Streams and Tables in Apache Kafka: A Primer

JAN 13, 2020 READ TIME: 9 MIN

This four-part series explores the core fundamentals of Kafka's storage and processing layers and how they interrelate. In this first part, we begin with an overview of events, streams, tables, and the stream-table duality to set the stage. The subsequent parts will take a closer look at Kafka's storage layer—the distributed "filesystem" for streams and tables—and then move to the processing layer on top.

In my daily work as a member of Confluent's Office of the CTO and as the former product manager for ksqlDB and Kafka Streams, I interact with many users of Apache Kafka—be it developers, operators, or architects. Some have a stream processing or Kafka background, some have their roots in relational databases like Oracle and MySQL, and some have neither. But many of them have the same set of technical questions, such as: what's the difference between an event stream and a database table? Is a Kafka topic the same as a stream? How can I best leverage all these pieces when I want to put my data in Kafka to use?

By the end of this series, you will have answers to each of these common questions and many more. If you are interested to learn about Kafka, I invite you to join me on this journey through Kafka's core fundamentals!

This article is the first in our series on Apache Kafka® fundamentals:

1. ***Streams and Tables in Apache Kafka: A Primer (this article)***
2. *Streams and Tables in Apache Kafka: Topics, Partitions, and Storage Fundamentals*
3. *Streams and Tables in Apache Kafka: Processing Fundamentals with Kafka Streams and ksqlDB*

WRITTEN BY

Michael Noll

Product Manager

4. *Streams and Tables in Apache Kafka: Elasticity, Fault Tolerance, and other Advanced Concepts*

Events, streams, and tables

Let us start with the basics: What is Apache Kafka? Kafka is an *event streaming platform*. As such it provides, next to many other features, three key functionalities in a scalable, fault-tolerant, and reliable manner:

1. It lets you *publish* and *subscribe* to events
2. It lets you *store* events for as long as you want
3. It lets you *process* and *analyze* events

This sounds like a very attractive piece of technology—but what *is* an event in this context?

An event records the fact that "something happened" in the world. Conceptually, an event has a key, value, and timestamp. A concrete event could be a plain notification without any additional information, but it could also include the full details of what exactly happened to facilitate subsequent processing. For instance:

- Event key: "Alice"
- Event value: "Has arrived in Rome"
- Event timestamp: "Dec. 3, 2019 at 9:06 a.m."

Other example events include:

- A good was sold
- A row was updated in a database table
- A wind turbine sensor measured 14 revolutions per minute
- An action occurred in a video game, such as "White moved the e2 pawn to e4"
- A payment of $200 was made by Frank to Sally on Nov. 24, 2019, at 5:11 p.m.

Surely you'll have your own examples as well of the many events that underlie and drive your company's business.

Events are captured by an event streaming platform into event streams. **An *event stream* records the history of what has happened in the world as a sequence of events.** An example stream is a sales ledger or the sequence of moves in a chess match. With Kafka, such a stream may record the history of your business for hundreds of years. This history is an ordered sequence or chain of events, so we know which event happened before another event to infer causality (e.g., “White moved the e2 pawn to e4, then Black moved the e7 pawn to e5”). A stream thus represents both the past and the present: as we go from today to tomorrow—or from one millisecond to the next—new events are constantly being appended to the history.

Compared to an event stream, **a *table* represents the state of the world** at a particular point in time, typically “now.” An example table is total sales or the current state of the board in a chess match. A table is a view of an event stream, and this view is continuously being updated whenever a new event is captured.

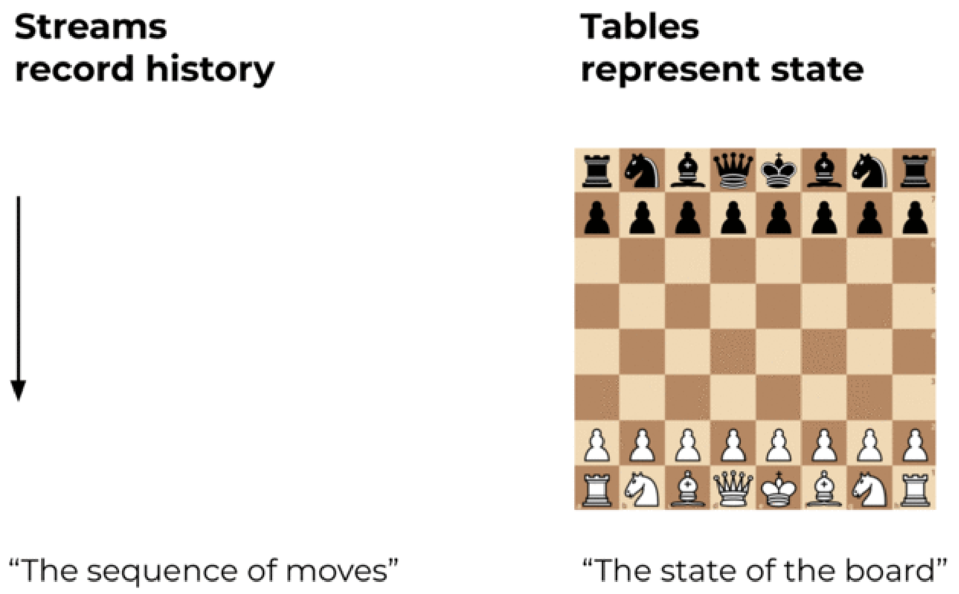


Figure 1. Streams record history. Tables represent state.

Streams and tables in Kafka differ in a few ways, notably with regard to whether their contents can be changed, i.e., whether they are *mutable*. (If you are a Kafka Streams user: when I say *table* I refer to what is called a *KTable* in Kafka Streams. I am not talking about state stores, which we will cover later on.)

- A **stream** provides immutable data. It supports only inserting (appending) new events, whereas existing events cannot be changed. Streams are persistent, durable, and fault tolerant. Events in a stream can be keyed, and you can have many events for one key, like "all of Bob's payments." If you squint a bit, you could consider a stream to be like a table in a relational database (RDBMS) that has no unique key constraint and that is append only.

- A **table** provides mutable data. New events—rows—can be inserted, and existing rows can be updated and deleted. Here, an event's key aka row key identifies which row is being mutated. Like streams, tables are persistent, durable, and fault tolerant. Today, a table behaves much like an RDBMS materialized view because it is being changed automatically as soon as any of its input streams or tables change, rather than letting you directly run insert, update, or delete operations against it. More on this later.

Here's a juxtaposition of their behavior:

	Stream	Table
First event with key bob arrives	Insert	Insert
Another event with key bob arrives	Insert	Update
Event with key bob and value null arrives	Insert	Delete
Event with key null arrives	Insert	<ignored>

Stream-table duality

Notwithstanding their differences, we can observe that there is a close relationship between a stream and a table. We call this the stream-table duality. What this means is:

- *We can turn a stream into a table* by aggregating the stream with operations such as COUNT() or SUM(), for example. In our chess analogy, we could reconstruct the board's latest state (table) by replaying all recorded moves (stream).
- *We can turn a table into a stream* by capturing the changes made to the table—inserts, updates, and deletes—into a "change stream." This process is often called change data capture or CDC for short. In the chess analogy, we could achieve this by observing the last played move and recording it (into the stream) or, alternatively,

by comparing the board's state (table) before and after the last move and then recording the difference of what changed (into the stream), though this is likely slower than the first option.

In fact, a table is fully defined by its underlying change stream. If you have ever worked with a relational database such as Oracle or MySQL, these change streams exist there, too! Here, however, they are a hidden implementation detail—albeit an absolutely critical one—and have names like redo log or binary log. In event streaming, the redo log is much more than an implementation detail. It's a first-class entity: a stream. We can turn streams into tables and tables into streams, which is one reason why we say that event streaming and Kafka are turning the database inside out.

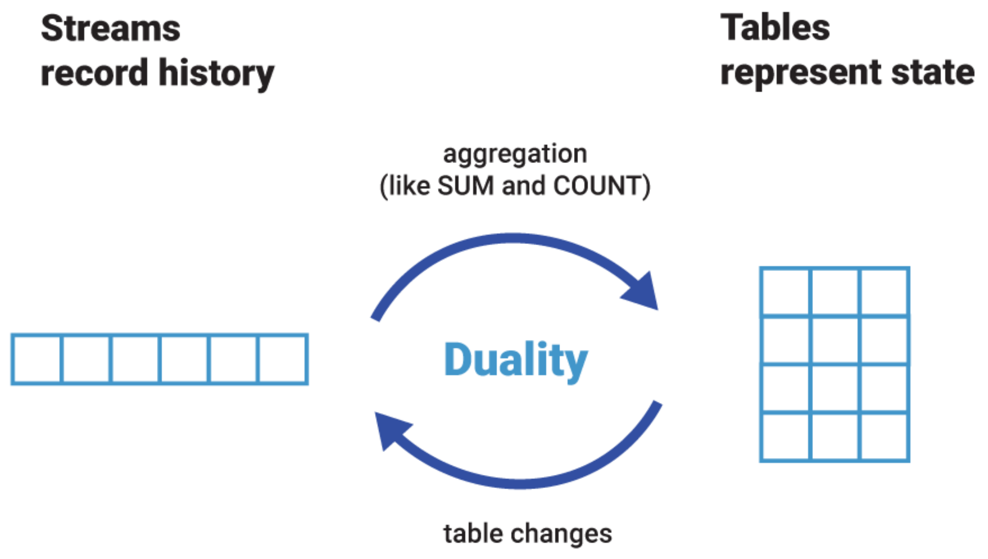


Figure 2. Because of the stream-table duality, we can easily turn a stream into a table, and vice versa. Even more, we can do this in a continuous, streaming manner so that both the stream and the table are always up to date with the latest events.

Below is an example of using COUNT() to aggregate a stream into a table. For illustrative reasons, I do not show event timestamps. The table is continuously updated as new events arrive in the stream, similar to a materialized view in relational databases, but it supports millions of events per second. Think of this as performing CDC on a table yielding an *output* change stream of the table. Performing an aggregation on an event stream is the opposite: the stream acts as the *input* change stream for the table.

In this example, a stream of events with key=username and value=location is being aggregated into a continuously updated table that tracks the number of visited locations per key=username:

Figure 3. Aggregating an event stream into a table

Here are the code examples for continuously aggregating a stream into a table:

- ksqlDB:
- Kafka Streams:

We can zoom out of the COUNT() example to also show the table's (output) change stream. The change stream can be used to react in real time to table changes, for example, to generate alerts. It can also be used for operational purposes, such as migrating a table from machine A to machine B in the case of infrastructure failure or when elastically scaling an application in and out.

Figure 4. Every table has its own change stream (also called a changelog).

We will return to the stream-table duality concept later in this series, as it is not only helpful when writing your own applications but is also so foundational that Kafka leverages it for elastic scalability and fault tolerance, to name a few!

Summary

This completes the first part of this series, where we learned about the basic elements of an event streaming platform: events, streams, and tables. We also introduced the stream-table duality and had a first look at why it's central to an event streaming platform like Apache Kafka. But of course, this was just the beginning! Part 2 will take a deep dive into Kafka topics, partitions, and storage fundamentals, where we explore topics and—in my opinion, the most important concept in Kafka: partitions.

Say Hello World to event streaming

If you're ready to get more hands on, there is a way for you to learn how to use Apache Kafka the way you want: by writing code. Apply functions to data, aggregate messages, and join streams and tables with Kafka Tutorials, where you'll find tested, executable examples of practical operations using Kafka, Kafka Streams, and ksqlDB.

Other articles in this series

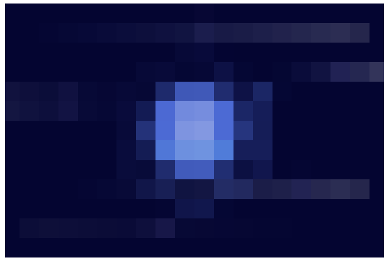
- Part 2 – Streams and Tables in Apache Kafka: Topics, Partitions, and Storage Fundamentals (next article)
- Part 3 – Streams and Tables in Apache Kafka: Processing Fundamentals with Kafka Streams and ksqlDB
- Part 4 – Streams and Tables in Apache Kafka: Elasticity, Fault Tolerance, and Other Advanced Concepts

Michael is a former principal technologist in the Office of the CTO at Confluent, the company founded by the original creators of Apache Kafka®. He focuses on longer-term product and technology strategy. Previously, Michael was the lead product manager for stream processing at Confluent, where his team created Kafka Streams and the streaming database ksqlDB. He is a well-known technology blogger in the big data community (WWW.MICHAEL-NOLL.COM) and a committer/contributor to open source projects such as Apache Storm and Apache Kafka.

Did you like this blog post? Share it now

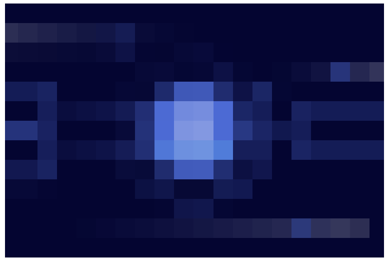
Technology < Use Cases

Subscribe to the Confluent blog



Introducing Versioned State Store in Kafka Streams

AUG 24, 2023



Delivery Guarantees and the Ethics of Teleportation

APR 25, 2023

Subscribe

Versioned key-value state stores, introduced to Kafka Streams in 3.5, enhance stateful processing capabilities by allowing users to...

VICTORIA XIA

This blog post discusses the two generals problems, how it impacts message delivery guarantees, and how those guarantees would...

WADE WALDRON

Product

Confluent Cloud
Confluent Platform
Connectors
Flink
Stream Governance
Confluent Hub
Subscription
Professional Services
Training
Customers

Cloud

Confluent Cloud
Support
Sign Up
Log In
Cloud FAQ

Solutions

Financial Services
Insurance
Retail and eCommerce
Automotive
Government
Gaming
Communication Service Providers
Technology
Manufacturing
Fraud Detection
Customer 360
Messaging Modernization
Streaming Data Pipelines
Event-driven Microservices
Mainframe Integration
SIEM Optimization
Hybrid and Multicloud

Developers

Confluent Developer
What is Kafka?
Resources
Events
Webinars
Meetups
Current: Data Streaming Event
Tutorials
Docs
Blog

About

Investor Relations
Startups
Company
Careers
Partners
News
Contact
Trust and Security

Internet of Things

Data Warehouse

Database