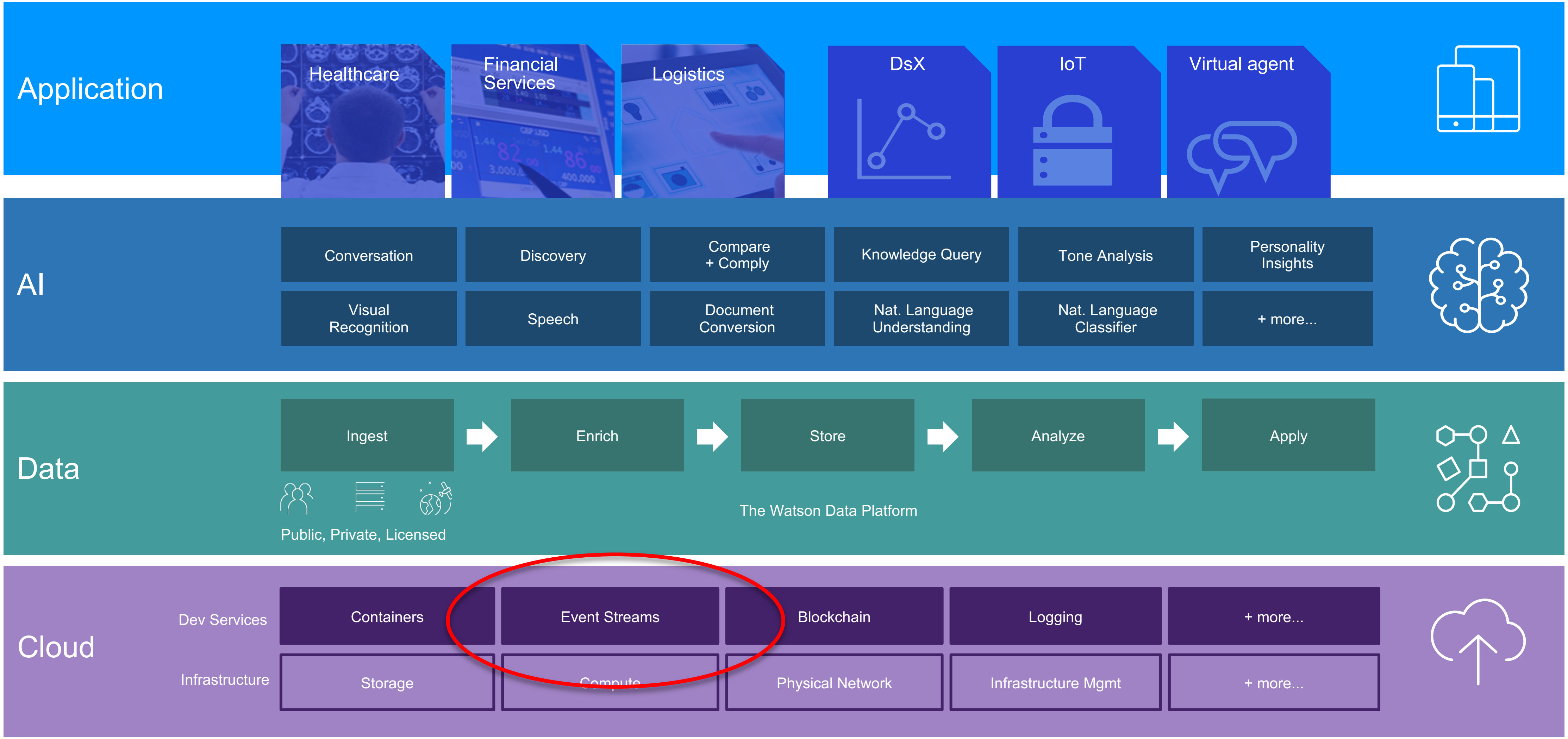
Intro to Kafka and Event Streams

Remko de Knikker
Developer Advocate

IBM **Developer**

IBM Watson and Cloud Platform



Application

Healthcare

Financial Services

Logistics

DsX

IoT

Virtual agent

AI

Conversation

Discovery

Compare + Comply

Knowledge Query

Tone Analysis

Personality Insights

Visual Recognition

Speech

Document Conversion

Nat. Language Understanding

Nat. Language Classifier

+ more...

Data

Ingest → Enrich → Store → Analyze → Apply

Public, Private, Licensed

The Watson Data Platform

Cloud

Dev Services

Containers

Event Streams

Blockchain

Logging

+ more...

Infrastructure

Storage

Compute

Physical Network

Infrastructure Mgmt

+ more...

IBM Event Streams is built on Apache Kafka

“Kafka adoption was accelerating…with production deployments of Kafka claiming **six of the top 10** travel companies, **seven of the top 10** global banks, **eight of the top 10** insurance companies, and **nine of the top 10** US telecom companies."

IBM Event Streams is built on Apache Kafka

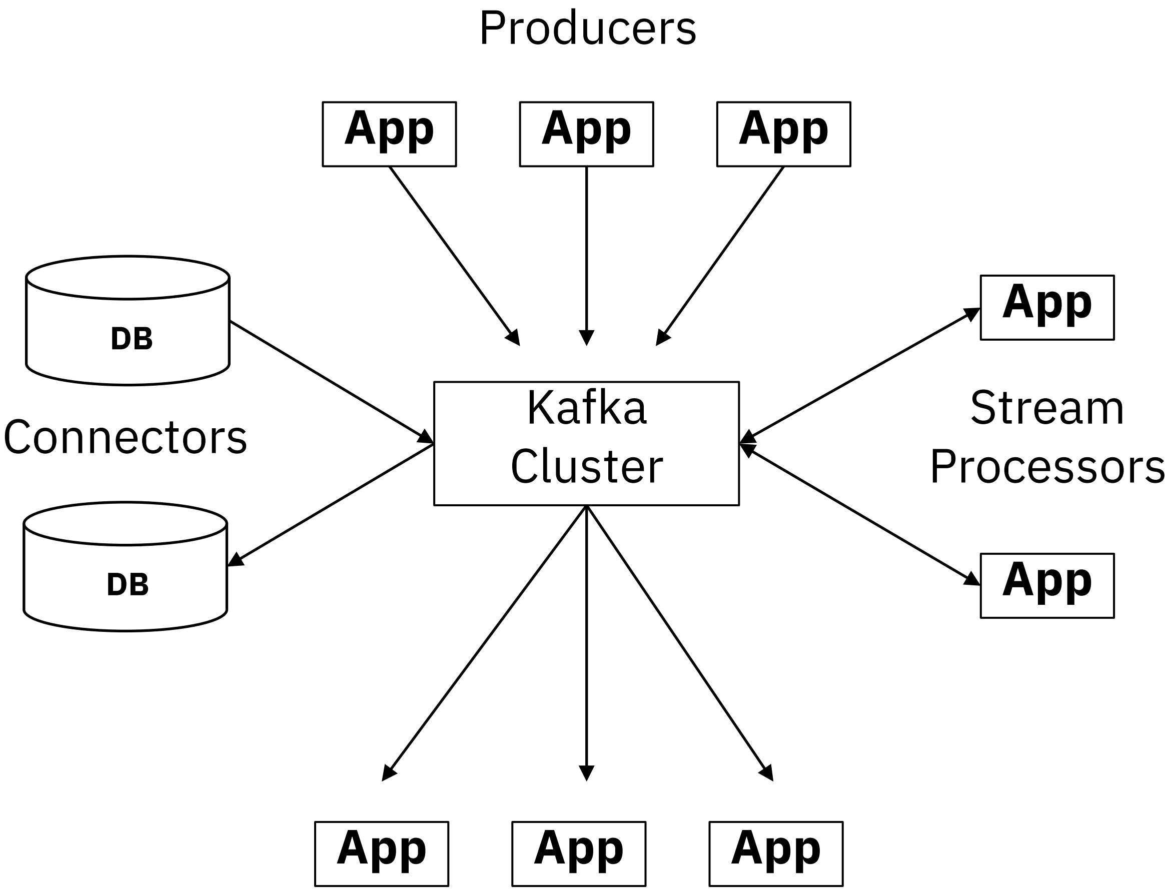
Adding years of operational expertise of running Apache Kafka for enterprises.

- Enterprise Connectors
- Geo-replication
- Enterprise grade security
- Schema registry with support for Apache Avro schemas
- Tight Integration with IBM IoT Platform, Object Storage, Streaming Analytics, Cloud Functions
- Scalable REST API
- CLI support for automation
- Enterprise Platform support: Red Hat OpenShift, IBM Z
- Fully Managed or Container certified,
- Zero-down time upgrade
- 24/7 Support

Apache Kafka

High-throughput message bus

Kafka is a Streaming Platform



Apache Kafka

A streaming platform has three key capabilities:

- Publish and subscribe to streams of data, similar to a message queue or enterprise messaging system.
- Store streams of data in a fault-tolerant durable way.
- Process streams of data on-demand.

Apache Kafka

Kafka is generally used for:

- Real-time, reliable streaming data pipelines between systems
- Real-time streaming applications that transform or react to the streams of data

Apache Kafka

Kafka includes five core APIs:

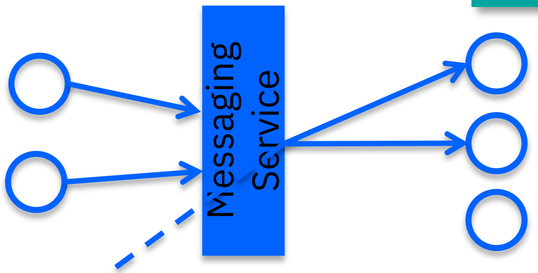
- The Producer API to send streams of data to topics.
- The Consumer API to read streams of data from topics.
- The Streams API allows transforming streams of data from input to output topics.
- The Connect API for implementing connectors to pull from a source system or push into a sink system.
- The AdminClient API for managing and inspecting topics, brokers, and other Kafka objects.

Use Cases

Worker Offload

Intensive work offloaded and distributed amongst worker processes to be performed asynchronously

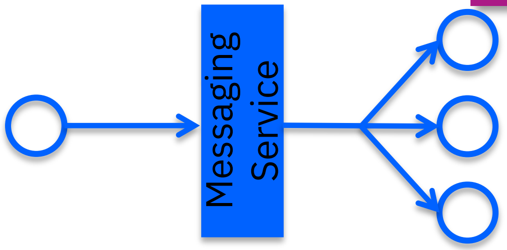
- Processing images or videos
- Performing text analytics



Event-Driven

Take one or more actions when something interesting happens

- Email logs and update dashboards when build finishes
- Upload videos once finished transcoding

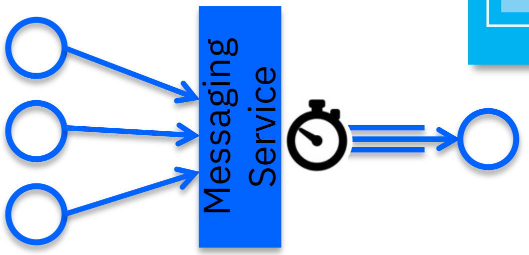


Use Cases

Delayed Processing

Schedule a task to happen at a specific time

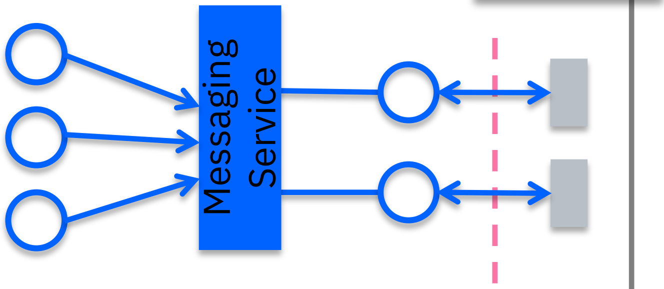
- Run detailed reports when app use is low
- Generate end of day summary



3rd Party Integration

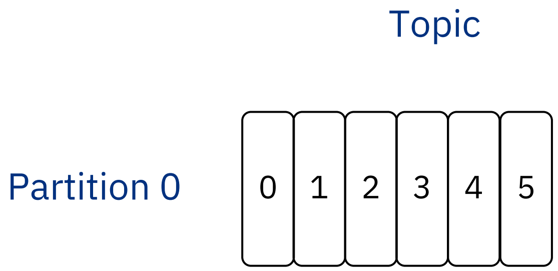
Ensure applications remain responsive even when 3rd party systems are not available or responding fast enough

- Updating existing CRM system
- Booking appointment



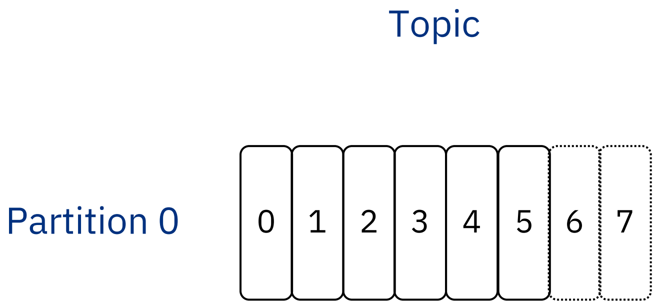
Kafka is: Built for Scale

Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log.



Kafka is: Built for Scale

Records in the partitions are each assigned a sequential id number called the *offset.*



Topic

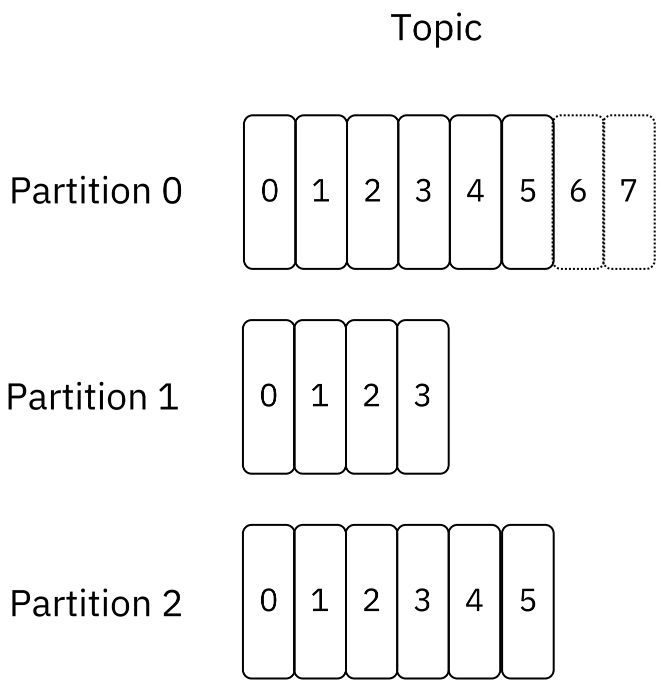
Partition 0

0 1 2 3 4 5 6 7

Kafka partitions messages for scale

Each topic consists of 1 or more partitions.

This topic has 3 partitions so the load can be distributed across 3 servers.



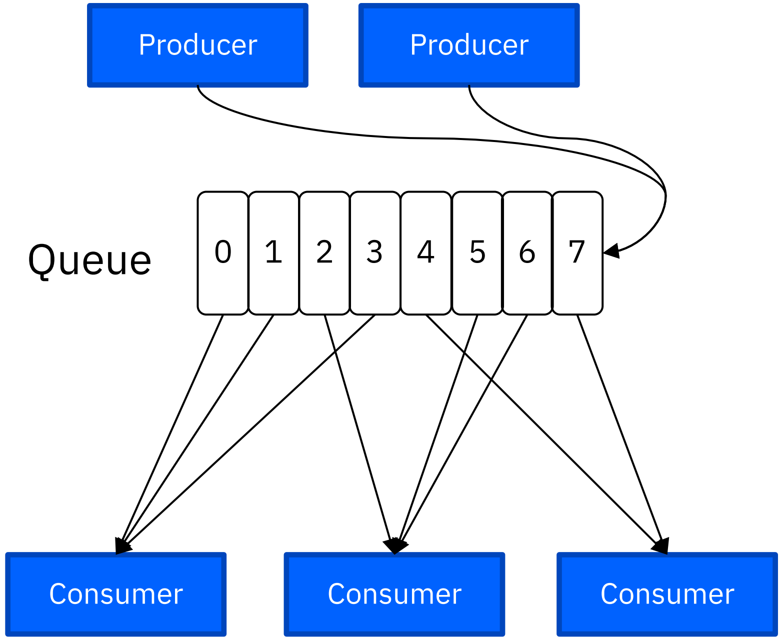
Partitions allow the log to scale beyond the size of a single server. Each individual partition must fit on the hosting server.

Publishers can influence partitioning.

Each partition is an ordered list of messages.

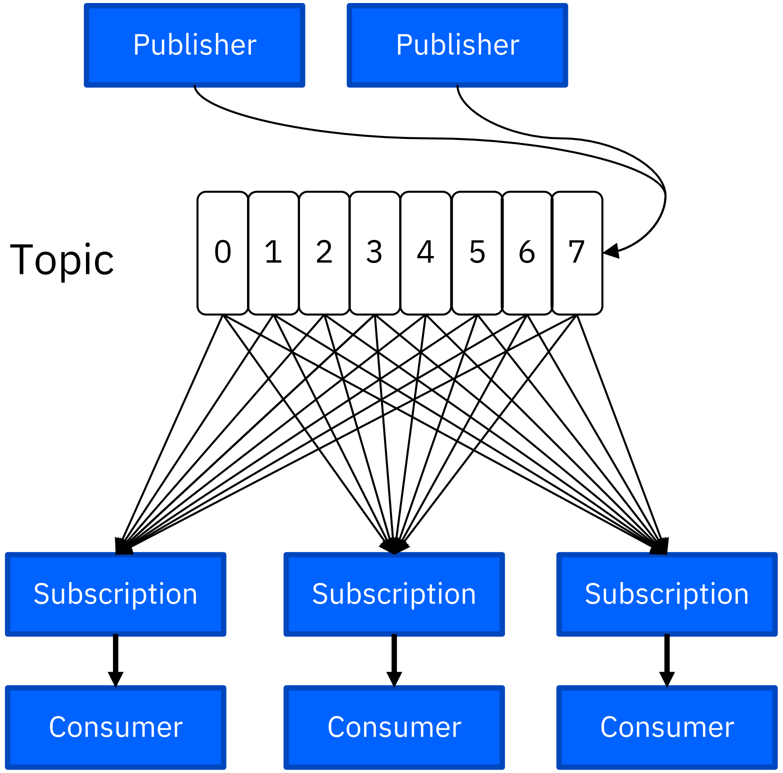
Partitioning and ordering are linked concepts in Kafka.

Queues vs topics



- Each message has 1 consumer
- Gives easy sharing among consumers

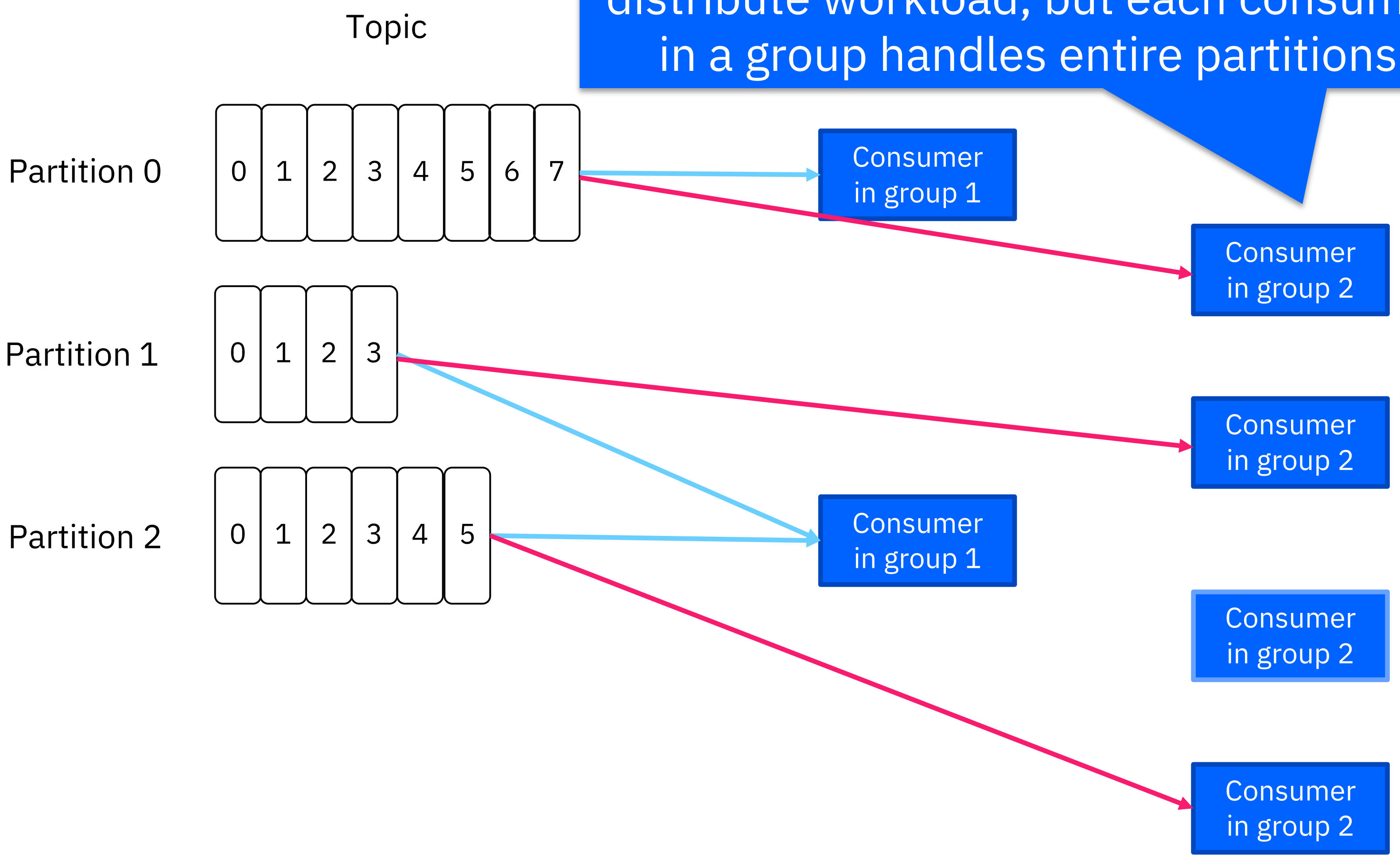
Publish-Subscribe and Queueing



- Each message has n consumers
- Hybrid models such as shared subscriptions or consumer groups exist

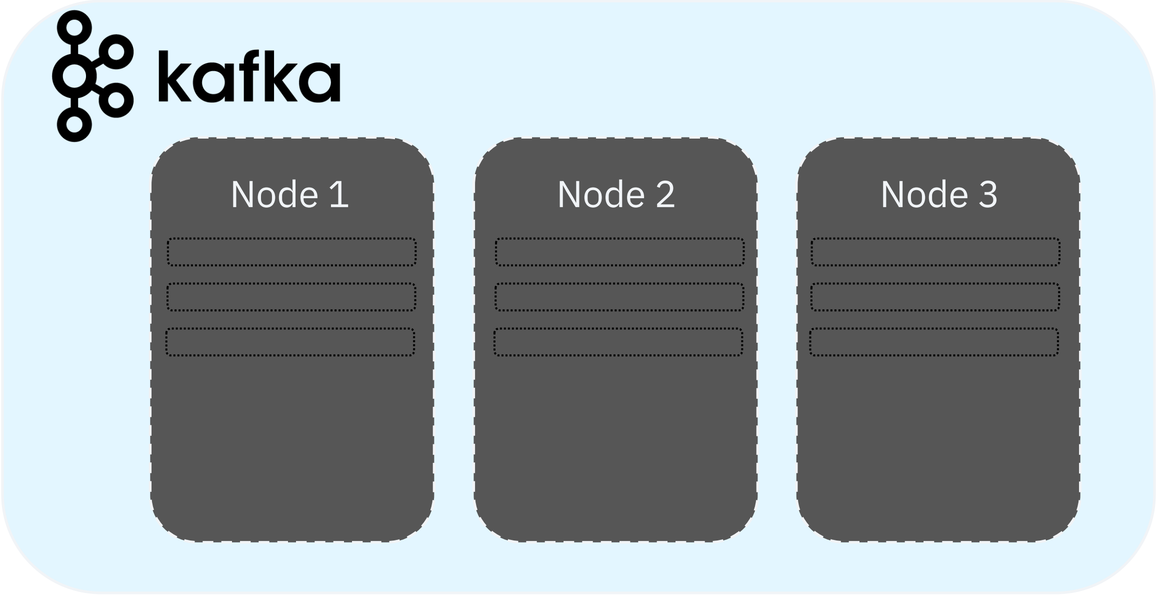
Workload distribution in Kafka

Consumer groups can be used to distribute workload, but each consumer in a group handles entire partitions



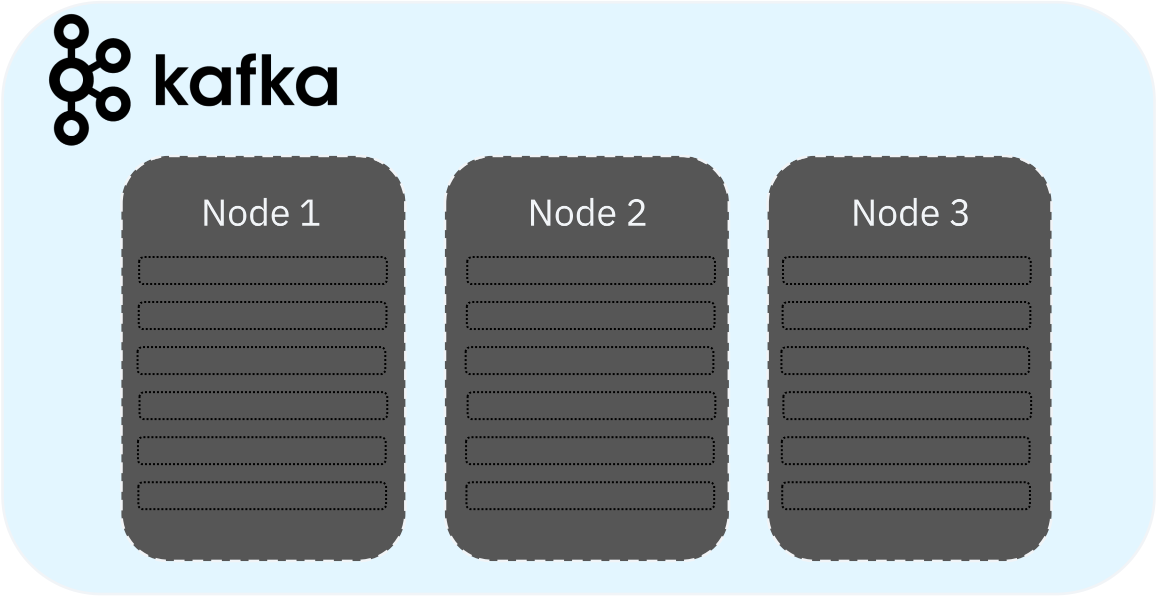
Kafka is: Built for Scale

Kafka is run as a cluster with one or more nodes or servers.



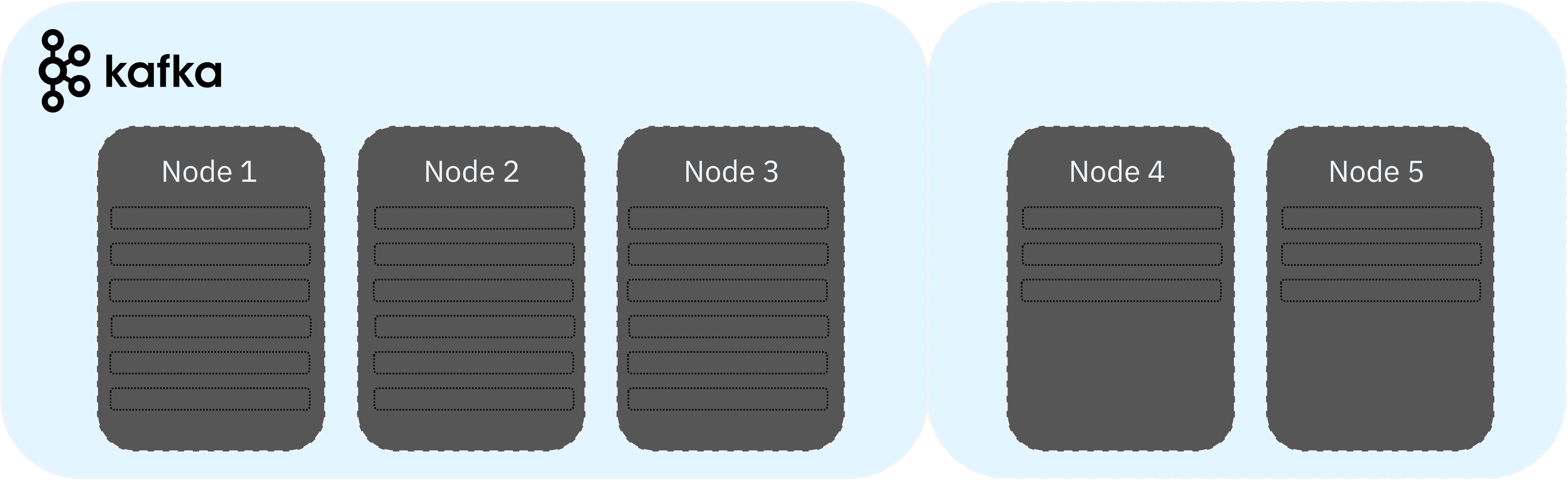
Kafka is: Built for Scale

Kafka is run as a cluster with one or more nodes or server, each node with one or more partitions.

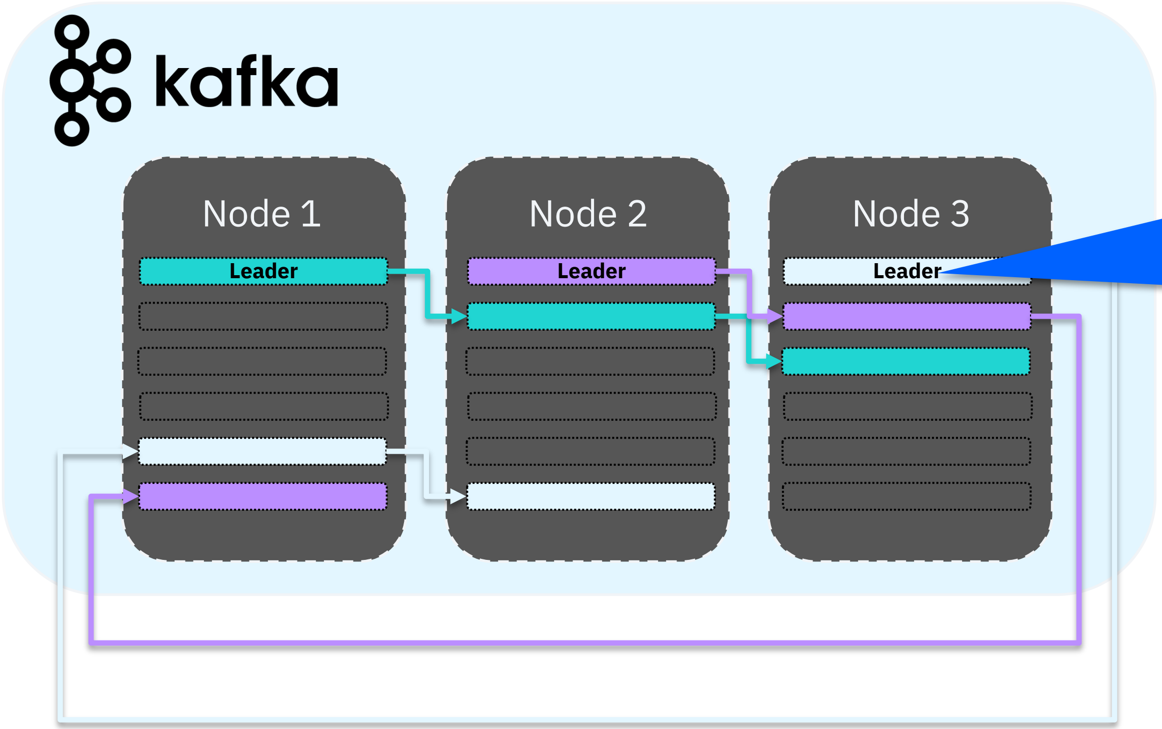


Kafka is: Built for Scale

Kafka is run as a cluster with one or more nodes or server, each node with one or more partitions. A cluster can span multiple datacenters.



Kafka replicates across the nodes in a cluster



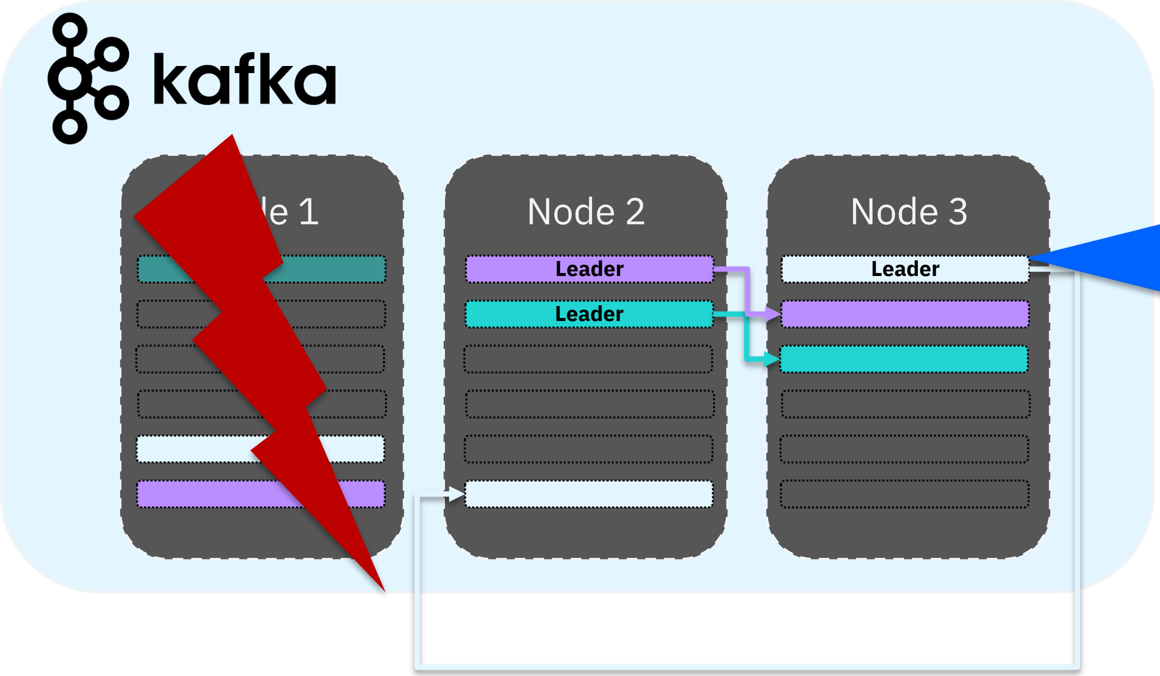
Each partition has one server acting as the "leader" and zero or more servers as "followers".

This is synchronous replication of ordered data streams. It is not just eventually consistent.

Each partition is replicated across a configurable number of servers for fault tolerance.

Kafka MirrorMaker provides geo-replication support for your clusters.

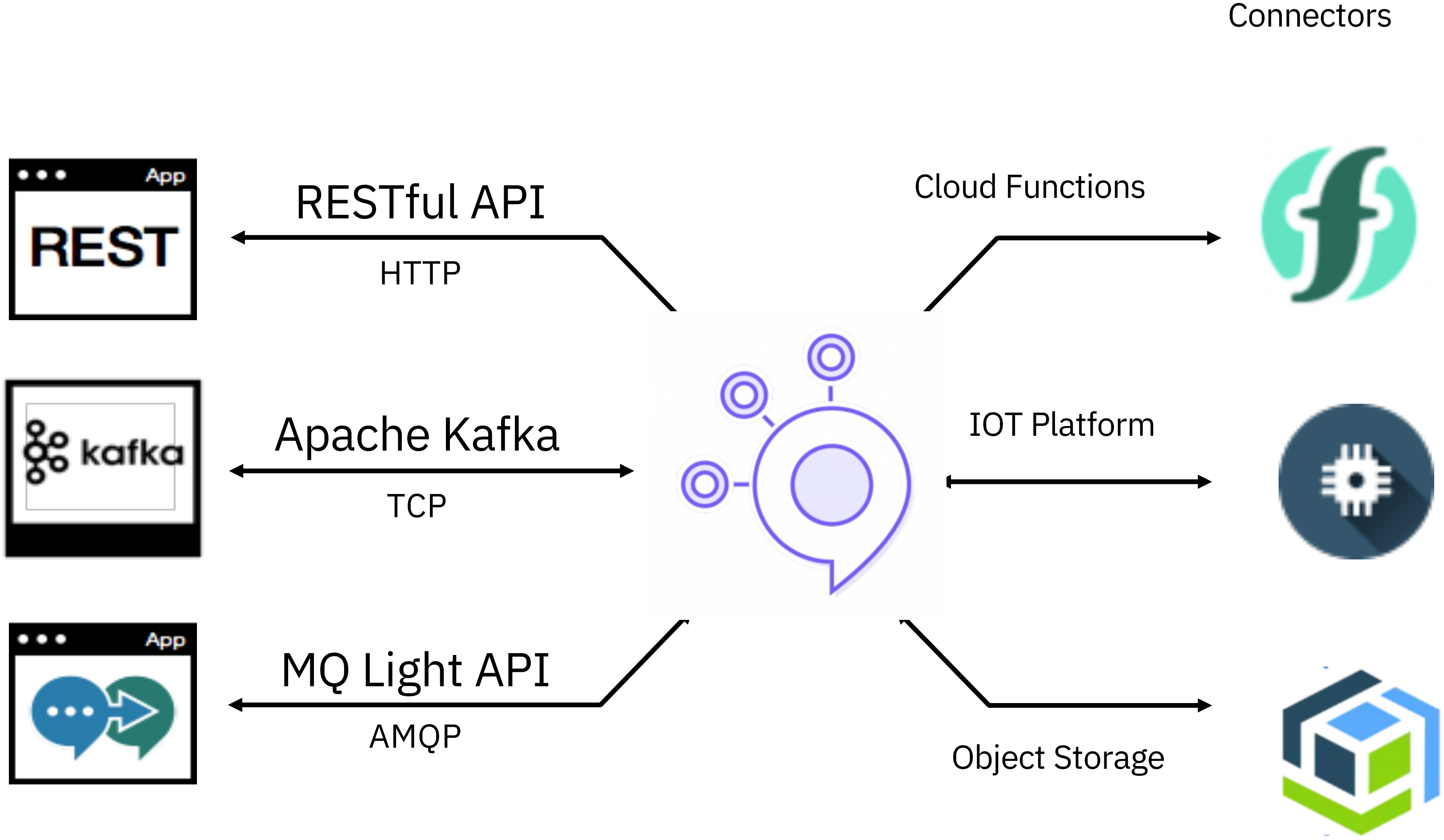
When a node fails, leaders are reassigned

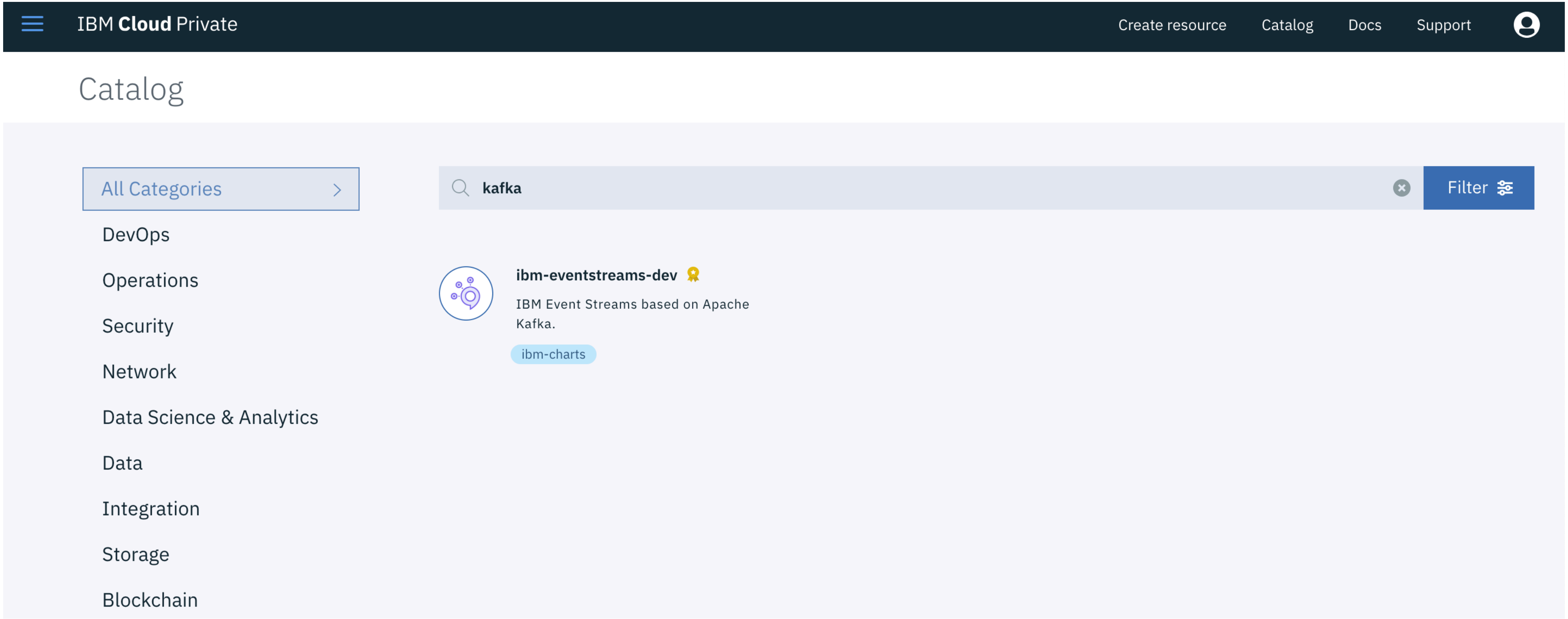


Each partition has a leader responsible for replicating the messages, in order, to the other replicas.

Recovery from node failure is very fast. There is no data loss and no need to wait for propagation of changes to get consistency restored.

IBM Event Streams



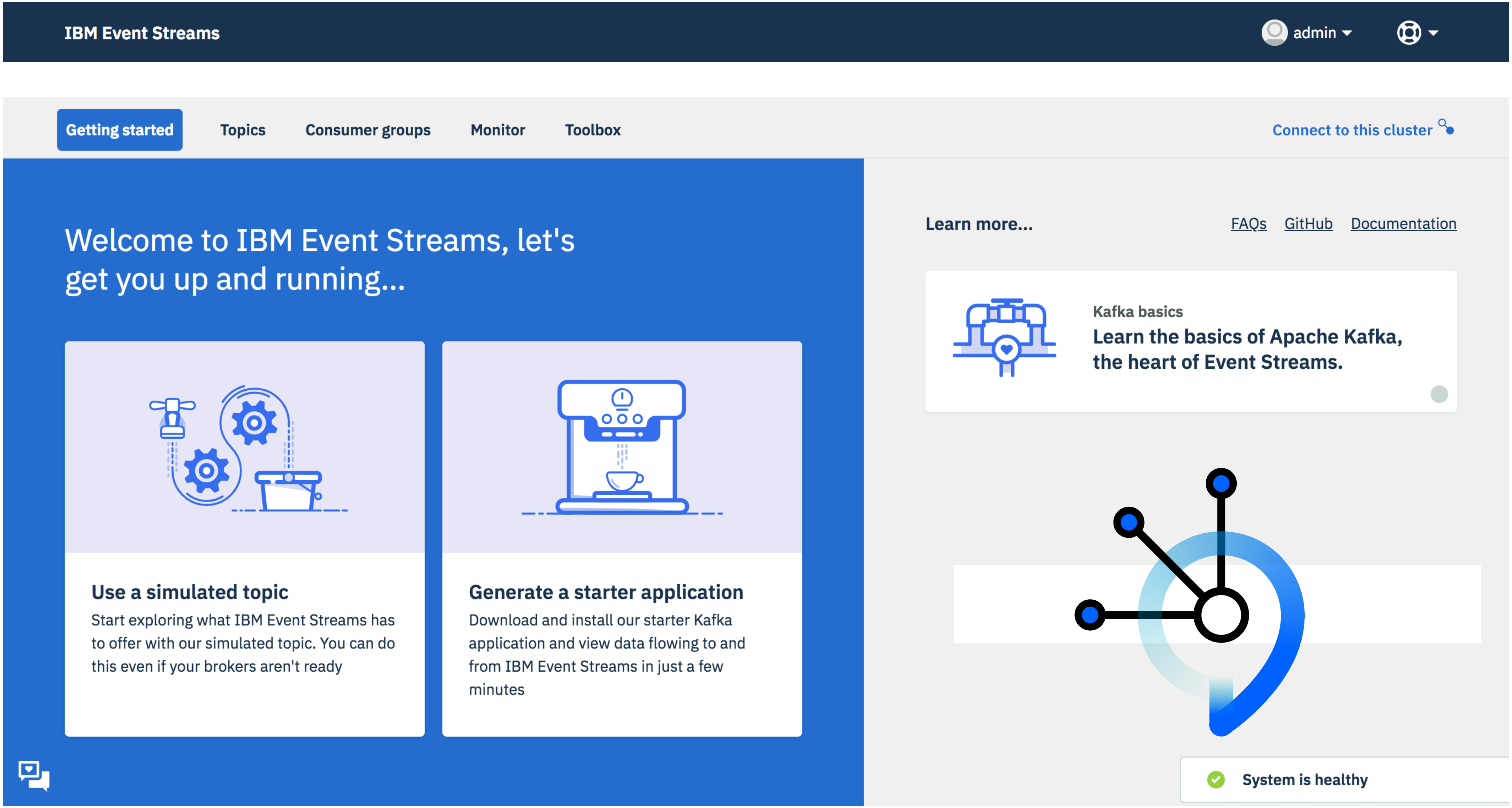


kafka

ibm-eventstreams-dev

IBM Event Streams based on Apache Kafka.

ibm-charts



IBM Event Streams

admin

Getting started

Topics

Consumer groups

Monitor

Toolbox

Connect to this cluster

Welcome to IBM Event Streams, let's get you up and running...

Use a simulated topic

Start exploring what IBM Event Streams has to offer with our simulated topic. You can do this even if your brokers aren't ready

Generate a starter application

Download and install our starter Kafka application and view data flowing to and from IBM Event Streams in just a few minutes

Learn more...

FAQs

GitHub

Documentation

Kafka basics

Learn the basics of Apache Kafka, the heart of Event Streams.

✓ System is healthy

Kafka Clients and APIs

Kafka APIs

Core APIs

- **Producer**
  - send messages to a topic
- **Consumer**
  - get messages from a topic
- **Streams**
  - transform streams from input topics to output topics
- **Connect**
  - pull from a source system / application into Kafka or push to a sink system /application
- **Admin Client APIs**

REST Proxy

https://github.com/confluentinc/kafka-rest

Other language clients (external to kafka.apache.org)

- librdkafka (C/C++)
- Node.js
- PHP
- Python
- Ruby
- C#/ .NET
- Go
- Rust
- Haskell
- OCaml

Producer API using Kafka Console Tools

Kafka.properties

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="USER" password="PASSWORD";

security.protocol=SASL_SSL

sasl.mechanism=PLAIN

ssl.protocol=TLSv1.2

ssl.enabled.protocols=TLSv1.2

ssl.endpoint.identification.algorithm=HTTPS
```

```
$ bash kafka-console-producer.sh --broker-list broker-1-a1bc2d3efg4hijkl.kafka.svc01.us-
south.eventstreams.cloud.ibm.com:9999,

broker-2-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,

broker-3-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,

broker-4-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,

broker-5-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,

broker-6-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999

--producer.config mykafka.properties --topic greetings
```

Producer API

```
import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerRecord;

import org.apache.kafka.clients.producer.RecordMetadata;


public KafkaProducer<byte[], byte[]> kafkaProducer;

Properties props = new Properties();


kafkaProducer = new KafkaProducer<byte[], byte[]>(props);


ProducerRecord<byte[], byte[]> record = new ProducerRecord<byte [], byte[]>(topic, key.getBytes("UTF-8"),
     value.getBytes(UTF-8)));


RecordMetadata m = kafkaProducer.send(record).get();


System.out.println("Message produced, offset: " + m.offset());
```

The producer is *thread safe*, sharing a single instance across threads will be faster

Producer API using Spring Boot for Kafka

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

@RestController
public class EventsStreamController {
  @Autowired
  private KafkaTemplate<String, String> template;
  private List<String> messages = new CopyOnWriteArrayList<>();
  @GetMapping(value = "/send/{msg}")
  public void send(@PathVariable String msg) throws Exception {
      this.template.sendDefault(msg);
  }
}
```

Consumer API using Kafka Console Tools

Kafka.properties

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="USER"
password="PASSWORD";
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
ssl.protocol=TLSv1.2
ssl.enabled.protocols=TLSv1.2
ssl.endpoint.identification.algorithm=HTTPS
```

```
bash kafka-console-consumer.sh --bootstrap-server broker-1-a1bc2d3efg4hijkl.kafka.svc01.us-
south.eventstreams.cloud.ibm.com:9999,
broker-2-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,
broker-3-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,
broker-4-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,
broker-5-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999,
broker-6-a1bc2d3efg4hijkl.kafka.svc01.us-south.eventstreams.cloud.ibm.com:9999
--consumer.config ../mykafka.properties --topic greetings –group 1
```

Consumer API

```
import org.apache.kafka.clients.consumer.KafkaProducer;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.ByteArrayDeserializer;

class ConsumerRunnable implements Runnable {
  private KafkaConsumer<byte[], byte[]> kafkaConsumer;
  ...
  ConsumerRunnable(...) {
    Properties props = new Properties();

    kafkaConsumer = new KafkaConsumer<byte[], byte[]>(props, new ByteArrayDeserializer, new ByteArrayDeserializer);
    kafkaConsumer.subscribe(topicList);
  }

   run() {
    while (...) {
      ConsumerRecord<byte[], byte[]> record = kafkaConsumer.poll(1000);
      System.out.println("Message consumed: " + record.value());
    }
  }
}
```

The consumer is **not** *thread safe*. The simple approach is to give each thread its own instance.

Consumer API using Spring Boot for Kafka

```
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

@RestController
public class EventsStreamController {
  @Autowired
  private KafkaTemplate<String, String> template;
  private List<String> messages = new CopyOnWriteArrayList<>();
  @KafkaListener(topics = "${listener.topic}", groupId = "channel1")
  public void listen(ConsumerRecord<String, String> cr) throws Exception {
      messages.add(cr.value());
  }
  @GetMapping("/received")
  public String recv() throws Exception {
```

REST API

Easy way to produce and consume messages – limited performance

```
# Get a list of topics

$ curl "http://kafkahost/topics"

[{"name":"test","num_partitions":3},{"name":"test2","num_partitions":1}]
```

```
# Produce a message using JSON with the value '{ "foo": "bar" }' to the topic test

$ curl -X POST -H "Content-Type: application/vnd.kafka.json.v1+json" --data '{"records":[{"value":{"foo":"bar"}}]}'  "http://kafkahost/topics/test"
{"offsets":[{"partition":0,"offset":0,"error_code":null,"error":null}],"key_schema_id":null,"value_schema_id":null}
```

```
# Create a consumer, read a message using the consumer, delete the consumer

$ curl -X POST -H "Content-Type: application/vnd.kafka.v1+json"  --data '{"format": "json", "auto.offset.reset": "smallest"}' \
http://kafkahost/consumers/my_json_consumer

{"instance_id":"rest-consumer-11561681-8ba5-4b46-bed0-
905ae1769bc6","base_uri":"http://kafkahost/consumers/my_json_consumer/instances/rest-consumer-11561681-8ba5-4b46-bed0-
905ae1769bc6"}

$ curl -X GET -H "Accept: application/vnd.kafka.json.v1+json" http://kafkahost/consumers/my_json_consumer/instances/rest-consumer-
11561681-8ba5-4b46-bed0-905ae1769bc6/topics/test

[{"key":null,"value":{"foo":"bar"},"partition":0,"offset":0}]

$ curl -X DELETE \ http://localhost:8082/consumers/my_json_consumer/instances/rest-consumer-11561681-8ba5-4b46-bed0-905ae1769bc6
```

IBM Event Streams – Admin REST API

API-based administrative controls for topics and bridges

`GET /admin/topics`	- returns the current list of topics
`POST /admin/topics`	- create a topic using values in request body
`DELETE /admin/topics/{name}`	- delete a topic with name {name}

`GET /admin/bridges`	- returns the current list of bridges
`POST /admin/bridges`	- creates a new instance of an Event Streams bridge
`GET /admin/bridges/{name}`	- get information about a bridge
`DELETE /admin/bridges/{name}`	- delete a bridge with name {name}
`PATCH,PUT /admin/bridges/{name}`	- update settings of bridge name {name}

IBM Event Streams CLI

```
ibmcloud es init
ibmcloud es broker
ibmcloud es broker-config
ibmcloud es cluster
ibmcloud es topic
ibmcloud es topic-create
ibmcloud es topic-delete
ibmcloud es topic-delete-records
ibmcloud es topic-partitions-set
ibmcloud es topic-update
ibmcloud es topics
ibmcloud es group
ibmcloud es group-reset
ibmcloud es groups
ibmcloud es group-delete
```

Lab Time