with the appropriate commands.

Testing and Validation

Quality is key. A pipeline that breaks silently is worse than no pipeline.

- You are encouraged to write unit tests for your PySpark transformations.
- Integration tests (checking if the S3 bucket exists before writing) are highly recommended.

Allowed/Forbidden

You are not coding in C anymore. Time to Python/Scala/SQL! Therefore:

- You are allowed to use almost everything from standard libraries.
- **Forbidden:** Hardcoding credentials (passwords, keys) in your source code. Use `.env` files. If you commit keys, your grade will be 0.

Memory leakage occurs in Java/Spark too (OOM errors). When you configure your executors, ensure you do not crash your container.

Chapter III

Exercise 00: The Foundation

Exercise : 00 Data Generation & Infrastructure
Turn-in directory : `ex00/`
Files to turn in : `docker-compose.yml, generator.py`
Forbidden functions : None

Just to make sure that everybody is awake, write a program that produces the raw data. Before analyzing data, we need data.

Your first task is to set up the infrastructure and generate synthetic data for the platform.

1. Batch Data Schemas (CSV/JSON) Generate the following datasets with appropriate data types:

- **Users.csv**: `user_id` (string), `signup_date` (timestamp), `country` (string).
- **Products.csv**: `product_id` (string), `category` (string), `price` (float).
- **Orders.csv**: `order_id` (string), `user_id` (string), `product_id` (string), `quantity` (int), `total_amount` (float).

2. Streaming Data (Kafka) The "Click Events" must be streamed in real-time.

- **Topic Name:** `clicks_topic`
- **Format:** JSON
- **Structure:**

"event$_id" : "evt_5501", "user_id" : "usr_99", "url" : "/product/shoes/1", "timestamp" : "2023 - 11 - 20T10 : 00 : 00", "action" : "view_item"$

Solve the exercises in a "Data Ops" manner. Everything should be reproducible with a single command.

Chapter IV

Exercise 01: The Lake

Exercise : 01 Ingestion & Storage
Turn-in directory : `ex01/`
Files to turn in : `ingestion.py, bucket_config.tf`
Forbidden functions : None

Welcome to the concept of the Data Lake. Raw data is messy, but it needs a home.
You have to implement two ingestion flows:

1. **Batch Ingestion:**
   - Ingest the generated static CSV files.
   - Upload them to a "Raw Layer" in Cloud Storage (Simulated S3 via MinIO).
   - *Naming convention:* `s3://raw-data/YYYY/MM/DD/orders.csv`
2. **Streaming Ingestion:**
   - Connect to the `clicks_topic`.
   - Ensure messages are being received and acknowledge them.

Network failures happen. If the S3 bucket is unreachable, your code should not crash silently. Define a retry mechanism.

Chapter V

Exercise 02: The Refinery

Exercise : 02 ETL & Warehousing
Turn-in directory : `ex02/`
Files to turn in : `etl_job.py, warehouse.sql`
Forbidden functions : None

Today is your first day at *GlobalBanksters United*. Your manager wants reports, and they want them fast.

Tasks:

- **PySpark Batch Pipeline:**
  - Read from the Raw Layer.
  - Clean data (remove nulls, cast types).
  - Perform joins (Orders ↔ Users ↔ Products).
  - Load into Warehouse Fact/Dim tables.
- **Streaming Pipeline:**
  - Read from `clicks_topic` using Spark Structured Streaming.
  - Aggregate "Clicks" by page in 1-minute tumbling windows.
  - Write the output to the `fact_page_views` table.

Don't forget to handle the "late data" problem in your streaming windows! Watermarking is your friend.

Chapter VI

Exercise 03: The Conductor

Exercise : 03 Orchestration with Airflow
Turn-in directory : `ex03/`
Files to turn in : `dags/, Dockerfile`
Forbidden functions : None

A script that runs manually is not a pipeline. It's a hobby. You need to automate your workflows.

Requirements:

- Install and configure **Apache Airflow**.
- Create a DAG named `ecommerce_daily_batch`.
- **Schedule:** Runs every day at 02:00 UTC.
- **Tasks:**
  - **Sensor:** Check if new files exist in S3.
  - **Ingest:** Trigger the ingestion script.
  - **Transform:** Trigger the PySpark job.
  - **Data Quality:** Run a simple SQL check (e.g., `SELECT COUNT(*) FROM orders` should be > 0).

You can use the `DockerOperator` or `SparkSubmitOperator` to keep your environment clean.

Chapter VII
Bonus part

Bonuses will be evaluated only if your mandatory part is PERFECT. "Perfect" means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonuses will not be evaluated at all.

You can implement one or more of the following features to add value to your platform:

1. **Data Quality Guardrails:** Implement `Great Expectations` in your Airflow DAG to validate data before it enters the warehouse (e.g., ensure prices are not negative).
2. **Advanced Storage (ACID):** Instead of standard Parquet, use **Delta Lake** or **Apache Iceberg** for your tables. Demonstrate Time Travel capabilities (querying an older version of the table).
3. **Visualization Dashboard:** Spin up a container with **Metabase** or **Apache Superset**. Connect it to your Warehouse and create a dashboard showing:
   - Real-time page views (Streaming).
   - Daily revenue trends (Batch).
4. **CI/CD Pipeline:** Set up a GitHub Actions workflow that automatically runs `pylint` and unit tests whenever you push to the repository.

Submission and peer-evaluation

Submit your assignment to your Git repository as usual. Only the work inside your repository will be evaluated during the defense.

Peer-Evaluation Checklist

During the defense, the evaluator will check the following "Success Metrics":

- ☐ **Infrastructure:** Does `docker-compose up` spin up Kafka, Airflow, Spark, and MinIO without errors?
- ☐ **Data Generation:** Does the Kafka topic `clicks_topic` receive JSON events?
- ☐ **Batch Pipeline:** Do the Parquet files appear in the Staging bucket?
- ☐ **Warehousing:** Can we query the `fact_orders` table via SQL?
- ☐ **Orchestration:** Does the Airflow DAG turn green (Success) when triggered?
- ☐ **Bonus:** Are there advanced features (Delta Lake, Dashboards, etc.)?

i The order in which the containers are started may differ depending on your docker-compose version.

```
????????????  XXXXXXXXXX = $3$$f15bc138aca1e76ec6f4cfd0797ec037
```

by Philanthropist **ayaarab**