Resvevents – Track and Monitor Contract Journey in the Truck Industry

Sameer Dongare

Data Engineer, U-Haul International Inc.

Abstract— **In the truck industry, tracking a contract's journey from reservation to receipt and monitoring key fields for changes during this process.**

Introduction

In many countries, especially in the West, self-moves involving truck rentals and moving supplies are common. People prefer this approach due to its cost-effectiveness, flexibility, and the convenience of managing the move on their terms. However, in India and other South Asian countries, the scenario is quite different due to the availability of cheap labor costs, challenges in urban infrastructure, complicated vehicle rental logistics, etc. This paper talks about a process that involves self-moves.

To secure equipment – Trucks, Trailers, or Tows- for their relocation, individuals utilize the moving company's website, app, or customer service to make online reservations. Alternatively, they may visit the company's stores to acquire equipment or purchase relevant items to facilitate their move. The reservation is made by choosing the type of equipment. Along with that, the customers may need some moving supplies like boxes, tapes, etc. Sometimes, they must rent utility/applicable dollies to help with their move. Subsequently, situations may arise where there is a necessity to change an existing reservation, which could involve adding additional items, swapping an already booked truck or trailer for a different model, obtaining upgraded insurance coverage, or modifying the pickup location. Many reasons could lead to the need for these adjustments, such as changes in plans, evolving requirements, or unforeseen circumstances.

Fundamentals

Here are some fundamental terms related to the truck rental industry and the technology mentioned in this article.

Q. What is a Contract?

Ans: When a reservation is made for equipment like a truck, trailer, or tow, a unique contract is created.

Q. What are Contract Events?

Ans: Contract Events refer to any changes made to the Contract during the rental period. These could include a reservation update, equipment pickup, modifications during the rental period, or equipment return. Each of these updates constitutes a Contract Event.

Q. What is the source of Contract Events?

Ans: Contract events are written in real-time and sent to event streaming platforms such as Apache Kafka, Event Hub, etc. This case study utilizes Kafka. You can find more information on this in the last section.

Q. Where are the processed events written to?

Ans: The processed events could be written to any persistent data storage, such as a Databricks Delta table. The last section provides more information on this.

Problem Statement

The Kafka topic serves as a platform for the real-time publication of contract transactions. We could call it events that happen to a reservation or a contract – hence the name Resvevents (Reservation Events!). The Data Team wants to analyze the progression of a contract, from the reservation initiation to the customer's equipment pick-up and, finally, the return of the equipment. Among the many fields in the Kafka topic, the team aims to scrutinize a few critical fields of interest. The team seeks to understand the frequency of updates to these fields and explore the underlying reasons for such modifications.

Objective

The team's objective is to capture alterations in the values of the critical fields mentioned earlier for each contract, tracking the changes from the reservation's inception to the equipment's return. This analysis is expected to provide insights into the factors influencing these changes. Furthermore, the team intends to capture these modifications in real time to establish a communication channel informing others of such occurrences.

Here's a table showing a few critical fields and the change in their values:

Field Name	**Original Value**	**Changed Value**
Pickup_Date	2024-08-01	2024-08-02
Pickup_City_State	Phoenix_AZ	Scottsdale_AZ
…		
…		
Truck_Model	10′	15′
Trailer_Model	Bike Trailer	Motorcycle Trailer

Solution Approach

In this case study, it has been determined that creating a real-time streaming consumer (Spark Structured Streaming Consumer) is essential to reading the Kafka topic effectively. This consumer is responsible for identifying any changes made to the vital fields within the micro-batch and marking these changes in a column of the Boolean array within the final Databricks Delta table. The choice to utilize an array of Booleans will be further discussed in later sections of this document.

This task is quite complex because the consumer processes around a million contract transactions within a single micro-batch before persisting them to a Databricks delta table. It is crucial that, before saving, the values in these critical fields are compared against those in previously stored records for each unique contract.

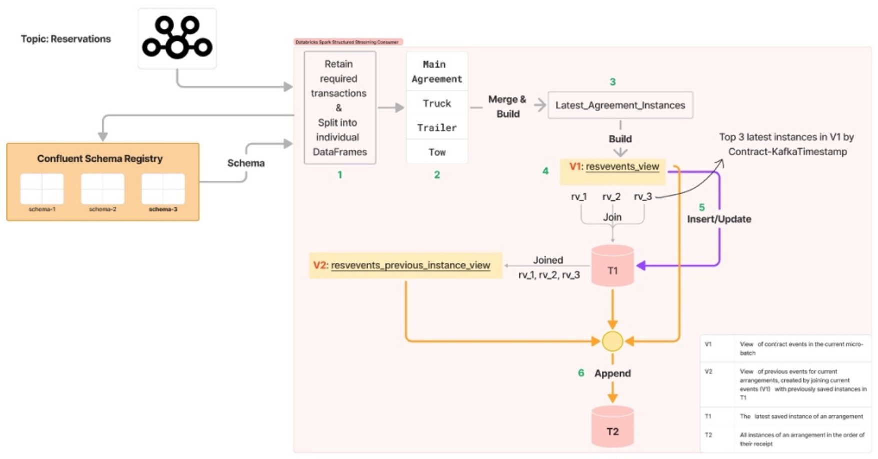
Main Comparison Logic

This solution utilizes a streamlined one-step process instead of the traditional read, transform, and save method. In a conventional process, the data is read, transformations are applied, and the transformed data is saved in a table. As part of this approach, we establish a procedure for storing the most recent contract instance in a separate table for future reference. When a new contract instance is received, we compare it with the previously saved contract in a separate table. We then identify the key fields with modified values and incorporate them into the main table. This innovative approach entails merging old values from the individual table, new values from the current feed, and a flagged comparison to create a new record, which is then added to the main table. This method significantly enhances the accuracy and real-time capabilities of our data analysis.

Architectural Flow

Figure 1 presents the technical architectural flow for generating contract events, referred to as Resvevents. This is achieved by reading the data from a Kafka topic into the consumer and storing variations in the target columns.

Fig. 1. Architectural Flow


Topic: Reservations
Databricks Spark Structured Streaming Consumer
Confluent Schema Registry
schema-1
schema-2
schema-3
Schema
Retain required transactions & Split into individual DataFrames
1
Main Agreement
Truck
Trailer
Tow
2
Merge & Build
3
Latest_Agreement_Instances
Build
4
V1: resvevents_view
Top 3 latest instances in V1 by Contract-KafkaTimestamp
rv_1
rv_2
rv_3
Join
5
Insert/Update
V2: resvevents_previous_instance_view
Joined rv_1, rv_2, rv_3
T1
6
Append
T2

V1	View of contract events in the current micro-batch
V2	View of previous events for current arrangements, created by joining current events (V1) with previously saved instances in T1
T1	The latest saved instance of an arrangement
T2	All instances of an arrangement in the order of their receipt

Legend

Id	Table/View Name	Description
T1	resvevents_latest	The latest saved instance of a contract
T2	resvevents_all	All instances of a contract in the order of their receipt
V1	resvevents_view	View of contract events in the current micro-batch
rv_v1	rv_v1	Latest contract instance in V1 by KafkaTimestamp
rv_v2	rv_v2	The second latest contract instance in V1 by KafkaTimestamp
rv_v3	rv_v3	The third latest contract instance in V1 by KafkaTimestamp
V2	resvevents_previous_instance_view	View of previous events for current contracts, created by joining current events (V1) with previously saved instances in T1

Architectural Flow Explanation

1. On the left is the source of contract transactions, the *Kafka topic* Contracts-Save, which the *Databricks Spark Structured Streaming consumer* consumes.
2. The consumer connects with the *Confluent Schema Registry* to get the contract transactions message schema.
3. The consumer filters out all the unwanted transactions and splits the micro-batch DataFrame into individual data frames, having only the required columns, representing the main contract – having all the standard fields, truck, trailer, and tow data frames.
4. All these DataFrames are merged by contract and ordered by the latest KafkaTimestamp, i.e., in the descending order in which they were written to the Kafka topic. The view—V1 resvevents_view—is created from the merged DataFrame. However, to keep the data volume in check, we consider each contract's top three contract instances, represented by rv_1, rv_2, and rv_3.
5. These contracts are joined with Table T1-resvevents_latest, which has data on previous iterations for them, and a view—V2 resvevents_previous_instance_view—is created. This step is essential as, ultimately, for every contract, we want to compare the current values of specific fields to their previous values.
6. Now comes the crucial step of creating the main record for the table T2—resvevents_all. Here, we join view V1, which has current micro-batch contract instances, table T1, which has previous iterations' contract instances, and view V2, which has previous iterations' contract instances.
7. For any contract, Table T2 always has the current and previous values (just the last iteration) for specific fields.
8. The latest contract instances are updated to table T1 for the next iteration.

Steps 4 -5 and the basic SQL code are shown in the next section.

Data Processing Logic

- Block 1 in the above Arch Flow—The Databricks Spark Structured Streaming Consumer is utilized to read the Kafka topic for the contract transactions. The consumer connects to the Confluent Schema Registry using the message's Schema ID to obtain the schema required for message deserialization. Following this, the contract transactions are filtered to retain Truck, Trailer, and Towing equipment transactions.

- Blocks 2, 3, and 4 – Standard fields are extracted from each transaction after filtering, and a main-contract DataFrame (DF) is created. This main-contract DF is sorted by the Unique ID and KafkaTimestamp in descending order, ensuring that the latest transactions are positioned at the top. Subsequently, the main-contract DF is merged with respective Truck, Trailer, and Towing transactions to create the Latest_Contract_Instances view. A view, named resvevents_view – V1, is then established based on this merged data.
- Further steps involve splitting the data into three segments to focus on the top three latest instances—rv_1, rv_2, and rv_3.
- Subsequent actions are pivotal to the overall process, represented by connections 5 and 6 in the above architectural flow.
- Table T1 stores the latest saved instance for a contract. The views rv_1, rv_2, and rv_3 each are independently joined with contracts in T1 to form a view for previous instances, known as resvevents_previous_instance_view. New contract entries are added to T1 while the existing ones are updated.
- Finally, the latest records are generated by merging the current contracts from V1, previous instances from V2, and the newest contract instances from T1 and appending them to T2. Before writing to T2, the current values for specific fields are compared to their previous values. The change status is then updated in the array of Boolean fields, as illustrated in the code snippet below.

SQL code template to Update/Insert into T1

```
MERGE INTO T1 a

USING V1 b

on a.UniqueID = b.UniqueID

WHEN MATCHED THEN UPDATE

 Set a.Status = b.Status,

    a.Field1 = b.Field1,

    a.Field2 = b.Field2,

    a.Field3 = b.Field3,

    a.Pickup_Date = b.Pickup_Date,

    a.Pickup_City_State = b.Pickup_City_State,

    …

    a.TruckModel = b.TruckModel,

    a.TrailerModel = b.TrailerModel,

    a.FieldsUpdated = array(

     cast(!(a.Pickup_Date = b.Pickup_Date) as
int),

     cast(!(a.Pickup_City_State =
b.Pickup_City_State) as int),

     ...

     ...
    cast(!(a.TruckModel = b.TruckModel) as
int),

    cast(!(a.TrailerModel = b.TrailerModel) as
int),

)

WHEN NOT MATCHED THEN INSERT

(UniqueID,Field1,Field2,Field3,Pickup_Date,
Pickup_City_State,…,TruckModel,TrailerModel,F
ieldsUpdated)

VALUES

(b.UniqueID,b.Field1,b.Field2,b.Field3,b.Pickup_
Date,
b.Pickup_City_State,…,b.TruckModel,

b.TrailerModel,array(0,0,…,1,0))
```

Identification of value change in a specific field

In the above SQL Code, the bold font compares the old field values to the new ones and casts the Boolean to an integer. So, if the value has changed, then this position of the FieldsUpdated array field will be 1; otherwise, it will be 0. In the above SQL Code, the TruckModel's value was updated from the previous transaction as it has a 1 in the FieldsUpdated array.

SQL code template to Update/Insert into T2

```
INSERT INTO T2

SELECT
a.UniqueID,a.Field1,a.Field2,a.Field3,a.Pickup_
Date,
a.Pickup_City_State,…,a.TruckModel,
a.TrailerModel, pi.Field1, pi.Field2,
pi.Field3,pi.Pickup_Date,pi.Pickup_City_State,
pi.TruckModel, pi.TrailerModel,
b.FieldsUpdated

FROM V1 a JOIN T1 b on a.UniqueID =
b.UniqueID

   LEFT JOIN V2 pi on a.UniqueID =
pi.UniqueID
```

In the above SQL Code, for a contract, the old values, the new values, and a comparison of the critical fields—FieldsUpdated—are added to the Databricks delta table resvevents_all.

Integration Details With Kafka

Brokers

This case study uses bootstrapped on-premise Kafka brokers to give us complete control over the Kafka deployment, configuration, and management. However, one could also use Kafka brokers on Confluent or any other cloud. This way, the cloud company does the deployment, configuration, and management, and we can concentrate on the business logic.

Topic and Partitions

This approach employs a topic comprising twenty partitions for the contract events data. The decision to configure the topic with this specific number of partitions aligns with the necessity to accommodate high-volume and high-velocity data.

Consumer Group

In the current consumer group setup, we implement a single consumer that aligns with the maxOffsetsPerTrigger specification and effectively fulfills its intended purpose. maxOffsetsPerTrigger is set to 500K, half its maximum allowed value.

Error Handling and Recovery

Checkpointing

Checkpointing is a crucial aspect of stream processing that ensures fault tolerance and enables failure recovery. This solution uses a persistent path on the Databricks Delta file system for checkpointing.

Handling Kafka Errors

As the consumer is a Databricks spark structured streaming job, set the retries to four if the job fails due to a Kafka or intermittent network error.

Monitoring and Metrics

Confluent Control Center

This would help read the message instantaneously to ensure that the schema returned by the Confluent Schema Registry is correct and can be decoded. This would be mostly done during the test phase.

Grafana

Grafana monitors the Kafka feed in real time to see if there has been any downtime.

Scaling Considerations Topic and Partitions

Auto-Scaling

While configuring the consumer job cluster, a maximum number of worker nodes can be given, and auto-scaling could be enabled to add and remove workers based on the processing volume.

Integration with Confluent Platform Features

Schema Retrieval

This solution uses Confluent Schema Registry to manage schemas for our Kafka topics. This ensures that data is serialized and deserialized correctly, avoiding issues with schema evolution and data consistency across services.

Future Enhancements

Future Enhancements

- This architecture allows upscaling to multiple consumers, each processing data for a specific equipment category and distributing the load across various consumers.
- Create a feedback channel by writing changes to the critical fields to a Kafka topic for all the stakeholders to consume.

Conclusion

This initiative is a significant advancement in the truck rental industry's use of big data technologies for real-time tracking and analysis of contract transactions. The case study sets the stage for a more insightful and efficient decision-making process within the industry, showcasing the power of data-driven approaches in improving operational frameworks and service offerings in the truck rental sector. It demonstrates the importance of data analysis and research in driving industry advancements. Tracking key fields and understanding factors influencing changes helps us make informed decisions and improve operations for a better customer experience.

Short Descriptions of the Technologies Used

Apache Kafka

Apache Kafka is a distributed event streaming platform that handles high-throughput, real-time data feeds. Originally developed by LinkedIn, Kafka allows for the publishing, storing, and processing large streams of events in a fault-tolerant manner, making it a core technology for real-time analytics and data integration.

Confluent Schema Registry

The Confluent Schema Registry is a centralized service for managing schemas and ensuring data compatibility in Apache Kafka environments. It supports versioning, validating, and evolving schemas in formats like Avro, JSON, and Protobuf, enabling consistent data serialization and deserialization across distributed systems.

Azure Event Hub

Azure Event Hub is a scalable data streaming service provided by Microsoft Azure. It enables the ingestion and processing of millions of events per second from various sources. Event Hub is designed for real-time analytics and integrates seamlessly with other Azure services for comprehensive data processing and analytics workflows.

Databricks Delta Table

Databricks Delta Table is a storage layer in the Databricks Unified Data Analytics Platform that combines the features of data lakes and data warehouses. It provides ACID transactions, scalable metadata handling, and unified batch and streaming data processing, ensuring data reliability and performance for large-scale analytics.

Spark Structured Streaming Consumer

Spark Structured Streaming Consumer is an Apache Spark component that enables real-time stream processing using a high-level API. It supports fault tolerance, exactly-once processing semantics, and integration with various streaming sources like Apache Kafka and Azure Event Hubs, making it ideal for building robust and scalable real-time data pipelines.