APACHE KAFKA

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

10 OUT OF 10

MANUFACTURING

7 OUT OF 10

BANKS

10 OUT OF 10

INSURANCE

8 OUT OF 10

TELECOM

CORE CAPABILITIES

HIGH THROUGHPUT

Deliver messages at network limited throughput using a cluster of machines with latencies as low as 2ms.

SCALABLE

Scale production clusters up to a thousand brokers, trillions of messages per day, petabytes of data, hundreds of thousands of partitions. Elastically expand and contract storage and processing.

PERMANENT STORAGE

Store streams of data safely in a distributed, durable, fault-tolerant cluster.

HIGH AVAILABILITY

Stretch clusters efficiently over availability zones or connect separate clusters across geographic regions.

Dictionary

Definitions from Oxford Languages · Learn more

throughput

/ˈθruːpʊt/

noun

the amount of material or items passing through a system or process.
"fast data throughput"

What is event streaming?

- Event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events
- storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time and routing the event streams to different destination technologies as needed.
- Event streaming thus ensures a continuous flow and interpretation of data so that the right information is at the right place, at the right time.

What can I use event streaming for ?

- To process payments and financial transactions in real-time, such as in stock exchanges, banks, and insurances.
- To track and monitor cars, trucks, fleets, and shipments in real-time, such as in logistics and the automotive industry.
- To continuously capture and analyze sensor data from IoT devices or other equipment, such as in factories and wind parks.
- To collect and immediately react to customer interactions and orders, such as in retail, the hotel and travel industry, and mobile applications.
- To monitor patients in hospital care and predict changes in condition to ensure timely treatment in emergencies.
- To connect, store, and make available data produced by different divisions of a company.
- To serve as the foundation for data platforms, event-driven architectures, and microservices.

Apache Kafka® is an event streaming platform. What does that mean?

Kafka combines three key capabilities so you can implement your use cases for event streaming end-to-end with a single battle-tested solution:

1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
2. To **store** streams of events durably and reliably for as long as you want.
3. To **process** streams of events as they occur or retrospectively.

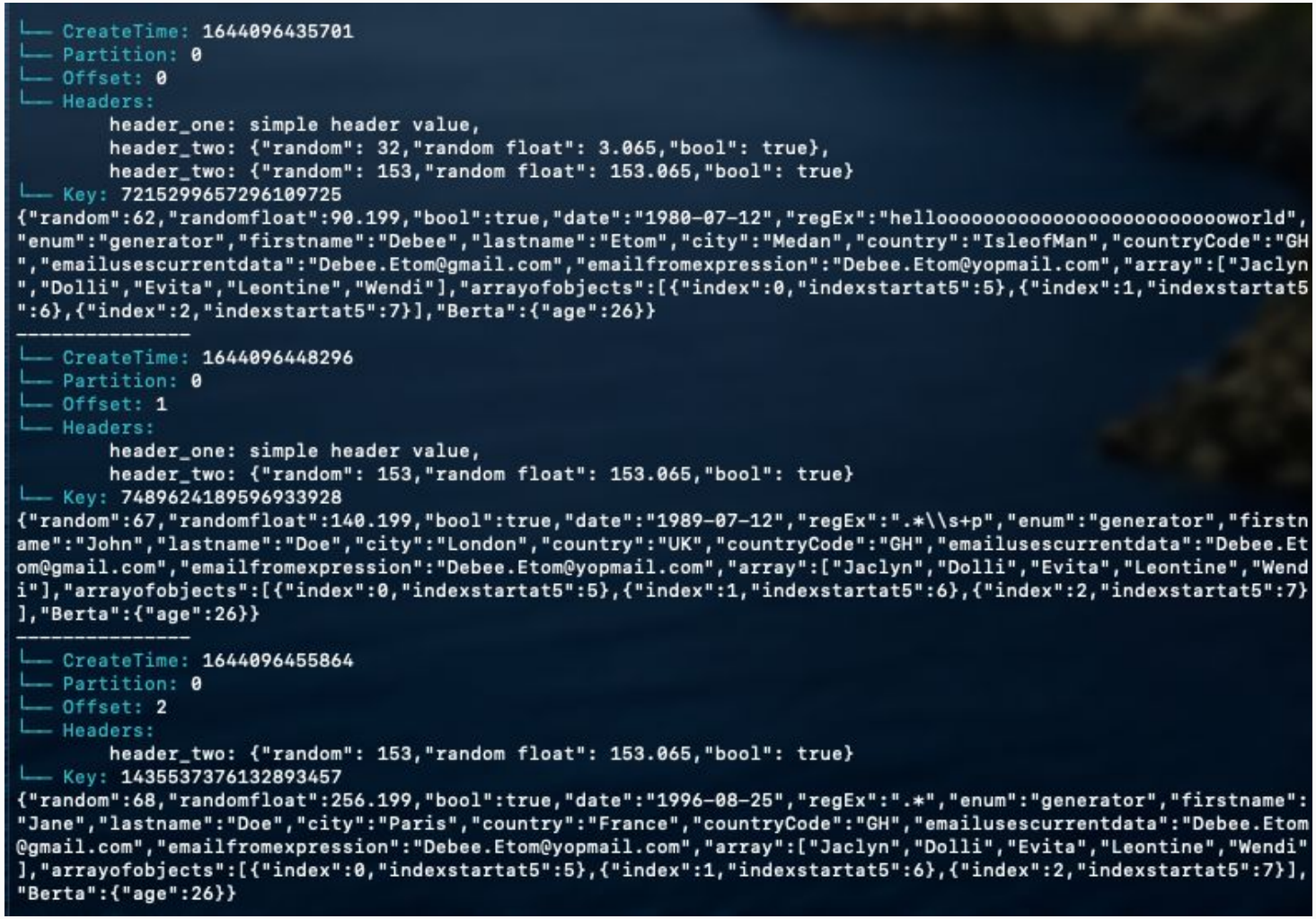
And all this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner. Kafka can be deployed on bare-metal hardware, virtual machines, and containers, and on-premises as well as in the cloud. You can choose between self-managing your Kafka environments and using fully managed services offered by a variety of vendors.

Main Concepts and Terminology

An event records the fact that "something happened" in the world or in your business. It is also called record or message. When you read or write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers. Here's an example event:

- Event key: "Alice"
- Event value: "Made a payment of $200 to Bob"
- Event timestamp: "Jun. 25, 2020 at 2:06 p.m."

```
Antonello.Benedetto@C02FN2XVMD6R kafka_demo % python kafka_producer.py
Kafka Producer has been initiated...
Produced message on topic user-tracker with value of {"user_id": 35814, "user_name": "Alex Juarez", "user_address": "886 Da
wn Vista | Gonzalezside | PK", "platform": "Laptop", "signup_at": "2022-07-15 20:41:44"}

Produced message on topic user-tracker with value of {"user_id": 72566, "user_name": "Matthew Robinson", "user_address": "7
0837 Michael Ridges Apt. 373 | Richardland | US", "platform": "Tablet", "signup_at": "2022-07-15 08:39:29"}

Produced message on topic user-tracker with value of {"user_id": 24346, "user_name": "Zachary Roy", "user_address": "8599 A
ngela Valley | Lake Michele | ET", "platform": "Laptop", "signup_at": "2022-07-20 12:13:15"}

Produced message on topic user-tracker with value of {"user_id": 62333, "user_name": "Ashley Nelson", "user_address": "1776
 Pamela Shore | New Katherine | KZ", "platform": "Tablet", "signup_at": "2022-07-05 17:44:49"}

Produced message on topic user-tracker with value of {"user_id": 84126, "user_name": "Tammy Mcdowell", "user_address": "732
 Tucker Track | Franklinport | CR", "platform": "Laptop", "signup_at": "2022-07-08 03:07:53"}

Produced message on topic user-tracker with value of {"user_id": 27146, "user_name": "Eric Underwood", "user_address": "349
6 Johnson Harbors | Camposton | GQ", "platform": "Tablet", "signup_at": "2022-07-23 05:36:58"}

Produced message on topic user-tracker with value of {"user_id": 49109, "user_name": "Brooke Park", "user_address": "866 Ke
vin Hollow Suite 150 | Leemouth | EG", "platform": "Tablet", "signup_at": "2022-07-06 21:36:28"}

Produced message on topic user-tracker with value of {"user_id": 85926, "user_name": "Maria Mathis", "user_address": "63809
 Mary Brook | Fletcherhaven | KW", "platform": "Mobile", "signup_at": "2022-07-07 02:39:21"}
```


└── CreateTime: 1644096435701
└── Partition: 0
└── Offset: 0
└── Headers:
header_one: simple header value,
header_two: {"random": 32,"random float": 3.065,"bool": true},
header_two: {"random": 153,"random float": 153.065,"bool": true}
└── Key: 7215299657296109725
{"random":62,"randomfloat":90.199,"bool":true,"date":"1980-07-12","regEx":"hellooooooooooooooooooooooooooworld",
"enum":"generator","firstname":"Debee","lastname":"Etom","city":"Medan","country":"IsleofMan","countryCode":"GH
","emailusescurrentdata":"Debee.Etom@gmail.com","emailfromexpression":"Debee.Etom@yopmail.com","array":["Jaclyn
","Dolli","Evita","Leontine","Wendi"],"arrayofobjects":[{"index":0,"indexstartat5":5},{"index":1,"indexstartat5
":6},{"index":2,"indexstartat5":7}],"Berta":{"age":26}}

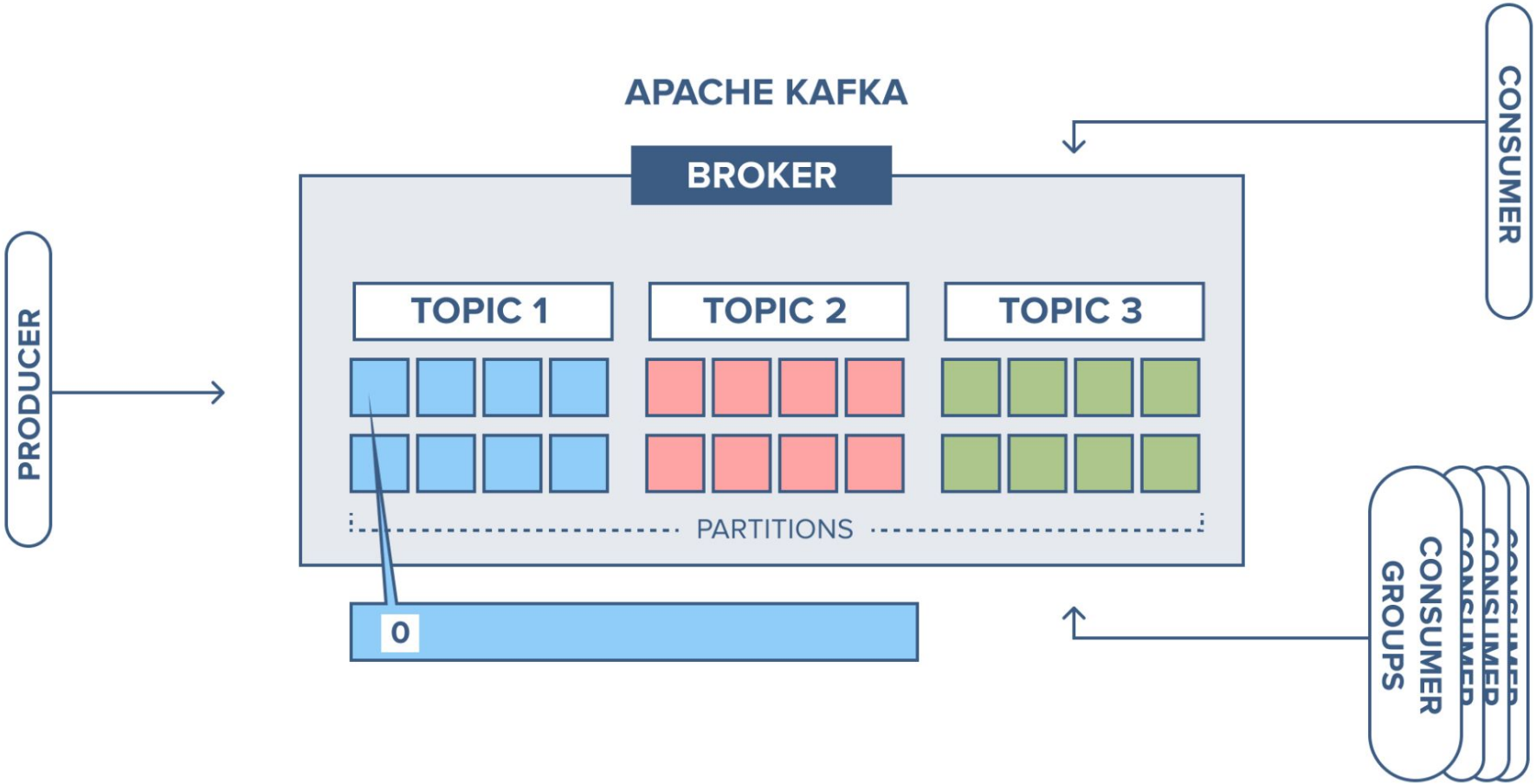
└── CreateTime: 1644096448296
└── Partition: 0
└── Offset: 1
└── Headers:
header_one: simple header value,
header_two: {"random": 153,"random float": 153.065,"bool": true}
└── Key: 7489624189596933928
{"random":67,"randomfloat":140.199,"bool":true,"date":"1989-07-12","regEx":".*\\s+p","enum":"generator","firstn
ame":"John","lastname":"Doe","city":"London","country":"UK","countryCode":"GH","emailusescurrentdata":"Debee.Et
om@gmail.com","emailfromexpression":"Debee.Etom@yopmail.com","array":["Jaclyn","Dolli","Evita","Leontine","Wend
i"],"arrayofobjects":[{"index":0,"indexstartat5":5},{"index":1,"indexstartat5":6},{"index":2,"indexstartat5":7}
],"Berta":{"age":26}}

└── CreateTime: 1644096455864
└── Partition: 0
└── Offset: 2
└── Headers:
header_two: {"random": 153,"random float": 153.065,"bool": true}
└── Key: 1435537376132893457
{"random":68,"randomfloat":256.199,"bool":true,"date":"1996-08-25","regEx":".*","enum":"generator","firstname":
"Jane","lastname":"Doe","city":"Paris","country":"France","countryCode":"GH","emailusescurrentdata":"Debee.Etom
@gmail.com","emailfromexpression":"Debee.Etom@yopmail.com","array":["Jaclyn","Dolli","Evita","Leontine","Wendi"
],"arrayofobjects":[{"index":0,"indexstartat5":5},{"index":1,"indexstartat5":6},{"index":2,"indexstartat5":7}],
"Berta":{"age":26}}

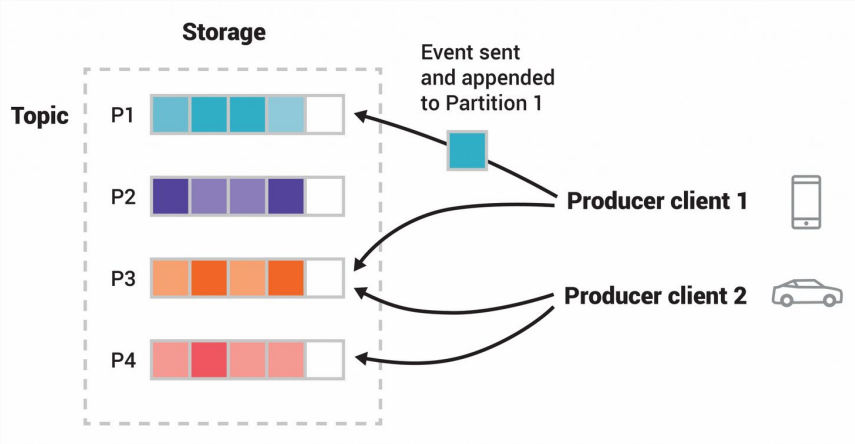
- Events are organized and durably stored in topics.

- Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder.

- An example topic name could be "payments". Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Events in a topic can be read as often as needed—unlike traditional messaging systems, events are not deleted after consumption. Instead, you define for how long Kafka should retain your events through a per-topic configuration setting, after which old events will be discarded. Kafka's performance is effectively constant with respect to data size, so storing data for a long time is perfectly fine.


APACHE KAFKA
BROKER
TOPIC 1
TOPIC 2
TOPIC 3
PARTITIONS
0
PRODUCER
CONSUMER
CONSUMER GROUPS

- Topics are partitioned, meaning a topic is spread over a number of "buckets" located on different Kafka brokers.
- This distributed placement of your data is very important for scalability because it allows client applications to both read and write the data from/to many brokers at the same time.
- When a new event is published to a topic, it is actually appended to one of the topic's partitions.
- Events with the same event key (e.g., a customer or vehicle ID) are written to the same partition, and Kafka guarantees that any consumer of a given topic-partition will always read that partition's events in exactly the same order as they were written.



- To make your data fault-tolerant and highly-available, every topic can be replicated, even across geo-regions or data centers, so that there are always multiple brokers that have a copy of the data just in case things go wrong, you want to do maintenance on the brokers, and so on.
- A common production setting is a replication factor of 3, i.e., there will always be three copies of your data. This replication is performed at the level of topic-partitions.

Order Placed: This topic could capture events when a customer places an order. It would include details about the order, such as the items ordered, quantities, customer information, and delivery preferences.

Order Status Updates: This topic could be used to track the status of orders in real time. It would include events such as order confirmed, food preparation started, food ready for pickup/delivery, and order delivered.

Location Tracking: For delivery tracking, a topic might be used to capture the real-time location updates of delivery drivers. This information would enable accurate tracking and estimated arrival times for customers.

Promotions and Offers: This topic could be used to notify customers about ongoing promotions, discounts, and special offers on menu items.

Inventory Updates: If the app integrates with restaurant partners' inventory systems, a topic could capture updates about the availability of menu items. This would help prevent orders for items that are out of stock.

Customer Reviews and Ratings: Real-time customer reviews and ratings for delivered orders could be captured in this topic. This data could be used for sentiment analysis and quality improvement.

Payment Events: This topic could capture events related to payment processing, including successful payments, payment failures, refunds, and invoice generation.

Customer Support Interactions: If the app offers in-app customer support, a topic could be used to capture messages and interactions between customers and support agents.

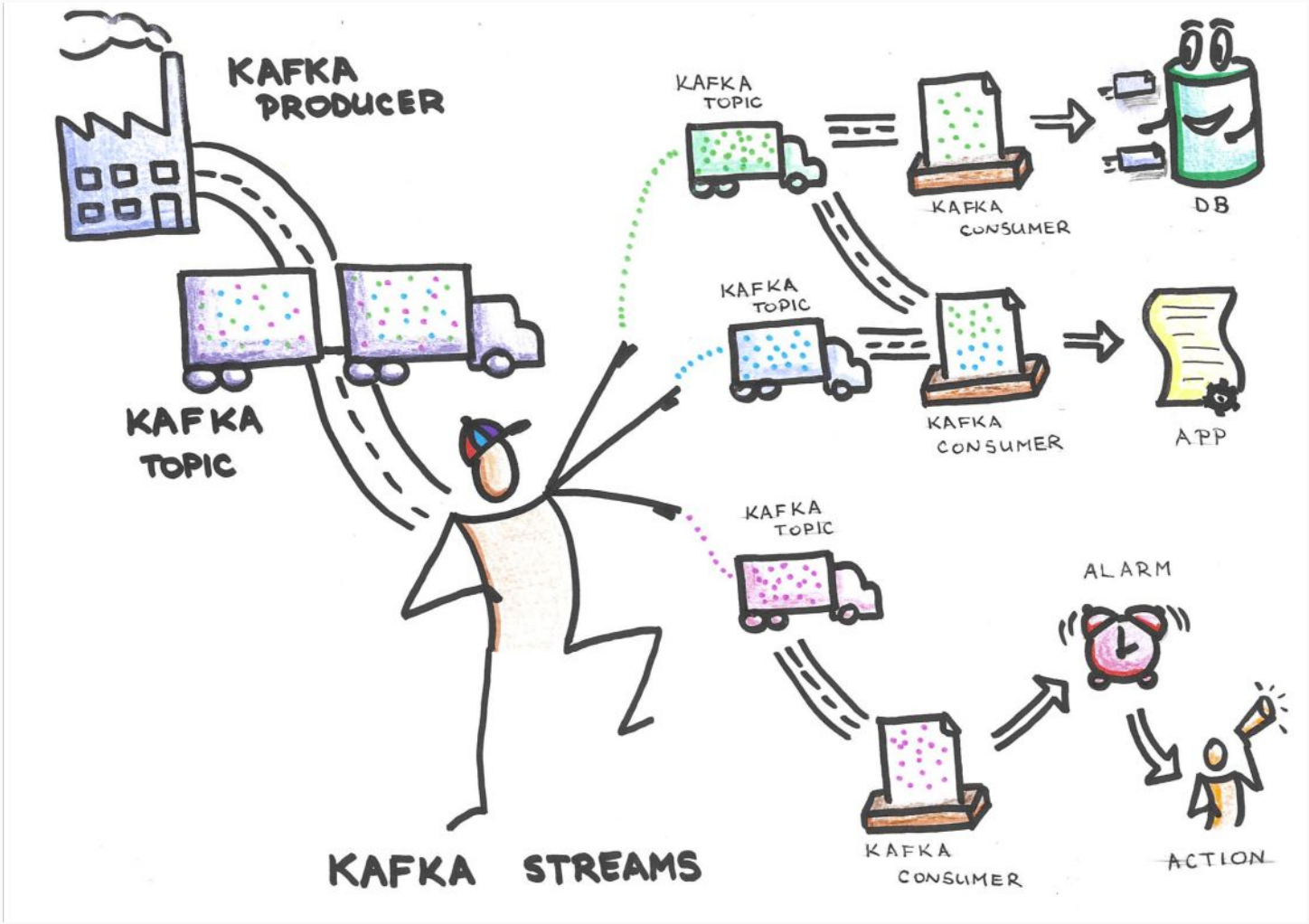
Menu Updates: If menu items change frequently, this topic could be used to communicate updates to the available menu items, including additions, removals, and modifications.

Driver Assignments: This topic could track the assignment of delivery drivers to specific orders, ensuring efficient routing and delivery management.

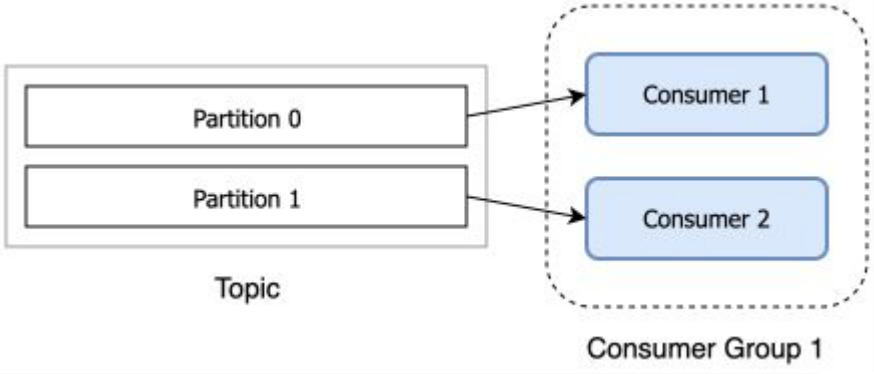
User Engagement Notifications: Notifications to users about their orders, delivery confirmations, and estimated arrival times could be sent via this topic.

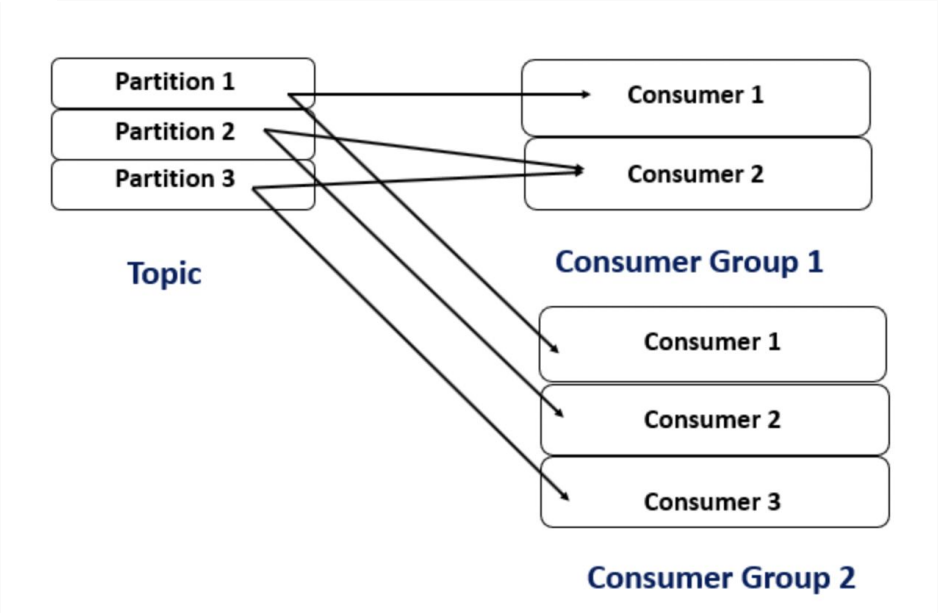
- LinkedIn: LinkedIn, one of the early adopters of Kafka, uses it for various purposes, including activity tracking, real-time data processing, and data integration.
- Uber: Uber uses Kafka for real-time event processing, including tracking driver and rider locations, trip updates, and handling real-time logistics.
- Netflix: Netflix employs Kafka for real-time data streaming to manage event data, monitor application health, and provide personalized recommendations to users.
- Airbnb: Airbnb utilizes Kafka for managing real-time events such as booking updates, notifications to hosts and guests, and tracking user interactions.
- Pinterest: Pinterest employs Kafka to capture user interactions, analyze user behavior, and personalize the content shown to users.

- Twitter: Twitter utilizes Kafka to handle the enormous volume of real-time tweets and events on their platform, ensuring tweets are distributed efficiently to users' timelines.
- Salesforce: Salesforce uses Kafka for event-driven architecture to enable real-time integration between various services and applications within their ecosystem.
- Cisco: Cisco utilizes Kafka to manage data streams generated by network devices, allowing for real-time monitoring and analysis of network performance.
- Walmart: Walmart uses Kafka for processing and analyzing real-time data from various sources, helping optimize inventory management and supply chain operations.
- Goldman Sachs: Goldman Sachs uses Kafka to process financial market data in real time, allowing them to make informed trading decisions.
- Microsoft: Microsoft Azure offers Azure Event Hubs, a fully managed real-time data streaming platform built on Kafka, allowing customers to ingest and process data from various sources.
- Intuit: Intuit, the company behind products like QuickBooks and TurboTax, uses Kafka to handle real-time data processing and analytics for their financial software applications.


KAFKA PRODUCER
KAFKA TOPIC
KAFKA TOPIC
KAFKA CONSUMER
DB
KAFKA TOPIC
KAFKA CONSUMER
APP
KAFKA TOPIC
ALARM
KAFKA CONSUMER
ACTION
KAFKA STREAMS

Kafka Streams is an API for writing client applications that transform data in Apache Kafka. You usually do this by publishing the transformed data onto a new topic. The data processing itself happens within your client application, not on a Kafka broker.


Partition 0
Partition 1
Topic
Consumer 1
Consumer 2
Consumer Group 1


Partition 1
Partition 2
Partition 3
Topic
Consumer 1
Consumer 2
Consumer Group 1
Consumer 1
Consumer 2
Consumer 3
Consumer Group 2

The offset is a position within a partition for the next message to be sent to a consumer. Kafka maintains two types of offsets.

- Current offset
- Committed offset

Current Offset

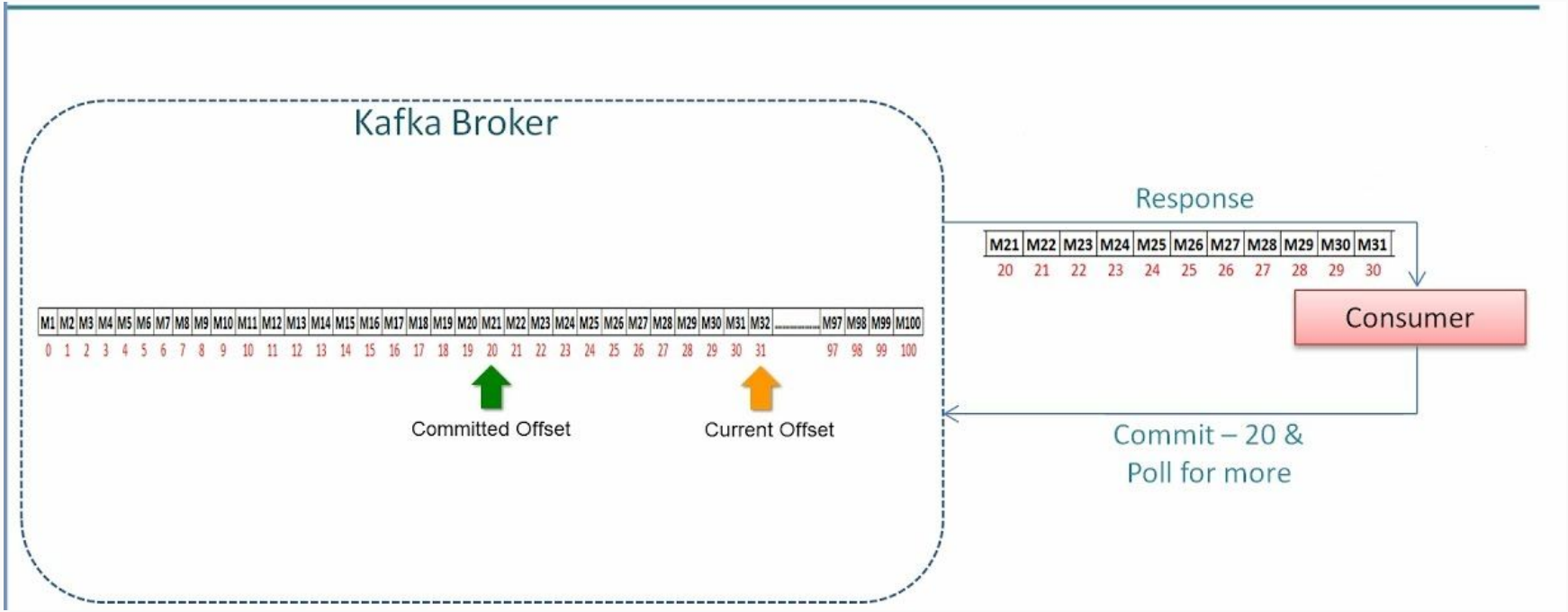
- When we call a poll method, Kafka sends some messages to us. Let us assume we have 100 records in the partition. The initial position of the current offset is 0.
- We made our first call and received 20 messages. Now Kafka will move the current offset to 20.
- When we make our next request, it will send some more messages starting from 20 and again move the current offset forward. The offset is a simple integer number that is used by Kafka to maintain the current position of a consumer. That's it.
- The current offset is a pointer to the last record that Kafka has already sent to a consumer in the most recent poll. So, the consumer doesn't get the same record twice because of the current offset.

Committed Offset

This offset is the position that a consumer has confirmed about processing. What does that mean? After receiving a list of messages, we want to process it. Right? This processing may be just storing them into HDFS. Once we are sure that we have successfully processed the record, we may want to commit the offset. So, the committed offset is a pointer to the last record that a consumer has successfully processed. For example, the consumer received 20 records. It is processing them one by one, and after processing each record, it is committing the offset.

Current offset -> Sent Records -> This is used to avoid resending same records again to the same consumer.

Committed offset -> Processed Records -> It is used to avoid resending same records to a new consumer in the event of partition rebalance.
The committed offset is critical in the case of partition rebalance.
In the event of rebalancing. When a new consumer is assigned a new partition, it should ask a question. Where to start? What is already processed by the previous owner? The answer to the question is the committed offset.


Kafka Broker
M1 M2 M3 M4 M5 M6 M7 M8 M9 M10 M11 M12 M13 M14 M15 M16 M17 M18 M19 M20 M21 M22 M23 M24 M25 M26 M27 M28 M29 M30 M31 M32 ..... M97 M98 M99 M100
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 97 98 99 100
Committed Offset
Current Offset
Response
M21 M22 M23 M24 M25 M26 M27 M28 M29 M30 M31
20 21 22 23 24 25 26 27 28 29 30
Consumer
Commit – 20 &
Poll for more

What is partition rebalancing in Kafka?

Rebalancing comes into play in Kafka when consumers join or leave a consumer group. In either case, there is a different number of consumers over which to distribute the partitions from the topic(s), and, so, they must be redistributed and rebalanced