**Flatten Deep Nested JSON**

Kafka Connect provides a built-in **Flatten SMT** that allows you to flatten nested fields in the message key or value.

**Example Connector Configuration:**

"transforms": "Flatten",

"transforms.Flatten.type": "org.apache.kafka.connect.transforms.Flatten$Value",

"transforms.Flatten.delimiter": "\_",

"transforms.Flatten.ignore.missing.or.null": "true"

**Explanation**:

* Flatten$Value: This applies flattening to the message value.
* delimiter: Specifies how nested fields are concatenated — here using \_.
* ignore.missing.or.null: Whether missing fields should be ignored.

**2. Define a Schema in Kafka Schema Registry**

If you're using **Confluent Schema Registry** with Avro or Protobuf, you can:

1. Define nested schemas explicitly.
2. Flatten them when producing or transforming data via Kafka Connect or a stream processor.

**3. Use Kafka Streams or KSQL for Transformation**

* **Kafka Streams** allows you to parse nested JSON and re-map fields to flat key-value pairs.
* **ksqlDB** supports JSON parsing and field extraction functions to build flattened views.

Example KSQL statement:

CREATE STREAM transactions\_raw (

user STRUCT<id INT, name STRING>,

transaction STRUCT<amount DOUBLE, currency STRING>,

timestamp STRING

) WITH (...);

CREATE STREAM transactions\_flat AS

SELECT

user->id AS user\_id,

user->name AS user\_name,

transaction->amount AS transaction\_amount,

transaction->currency AS transaction\_currency,

timestamp

FROM transactions\_raw;

**4. Flattening Outside Kafka**

Alternatively, flatten the JSON at the producer before sending it to Kafka, or at the consumer after retrieving it.

* **In Python**: Use libraries like json\_normalize from pandas.
* **In Java**: Manually parse the JSON and reconstruct the flat version.
* **In Spark**: Use built-in flattening functions to expand nested structures.