redhat.

Meet Apache Kafka

Data streaming in your hands

Paolo Patierno
Principal Software Engineer @ Red Hat
21/11/2018

Who am I ?

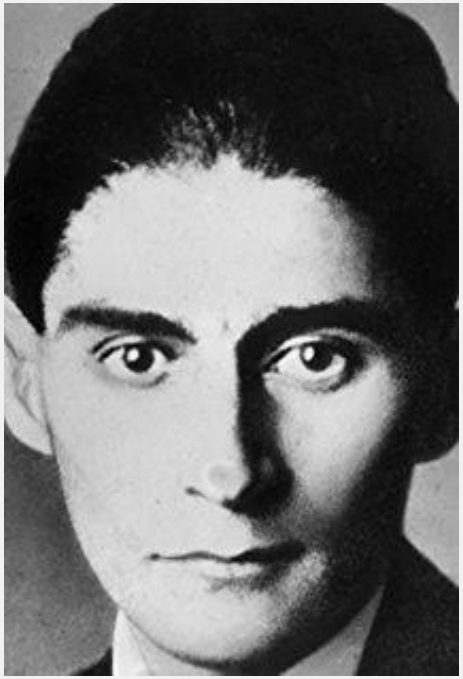
@ppatierno

- Principal Software Engineer @ Red Hat
  - Messaging & IoT team
- Lead/Committer @ Eclipse Foundation
  - Hono, Paho and Vert.x projects
- Microsoft MVP Azure/IoT
- Hacking low constrained devices in spare time (from previous life)
- Dad of two
- VR46's fan (I don't like MM93)
- Try to be a runner ...

redhat.

Kafka ?

“ did you join the wrong meetup ?”



redhat.

What is Kafka ?

At the beginning ...

“ … a publish/subscribe messaging system …”

redhat.

What is Kafka ?

… today …

“ … a streaming

data platform …”

redhat.

What is Kafka ?

... but at the core

“ … a distributed,
horizontally-scalable,
fault-tolerant, commit log …”

redhat.

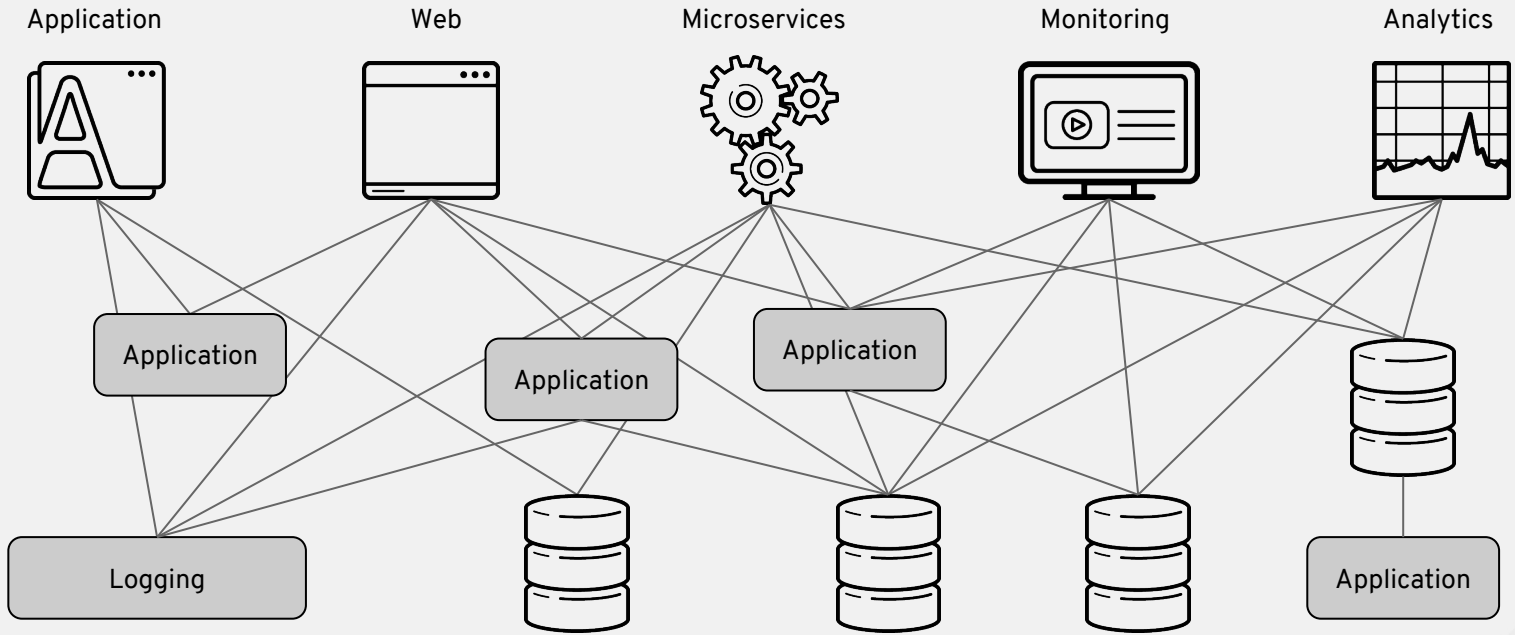
What is Kafka ?

- Developed at Linkedin back in 2010, open sourced in 2011
- Designed to be fast, scalable, durable and available
- Distributed by nature
- Data partitioning (sharding)
- High throughput / low latency
- Ability to handle huge number of consumers

redhat.

Applications data flow

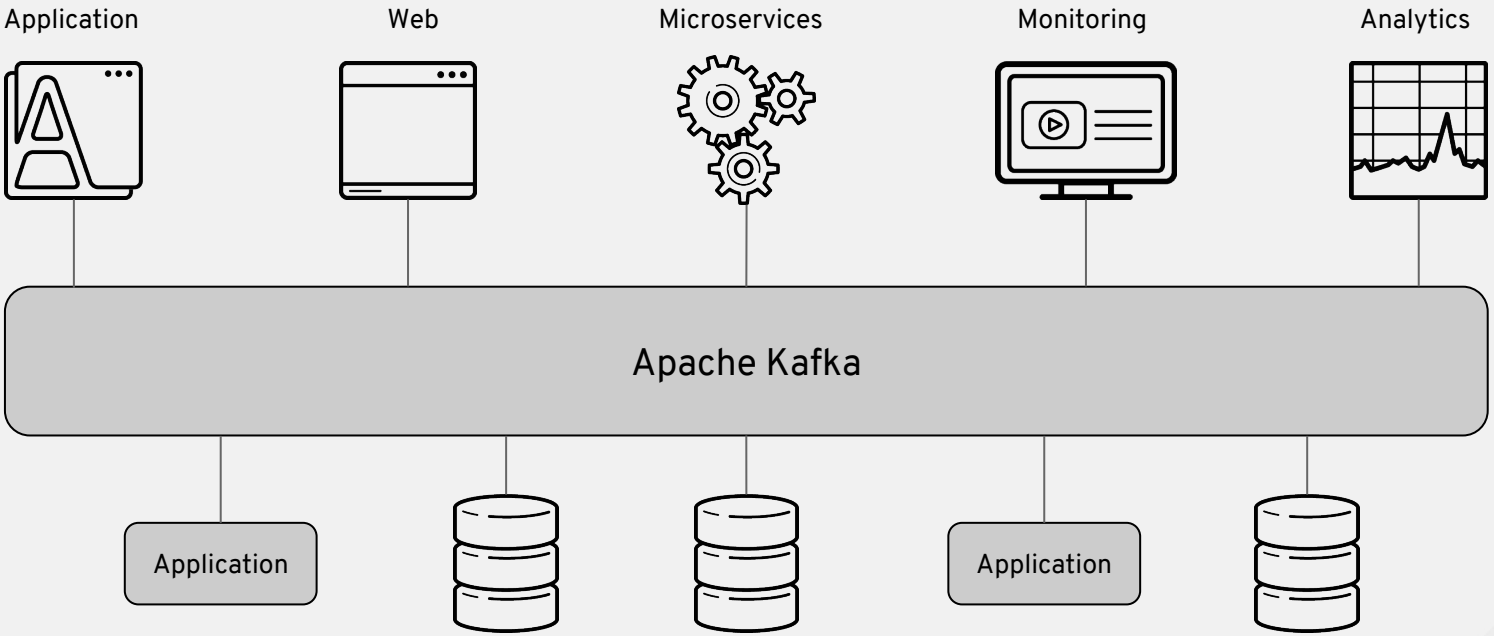
Without using Kafka



redhat.

Applications data flow

Kafka as a central hub



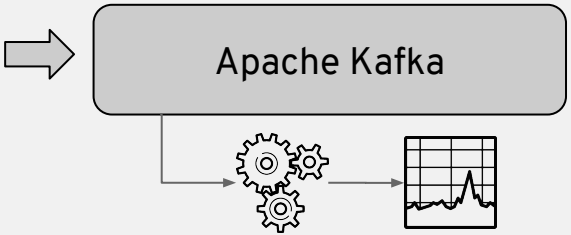
redhat.

Common use cases

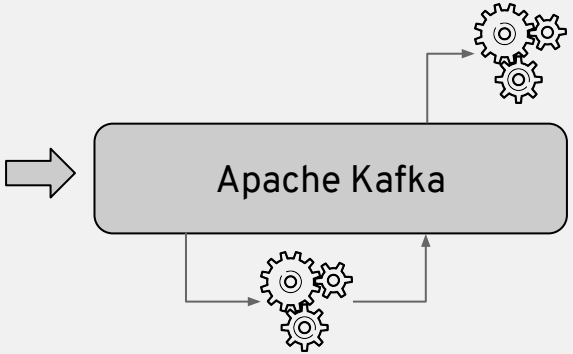
- Messaging
  - As the “traditional” messaging systems
- Website activity tracking
  - Events like page views, searches, …
- Operational metrics
  - Alerting and reporting on operational metrics
- Log aggregation
  - Collect logs from multiple services
- Stream processing
  - Read, process and write stream for real-time analysis

redhat.

Key use cases



Events buffer for data analytics/ML



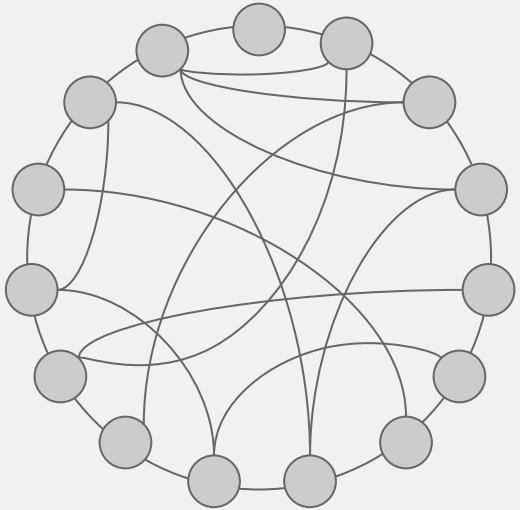
Event-driven microservices

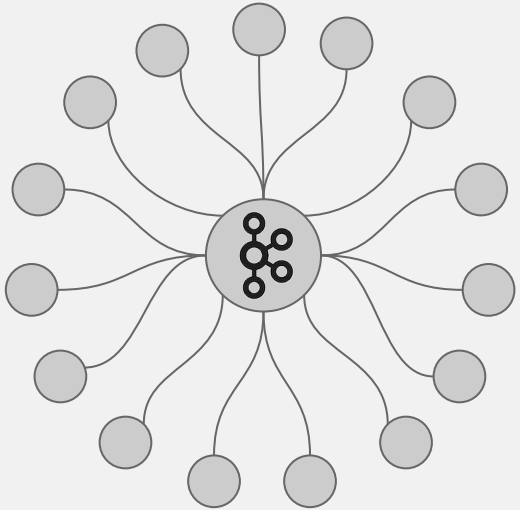
Bridge from on-prem to cloud native

redhat.

Microservices Architectures

Orchestration vs Choreography





redhat.

Apache Kafka ecosystem

- Apache Kafka has an ecosystem consisting of many components / tools
  - Kafka Core
    - Broker
    - Clients library (producer, consumer, admin)
  - Kafka Connect
  - Kafka Streams
  - Mirror Maker

redhat.

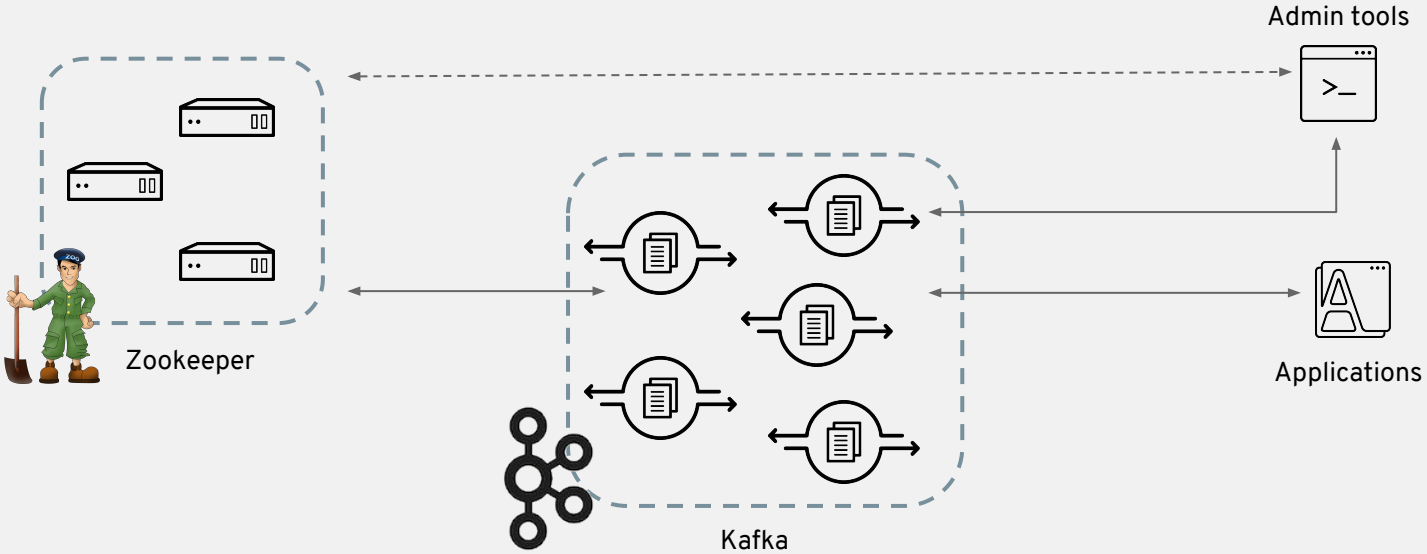
Apache Kafka ecosystem

Apache Kafka components

- **Kafka Broker**
  - Central component responsible for hosting topics and delivering messages
  - One or more brokers run in a cluster alongside with a Zookeeper ensemble
- **Kafka Producers and Consumers**
  - Java-based clients for sending and receiving messages
- **Kafka Admin tools**
  - Java- and Scala- based tools for managing Kafka brokers
  - Managing topics, ACLs, monitoring etc.

redhat.

Kafka & Zookeeper



redhat.

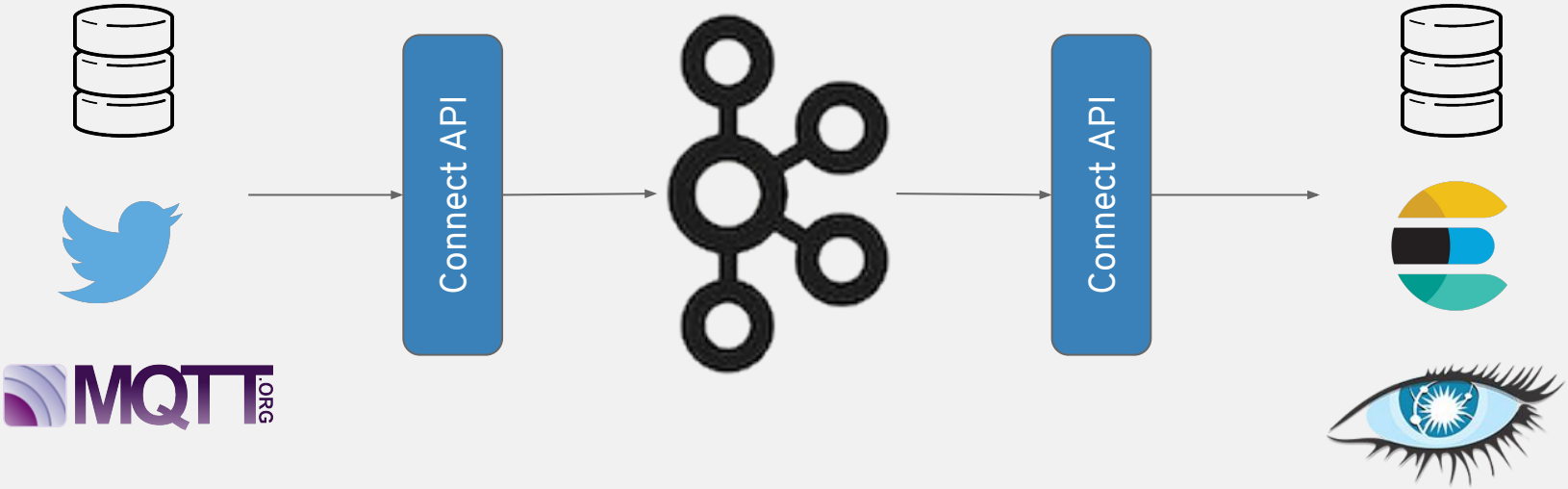
Kafka ecosystem

Apache Kafka components

- **Kafka Connect**
  - Framework for transferring data between Kafka and other data systems
  - Facilitate data conversion, scaling, load balancing, fault tolerance, …
  - Connector plugins are deployed into Kafka connect cluster
    - Well defined API for creating new connectors (with Sink/Source)
    - Apache Kafka itself includes only FileSink and FileSource plugins (reading records from file and posting them as Kafka messages / writing Kafka messages to files)
    - Many additional plugins are available outside of Apache Kafka
- Main use case CDC (Change Data Capture)

redhat.

Kafka Connect



redhat.

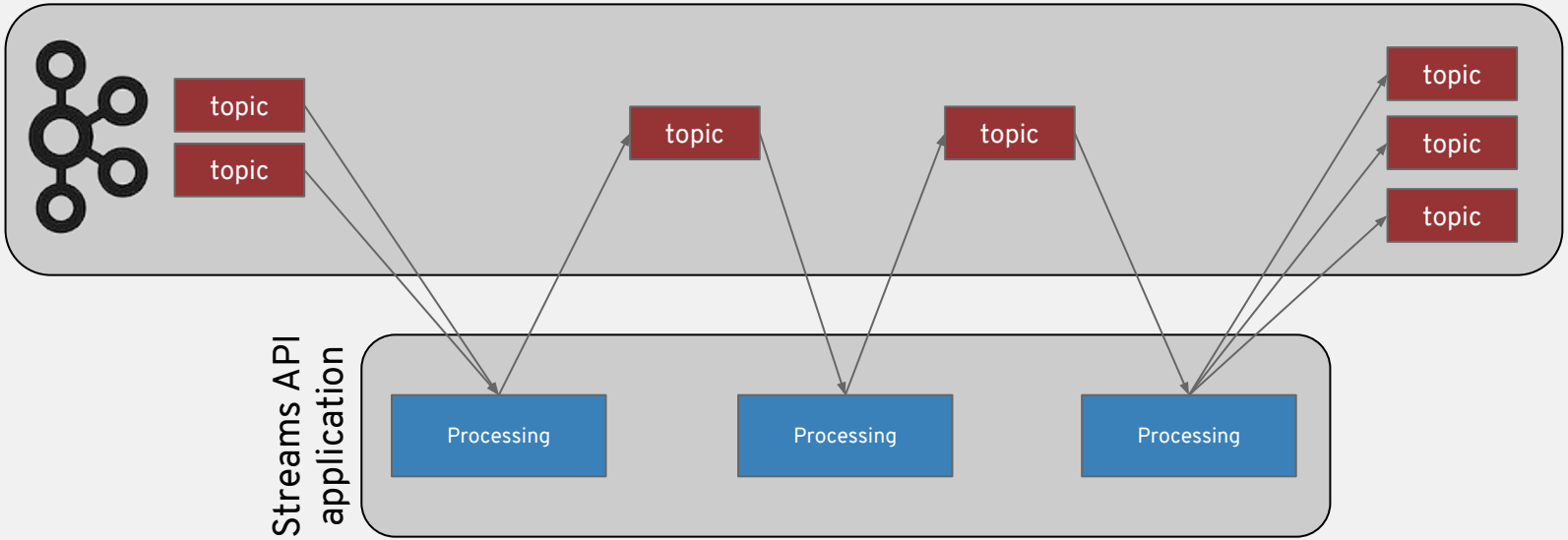
Kafka ecosystem

Apache Kafka components

- **Kafka Streams**
  - Stream processing framework
  - Streams are Kafka topics (as input and output)
  - It's really just a Java library to include in your application
  - Scaling the stream application horizontally
  - Creates a topology of processing nodes (filter, map, join etc) acting on a stream
    - Low level processor API
    - High level DSL
    - Using "internal" topics (when re-partitioning is needed or for "stateful" transformations)

redhat.

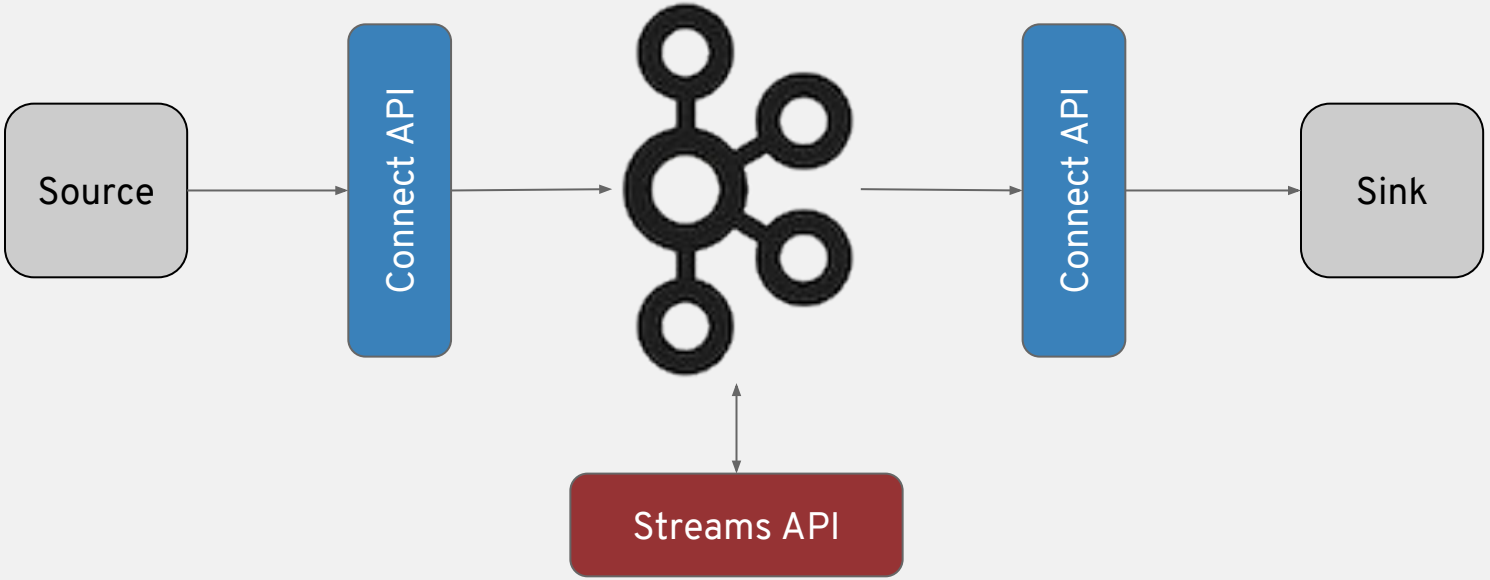
Kafka Streams API



redhat.

Kafka ecosystem

Extract ... Transform ... Load



redhat.

Kafka ecosystem

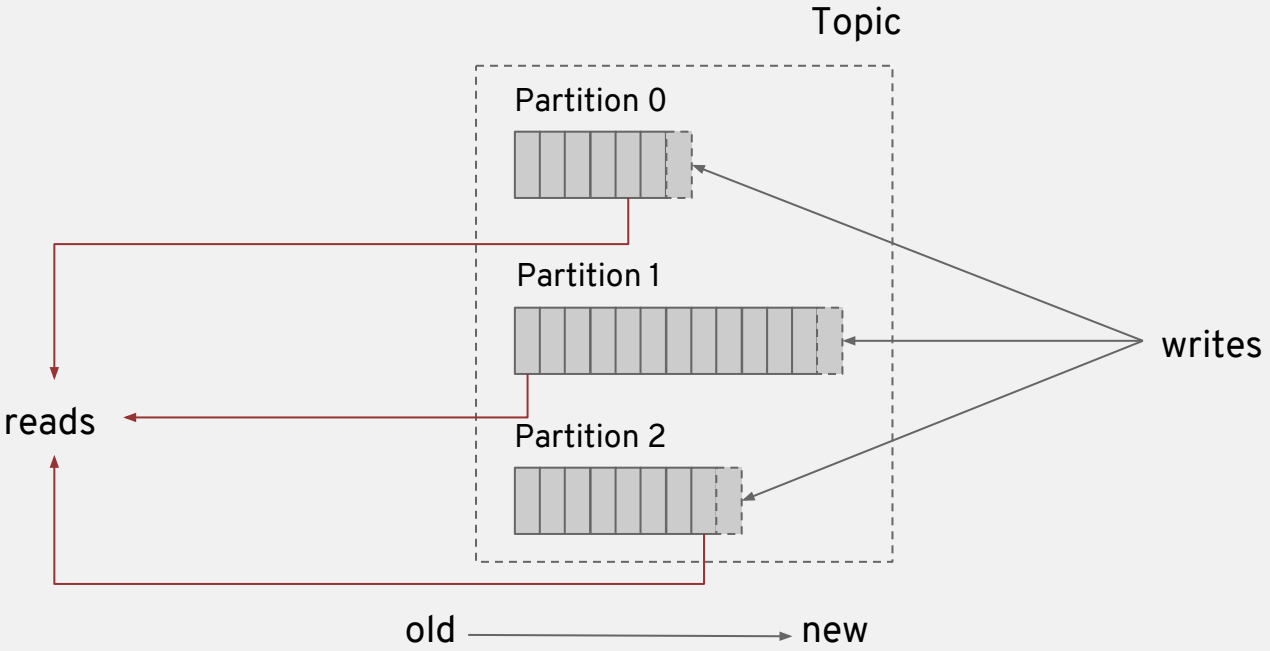
Apache Kafka components

- **Mirror Maker**
  - Kafka clusters do not work well when split across multiple datacenters
    - Low bandwidth, High latency
    - For use within multiple datacenters it is recommended to setup independent cluster in each data center and mirror the data
  - Tool for replication of topics between different clusters

redhat.

Topic & Partitions

Topic anatomy



redhat.

Replication

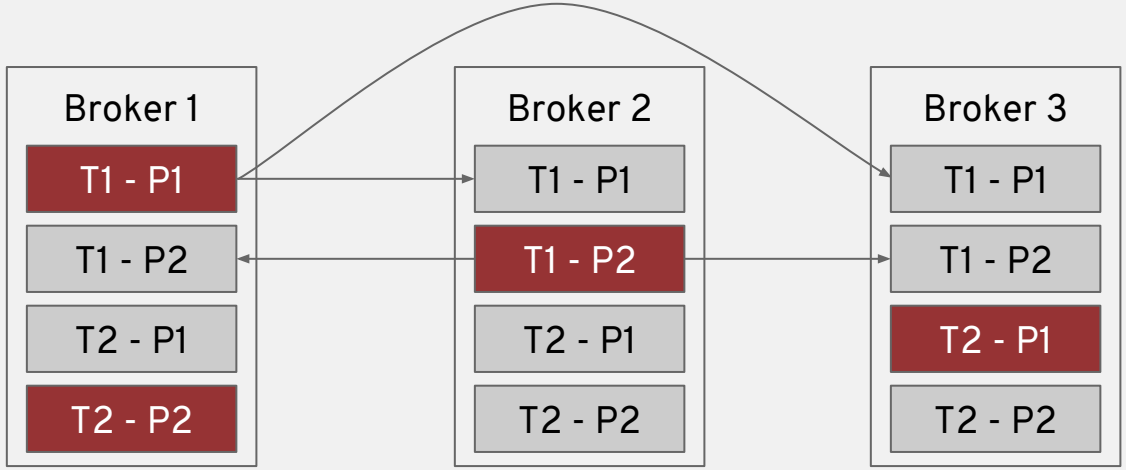
Leaders & Followers

- They are “backup” for a partition
  - Provide redundancy
- It’s the way Kafka guarantees availability and durability in case of node failures
  - Kafka tolerates “replicas - 1” dead brokers before losing data
- Two roles :
  - Leader : a replica used by producers/consumers for exchanging messages
  - Followers : all the other replicas
    - They don’t serve client requests
    - They replicate messages from the leader to be “in-sync” (ISR)
  - A replica changes its role as brokers come and go
- Messages ordering is “per partition”

redhat.

Partitions Distribution

Leaders & Followers

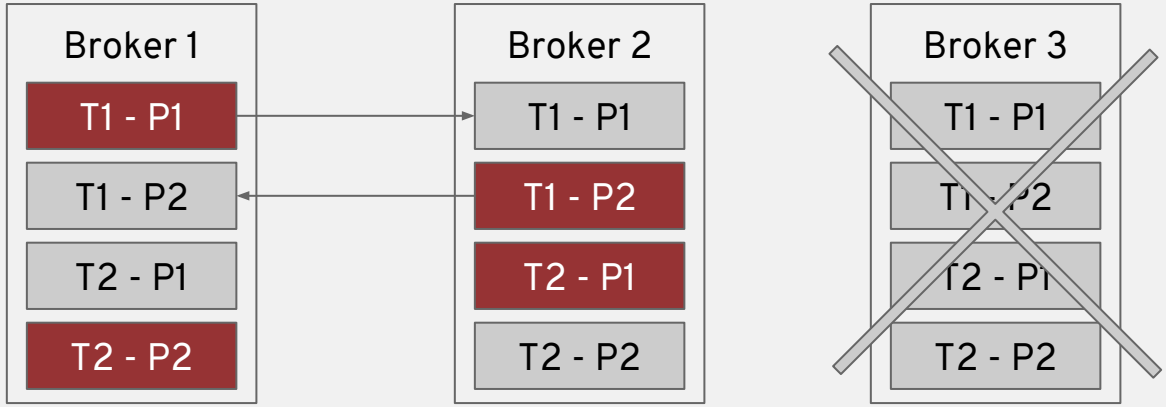


- Leaders and followers spread across the cluster
  - producers/consumers connect to leaders
  - multiple connections needed for reading different partitions

redhat.

Partitions Distribution

Leader election



- A broker with leader partition goes down
- New leader partition is elected on different node

redhat.

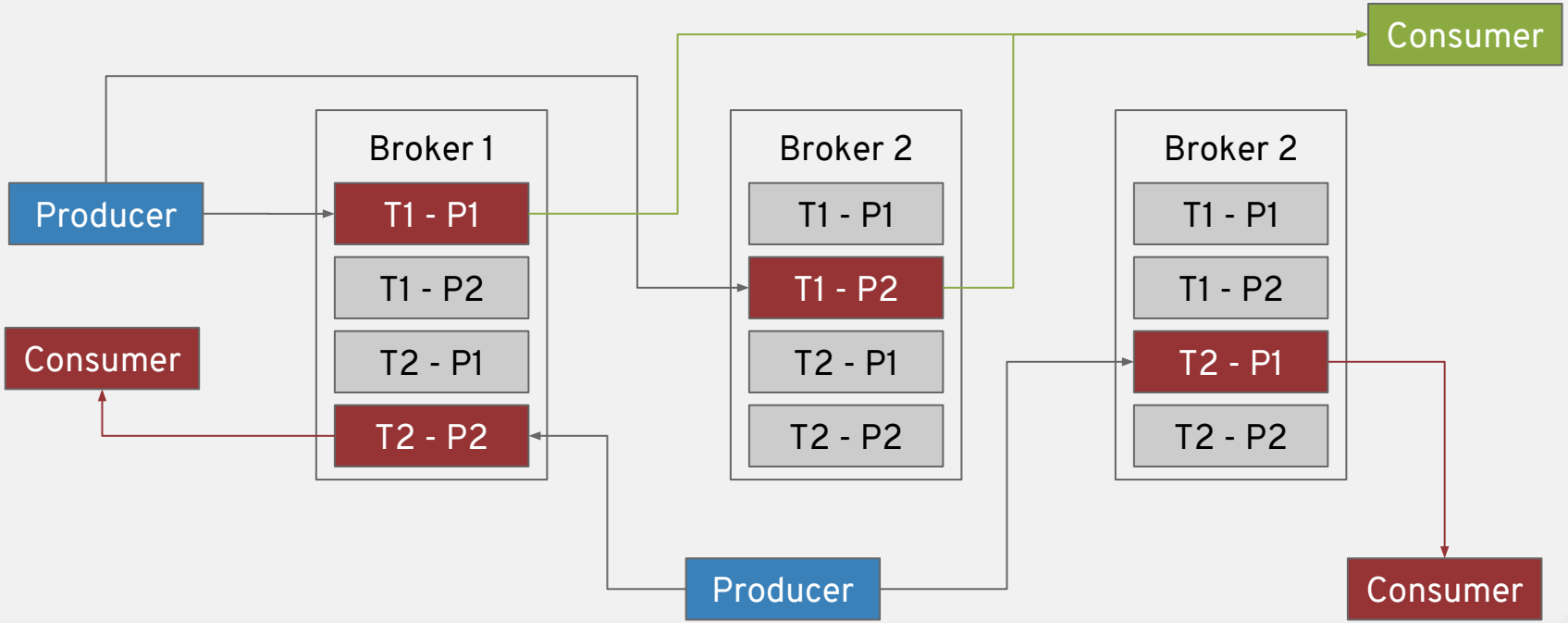
Clients

- They are really “smart” (unlike “traditional” messaging)
- Configured with a bootstrap servers list for fetching first metadata
  - Where are interested topics ? Connect to broker which holds partition leaders
  - Producer specifies destination partition
  - Consumer handles messages offsets to read
  - If error happens, refresh metadata (something is changed in the cluster)
- Batching on producing/consuming

redhat.

Producers & Consumers

Writing/Reading to/from leaders



redhat.

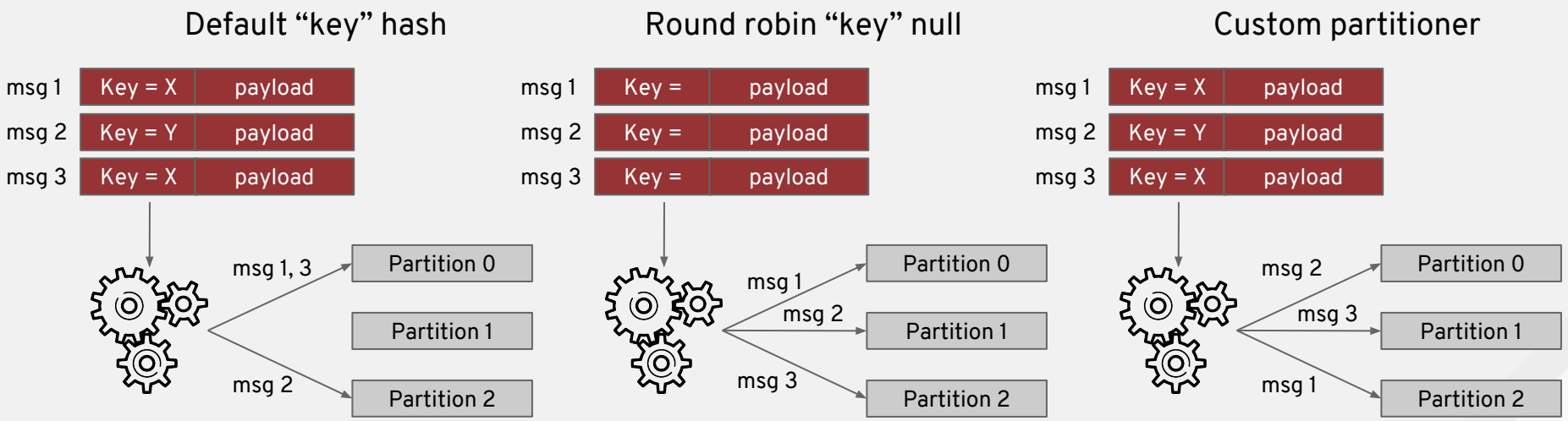
Producers

- Destination partition computed on client
  - Round robin
  - Specified by hashing the “key” in the message
  - Custom partitioning
- Writes messages to “leader” for a partition
- Acknowledge :
  - No ack
  - Ack on message written to “leader”
  - Ack on message also replicated to “in-sync” replicas

redhat.

Producers

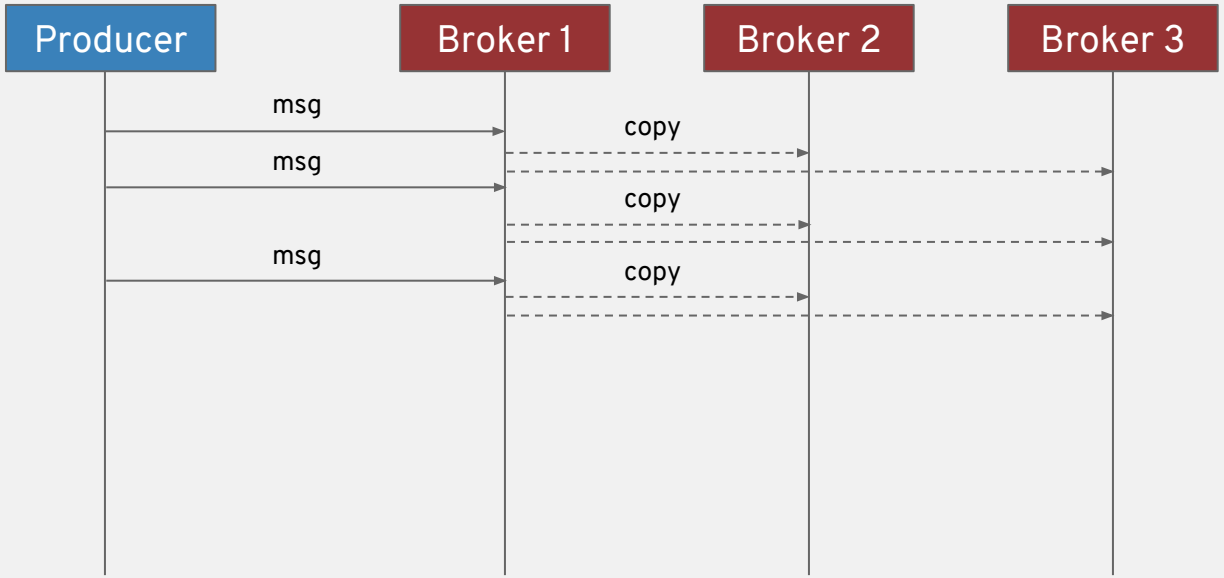
Partitioners



redhat.

Producers

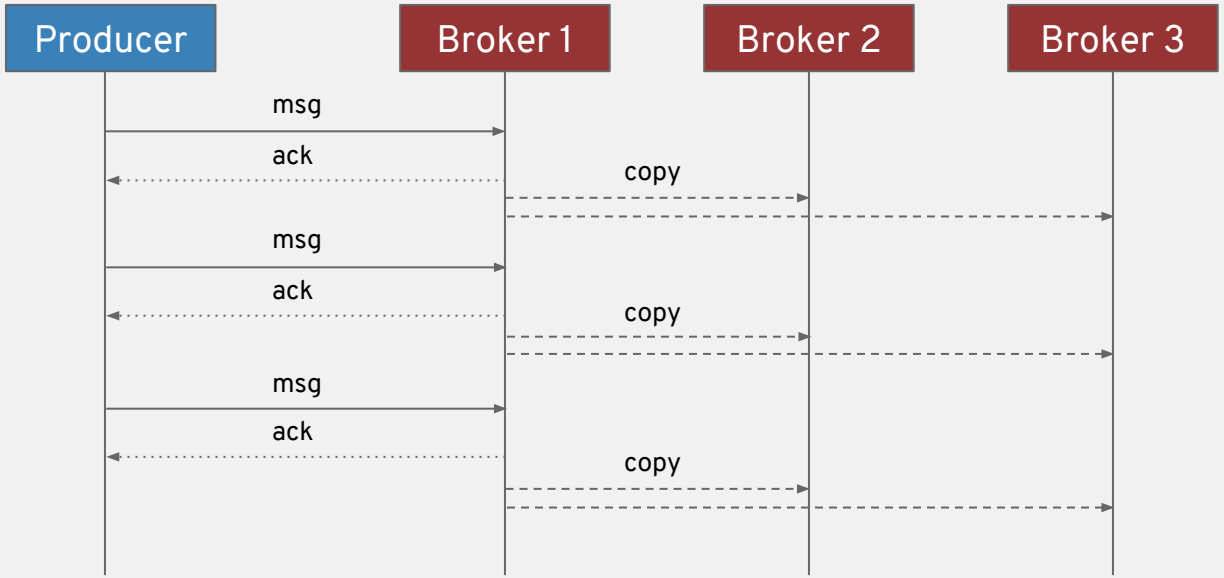
No ack



redhat.

Producers

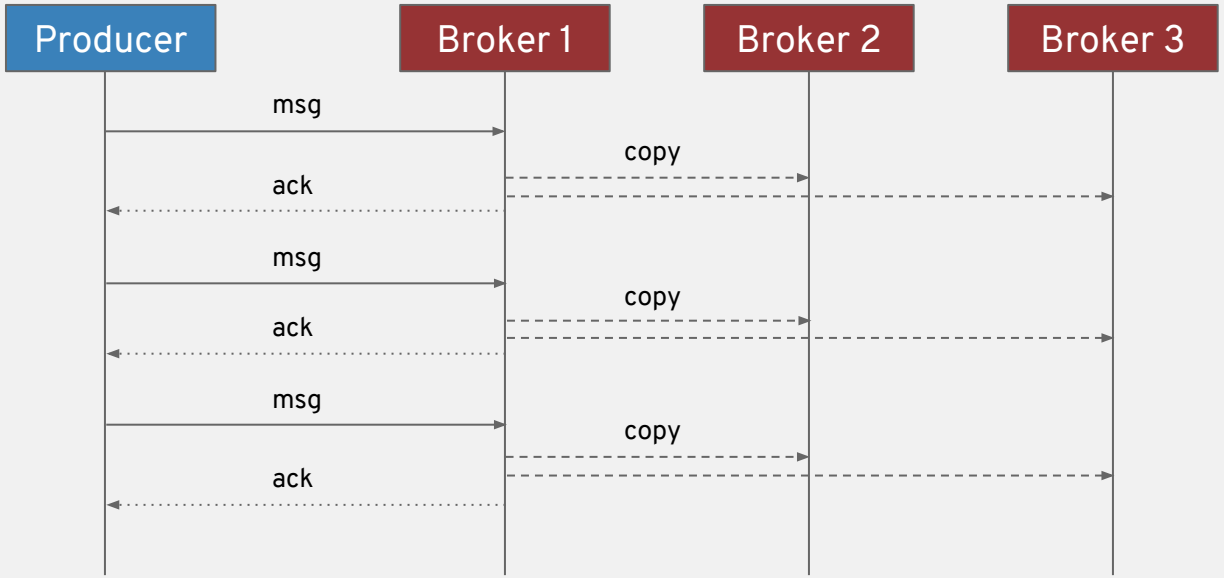
Ack on message written to “leader”



redhat.

Producers

Ack on message also replicated to “in-sync” replicas



redhat.

Consumers

- Read from one (or more) partition(s)
- Track (commit) the offset for given partition
  - A partitioned topic “__consumer_offsets” is used for that
  - Key → [group, topic, partition], Value → [offset]
  - Offset is shared inside the consumer group
- QoS
  - At most once : read message, commit offset, process message
  - At least once : read message, process message, commit offset
  - Exactly once : read message, commit message output and offset to a transactional system
- Gets only “committed” messages (depends on producer “ack” level)

redhat.

Consumer : partitions assignment

Available approaches

- The consumer asks for a specific partition (assign)
  - An application using one or more consumers has to handle such assignment on its own, the scaling as well
- The consumer is part of a “consumer group”
  - Consumer groups are an easier way to scale up consumption
  - One of the consumers, as “group lead”, applies a strategy to assign partitions to consumers in the group
  - When new consumers join or leave, a rebalancing happens to reassign partitions
  - This allows pluggable strategies for partition assignment (e.g. stickiness)

redhat.

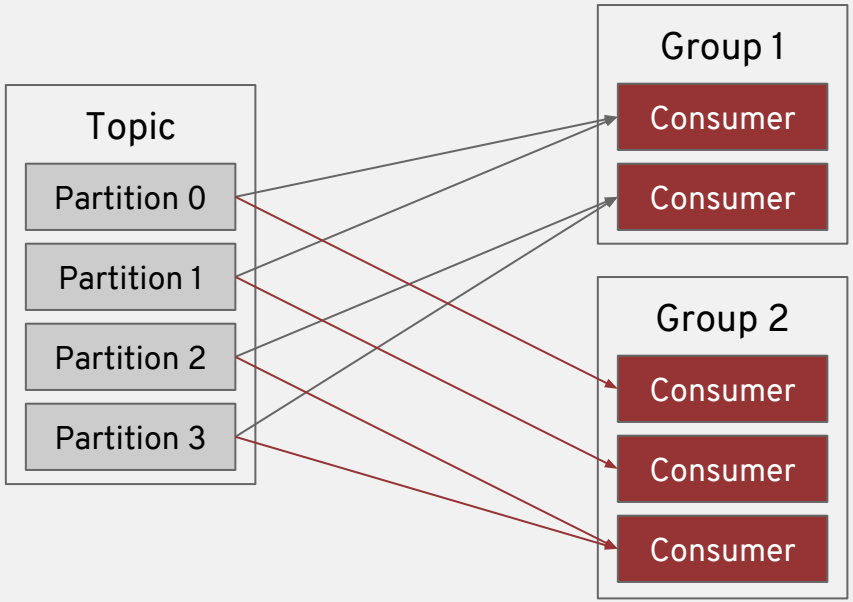
Consumer Groups

- Consumer Group
  - Grouping multiple consumers
  - Each consumer reads from a “unique” subset of partition → max consumers = num partitions
  - They are “competing” consumers on the topic, each message delivered to one consumer
  - Messages with same “key” delivered to same consumer
- More consumer groups
  - Allows publish/subscribe
  - Same messages delivered to different consumers in different consumer groups

redhat.

Consumer Groups

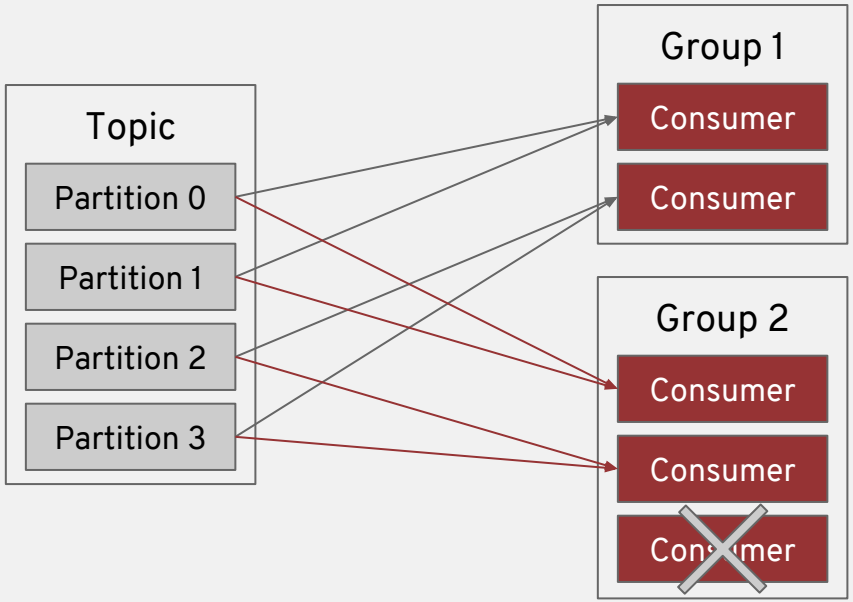
Partitions assignment



redhat.

Consumer Groups

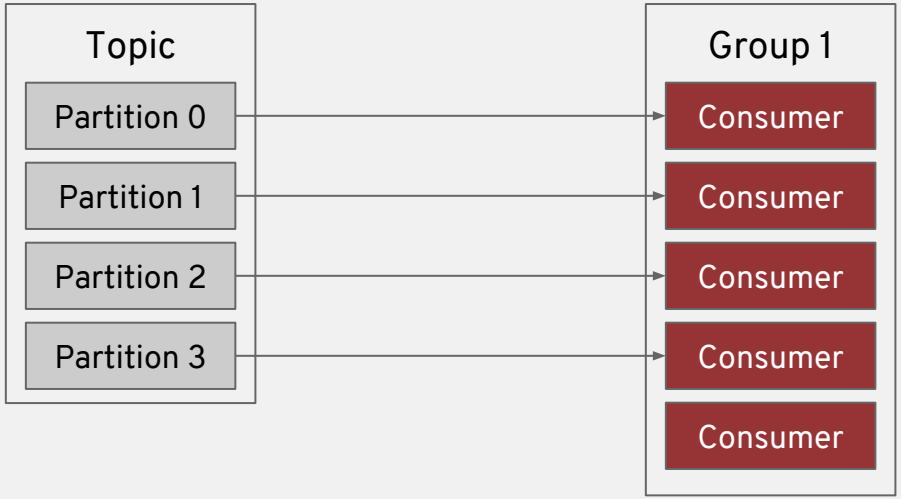
Rebalancing



redhat.

Consumer Groups

Max parallelism & Idle consumer



redhat.

Security

- Encryption between clients and brokers and between brokers
  - Using SSL
- Authentication of clients (and brokers) connecting to brokers
  - Using SSL (mutual authentication)
  - Using SASL (with PLAIN, Kerberos or SCRAM-SHA as mechanisms)
- Authorization of read/writes operation by clients
  - ACLs on resources such as topics
  - Authenticated “principal” for configuring ACLs
  - Pluggable
- It’s possible to mix encryption/no-encryption and authentication/no-authentication

redhat.

“ … and what about the traditional messaging brokers ?”

redhat.

Messaging : “traditional” brokers

- Have to track consumers status, dispatching messages
- Delete messages, no long-time persistence
  - Support for TTL
- No re-reading the messages feature is available
- Consumers are fairly simple
- Open and standard protocols support (AMQP, MQTT, JMS, STOMP, …)
- Selectors
- Best fit for request/reply pattern

redhat.

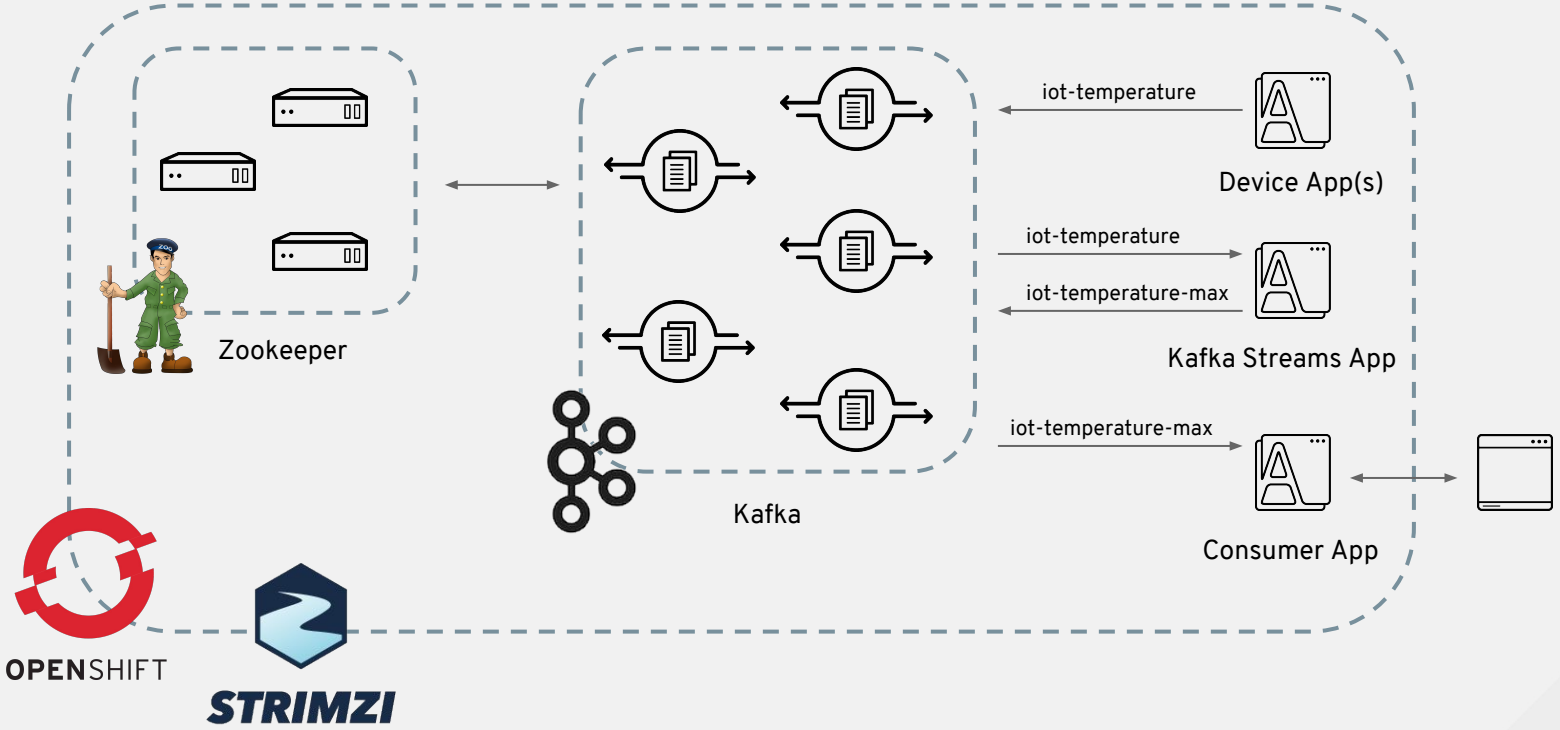
Messaging with “traditional” brokers

Key differences

	ActiveMQ Artemis	Kafka
Model	“Smart broker, dumb clients”	“Dumb broker, smart clients”
Durability	Volatile or durable storage	Durable storage
Storage duration	Temporary storage of messages	Potential long-term storage of messages
Message retention	Retained until consumed	Retained until expired or compacted
Consumer state	Broker managed	Client managed (can be stored in broker)
Selectors	Yes, per consumer	No
Stream replay	No	Yes
High-availability	Replication	Replication
Protocols	AMQP, MQTT, OpenWire, Core, STOMP	Kafka protocol
Delivery guarantees	Best-effort or guaranteed	Best-effort or guaranteed

redhat.

Demo: an IoT use case



redhat.

Resources

- **Apache Kafka** : https://kafka.apache.org/
- **Strimzi** : http://strimzi.io/
- **Kubernetes** : https://kubernetes.io/
- **OpenShift** : https://www.openshift.org/
- **Demo** : https://github.com/strimzi/strimzi-lab/tree/master/iot-demo
- **Paolo's blog** : https://paolopatierno.wordpress.com/

redhat.

Thank you ! Questions ?

@ppatierno