1 *Welcome to the kafka event streaming platform*

This chapter covers

- Defining event streaming and events
- Introducing the Kafka event streaming platform
- Applying the platform to a concrete example

We live in a world today of unprecedented connectivity. We can watch movies on demand on an IPad, get instant notification of various accounts' status, pay bills, and deposit checks from our smartphones. If you chose to, you can receive updates on events happening around the world 24/7 by watching your social media accounts.

While this constant influx of information creates more entertainment and opportunities for the human consumer, more and more of the users of this information are software systems using other software systems. Consequently, businesses are forced to find ways to keep up with the demand and leverage the available flow of information to improve the customer experience and improve their bottom lines. For today's developer, we can sum up all this digital activity in one term: event streaming.

1.1 What is event streaming ?

In a nutshell, event streaming is capturing events generated from different sources like mobile devices, customer interaction with websites, online activity, shipment tracking, and business transactions. Event streaming is analogous to our nervous system, processing millions of events and sending signals to the appropriate parts of our body. Some signals are generated by our actions such as reaching for an apple, and other signals are handled unconsciously, as when your heart rate increases in anticipation of some exciting news. We could also see activities from machines such as sensors and inventory control as event streaming.

But event streaming doesn't stop at capturing events; it also means processing and durable storage.

The ability to process the event stream immediately is essential for making decisions based on real-time information. For example, does this purchase from customer X seem suspicious? Are the signals coming from this temperature sensor seem to indicate that something has gone wrong in a manufacturing process? Has the routing information been sent to the appropriate department of a business?

The value of the event stream is not limited to immediate information. By providing durable storage, we can go back and look at event stream data-in its raw form or perform some manipulation of the data for more insight.

1.1.1 What is an event ?

So we've defined what an event stream is, but what is an event? We'll define event very simply as "something that happens"[1]. While the term event probably brings something to mind something *notable* happening like the birth of a child, a wedding, or sporting event, we're going to focus on smaller, more constant events like a customer making a purchase (online or in-person), or clicking a link on a web-page, or a sensor transmitting data. Either people or machines can generate events. It's the sequence of events and the constant flow of them that make up an event stream.

Events conceptually contain three main components:

1. Key - an identifier for the event
2. Value - the event itself
3. timestamp - when the event occurred

Let's discuss each of these parts of an event in a little more detail. The key could be an identifier for the event, and as we'll learn in later chapters, it plays a role in routing and grouping events. Think of an online purchase, and using the customer id is an excellent example of the key. The value is the event payload itself. The event value could be a trigger such as activating a sensor

when someone opens a door or a result of some action like the item purchased in the online sale. Finally, the timestamp is the date-time when recording when the event occurred. As we go through the various chapters in this book, we'll encounter all three components of this "event trinity" regularly.

1.1.2 An event stream example

Let's say you've purchased a Flux Capacitor, and you're excited to receive your new purchase. Let's walk through the events leading up to the time you get your brand new Flux Capacitor, using the following illustration as your guide.

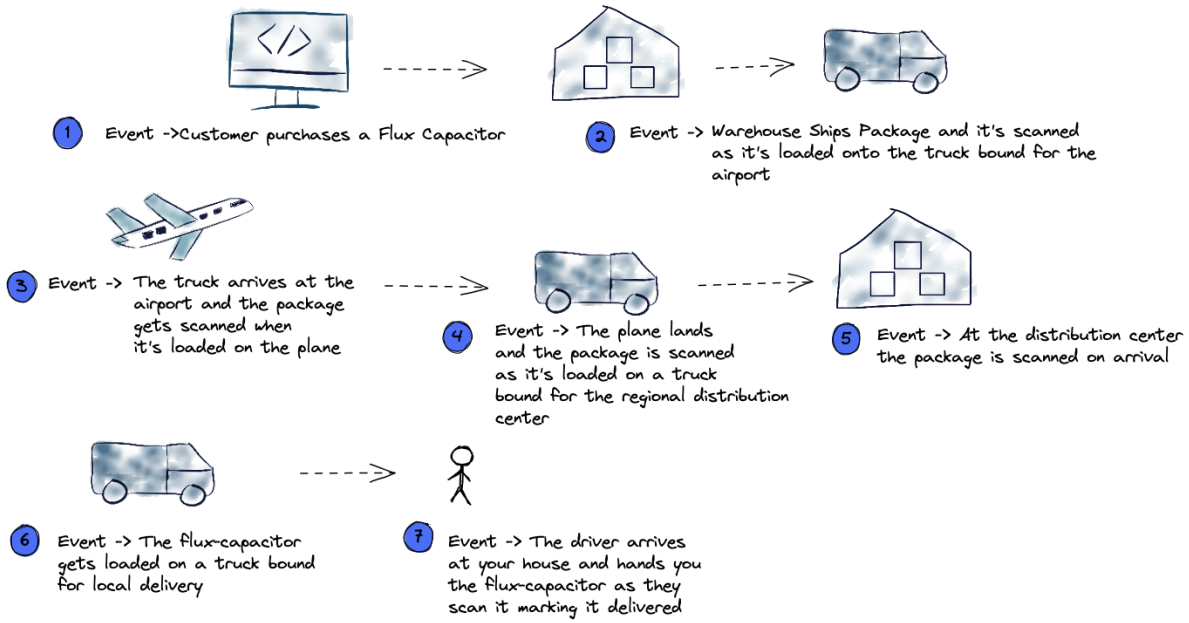

Figure 1.1 A sequence of events comprising an event stream starting with the online purchase of the flux ch01capacitor

1. You complete the purchase on the retailer's website, and the site provides a tracking number.
2. The retailer's warehouse receives the purchase event information and puts the Flux Capacitor on a shipping truck, recording the date and time your purchase left the warehouse.
3. The truck arrives at the airport, the driver loads the Flux Capacitor on a plane, and scans a barcode recording the date and time.
4. The plane lands, and the package is loaded on a truck again headed for the regional distribution center. The delivery service records the date and time when they've loaded your Flux Capacitor.
5. The truck from the airport arrives at the regional distribution center. A delivery service employee unloads the Flux Capacitor , scanning the date and time of the arrival at the distribution center.
6. Another employee takes your Flux Capacitor, scans the package saving the date and time, and loads it on a truck bound for delivery to you.
7. The driver arrives at your house, scans the package one last time, and hands it to you.

You can start building your time-traveling car!

From our example here, you can see how everyday actions create events, hence an event stream. The individual events here are the initial purchase, each time the package changes custody, and the final delivery. This scenario represents events generated by just one purchase. But if you think of the event streams generated by purchases from Amazon and the various shippers of the products, the number of events could easily number in the billions or trillions.

1.1.3 Who needs event streaming applications

Since everything in life can be considered an event, then pretty much any problem domain will benefit from using event streams. But there are some areas where it's more important to do so. Here are some typical examples

- *Credit card fraud* — A credit card owner may be unaware of unauthorized use. By reviewing purchases as they happen against established patterns (location, general spending habits), you may be able to detect a stolen credit card and alert the owner.
- *Intrusion detection* — The ability to monitor aberrant behavior in real-time is critical for the protection of sensitive data and well being of an organization.
- *The Internet of Things* - With IoT, there are sensors located in all kinds of places, and they all send back data very frequently. The ability to quickly capture this data and process it in a meaningful way is essential; anything less diminishes the effect of having these sensors deployed.
- *The financial industry* — The ability to track market prices and direction in real-time is essential for brokers and consumers to make effective decisions about when to sell or buy.
- *Sharing data in real-time* - Large organizations, like corporations or conglomerates, that have many applications need to share data in a standard, accurate, and real-time way

If the event-stream provides essential and actionable information, businesses and organizations need event-driven applications to capitalize on the information provided. In the next section, we'll break down the different components of the Kafka event streaming platform.

I've made a case for building event-streaming applications. But streaming applications aren't a fit for every situation.

Event-streaming applications become a necessity when you have data in different places or you have a large volume of events that you need to use distributed data stores to handle the volume. So if you can manage with a single database instance, then streaming is not a necessity. For example, a small e-commerce business or a local government website with mostly static data aren't good candidates for building an event-streaming solution.

1.2 Introducing the Apache Kafka® event streaming platform

The Kafka event streaming platform provides the core capabilities for you to implement your event streaming application from end-to-end. We can break down these capabilities into three main areas: publish/consume, durable storage, and processing. This move, store, and process trilogy enables Kafka to operate as the central nervous system for your data.

Before we go on, it will be useful to give you an illustration of what it means for Kafka to be the central nervous system for your data. We'll do this by showing before and after illustrations.

Let's first look at an event-streaming solution where each input source requires separate infrastructure:

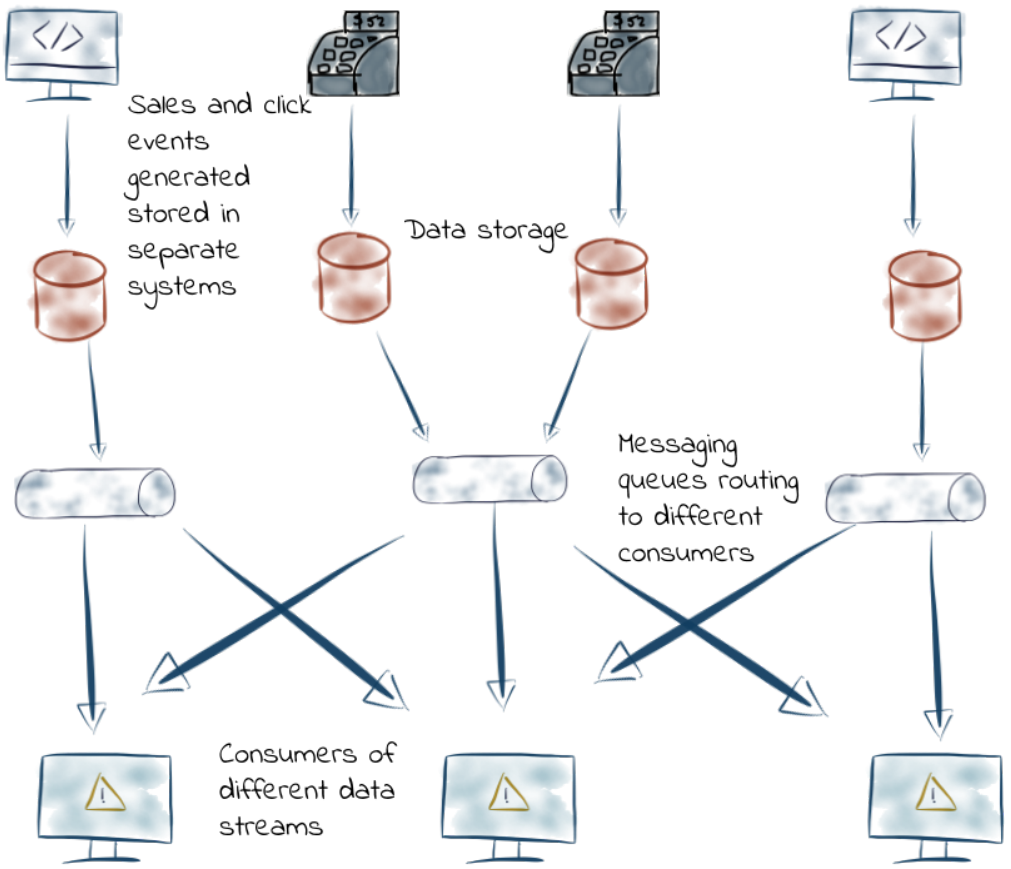

Figure 1.2 Initial event-streaming architecture leads to complexity as the different departments and data streams sources need to be aware of the other sources of events

In the above illustration, you have individual departments creating separate infrastructure to meet their requirements. But other departments may be interested in consuming the same data, which leads to a more complicated architecture to connect the various input streams.

Now let's take a look at how using the Kafka event streaming platform can change things.

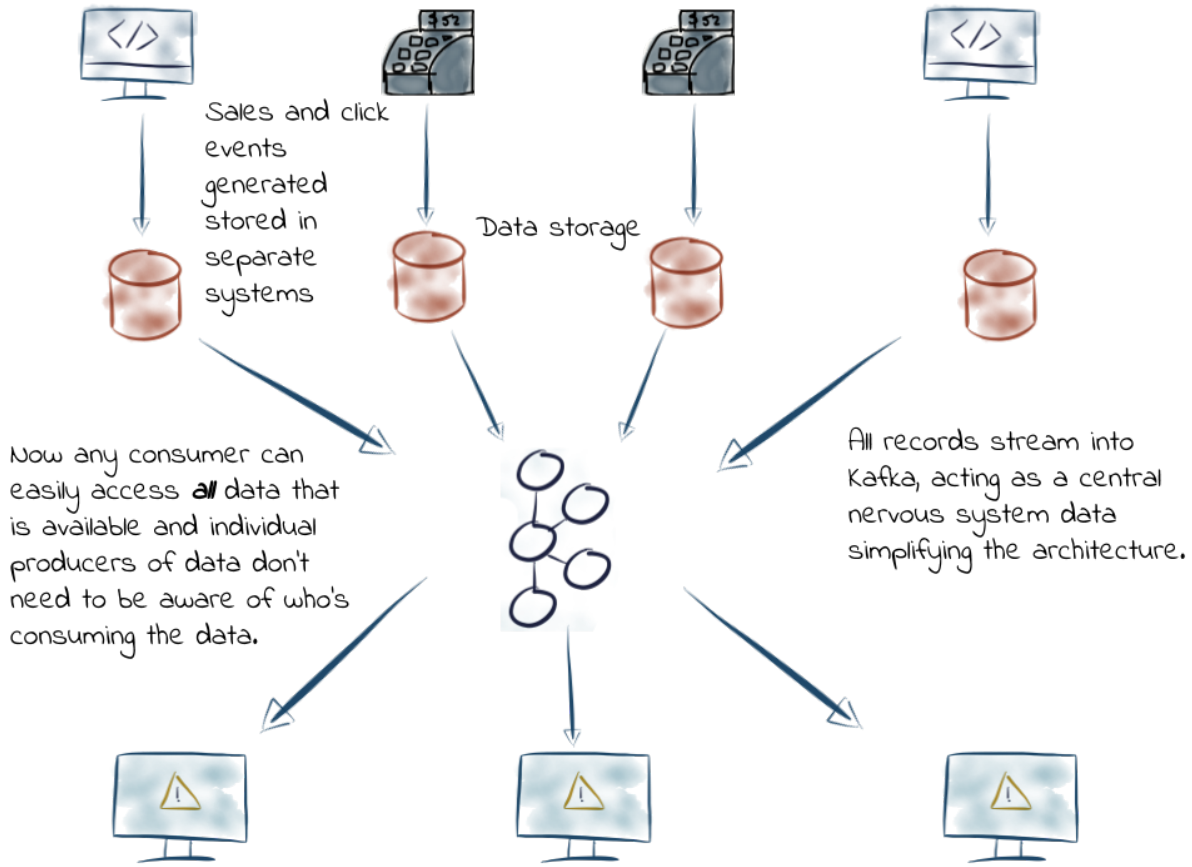

Figure 1.3 Using the Kafka event streaming platform the architecture is simplified

As you can see from this updated illustration, the architecture is greatly simplified with the Kafka event streaming platform's addition. All components now send their records to Kafka. Additionally, consumers read data from Kafka with no awareness of the producers.

At a high level, Kafka is a distributed system of servers and clients. The servers are called brokers, and the clients are record producers sending records to the brokers, and the consumer clients read records for the processing of events.

1.2.1 Kafka brokers

Kafka brokers durably ***store*** your records in contrast with traditional messaging systems (RabbitMQ or ActiveMQ) where the messages are ephemeral. The brokers store the data agnostically as the key-value pairs (and some other metadata fields) in byte format and are somewhat of a black box to the broker.

Providing storage of events has more profound implications as well concerning the difference between messages and events. You can think of messages as "tactical" communication between two machines, while events represent business-critical data that you don't want to throw away.

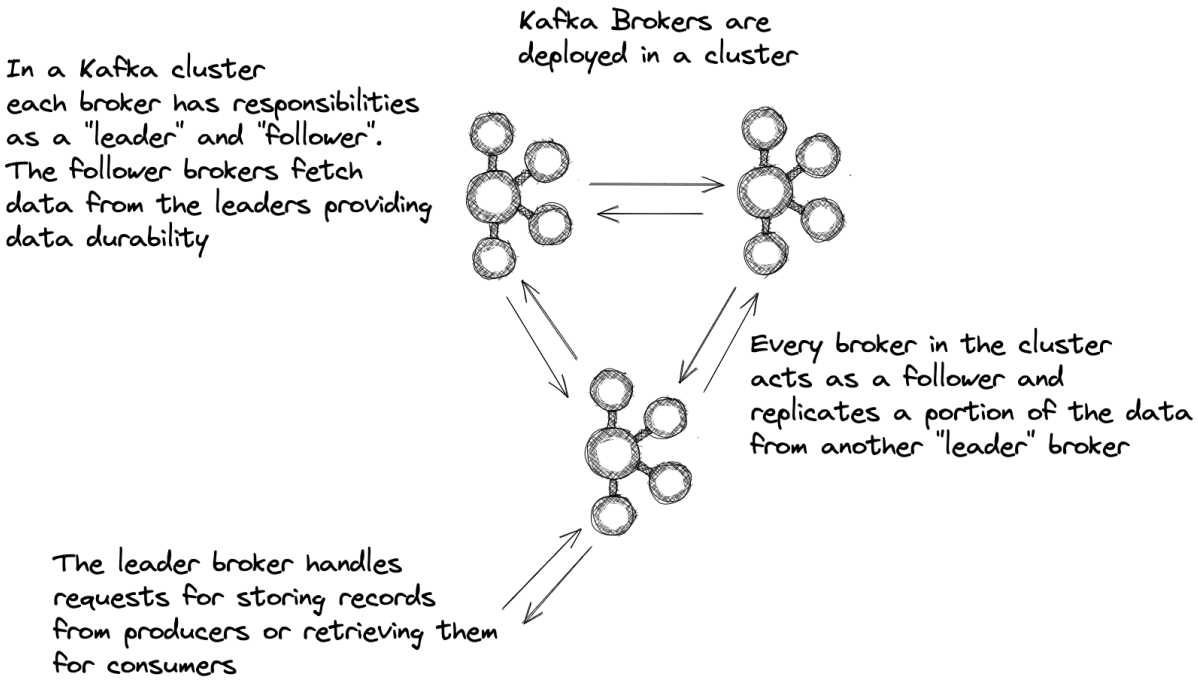

Figure 1.4 You deploy brokers in a cluster, and brokers replicate data for durable storage

From this illustration, you can see that Kafka brokers are the storage layer within the Kafka architecture and sit in the "storage" portion of the event-streaming trilogy. But in addition to acting as the storage layer, the brokers provide other essential functions such as serving requests from clients to providing coordination for consumers. We'll go into details of broker functionality in chapter 2.

1.2.2 Schema registry

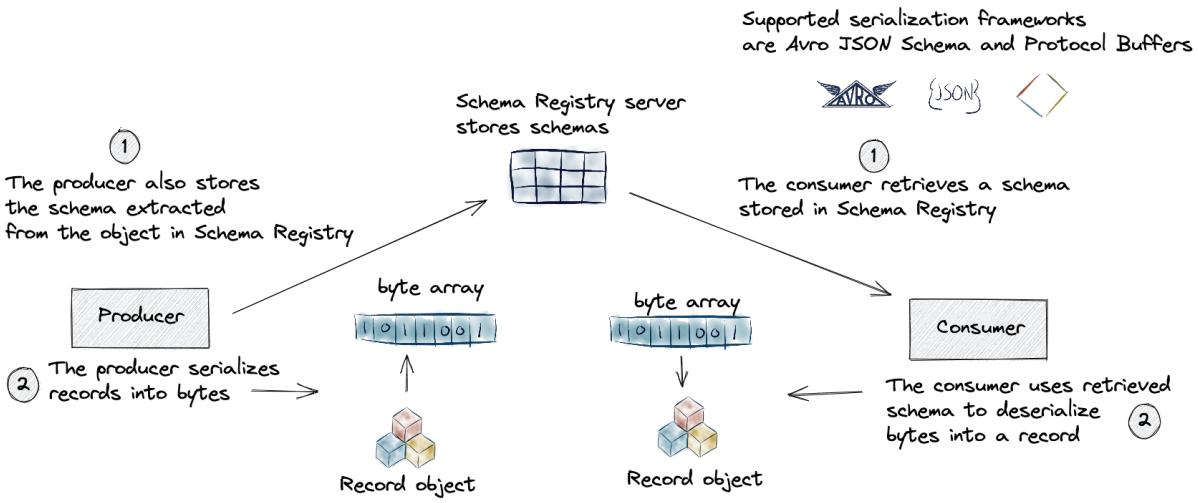

Figure 1.5 Schema registry enforces data modeling across the platform

Data governance is vital, to begin with, and its importance only increases as the size and diversity of an organization grows. Schema Registry stores schemas of the event records.

Schemas enforce a contract for data between producers and consumers. Schema Registry also provides serializers and deserializers supporting different tools that are Schema Registry aware. Providing (de)serializers means you don't have to write your serialization code. We'll cover Schema Registry in chapter 3.

1.2.3 Producer and consumer clients

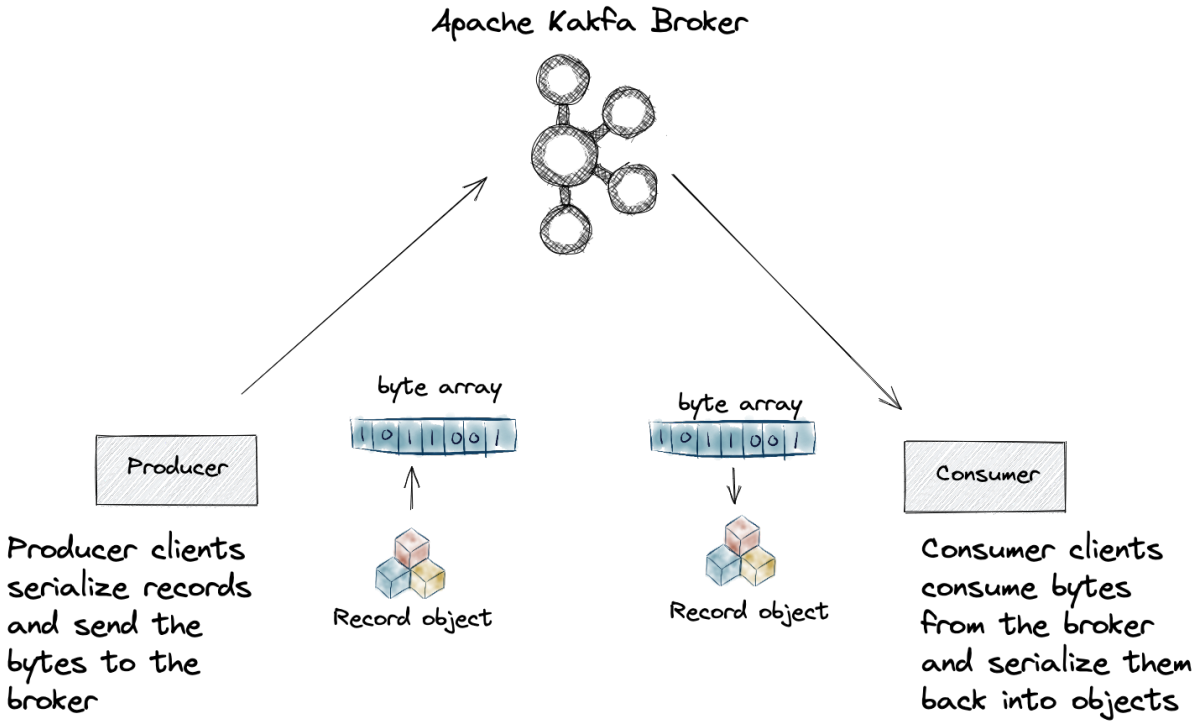

Figure 1.6 producers write records into Kafka, and consumers read records

The Producer client is responsible for sending records into Kafka. The consumer is responsible for reading records from Kafka. These two clients form the basic building blocks for creating an event-driven application and are agnostic to each other, allowing for greater scalability. The producer and consumer client also form the foundation for any higher-level abstraction working with Apache Kafka. We cover clients in chapter 4.

1.2.4 Kafka Connect

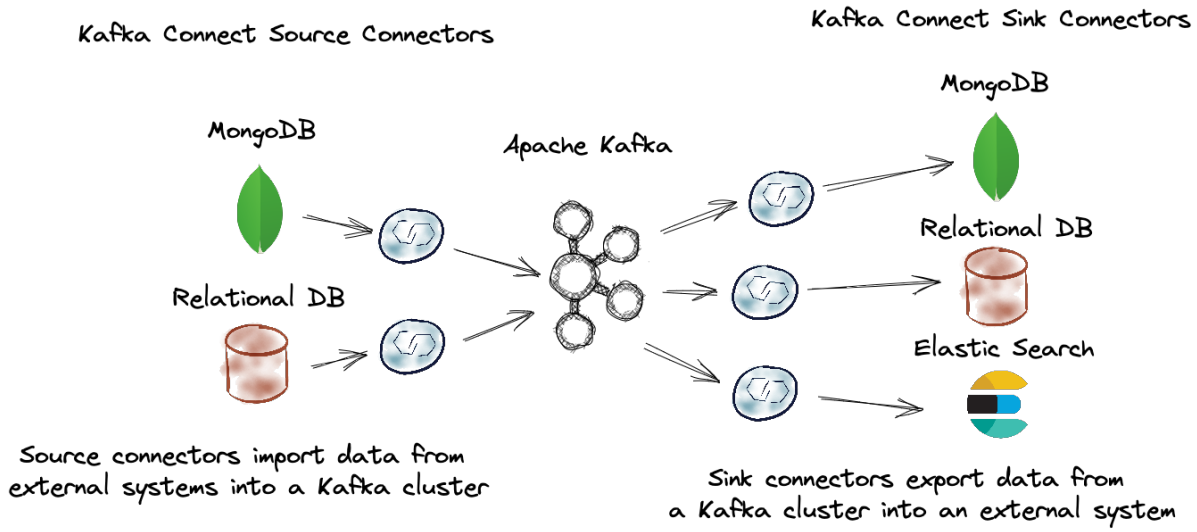

Figure 1.7 Kafka Connect bridges the gap between external systems and Apache Kafka

Kafka Connect provides an abstraction over the producer and consumer clients for importing data to and exporting data from Apache Kafka. Kafka connect is essential in connecting external data stores with Apache Kafka. It also provides an opportunity to perform light-weight transformations of data with Simple Messages Transforms when either exporting or importing data. We'll go into details of Kafka Connect in a later chapter.

1.2.5 Kafka Streams

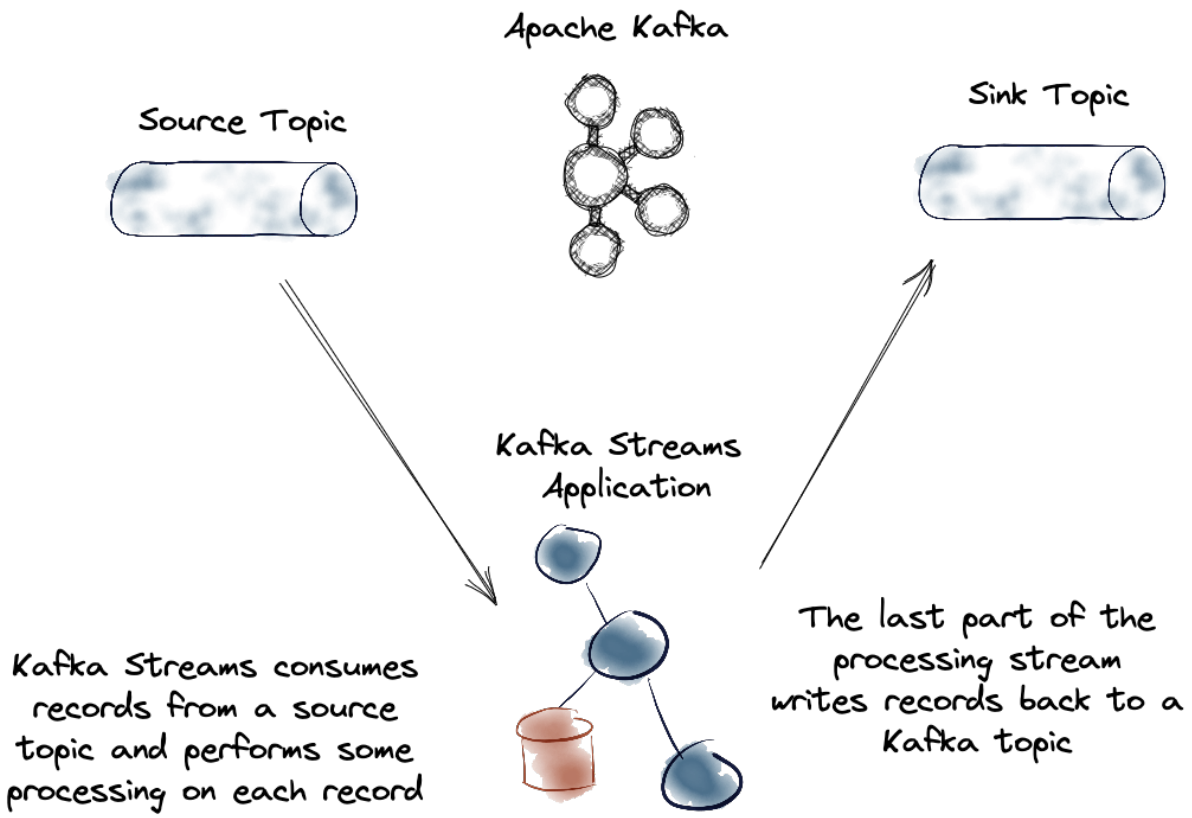

Figure 1.8 Kafka Streams is the stream processing API for Kafka

Kafka Streams is the native stream processing library for Kafka. Kafka Streams is written in the Java programming language and is used by client applications at the perimeter of a Kafka cluster; it is **not** run inside a Kafka broker. It provides support for performing operations on event data, including transformations, stateful operations like joins, and aggregations. Kafka Streams is where you'll do the heart of your work when dealing with events. Chapters 6, 7, and 8 cover Kafka Streams in detail.

1.2.6 ksqlDB

ksqlDB is an event streaming database. It does this by applying a SQL interface for event stream processing. Under the covers, ksqlDB uses Kafka Streams for performing its event streaming tasks. A key advantage of ksqlDB is that it allows you to specify your event streaming operation in SQL; no code is required. We'll discuss ksqlDB in chapters 8 and 9.

```
CREATE TABLE activePromotions AS
  SELECT rideId,
         qualifyPromotion(kmToDst) AS promotion
  FROM locations
  GROUP BY rideId
  EMIT CHANGES;


  SELECT rideId, promotion
  FROM activePromotions
  WHERE ROWKEY = '6fd0fcdb';
```

Figure 1.9 ksqlDB provides streaming database capabilities

Now that we've gone over how the Kafka event streaming platform works, including the individual components, let's apply a concrete example of a retail operation demonstrating how the Kafka event streaming platform works.

1.3 A concrete example of applying the Kafka event streaming platform

Let's say there is a consumer named Jane Doe, and she checks her email. There's one email from ZMart with a link to a page on the ZMart website containing coupons for 15% off the total purchase price. Once on the web page, Jane clicks another link to activate the coupons and print them out. While this whole sequence is just another online purchase for Jane, it represents clickstream events for ZMart.

Let's take a moment here to pause our scenario so we discuss the relationship between these simple events and how they interact with the Kafka event streaming platform.

The data generated by the initial clicks to navigate to and print the coupons create clickstream information captured and produced directly into Kafka with a producer microservice. The marketing department started a new campaign and wants to measure its effectiveness, so the clickstream events available at this point are valuable.

The first sign of a successful project is that users click on the email links to retrieve the coupons. Additionally, the data science group is interested in the pre-purchase clickstream data as well. The data science team can track customers' initial actions and later attribute purchases to those initial clicks and marketing campaigns. The amount of data from this single activity may seem

small. When you factor in a large customer base and several different marketing campaigns, you end up with a significant amount of data.

Now let's resume our shopping example.

It's late summer, and Jane has been meaning to get out shopping to get her children some back-to-school supplies. Since tonight is a rare night with no family activities, Jane decides to stop off at ZMart on her way home.

Walking through the store after grabbing everything she needs, Jane walks by the footwear section and notices some new designer shoes that would go great with her new suit. She realizes that's not what she came in for, but what the heck life is short (ZMart thrives on impulse purchases!), so Jane gets the shoes.

As Jane approaches the self-checkout aisle, she first scans her ZMart member card. After scanning all the items, she scans the coupon, which reduces the purchase by 15%. Then Jane pays for the transaction with her debit card, takes the receipt, and walks out of the store. A little later that evening, Jane checks her email, and there's a message from ZMart thanking her for her patronage, with coupons for discounts on a new line of designer clothes.

Let's dissect the purchase transaction and see this one event triggers a sequence of operations performed by the Kafka event streaming platform.

So now ZMart's sales data streams into Kafka. In this case, ZMart uses Kafka Connect to create a source connector to capture the sales as they occur and send them into Kafka. The sale transaction brings us to the first requirement, the protection of customer data. In this case, ZMart uses an SMT or Simple Message Transform to mask the credit card data as it goes into Kafka.

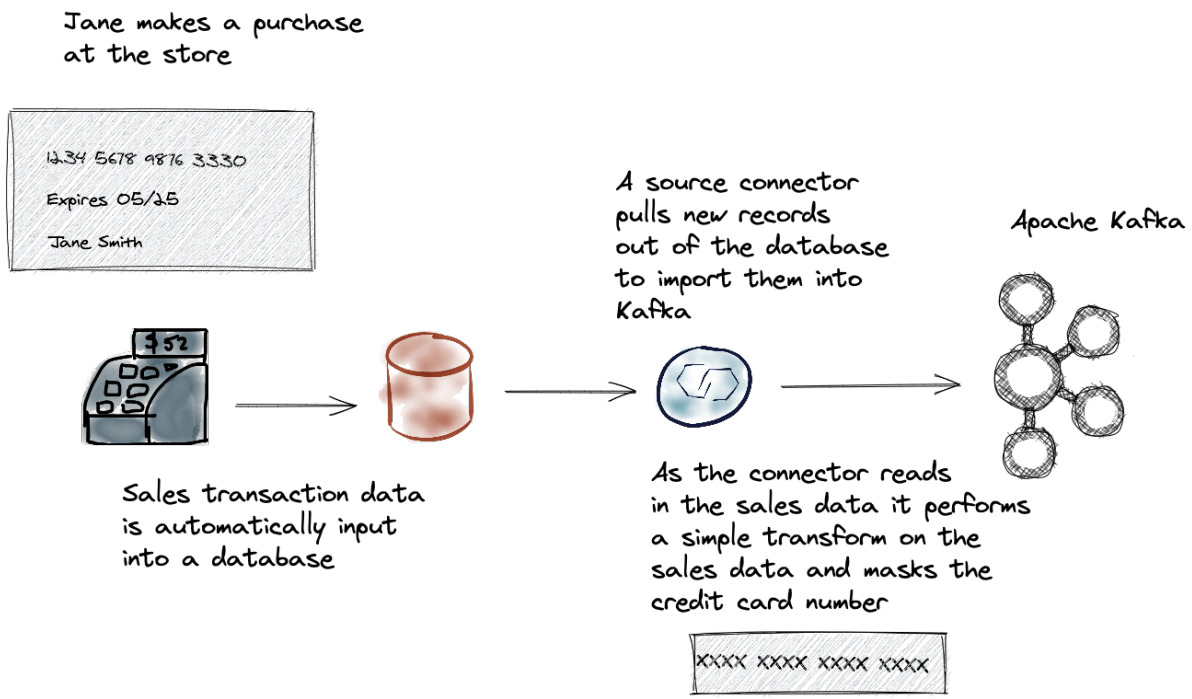

Figure 1.10 Sending all of the sales data directly into Kafka with connect masking the credit card numbers as part of the process

As connect writes records into Kafka, they are immediately consumed by different organizations within ZMart. The department in charge of promotions created an application for consuming sales data for assigning purchase rewards if they are a member of the loyalty club. If the customer reaches a threshold for earning a bonus, an email with a coupon goes out to the customer.

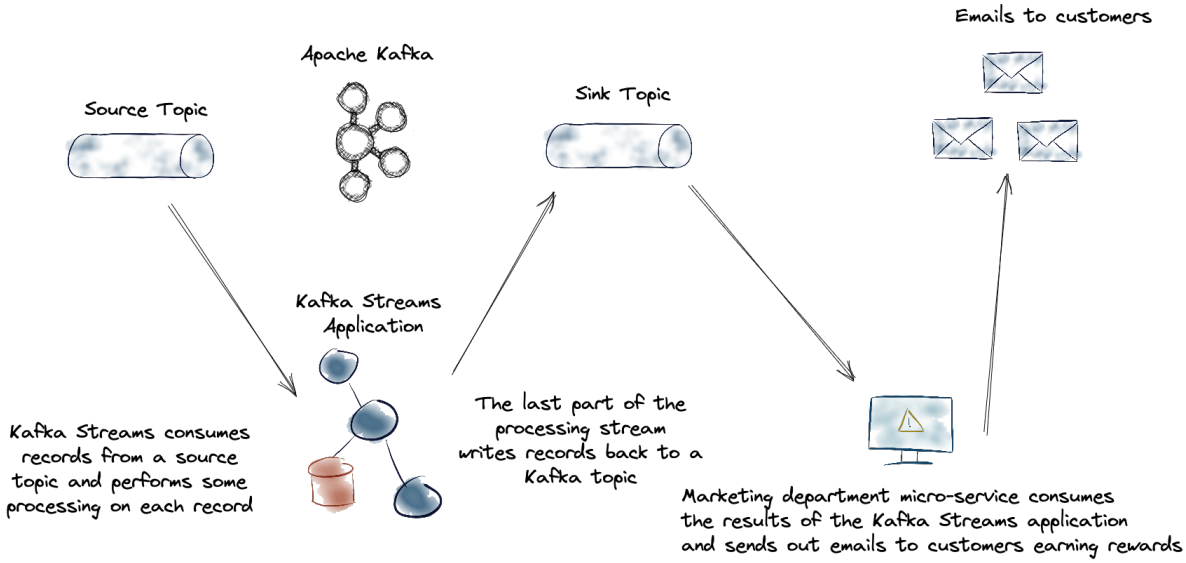

Figure 1.11 Marketing department application for processing customer points and sending out earned emails

It's important to note that ZMart processes sales records immediately after the sale. So customers

get timely emails with their rewards within a few minutes of completing their purchases. By acting on the purchase events as they happen allows ZMart a quick response time to offer customer bonuses.

The Data Science group within ZMart uses the sales data topic as well. The DS group uses a Kafka Streams application to process the sales data building up purchase patterns of what customers in different locations are purchasing the most. The Kafka Streams application crunches the data in real-time and sends the results out to a sales-trends topic.

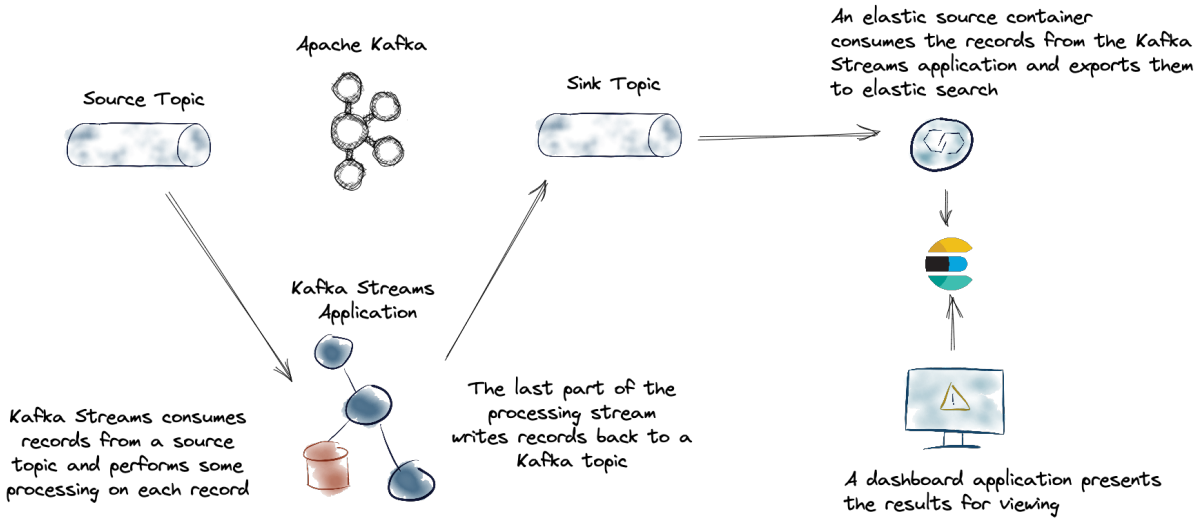

Figure 1.12 Kafka Streams application crunching sales data and connect exporting the data for a dashboard application

ZMart uses another Kafka connector to export the sales trends to an external application that publishes the results in a dashboard application. Another group also consumes from the sales topic to keep track of inventory and order new items if they drop below a given threshold, signaling the need to order more of that product.

At this point, you can see how ZMart leverages the Kafka platform. It is important to remember that with an event streaming approach, ZMart responds to data as it arrives, allowing them to make quick and efficient decisions immediately. Also, note how you write into Kafka *once*, yet multiple groups consume it at different times, independently in a way that one group's activity doesn't impede another's.

In this book, you'll learn what event-stream development is, why it's essential, and how to use the Kafka event streaming platform to build robust and responsive applications. From extract, transform, and load (ETL) applications to advanced stateful applications requiring complex transformations, we'll cover the Kafka streaming platform's components so you can solve the kinds of challenges presented earlier with an event-streaming approach. This book is suitable for any developer looking to get into building event streaming applications.

1.4 Summary

- Event streaming is capturing events generated from different sources like mobile devices, customer interaction with websites, online activity, shipment tracking, and business transactions. Event streaming is analogous to our nervous system.
- An event is "something that happens," and the ability to react immediately and review later is an essential concept of an event streaming platform
- Kafka acts as a central nervous system for your data and simplifies your event stream processing architecture
- The Kafka event streaming platform provides the core capabilities for you to implement your event streaming application from end-to-end by delivering the three main components of publish/consume, durable storage, and processing.
- Kafka broker are the storage layer and service requests from clients for writing and reading records. The brokers store records as bytes and do no touch or alter the contents.
- Schema Registry provides a way to ensure compatibility of records between producers and consumers.
- Producer clients write (produce) records to the broker. Consumer clients consume records from the broker. The producer and consumer clients are agnostic of each other. Additionally, the Kafka broker doesn’t have any knowledge of who the individual clients are, they just process the requests.
- Kafka Connect provides a mechanism for integrating existing systems such as external storage for getting data into and out of Kafka.
- Kafka Streams is the native stream processing library for Kafka. It runs at the perimeter of a Kafka cluster, not inside the brokers and provides support for transforming data including joins and stateful transformations.
- ksqlDB is an event streaming database for Kafka. It allows you to build powerful real-time systems with just a few lines of SQL.