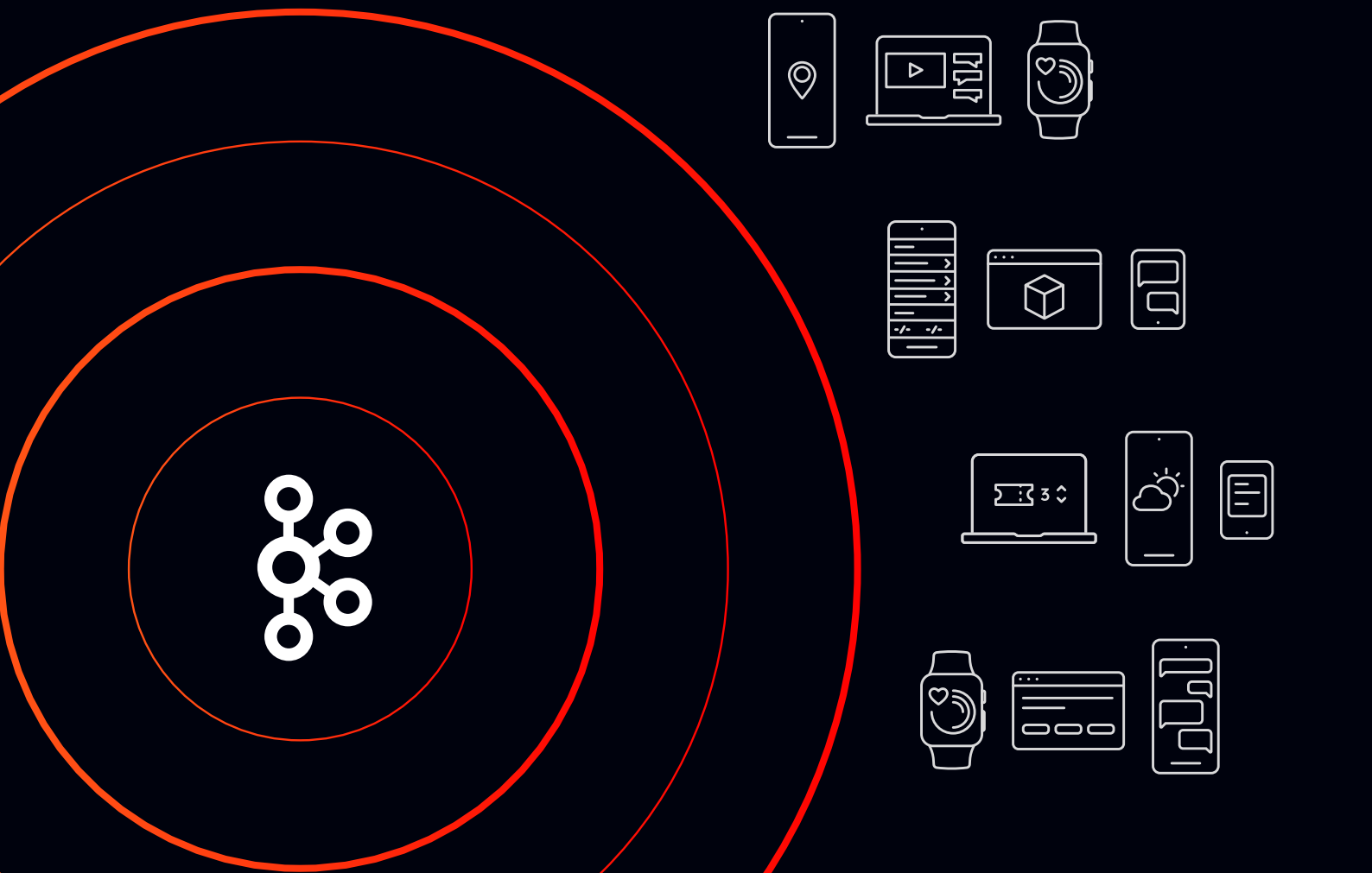
ably

WHITEPAPER

Connect Kafka to client devices at the edge

Ably's serverless WebSocket platform offers a simple and reliable way to stream Kafka events in realtime to millions of web, mobile, and IoT clients.



Contents

Connect Kafka to client devices at the edge

Realtime data and event-driven architectures are on the rise

Our everyday digital experiences are in the midst of a revolution. Users increasingly expect their online experiences to be interactive, responsive, immersive, and *realtime* by default.

> Estimates suggest that **30% of all global data** consumed by 2025 will result from information exchange in realtime.[1]

The need to satisfy user expectations is driving the adoption of *event-driven architectures (EDA)* in organizations of all shapes and sizes. Event-driven architecture is not a new concept. It's been around since the 1970s, has been extensively battle-tested since, and has been patiently waiting for its moment. What's new is the way in which EDA is taking center stage to solve modern business problems and help deliver realtime user experiences.

> According to a survey[2] (on how organizations use event-driven architectures), **85% of the 840 respondents** recognize the critical business value of adopting EDA. The main benefits of implementing EDA are:
>
> - Improving application responsiveness (46%)
> - Improving customer experiences (44%)
> - Responding to events and changes in realtime (43%)

[1] Source: https://go.ably.com/513

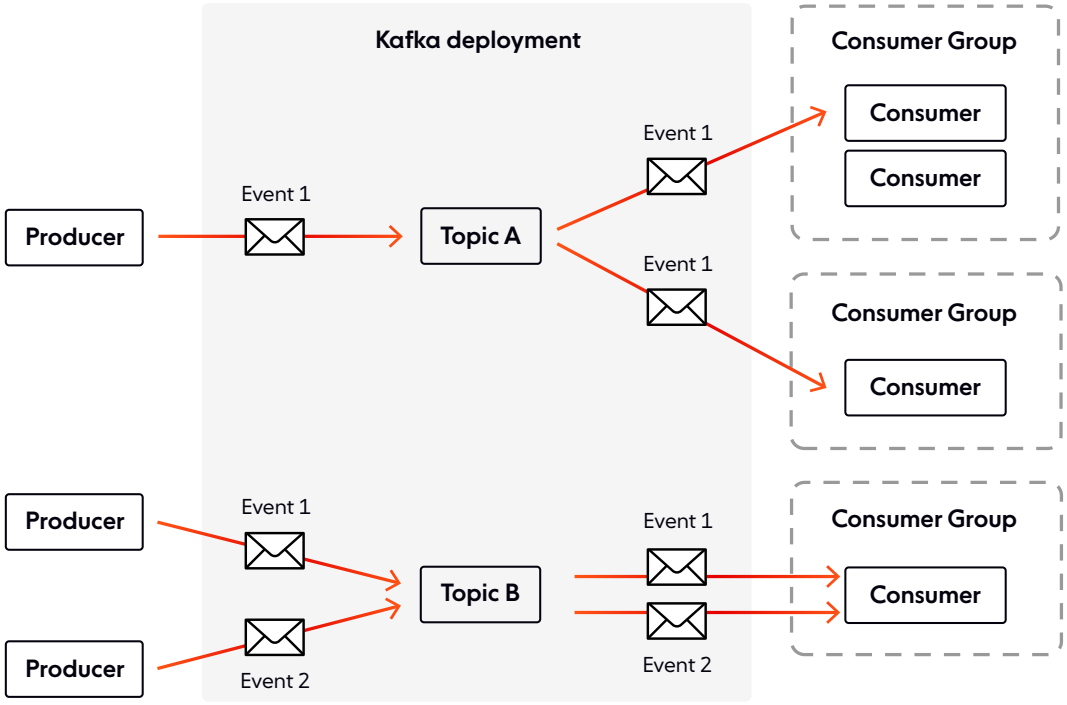
[2] Source: https://go.ably.com/js3

Where does Kafka fit in?

Created about a decade ago, Apache Kafka is a widely adopted event streaming solution. Kafka uses the pub/sub pattern and acts as a message broker to enable realtime communication between various backend components.

Here's a summary of Kafka's key concepts:

- **Events** are Kafka's smallest building blocks. An event records that something relevant has happened; for example, a user has made a payment.
- **Topics** are ordered sequences of events stored durably, for as long as needed. Each topic consists of multiple partitions. The benefit is that partitioning allows you to parallelize a topic by splitting its data across multiple Kafka brokers.
- **Producers and consumers.** Producers are services that publish (write) to Kafka topics, while consumers subscribe to Kafka topics to consume (read) events.



To enhance and complement its core event streaming capabilities, Kafka leverages a rich ecosystem, with additional components and APIs, like Kafka Streams, ksqlDB, and Kafka Connect.

Kafka Streams enables you to build realtime backend apps and microservices, where the input and output data are stored in Kafka clusters. Streams is used to process (group, aggregate, filter, and enrich) streams of data in realtime.

ksqlDB is a database designed specifically for stream processing apps. You can use ksqlDB to build event streaming applications from Kafka topics.

Kafka Connect is a tool designed for reliably moving large volumes of data between Kafka and other systems (such as Elasticsearch, Hadoop, or MongoDB, to name just a few). So-called “connectors” are used to transfer data in and out of Kafka. There are two types of connectors:

- **Sink connector.** Used for streaming data from Kafka topics into another system.
- **Source connector.** Used for ingesting data from another system into Kafka.

Kafka displays robust characteristics, that make it a strong choice for building reliable event-driven systems:

- Low latency
- Scalability & high throughput
- Guaranteed ordering and delivery
- Durability
- High availability and fault tolerance

Thousands of companies use Kafka for high-performance data pipelines, streaming analytics, and mission-critical applications. Here’s a non-exhaustive list of use cases where event-driven architectures powered by Kafka are a great fit:

- Banking and fintech (e.g., realtime payment processing and fraud detection).
- Asset tracking and logistics (monitoring cars, assets, and shipments in realtime)
- Collecting and immediately reacting to user actions (for example, pushing a personalized recommendation to a customer browsing an eCommerce website after they’ve added something to their cart).
- Continuously capturing and analyzing data from various sources, such as IoT devices.
- Live dashboards and realtime analytics.
- Processing and distributing sports and betting data in realtime.

Connecting Kafka to users at the network edge

For many use cases, Kafka is crucial to having an event-driven, time-ordered, highly available, and fault-tolerant dataspace.

A key thing to bear in mind is that **Kafka is designed and optimized for internal event streaming**, enabling streams of data to flow in realtime between microservices, databases, and other types of components within your backend ecosystem.

> Despite its strengths, it's important to highlight that **Kafka is not a proxy for millions of client devices**.
>
> So, the question is, **how do you connect your Kafka pipeline to end-users at the network edge** in realtime?
>
> **The solution is to use Kafka in combination with an intelligent internet-facing middleware (messaging layer) built specifically for data distribution to user devices.**

Ideally, this middleware should provide the same level of guarantees and display similar characteristics to Kafka; you don't want to degrade the overall dependability of your system by pairing Kafka with a less reliable internet-facing middleware.

What are the traits of a dependable internet-facing middleware?

A middleware that you can trust to reliably deliver Kafka events to end-users should exhibit the following properties:

- **Low latency**
  Essential for an optimum user experience, and critical for certain use cases, such as streaming live score updates. Just as Kafka streams events in your backend in milliseconds, your internet-facing middleware should allow you to send data to your users with consistently low latencies.

- **Message ordering and guaranteed delivery**
  For many use cases, it's unacceptable for messages to be sent out of order. Think, for example, of chat apps, and imagine how frustrating and confusing it can be if the replies are not delivered in the correct order. Additionally, guaranteed delivery (preferably exactly once) is critical for many use cases - for example, sending fraud alerts.

- **Fault tolerance**
  The assumption has to be that component failures will happen sooner or later. What's important is that when failures do occur, your system has enough redundancy to continue operating, with functionality and user experience preserved as effectively as possible.

- **Scalability**
  The public internet is a volatile and unpredictable source of traffic. You don't know how many users might connect to your system simultaneously - this can range between hundreds and millions. If your internet-facing middleware cannot sustain incoming connections and the fluctuating volume of traffic, system-breaking complications can arise, negatively affecting availability, uptime, and user experience. Your internet-facing middleware should be dynamically elastic so that it can quickly react and scale out to handle potential spikes in usage.

The challenges of building your own solution to connect Kafka to client devices

Building your own middleware to extend Kafka to client devices at the edge is often time-consuming, expensive, and complex. Here are the key challenges you have to face with a DIY solution:

- Large financial costs to build the infrastructure, and additional costs to maintain and improve it.
- Significant engineering effort to build and maintain, shifting focus away from product development, and increasing your time to market.
- Developing a system you can trust to deliver at scale is a formidable engineering challenge. Failing to do it right (e.g., sub-optimal latency or outages) will lead to user dissatisfaction.

Take, for example, the WebSocket technology. Due to its low latency, event-driven nature, maturity, and widespread adoption, WebSocket is a prime choice for extending your internal Kafka pipeline to client devices. However, engineering a reliable, scalable WebSocket middleware that's fit for the job is far from trivial.

A recent survey[3] highlights just how complex it is to build WebSocket infrastructure for last-mile delivery in-house:

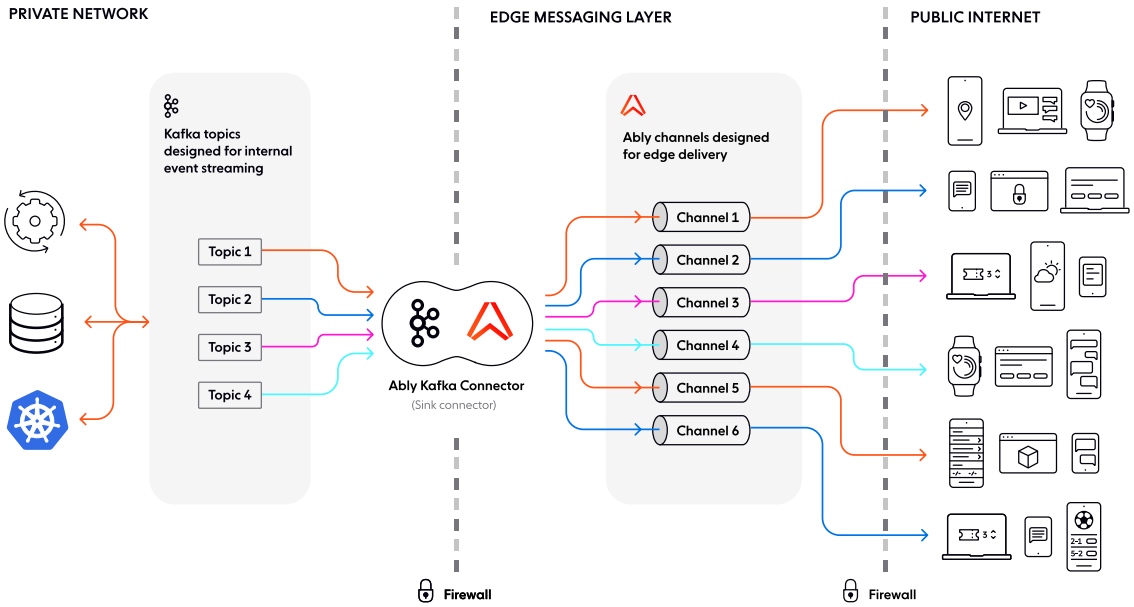
- **65%** of DIY solutions had an outage or significant downtime in the last 12-18 months.
- **10.2** person-months is the average time to build basic infrastructure, with limited scalability, in-house.
- Half of all self-built realtime data platforms require **$100K-$200K a year** in upkeep.

It's often more convenient and cost-effective to offload the complexity of connecting Kafka to end-users to a managed, specialized PaaS provider.

[3] Source: https://go.ably.com/2fg

Introducing Ably

Ably is a serverless WebSocket platform designed for realtime pub/sub messaging at the edge. Ably is often used as middleware for sending mission-critical Kafka data to millions of web, mobile, and IoT devices, via a fault-tolerant, autoscaling global edge network.



Connecting your Kafka deployment to Ably is straightforward. This is made possible with the help of the Ably Kafka Connector, a sink connector built on top of Kafka Connect.

The Ably Kafka Connector provides a ready-made integration that enables you to publish Kafka events into Ably channels with ease and speed. Events can then be distributed in realtime to millions of web, mobile, and IoT clients using Ably's WebSocket-based pub/sub messaging.

The connector can be self-hosted, or hosted with a third-party provider such as the Confluent Platform. The Ably Kafka Connector is a Verified Gold Connector on Confluent.

Going beyond the Connector (useful for egress scenarios), Ably makes it easy to stream data generated by end-users back into Kafka (ingress use cases). This is achieved using the Ably Kafka rule.

Take things to the next level with Kafka and Ably

Ably and Kafka are complementary solutions; they are both event-driven, and they share similar guarantees, messaging semantics, and properties. With Ably as a broker in the middle, you can seamlessly and scalably extend your Kafka event streaming pipeline to web, mobile, and IoT clients at the edge.

kafka	ably
Designed for internal event streaming. Enables event-driven communication between various backend components.	Designed for last-mile delivery. Powers live and collaborative experiences for end-users at the edge, primarily over WebSockets.
Works best with a low number of topics, and a defined, predictable number of producers and consumers.	Designed to work with an unknown and rapidly changing number of channels (Ably's equivalent of topics) and subscribers. Ably can quickly scale horizontally to handle millions of consumers.
Distributed pub/sub system: • High throughput • Sub-second latencies • Fault tolerance • Message ordering and delivery guarantees	Distributed pub/sub system: • High throughput & concurrency • < 65ms global median roundtrip latency • Regional & global fault tolerance • Guaranteed ordering and delivery even in unreliable network conditions

Explore Ably's features and capabilities →

The benefits of using Ably alongside Kafka

Your Kafka data is secure

By using Ably as your public internet-facing middleware, you decouple your backend Kafka deployment from the public internet, and protect the integrity of your event-driven pipeline at all times.

Client devices never connect directly to Kafka. Instead, they are only allowed to subscribe to the Ably channels for which they have permissions. In addition, Ably provides robust security mechanisms suitable for event streaming to public internet consumers:

- Message-level encryption.
- DoS protection and rate limiting.
- Flexible authentication (API keys and tokens) with fine-grained access control.
- SOC 2 Type 2, HIPAA, and EU GDPR compliance.

Reduced infrastructure costs and engineering complexity

Building a proprietary internet-facing middleware that can extend Kafka to user devices is time-consuming, risky, and involves significant DevOps, engineering, and financial resources.

As a fully-managed, serverless platform, Ably removes the burden of maintaining complex realtime infrastructure for last-mile delivery. You don't have to manually provision capacity or worry about scaling up and down to meet demand, and with our flexible pricing model, you will only pay for what you use.

Faster time to market

Since there's no realtime infrastructure for last-mile delivery to maintain, you are free to focus on building, improving, and releasing new products and features. Complementing Kafka's event streaming capabilities, Ably provides out-of-the-box features like presence, automatic reconnections, or push notifications, empowering you to quickly develop and refine live and collaborative experiences for your users.

Want to better understand how organizations of all shapes and sizes benefit from using Ably to build live and collaborative experiences?

Explore Ably's customer stories →

COMPANIES BUILDING ON ABLY'S SERVERLESS WEBSOCKET PLATFORM

HubSpot split EXPERITY Genius SPORTS Mentimeter tennis

Kafka + Ably system characteristics

An event-driven system built with Kafka and Ably displays the following characteristics:

Low latency

Kafka can stream and process events in milliseconds. Complementary, Ably streams events to client devices with sub-second latencies (<65 ms global median roundtrip latency), irrespective of where they are in the world. This is made possible by Ably's globally-distributed infrastructure.

Scalability

Kafka is known as a scalable solution, although scaling it yourself is not necessarily the easiest path. Things are made easier if you use a managed provider such as Confluent that provides elasticity and self-serve provisioning, allowing you to quickly and effortlessly scale your Kafka ecosystem as needed.

Similarly, Ably provides a fully managed layer between the backend and the frontend that can autoscale horizontally to handle up to millions of concurrently connected end-user devices.

Reliability and high availability

Kafka is designed with failure in mind (which is inevitable in distributed systems) and fail-over capabilities. Kafka achieves high availability by replicating the log for each topic's partitions across a configurable number of brokers. Note that replicas can live in different data centers, across different regions.

Ably provides fault tolerance at regional and global levels, so it can survive multiple failures without outages. Ably guarantees 99.999999% message survivability for instance and datacenter failures, and offers a 99.999% uptime SLA.

Together, Kafka and Ably create a distributed, robust, and trustworthy event-driven pipeline that is always dependably available to end-users.

Superior messaging QoS guarantees

Kafka can be configured to support ordering and exactly-once semantics, while Ably guarantees message ordering and exactly-once delivery, even in unreliable network conditions. Together, Kafka and Ably provide data integrity end-to-end: users always receive events in the correct order, without messages being lost or delivered multiple times.

Interoperability

There are numerous Kafka sink and source connectors that allow you to efficiently move large volumes of data between Kafka and other systems, such as Elasticsearch, Hadoop, or MongoDB, to name just a few.

Ably offers:

- 25+ client SDKs targeting every major programming language.
- Multi-protocol capabilities (WebSockets, MQTT, Server-Sent Events, and more).
- Managed integrations with numerous systems, such as serverless function providers.

Combined, Kafka and Ably provide a highly interoperable solution, making it easy to add additional components to your architecture, and connect different backend systems with various client devices in realtime.

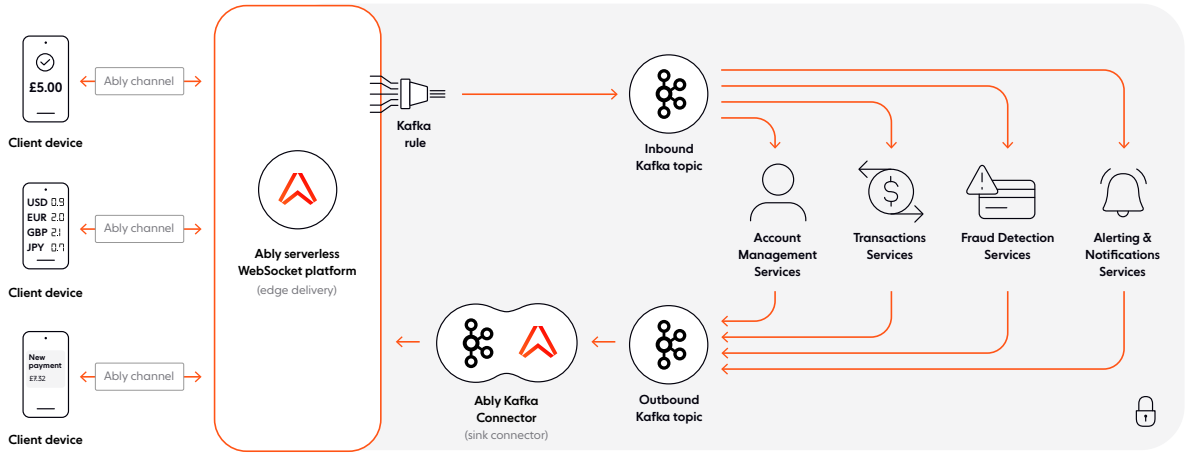
Kafka + Ably use cases

We will now look at some examples to better understand how Kafka and Ably can be combined for various use cases - banking apps, eCommerce, live dashboards, and more.

Note that in addition to the use cases covered in the following pages, you can use Kafka + Ably for practically any scenario where time-sensitive data needs to be processed and must flow between the data center and client devices at the network edge in (milli)seconds.

Banking

A banking ecosystem powered by Kafka and Ably might look something like this:



There are only two Kafka topics - one for inbound data, and another one for outbound data. Events are consumed from the inbound Kafka topic by various backend services (e.g., fraud detection) for processing. Whenever new data is available, these services write it to the outbound Kafka topic.

Ably acts as an internet-facing message broker, decoupling components, and intermediating the data flow between your banking app users and your backend Kafka pipeline. Events from the outbound Kafka topic (e.g., fraud alert events) can be easily transferred to Ably via the Ably Kafka Connector. Ably then distributes the events to relevant users over WebSocket-based pub/sub channels (you can also use push notifications for iOS and Android users).

With the help of the Kafka rule, actions performed by users (such as a payment confirmation) can be streamed from Ably into the inbound Kafka topic (and from there, to various services for processing).

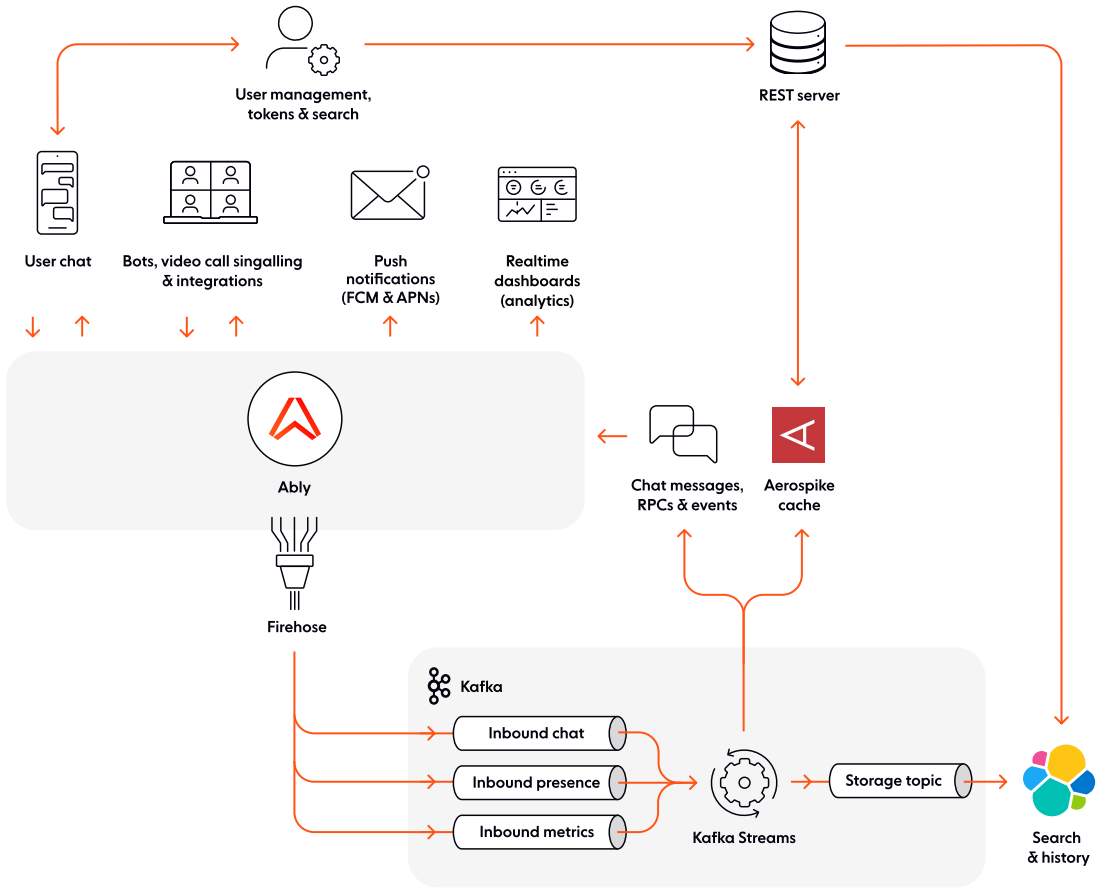
You can use Kafka and Ably for the following banking use cases:

- Sending in-app notifications.
- Realtime payments.
- In-app chat.
- Realtime analytics dashboards.
- Realtime and account balance updates.

Learn more about Kafka + Ably for banking →

Chat

A high-level chat architecture with Kafka and Ably could look like this:



Ably powers 1:1 and group chat for end-users - this is done over Ably pub/sub channels. Ably uses WebSockets under the hood, which means that a user can send and receive chat messages over the same connection, rather than having to open multiple connections.

Ably guarantees ordering, delivery, and exactly-once semantics, even if brief disconnections are involved (for example, a user switches from a mobile network to Wi-Fi); that's because our client SDKs automatically re-establish connections and ensure stream continuity.

Chat data sent to Ably can be streamed into Kafka for processing, by making use of the Kafka rule. Ably can collect and publish different types of data into Kafka, from chat messages to presence events (such as someone joining a chat room). Kafka Streams workers are used to process data from the inbound Kafka topics.

Once processed, chat messages and presence events are sent to Ably pub/sub channels (via the Ably Kafka Connector) for distribution to end-user devices. There is an Aerospike cache used to store user account data (such as user profiles, passwords, etc.). User search and message history are written by Kafka Streams workers to a storage topic, and then sent to Elasticsearch. End-users perform account-specific operations (such as changing their passwords), and retrieve search and message history through a REST API.

Going beyond live chat, you can use Ably to send push notifications to offline chat users.

If you're planning to add video support to your chat app, you'll most likely have to use something like WebRTC, which allows for peer-to-peer communication. You'd still need servers with WebRTC, so peers can exchange metadata to coordinate communication through a process called signaling. The WebRTC API itself doesn't offer a signaling mechanism, but you can use Ably to quickly implement dependable signaling mechanisms for WebRTC apps.

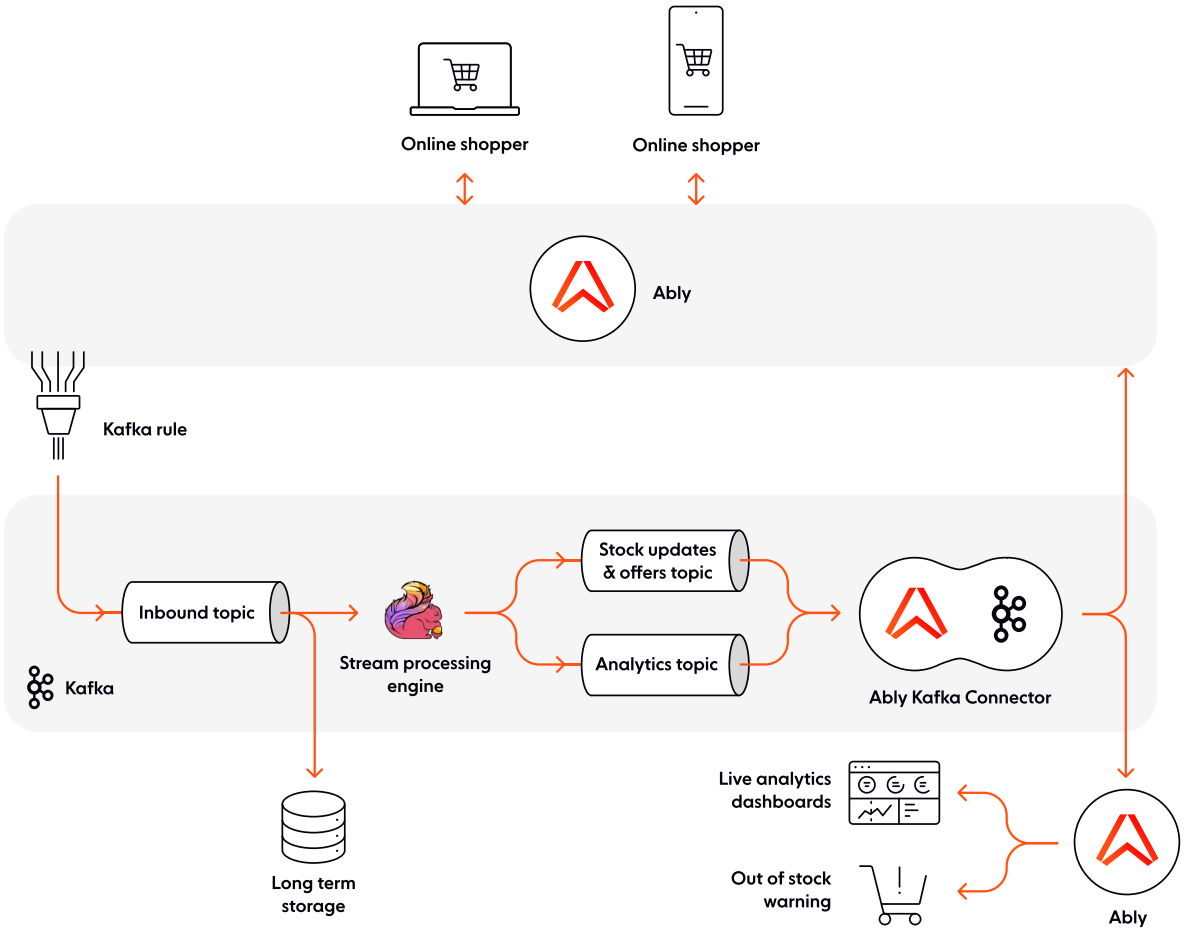
You can use Kafka and Ably for the following chat use cases:

- Chatbots, 1:1, and group chat, with message history, presence, emoji reactions, typing indicators, and more.
- Sending push notifications to offline chat users.
- Collecting and streaming chat (meta)data into the Kafka pipeline.

Learn more about Kafka + Ably for chat →

eCommerce

Here's how the architecture of an eCommerce system built with Kafka and Ably might look like:



Whenever a user buys something or adds an item to their cart, an event (message) is sent to Ably via WebSocket-based pub/sub channels. This stream of events (from all users) is then sent from Ably to an inbound Kafka topic; this is achieved with the help of the Kafka rule.

A stream processing engine then consumes data from the inbound Kafka topic; in our example, we've used Apache Flink, a scalable, high-performance solution capable of processing streams of events with low latencies, to extract analytics and insights from raw data. We've included Flink to make things more relatable, and because it integrates well with Kafka. However, its use is indicative, not prescriptive. There are other solutions you could use instead of Flink (depending on the specifics of your use case), such as Apache Pinot, ksqlDB, or even a microservice you build yourself.

Once Flink processes the stream of events, the output is written to two different Kafka topics: one is used for stock updates & personalized offers, and the other for internal analytics. Data from both these topics is synced to Ably, with the help of the Ably Kafka Connector. Ably then routes data to end-user devices over pub/sub channels.

While there is a single topic to handle stock updates and offers for all users, you can flexibly shard and route this data to client devices once it's moved from Kafka into Ably. For example, all users of the eCommerce platform could be subscribed to a "stock updates" Ably channel, and receive realtime updates about the availability of any given product at any point in time. More than that, you can have an Ably channel for each client device, so you can push personalized offers, depending on each user's activity and shopping habits.

With Ably's help, data from the analytics topic in Kafka can help the eCommerce provider analyze, understand, and improve business efficiency. For example, you can have live BI dashboards to keep track of things like the number of apples (or any other item) sold, or the number of sales between 1 pm and 2 pm. Or you can receive warnings when you are running low on an item, so you can restock.

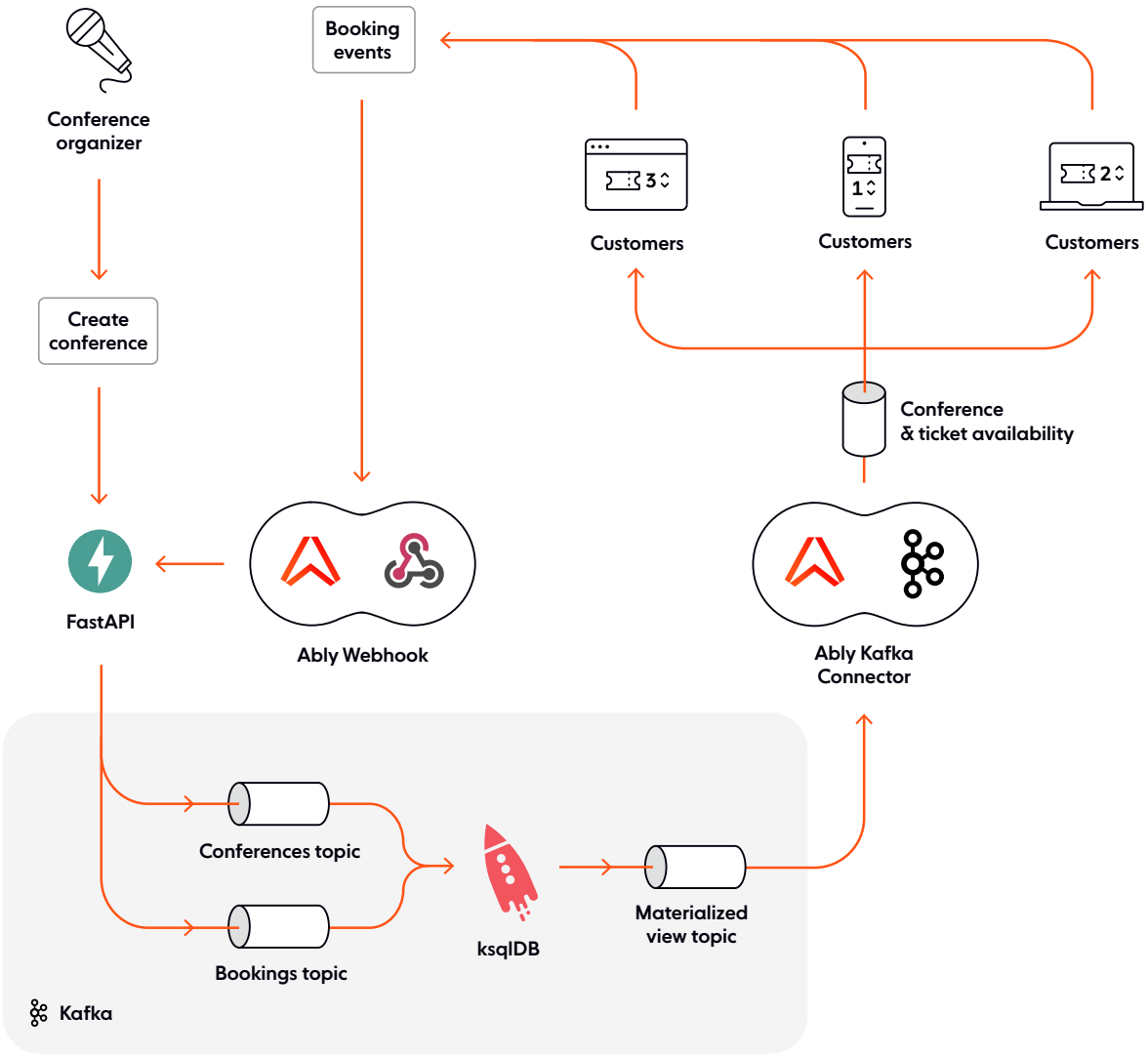
You can use Kafka and Ably for the following eCommerce use cases:

- Collecting and sending user activity (e.g., someone adding an item to their cart) into Kafka for processing.
- Pushing stock updates and personalized offers to users.
- Live BI dashboards for your internal teams to keep track of relevant metrics (e.g., remaining stock).
- Out-of-stock warnings and push notifications.
- In-app chat.

Learn more about Kafka + Ably for eCommerce →

Ticket booking

This diagram presents the high-level architecture of a realtime ticket booking solution built with Kafka and Ably:



Now, let's dive into details and see how the system works end-to-end. Whenever a new conference is planned, the organizer sends a "Create conference" API request to the frontend API component (FastAPI).

Next, the frontend API publishes the conference-related data as a record in the "Conferences" topic in Kafka. Records stored in the "Conferences" topic are then processed by ksqlDB, the stream processing component in our architecture. The output is written to the "Materialized view" Kafka topic.

Kafka data from the "Materialized view" topic is sent to the "Conference & ticket availability" Ably channel, via the Ably Kafka Connector. Ably then broadcasts the data in realtime to all client devices subscribed to the "Conference & ticket availability" channel.

Whenever a user books a ticket for a conference, Ably sends a webhook to the FastAPI component. FastAPI publishes the booking-related data to Kafka. However, this time, data is written to the "Bookings" topic instead of the "Conferences" topic. Each time a new record is written to the "Bookings" or "Conferences" topics, ksqlDB merges the changes into a unified view, the "Materialized view" topic - which reflects the latest state. Note that the latest state is always delivered by Ably to end-users with sub-second latencies.

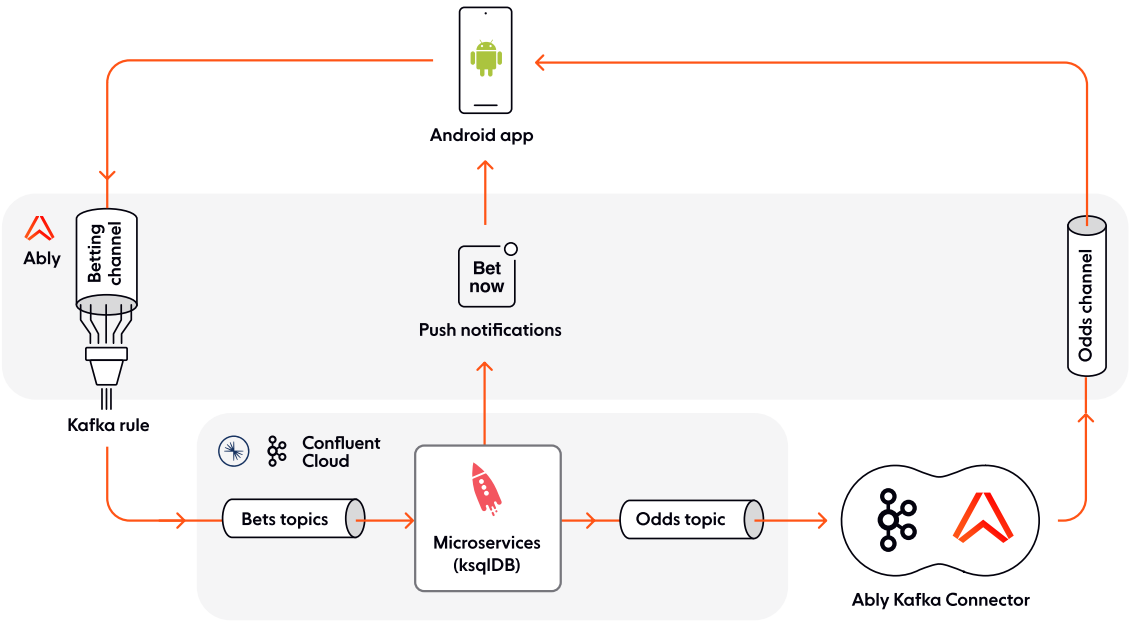
This architecture ensures that:

- Users have an accurate, always up-to-date view of all upcoming conferences, together with the number of available tickets for each conference.
- Data travels in realtime, with ordering and exactly-once delivery ensured by both Ably and Kafka.
- The system can scale to handle a high and rapidly changing number of concurrent users, due to Kafka and Ably both being highly scalable solutions.

Learn how to build a ticket booking system with Kafka and Ably →

Live score updates and betting

To demonstrate how Confluent Cloud and Ably work together when engineering realtime betting functionality, we've built a demo app with the following architecture:



Whenever new odds are available, they are sent from Confluent Cloud to Ably via the Ably Kafka Connector. Ably then distributes the odds in realtime to any number of Android app users over pub/sub channels.

In the Android app, users can see odds changing in realtime, and they can place their bets. Since Ably uses event-driven, bidirectional WebSockets as the primary transport protocol, a user not only receives odds, but they are also able to quickly place their bets over the same connection.

Ably streams the firehose of bets made into Confluent Cloud (through the Kafka rule). ksqlDB is then used to create a materialized view of the current state of the order book, and to push updated odds into the Odds topic.

From here, the process we've covered so far repeats - there's a continuous loop of data being processed and traveling in realtime between the backend and the Android app, with Ably sitting as an edge messaging broker between Kafka and end-users.

We've also built a Python microservice that queries ksqlDB, the purpose being to send personalized notifications to users, making them aware of various events: a win, a loss, or a new race starting soon. Note that these updates are delivered to Android users with the help of Ably's push notifications functionality.

Although the example we've provided focuses on betting, you can easily extend it to include additional functionality:

- Live score updates and activity feeds.
- In-app chat.
- Other related functionality, such as payment processing and fraud detection.

Learn how to build a betting app with Confluent and Ably →

Live dashboards

Experity provides technology solutions for the healthcare industry. One of its core products is a BI dashboard that enables urgent care providers to drive efficiency and enhance patient care in realtime. The data behind Experity's dashboard is drawn from multiple sources and processed in Kafka.

Experity decided to use Ably as their Internet-facing middleware because our platform works seamlessly with Kafka to stream mission-critical and time-sensitive realtime data to end-user devices. Ably extends and enhances Kafka's guarantees around speed, reliability, integrity, and performance. Furthermore, Ably frees Experity from managing complex realtime infrastructure designed for last-mile delivery. This saves Experity hundreds of hours of development time and enables the organization to channel its resources and focus on building its core offerings.

> "Ably is awesome. It was a life-saver for me. Not only was it the only HIPAA-compliant realtime solution capable of integrating with Kafka streams and giving us the performance guarantees we need, but the set-up was just incredibly straightforward. It instantly transformed the value of the dashboard for our customers.
>
> 
> **Andrew Hanisch**
> System Architect, Experity
>
> EXPERITY

Learn how Experity benefits from using Kafka + Ably →

Get started with Kafka and Ably today

Ready to maximize the value of your Kafka pipeline by extending it to millions of client devices in realtime?

Sign up for a free Ably account

Then read our documentation to seamlessly integrate your Kafka deployment with Ably:

- The Ably Kafka Connector
- The Kafka rule

If you have questions or want to find out more about the benefits of pairing Kafka and Ably, reach out.

> Ably is one of the key technologies that underpins our business. Its realtime platform and infrastructure layer seamlessly supports HubSpot's entire realtime needs, helping us to meet our technical, business, and product development requirements. With Ably, we have innovated and taken new products to market much faster while significantly reducing our operational engineering overhead.

Max Freiert
Product Group Lead, Hubspot

HubSpot

Get in touch

Visit
ably.com

Call
+44 20 3318 4689 (UK)
+1 877 434 5287 (USA)

Email
hello@ably.com