REVIEW DOCUMENT

AWS Automated Data Pipeline

Preparation Date: 05-03-24
Prepared By: Ashutosh Kumar Maurya, Brhat B G

Epsilon

Introduction

The AWS Automated Data Pipeline project undertakes a detailed exploration of automating data pipelines using AWS EMR and Glue services. Through iterative development, it assesses methods ranging from hardcoded schemas to dynamic schema discovery and processing, aiming to refine data handling within the AWS ecosystem. This concise overview sets the stage for a deep dive into the project's iterations, their strengths, and areas for future improvement, focusing on enhancing adaptability, efficiency, and scalability of data pipelines.

Design and Code Review Comments

AWS EMR Solution

Iteration 1: Hardcoded Schema

(File: emr_solution_1.txt and emr_solution_1_lambda_function.txt)

- **Schema Determination:** Static, utilizes a predefined schema specified within the code.
- **Description:** This script employs PySpark to process CSV files by defining a fixed schema upfront. It reads two datasets with the established schema and merges them, removes duplicates, and then orders by a specific column (Id). The merged and clean dataset is then saved to a designated S3 bucket.
- **Strengths:** Predictable and stable processing when the schema is known and unchanging. It ensures that the data conforms to the expected format and types without the need for schema inference during runtime.
- **Improvements:** This method lacks flexibility as any deviations in the incoming data schema or new columns would require code changes. Introducing schema discovery or using Spark's schema inference capabilities could allow the code to handle varying schemas. Error handling can be added to manage inconsistencies in data types or missing fields in the input datasets.

Iteration 2: Dynamic Bucket Handling with Unique Column Identification

(File: emr_solution_2.txt and emr_solution_2and3_lambda_function.txt)

- **Schema Determination:** Dynamic, leverages runtime analysis to deduplicate and order data.

- **Description:** This PySpark script dynamically retrieves file names from specified S3 buckets. It reads data with headers from one source, infers schema, and applies it to another source without headers. The script then merges the data, utilizes a temporary unique ID for ordering, identifies a unique column for deduplication (if present), and orders the data based on the unique column found.
- **Strengths:** Adapts to varying file names and schemas by using the schema of the source file with headers. It handles unique column detection for deduplication and ordering, enhancing data integrity.
- **Improvements:** Considerations for optimizing the deduplication process and handling large datasets where monotonically_increasing_id may lead to performance issues due to its nature of creating non-consecutive, sparse ID numbers that can potentially cause large shuffles in distributed systems.

Iteration 3: Dynamic Schema Inference with Conditional Header Processing

(File: emr_solution_3.txt and emr_solution_2and3_lambda_function.txt)

- **Schema Determination:** Dynamic, uses schema inference and conditional header detection.
- **Description:** This script first lists files in two specified S3 buckets. It then reads the first file, inferring the schema and determining if a header is present. It reads the second file, assuming it has a header. After that, it merges the two datasets, removes duplicates, and orders them by the first column identified in the second dataset. The final dataframe is then saved to an output location with the header included.
- **Strengths:** More adaptable compared to the hardcoded schema approach, as it dynamically infers the schema of the CSV files. The header check adds flexibility to process files with or without headers correctly.
- **Improvements:** This approach relies on correct schema inference, which can be problematic with ambiguous data types or complex data structures. It also assumes that the first row of the first file is the header if it meets certain conditions, which might not always be accurate. Additionally, it could be enhanced by better error handling and validation to ensure that merging is correctly performed, especially when dealing with missing columns.

AWS Glue Solution

Iteration 1: Hardcoded Schema in AWS Glue Jobs

(File: group.txt and outlier.txt)

- **Schema Determination:** Static, with a predefined schema explicitly defined within the AWS Glue job script.
- **Description:** This iteration uses an AWS Glue job to process data, where the schema is hardcoded within the script itself. The job reads data from S3, applies the hardcoded schema to understand the structure of the data, and then performs various transformations, including merging datasets, deduplication, and sorting. The results are then saved back to an S3 bucket. This method ensures data processing aligns with expected formats and types without requiring runtime schema inference.
- **Strengths:** Provides a stable and predictable environment for data processing, ideal for datasets with a consistent schema. This approach simplifies the development process when dealing with well-known data structures.
- **Improvements:** To enhance flexibility and reduce maintenance overhead, transitioning to dynamic schema discovery could be beneficial. Integrating error handling mechanisms can also improve the job's resilience against data anomalies and schema deviations.

Iteration 2: Dynamic Schema Discovery with AWS Glue Crawlers and Data Catalog

(File: clean.txt , normalize.txt and merge.txt)

- **Schema Determination:** Dynamic, leveraging AWS Glue Crawlers for schema discovery, with the schema stored and managed in the AWS Glue Data Catalog.
- **Description:** This advanced iteration introduces AWS Glue Crawlers to automatically discover and identify the schema of data stored in S3. The crawler scans the data source, detects the schema, and registers this schema in the AWS Glue Data Catalog. When executing AWS Glue jobs, the script dynamically retrieves the schema from the Data Catalog, ensuring data processing scripts are automatically aligned with the current data structure. This approach supports efficient processing of evolving data sources, handling schema changes seamlessly without manual script updates.

- **Strengths:** Significantly improves the adaptability and efficiency of data processing by automating schema discovery and leveraging centralized schema management. This method reduces manual coding and schema maintenance efforts, facilitating the processing of diverse and changing data sources.
- **Improvements:** Managing the lifecycle of schemas in the Data Catalog and optimizing crawler configurations to accurately reflect data changes can further refine this approach. It's also beneficial to implement strategies for version control and access management within the Data Catalog to maintain data governance and quality.

Conclusion

The project's iterative approach highlights a commitment to enhancing data processing workflows' adaptability, efficiency, and reliability. By transitioning from static, predefined schemas to dynamic schema inference and leveraging AWS Glue for schema discovery, the project has made significant strides in automating and optimizing data pipeline processes. The strengths of each iteration, from predictable and stable processing to adaptability in schema management, are commendable. However, the identified areas for improvement, such as the need for enhanced error handling, schema management, and performance optimization, provide a roadmap for future enhancements.