NSP Bolt Ride – Real-Time Trip Processing Project

Project Overview

This project is a data pipeline designed to process real time trip data, store it in a DynamoDB table, and compute daily Key Performance Indicators (KPIs) using AWS services. The pipeline ingests trip start and trip end events from CSV files, streams them through Kinesis Data Streams, processes them in real-time using a Lambda function, and aggregates the data at the end of each day using an AWS Glue job.

Objectives

- Ingest taxi trip data (trip start and trip end events) from CSV files.
- Stream the data through Kinesis for real-time processing.
- Process and correlate trip start and end events using a Lambda function, storing the results in DynamoDB.
- Compute daily KPIs (e.g., total trips, total fare, average trip distance) using a scheduled AWS Glue job.

Pipeline Flow

The pipeline consists of the following components and flow:

1. **Data Ingestion**:
    - Two CSV files (trip_start.csv and trip_end.csv) in the data/ directory contain taxi trip data.
    - The send_to_kinesis.py script reads the CSV files and sends them to two Kinesis Data Streams: TripStartStream for trip start events and TripEndStream for trip end events.
    - A 5-minute delay is introduced between sending trip start and trip end records to simulate the time gap between the events.
2. **Kinesis Data Streams**:
    - Two Kinesis streams are used:
        - TripStartStream: Receives trip start events.
        - TripEndStream: Receives trip end events.
    - Each stream has 1 shard, and the data is sent as raw JSON (not base64-encoded).
    - The Kinesis triggers for the Lambda function are configured with:

        - Batch size: 100 records.
        - Batch window: 100 seconds.
        - Starting position: LATEST.

3. **Lambda Processing**:
    - The TripProcessor Lambda function (trip_processor.py) is triggered by both Kinesis streams.
    - It processes each record as follows:
        - For TripStartStream events:
            - Extracts fields like trip_id, pickup_location_id, dropoff_location_id, vendor_id, pickup_datetime, estimated_dropoff_datetime, estimated_fare_amount and date from pickup_datetime.
            - Stores the data in the TripData DynamoDB table with status: "Started".
        - For TripEndStream events:
            - Looks up the corresponding trip start record in DynamoDB using trip_id.
            - If found, updates the record with dropoff_datetime, rate_code, passenger_count, trip_distance, fare_amount, tip_amount, payment_type, trip_type, and sets status: "Completed".
            - If not found, logs a warning and skips the record.
    - The Lambda function uses logging for debugging and converts float values to decimal.Decimal using TypeSerializer to ensure compatibility with DynamoDB.
4. **DynamoDB Storage**:
    - The TripData DynamoDB table stores the processed trip data.
    - The table uses date (Extracted from pickup_datetime) as the partition key
    - The table uses trip_id (String) as the sort key.
    - Each record contains fields like pickup_datetime, estimated_fare_amount, dropoff_datetime, fare_amount, and status.
5. **Daily KPI Aggregation**:

- An AWS Glue job (daily_kpi_aggregation.py) is scheduled to run at the end of each day.
- The Glue job reads the completed trip data from the TripData DynamoDB table.
- It computes daily KPIs (e.g., total trips, total fare, average trip distance) and stores the results in a designated location (e.g., an S3 bucket).

Setup Instructions

Follow these steps to set up and run the project:

Prerequisites

- An AWS account with permissions to create and manage Kinesis Data Streams, Lambda functions, DynamoDB tables, Glue jobs, and CloudWatch Logs.
- Python 3.9 or later installed on your local machine.
- AWS CLI configured with your credentials.
- The boto3, pandas, and python-dotenv Python packages installed:
- pip install -r requirements.txt

Step 1: Set Up AWS Resources

1. **Create Kinesis Data Streams**:
    - Go to **AWS Console > Kinesis > Data Streams**.
    - Create two streams:
        - Name: TripStartStream, Shards: 4.
        - Name: TripEndStream, Shards: 4.
    - Wait for both streams to become "Active".
2. **Create the DynamoDB Table**:
    - Go to **AWS Console > DynamoDB > Tables**.
    - Create a table:
        - Name: TripData.
        - Partition key: date(String).
        - Sort key: trip_id (String).

        - Use default settings (on-demand capacity).
    - Click **Create table**.
3. **Create the Lambda Function**:
    - Go to **AWS Console > Lambda > Functions**.
    - Create a function:
        - Name: TripProcessor.
        - Runtime: Python 3.9.
        - Copy the code from src/lambda_functions/trip_processor.py.
        - Click **Deploy**.
    - Add triggers:
        - Add a Kinesis trigger for TripStartStream:
            - Batch size: 100.
            - Batch window: 100 seconds.
            - Starting position: LATEST.
        - Add a Kinesis trigger for TripEndStream:
            - Batch size: 100.
            - Batch window: 100 seconds.
            - Starting position: LATEST.
    - Set the Lambda execution role to have permissions for:
        - Kinesis: kinesis:DescribeStream, kinesis:GetShardIterator, kinesis:GetRecords.
        - DynamoDB: dynamodb:PutItem, dynamodb:GetItem.
        - CloudWatch Logs: logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents.
4. **Set Up the AWS Glue Job**:
    - Go to **AWS Console > Glue > ETL Jobs**.
    - Create a job:
        - Name: DailyKPIAggregation.

        - Upload the script from src/glue_scripts/daily_kpi_aggregation.py.
        - Configure the job to read from the TripData DynamoDB table and write to an S3 bucket (as per your implementation).
        - Schedule the job to run daily at the end of the day (e.g., 00:00 UTC).
        - Add s3 bucket name and DynamoDB table name to the **JOB PARAMETERS**.
    - Ensure the Glue job role has permissions for:
        - DynamoDB: dynamodb:Scan.
        - S3: s3:PutObject (for the output bucket).

Step 2: Prepare the Local Environment

1. **Set Up Environment Variables**:
    - Create a .env file in the Project 7 directory:
    - AWS_REGION=us-east-1
    - Ensure your AWS credentials are configured in ~/.aws/credentials or set as environment variables.
2. **Verify Data Files**:
    - Ensure the data/trip_start.csv and data/trip_end.csv files exist and contain valid data with the expected columns (e.g., trip_id, pickup_datetime, estimated_fare_amount, etc.).

Step 3: Run the Pipeline

1. **Send Data to Kinesis**:
    - Navigate to the src/ directory.
    - Run the send_to_kinesis.py script:
    - python send_to_kinesis.py
    - This will send trip start records and trip end records to the respective Kinesis streams with a 5-minute delay between them.
2. **Monitor Lambda Processing**:
    - The TripProcessor Lambda function will be triggered automatically by the Kinesis streams.

- Check CloudWatch Logs for the Lambda function:
    - Go to **AWS Console > CloudWatch > Logs > Log groups > /aws/lambda/TripProcessor**.
    - Look for INFO messages like Trip {trip_id} completed: {...}.

3. **Verify DynamoDB Records**:
    - Go to **AWS Console > DynamoDB > Tables > TripData**.
    - Confirm that all records are stored with status: "Completed".
4. **Monitor the Glue Job**:
    - The DailyKPIAggregation Glue job will run at the scheduled time (end of day).
    - Check the Glue job logs in **AWS Console > Glue > ETL Jobs > Runs** to confirm it processed the data and wrote the KPIs to the specified S3 bucket.

Troubleshooting

Here are common issues and how to resolve them:

1. send_to_kinesis.py Fails to Run

- **Symptom**: FileNotFoundError when reading trip_start.csv or trip_end.csv.
- **Cause**: The script uses relative paths (../data/trip_start.csv), and the working directory is incorrect.
- **Fix**:
    - Ensure you're running the script from the src/ directory:
    - cd src
    - python send_to_kinesis.py
    - Alternatively, update the script to use absolute paths (e.g., C:/path/to/trip_start.csv).
- **Symptom**: ClientError: An error occurred (AccessDeniedException) when calling the PutRecord operation.
- **Cause**: The AWS credentials lack permissions to write to Kinesis.
- **Fix**:
    - Verify that your AWS CLI credentials have permissions for kinesis:PutRecord.
    - Update the IAM role/user policy to include:

- {
- "Effect": "Allow",
- "Action": "kinesis:PutRecord",
- "Resource": "arn:aws:kinesis:us-east-1:ACCOUNT_ID:stream/Trip*"
- }

2. Lambda Function Fails to Process Records

- **Symptom**: CloudWatch logs show Error processing record: JSONDecodeError.
- **Cause**: The Kinesis record data is not in the expected format.
- **Fix**:
  - Verify the data sent to Kinesis:
  - aws kinesis get-shard-iterator --stream-name TripStartStream --shard-id shardId-000000000000 --shard-iterator-type TRIM_HORIZON
  - aws kinesis get-records --shard-iterator <ShardIterator>
  - Decode the Data field (base64-encoded) to confirm it's valid JSON:
  - import base64
  - encoded = "<Data field>"
  - decoded_bytes = base64.b64decode(encoded)
  - print(decoded_bytes.decode('utf-8'))
  - Ensure the send_to_kinesis.py script sends properly formatted JSON.
- **Symptom**: CloudWatch logs show Error processing record: Float types are not supported.
- **Cause**: The Lambda function tries to write float values to DynamoDB.
- **Fix**:
  - The current script already uses TypeSerializer to handle this. If the error persists, ensure the Lambda function code matches the provided version.
- **Symptom**: CloudWatch logs show Trip start not found for trip_id: {trip_id}.
- **Cause**: A trip end event arrived before the corresponding trip start event was processed.

- **Fix**:
    - Ensure the send_to_kinesis.py script sends trip start records before trip end records (it currently includes a 5-minute delay, which should prevent this).
    - Check if the Kinesis triggers are correctly configured with LATEST to process current record streams

3. DynamoDB Records Are Missing or Incomplete

- **Symptom**: Records not found in TripData, or status is stuck at Started.
- **Cause**: The Lambda function failed to process some records, or trip end events were skipped.
- **Fix**:
    - Check CloudWatch logs for errors or warnings.
    - Verify that both Kinesis streams (TripStartStream and TripEndStream) contain the expected records (use AWS CLI as above).
    - Ensure the Lambda function has permissions to write to DynamoDB (dynamodb:PutItem, dynamodb:GetItem).

4. Glue Job Fails to Run

- **Symptom**: Glue job fails with AccessDeniedException.
- **Cause**: The Glue job role lacks permissions to read from DynamoDB or write to S3.
- **Fix**:
    - Update the Glue job role to include:
    - {
    - "Effect": "Allow",
    - "Action": [
    - "dynamodb:Scan",
    - "s3:PutObject"
    -],
    - "Resource": [
    - "arn:aws:dynamodb:us-east-1:ACCOUNT_ID:table/TripData",
    - "arn:aws:s3:::your-output-bucket/*"

    -]
    - }
- **Symptom**: Glue job runs but produces no output.
- **Cause**: The TripData table is empty, or the job logic is incorrect.
- **Fix**:
    - Verify that the TripData table contains data.
    - Check the Glue job logs for errors and review the daily_kpi_aggregation.py script logic.

5. General Tips

- **CloudWatch Logs**: Always check CloudWatch Logs for both the Lambda function (/aws/lambda/TripProcessor) and the Glue job to identify errors.
- **IAM Permissions**: Ensure all roles (Lambda, Glue) have the necessary permissions for Kinesis, DynamoDB, S3, and CloudWatch Logs.
- **Data Validation**: Validate the CSV files to ensure they contain the expected columns and no missing or malformed data.

Conclusion

This project successfully demonstrates a real-time data pipeline for processing taxi trip data using AWS services. The pipeline ingests data via Kinesis, processes it with Lambda, stores it in DynamoDB, and aggregates daily KPIs using Glue. Follow the setup instructions and troubleshooting steps to ensure smooth operation.