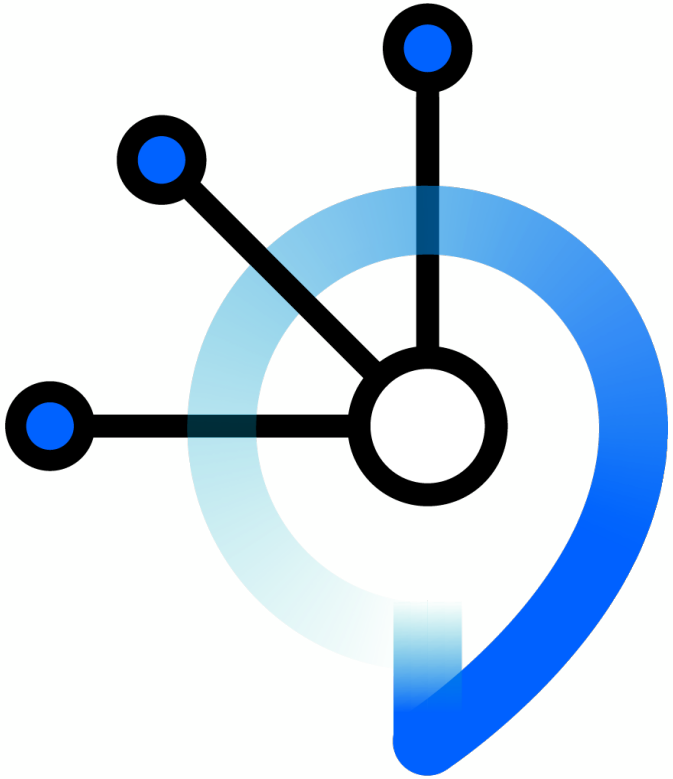


IBM Event Streams

Apache Kafka® for the Enterprise

A core part of Cloud Pak for Integration

Overview

Mangesh Patankar

Developer Advocate
Developer EcoSystem Group

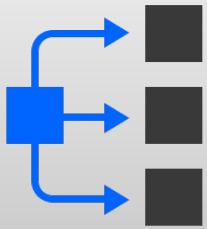
Agenda

- **Event Streaming - MQ**
- **Event Streaming**
  - **Kafka**
  - **Use cases**
  - **IBM Event Streaming - Overview**

Event Streaming & Message Queuing Need Different Capabilities

Event Streaming

Stream History



Scalable consumption

Immutable data

Message Queueing

Transient Data Persistence

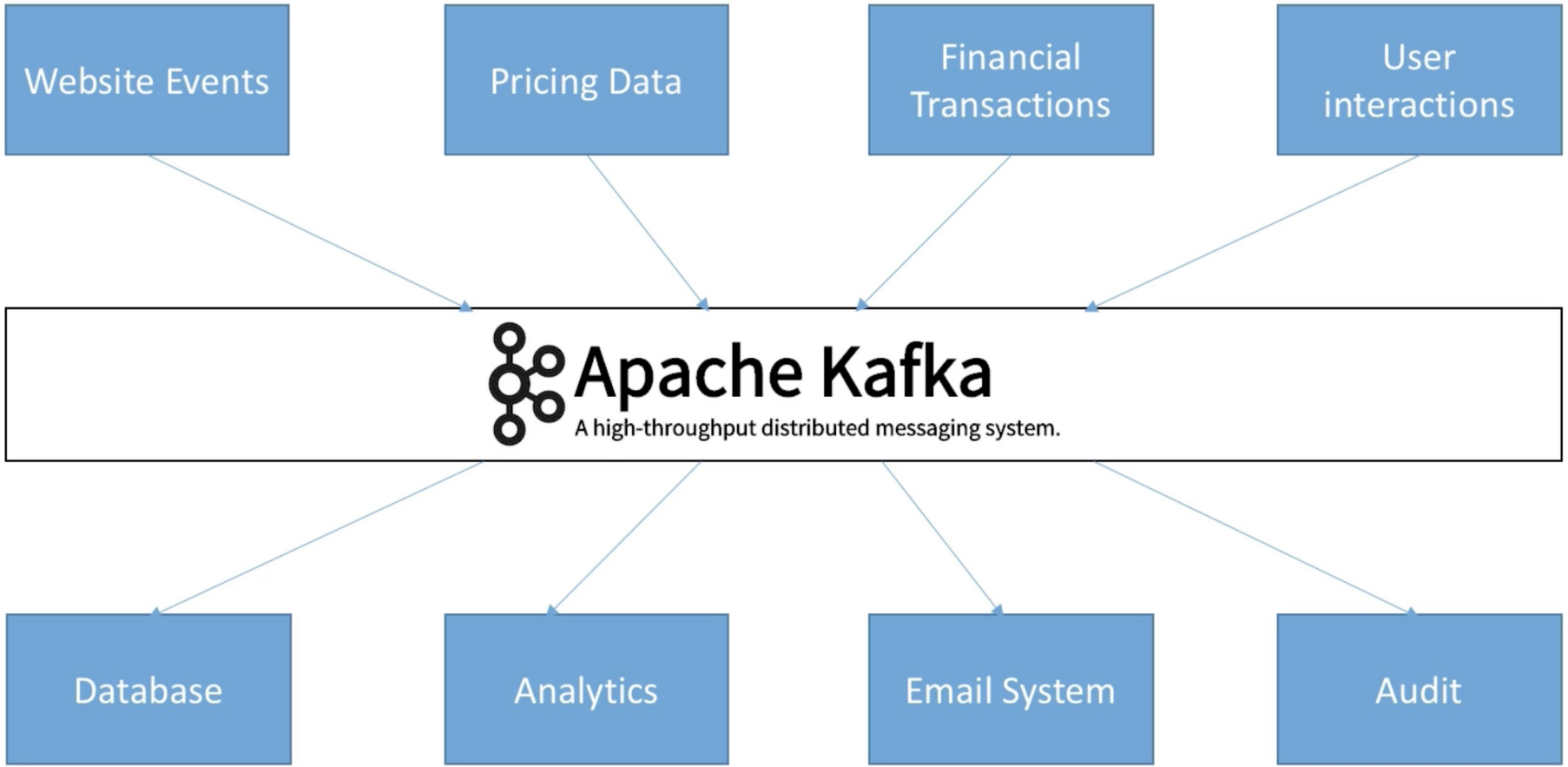
Request / Reply

Targeted Reliable Delivery

Kafka

- Created by LinkedIn, now Open Source Project mainly maintained by Confluent
- Distributed, resilient architecture, fault tolerant
- Horizontal scalability:
  - Can scale to 100s of brokers
  - Can scale to millions of messages per second
- High performance (latency of less than 10ms) – real time
- Used by the 2000+ firms, 35% of the Fortune 500:

Examples


Website Events
Pricing Data
Financial Transactions
User interactions
Apache Kafka
A high-throughput distributed messaging system.
Database
Analytics
Email System
Audit

Kafka - Use Cases

- Messaging System
- Activity Tracking
- Gather metrics from many different locations
- Application Logs gathering
- Stream processing (with the Kafka Streams API or Spark for example)
- De-coupling of system dependencies
- Integration with Spark, Flink, Storm, Hadoop, and many other Big Data technologies

Examples

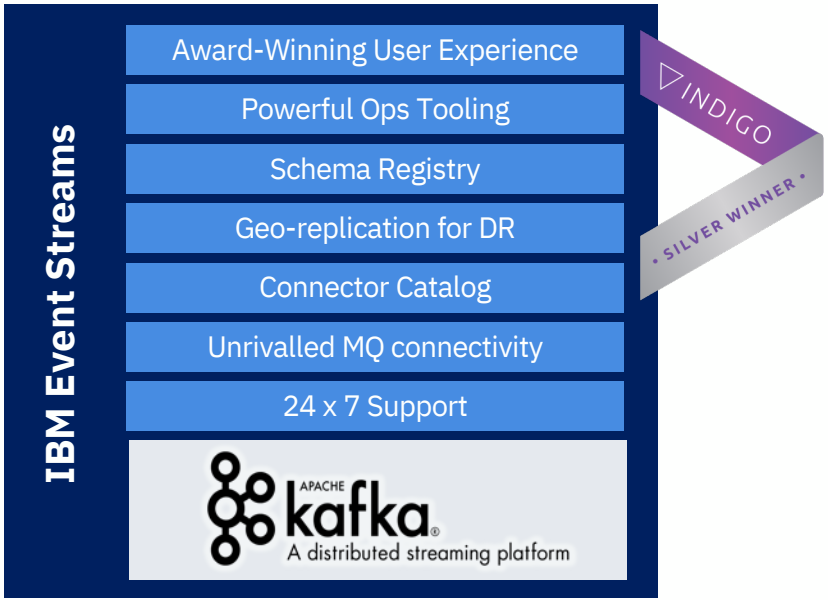
- **Netflix** uses Kafka to apply recommendations in real-time while you're watching TV shows
- **Uber** uses Kafka to gather user, taxi and trip data in real-time to compute and forecast demand, and compute surge pricing in real-time
- **LinkedIn** uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real time.

- Remember that Kafka is only used as a transportation mechanism!

IBM – Apache Kafka

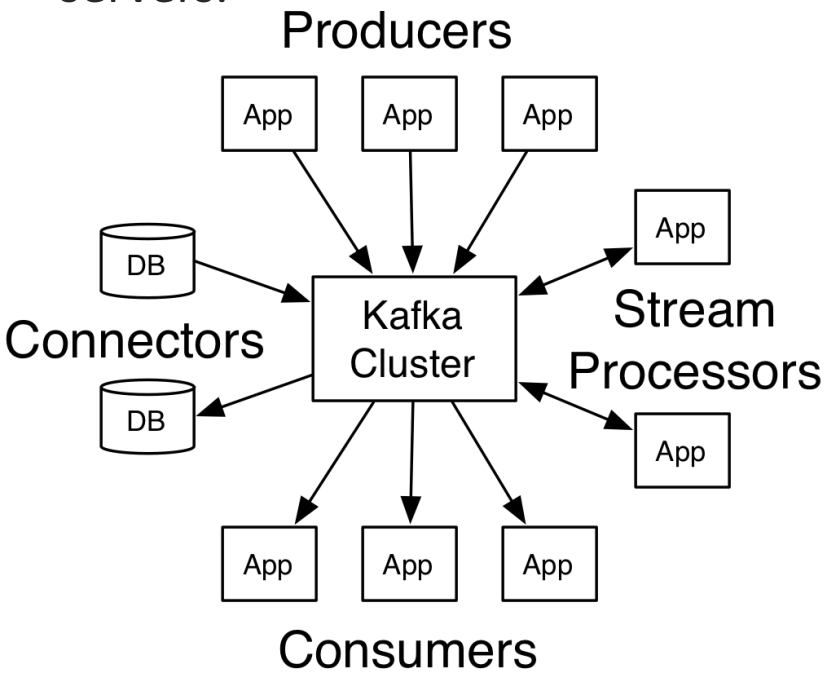
- Some vendors provide commercially supported versions of Kafka
- Our implementation is IBM Event Streams.
- IBM is also a key contributor to the Apache Kafka open source project as committers.

IBM Event Streams is fully supported Apache Kafka® with value-add capabilities



What is Apache Kafka?

Apache Kafka is a ***publish-subscribe*** messaging system which lets you send messages between processes, applications, and servers.



Providers/Producers: Emits and publishes events

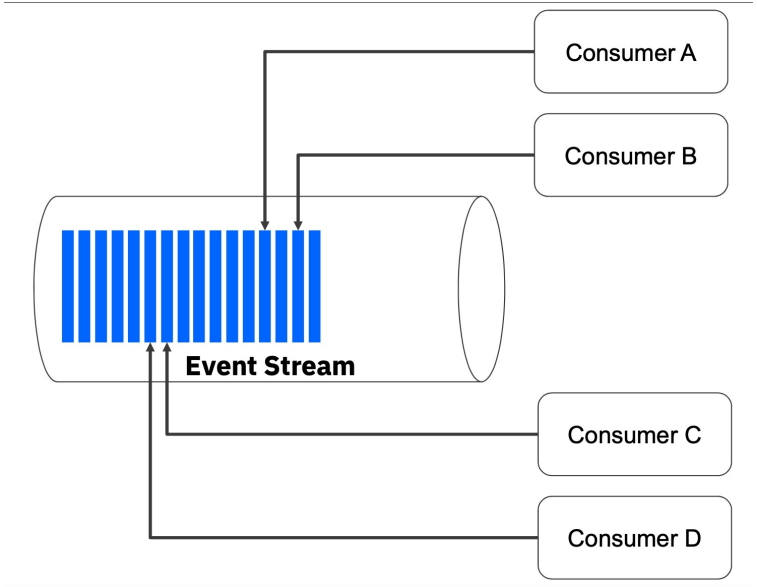
A server that stores events and manages the topics

Client that allows applications to interact as a provider or consumer.

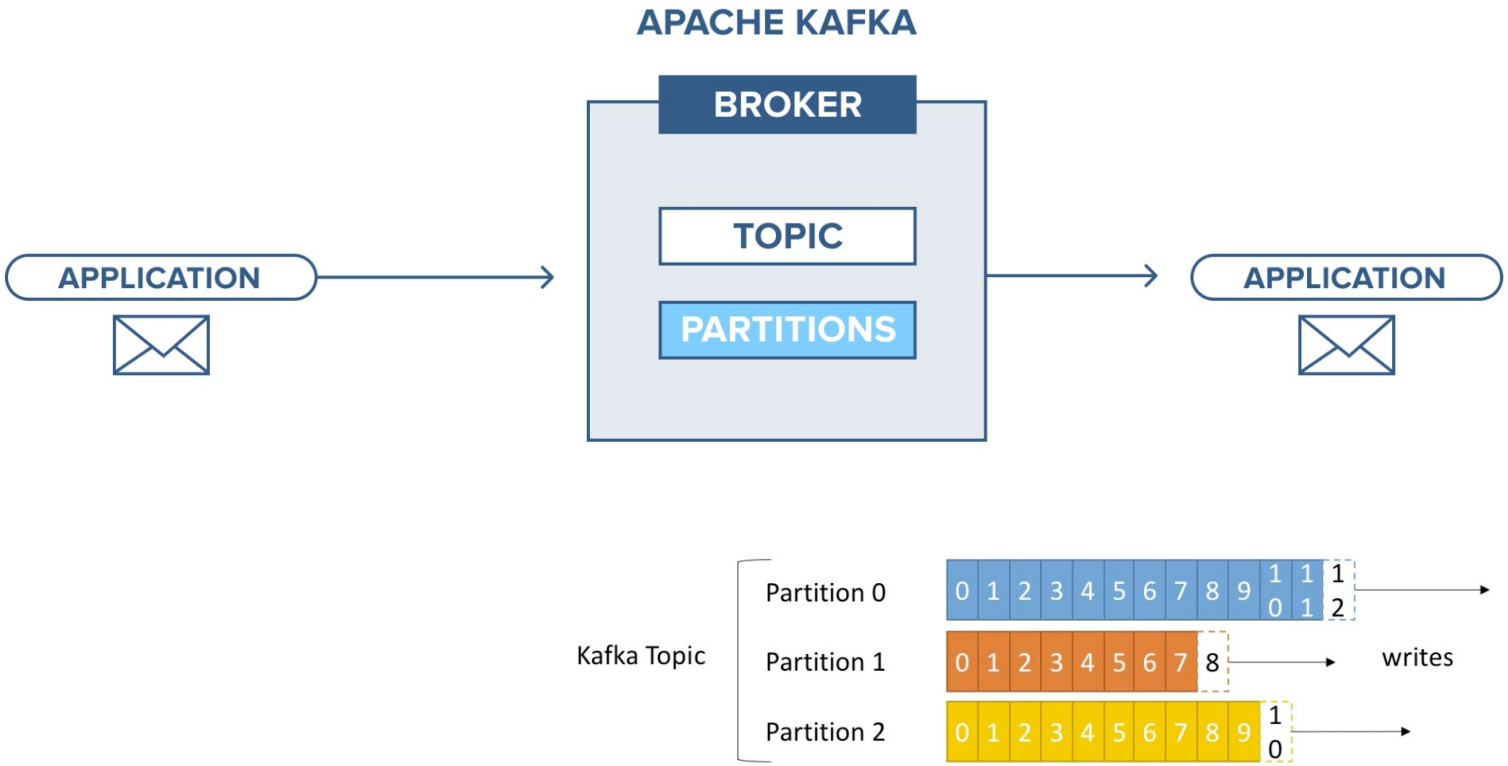
Cosumers/Subscribers: Read events

Capabilities for an event streaming technology

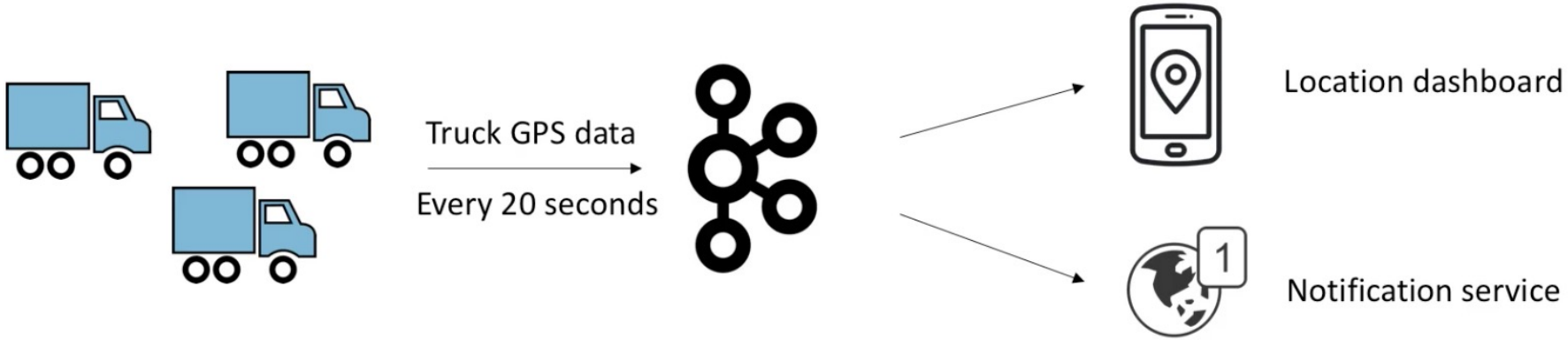
- Stream history : By using Topics The topic can be considered a stream history of all the events that are emitted, and allows subscribers to rewind to different locations of the topic
- High performance : designed to handle millions of events a second
- Scalable subscription: increases in the number of subscribers to a topic has a minimal impact on the resource
- Decoupled communication: provides an intermediary between the two applications, which means that they are decoupled



Publish-subscribe durable messaging system

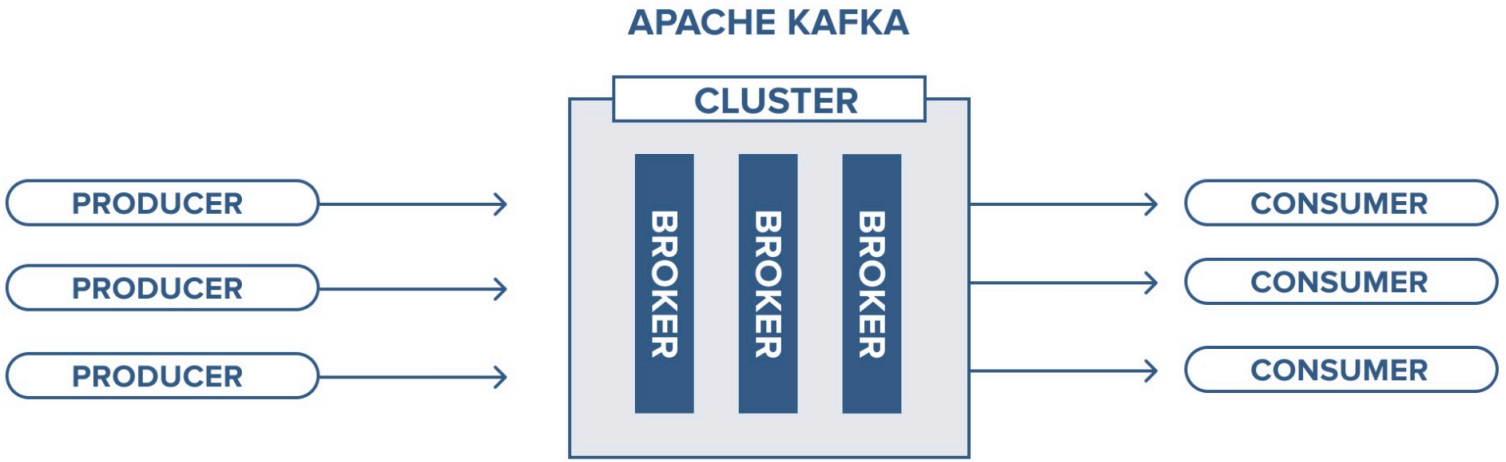


Topic examples: truck_gps

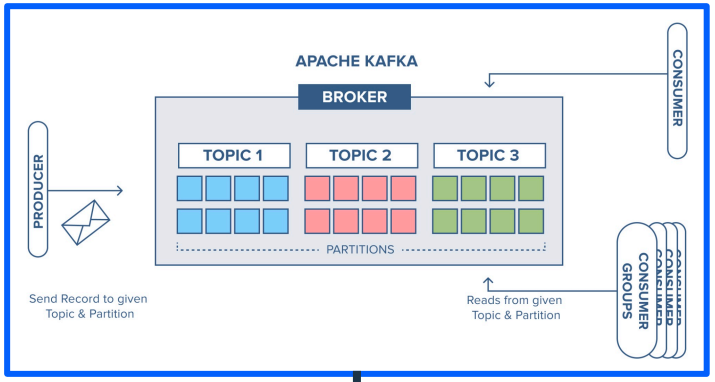


- Say you have a fleet of trucks, each truck reports its GPS position to Kafka.
- You can have a topic trucks_gps that contains the position of all trucks.
- Each truck will send a message to Kafka every 20 seconds, each message will contain the truck ID and the truck position (latitude and longitude)
- We choose to create that topic with 10 partitions (arbitrary number)

Kafka Broker



Record Flow in Kafka



Scenario: 3 topics, where each topic has 8 partitions.

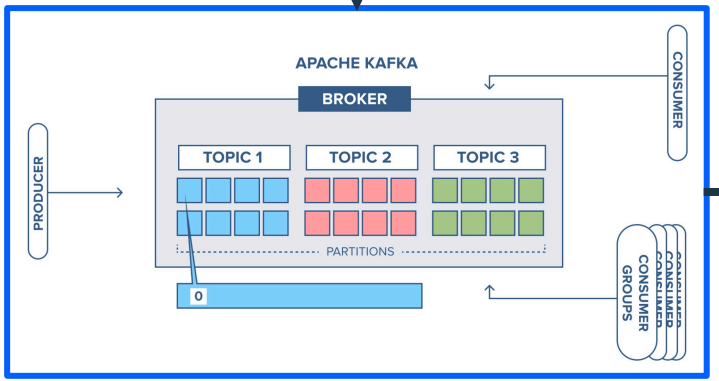
The producer sends a record to partition 1 in topic 1

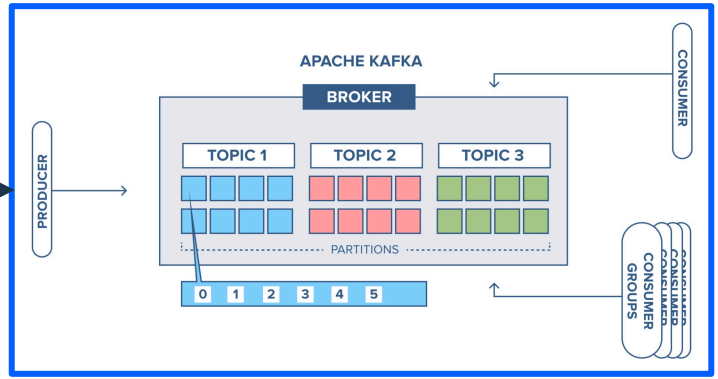
and since the partition is empty the record ends up at offset 0.

Next record is added to partition 1 will and up at offset 1, and the next record at offset 2 and so on.
This is referred as Commit Log

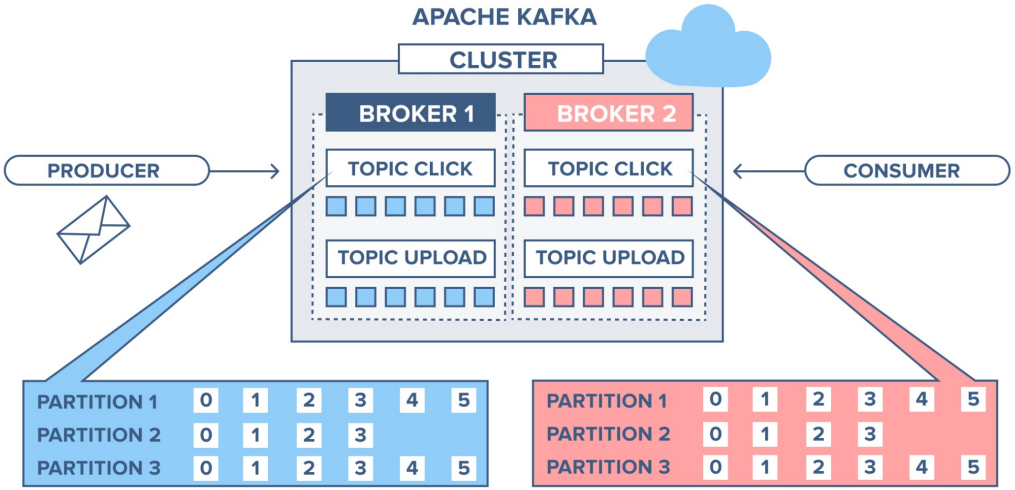
Each record is appended to the log and there is no way to change the existing records in the log.

This is also the same offset that the consumer uses to specify where to start reading.





Example (Use case) Website activity tracking



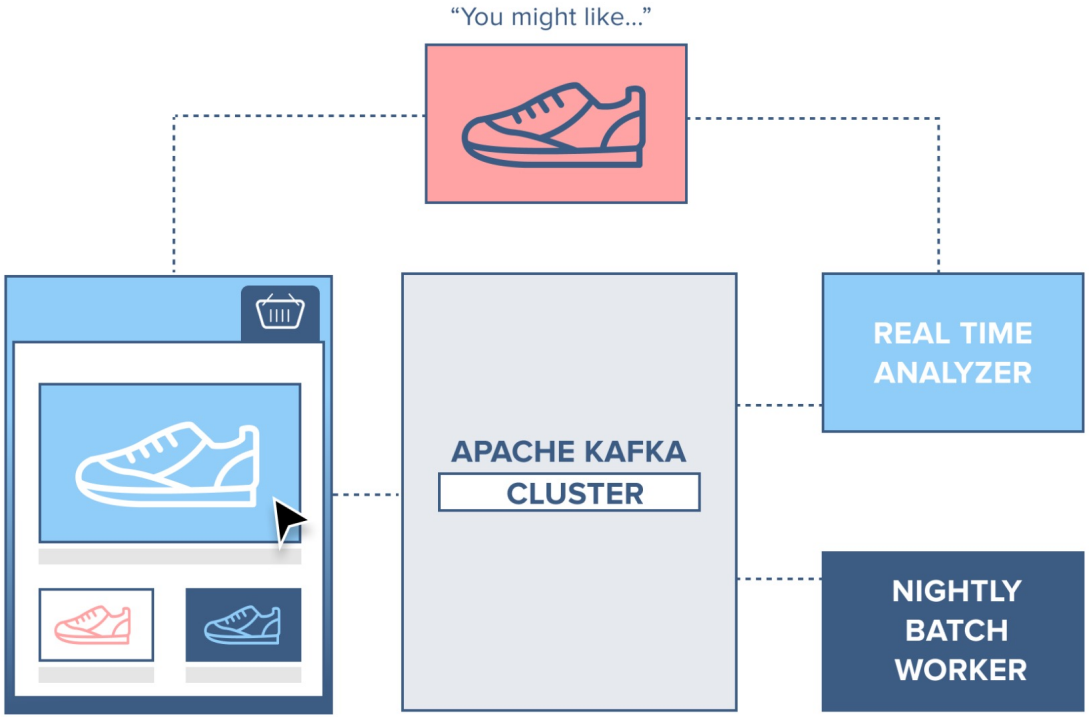
A user with user-id 0 clicks on a button on the website.

The web application publishes a record to partition 0 in the topic "click".

The record is appended to its commit log and the message offset is incremented.

The consumer can pull messages from the click-topic and show monitoring usage in real-time, or it can replay previously consumed messages by setting the offset to an earlier one.

Web Shop



, each action performed by a consumer is recorded and sent to Kafka.

A separate application comes along and consumes these messages, filtering out the products the consumer has shown an interest in and gathering information on similar products.

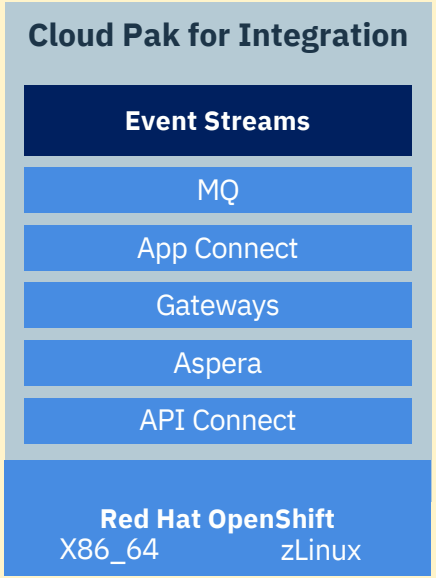
This ‘similar product’ information is then sent back to the webshop for it to display to the consumer in real-time.

Or

data is persistent in Kafka, a batch job can run overnight on the ‘similar product’ information gathered by the system, generating an email for the customer with suggestions of products

Packaging and Deployment Options to Suit Different Needs

Self Managed Software

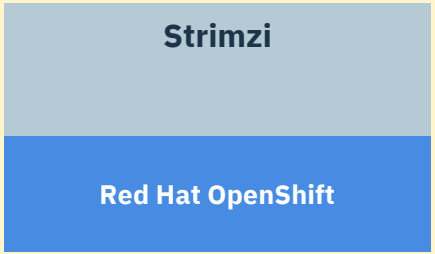


Fully Managed Service

Hosted service on IBM public cloud

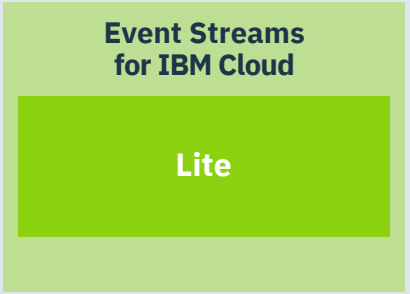
Getting started

Self Managed Software



https://strimzi.io

Fully Managed Service



Hosted service on IBM public cloud

Get Started at: ibm.com/cloud/event-streams/get-started

Building on the Red Hat OpenShift Container Platform

Develop, deploy, and manage existing and container-based apps seamlessly across physical, virtual, and public cloud infrastructures

Automated workflows

Integrates with CI/CD tools

Built on proven Open Source Technologies

Application Development teams

IT Platform teams

Self-serve Provision, Build and Deploy Apps

Modernize applications

Accelerate Development Processes

Red Hat OpenShift

Policy based Control and Automation

Cluster, Schedule, Orchestrate services

Load Balancing Auto Scaling

Secure Enterprise Grade Kubernetes

Operators for Kafka

- **Kubernetes** allows you to automatically deploy and run workloads
- **Operators** extends that to be able to control stateful application resources in the same way

Allows the system to behave like a human operator that has deep knowledge of how the system ought to behave

Manage stateful apps	*Reduce Complexity*	*Kubernetes native*	*Failure recovery*	*Scaling*	*Manage updates*

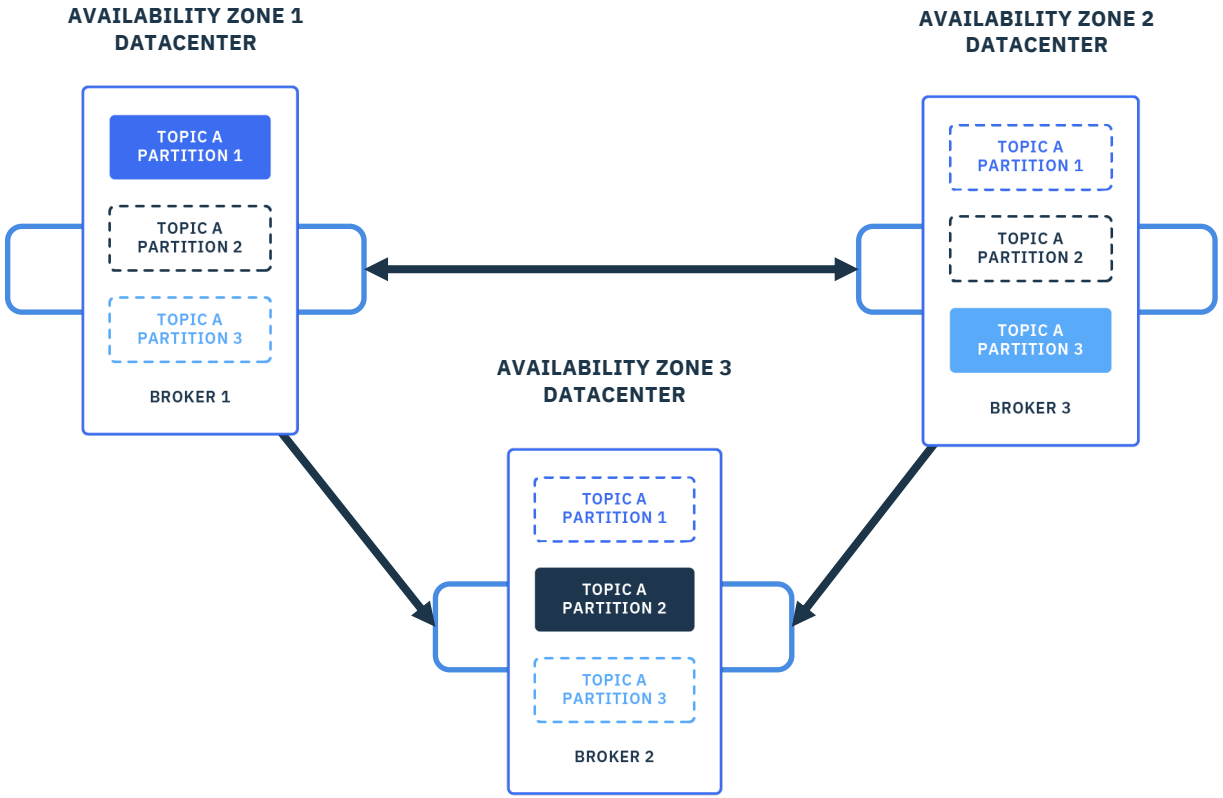
- Describe declaratively what the state of the system should be, and the system automatically responds to achieve the desired state
- For Kafka that means you can create topics, control partitioning and replication from a kubectl command
- Deploy an end-to-end application with all its resources from a single *kubectl apply*

Operators for Kafka

- Operator is responsible for managing all Event Streams resources
  - Specialised operator is able to provide much better control and information than a general purpose tool like Helm
  - Operator understands Kafka concepts like users, topics and partitions
- All the well established Kubernetes tools and techniques can now be applied to Kafka
- Put all resources under change control in Kubernetes manifests
  - Infrastructure as code

Manage stateful apps	*Reduce complexity*	*Kubernetes native*	*Failure recovery*	*Scaling*	*Manage updates*

Enhanced resilience with clusters across multiple zones

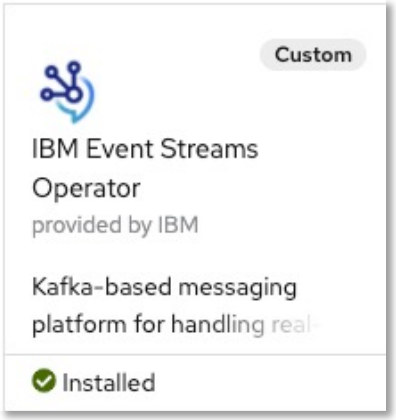


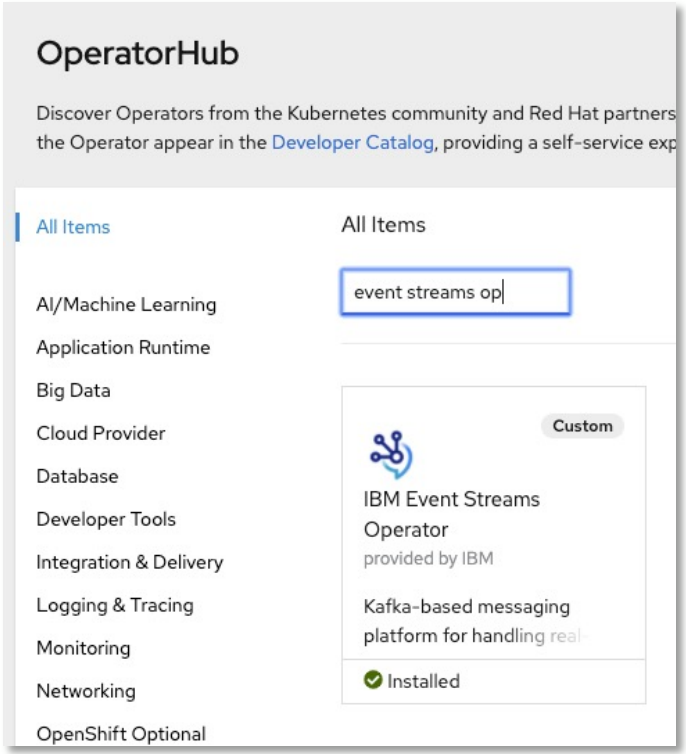
Multi-Availability Zone

- Must have at least 3 zones
- Kafka brokers and ZooKeeper servers span across zones
- Can tolerate failure of a zone with no service degradation
- High-speed network with low latency between zones required (< 20ms)

Making Apache Kafka Intuitive and Easy

Native deployment on OpenShift with Operators



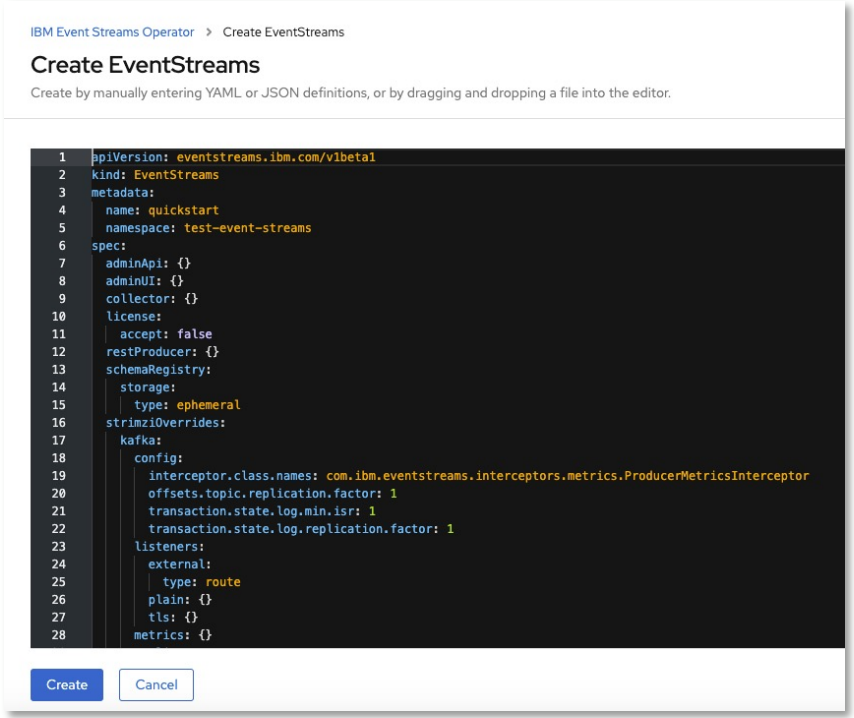


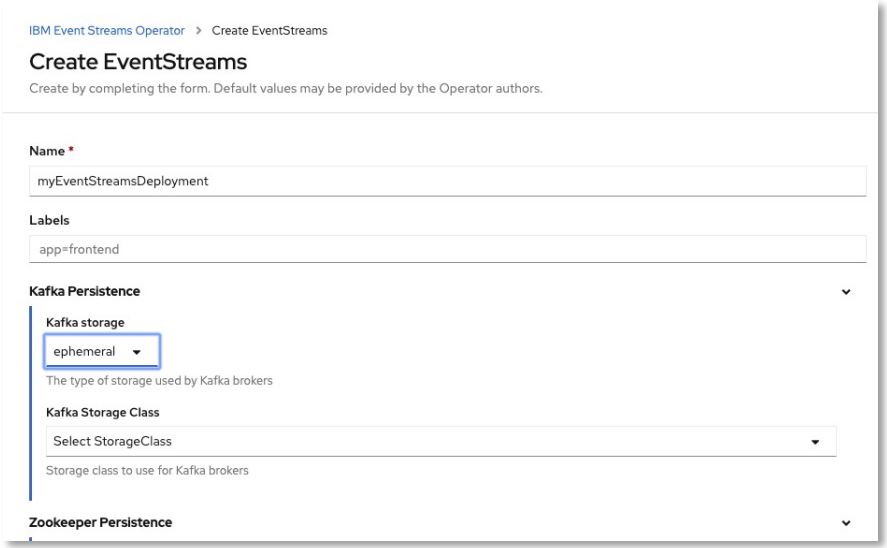
- Kafka has many distinct components to deploy, configure and coordinate for secure connectivity
- Container placement critical to ensure production-level availability
- Secured network traffic ingress
- Ensuring consistent and repeatable deployment

Simple to deploy and Manage

Making Apache Kafka Intuitive and Easy

Native deployment on OpenShift with Operators

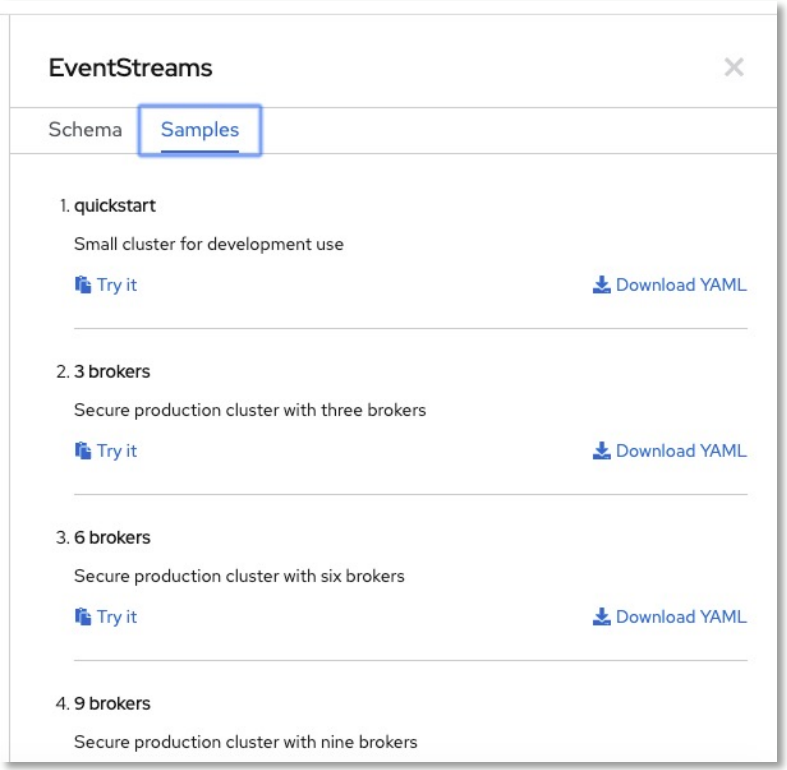




- YAML for full control
- Form-based entry for getting started quickly

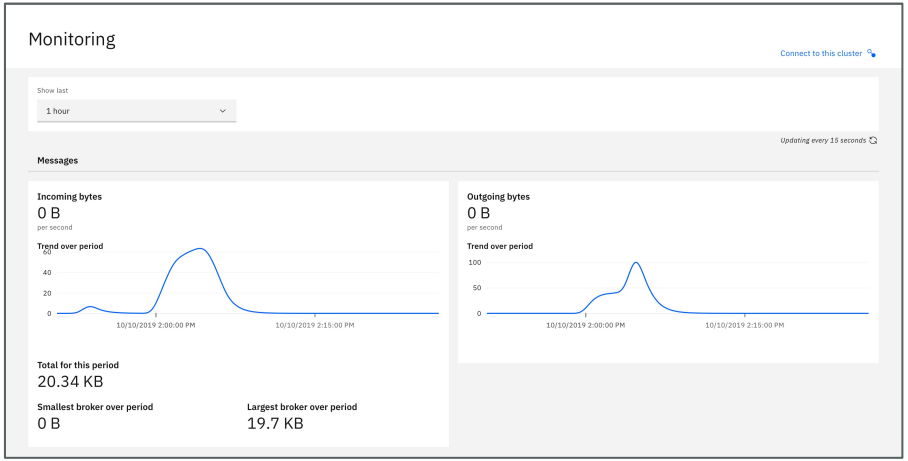
Making Apache Kafka Intuitive and Easy

Native deployment on OpenShift with Operators



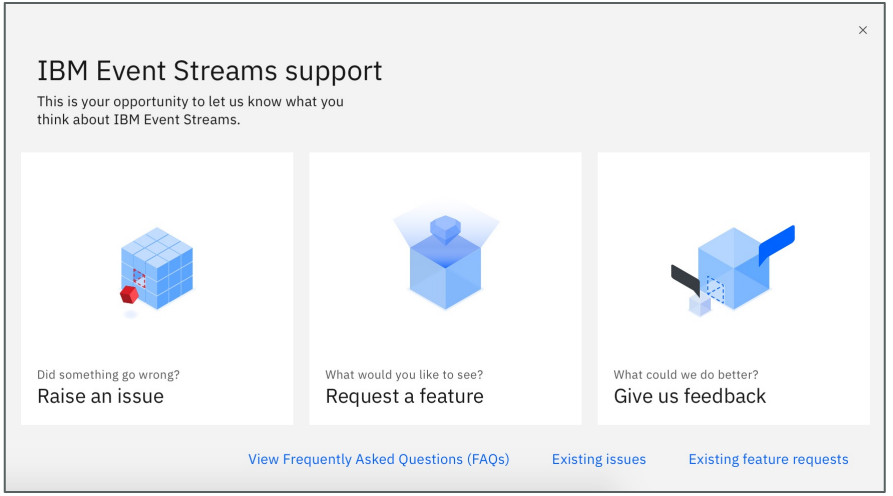
Operator templates to get started quickly with full example configurations

Making Apache Kafka Intuitive and Easy

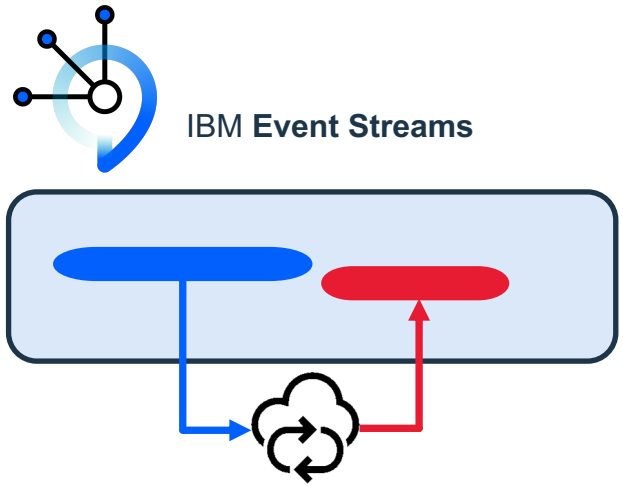


Monitor status at a glance

Integrated feedback and support

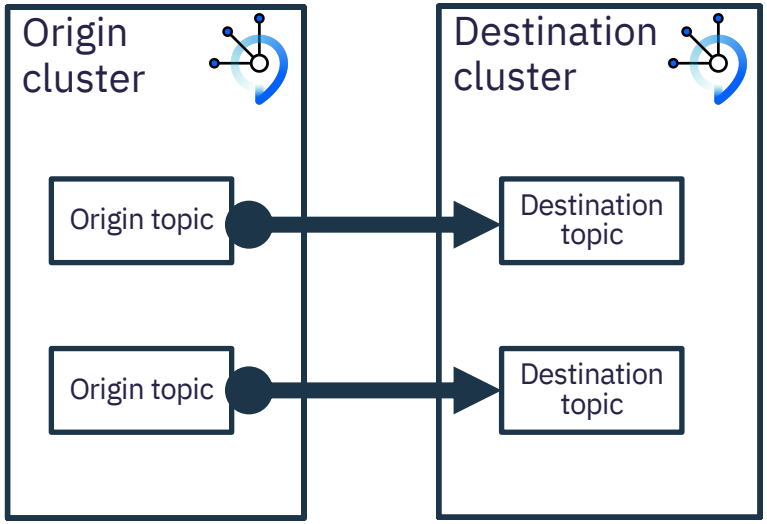


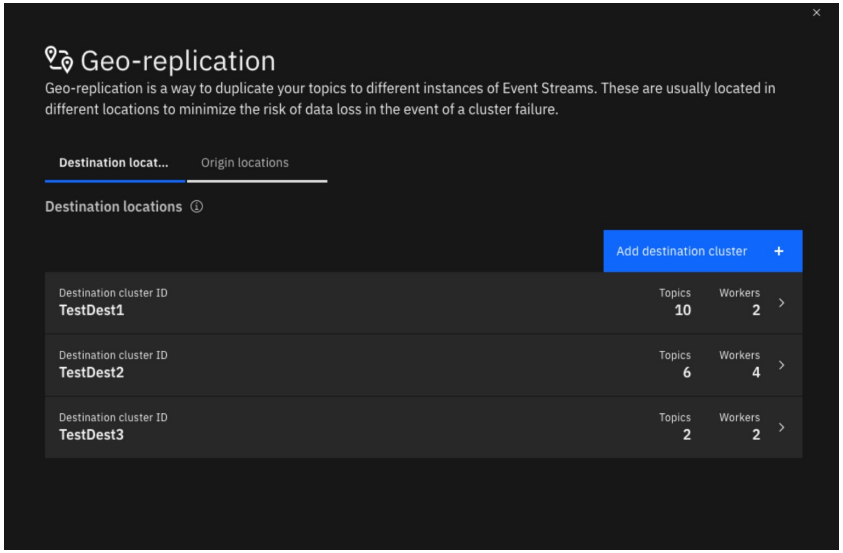
Integrated with Key Monitoring Tools



External monitoring tools
Datadog, Splunk, etc

Geo-Replication Makes Disaster Recovery Simple





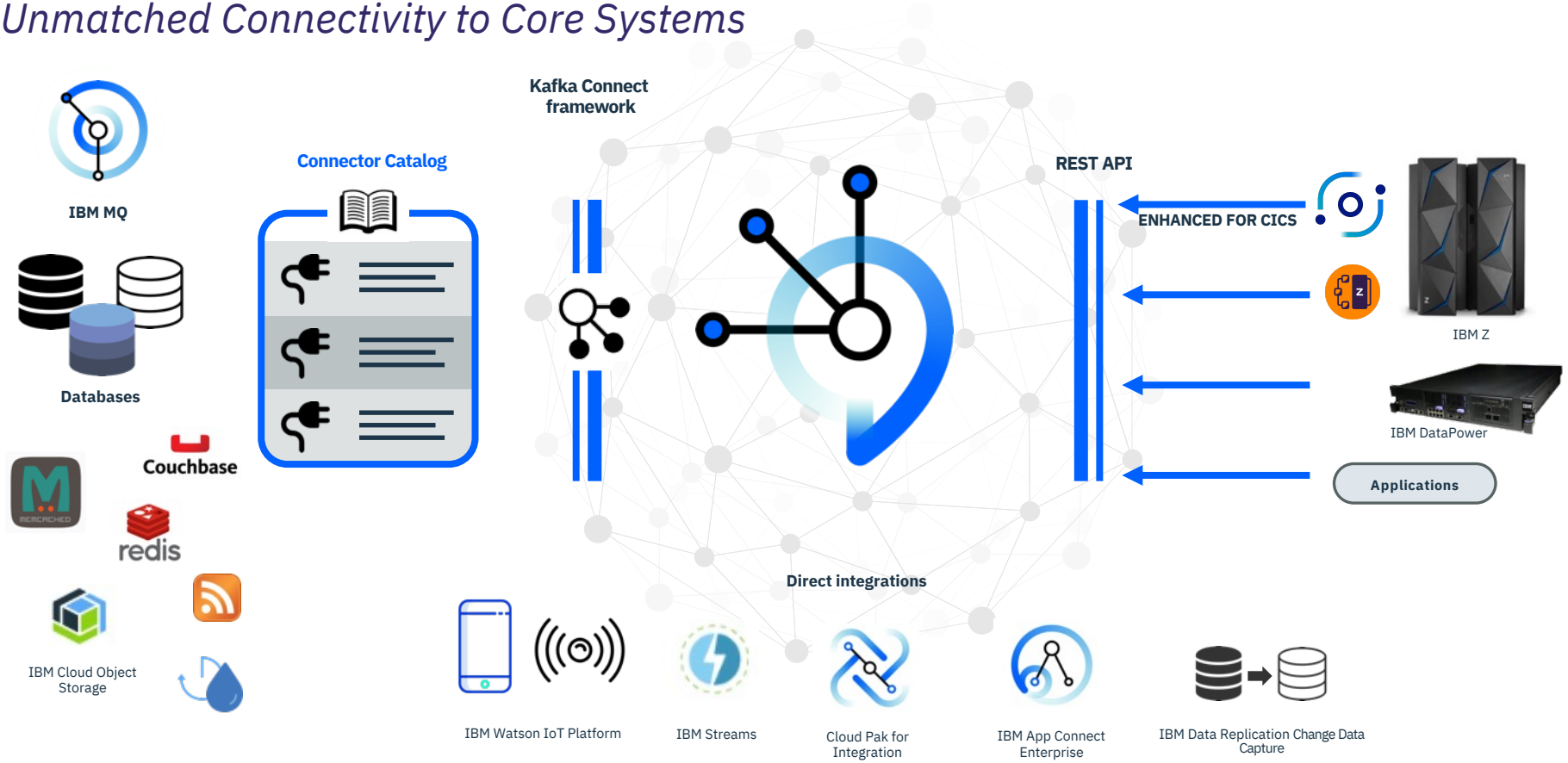
Target is take-over of workload on the destination cluster by business applications within 15 minutes

Easy configuration using the Event Streams UI from the origin cluster sets up the replicator and security credentials

At-least-once reliability so messages are not lost

Use Existing Data in New Ways that Yield Competitive Advantage

Unmatched Connectivity to Core Systems



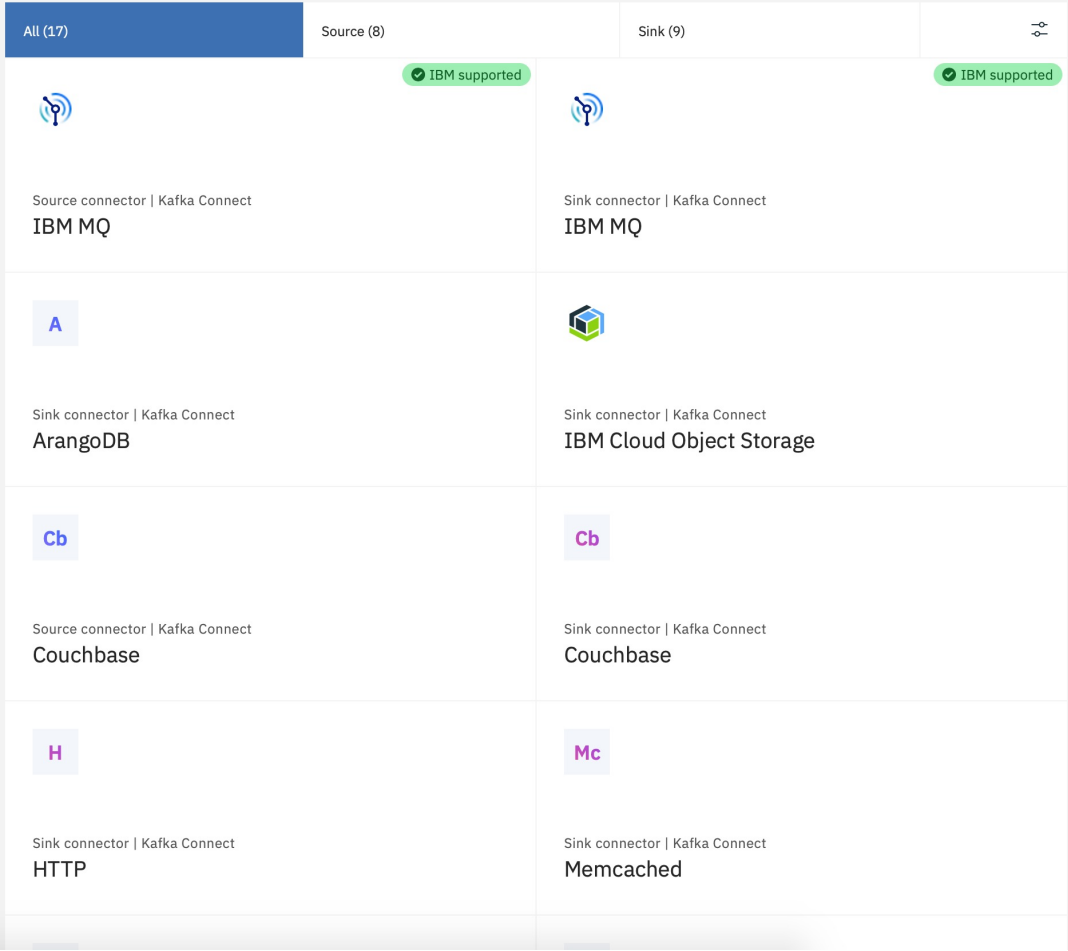
Welcome to the IBM Event Streams

Connector catalog

Kafka Connect is a framework for connecting Kafka to external systems. It uses source connectors to move data into Kafka, and sink connectors to move data out of Kafka.

The Event Streams connector catalog contains a list of tried and tested connectors from both the community and IBM.

Find out more about Kafka Connect →



Ready for Mission-Critical Workloads

All with IBM 24x7 worldwide support

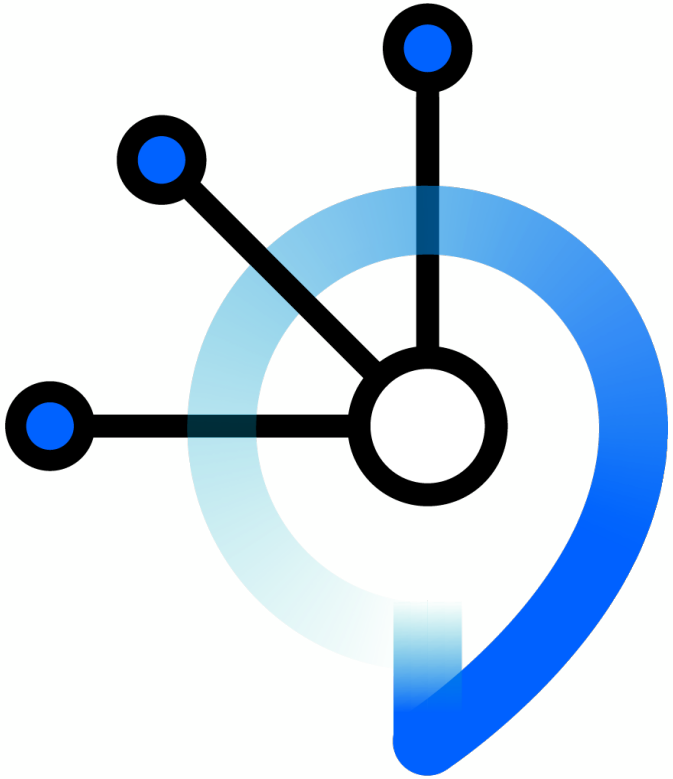
IBM has years of experience running Apache Kafka across the globe

Find Out More

Explore IBM Event Streams at

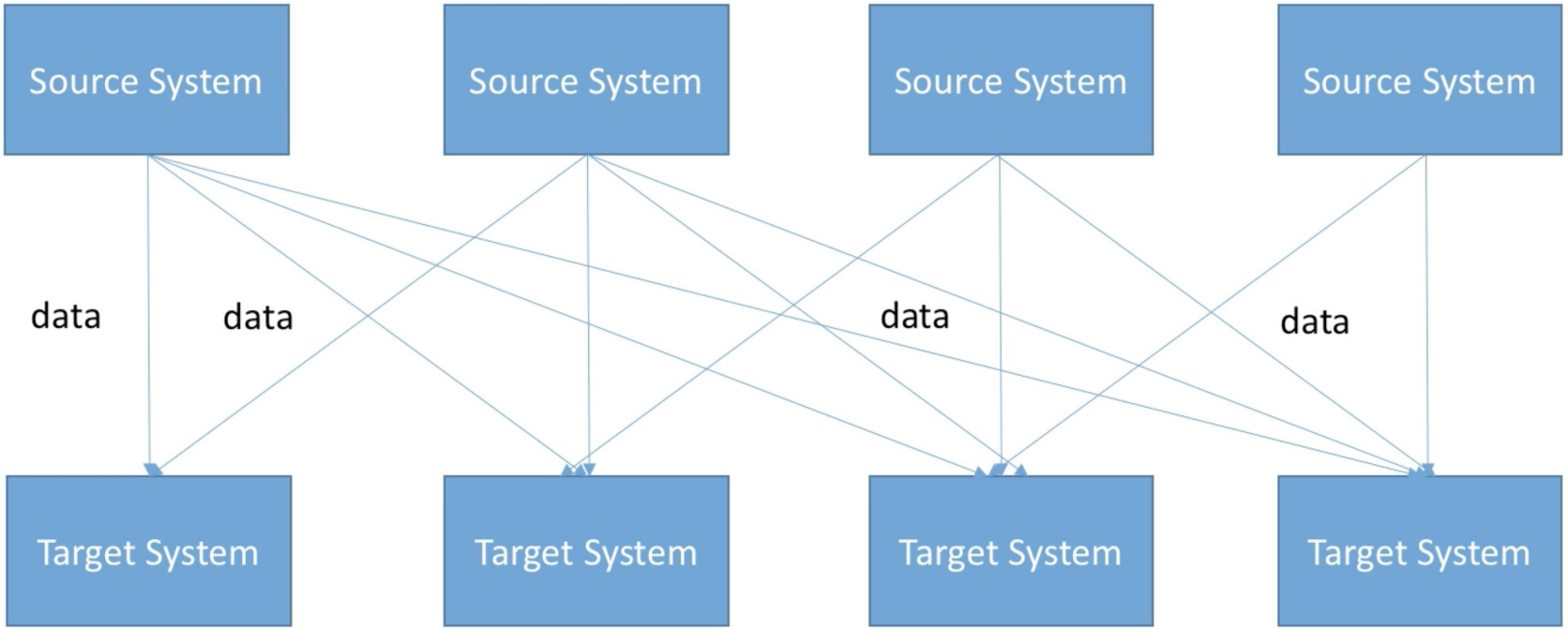
https://ibm.github.io/event-streams/

Contact us: eventstreams@uk.ibm.com



Thank You

Integration



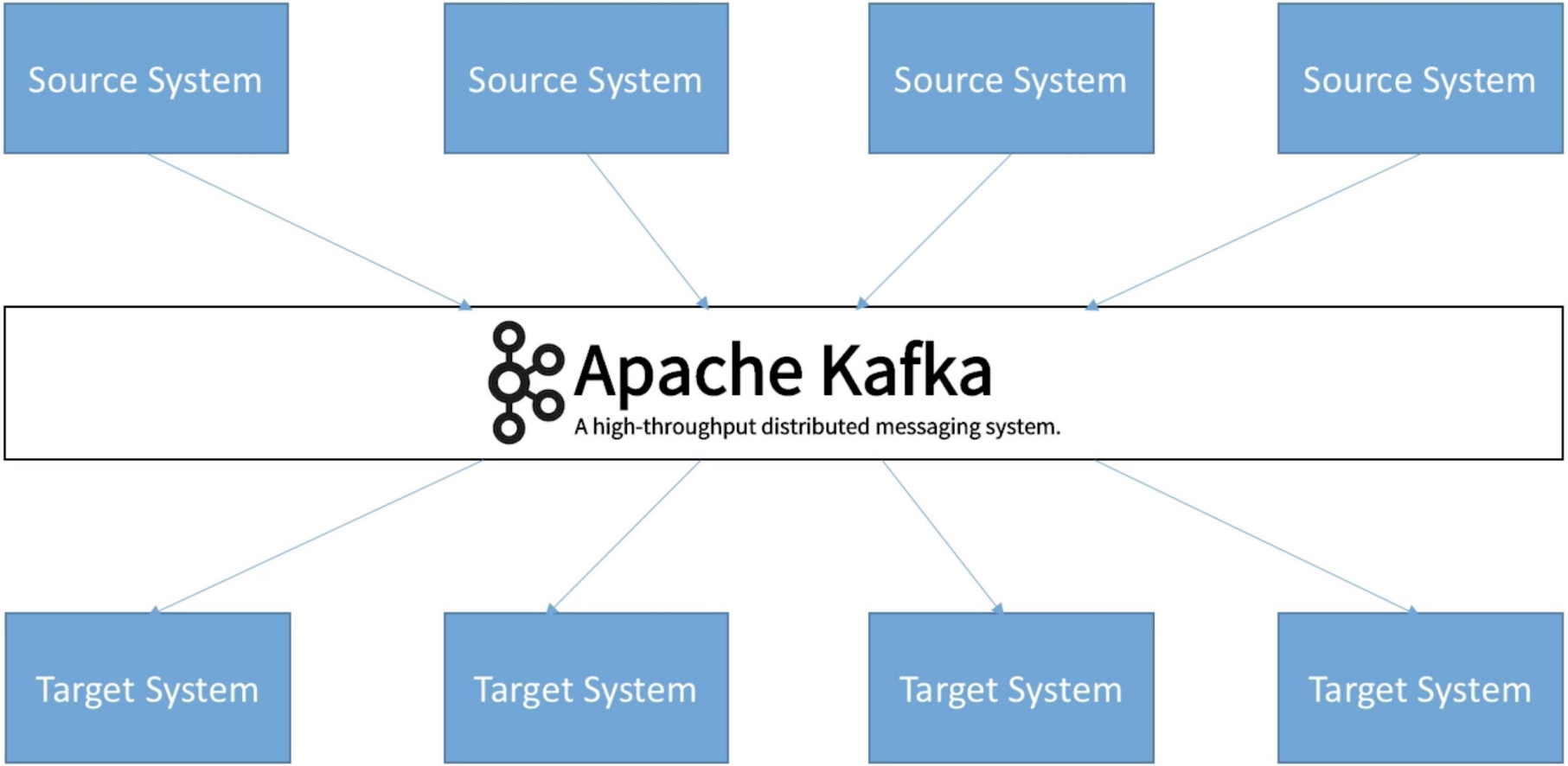
Problems faced

- If you have 4 source systems, and 6 target systems, you need to write 24 integrations!

- Each integration comes with difficulties around
  - Protocol – how the data is transported *(TCP, HTTP, REST, FTP, JDBC…)*
  - Data format – how the data is parsed *(Binary, CSV, JSON, Avro…)*
  - Data schema & evolution – how the data is shaped and may change

- Each source system will have an increased load from the connections

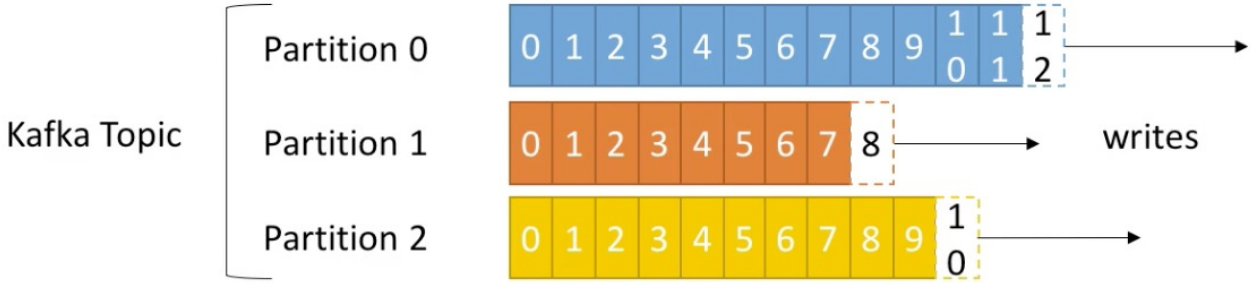
Where Kafka fits: Decoupling of data streams & systems



Brokers

- A Kafka cluster is composed of multiple brokers (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is 3 brokers, but some big clusters have over 100 brokers
- In these examples we choose to number brokers starting at 100 (arbitrary)

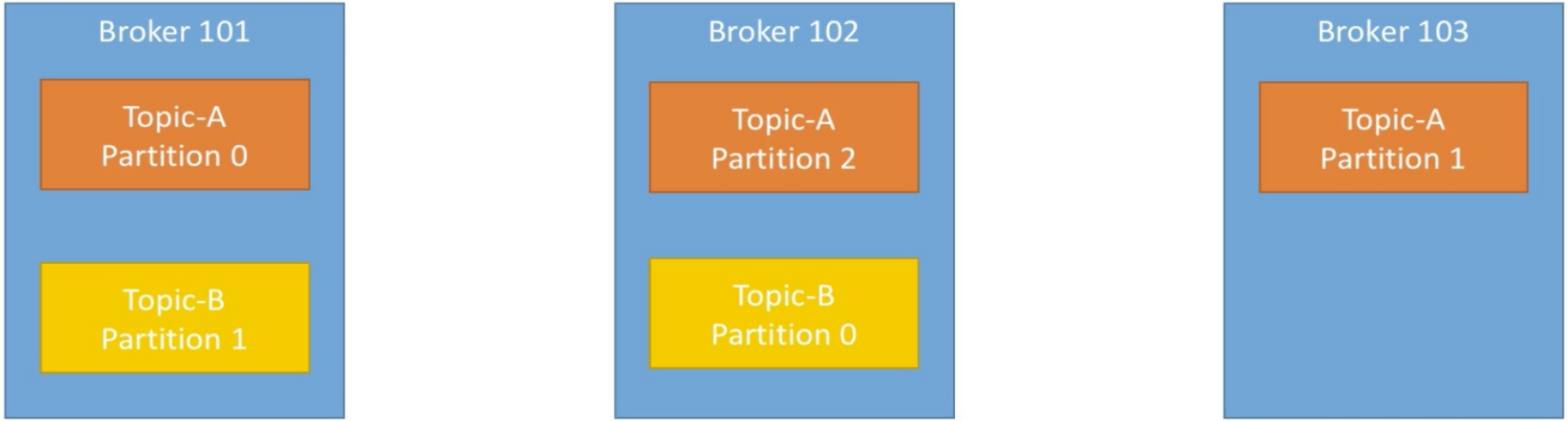
Topics, partitions and offsets



- Offset only have a meaning for a specific partition.
  - E.g. offset 3 in partition 0 doesn't represent the same data as offset 3 in partition 1
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is one week)
- Once the data is written to a partition, it can't be changed (immutability)
- Data is assigned randomly to a partition unless a key is provided (more on this later)

Broker and topics

- Example of **Topic-A** with **3 partitions**
- Example of **Topic-B** with **2 partitions**



- Note: Data is distributed and Broker 103 doesn't have any **Topic B** data