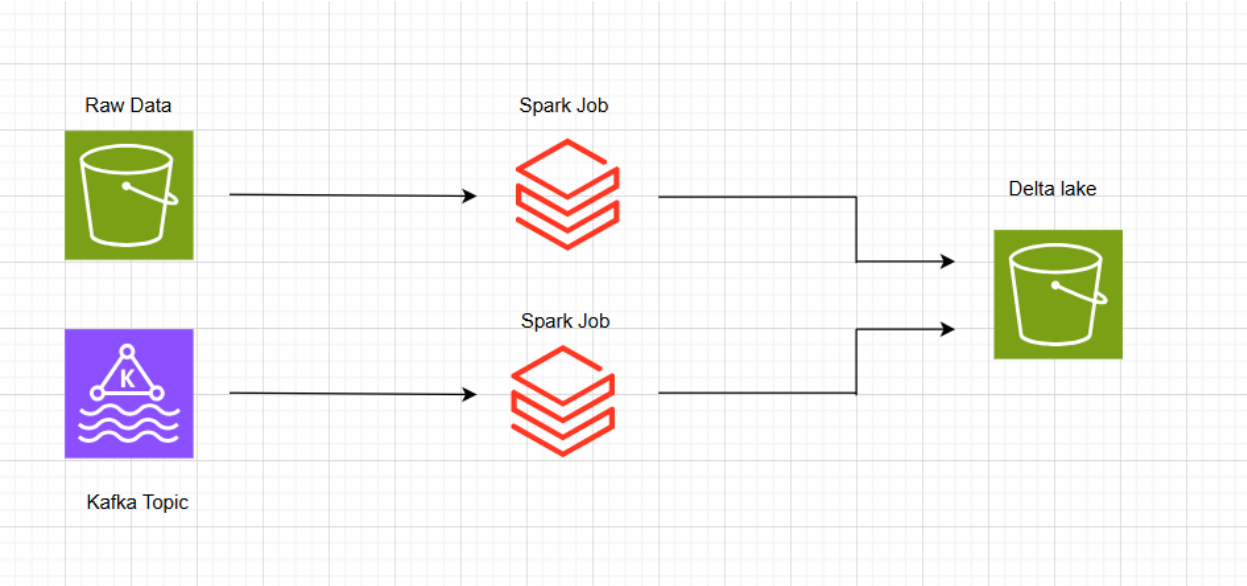
Architecture Diagram:



NYC TLC Data Pipeline Architecture Design Choices

Handling Late-Arriving Taxi Trip Data

Late-arriving data is a common challenge in trip-based datasets. To ensure data integrity and consistency, we implement the following strategies:

1. **Event Time Processing:**
    - Use timestamps from the trip data instead of ingestion time to determine when a trip occurred.
    - Configure Spark Structured Streaming with watermarking to handle delayed events efficiently.
2. **Change Data Capture (CDC) & Upserts:**
    - Utilize **Delta Lake's MERGE INTO** functionality to update existing records when late data arrives.
    - Define primary keys (trip_id, pickup_datetime) to identify and update duplicate records.

3. **Soft Deletes & Reprocessing:**
    - Maintain a column (is_active) to flag late-arriving records.
    - Schedule periodic reprocessing jobs to reconcile late data with historical records.

Storage Partitioning Strategy

Efficient partitioning is key for fast query performance and cost optimization. The following strategy is used:

1. **Partition by Date:**
    - Partition data by pickup_date (derived from pickup_datetime) to support time-based queries.
2. **Secondary Partitioning (Bucketing):**
    - Bucket data by vendor_id or pickup_location_id to optimize joins.
    - This reduces shuffle operations in Spark and improves query execution time.
3. **Compact Small Files:**
    - Use **Auto Optimize and Z-Ordering (Databricks Delta Lake)** to coalesce small files and improve scan efficiency.

Performance Optimization for Common Queries

To ensure low-latency query execution, the following optimizations are applied:

1. **Indexing & Caching:**
    - Utilize **Bloom Filters** on high-cardinality columns like trip_id.
    - Cache frequently accessed tables (e.g., reference tables like fare rates, driver details).
2. **Materialized Views:**
    - Precompute aggregations (e.g., daily total trips per zone) using materialized views in Redshift/Snowflake.
    - Store aggregated results in separate tables for fast dashboard queries.
3. **Query Pruning:**

Classification: Public

- Implement **dynamic partition pruning** to filter out unnecessary partitions during query execution.
- Optimize joins using **broadcast joins** where applicable.

Data Retention Policies

To manage storage costs and comply with regulations, we enforce data retention policies:

1. **Historical Data Retention:**
   - Keep raw trip data for **5 years** in cold storage (S3 Glacier, Azure Archive Storage).
   - Maintain processed trip data for **2 years** in the data warehouse.
2. **Real-Time Data Retention:**
   - Store streaming events for **7 days** in a Kafka topic before persisting in Delta Lake.
3. **Archival & Purging:**
   - Run scheduled jobs to delete or archive old data based on retention rules.
   - Use **Time Travel (Delta Lake)** to maintain rollback capabilities for a limited period (30 days).

Dimensional Model Design

To support various analytics needs, the following dimensional model is designed:

Fact Table: Trip Fact

- trip_id (Primary Key)
- driver_id (FK to Driver Dimension)
- pickup_location_id (FK to Location Dimension)
- dropoff_location_id (FK to Location Dimension)
- rate_code_id (FK to Rate Code Dimension)
- pickup_datetime
- dropoff_datetime
- fare_amount

Classification: Public

- total_amount
- trip_duration
- trip_distance

Dimension Tables:

Date Dimension

- date_key (Primary Key)
- date
- hour_of_day
- day_of_week
- month
- year

Location Dimension (Slowly Changing Dimension Type 2)

- location_id (Primary Key)
- borough
- zone
- latitude
- longitude
- effective_date
- expiration_date
- is_current

Driver Dimension

- driver_id (Primary Key)
- driver_name
- license_number
- rating
- total_trips

Classification: Public

- total_revenue

Rate Code Dimension (Slowly Changing Dimension Type 2)

- rate_code_id (Primary Key)
- rate_code_description
- effective_date
- expiration_date
- is_current

Aggregation Tables for Common Queries

To improve query performance, the following aggregation tables are introduced:

Aggregated Revenue Table

- date_key
- total_revenue
- total_trips
- avg_fare_per_trip

Geographic Pickup/Dropoff Trends

- location_id
- date_key
- total_pickups
- total_dropoffs

Driver Performance Metrics

- driver_id
- date_key
- total_trips
- avg_rating
- total_revenue

Partitioning Strategy

To optimize query performance and cost, we apply the following partitioning strategy:

1. **Fact Table Partitioning:**
    - Partitioned by pickup_date (derived from pickup_datetime).
    - Enables fast retrieval of time-based queries.
2. **Dimension Table Partitioning:**
    - Slowly changing dimensions (Location, Rate Code) use partitioning by effective_date.
    - Ensures efficient filtering of historical records.
3. **Aggregation Table Partitioning:**
    - Partitioned by date_key for fast retrieval of aggregated data.

Classification: Public