Apache Kafka

Apache Kafka is an open-source distributed event streaming platform.

What is the need for Kafka when we have Spark Structured Streaming?

2 main functionalities offered by Kafka :

1. Data Integration / Collection Capabilities -> We can collect data coming at a very high speed - 100s of records coming in every second. It allows us to collect and store the data coming at such a fast pace.
2. Data Processing -> It allows processing of the collected data using Kafka Streams.

What is Socket and why is it not used in production?

- Socket is a combination of IP Address and Port
- Reasons why socket is not used in production environment is :

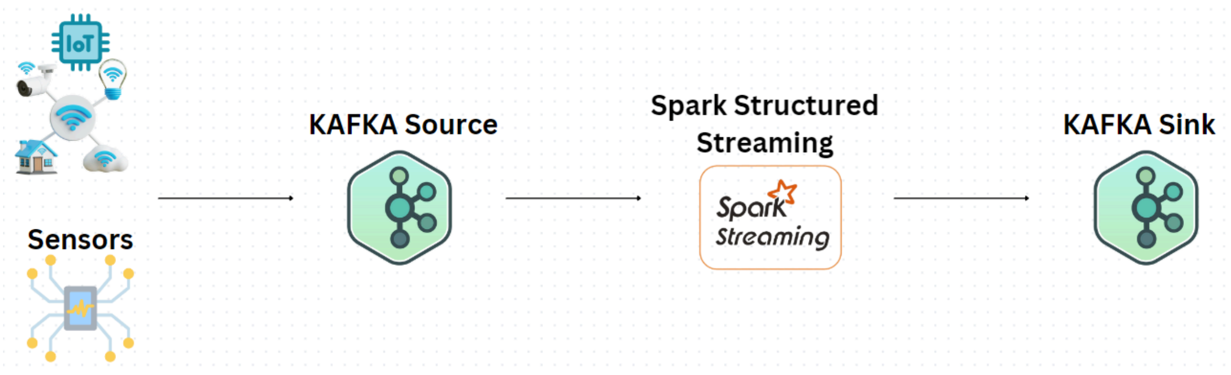
  -> **It is not a replayable source** : There cannot be multiple consumers for the same data, Once the data is consumed, it cannot be retrieved back.

  -> **Socket cannot store the data and doesn't have buffering capabilities** : This could lead to a data loss issue when we have a fast producer but a slow consumer.

(However, by default, the data can be retained for 7 days in case of Kafka. This will help in decoupling the producer and consumer.)

Note:

- Kafka is majorly used for Data Integration over Data Processing. Kafka can be used as a Source / Sink.

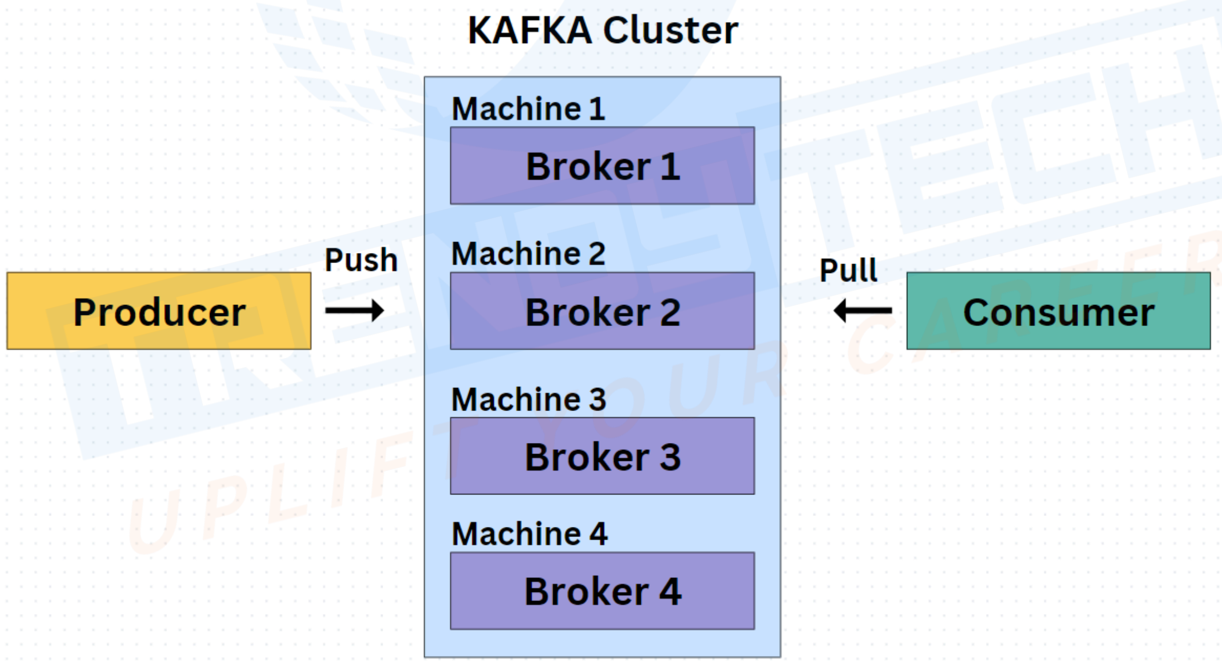


- Kafka is a messaging utility with a publisher-subscriber model.
- Producer -> Kafka Cluster -> Consumer
- Important to understand Producer API and Consumer API.

Kafka Cluster Internals / Architecture

Kafka Cluster is composed of Brokers.

Lets say we need a 4 Node Kafka Cluster -> For this we would need 4 Nodes/ Machines with Broker Software installed on each machine.



Best way to practise Kafka

-> **Confluent.io** : A Managed Kafka Environment on Cloud developed by creators of Kafka.

Core Concepts of Apache Kafka

- **Producer**
- **Consumer**
- **Broker**
- **Cluster**
- **Topic**
- **Partitions**
- **Partition Offset**
- **Consumer Groups**

Kafka as a Data Integration Platform

In traditional messaging systems, there can be only one consumer, as the data once consumed cannot be retrieved or used by other consumers.

In case of Kafka, there can be multiple consumers consuming the same data as the data is retained in the Kafka cluster by default for 7 days. I.e., we can have a single producer but multiple consumers.

Producer - Application that produces the data

Ex - Twitter, Instagram, etc

Consumer - Application that consumes the data

Ex - Spark streaming application

Broker - A software application installed on a node. These nodes with the broker installed on it collectively form the Kafka Cluster.

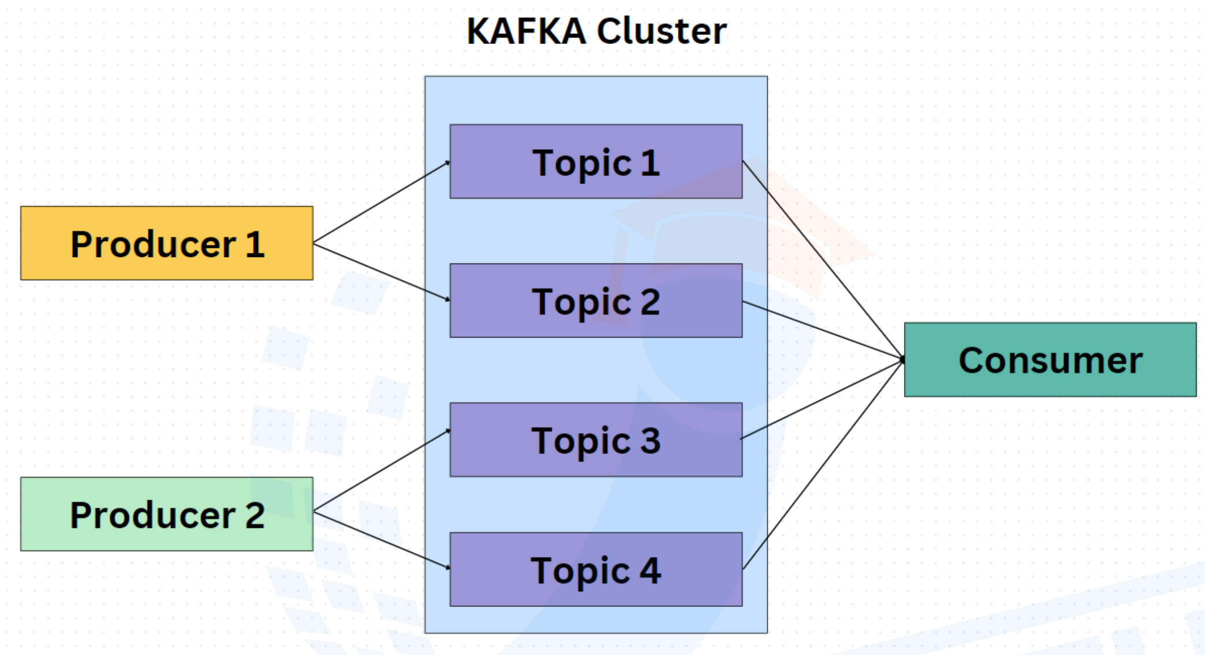
Cluster - A group of different brokers form a Cluster

Ex - A 10 node cluster is the one consisting of 10 broker nodes.

Topic - Is similar to a table in a database that consists of specific kinds of data. A unique name that stores a particular kind of data.

Ex - tweets_data, banking_data, employee_data

Note : Producer produces the data and this data is organised in specific topics. The consumer subscribes to the topic. Whenever there is new data getting added to the topic, the consumer will be notified.



Partitions - When a data grows huge and cannot be accommodated on a single topic of a machine, it becomes essential to partition the data to be able to store it across multiple Machines.

A topic can be divided into partitions and each partition can be stored on different Machines. The concept of Partitions enables the system to be scalable in a distributed environment.

There is no default number of partitions set. It is rather a design time decision and needs to be defined while creating the Topics.

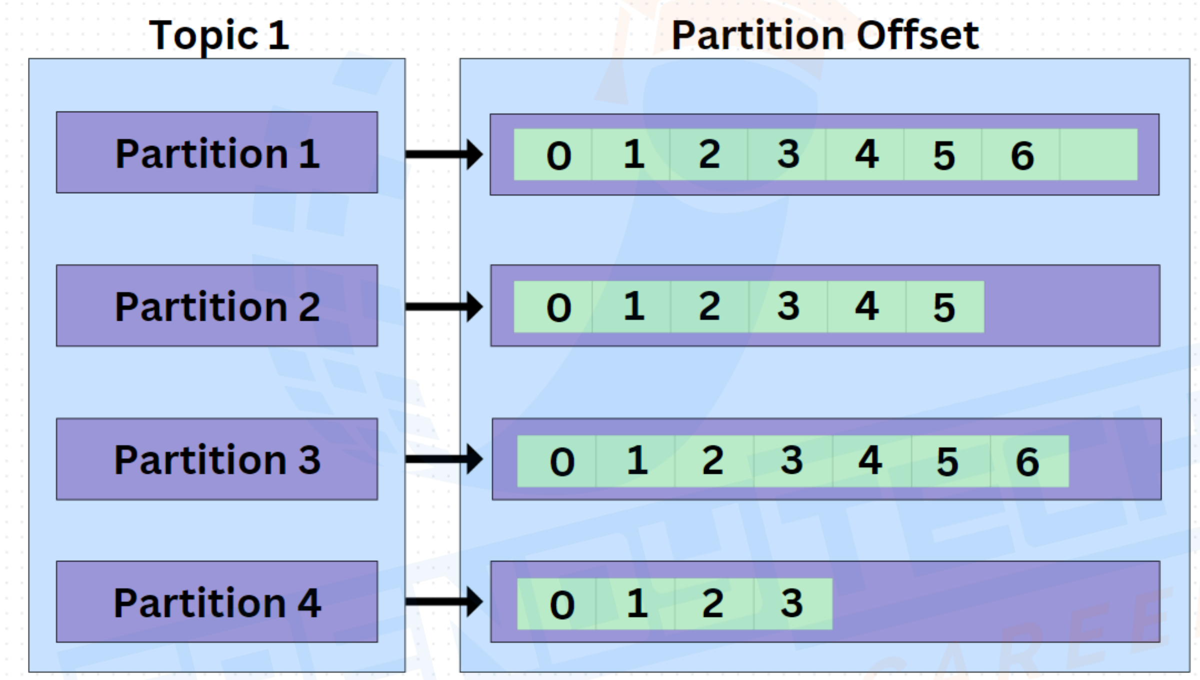
Partition offset - Is a Sequence ID for messages. In each partition, messages are present in a sequence and the latest arriving messages will have the higher offset.

Requirement : If we are required to read a specific message in the Kafka Cluster, what information would be required to fetch the desired message?

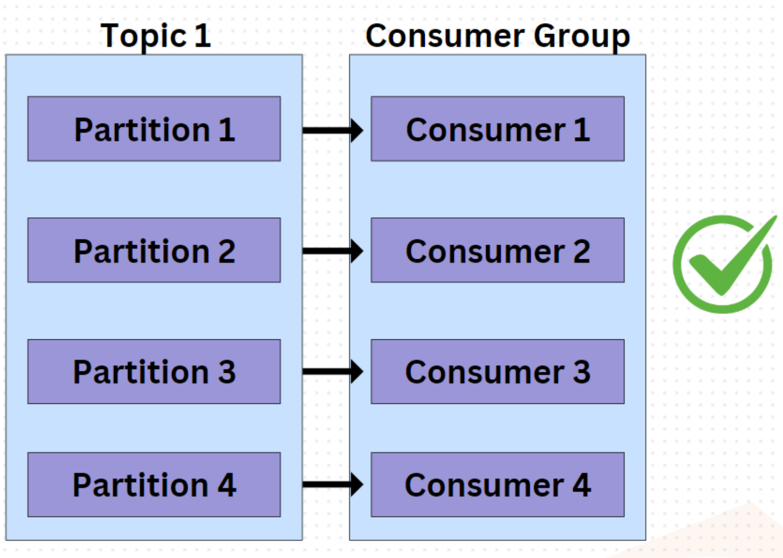
The following 3 important information would be required to fetch a particular message from the kafka cluster :

1. Topic Name
2. Partition Number
3. Partition Offset

Note : Topic is divided into Partitions to handle large volumes of data and make the system scalable. Each partition consists of a sequence of messages.

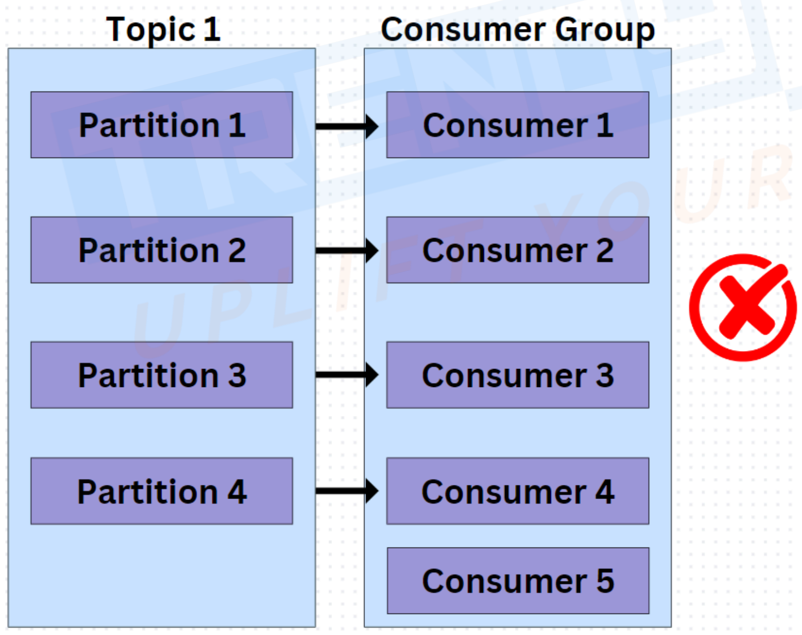


Consumer Group - If there are multiple Producers producing the data and we have multiple brokers to process this data. If the data is produced and processed at a very fast pace, then a single Consumer will not be able to take the complete load. In this scenario we would need a group of Consumers, who can share the load.



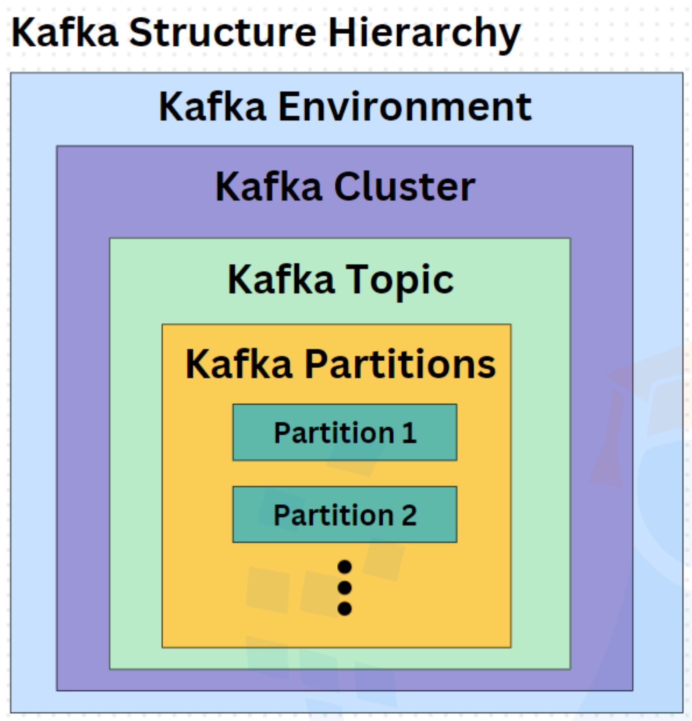
Note :

- The number of consumers cannot be more than the number of partitions in the topic. If the consumers are more than the partitions then some of the consumers will be idle.
- Consumer Groups provide scalability as the load is taken by multiple consumers and helps in faster processing. This ensures there is not consumer bottleneck
- One partitions cannot be handled by two different consumers as it can lead to inconsistent data.
- However, one consumer can handle multiple partitions.



Kafka Practice Environment

Creators of Kafka developed this cloud based commercial distribution platform https://www.confluent.io/ where we can execute Kafka.



Kafka Practicals

1. Create a Cluster
2. Create a Topic
3. Produce Message through **UI**
    - Select the created Topic -> Messages tab -> Actions button -> Produce New Message
    - Provide the Key and the JSON Value in the pop-up window to produce the message.
    - This message gets produced in partition-0 and offset-0. Calculation of the partition depends on the consistent hash function used.
    - With the consistent Hash function, the same key always goes to the same partition.
4. Kafka Messages can be produced through **CLI** as well.
    - environment list (command used to list the environment in the Kafka confluent CLI)

- environment use <resource-name> (connecting to the required environment)
- kafka cluster list (command used to list the clusters)
- kafka cluster use <resource-name> (connecting to the required cluster)
- api-key create --resource <cluster-name> (command to create API key and secret. Save the API key and the secret value)
- api-key use <generated-api-key> --resource <cluster-name>

  (Api key is set-up in order to produce or consume messages.)

- kafka topic list (command to list the topics along with the replication factor and partition count mentioned)
- kafka topic create <topic-name> (command to create a topic)
- kafka topic consume --from-beginning <topic-name> (command used to consume data from a particular topic from the beginning)
- kafka topic produce <topic-name> --parse-key (command used to produce the data)

  (need to feed the json data with the parse-key, i.e., the primary key in the beginning of the json data and enter in the terminal after executing the kafka produce command)

- Two terminals can be opened, one for the Producer and the other for the Consumer
- kafka topic describe <topic-name> (command used to get more details about a topic)
- kafka topic create --partitions <num-of-partitions> <topic-name> (command to create a topic with the desired number of partitions)

Kafka Message comprises of 2 important parts

(Key, Value)

Note: Assigning a Column as a Primary Key is a Conscious Design Decision, as it would involve choosing the right column which can optimise the operations (reduce the shuffling involved while aggregations)

Example : Say we have Customers dataset, then making Customer_id as a primary key will help in grouping all the orders placed by this customer into one partition and thereby reducing the shuffling of data required while aggregating.

Programmatic Approach to Implement a Use-case using Kafka

Consider a Retail-Chain use-case (like Walmart Store) with an orders dataset.

Code for Kafka Producer

- Whenever a purchase is made, the retail-store sends this transaction to Kafka topic
- Simulating a real-time scenario using a file consisting of transaction entries and using kafka producer to generate and mimic the real-time transactions.
- Design decision to choose the primary key : Since the business requirement is that the same customer transactions should always be sent to the same kafka partition always. Therefore, customer_id would be the primary key.
- Create a new project in PyCharm. Create a new file called myproducer.py in this project.
- Install **Confluent kafka** from the python packages in the PyCharm IDE and set the configurations by getting the details from the kafka cluster configuration tab.
- **Bootstrap servers** : Say we have a 100 node kafka cluster, it is not possible to give the IP addresses of all these, rather we provide the IP address of a few servers. The client can connect to one of these servers and get to know all the metadata of the other brokers in the cluster.
- **producer.produce(topic, key, value, callback)** is an asynchronous method used to produce the data to a given topic.
- **producer.poll()** : the producer() method asynchronously sends the messages and doesn't wait for the acknowledgement if the message was delivered. It will not execute the callback function. Therefore, the poll() method is used wherein the producer checks for any asynchronous events that need to be processed. Events include acknowledgement from brokers for messages sent / error. These messages are polled when the poll() method is executed.
- **producer.flush()** : This method is called prior to shutting down the producer to ensure all outstanding/ queued/ in-flight messages are delivered.

```
orders_df \
.writeStream \
.queryName(“ingestionQuery”) \
.format(“kafka”) \
.outputMode(“append”) \
.option(“checkpointLocation”, “<checkpoint-path>”) \
.option(“kafka.bootstrap.servers”, <value>) \
.option(“kafka.security.protocol”, “SASL_SSL”) \
.option(“kafka.sasl.mechanism”, “PLAIN”) \
.option(“kafka.sasl.jaas.config”, <value>) \
.option(“kafka.ssl.endpoint.identification.algorithm”, “https”) \
.option(“topic”, <confluent-topic-name>) \
.start()
```

Kafka Consumer

Requirement : Read the data from the Kafka Topic and persist the data into storage layer (Ex - Delta Table / Disk)

Steps :

1. In the **Databricks community edition**, create a cluster and under **Libraries** install the kafka package “**confluent-kafka[avro,json,protobuf]>=1.4.2**” under the type PyPi.

   With this package you can connect & communicate with the kafka cluster from databricks community edition.

2. Required details to connect to kafka cluster
   - Bootstrap server
   - API Key
   - API Secret
   - Topic name

All of the above details can be taken from the kafka cluster and in the databricks notebook, we need to pass the above values as part of configuration settings.

3. Reading from kafka using Spark

```
orders_df = spark \
.read \
.format(“kafka”) \
.option(“kafka.bootstrap.servers”, <value>) \
.option(“kafka.security.protocol”, “SASL_SSL”) \
.option("kafka.sasl.mechanism", "PLAIN") \
.option("kafka.sasl.jaas.config", <value>) \
.option("kafka.ssl.endpoint.identification.algorithm", "https") \
.option("subscribe", <confluent-topic-name>) \
.load()

display(orders_df)
```

Data will be displayed in binary format. Converting from binary to string :

```
converted_orders_df = orders_df.selectExpr("CAST(key as string) AS
key", "CAST(value as string) AS value", "topic", "partition", "offset",
"timestamp", "timestampType")
```

Note : This is a Batch Mode of reading data. As it will only read the data currently present in the kafka topic and will not pick the newly arriving data.

Streaming Mode of Reading the Data from Kafka :

Requirement : Read the data from the Kafka Topic and persist the data into storage layer (Ex - Delta Table / Disk)

Steps :

1. In the **Databricks community edition**, create a cluster and under **Libraries** install the kafka package "**confluent-kafka[avro,json,protobuf]>=1.4.2**" under the type PyPi.

   With this package you can connect & communicate with the kafka cluster from databricks community edition.

2. Required details to connect to kafka cluster
   - Bootstrap server
   - API Key
   - API Secret
   - Topic name

All of the above details can be taken from the kafka cluster and in the databricks notebook, we need to pass the above values as part of configuration settings.

3. Reading from kafka, the **Streaming data** using Spark in the form of **micro-batches**

```
orders_df = spark \
.readStream \
.format(“kafka”) \
.option(“kafka.bootstrap.servers”, <value>) \
.option(“kafka.security.protocol”, “SASL_SSL”) \
.option(“kafka.sasl.mechanism”, “PLAIN”) \
.option(“kafka.sasl.jaas.config”, <value>) \
.option(“kafka.ssl.endpoint.identification.algorithm”, “https”) \
.option(“subscribe”, <confluent-topic-name>) \
.option(“startingTimestamp”,1) \
.option(“maxOffsetsPerTrigger”,50) \
.load()

display(orders_df)
```

Data will be displayed in binary format. Converting from binary to string :

```
converted_orders_df = orders_df.selectExpr(“CAST(key as string) AS key”, “CAST(value as string) AS value”, “topic”, “partition”, “offset”, “timestamp”, “timestampType”)
```

- maxOffsetsPerTrigger option ensures that all the micro-batches are not uneven.

4. Writing the data to the delta table

```
converted_orders_df \
.writeStream \
.queryName(“ingestionquery”) \
.format(“delta”) \
.outputmode(“append”) \
.option(“checkpointLocation”, “<checkpoint-path>”) \
.toTable(“<table-name>”)
```

Persisting the Cleaned & Formatted Data to Delta Table :

Imposing JSON structure

- Import the following to use the from_json function

```
from pyspark.sql.functions import *
```

- create a orders schema

orders_schema = “order_id long, customer_id long, customer_fname string, customer_lname string, city string, state string, pincode long, line_items array<struct<order_item_id : long, order_item_product_id : long, order_item_quantity : long, order_item_product_price : float, order_item_subtotal : float>>”

parsed_orders_df = converted_orders_df.select(“key”, from_json(“value”, orders_schema).alias(“value”), “topic”, “partition”, “offset”. “timestamp”, “timestampType”

parsed_orders_df.createOrReplaceTempView(“orders”)

exploded_orders = spark.sql(“““select key, value.order_id as order_id, value.customer_id as customer_id, value.customer_fname as customer_fname, value.customer_lname as customer_lname, value.city as city, value.state as state, value.pincode as pincode, explode(value.line_items as lines from orders”””)

exploded_orders.createOrReplaceTempView(“exploded_orders”)

flattened_orders = spark.sql(“““select key, value.order_id as order_id, value.customer_id as customer_id, value.customer_fname as customer_fname, value.customer_lname as customer_lname, value.city as city, value.state as state, value.pincode as pincode, lines.order_item_id as item_id, lines.order_item_product_id as product_id, lines.order_item_quantity as quantity, lines.order_item_product_price as product_price, lines.order_item_subtotal as subtotal from exploded_orders”””)

Kafka Producer Streaming :

Writing the Data to a Kafka Sink - Producing the data to a Kafka Topic(that is acting as a sink) after processing.

Example : We need to retain only those order entries where the **city = “Chicago”**

filtered_orders = spark.sql(“select CAST(key as string) as key, CAST(value as string) as value from orders **where value.city = ‘chicago’”)**

```
//writing to topic

        filtered_orders \
        .writeStream \
        .queryName(“ingestionQuery”) \
        .format(“kafka”) \
        .outputMode(“append”) \
        .option(“checkpointLocation”, “<checkpoint-path>”) \
        .option(“kafka.bootstrap.servers”, <value>) \
        .option(“kafka.security.protocol”, “SASL_SSL”) \
        .option(“kafka.sasl.mechanism”, “PLAIN”) \
        .option(“kafka.sasl.jaas.config”, <value>) \
        .option(“kafka.ssl.endpoint.identification.algorithm”, “https”) \
        .option(“topic”, <confluent-target-topic-name>) \
        .start()
```