Technical Challenge for Data Engineer Candidate

Objective

Showcase your ability to process data, handle fault tolerance, implement data-driven testing, and troubleshoot performance issues in a distributed data processing application using Scala and Apache Spark.

Challenge

Part 1: Data Processing Task

Input Data

Link to .csv with columns:

- user_id (Integer)
- user_name (String)
- email (String)
- purchase_datetime (String/Date)
- purchase_amount (Float)

Instructions

1. Write a Scala application using Apache Spark to read the CSV file.
2. Process the data with the following steps:
    a. Clean the dataset by removing any records with null values in critical columns (user_id, email, purchase_datetime).
    b. Convert the purchase_datetime column to a proper timestamp format. c. Filter the records to include only those within a specific date range (e.g., between '2023-01-01' and '2023-12-31').
    d. Aggregate the data to compute the total purchase_amount per user_id.
3. Save the processed data as Parquet files to a Cloud Storage location of your preference (e.g., AWS S3, Google Cloud Storage).

Deliverable

Scala code in an accessible GitHub repository implementing the above logic.

Part 2: Data-Driven Testing

Instructions

Create unit tests for the data processing logic using a Scala testing framework (e.g., ScalaTest or MUnit). Your tests should cover:

- Valid inputs: verifying that the outputs match expected results for correctly formatted and complete CSV records.
- Invalid inputs: verifying that records with null values or incorrect date formats are handled appropriately (e.g., are removed or trigger exceptions).
- Boundary cases: such as dealing with an empty CSV file and ensuring the application handles it gracefully.

Deliverable

Scala test cases in the same GitHub repository as part 1 covering the processing logic described above.

Part 3: Fault Tolerance

Instructions

Describe resiliency patterns you would implement to ensure fault tolerance in the distributed Spark job. Consider aspects such as:

1. Handling node failures during processing.
2. Strategies for retrying failed tasks automatically.
3. The use of checkpointing mechanisms to save processing state and recover from interruptions.
4. Any additional strategies to minimize downtime in the event of transient issues.

Deliverable

A brief report (1-2 pages) explaining your fault tolerance strategies for the Spark application.

Part 4: Performance Troubleshooting

Instructions

Present a scenario where the Spark job takes significantly longer to process than expected. 1. Provide a list of at least 5 performance troubleshooting questions you would ask the team along with an explanation for each.
2. Describe actions you would take to diagnose performance bottlenecks

Deliverable

A document listing performance troubleshooting questions, their rationale, and proposed diagnostic actions.

Evaluation Criteria

Coding Skills

Clarity, efficiency, and structure of the Scala code. The code must correctly implement the CSV to Parquet transformation, taking into account cleansing, formatting, filtering, and aggregation.

Testing Ability

Comprehensive coverage and effectiveness of the data-driven tests written using Scala’s testing frameworks.

Fault Tolerance Understanding

Depth of knowledge regarding resiliency patterns for distributed systems and a clear explanation of strategies to handle failures in Spark.

Performance Consideration

Insightfulness and thoroughness in the performance troubleshooting approach, showcasing the candidate's ability to identify and resolve issues in distributed data processing.