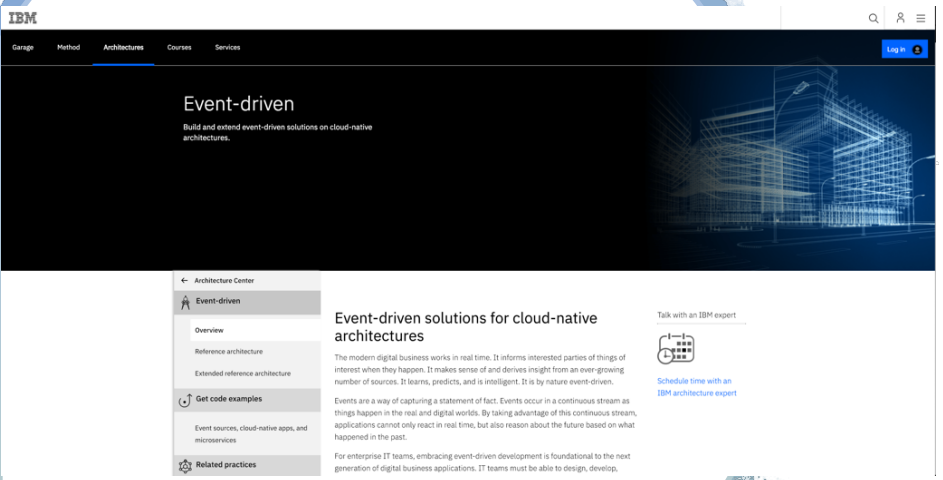
IBM

Event Driven Architecture

Introduction

Cloud Garage Solution Engineering

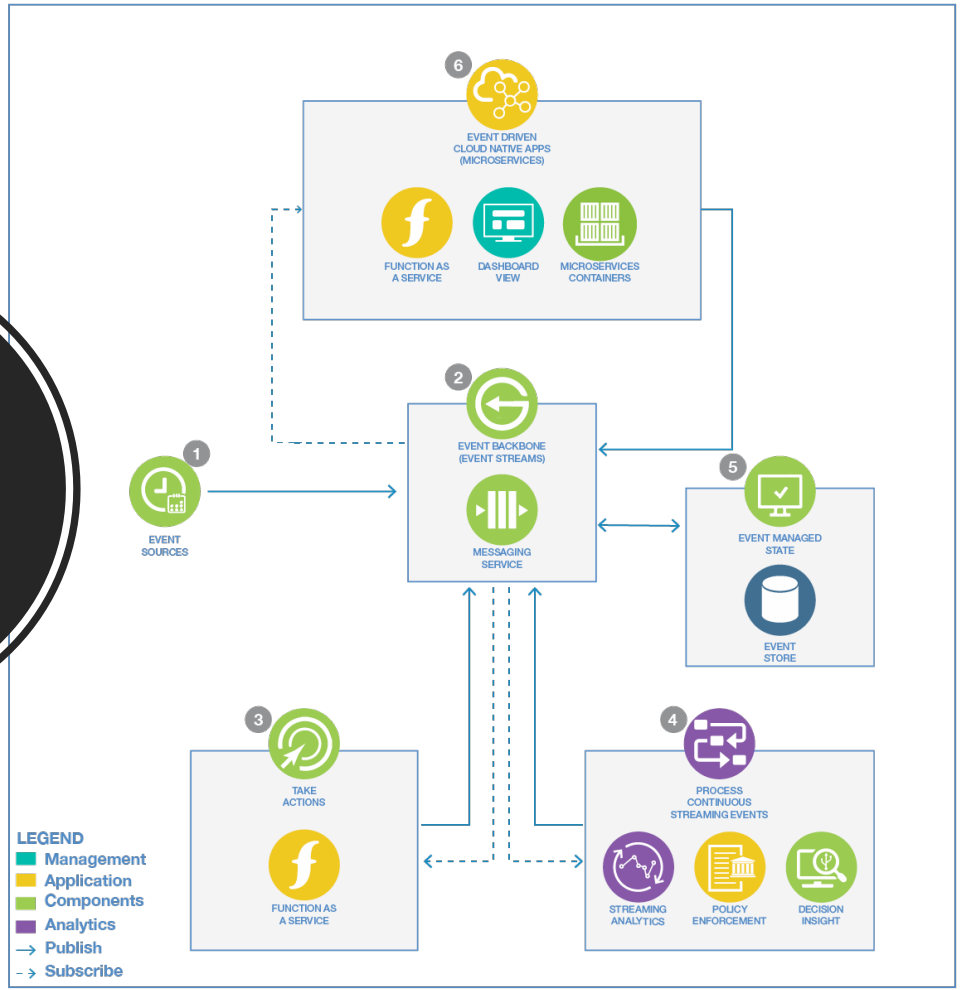


Event-Driven Architecture Reference Architecture

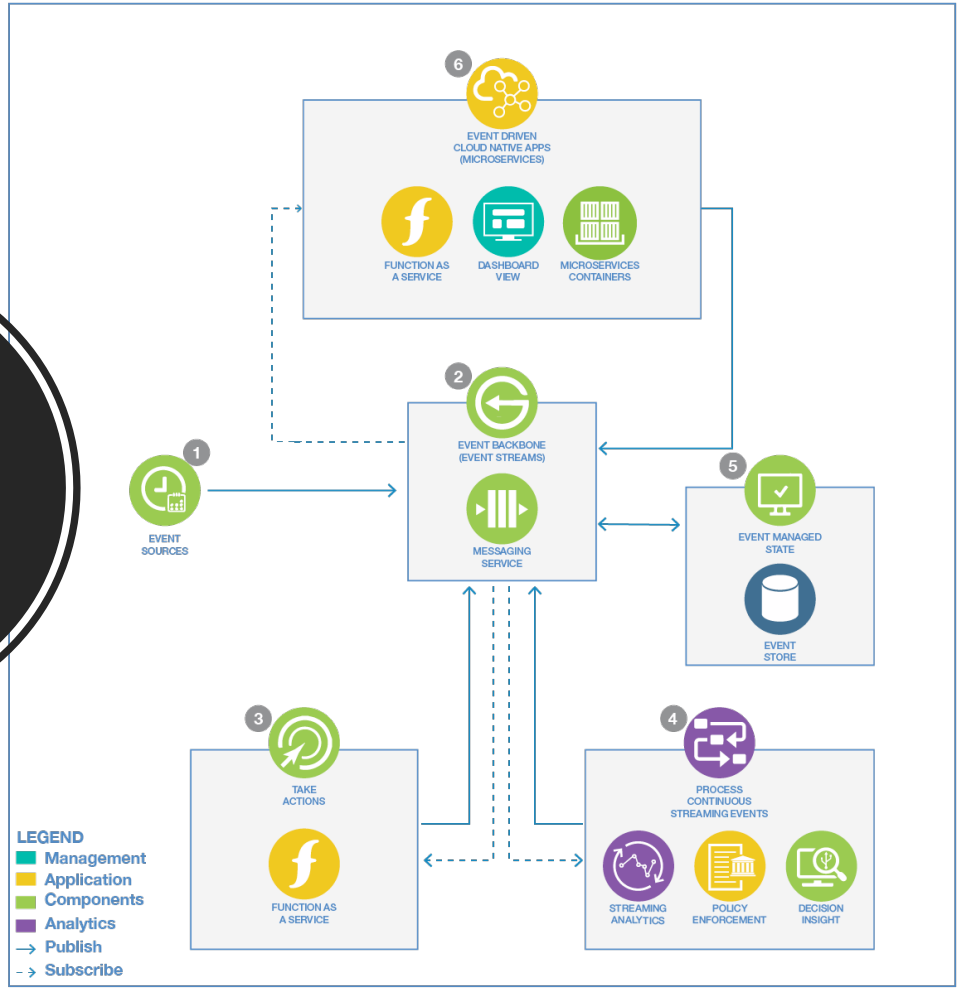
- Event-Driven Concepts
- Event-Driven Reference Architecture details
- Extended Architecture details
- Event-Driven Microservices patterns
- Designing event-driven solutions with Event and Insight Storming.

https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/

EDA and Real Time Analytics



EDA and Real Time Analytics



1. Event sources generate events and event streams from sources such as IoT devices, web apps, mobile apps, and microservices.
2. IBM Event Streams provides an event backbone that supports publish/subscribe communication, an event log, and simple event stream processing based on Apache Kafka.
3. IBM Cloud Functions provides a simplified programming model to act on an event through a serverless function-based compute model.
4. Streaming Analytics provide continuous ingest and analytical processing across multiple event streams. Decision Server Insights provides the means to act on events and event streams through business rules.
5. Event Stores provide optimized persistence (data stores), for event sourcing, Command Query Response Separation (CQRS), and analytical use cases.
6. Event-driven microservices run as serverless functions or containerized workloads that are connected by using publish/subscribe event communication through the event backbone.

IBM

Event Driven Architecture

Use Cases

Cloud Garage Solution Engineering

Intelligent Applications Pattern

Event streams are an essential data source for machine learning

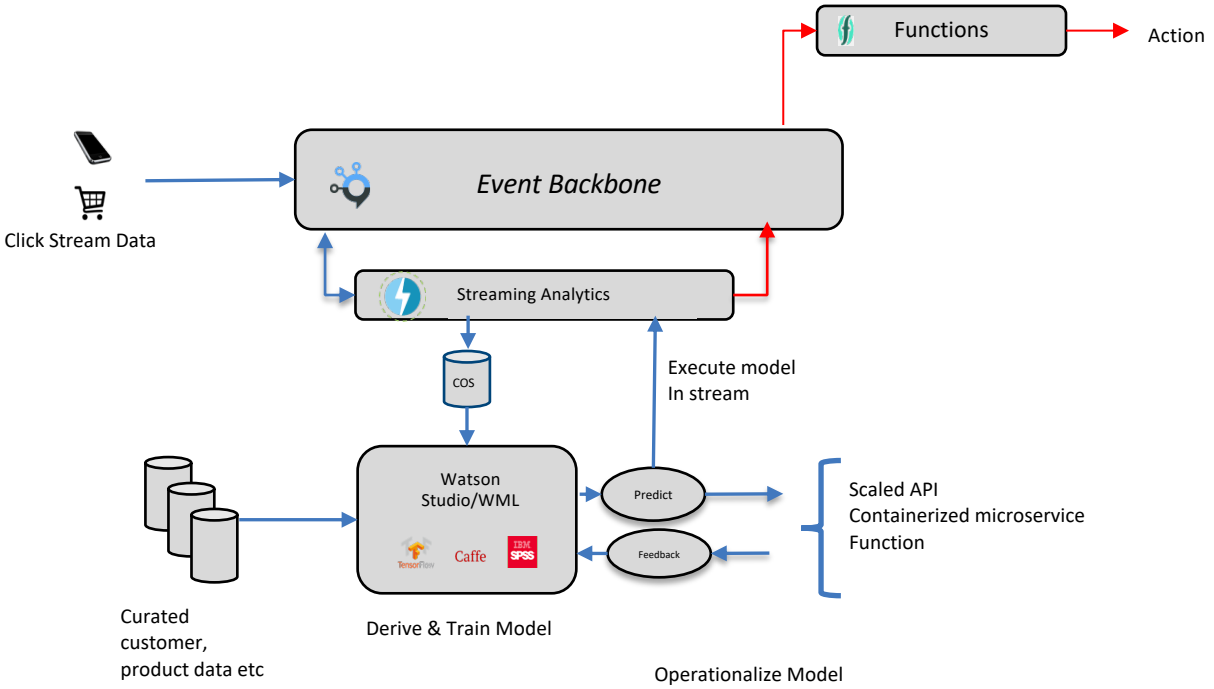
- User behavior
- Mechanical/electric signatures
- Location Information
- Environment information

Event Backbones connects event streams

Streaming analytics processes event streams and collates event data for data scientist to analyze, derive and train models from.

ML models are *operationalized* for consumption by developers

- Deployed in the streaming analytics for execution with the event stream
- Exposed as API's or containerized work loads for consumption by application developers



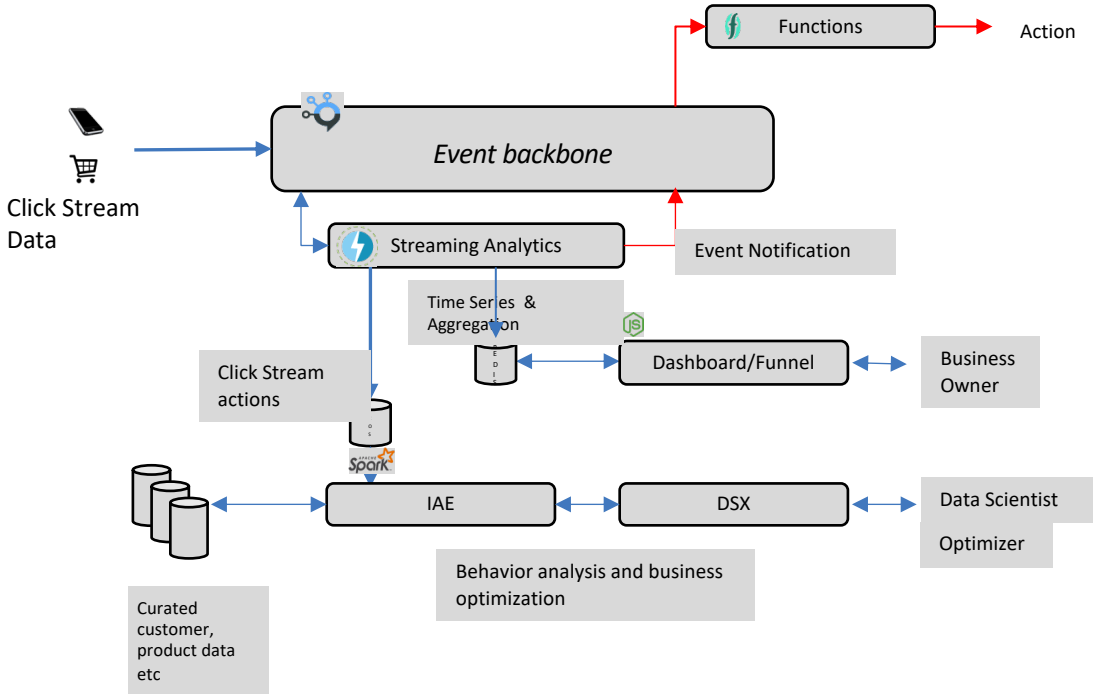
Click Stream Pattern

Click Stream Collector passes click stream events to the event backbone

Streaming analytics processes the events

- Saves aggregated click stream actions as files in object storage for later analysis of user behavior
- Analyzes click stream data to understand current state. Eg for ecommerce applications identify customers/items/value at each stage of the sales funnel.
- Publish an event to take action, eg. High number of Customer cart abandonments over short time period.

Data Scientists process the extracted click stream data to analyze to look for struggles and. optimization opportunities.



IOT solution patterns

IOT Platform provides device connectivity and management

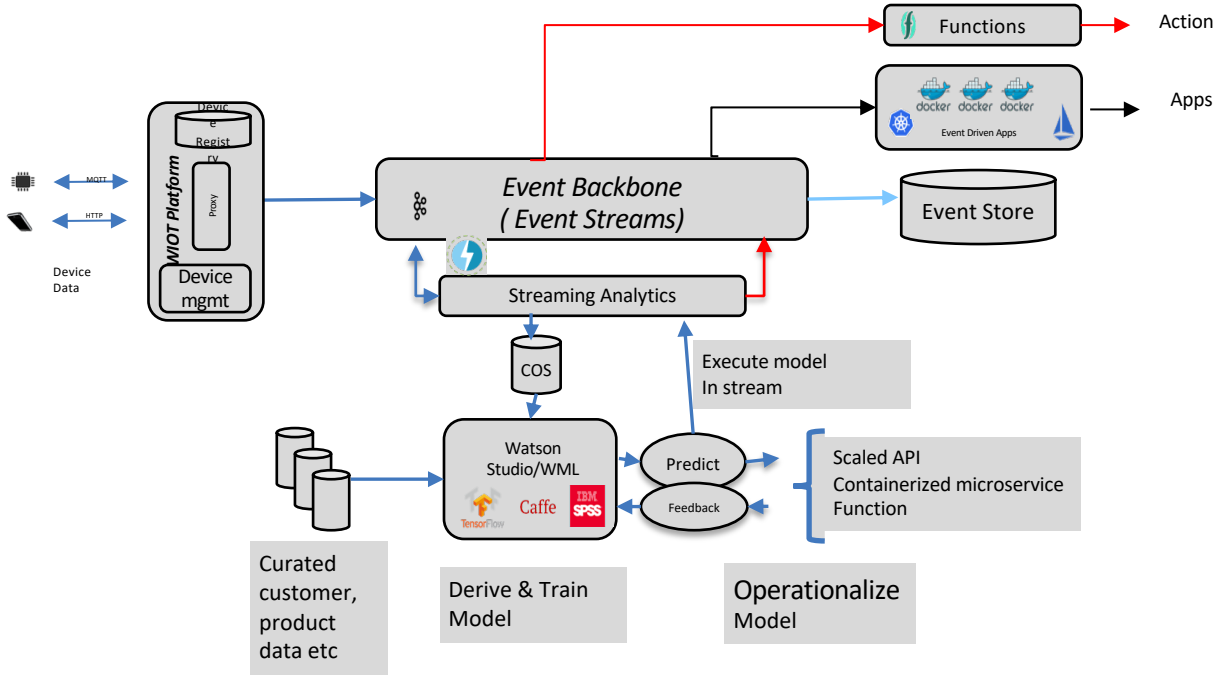
IOT platform published interesting events to event backbone

Apply the event patterns to develop IOT application with

- Real time intelligence
- Event Stores
- Event Driven Microservices
- Event Stream processing

Event patterns combine to simplify complex solutions

Enables applications to be developed in manageable step



Event Driven Architecture Is Broadly Applicable

Telecommunications
- Cognitive call center
- CDR processing
- Social analysis
- Churn prediction
- Geo mapping

Transportation
- Intelligent traffic management
- Automotive telematics
- Trend Detection

Energy & Utilities
- Call Deflection and Optimization
- Transactive control
- Phasor Monitoring Unit
- Down hole sensor monitoring

Health & Life Sciences
- ICU monitoring
- Epidemic early warning system
- Remote healthcare monitoring
- HIPAA, PCI Compliance
- Transcription Services

Natural Systems
- Wildfire management
- Water management

Law Enforcement, Defense and Cyber Security
- Real-time multimodal surveillance
- Situational awareness
- Cyber security detection
- Data Leakage

Stock market and Banking
- Impact of weather on securities prices
- Analyze market data at ultra-low latencies
- Momentum Calculator
- Regulatory Monitoring and Audits

Insurance and Fraud prevention
- Detecting multi-party fraud
- Real time fraud prevention
- Know your customer (360)
- Next Best Offer

e-Science
- Space weather prediction
- Detection of transient events
- Synchrotron atomic research
- Genomic Research

IBM

Event Driven Architecture

Core Concepts

Cloud Garage Solution Engineering

Concepts

- **Events**
  - Events are notifications of change of state.
  - Typically, events represent the change of state of something of interest to the business.
  - Events are records of something that has happened.
  - Events can't be changed, that is, they are immutable. (***We can't change the past!***).
- **Event streams**
  - An event stream is a continuous unbounded series of events.
  - The start of the stream may have occurred before we started to process the stream.
  - The end of the stream is at some unknown point in the future.
  - Events are ordered by the point in time at which each event occurred.
  - When developing event driven solutions, you will typically see two types of event streams:
    - Event streams whose events are defined and published into a stream as part of a solution.
    - Event streams that connect to a real-time event stream, for example from an IOT device, a voice stream from a telephone system, a video stream, or ship or plane locations from global positioning systems.

Concepts

- **Event sources**
  - In an event-driven architecture, event producers and event consumers are the interaction points with events.
  - Similarly, microservices play the role of both event producers and event consumers in an event-driven architecture, with the events being passed as the communication payload between them.
  - However, as you look at the wider opportunities that being event-driven offers, you must consider event sources that come from beyond the application itself, often providing business relevance or enabling valuable insights into things that directly impact your business.
  - A list of common event sources would include:
    - IOT devices or sensors showing device status changes
    - Clickstream data from web or mobile applications
    - Weather alerts
    - Mobile applications (HTTP to Backend for Frontend service and then to topic)
    - Geospatial data
    - Social media feeds
    - Real-time voice feeds

Concepts

- **Loose coupling**
  - Loose coupling is one of the main benefits of event-driven processing.
  - It allows event producers to emit events without any knowledge about who is going to consume those events.
  - Likewise, event consumers don't need to be aware of the event emitters.
  - Because of this, event consuming modules and event producer modules can be implemented in different languages or use technologies that are different and appropriate for specific jobs.

  - Loose coupling, however, does not mean "*no coupling*".
  - An event consumer consumes events that are useful in achieving its goals and in doing so establishes what data it needs and the type and format of that data.
  - The event producer emits events that it hopes are understood and useful to consumers thus establishing an implicit contract with potential consumers.

Concepts

- **Cohesion**
  - Cohesion is the degree to which related things are encapsulated together in the same software module.
  - For the purposes of this event-driven architecture-based discussion, a module is defined as an independently deployable software unit that has high cohesion.
  - Cohesion is strongly related to **coupling** in the sense that a highly cohesive module communicates less with other modules, thus reducing the number of events and event types in the system.
  - The less frequently modules interact with each other, the less coupled they are.
  - Achieving cohesion in software while optimizing module size for flexibility and adaptability is difficult, but something to strive for.
  - Designing for cohesion starts with a holistic understanding of the problem domain and good analysis work.

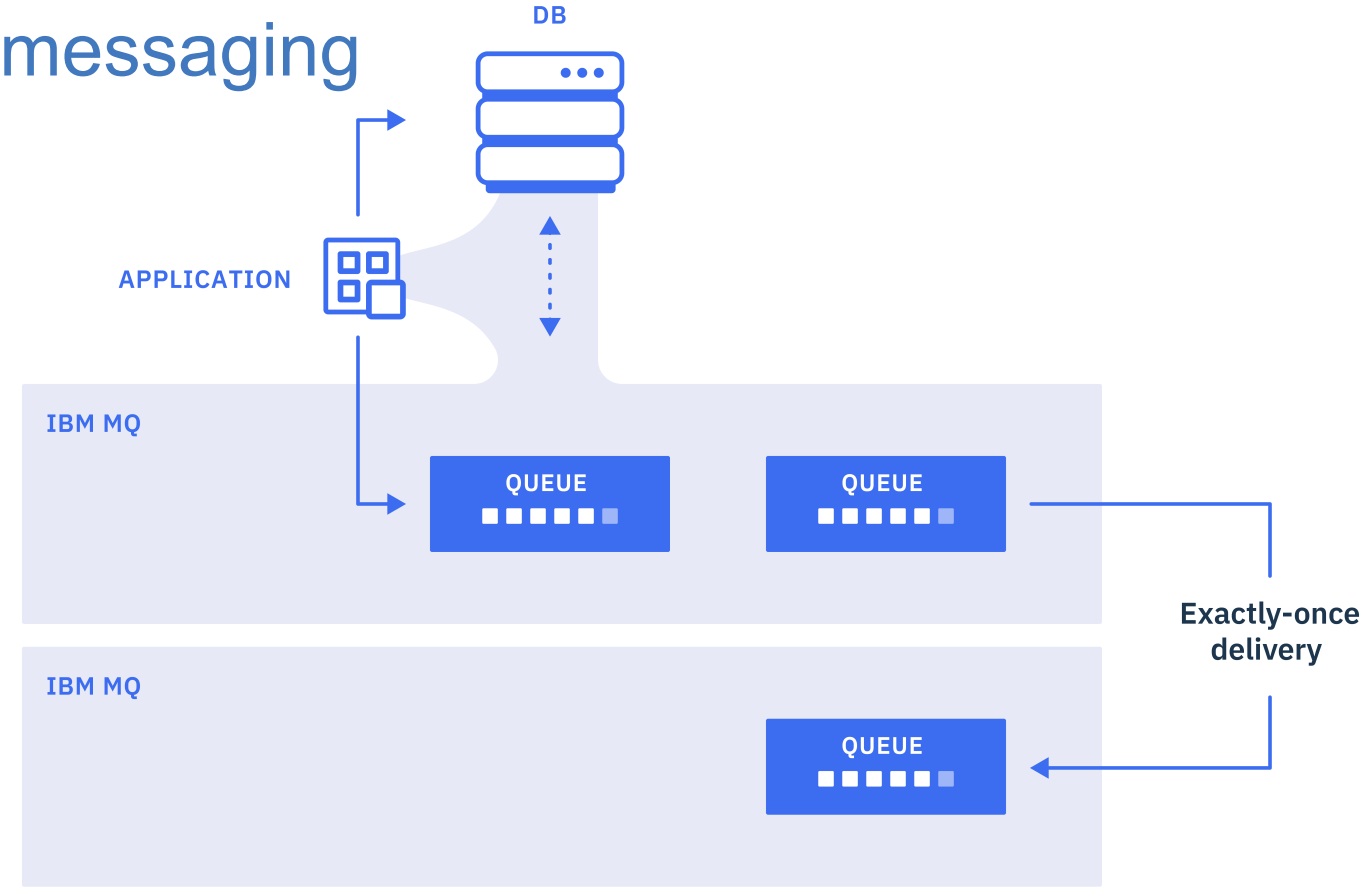
IBM

Event Driven Architecture

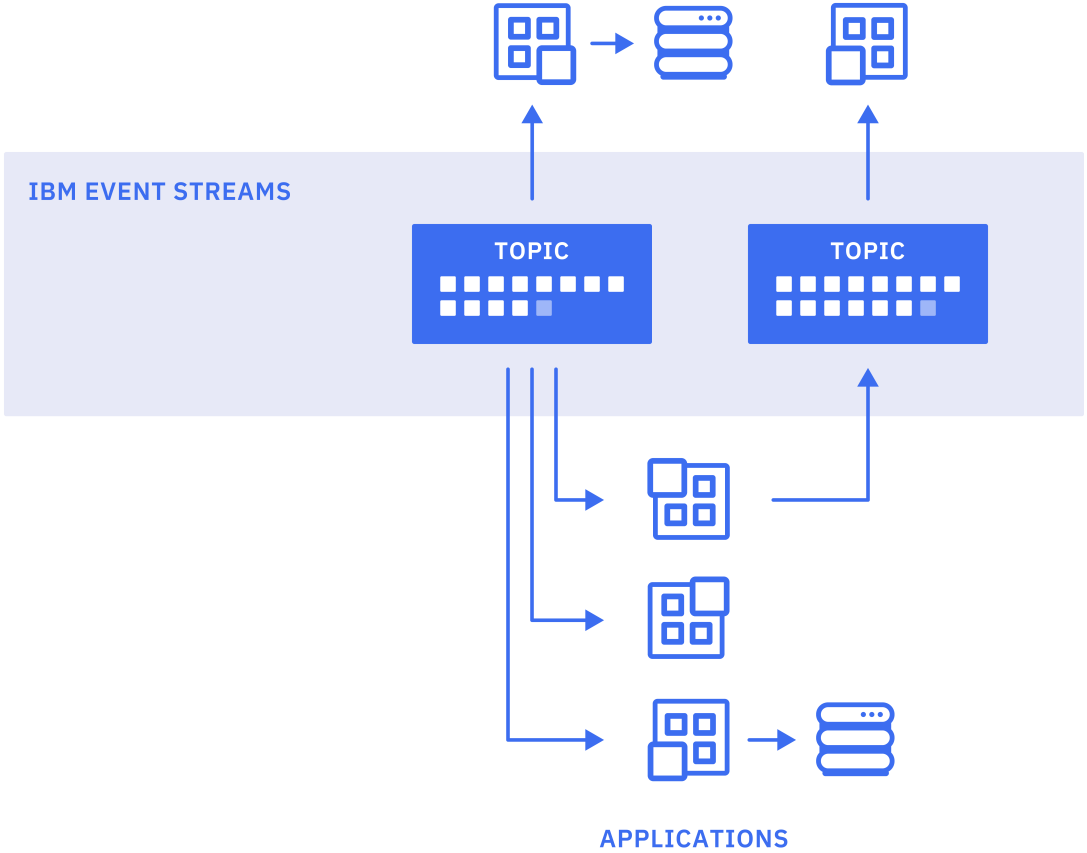
Event Streaming vs Messaging

Cloud Garage Solution Engineering

A history of messaging



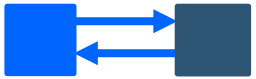
A stream of events



Two distinct communication styles

MESSAGE QUEUING

Transient data persistence

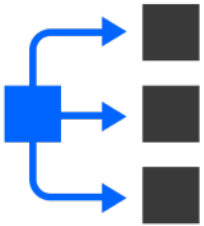


Request / Reply

Targeted delivery

EVENT STREAMING

Stream history

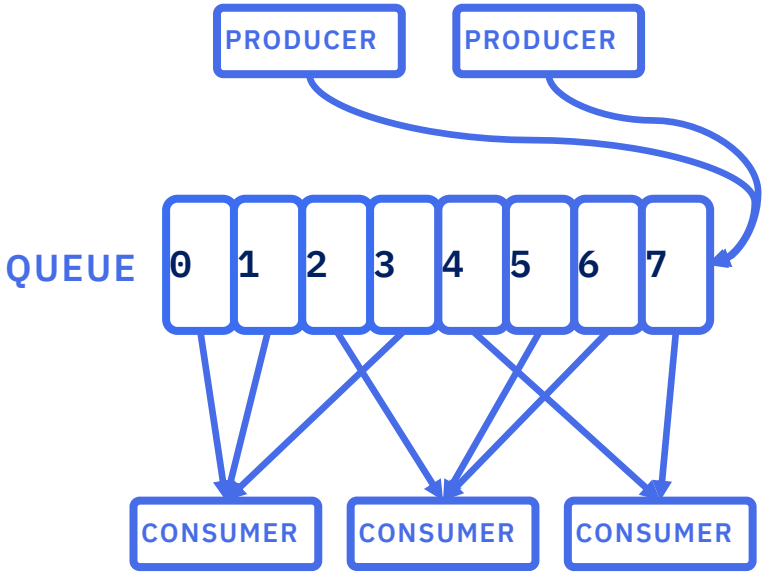


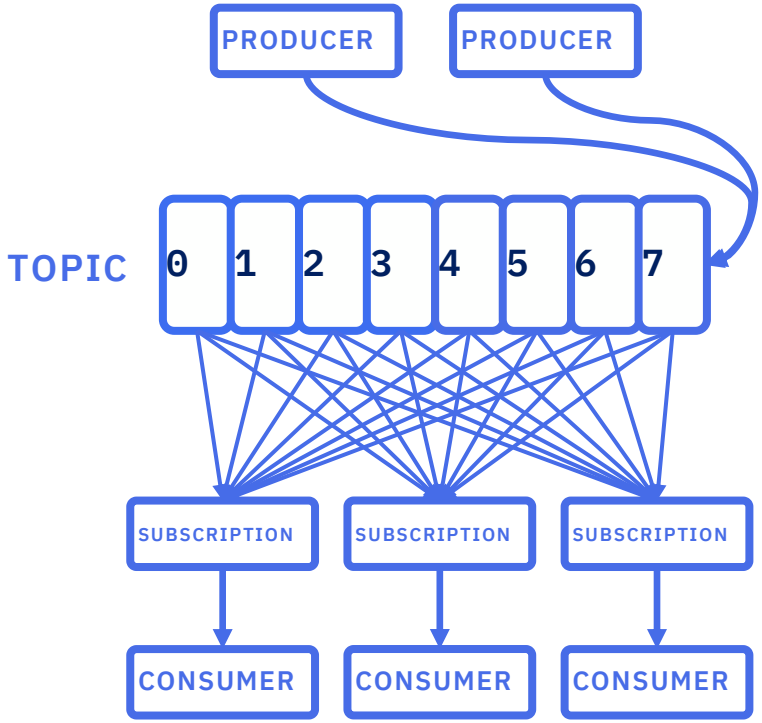
Scalable consumption

Immutable data

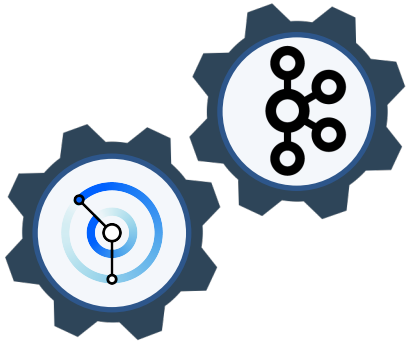
https://ibm-cloud-architecture.github.io/refarch-eda/concepts/events-versus-messages/

MQ: Queues and Topics





How do I make sense of this?

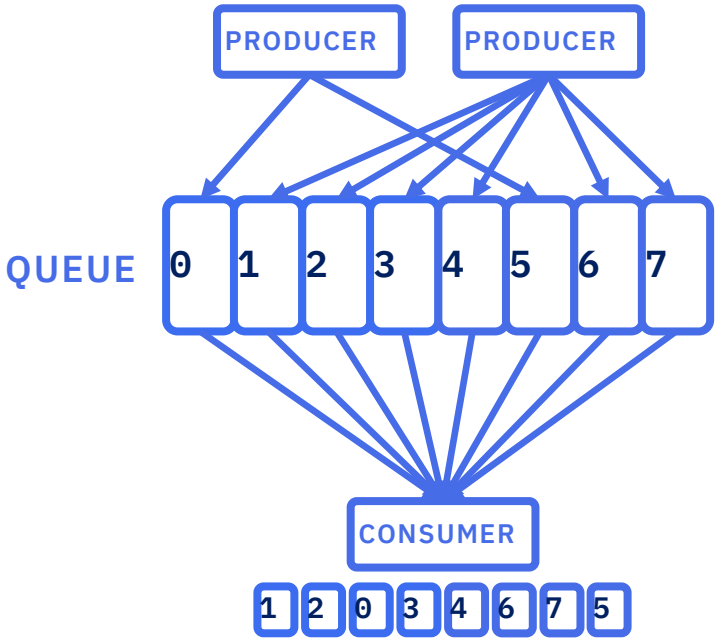


- Queues are good for:
  - Easy workload distribution with multiple consumers
  - Out of order message acknowledgement

- Event streams are good for:
  - Stream history because consumption is always non-destructive
  - Ordering with workload distribution, but the consumers must keep up

- You might well want to use them together
  - Queues for commands, event streaming for events

Transactional message visibility in MQ

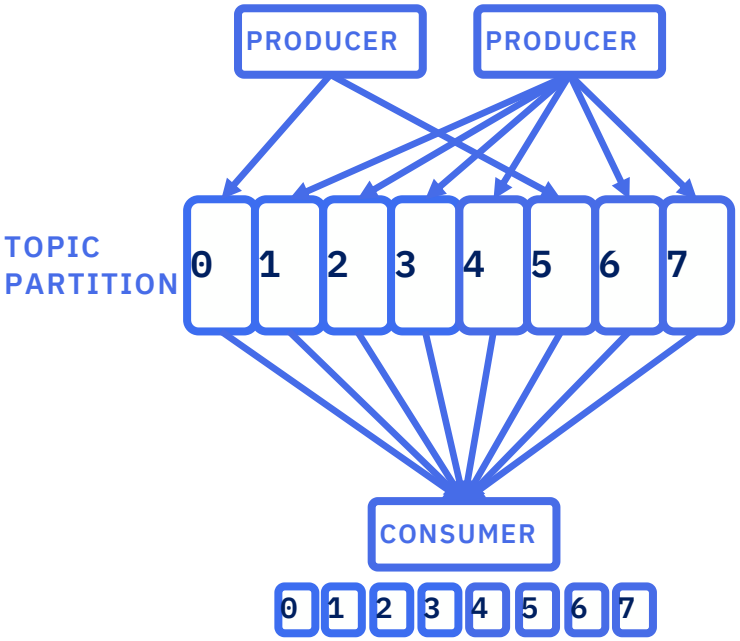


Ordered when produced

Uncommitted are invisible

Uncommitted are skipped

Transactional message visibility in Kafka



Ordered when produced

Consumer gets same order

Two isolation levels
- Read uncommitted
- Read committed

IBM

Event Driven Architecture

IBM Event Streams - Apache Kafka for the Enterprise

Cloud Garage Solution Engineering

Apache Kafka is an Open-Source Streaming Platform

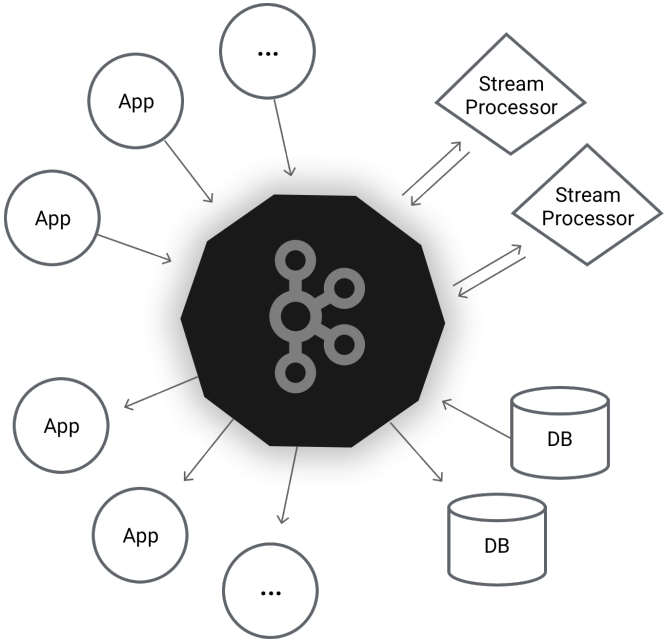
PUBLISH & SUBSCRIBE
Read and write streams of data like a messaging system. Capable of message replay. Durability, and horizontal scaling

PROCESS
Write scalable stream processing applications that react to events in real-time.

STORE
Store streams of data safely in a distributed, scalable, replicated, fault-tolerant cluster. (Event sourcing)

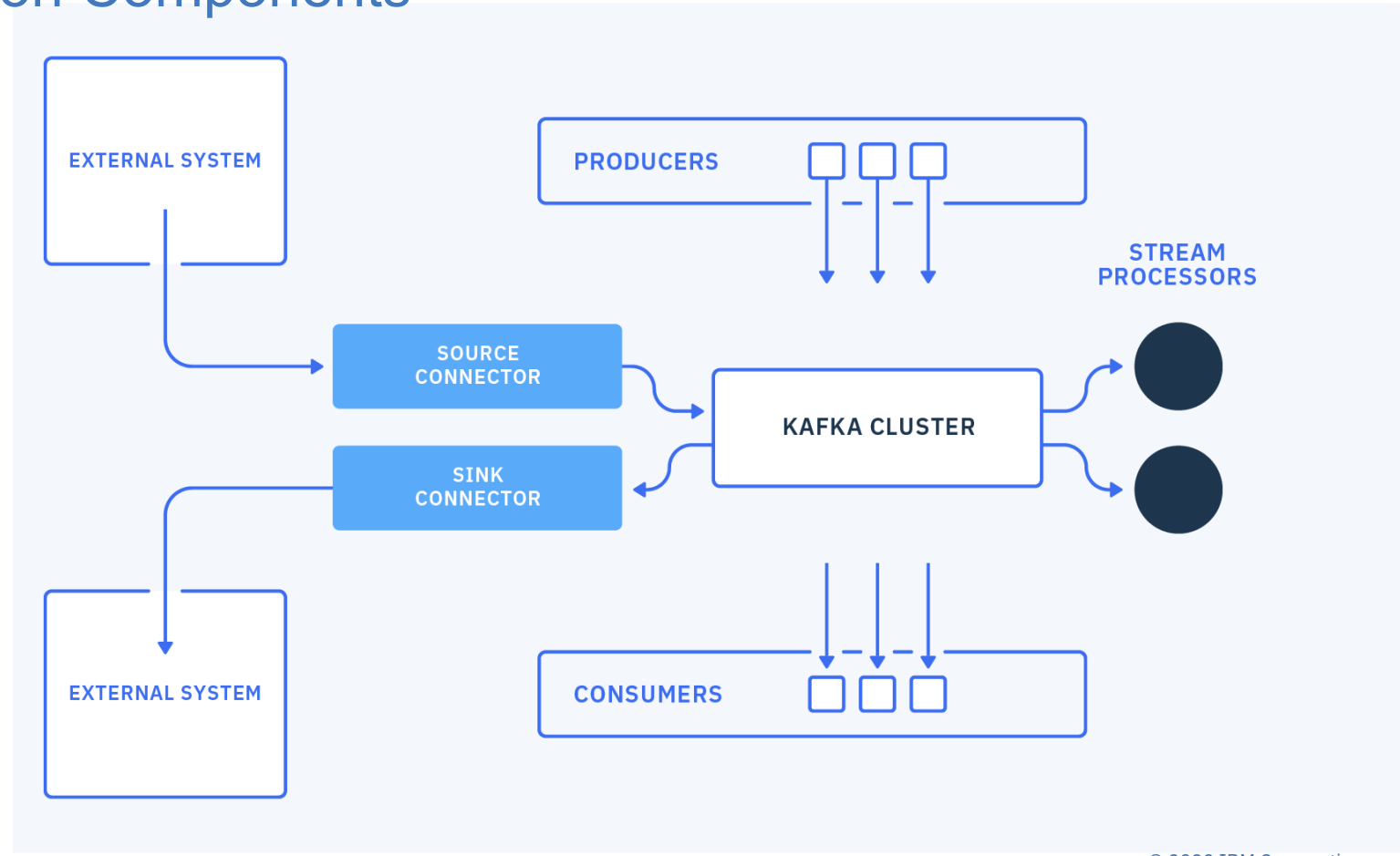
https://kafka.apache.org/

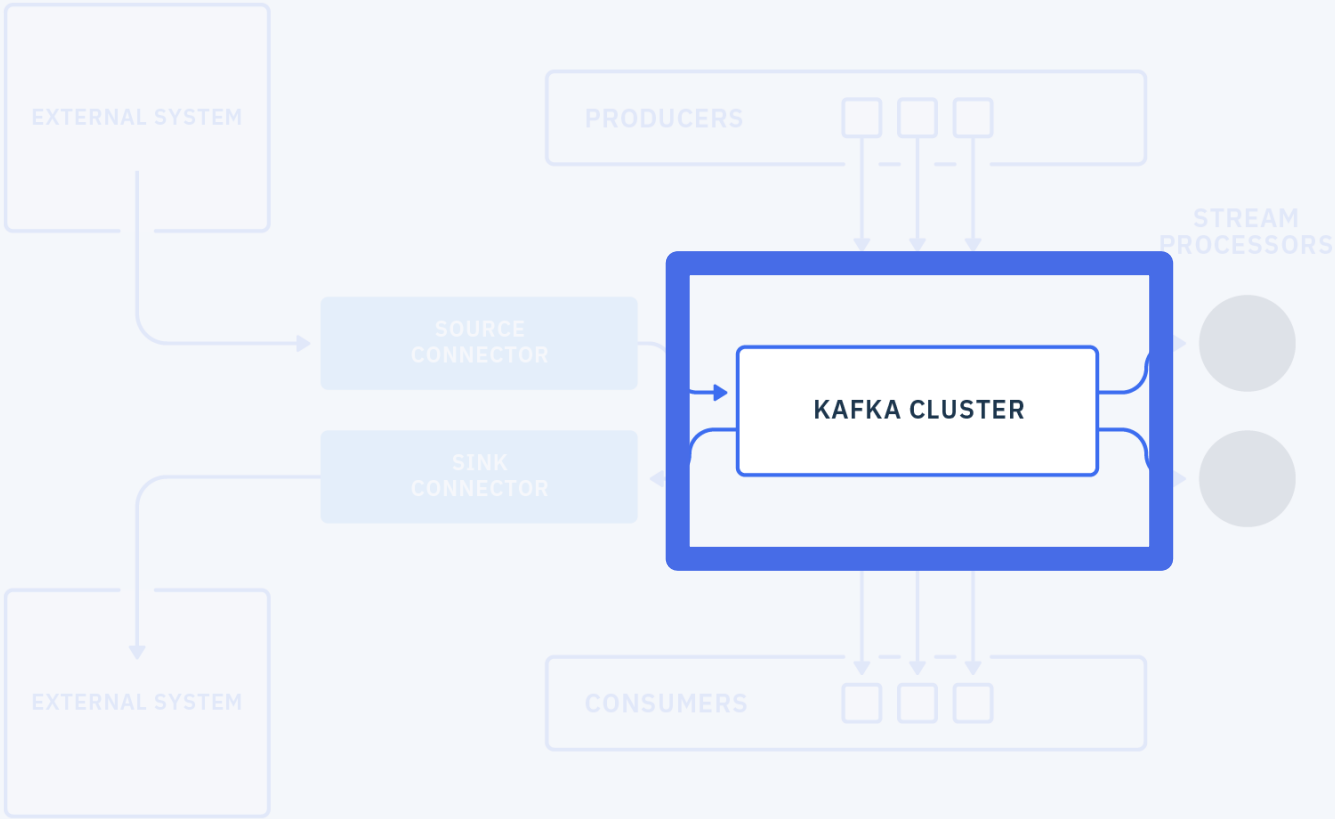
IBM Cloud Architecture Kafka notes



APACHE kafka
A distributed streaming platform

Solution Components

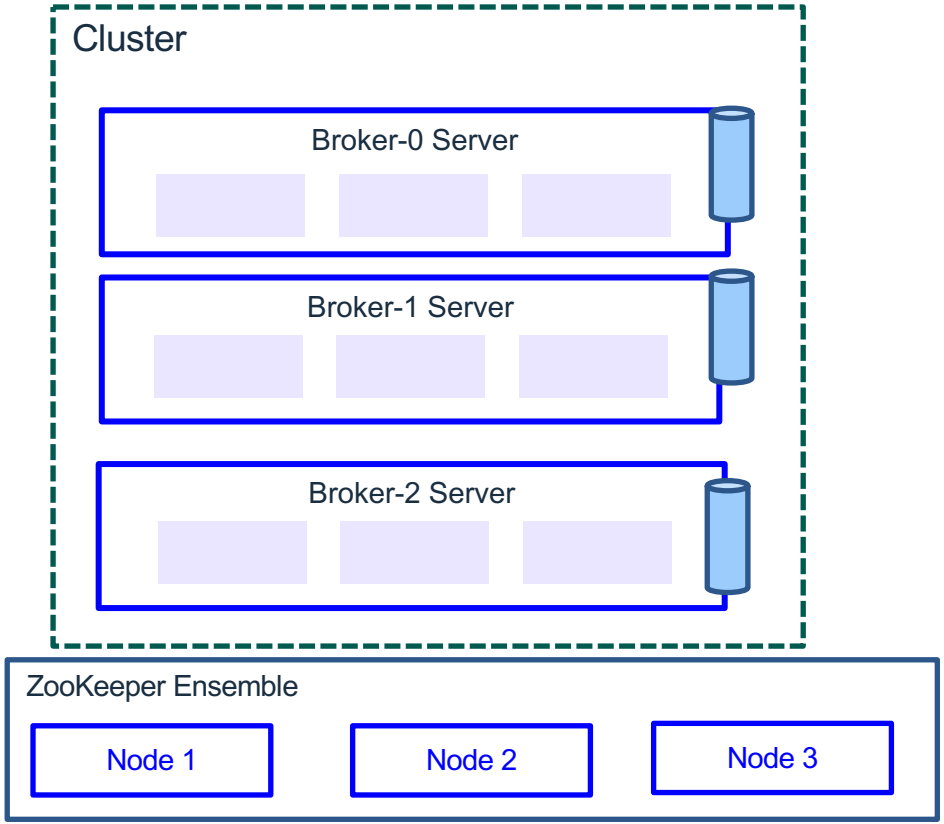




Brokers and cluster

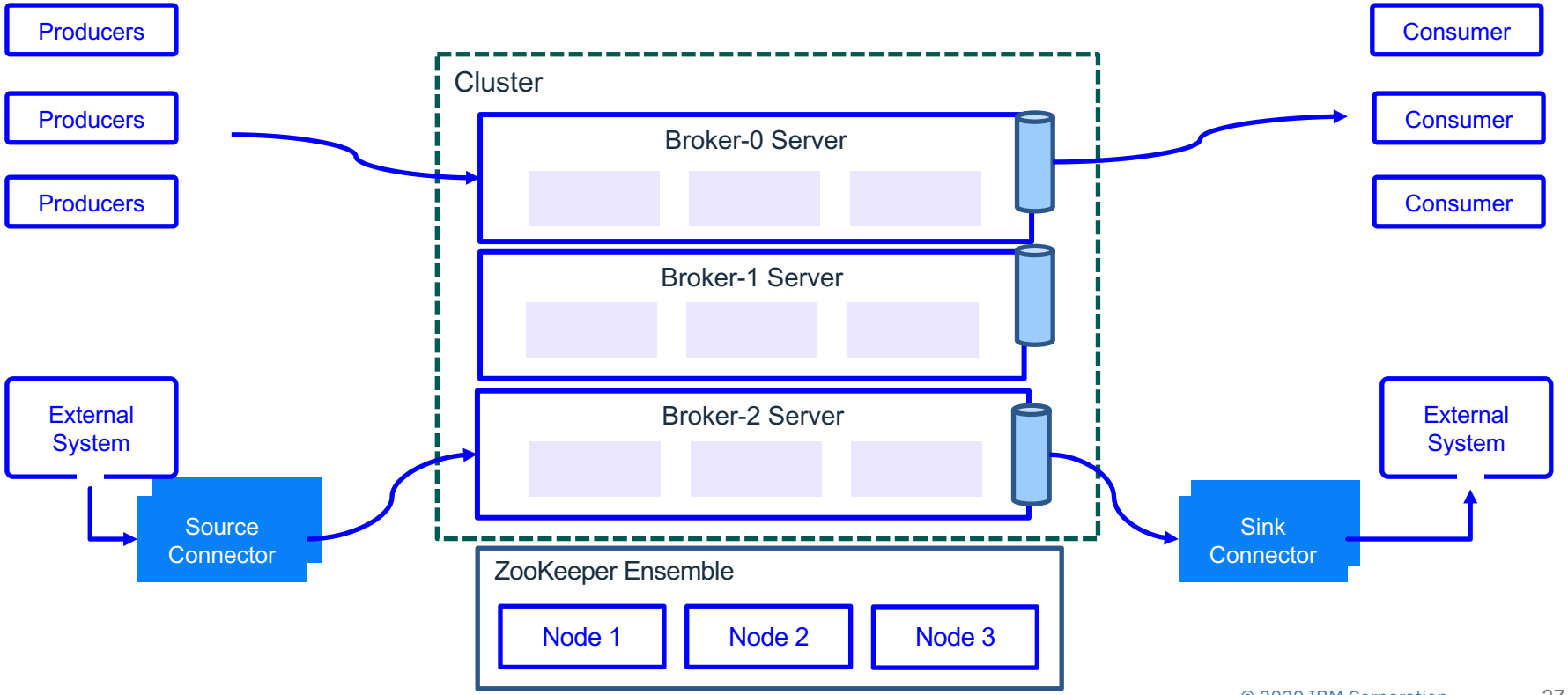
- A Kafka cluster consists of a set of brokers

- A broker is an instance of Kafka uniquely identified in the cluster
- All brokers in a cluster know about all other brokers
- All information is continuously written to disk
- A connection to a broker is called the bootstrap broker
- A cluster durably persists all published records for the retention period

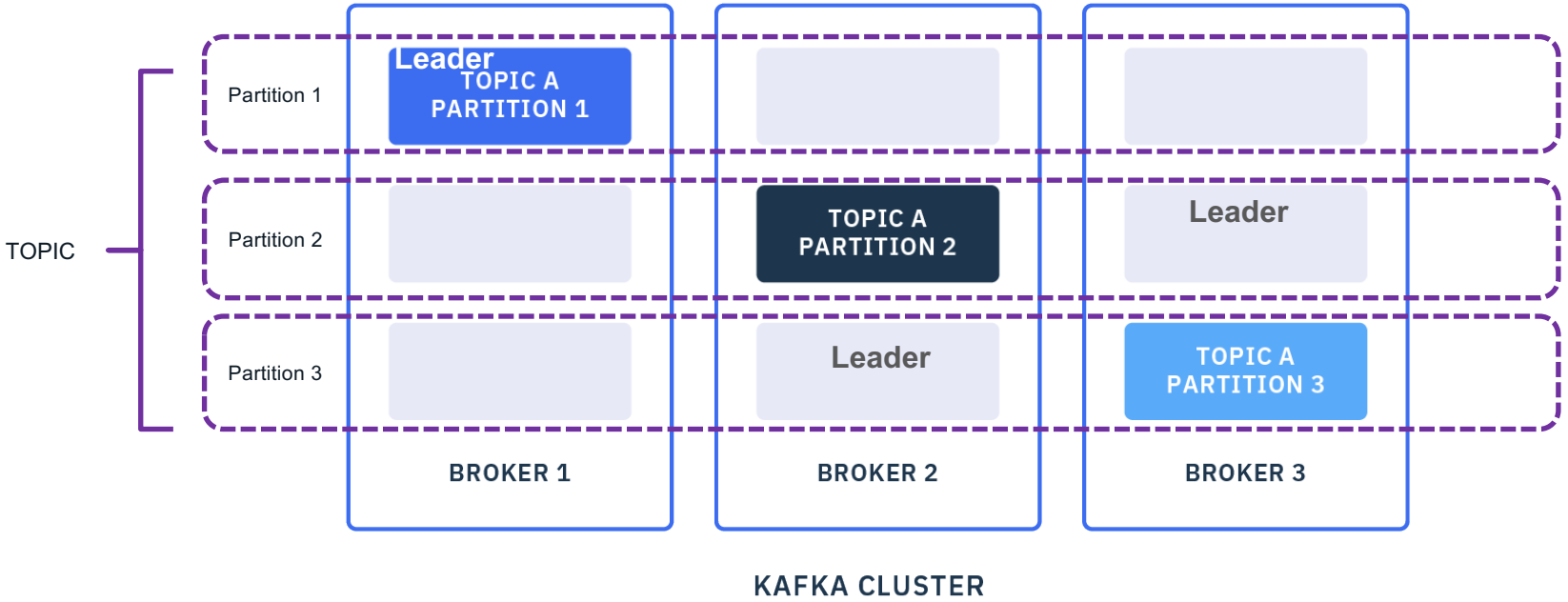


https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-overview/#brokers

Kafka Architecture



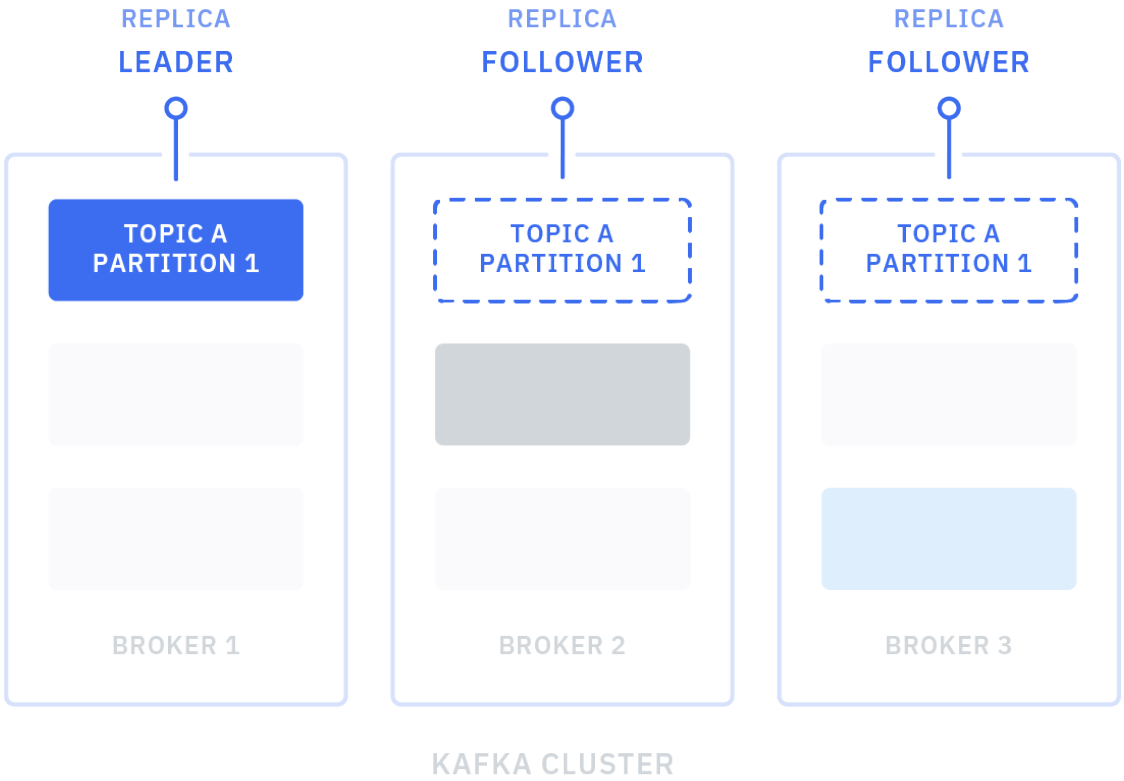
Partitions



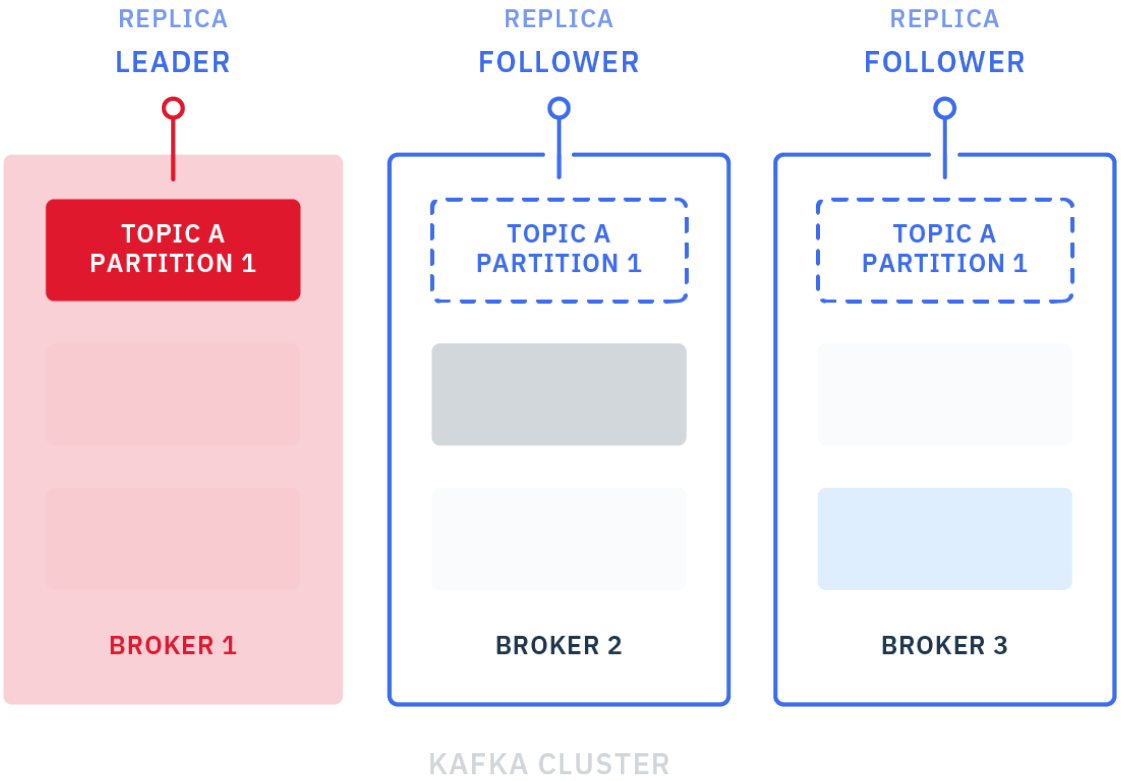
https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-overview/#architecture

Replication

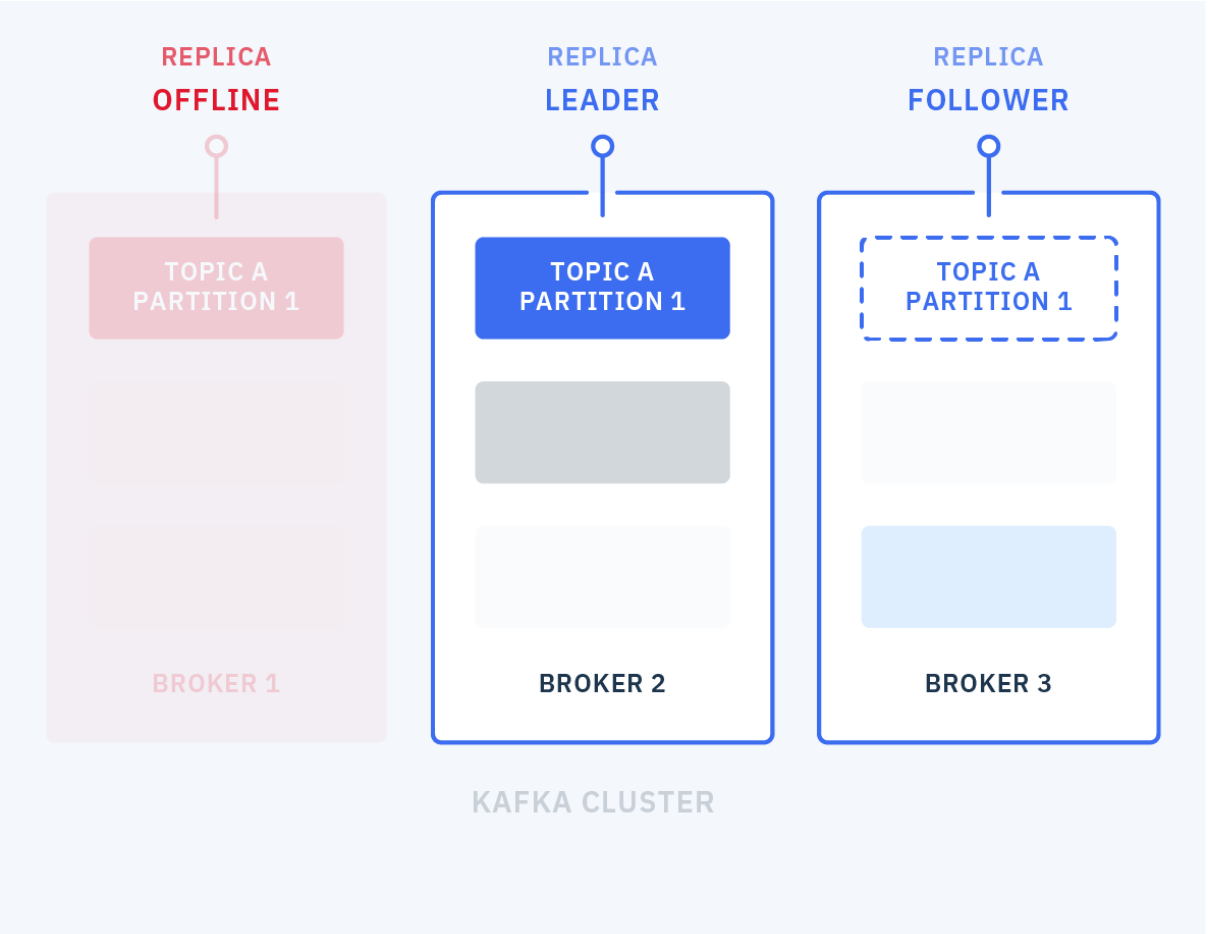
Partition is what is replicated



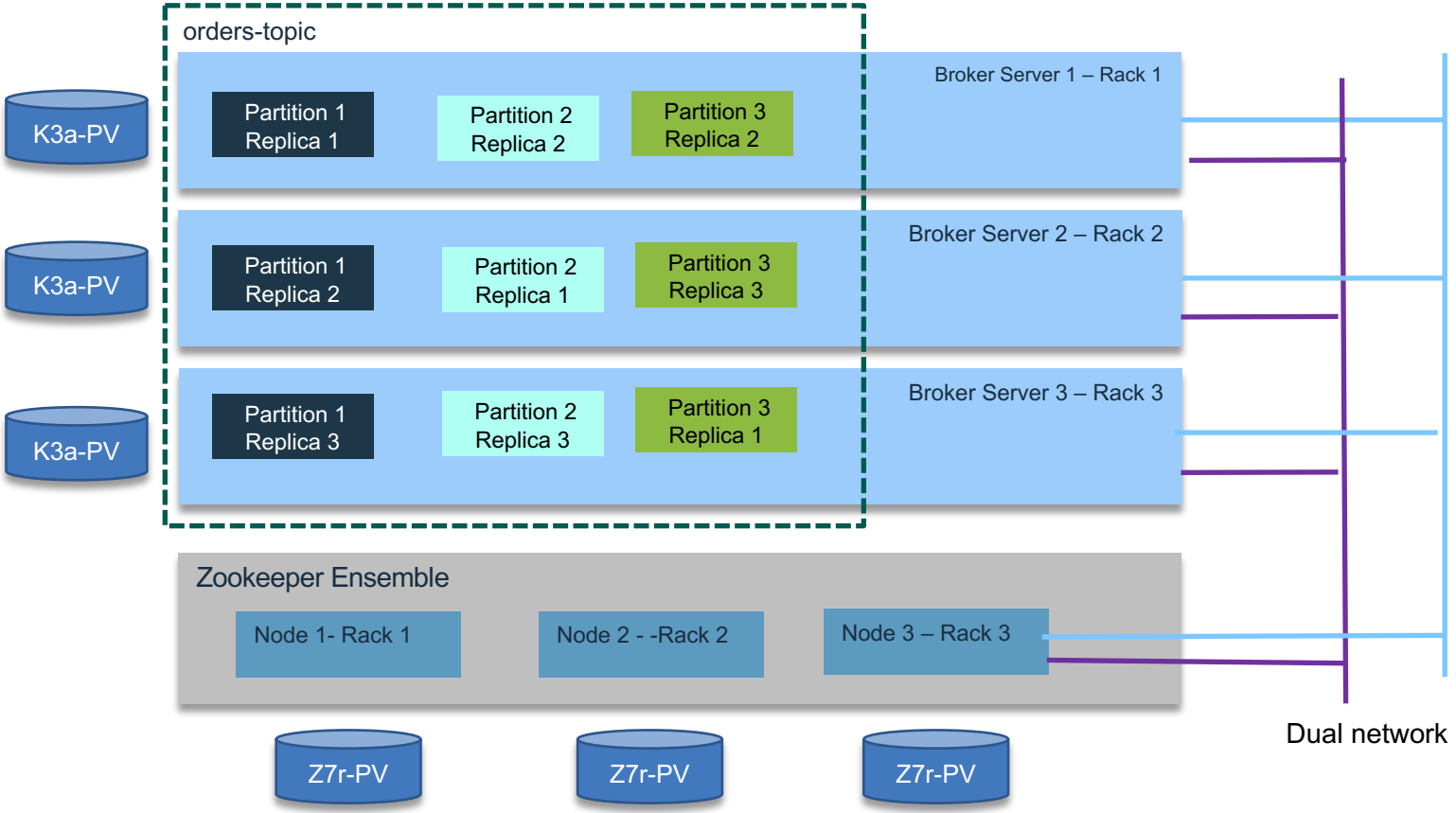
Replication



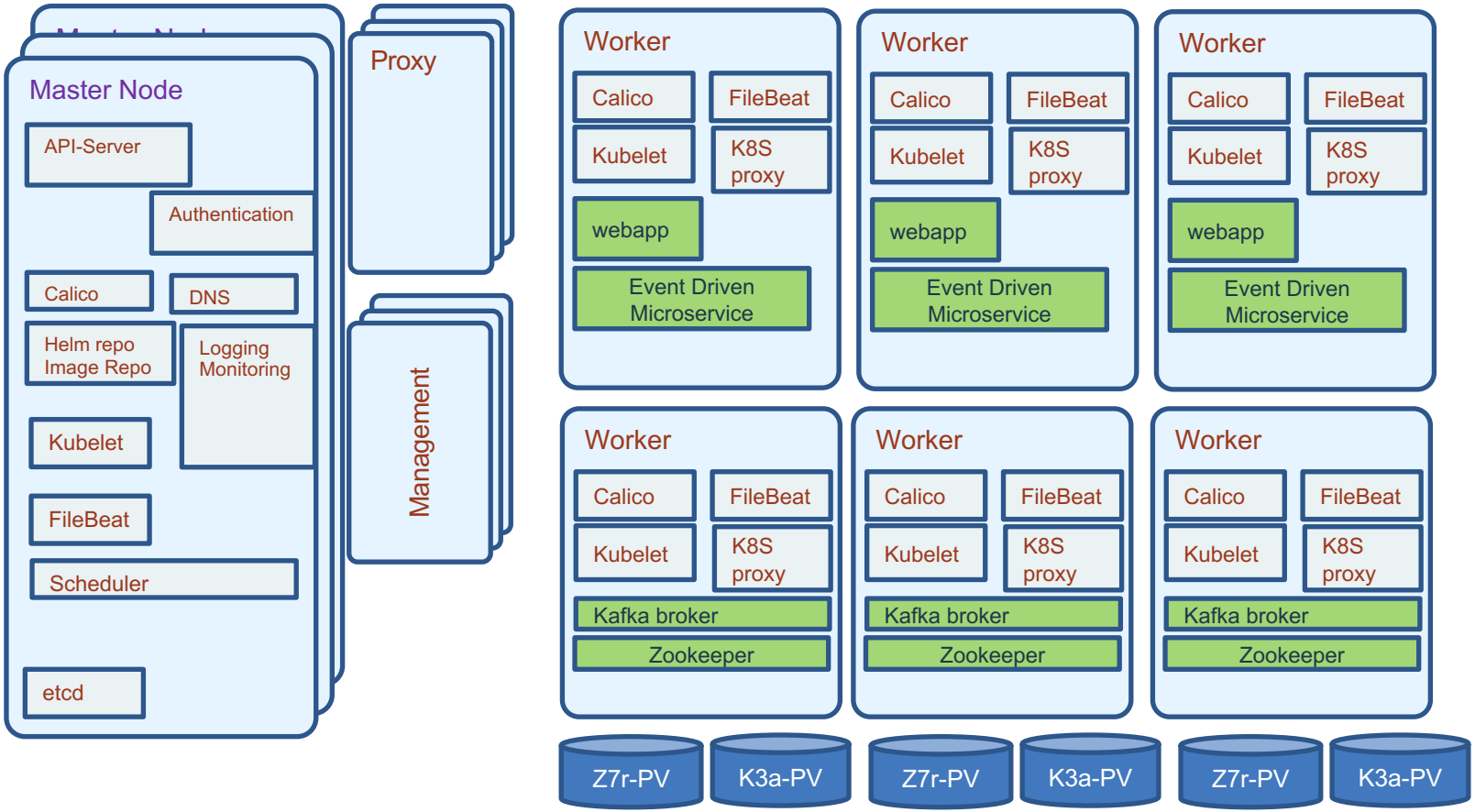
Replication

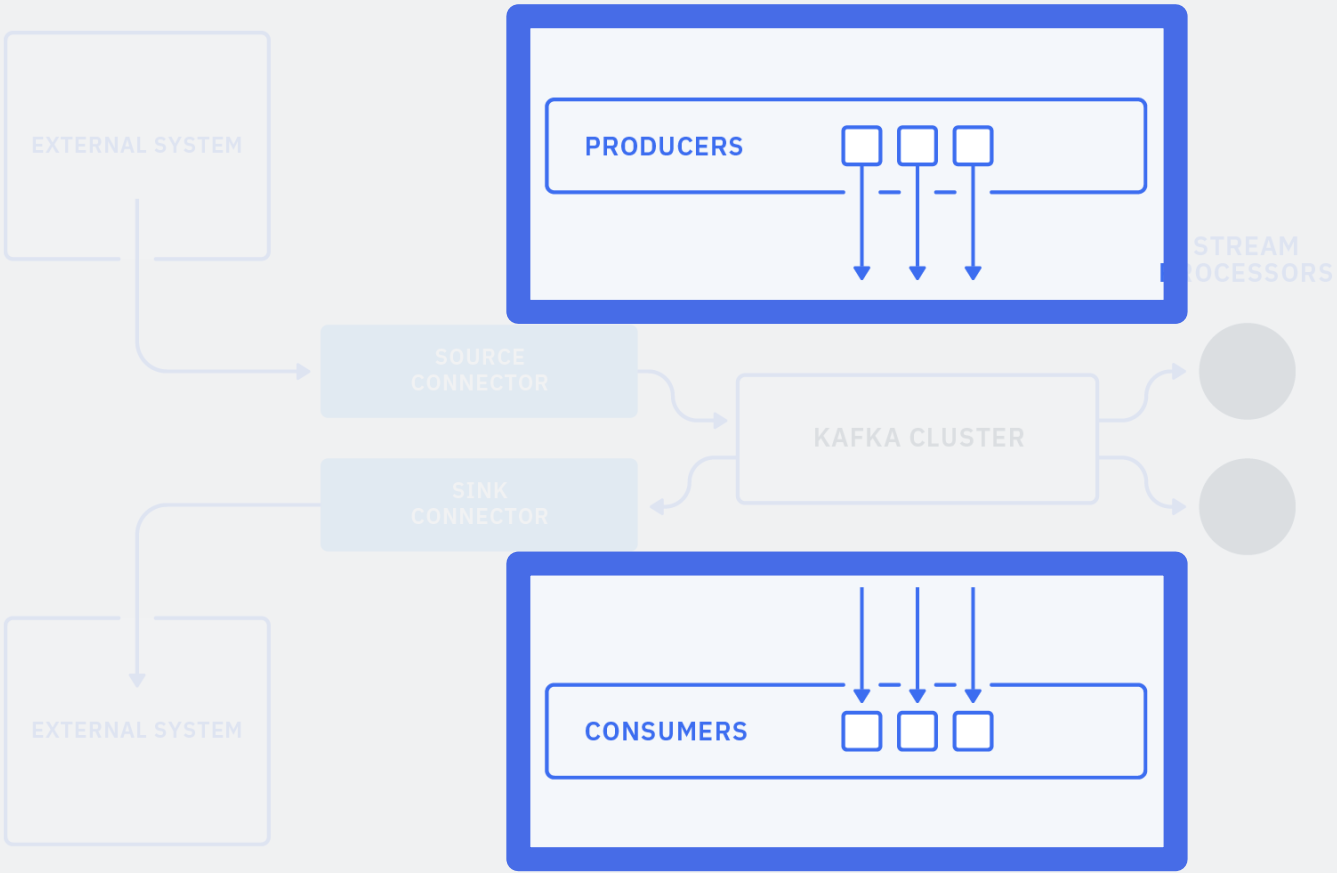


Intra-cluster High Availability

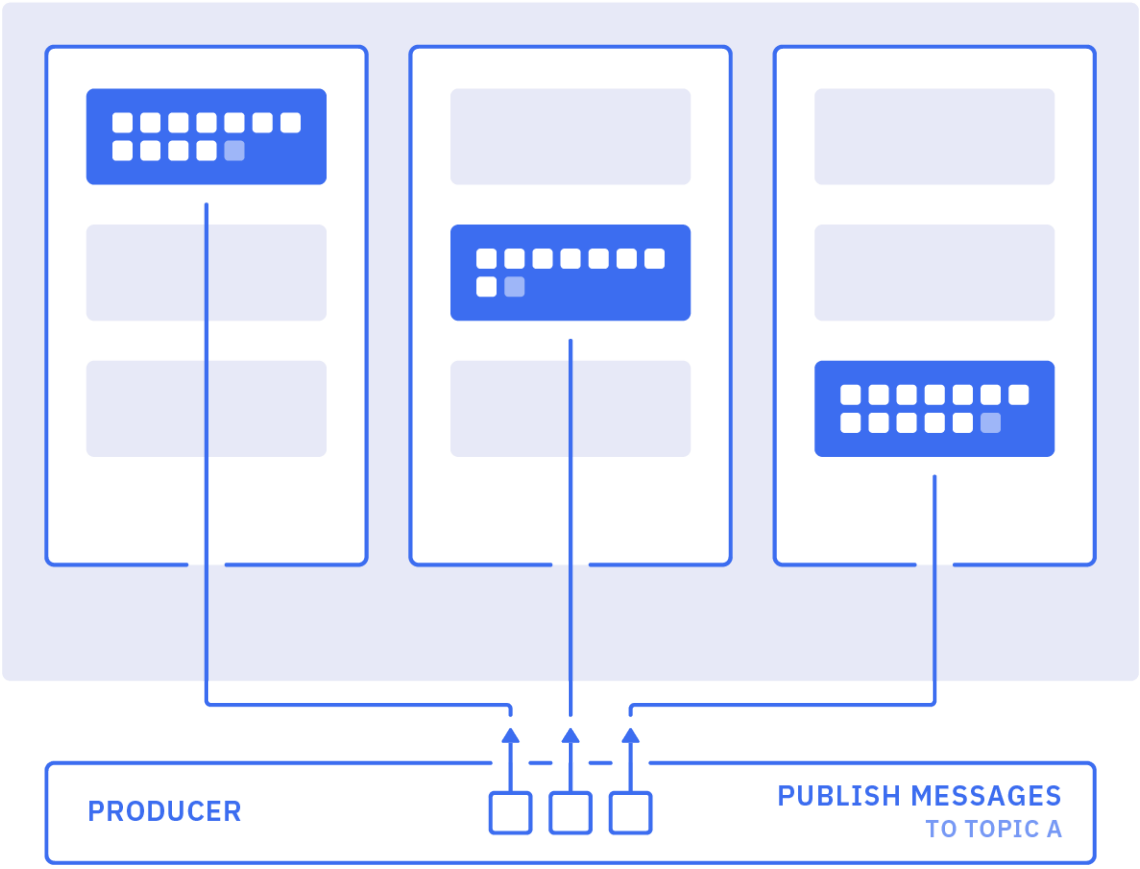


Kafka on Kubernetes

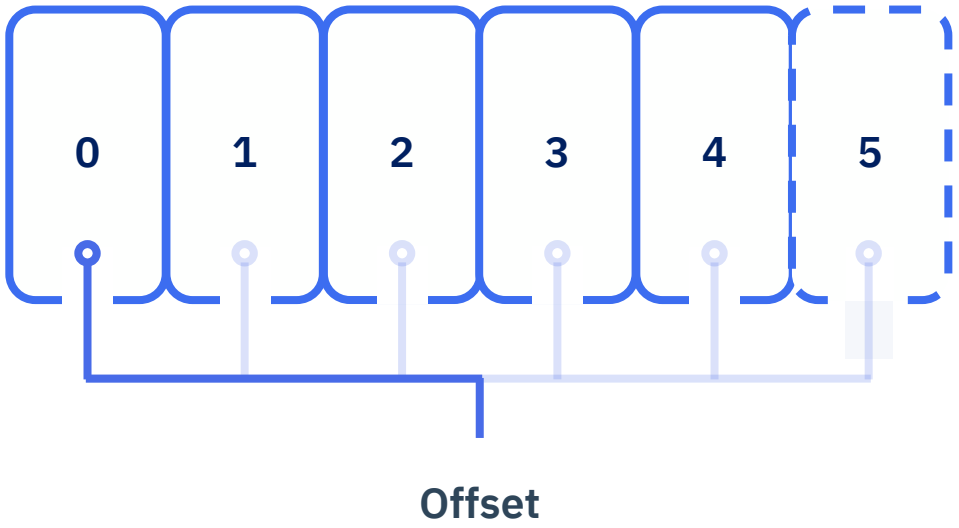




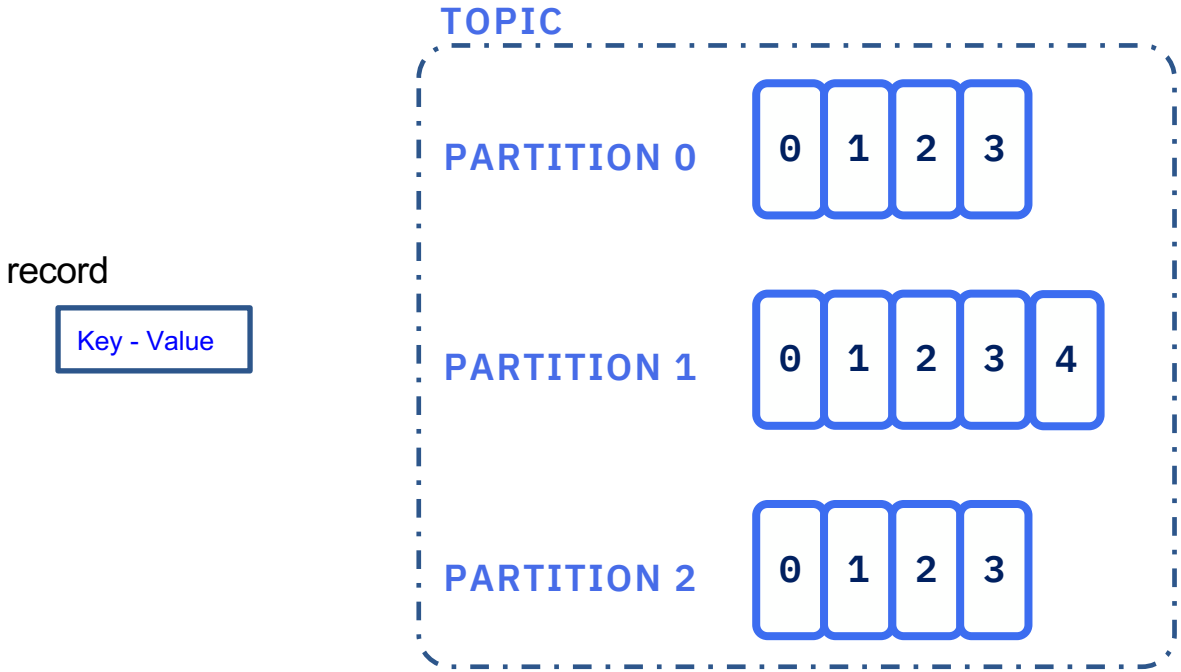
Producers



Topics

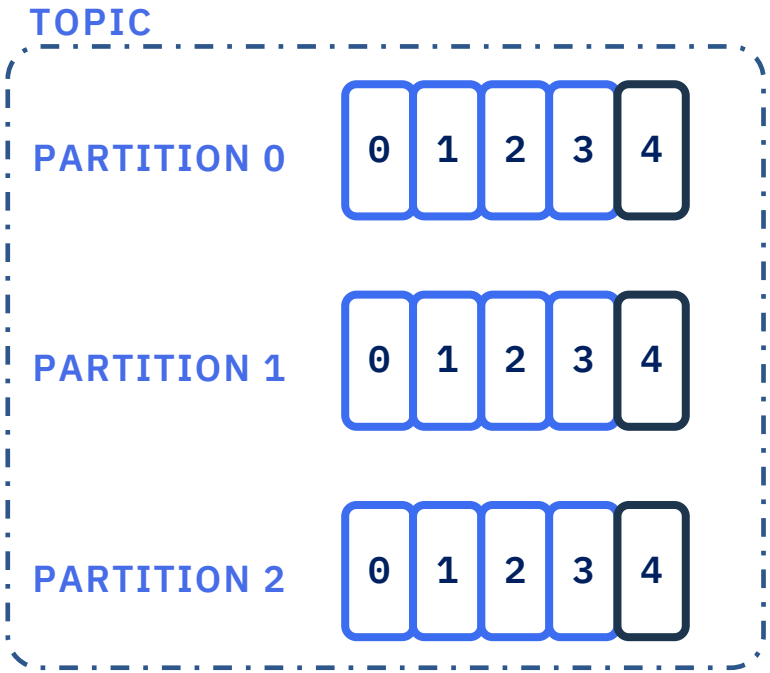


Topics and Keys



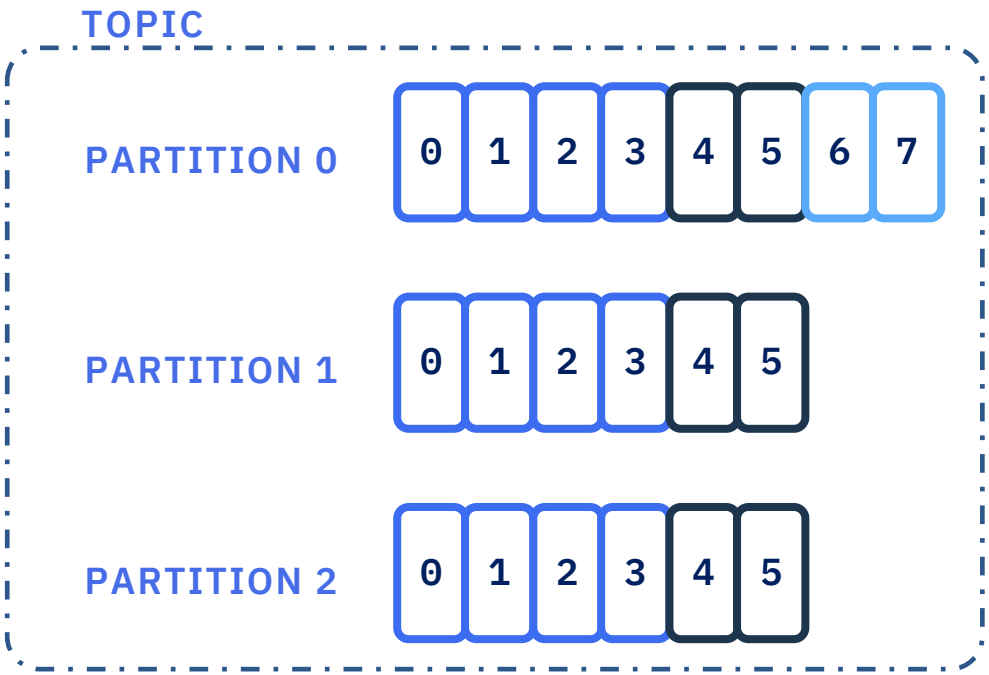
https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-overview/#topics

Producers and Keys



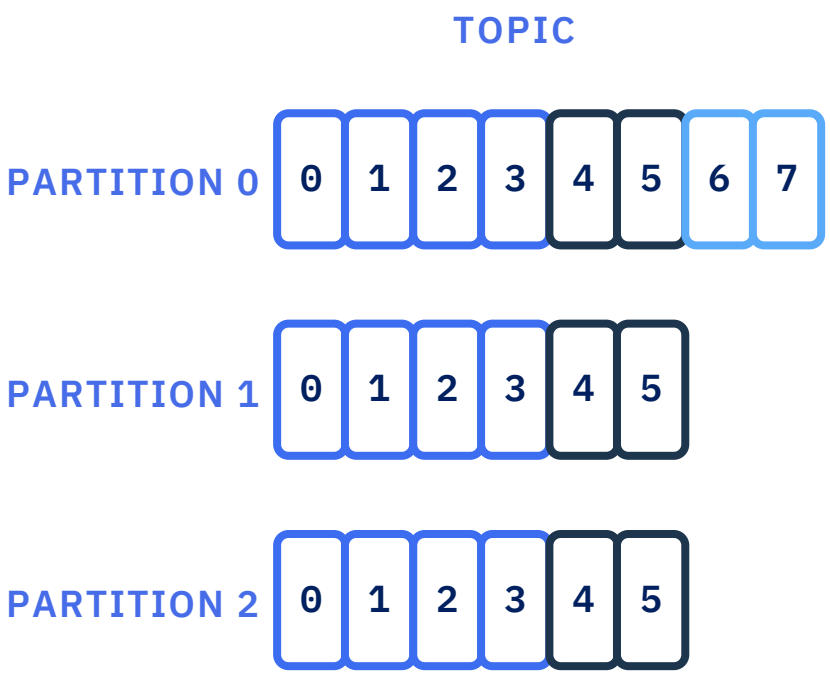
When no key is set, producers records will be appended in a round-robin fashion

Producers and Keys



If a key is set, Kafka will always append records with that same key to the same partition

Producers



Producer can choose acknowledgement level:

- `0` Fire-and-forget
  Fast, but risky
- `1` Waits for 1 broker to acknowledge
- `ALL` Waits for all replica brokers to acknowledge

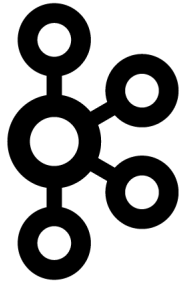
Producer can choose whether to retry:

- `0` Do not retry
  Loses messages on error
- `>0` Retry
  Retry, might result in duplicates on error

Producer can also choose idempotence

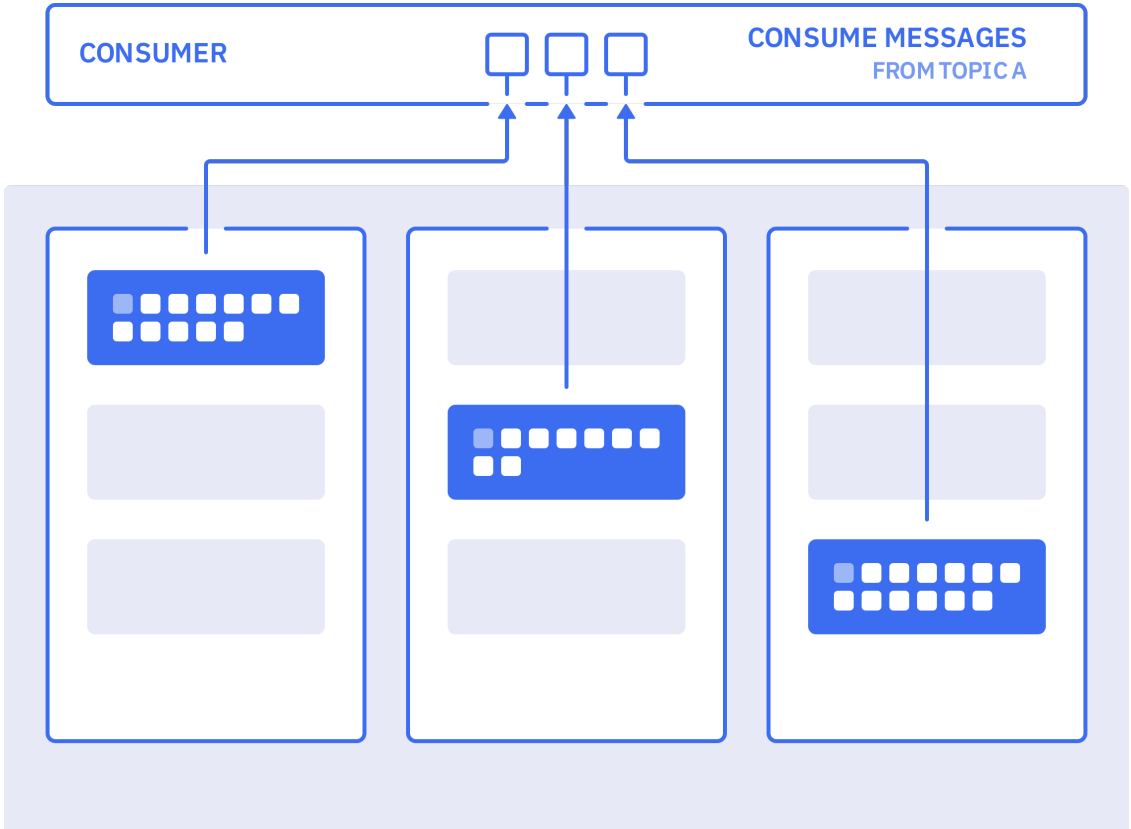
Can retry without risking duplicates

Kafka Producers

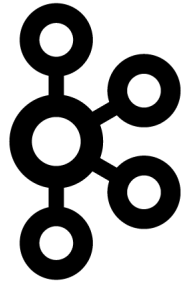


- Producers can send messages in batches for efficiency
  - By default, 5 messages can be in flight at a time
  - More messages are placed in batch and sent all at once
  - Creating a small delay in processing can lead to better performance
  - Batches wait until the delay has expired or batch size is filled
  - Messages larger than the batch size will not be batched
- If Producers are sending faster than Brokers can handle, then Producers can be throttled
  - Set the buffer memory for storage
  - Set the blocking time (milliseconds)
  - Throw an error message that the records cannot be sent
- Schema Registry is available to validate data using Schema Registry
  - Uses Apache Avro
  - Protects from bad data or mismatches
  - Self describing

Consumers

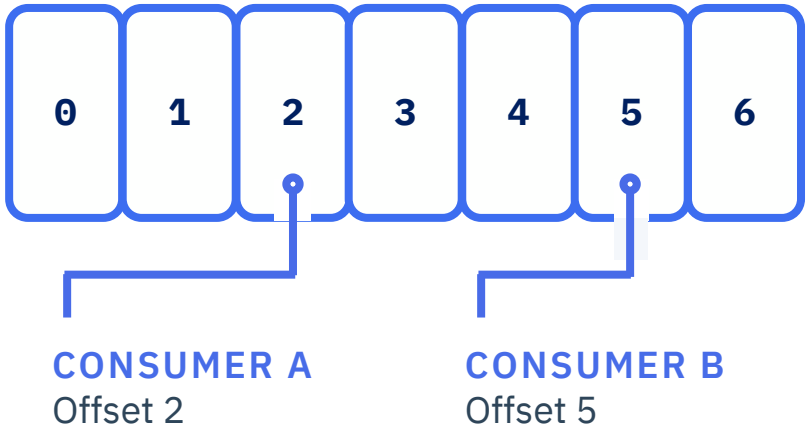


Kafka Consumers



- Consumers subscribe to 1 or more Topics
  - Read from all partitions from the last offset and consumes records in FIFO order
  - Can have multiple consumers subscribed to a topic
  - Consumers can set the offset if records need to be re-processed
- Consumers can be idempotent by coding
- Schema Registry is available to validate message to schema using Apache Avro

Consumers



Consumer can choose how to commit offsets:

`Automatic`	Commits might go faster than processing
`Manual, asynchronous`	Fairly safe, but could re-process messages
`Manual, synchronous`	Safe, but slows down processing

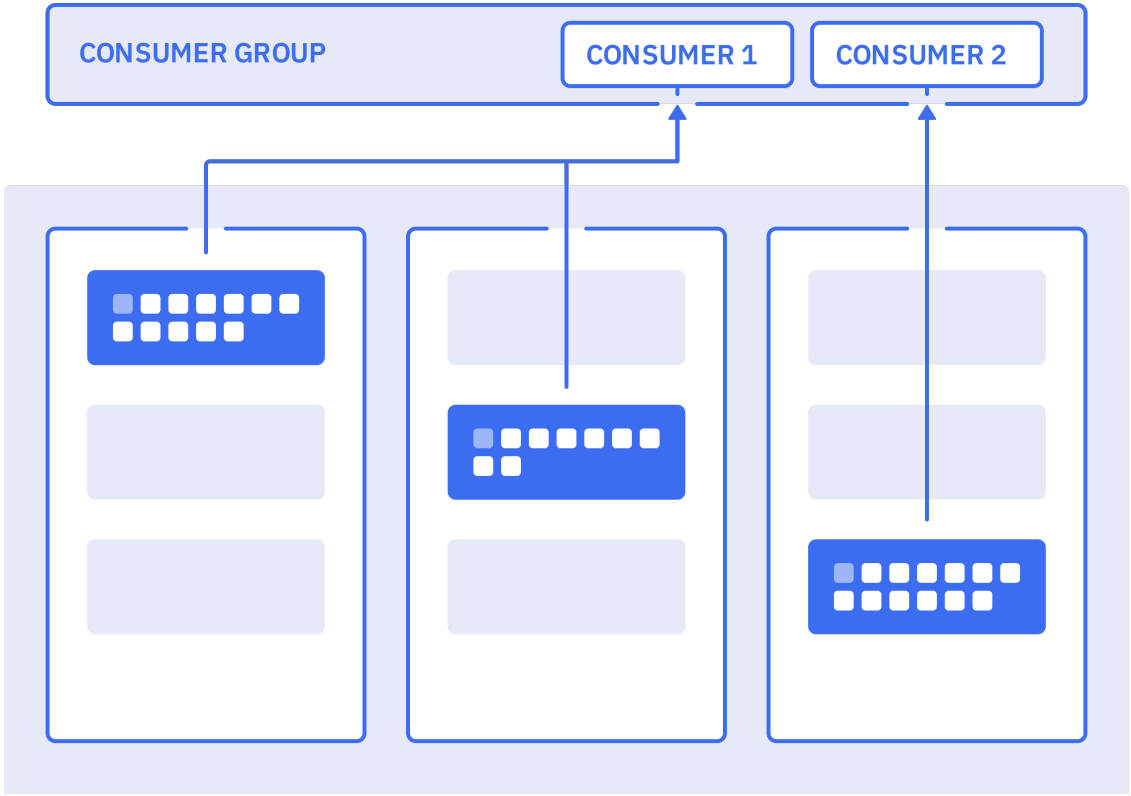
A common pattern is to commit offsets on a timer

Exactly once semantics

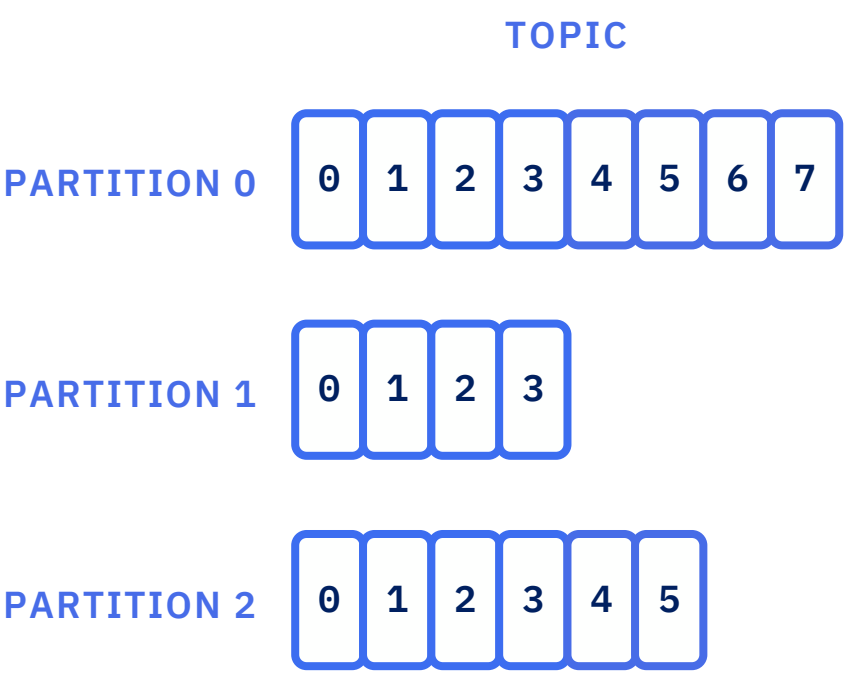
Can group sending messages and committing offsets into transactions

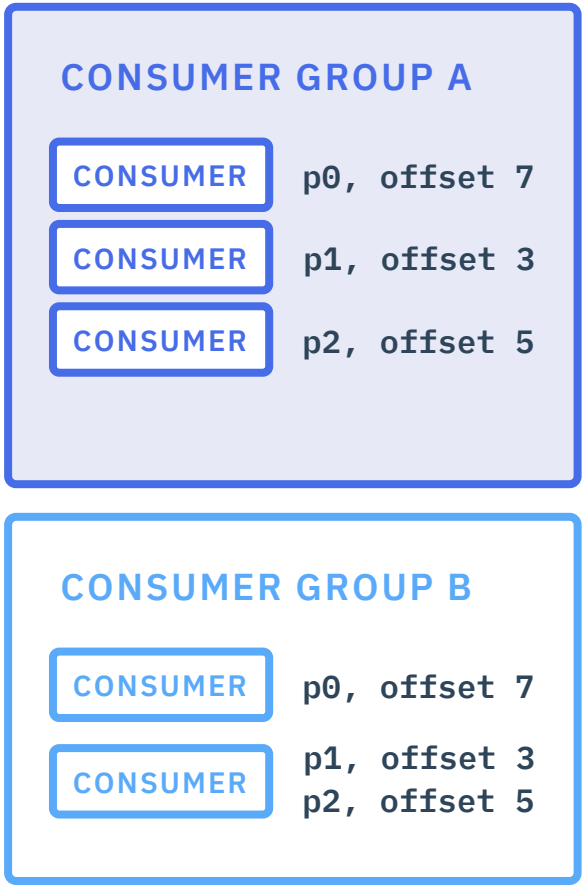
Primarily aimed at stream processing applications

Consumer Groups

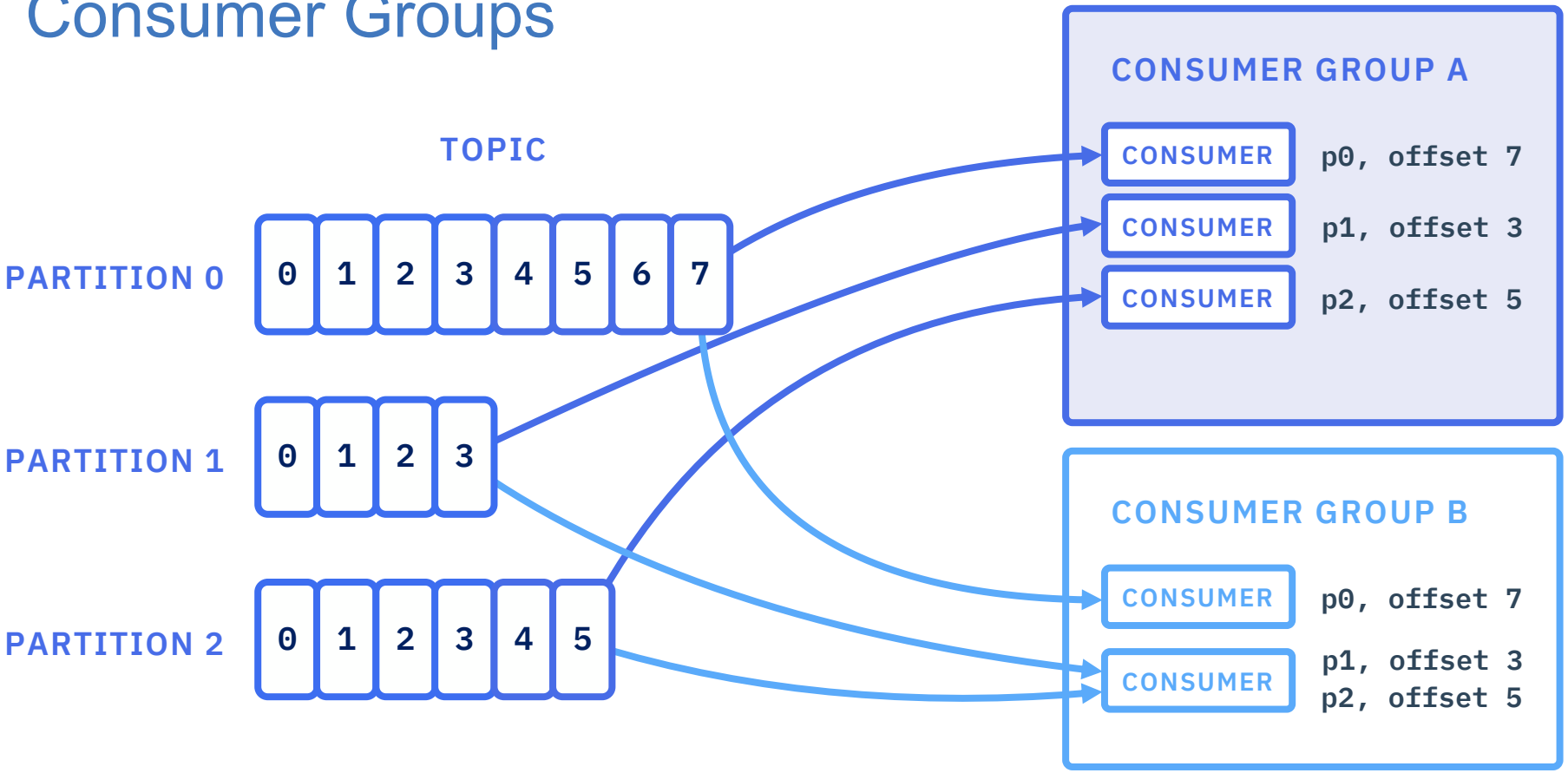


Consumer Groups





Consumer Groups



Kafka Consumer – Things to consider

- Parallel processing or not to keep complete event order
  - Number of consumers in a group
- Number of topics to read from
- Offset commitment logic
  - Auto commit
  - Apply business logic before committing by API
- Polling timeout
- Idempotence support or not
  - Accept to process events twice

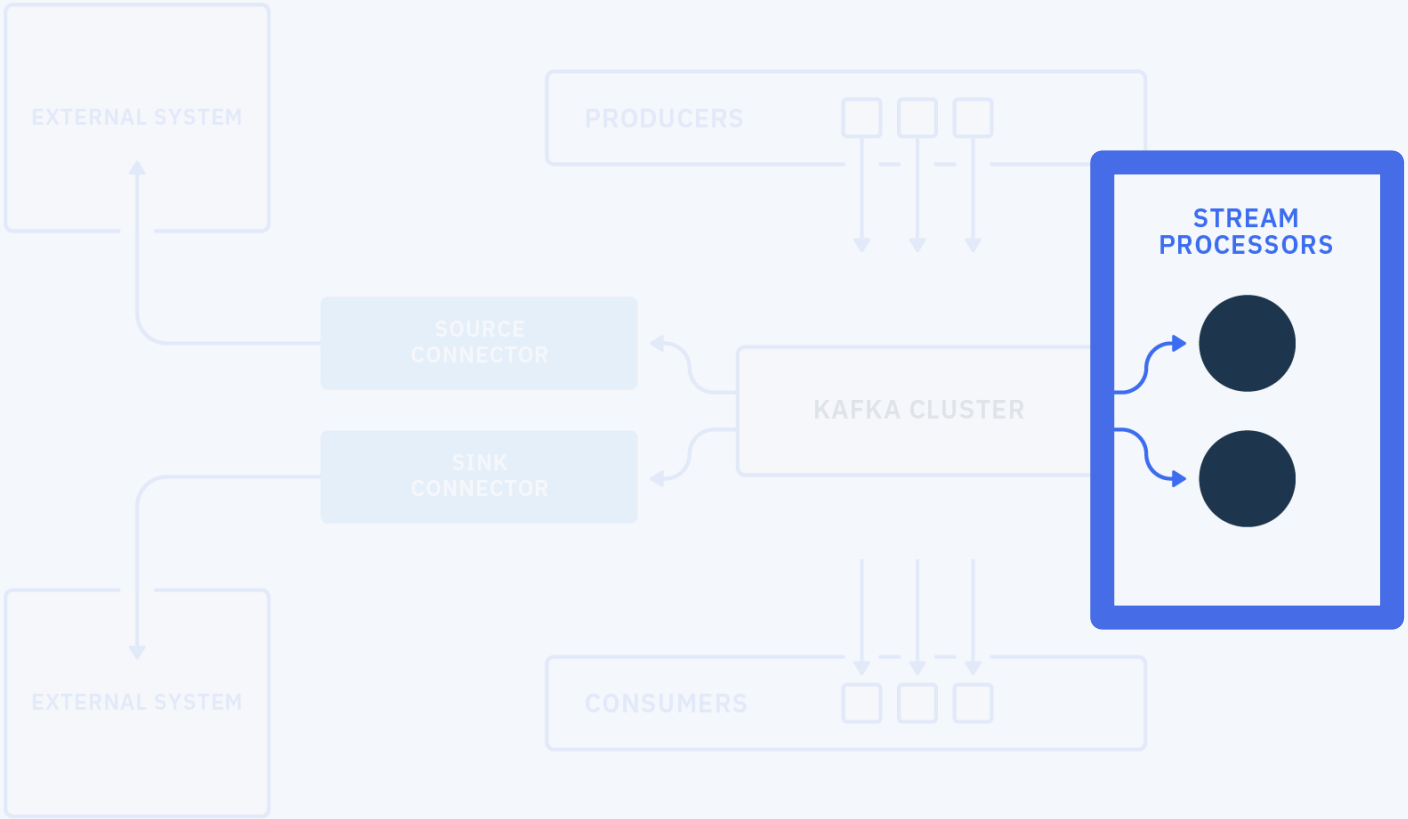
Transmission mode

- Producer
  - Async: no guarantee
  - Committed to leader, one acknowledge
  - Committed to quorum, leader waits for all acknowledges before responding
- Consumer
  - At most once (default): auto commit, with small commit interval
  - At least once: manual commit, **consumer.commitSync()** implement **'idempotent'** behavior within consumer to avoid reprocessing of the duplicate messages
  - Effectively once
  - Exactly once

Exactly once

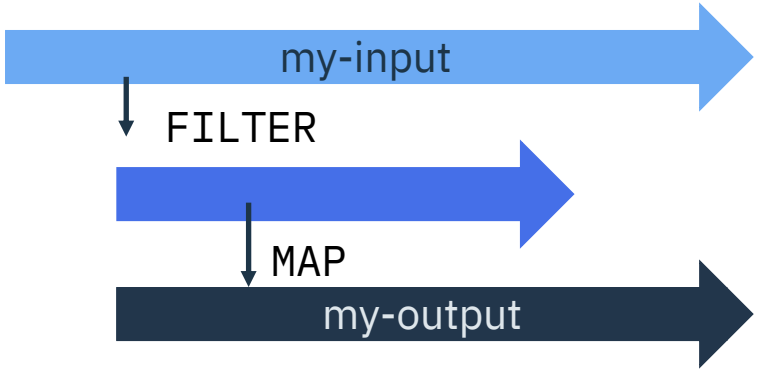
- Cooperation between producer, brokers, and consumers
- Producer can send duplicate records; the broker will send only one to consumer
  - Idempotent producer
- Write to multiple partitions is atomic
- Producer uses begin transaction and commit transaction
- Consumer is accepting only messages that are committed

https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-producers-consumers/#how-to-support-exactly-once-delivery



Kafka Streams

- Client library for processing and analysing data stored in Kafka
- Processing happens in the app
- Supports per-record processing – no batching
- Functional programming approach



Streams & Tables

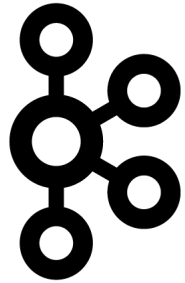
KStreams

- Abstraction of an unending record stream
- Immutable individual entries
- Each data record is considered an isolated datum.

KTables

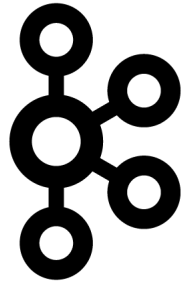
- Abstraction of a changelog stream
- Mutable individual entries
- Each data record is considered a contextual update.
- ***GlobalKTables*** are available to aggregate across partitions.

Stateless Transforms



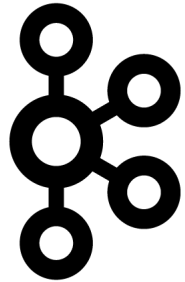
Transform	Supported on	Description
Branch	KStream -> KStream[]	Logically split a KStream & record to only one of N branches
Filter / Inverse Filter	KStream -> KStream KTable -> KTable	Evaluate a Boolean function to determine whether to retain record or not
FlatMap / FlatMapValues	KStream -> KStream	Process a single record to produce one or more records. Options include modifying keys, values, and types.
GroupByKey	KStream -> KGroupedStream	Groups records by the existing key
GroupBy	KStream -> KGStream KTable -> KGTable	Groups records by a new key, which can be a new key type. Can also support new value and value type.
Map / MapValues	KStream -> KStream	Process a single record and produces a single record. Options include modifying keys, values, and types.

Stateless Transforms



Transform	Supported on	Description
Foreach	KStream -> void KTable -> void	Perform a stateless action on each record and terminate stream processing. Think *side effects.*
Peek	KStream -> KStream	Perform a states action on each record and return an unchanged stream. Think side effects.
SelectKey	KStream -> KStream	Assigns a new key, including support for a new key type, to each record.
Table to Stream	KTable -> KStream	Get the changelog stream of a table.
Stream to Table	KStream -> KTable	Convert event stream into a table. Also known as a changelog stream.
To	KStream -> void	Write the records to a Kafka topic and end processing.
Through	KStream -> KStream KTable -> KTable	Write the records to a Kafka topic and create a new stream/table from that topic to continue processing.

Stateful Transforms



Transform	Supported on	Description
Aggregate*	KGStream → KTable KGTable → KTable	Aggregate values of records by the grouped key. Resultant value type can be changed. Generalization of *Reduce.*
Reduce*	KGStream → KTable KGTable → KTable	Combines values of records by the grouped key. Resultant value type cannot be changed.
Count*	KGStream → KTable KGTable → KTable	Counts the number of records by the grouped key.
Transform	KStream→ KStream	Apply a Transformer to each record, allowing low-level Processor API integration for updating both keys and values.
TransformValues	KStream → KStream KTable → KTable	Apply a ValueTransformer to each record, allowing low-level Processor API integration for updating only values.
Inner Join	See next chart	Performs an INNER JOIN of the current stream with another stream.
Left Join	See next chart	Performs a LEFT JOIN of the current stream with another stream.
Outer Join	See next chart	Performs an OUTER JOIN of the current stream with another stream.

*Supports both rolling and windowed aggregation.

Joins

Join operands	Type	(INNER) JOIN	LEFT JOIN	OUTER JOIN
KStream-to-KStream	Windowed	Supported	Supported	Supported
KTable-to-KTable	Non-windowed	Supported	Supported	Supported
KTable-to-KTable Foreign-Key Join	Non-windowed	Supported	Supported	Not Supported
KStream-to-KTable	Non-windowed	Supported	Supported	Not Supported
KStream-to-GlobalKTable	Non-windowed	Supported	Supported	Not Supported
KTable-to-GlobalKTable	N/A	Not Supported	Not Supported	Not Supported

Windows

- *Hopping time window* – fixed-sized, overlapping windows
- *Tumbling time window* – fixed-sized, non-overlapping, gap-less windows
- *Sliding time windows* – fixed-sized, overlapping windows that focus on record timestamps
- *Session windows* – dynamically-sized, non-overlapping, data-based windows

Source: https://kafka.apache.org/25/documentation/streams/developer-guide/dsl-api.html#joining

Pros & Cons

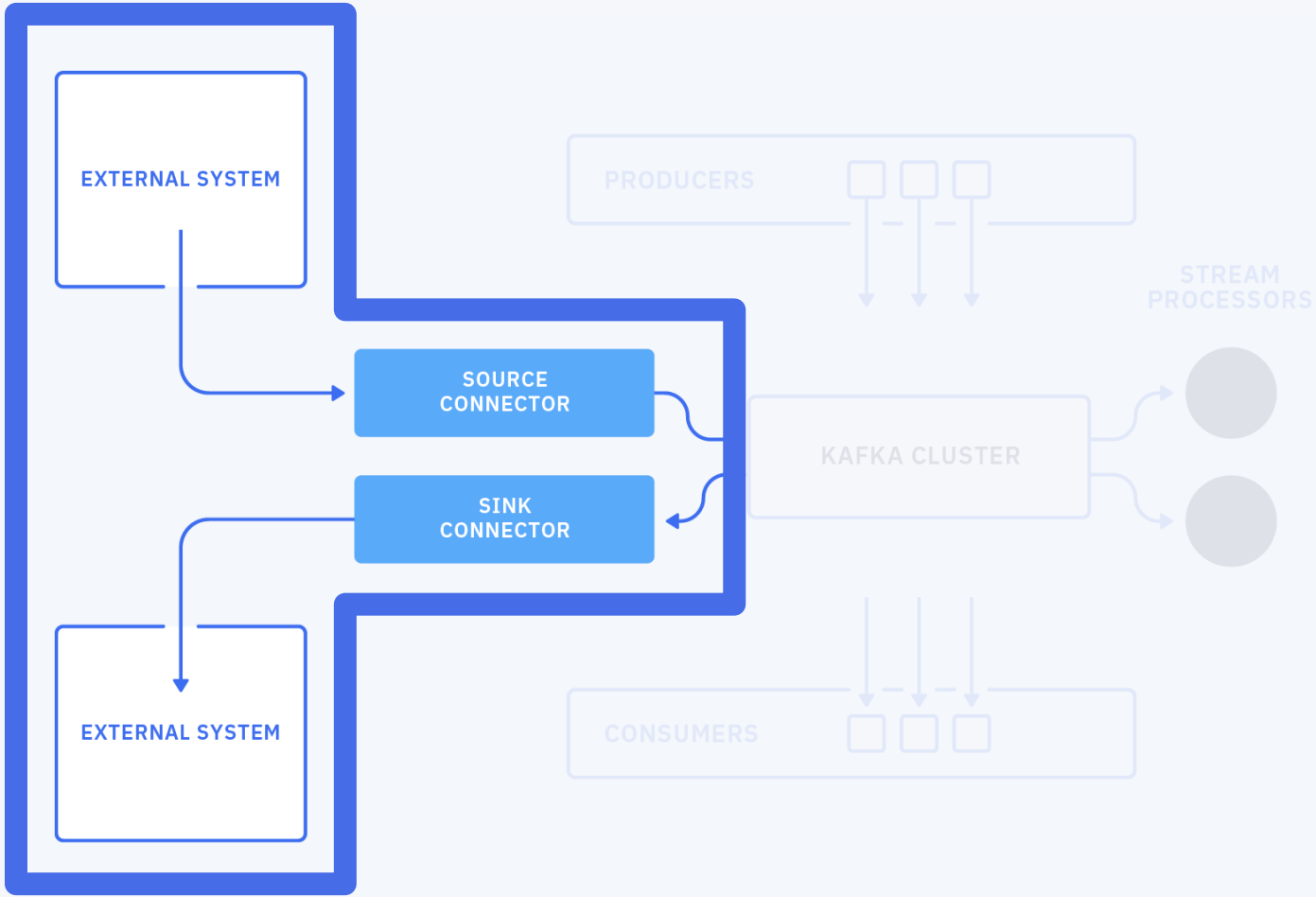
Considerations to what does this all mean…

Pros

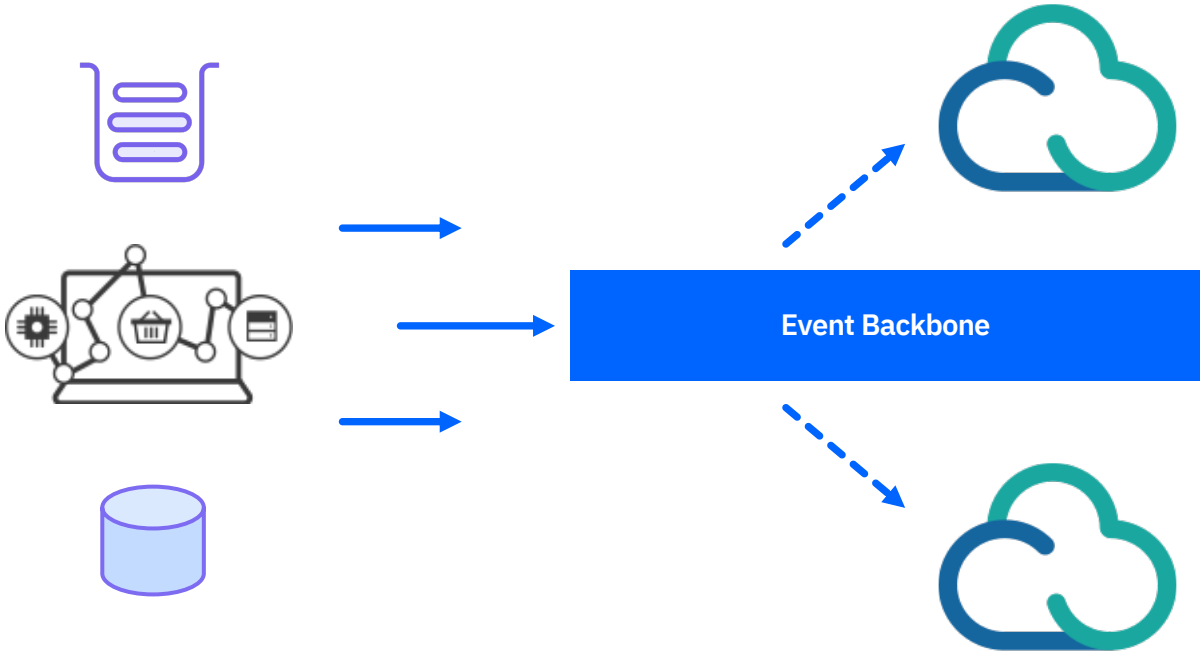
- A lightweight client library provides a lot of processing power
- Manual creation of streaming transforms helps to keep use cases small in scope
- No additional middleware is required besides your app and Kafka
- Simple connectivity to existing Kafka clusters.
- Freely licensed OSS component of Apache Kafka.

Cons

- Java (and Scala) only
- Deeper Kafka experience is required
- Ability to manually create abstracted transforms can quickly create performance hogs
- Maintain integrations & environments via Dev teams.

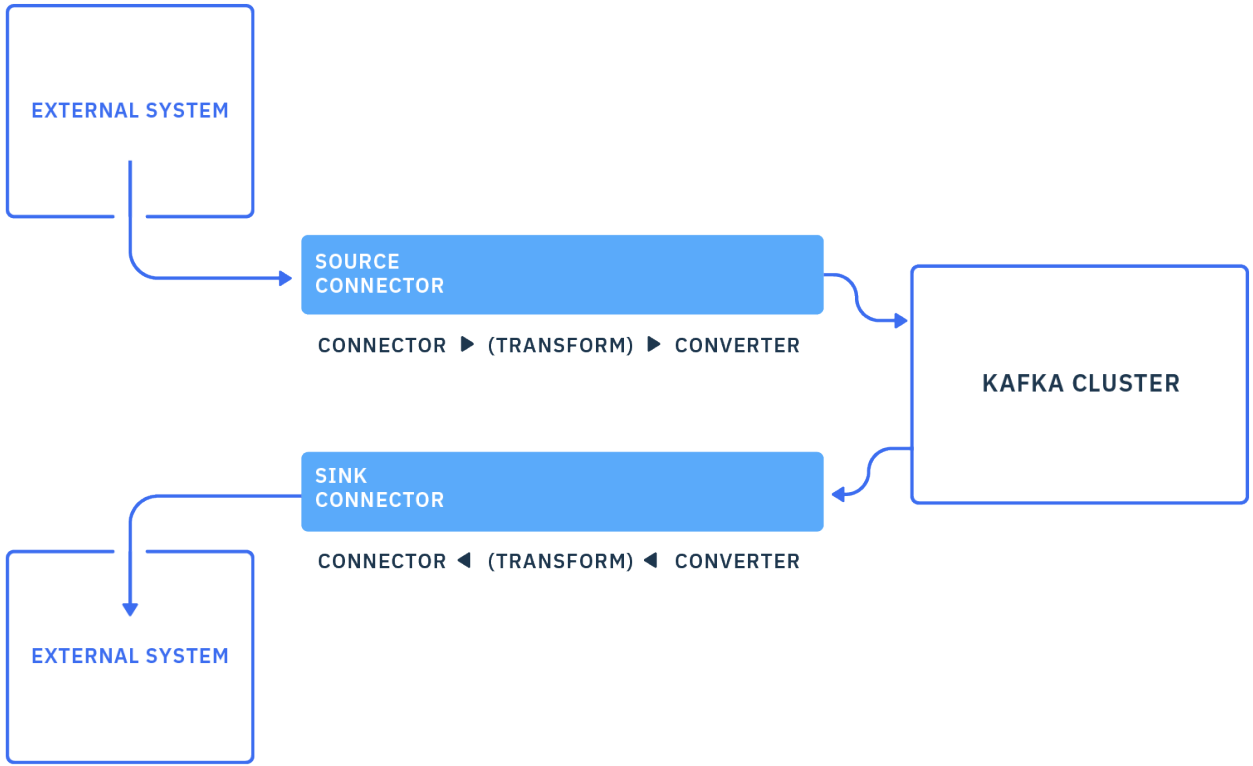


Kafka Connect – bridge to cloud-native apps



Define *connectors* that move large collections of data into and out of Kafka

Kafka Connect



Over 80 connectors

HDFS

Elasticsearch

MySQL

JDBC

IBM MQ

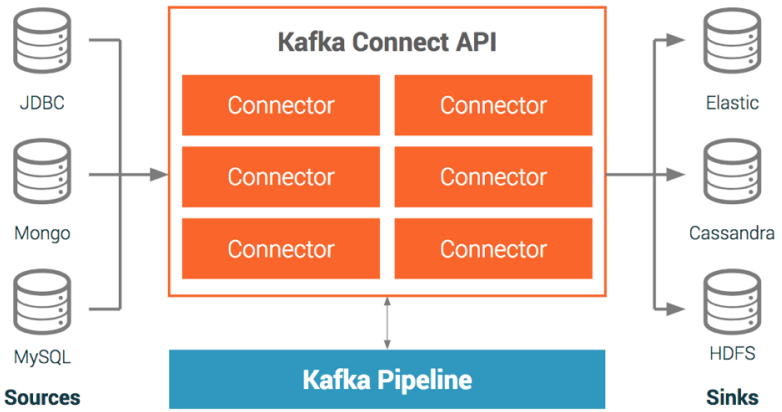
MQTT

CoAP

+ many others

Kafka Connectors

- Connectors allow for integration from sources to sinks and vice versa
  - Import from sources like DBs, JDBC, Blockchain, Salesforce, Twitter, etc
  - Export to AWS S3, Elastic Search, JDBC, DB, Twitter, Splunk, etc
  - Run a connect cluster to pull from source and publish it to Kafka
  - Can be used with Streams
  - Confluent Hub has many connectors already available
- Connectors can be managed with REST calls

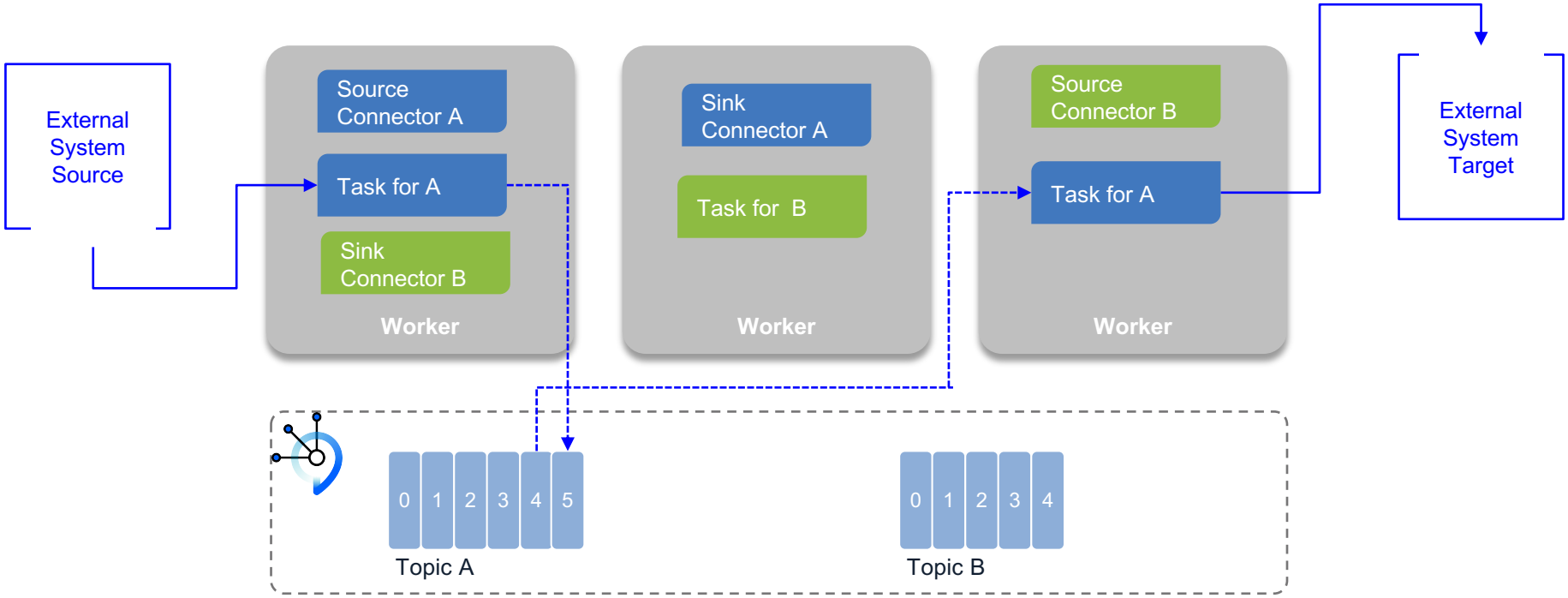


Kafka Connect

- A common framework for connecting external system to Kafka brokers
- Can run standalone or distributed *(distributed preferred for k8s-based systems)*
- Define connector cluster with properties file and then connectors
- Source and sink connectors to integrate with different external systems
- REST Interface to add connector dynamically
- Automatic offset management
- Streaming/batch integration

Kafka Connect

- Distributed deployment



IBM Event Streams

IBM Event Streams Delivers Differentiated Value

IBM offers **Event Streams** in several form factors:

Private Cloud

Container–native Software

IBM Cloud Pak for Integration

Red Hat OpenShift

X86_64 & **zLinux**

IBM Public Cloud

Isolated aaS

Multi-tenant aaS

- **IBM has years of operational expertise** running Apache Kafka for Enterprises
  - This experience has been embedded in the DNA of Event Streams
  - As a service since 2015 with 99.999% availability
- Event Streams makes Kafka easy to run, manage & consume, **reducing skill requirements** and increasing speed of deployment for **faster time to value**
- IBM Cloud Pak for Integration security integration **simplifies Kafka access control** using roles and policies
- **IBM's experience in enterprise-critical software** has shaped features like geo-replication for Disaster Recovery & integration with IBM MQ, to give confidence deploying **mission-critical workloads**
- **Support you can trust** – IBM has decades of experience supporting the World's toughest environments

https://www.ibm.com/cloud/event-streams

Security – Authentication and Access Control

- User and group information controlled centrally
  - Integrate with your corporate LDAP through IBM Cloud Private
- Control access to Event Streams resources using role-based access control policies
  - Assign roles to users: Viewer, Editor, Operator, Administrator
  - Optionally, specify access for individual resources, such as topic T
- Application access control using service IDs
  - Same concepts as controlling user access
  - Can restrict application access to exactly the resources they need
    - Cluster, topic, consumer group, transaction
  - Prevent accidental or intentional interference between applications

Example policy

Permit Bob to write to topic T

User	bob
Role	Editor
Service	Event Streams instance R
Resource	T
Type	topic

Service action	Roles	Permissions
topic.read	Viewer and above	Read messages or config
topic.write	Editor and above	Write messages
topic.manage	Operator and Administrator	Delete or change config

Security – Authentication and Access Control

- SASL with Bearer token using API key for broker connection
- Separate network for zookeepers and brokers communication
- TLS key and certificates
- Use storage encryption and network encryption for pod to pod communications
- Application access restricted with Service ID

Publish Events from Anywhere with the REST Producer API

IBM has created a new easy-to-use REST Producer API

POST /topics/{topic_name}/records
Content-Type: text/plain
Authorization: Bearer {bearer_token}
Hello Event Streams

- Use it wherever it's difficult to use a real Kafka client e.g. DataPower, z/OS
- Straightforward design makes it easy to use from the command line and developer tools
- Supports partitioning keys and headers
- So easy you can use it from the command line with cURL

It’s easy to connect IBM MQ to Apache Kafka

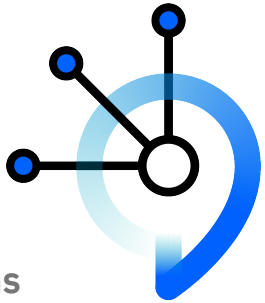
- IBM has created a pair of connectors, available as source code or as part of IBM Event Streams
- **Source connector**

  From MQ queue to Kafka topic

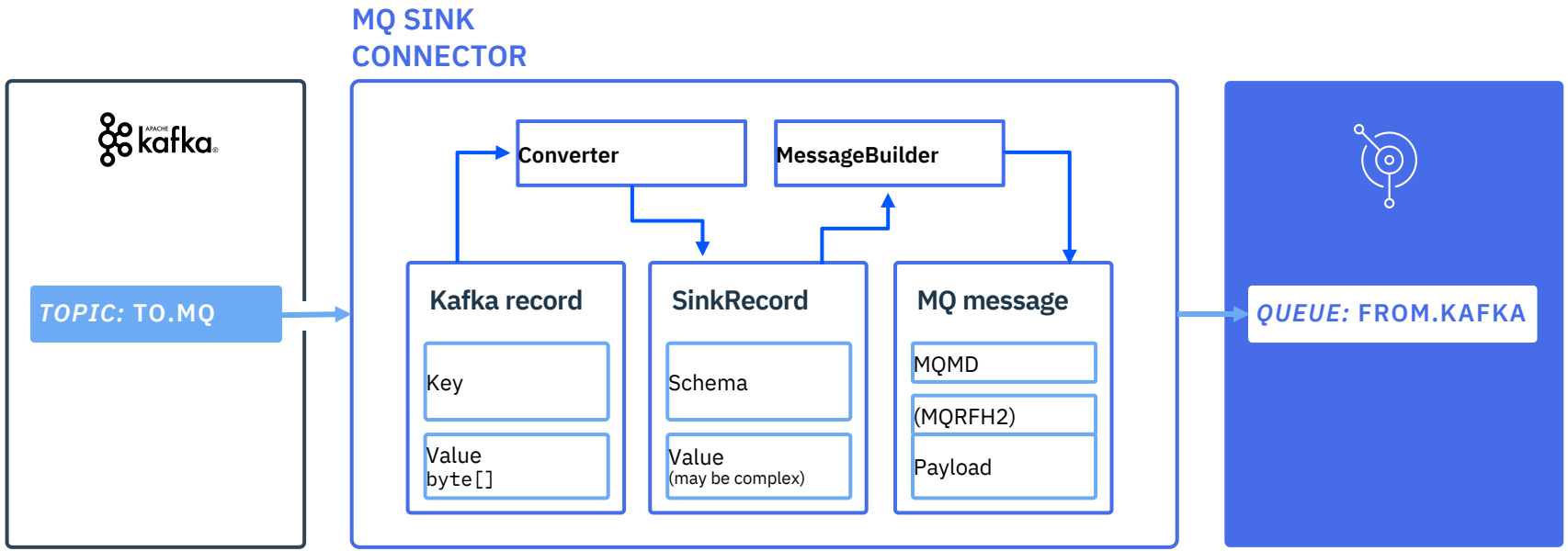
  https://github.com/ibm-messaging/kafka-connect-mq-source
- **Sink connector**

  From Kafka topic to MQ queue

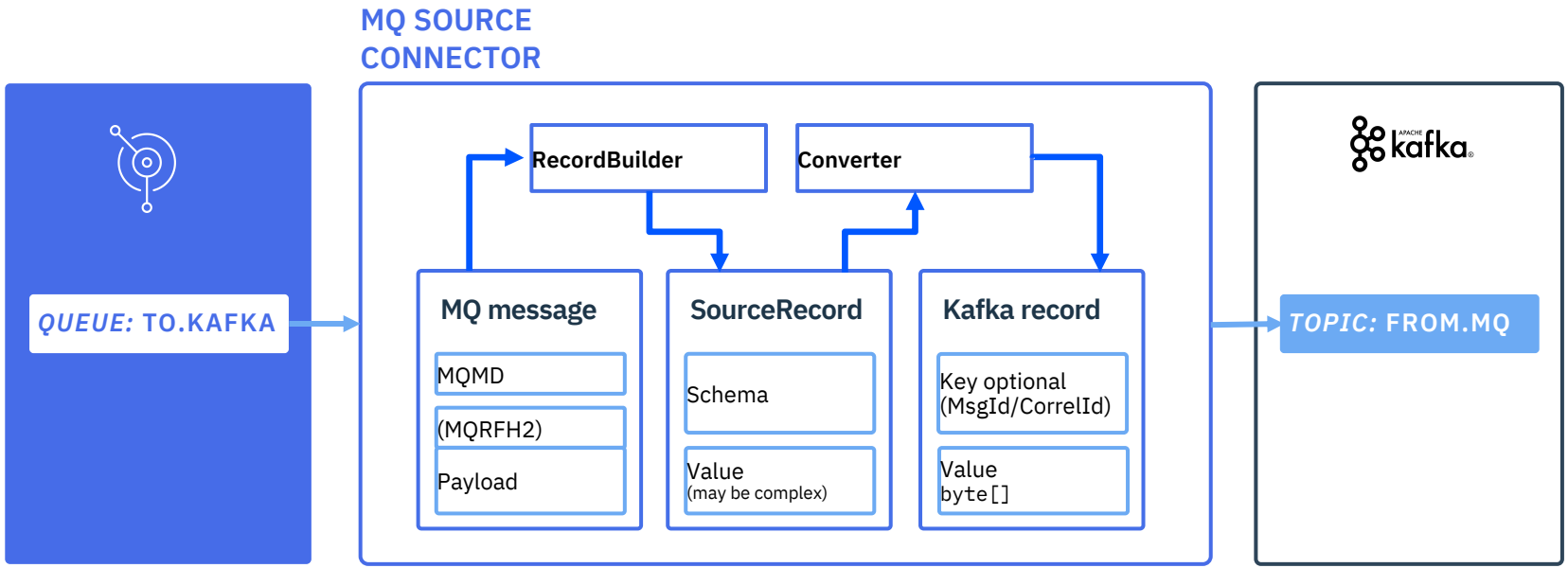
  https://github.com/ibm-messaging/kafka-connect-mq-sink
- **Fully supported by IBM for customers with support entitlement for IBM Event Streams**



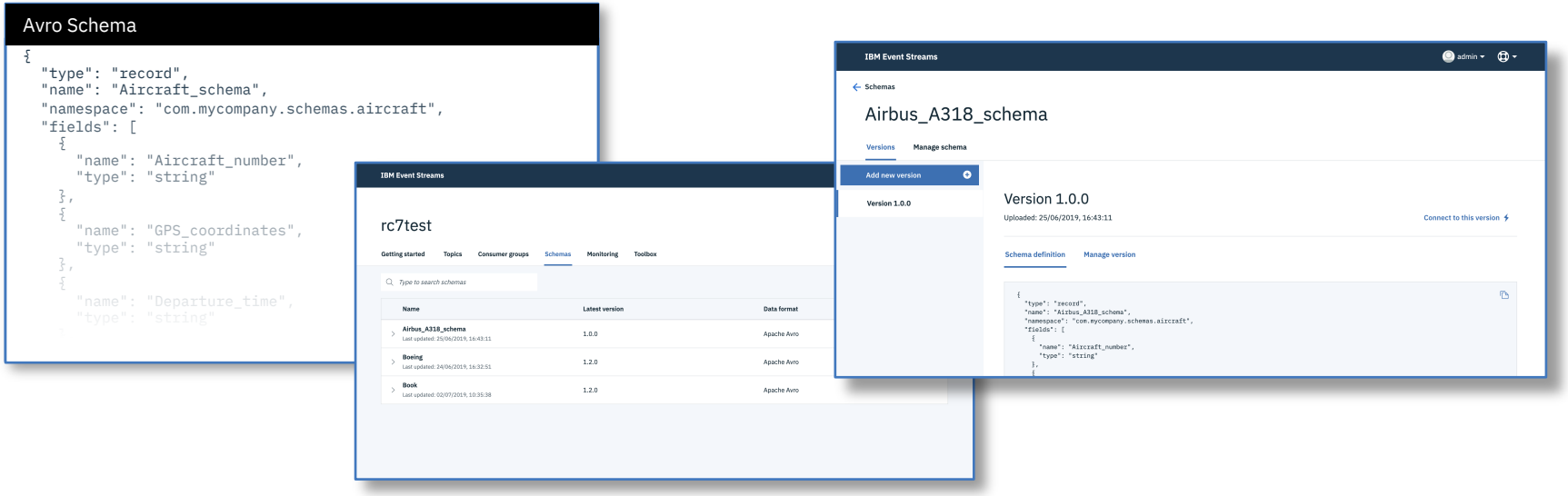
MQ sink connector

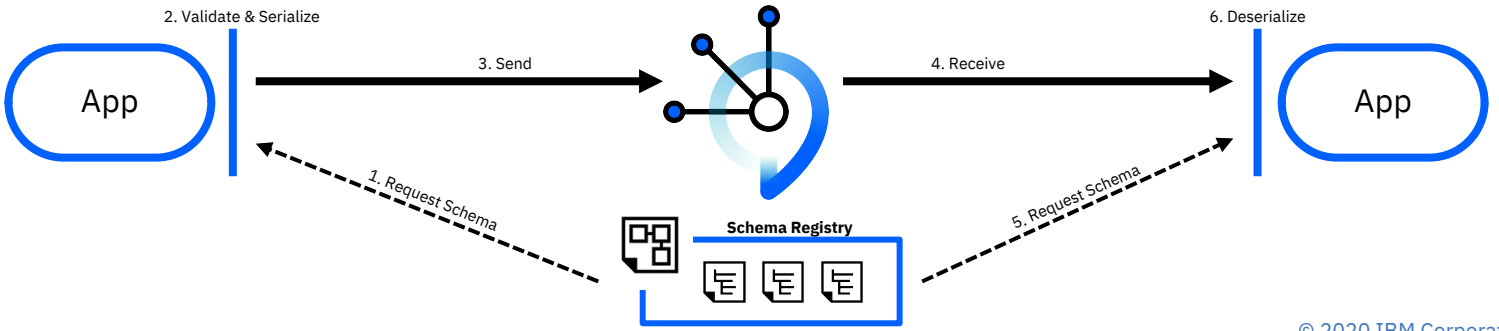


MQ source connector



IBM Event Streams | **Schema Registry**





Schema Registry

IBM Event Streams Schema Registry has been built for **flexibility and extensibility**. The Apache Avro supported format is only the first implementation, enabling our customers to standardize across their enterprise on their selection from a range of schema formats, and rely on IBM Event Streams to transmit, validate and interact with event records maintaining that format

IBM Event Streams schemas are defined as first-class objects with independent security controls, enabling **fully secure** isolation across application domains

IBM Event Streams message browsing is fully schema-aware, such that **debugging application interactions is simplified**, regardless of the choice of event record formatting

IBM Event Streams Schema Registry provides management of a full lifecycle for schemas and versions to support **effective application lifecycle management**

IBM

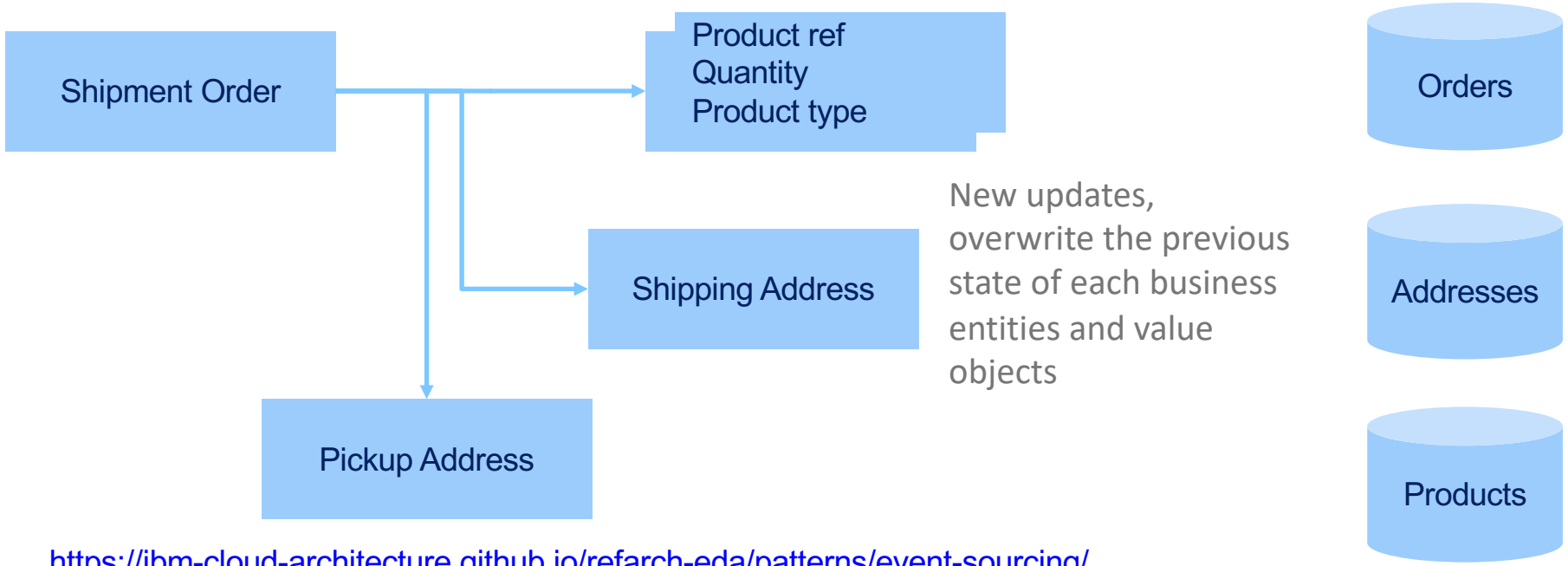
Event Driven Architecture

Data Patterns

Cloud Garage Solution Engineering

1. Event Sourcing

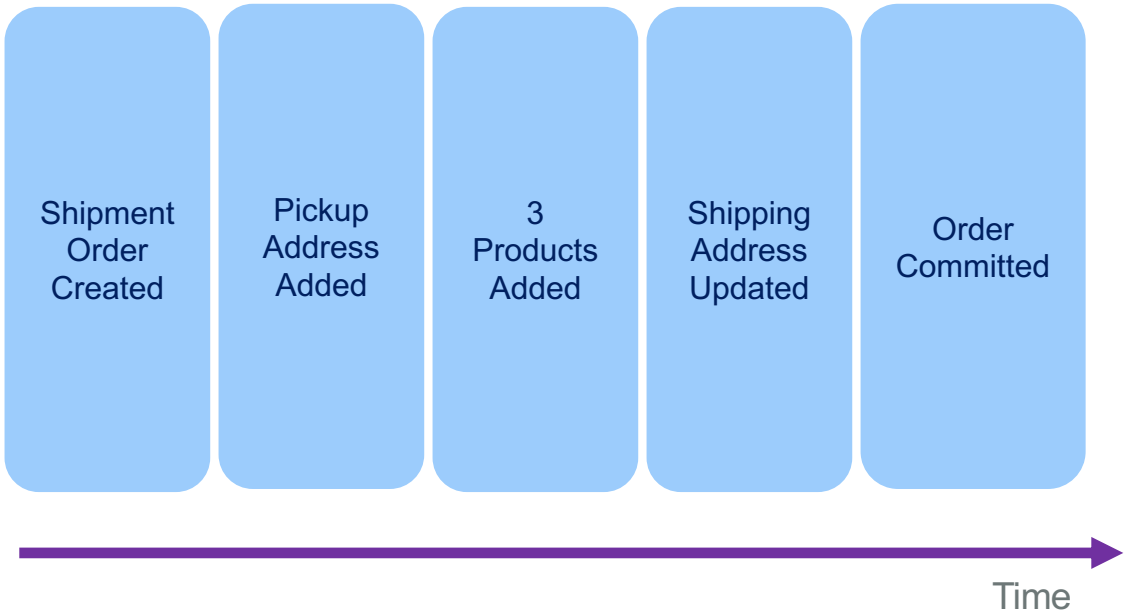
Event Sourcing: From a relational model…



https://ibm-cloud-architecture.github.io/refarch-eda/patterns/event-sourcing/

Greg Young Event Sourcing Video

Event Sourcing



Approach

- Immutable
- Ordered
- Only append
- Keep a ledger of all the changes
- Replay from past event

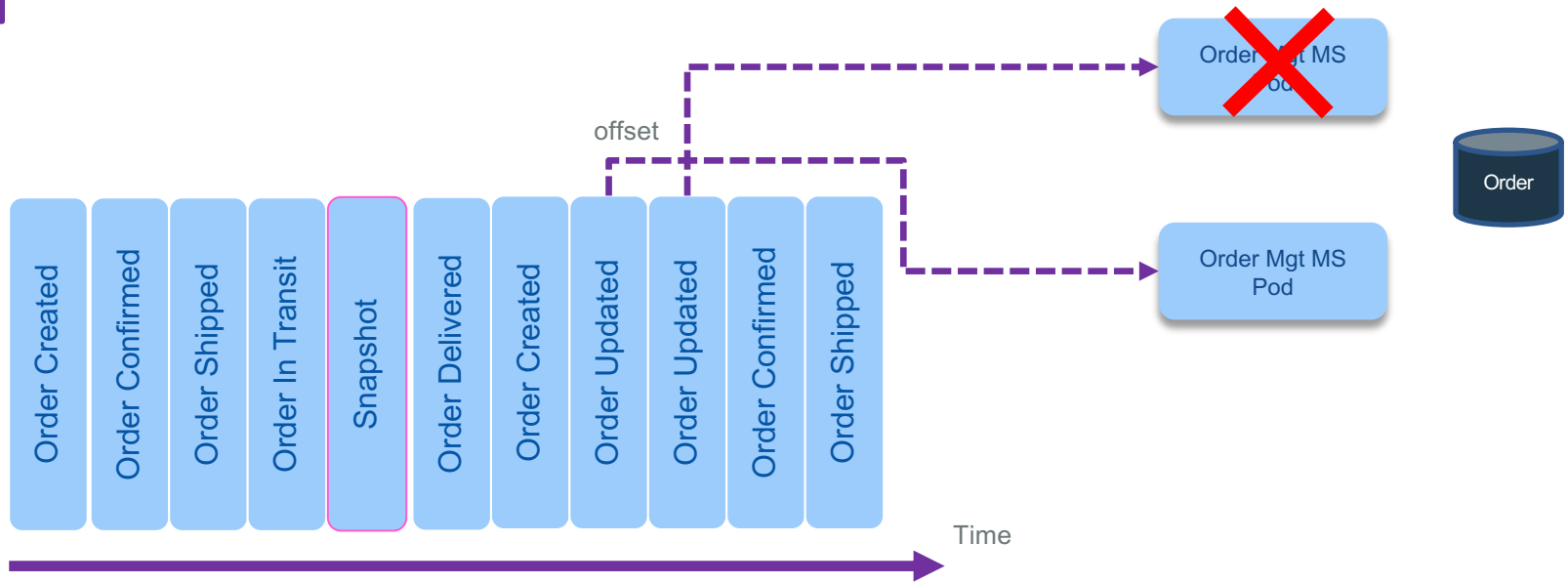
Benefits

- Does not lose information
- Keep facts about data
- Derive state from facts
- Derive projections / views from records
- Entities and value-objects in records
- Auditing
- Time-based queries
- Identify behavior pattern over time

Started in 2006 on transactional domain by Greg Young

https://ibm-cloud-architecture.github.io/refarch-eda/patterns/event-sourcing/

Event Sourcing

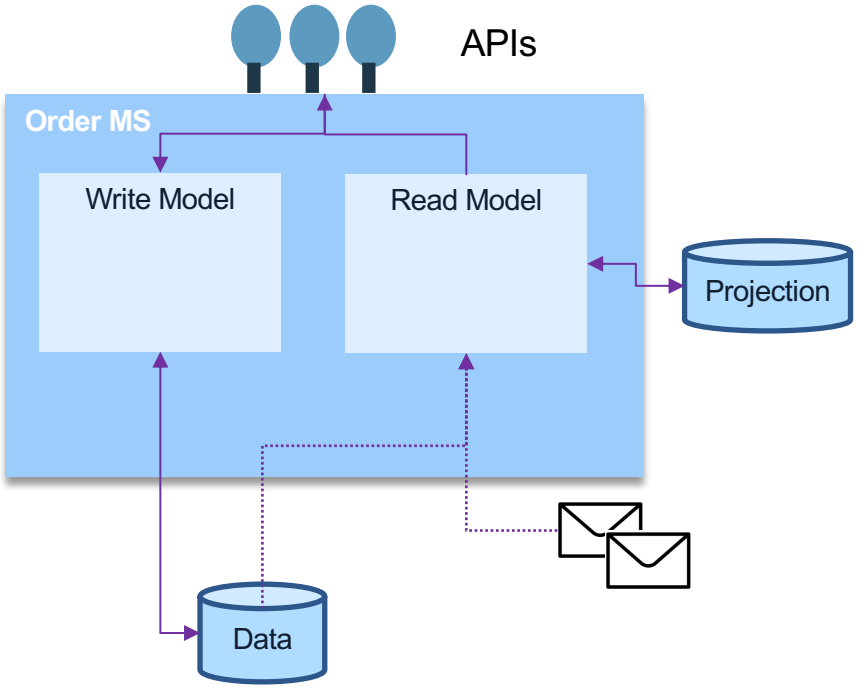


When a pod crashes, the new instance can reload from a previously committed offset, or from the last snapshot

https://chriskiehl.com/article/event-sourcing-is-hard

2. Command-Query Responsibility Segregation

Command Query Responsibility Segregation Pattern



Separate the read model from the write model

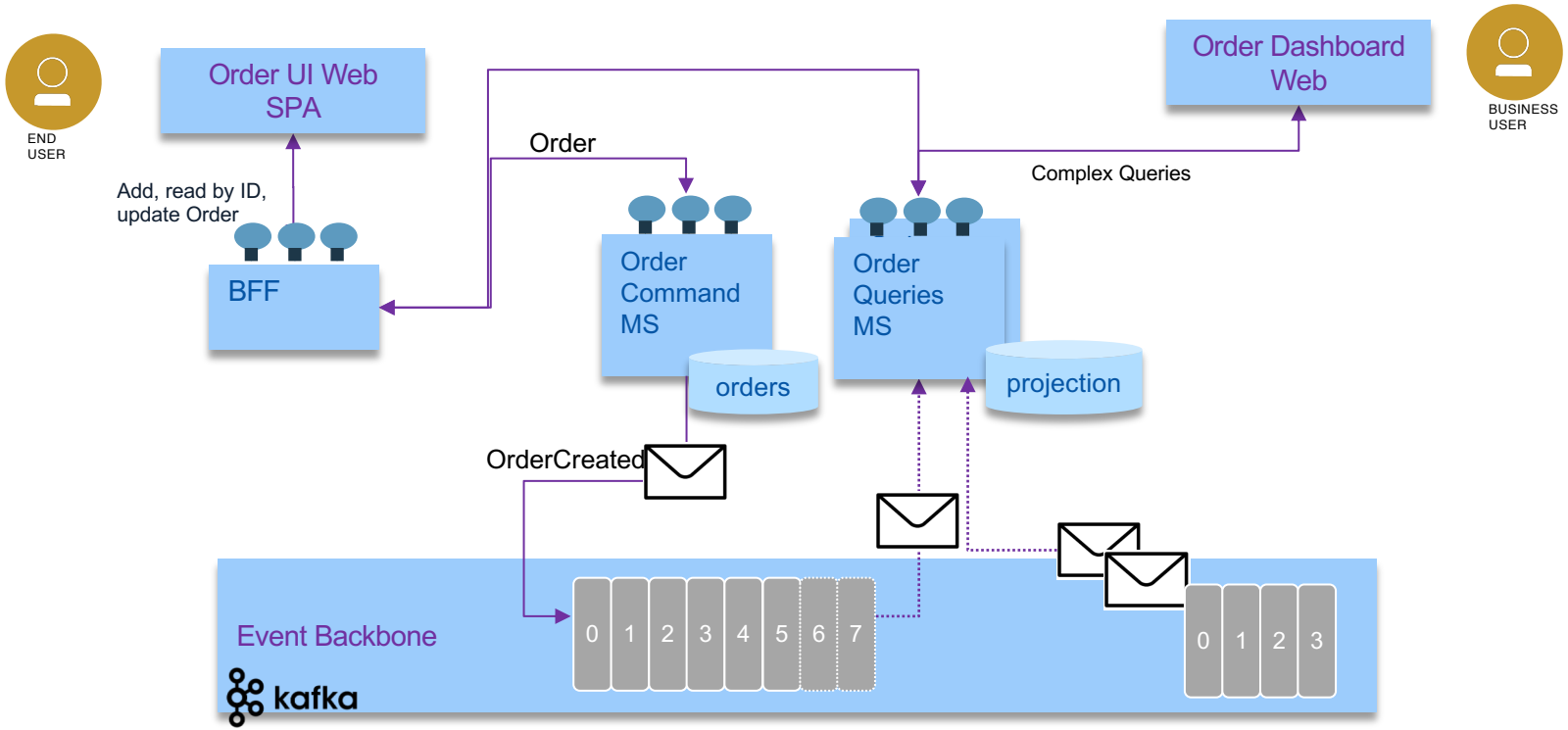
CUD on write model
… with basic read by ID, name….

Complex query done on Read model

How to **join** data / entities ?

https://ibm-cloud-architecture.github.io/refarch-eda/patterns/cqrs/

Event Sourcing & CQRS



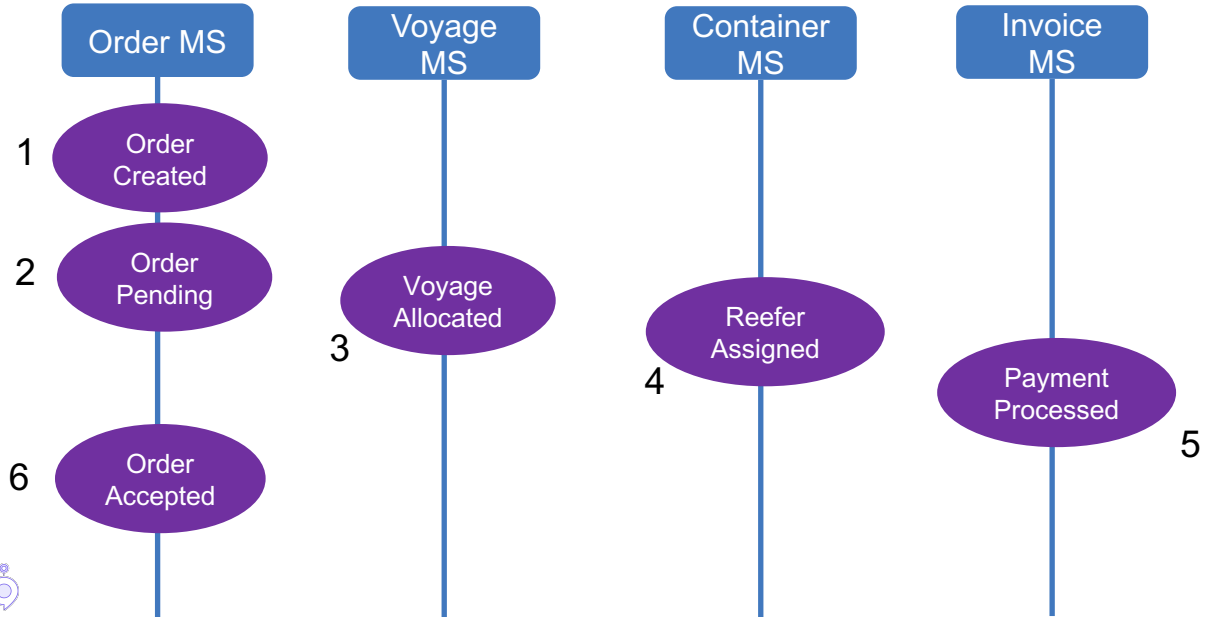
CQRS challenges

- Adapt to business problem and not imagine a generic CQRS framework
- Avoid naïve approach
- Start small
- Think about ad-hoc business process and not a linear set of steps or events

3. SAGA

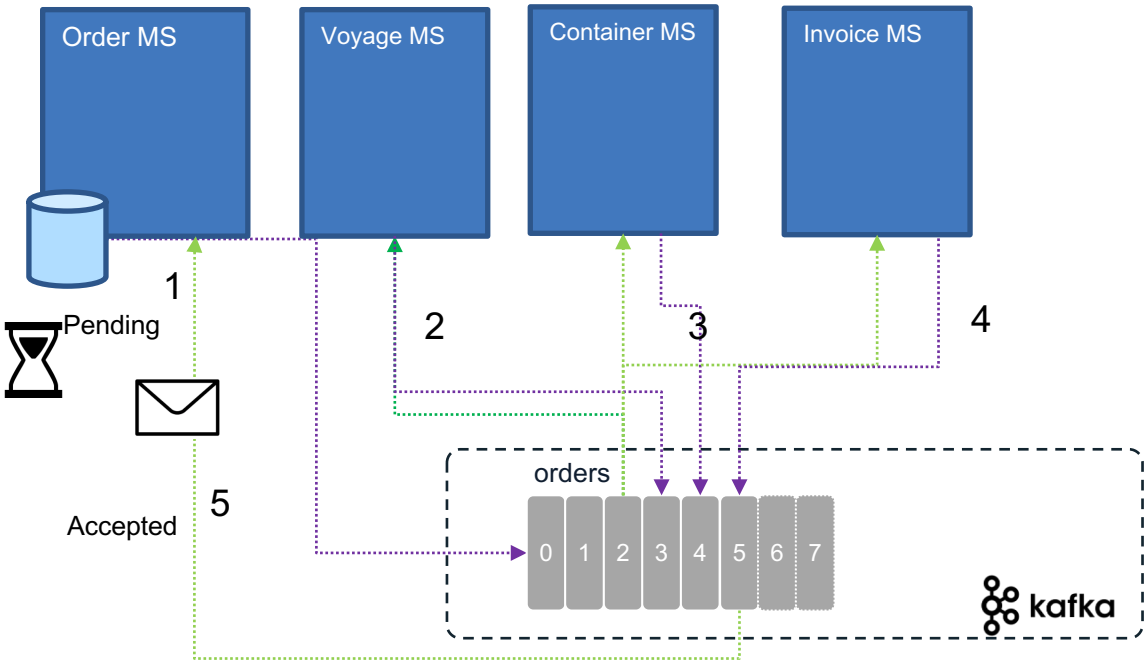
SAGA Pattern

- A long running transaction that can be broken up to a collection of sub-transactions that can be interleaved any way with other transactions. Src Garcia-Molina, Salem 1987
- With microservice each transaction updates data within a single service, each subsequent step may be triggered by previous completion.



Saga pattern - Choreography

Each service produces and listens to other service's events and decides if an action should be taken or not



The distributed transaction ends when the last service executes.

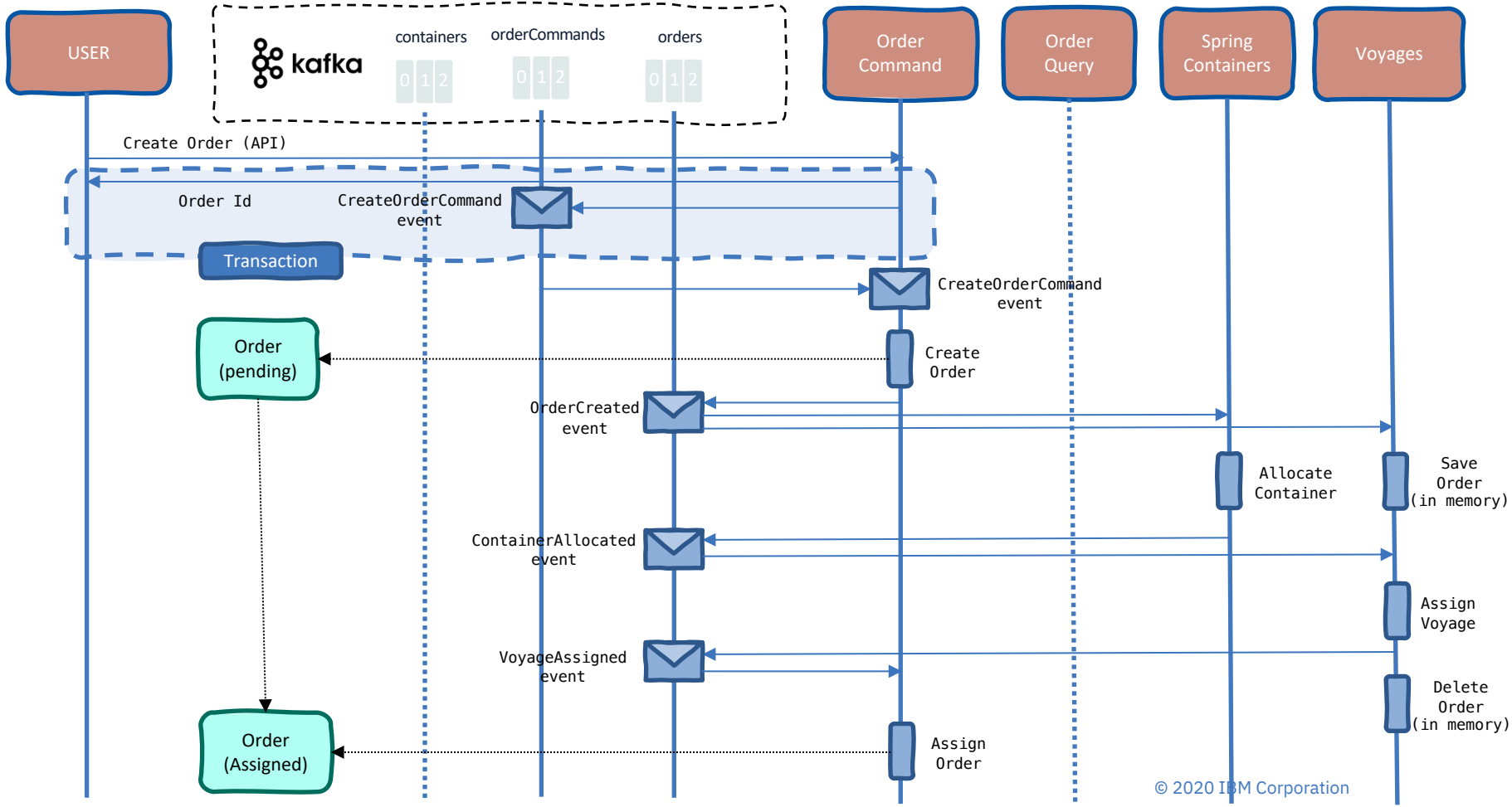
2- <orderID, voyageID>

3- <orderID, reeferID>

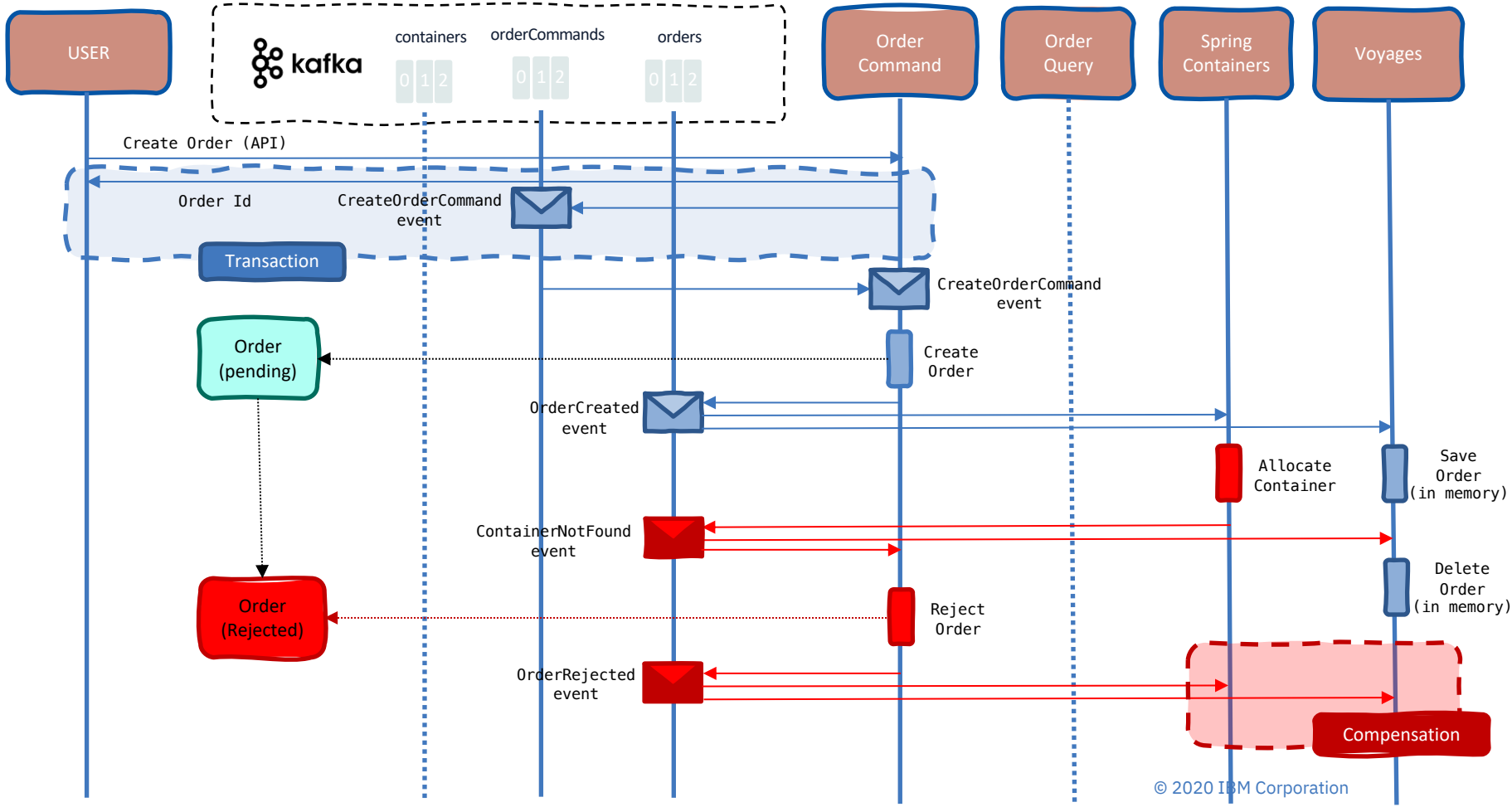
4- <orderID, transactionID>

5- Order has accepted

SAGA Pattern (Choreography) - Sequence Diagram – Create Order

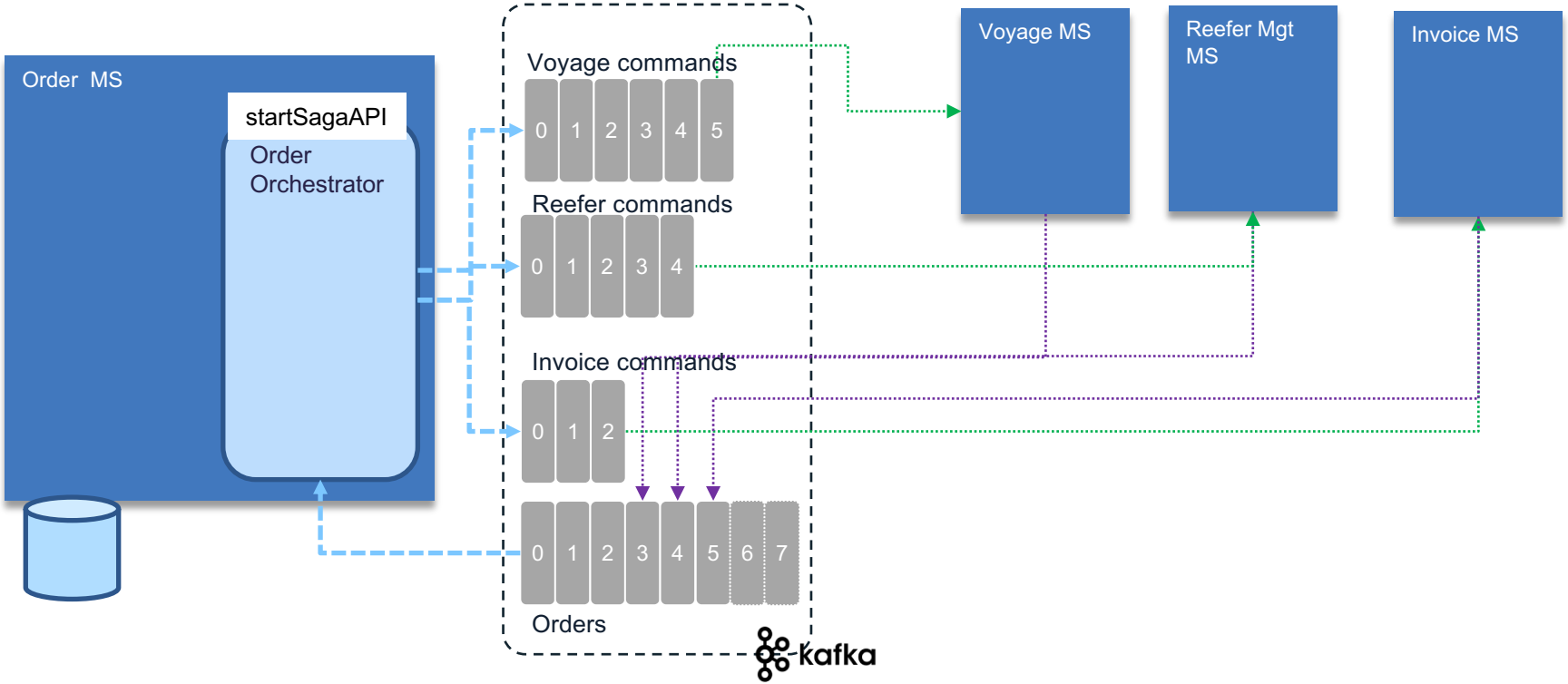


SAGA Pattern (Choreography) - Sequence Diagram – No Container



Saga pattern - Orchestration

One service is responsible to drive each participant on what to do and when.

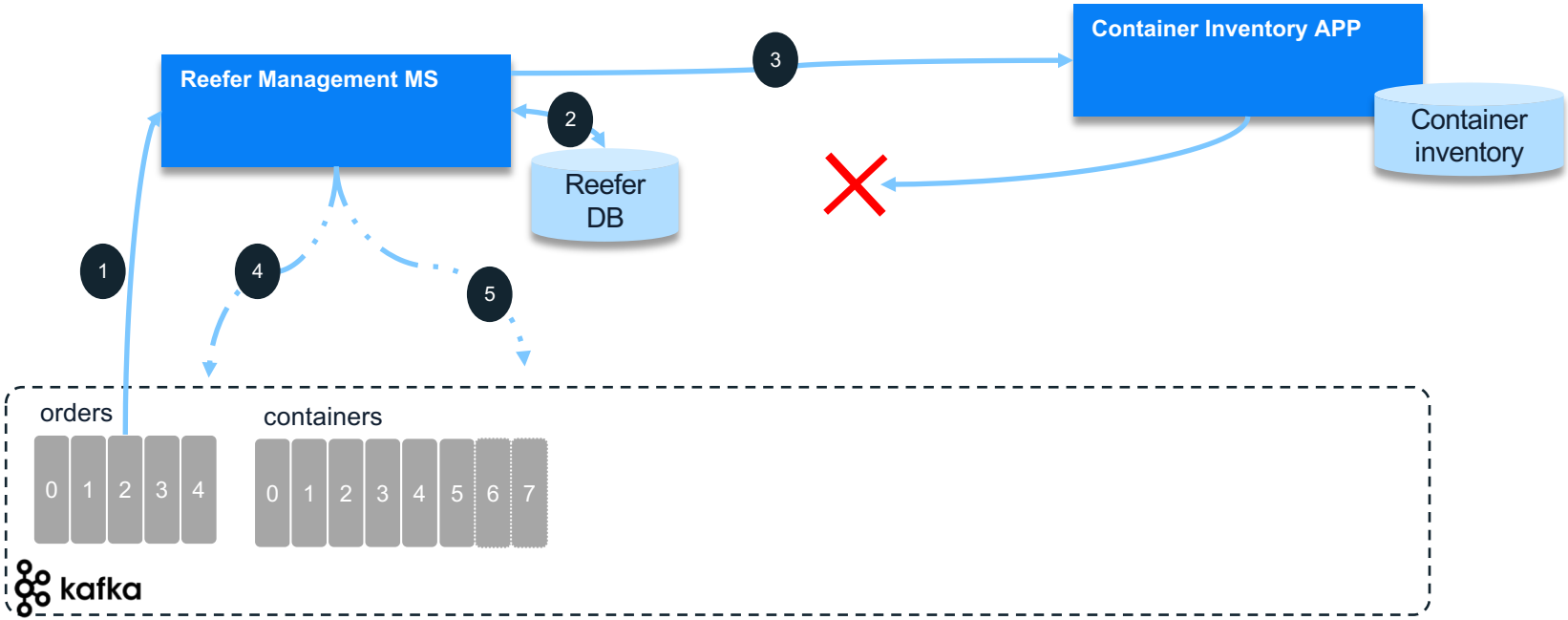


Saga pattern

	Choreography	Orchestration
Pros	+ simple, does not require much effort to build + participants are loosely coupled.	Avoid cyclic dependencies between services Centralize the orchestration of the distributed transaction in one state machine Participants are simpler as they respond to command and send events
Cons	- difficult to track which services listen to which events	Risk to have dump service and big brain orchestrator design command and response event structure
Fail over	Producers must ensure event delivery and idempotence to do not generate same order event multiple times Delivery once	Same logic needs to be done, but centralized in the orchestration component. Write to the Order response topics needs to be transactional
Rollback	Create compensation plan in the producer	Create compensation plan in the orchestrator

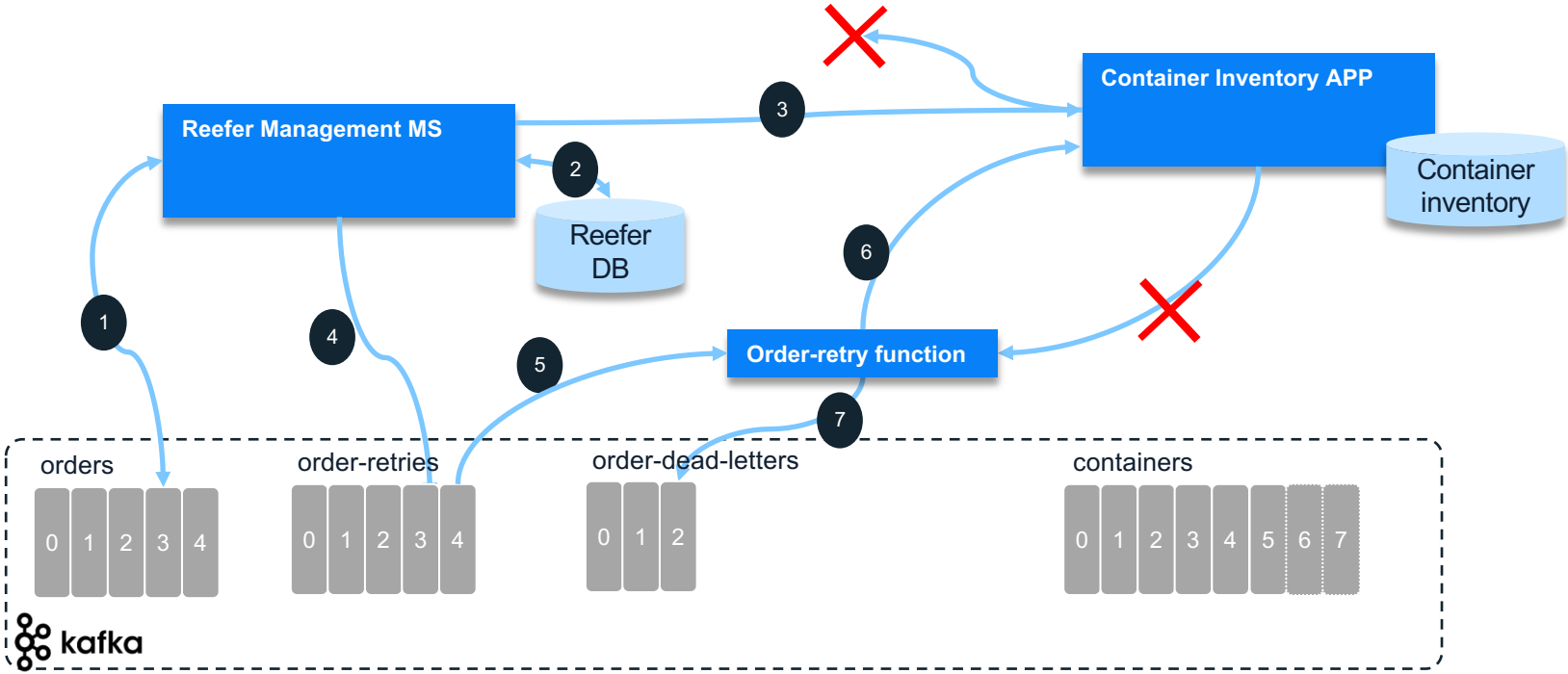
4. Dead letter queues

Reprocessing and dead letter



https://ibm-cloud-architecture.github.io/refarch-eda/patterns/dlq/

Reprocessing and dead letter



IBM

Event Driven Architecture

Additional resources

Cloud Garage Solution Engineering

Additional resources

- Event-Driven Reference Architecture on IBM Cloud Garage Architecture Center

- IBM Garage for Cloud Solution Engineering
  - Event-Driven Architecture Workspace
  - Reference Implementation

- IBM Event Streams
- IBM Cloud Pak for Integration