IOT DATA STREAMING WITH APACHE KAFKA

From the data to the wisdom

Paolo Patierno
Principal Software Engineer @ Red Hat
15/12/2018

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
- Dad of two
- VR46's fan (I don't like MM93)
- Try to be a runner ...

IOT USE CASES

- Smart city
- Healthcare
- Connected cars
- Logistics
- Home automation
- Airlines
- Farmers
- You!

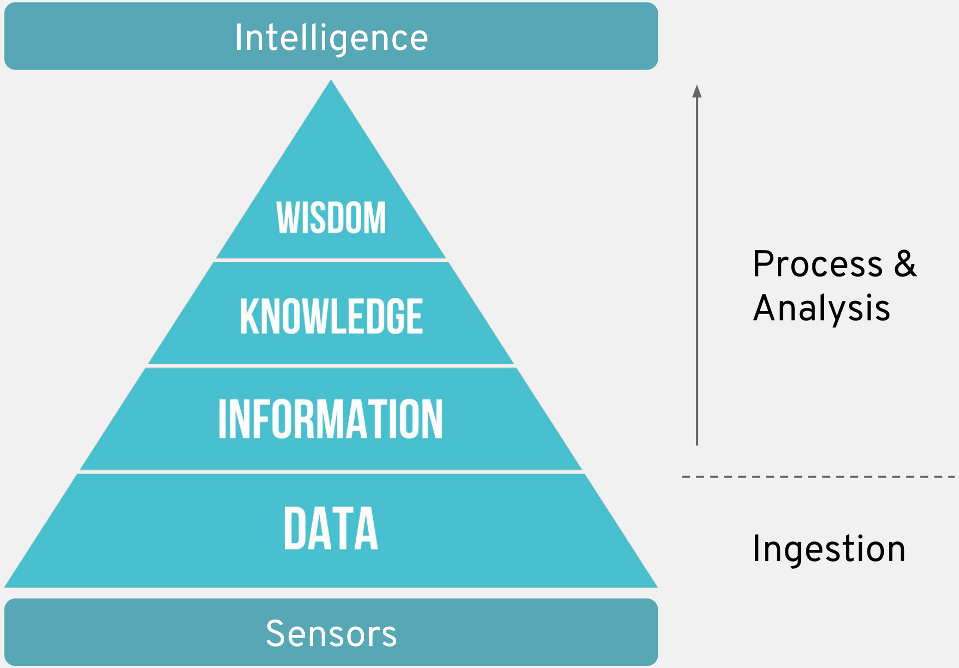
WHAT THEY HAVE IN COMMON?

IOT WORKLOAD

- Huge amount of data to ingest
- Unbounded dataset
- Need for real time analytics
- Need for real time reactions
- Sometimes the need to join :
  - with static data
  - with historical data

IOT WORKLOAD

From raw data to the intelligence



THE NEED FOR A PARADIGM DIFFERENT FROM BATCH PROCESSING

STREAM PROCESSING

- A type of data processing engine that is designed with infinite datasets in mind
- Working with data as they arrive
- Working with an infinite stream of data
    - Moving boundaries with data in “windows”
- Tradeoffs between latency/cost/correctness

STREAM PROCESSING

Versus batch processing

Compared to batch based flows, streaming flows ...

- ... are often more time-sensitive
- ... could be larger in volume
- ... may or may not be of long-term value after processing

So that you can ...

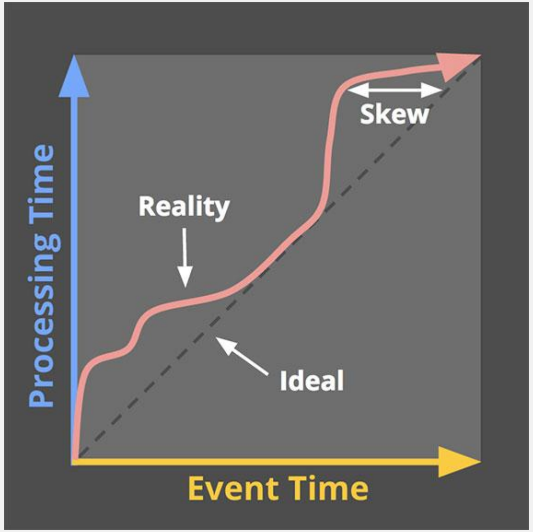
- ... capture and aggregate millions of events per second
- ... instantly take action
- ... respond to changing conditions

STREAM PROCESSING

Because time matters

Time

- Event time, which is the time at which events actually occurred
- Processing time, which is the time at which events are observed in the system



STREAM PROCESSING & IOT

What are the good parts for IoT?

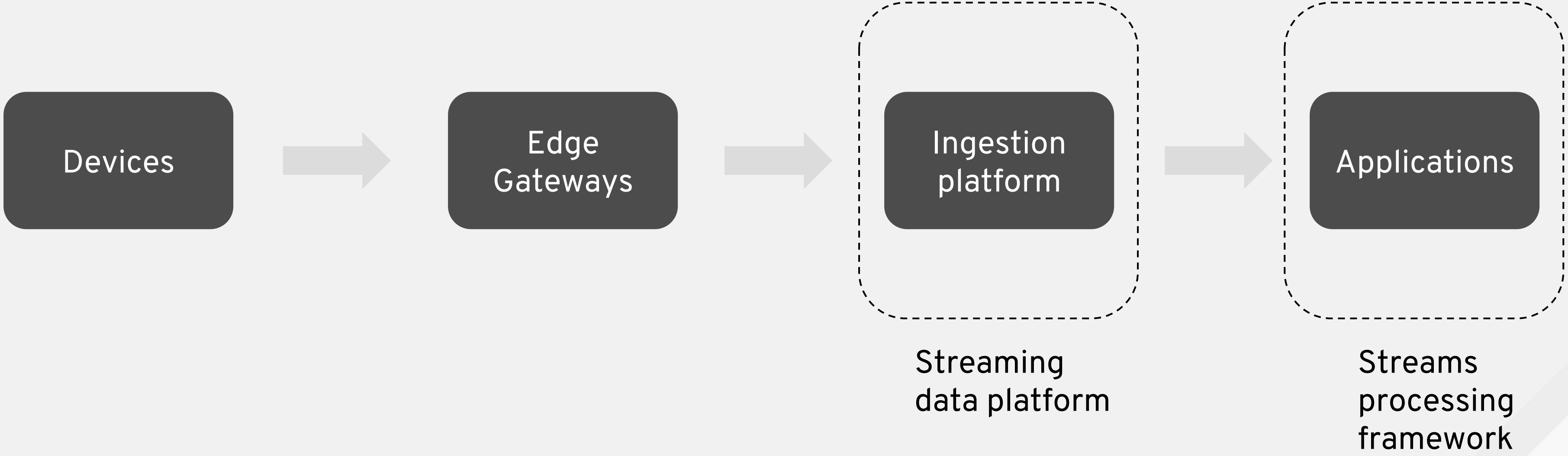
- Ordering
  - Handling devices data in the order they are sent
  - Handling “out of order” data when the device is not able to guarantee order
- Timing
  - Allowing “event time” which makes much sense in IoT scenarios
  - Supporting “windowing” for getting analytics in a specific time window
- State management
  - Be fast on joining real time data with metadata to enrich information content
  - Be fast on data aggregation
  - Be able to join different streams of data, from different devices

STREAM PROCESSING & IOT

What are the good parts for IoT?

- Re-processing
    - Supporting the possibility to re-read the stream of data
- Fault tolerance
    - Allowing to get data from devices continuously even in case of consumers crash
- Scalability/Partitioning
    - Being able to handle bursts of traffic
    - Being able to handle more and more devices data

STREAM PROCESSING & IOT



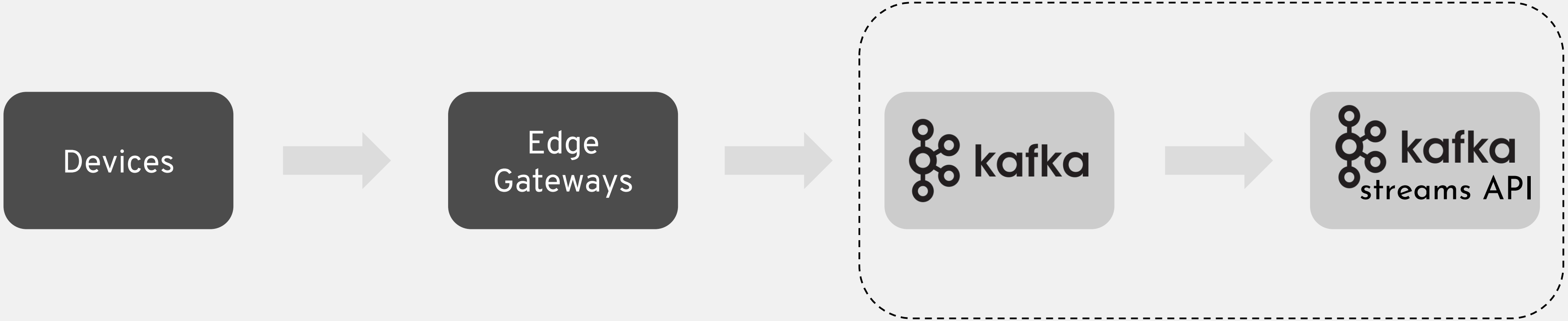
STREAM PROCESSING & IOT

It’s a wild west out there

- Streaming data platform
  - Apache Kafka
- Streams processing framework
  - Apache Spark (streaming)
  - Apache Samza
  - Apache Flink
  - ...

DO WE REALLY NEED MORE THAN ONE PLATFORMS IN PLACE?

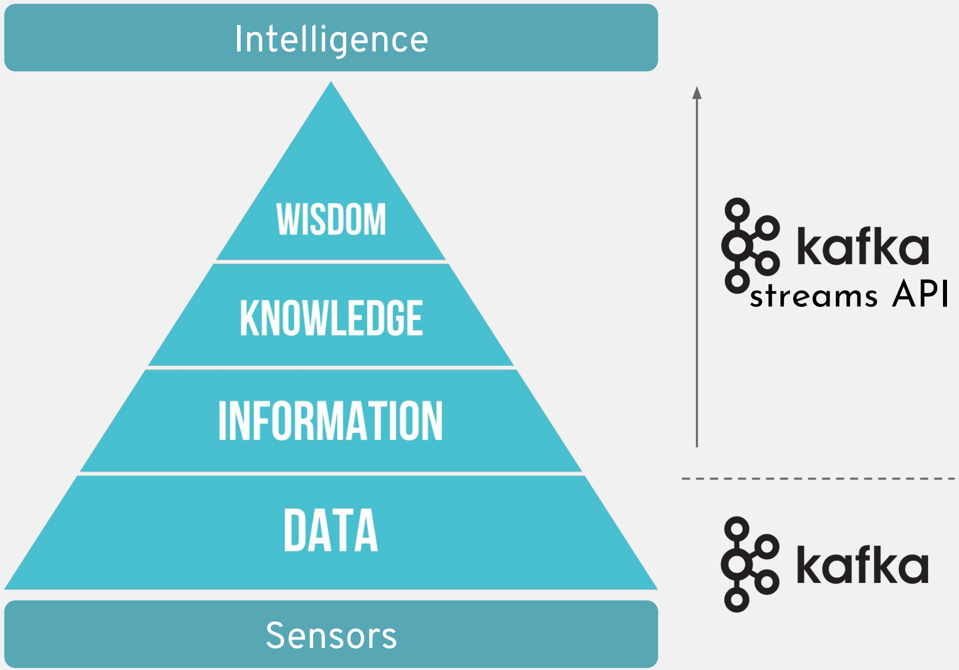
STREAM PROCESSING & IOT



Apache Kafka + Kafka Streams API

IOT WORKLOAD

From raw data to the intelligence



APACHE KAFKA

WHAT IS APACHE KAFKA

A publish/subscribe messaging system?

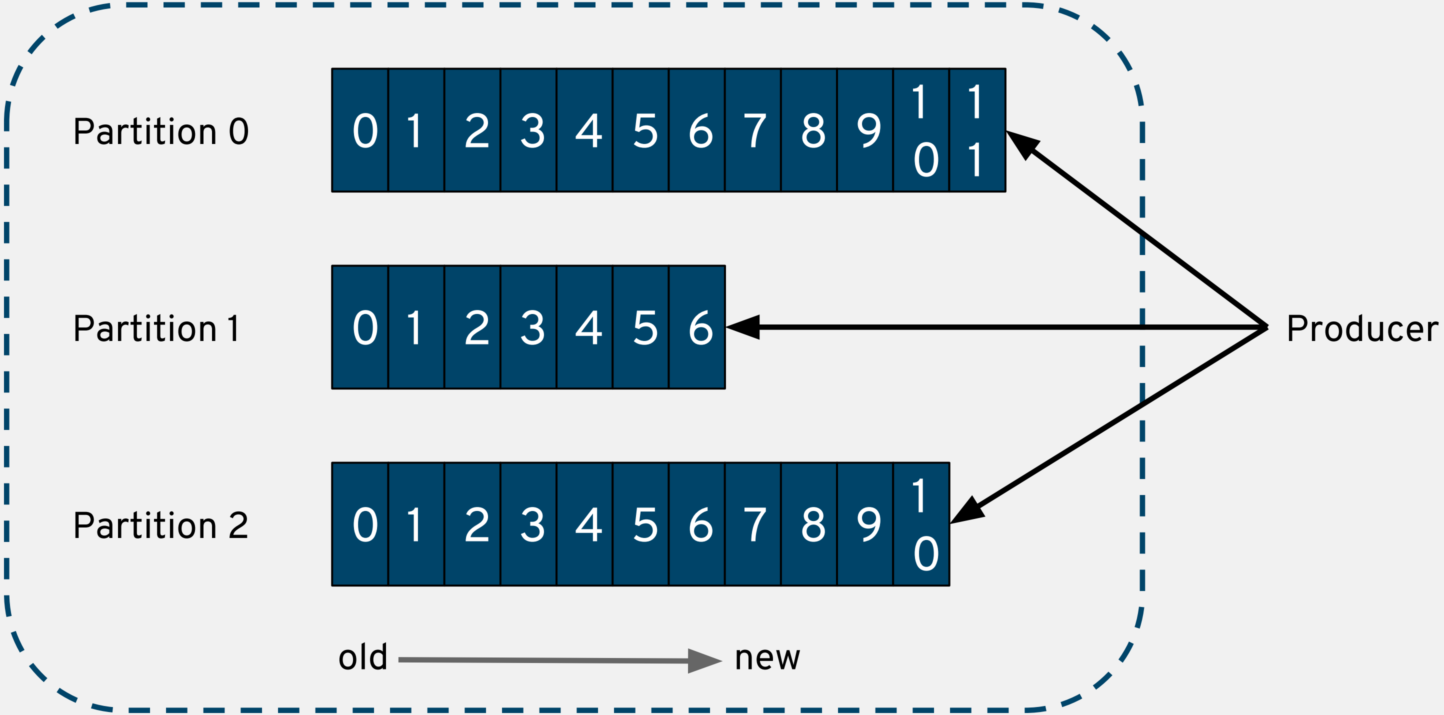
A streaming data platform?

A distributed, horizontally-scalable, fault-tolerant, commit log?

APACHE KAFKA CONCEPTS

- Messages are sent to and received from a topic
  - Topics are split into one or more partitions (aka shards)
  - All actual work is done on partition level, topic is just a virtual object
- Each message is written only into a one selected partition
  - Partitioning is usually done based on the message key
  - Message ordering within the partition is fixed
- Retention
  - Based on size / message age
  - Compacted based on message key
- Replication
  - Each partition can exist in one or more copies to achieve high availability

TOPIC & PARTITIONS



Partition 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11

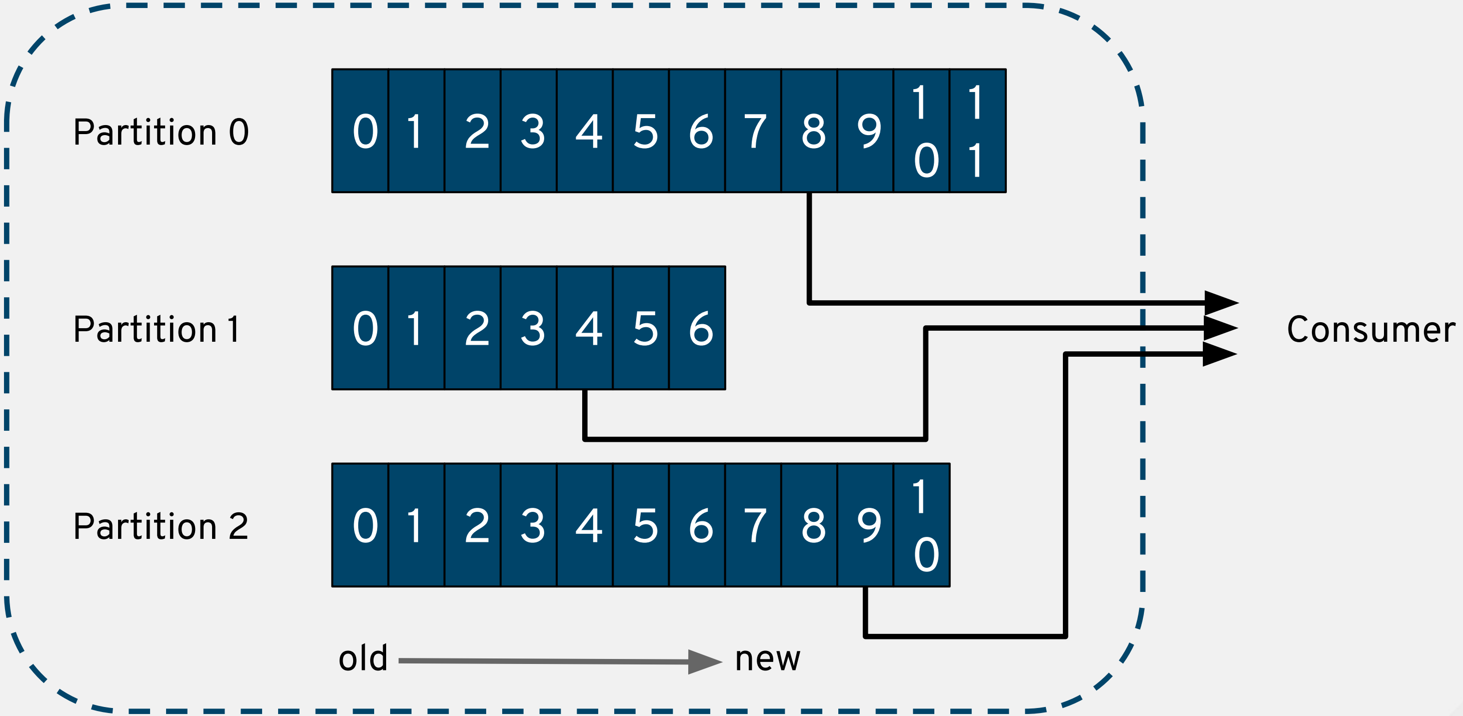
Partition 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6

Partition 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

Producer

old → new

TOPICS & PARTITIONS



Partition 0

0 1 2 3 4 5 6 7 8 9 10 11

Partition 1

0 1 2 3 4 5 6

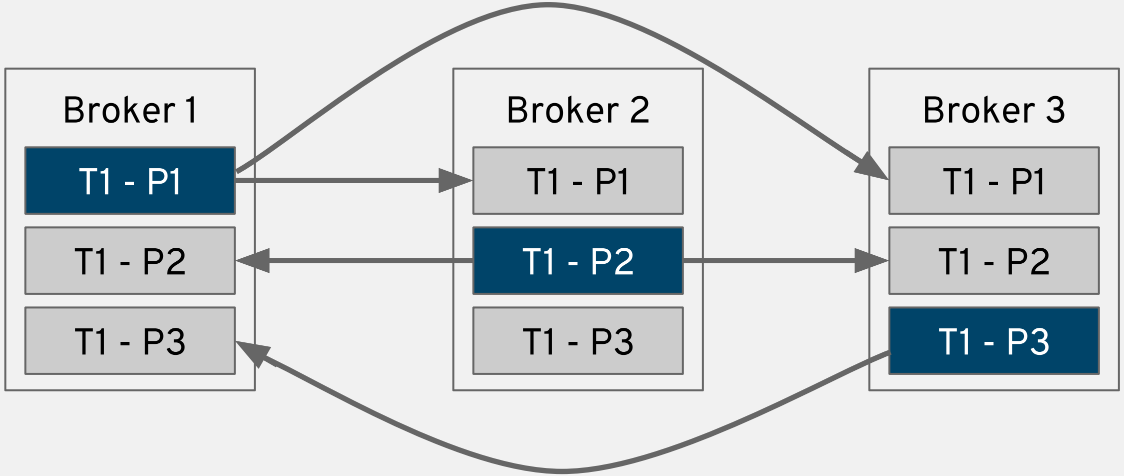
Partition 2

0 1 2 3 4 5 6 7 8 9 10

old → new

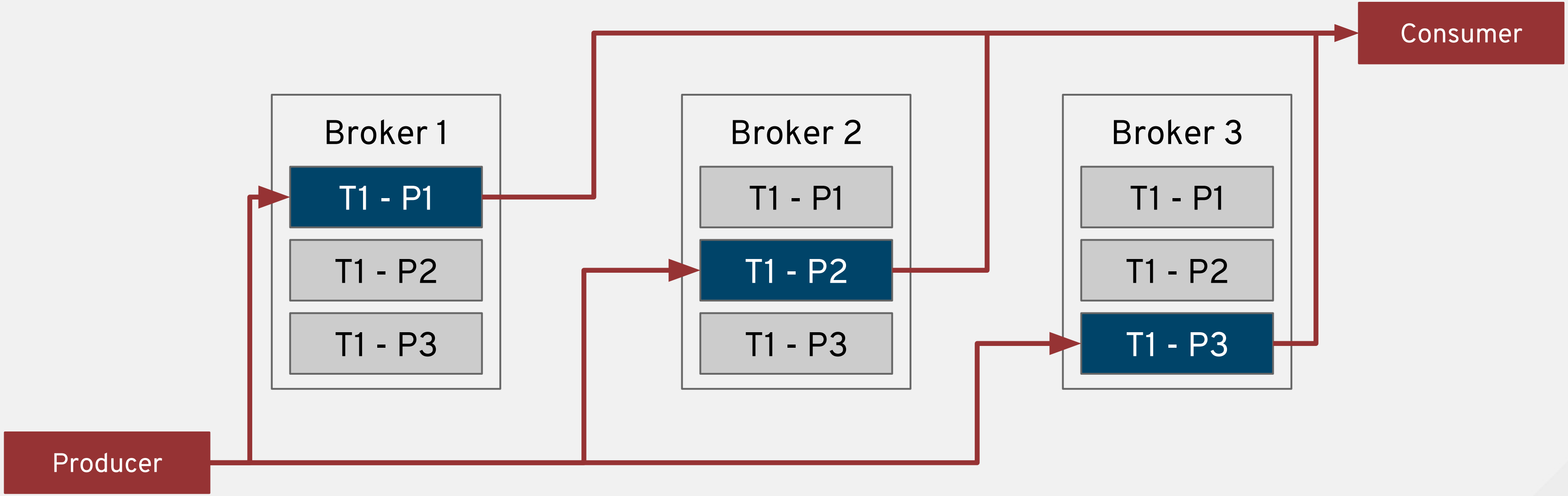
Consumer

HIGH AVAILABILITY



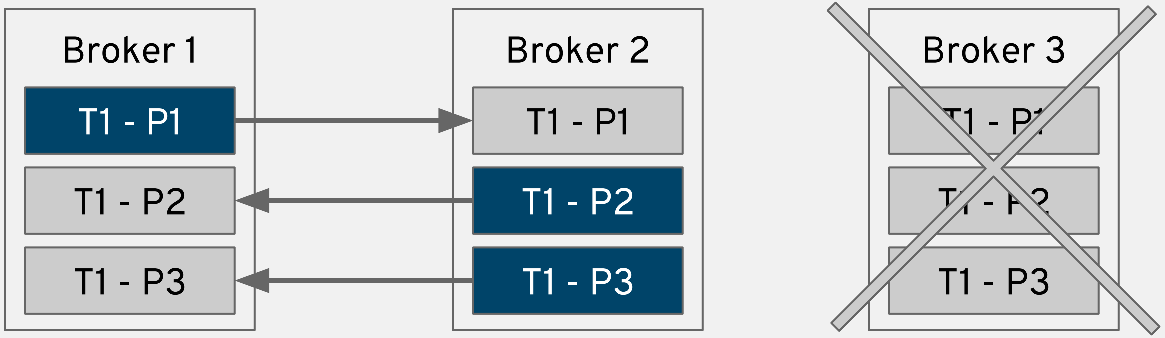
Leaders and followers spread across the cluster.

HIGH AVAILABILITY



Clients always connect only to the leaders.

HIGH AVAILABILITY

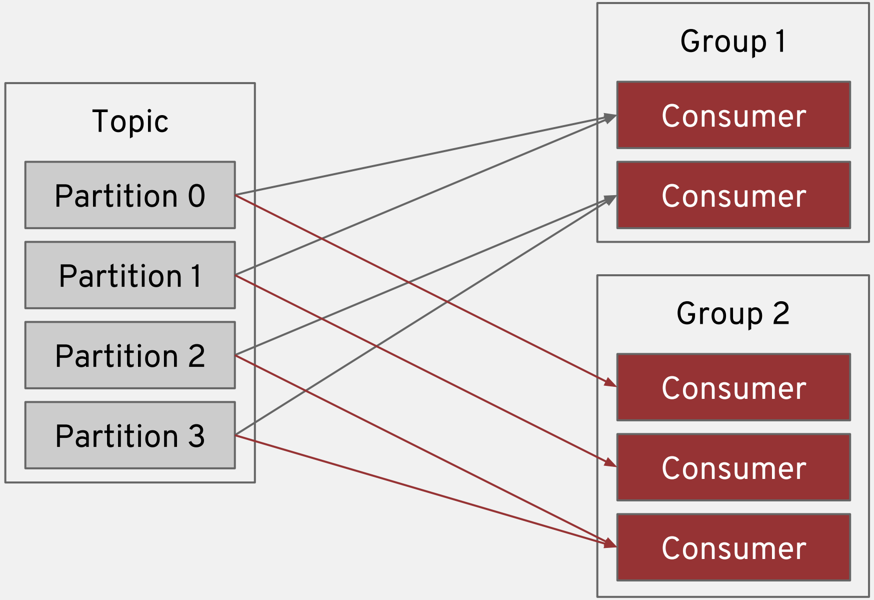


If a broker with leader partition goes down, a new leader partition is elected on different node

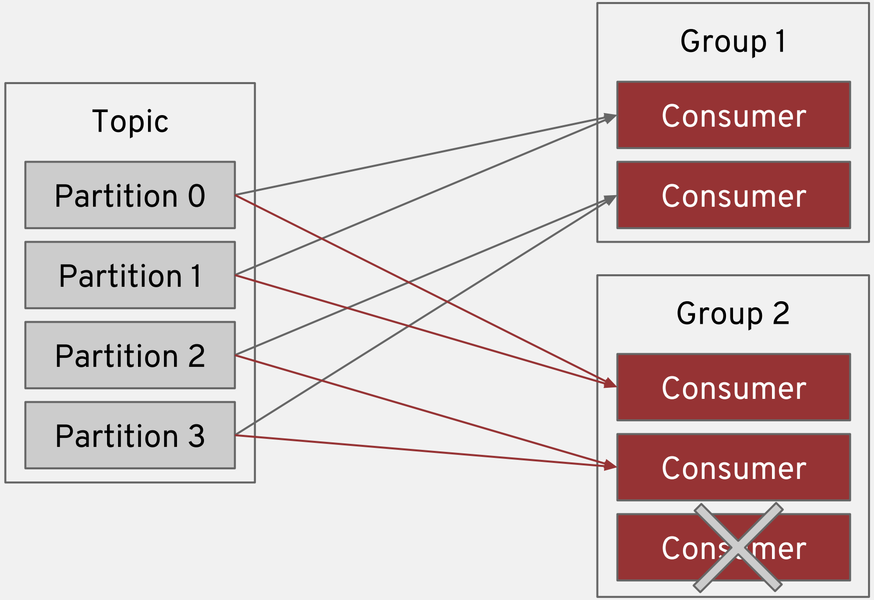
CONSUMER GROUPS

- 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

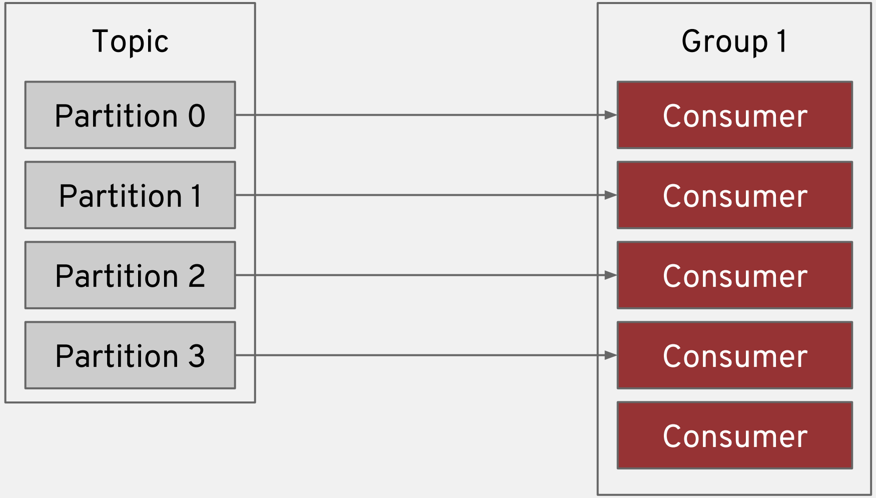
CONSUMER GROUPS



CONSUMER GROUPS



CONSUMER GROUPS



APACHE KAFKA FOR IOT

Pros & Cons

Pros

- “Big” buffer for handling back pressure
- High throughput
- High availability
- Long term storage
- Reprocessing devices data

Cons

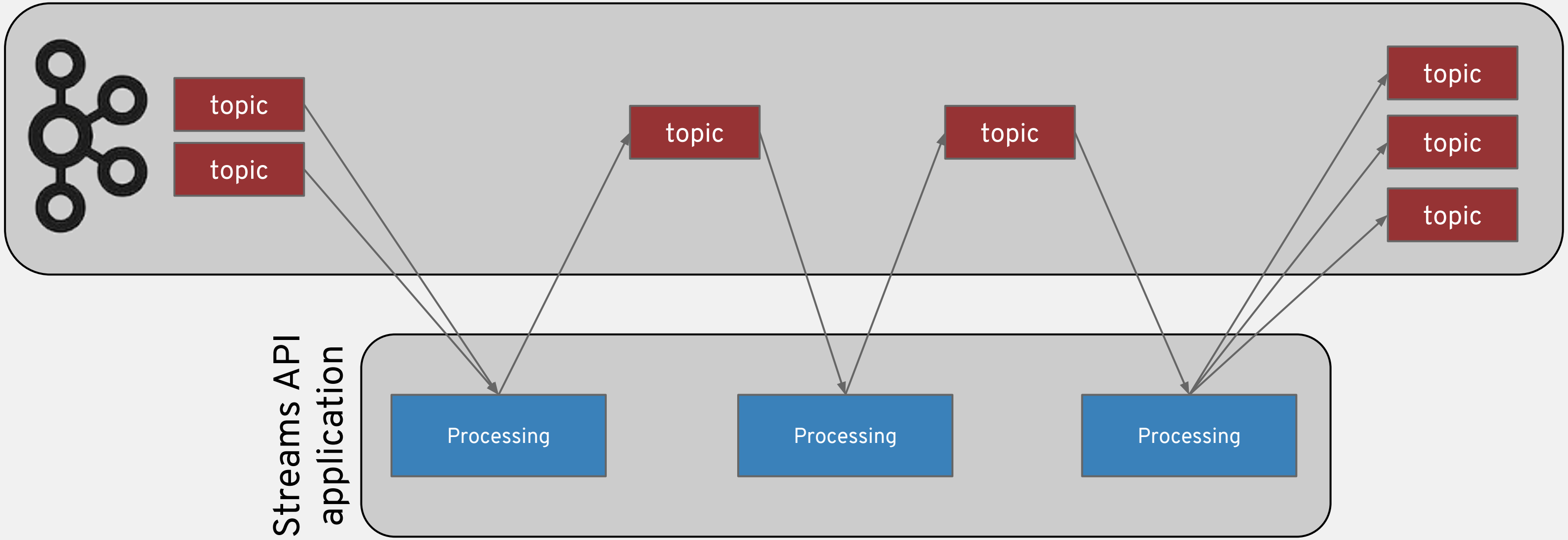
- Cannot handle a huge number of device connections
- No standard IoT devices oriented protocol
- Network needs to be stable

KAFKA STREAMS

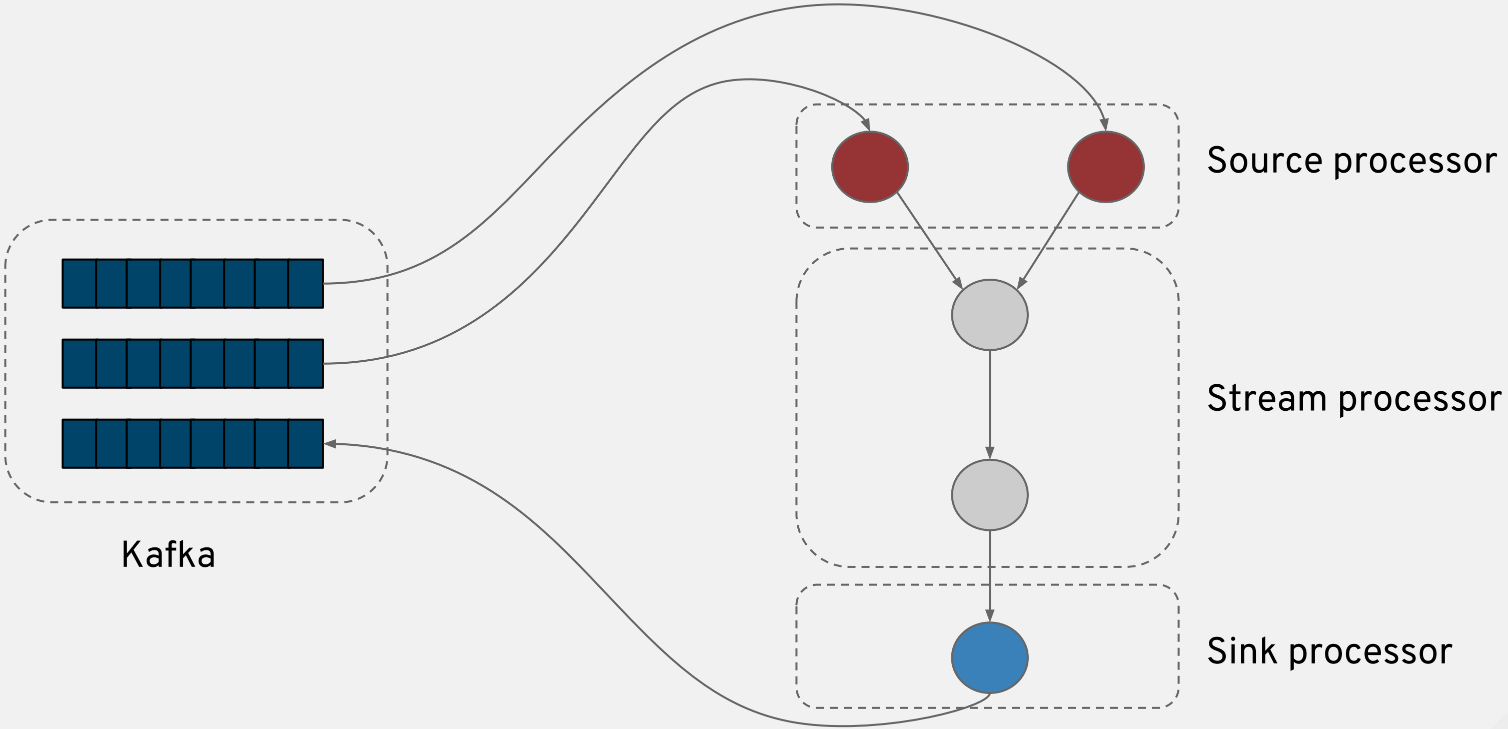
KAFKA STREAMS API

- 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)

KAFKA STREAMS API



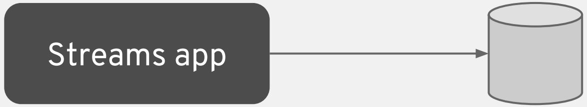
PROCESSORS TOPOLOGY



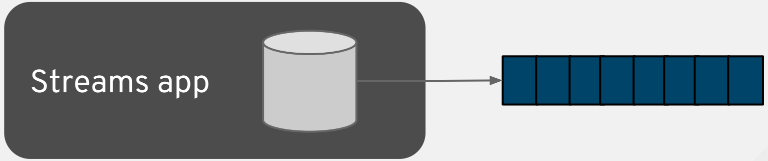
STATE MANAGEMENT

Stateless vs Stateful processing

- Stateless (filter, map) vs Stateful (join, aggregate)
- State handled locally
  - Faster
  - Better isolation
  - Flexible
- It’s based on RocksDB and backed by a Kafka topic as well
  - For rebuilding state on a different Streams node



Remote state



Local state

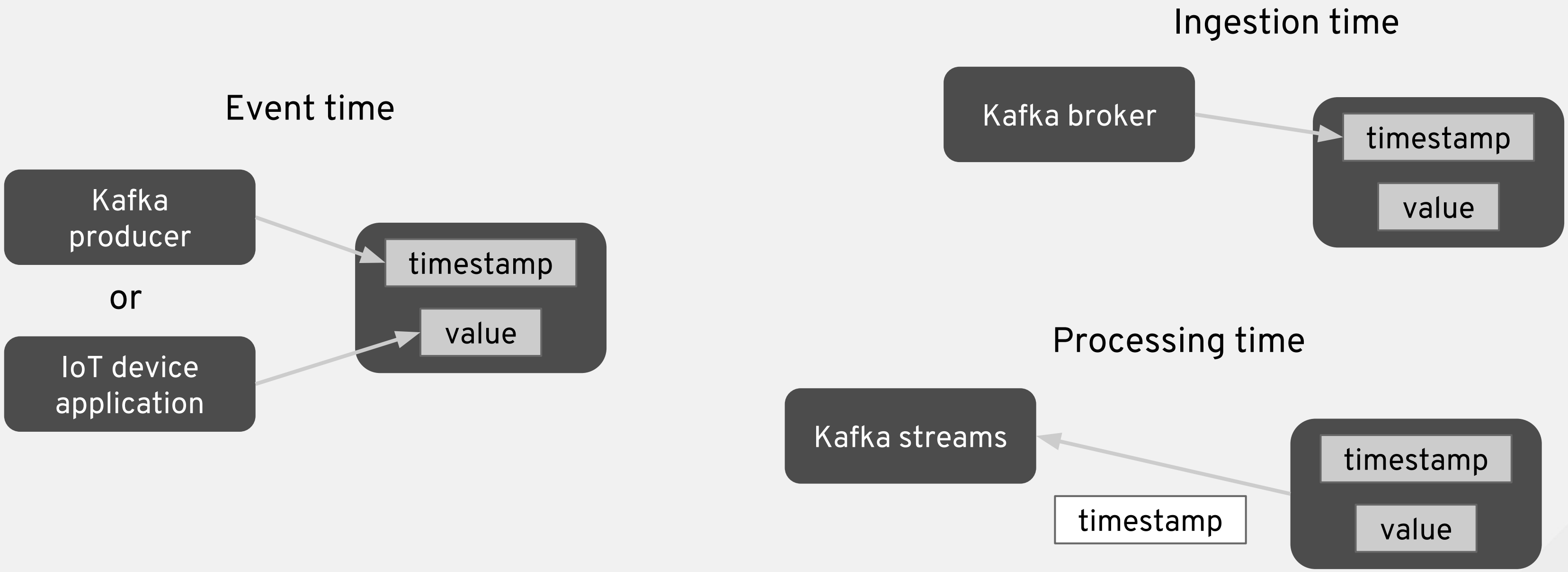
EVENTS TIMESTAMP

Always because time matters

Time

- Event time, which is the time at which events actually occurred
  - As part of the message payload
  - Timestamp on the message (set by the producer)
- Ingestion time, which is the time when the data enters the processing pipeline
  - The broker sets the timestamp on the message
- Processing time, which is the time at which events are observed in the system
  - It’s also called “wall-clock” time

EVENTS TIMESTAMP



Event time

Kafka producer

or

IoT device application

timestamp

value

Ingestion time

Kafka broker

timestamp

value

Processing time

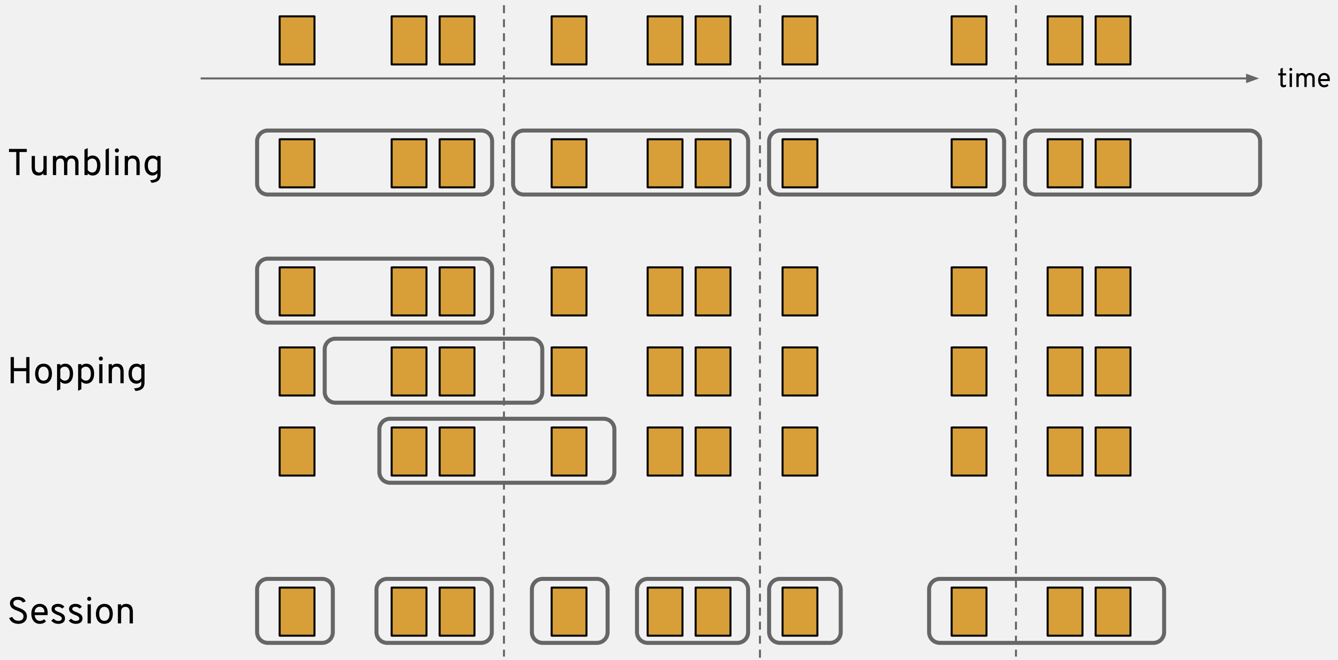
Kafka streams

timestamp

timestamp

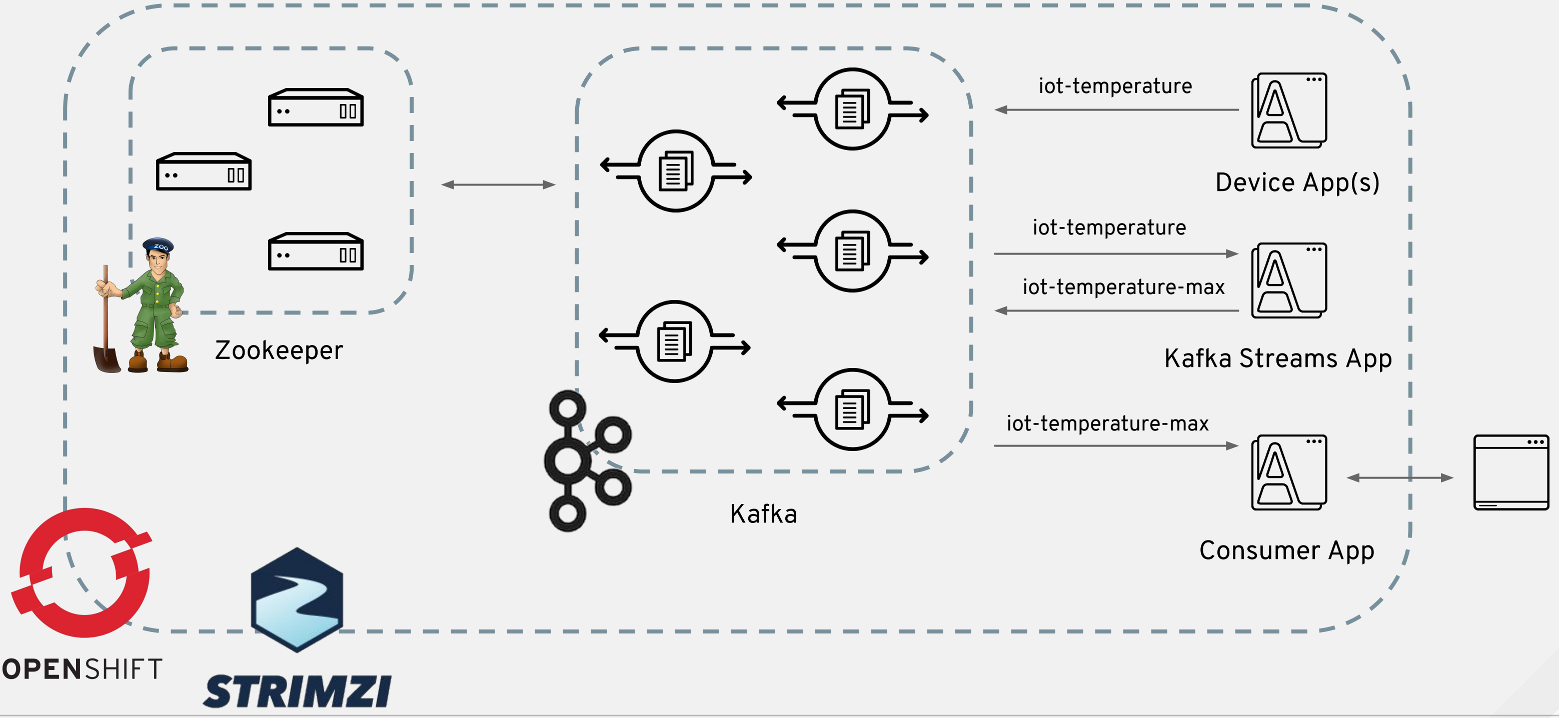
value

WINDOWING



DEMO

DEMO ARCHITECTURE



redhat.

THANK YOU

- plus.google.com/+RedHat
- linkedin.com/company/red-hat
- youtube.com/user/RedHatVideos
- facebook.com/redhatinc
- twitter.com/RedHat