Applied Big Data Engineering

Mini Project Assessment

Weighting: 25% of Total Module Grade
Type: Group Project (Recommended 2-3 members) or Individual
Duration: 2 Weeks

Project Objective

Students are required to architect and implement an end-to-end **Lambda or Kappa architecture** data pipeline. The system must simulate a real-world environment where data is ingested in real-time, processed for immediate insights, and orchestrated for historical analysis/reporting.

Preferred Technical Stack

To satisfy the Learning Outcomes, the solution must utilize:

- **Ingestion:** Apache Kafka (Producers, Topics, Partitions).
- **Stream Processing:** Apache Spark (Structured Streaming) OR Apache Storm.
- **Orchestration:** Apache Airflow (for managing batch jobs or reporting pipelines).
- **Storage/Sink:** A suitable database (PostgreSQL, Cassandra) or File System (HDFS/S3 bucket with Parquet).

Note: You need to justify the tech stack selection against the selected scenario.

Deliverables

1. **Architecture Diagram:** A visual representation of the pipeline (Source $\rightarrow$ Kafka $\rightarrow$ Processing $\rightarrow$ Storage $\rightarrow$ Airflow/Report).
2. **Source Code:**
    - Python producers (mock data generators).
    - Spark/Storm processing scripts.
    - Airflow DAG files.
    - Docker Compose file (if used to spin up the stack).
3. **The Analyzed Report:** The final output generated by your pipeline (PDF, CSV, or Dashboard screenshot) based on the logic defined in the scenario.
4. **Project Report (1500 words)-** brief and organized explanations preferred**:**
    - Justification of the tools selected.
    - Explanation of how **Event Time vs. Processing Time** was handled.
    - **Ethics Section:** Discuss the privacy implications of the chosen scenario (e.g., Surveillance in Smart City, User profiling in E-commerce) and how data governance should be applied.

Select One Scenario

Students must choose **one** of the following three scenarios. Each requires the generation of synthetic data (producers), real-time processing, and a scheduled analytical report.

Note: Use simple generator scripts to generate random generation of data. UI implementations aren't necessary. Generator logs should be clean and self explainable.

Scenario 1: The "Smart City" Traffic & Congestion System

Domain: IoT & Logistics

Context: The city of Colombo wants to modernize its traffic management. Sensors at major intersections broadcast vehicle counts and speeds every second.

- **Data Source (Producer):** Python script simulating sensors from 4 different junctions, sending JSON data: {sensor_id, timestamp, vehicle_count, avg_speed}.
- **Real-Time Requirement (Stream Layer):**
  - Ingest data via **Kafka**.
  - Use **Spark/Storm** with windowing (e.g., 5-minute tumbling windows) to calculate the "Congestion Index."
  - **Trigger:** If avg_speed drops below 10km/h, write an alert to a "Critical-Traffic" topic or database table immediately. (Note: generator script should be controlled i.e. the critical traffic mimicking data generated occasionally)
- **Orchestration Requirement (Batch/Airflow):**
  - Use **Airflow** to schedule a nightly job.
  - The job aggregates the day's data to find the "Peak Traffic Hour" for each junction.
  - Generate a report indicating which junction requires physical traffic police intervention the next day.
- **Analytic Report:** A visualization or table showing Traffic Volume vs. Time of Day.

Scenario 2: FinTech Fraud Detection Pipeline

Domain: Banking & Security

Context: A digital wallet provider needs to detect fraudulent transactions in real-time while maintaining a daily ledger of transaction volumes.

- **Data Source (Producer):** Simulator generating credit card transactions: {user_id, timestamp, merchant_category, amount, location}. Inject occasional high-value transactions from foreign locations.
- **Real-Time Requirement (Stream Layer):**
  - Ingest transaction stream via **Kafka**.

    - Use **Spark/Storm** to filter transactions.
    - **Trigger:** If a user makes transactions from two different countries within 10 minutes (impossible travel) OR spends > $5000, flag as FRAUD and push to a secure sink immediately. (Note: generator script should be controlled i.e. the fraud activities generated occasionally)
- **Orchestration Requirement (Batch/Airflow):**
    - Use **Airflow** to trigger an ETL process every 6 hours.
    - The process moves validated (non-fraud) data into a Data Warehouse (e.g., Parquet files) and calculates the total volume processed.
    - Generate a "Reconciliation Report" comparing Total Ingress Amount vs. Validated Amount.
- **Analytic Report:** Analysis of "Fraud Attempts by Merchant Category."

Scenario 3: E-Commerce Clickstream & Inventory Watch

Domain: Retail & Marketing

Context: An online electronics store wants to track user activity to adjust prices dynamically and monitor stock interest.

- **Data Source (Producer):** Simulator generating user events: {user_id, product_id, event_type (view/add_to_cart/purchase), timestamp}.
- **Real-Time Requirement (Stream Layer):**
    - Ingest clickstream data via **Kafka**.
    - Use **Spark/Storm** to aggregate "Views" per product in sliding windows (e.g., last 10 mins).
    - **Trigger:** If a product receives > 100 views but < 5 purchases (High Interest, Low Conversion), output a notification suggesting a "Flash Sale" or discount.
- **Orchestration Requirement (Batch/Airflow):**
    - Use **Airflow** to orchestrate a "Daily User Segmentation" pipeline.
    - Process the raw logs to categorize users into Window Shoppers vs Buyers.
    - Generate a formatted email/text file summary of the top 5 most viewed products of the day.
- **Analytic Report:** A summary of Conversion Rates (Purchases / Views) per product category.