Fun With Kafka

Overview and Architecture

Stephen Nimmo
Senior Specialist Solution Architect
Energy Pod - Houston

Red Hat

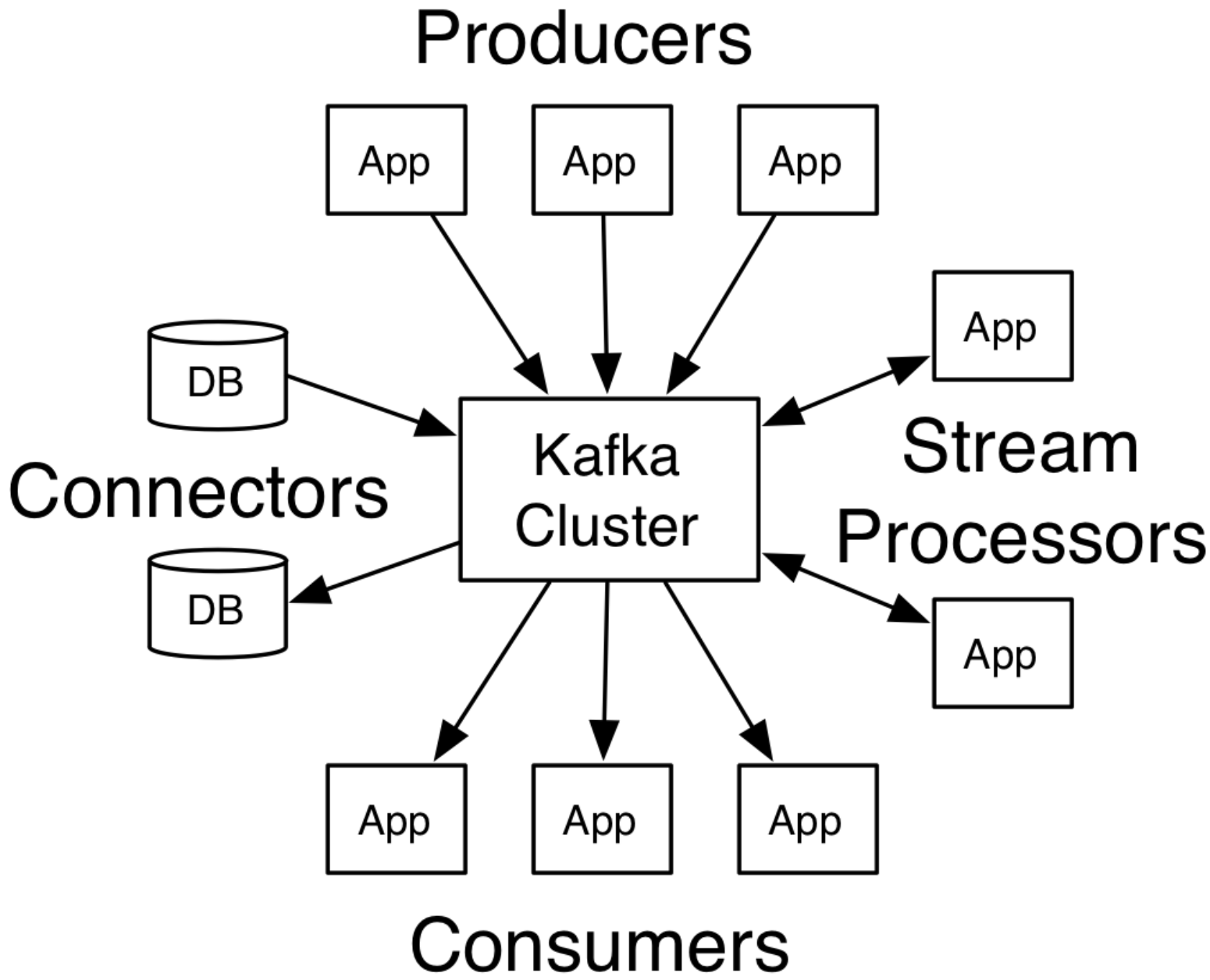
What is Kafka ?

- 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

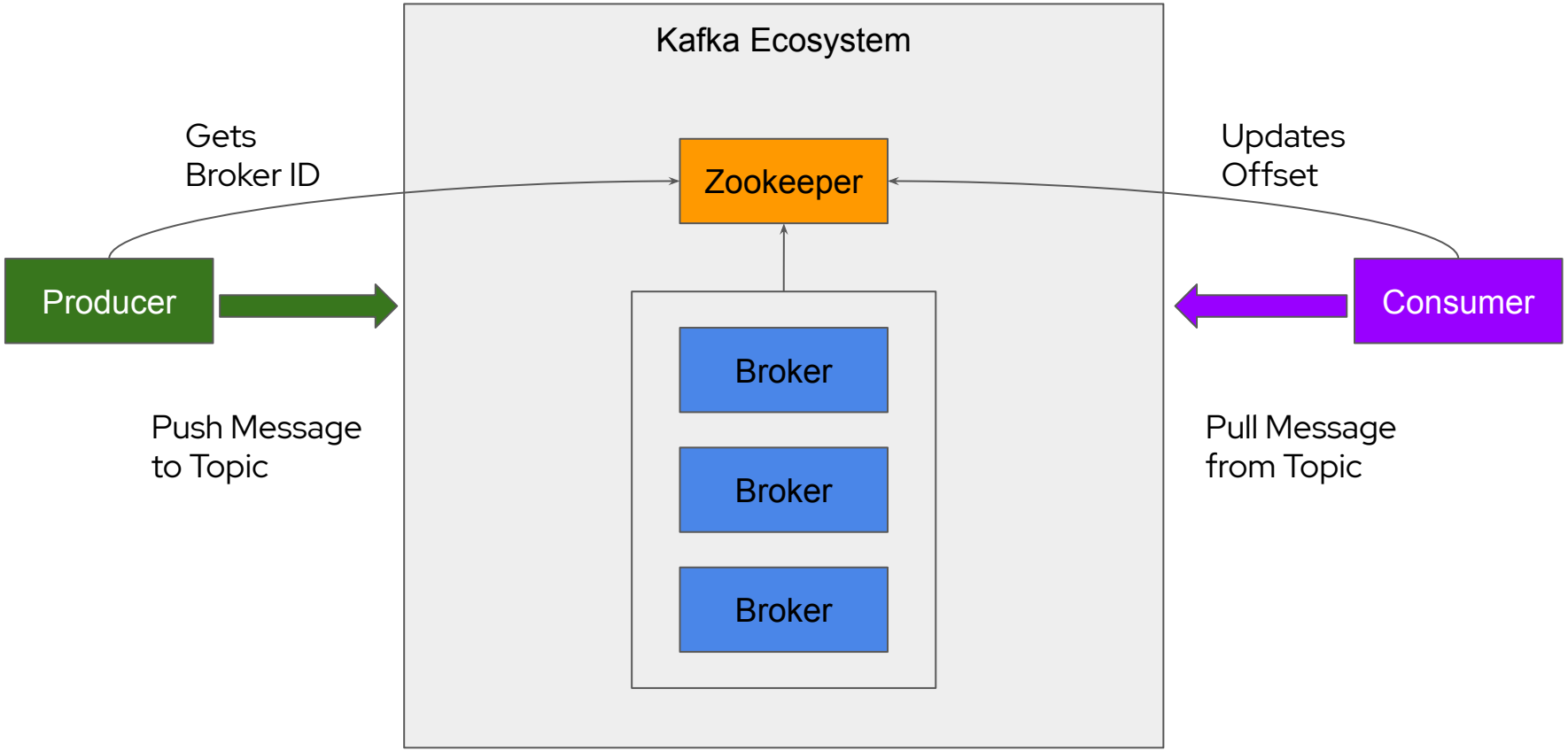
What is Kafka ?

- “ … a publish/subscribe messaging system …”
- “ … a streaming data platform …”
- “ … a distributed, horizontally-scalable, fault-tolerant, commit log …”

Concepts


Producers
App
App
App
DB
DB
Connectors
Kafka
Cluster
App
Stream
Processors
App
App
App
App
Consumers

Architecture

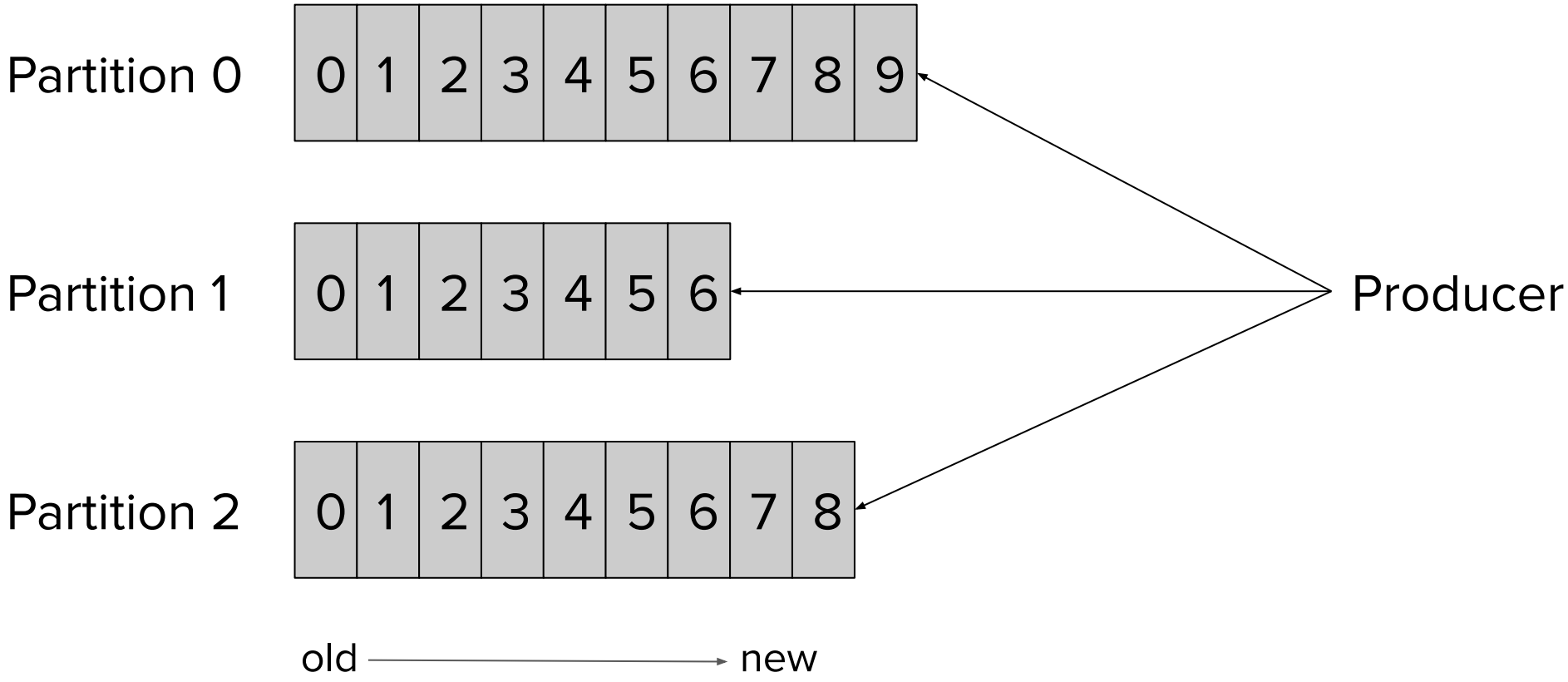


Topic & Partitions

- Messages / records are sent to / received from topic
  - Topics are split into one or more partitions
  - Partition = Shard
  - 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
- Clean-up policies
  - Based on size / message age
  - Compacted based on message key

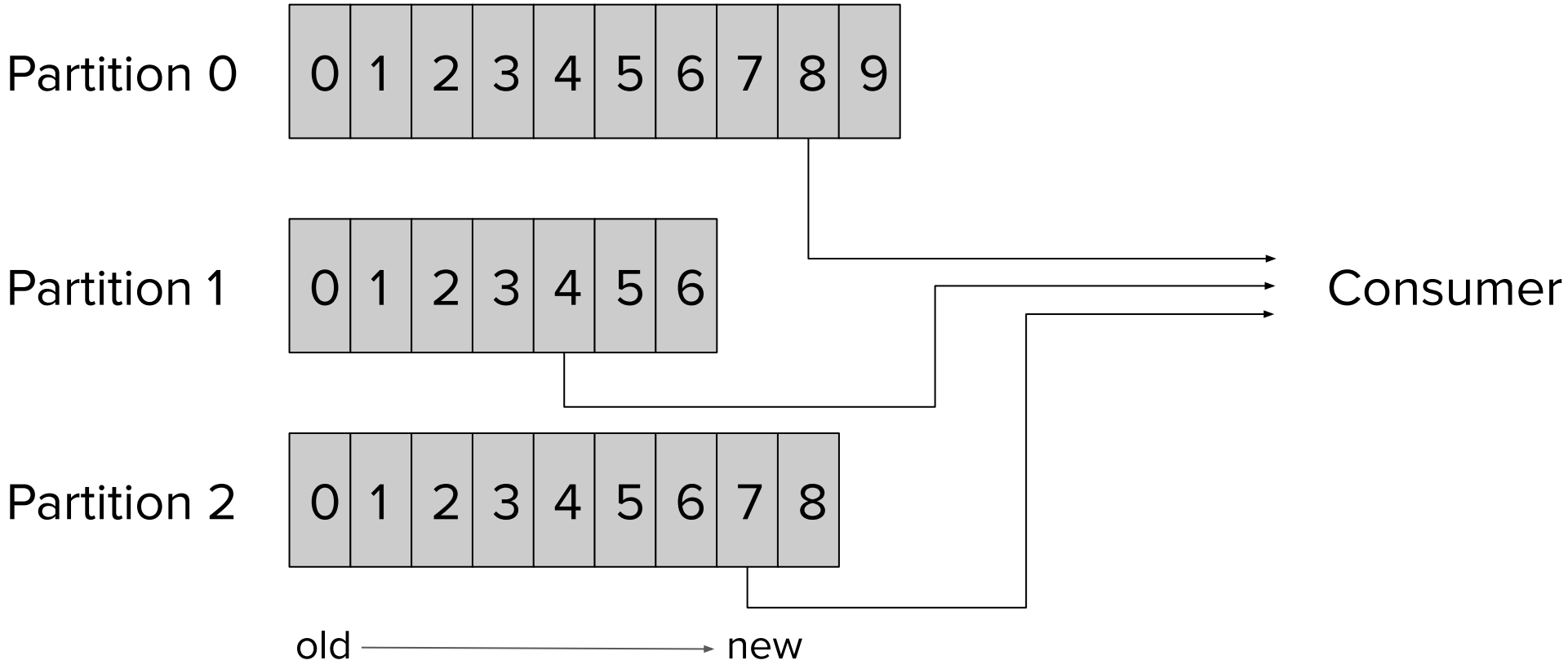
Topic & Partitions

Producing messages



Topic & Partitions

Consuming messages



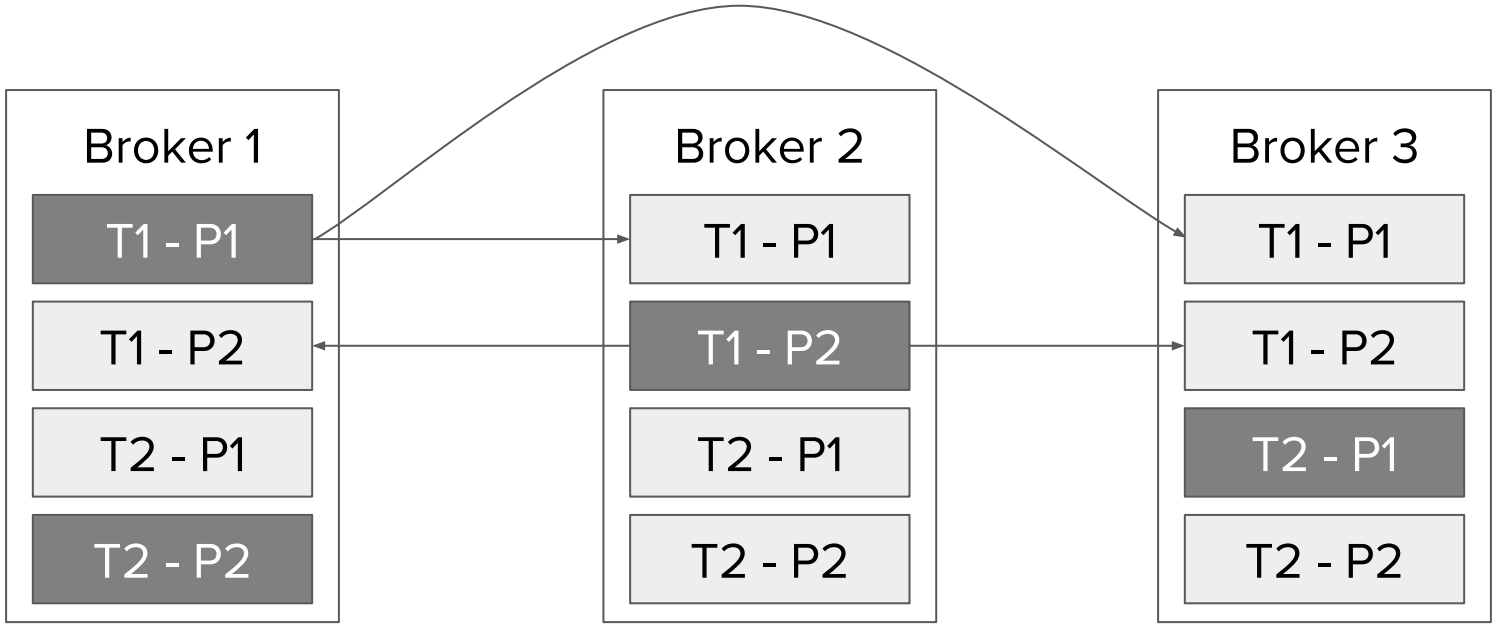
Replication

Leaders & Followers

- They are “backup” for a partition
  - Provide redundancy
- It’s the way Kafka guarantees availability and durability in case of node failures
- 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

Partitions Distribution

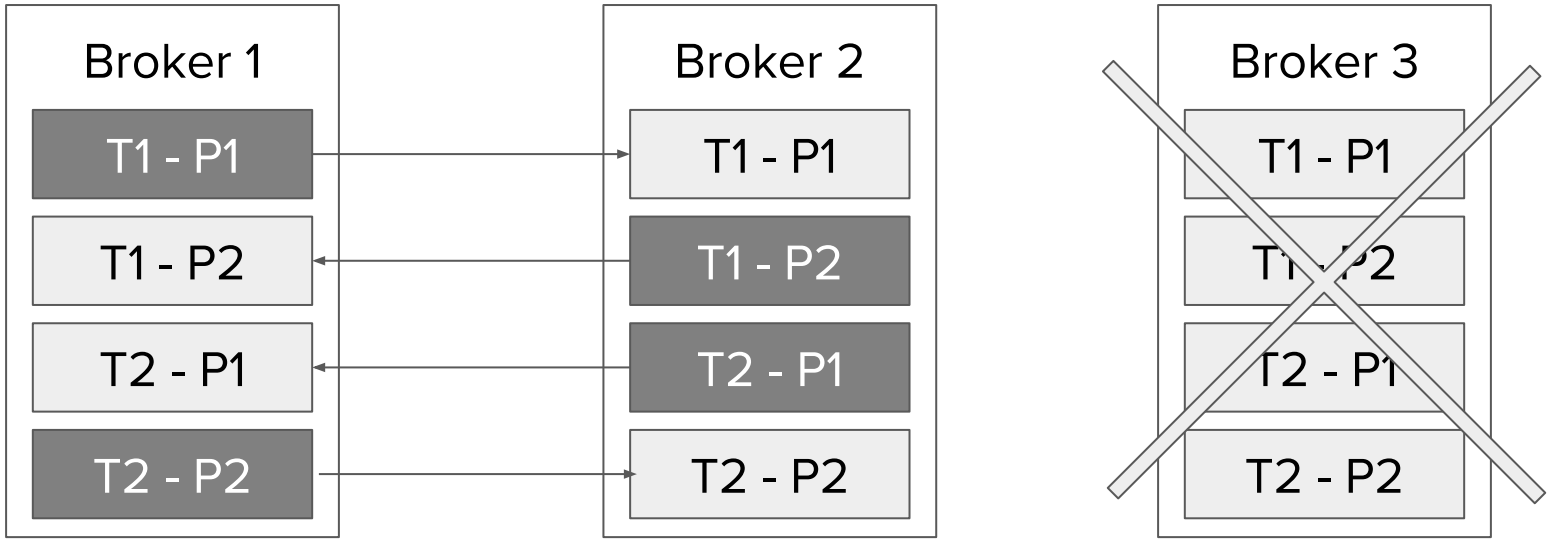
Leaders & Followers



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

Partitions Distribution

Leader election



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

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

Producers

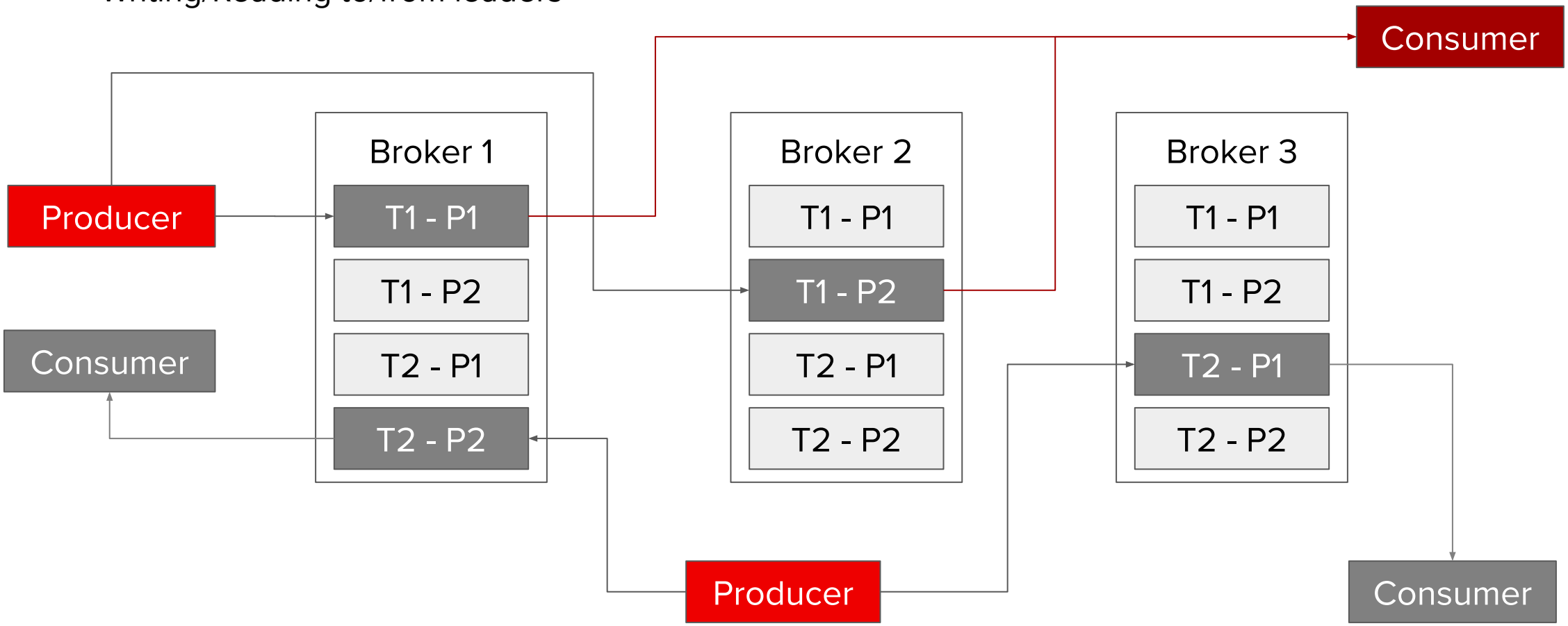
- Destination partition computed on client
  - Round robin
  - Specified by hashing the “key” in the message
  - Custom partitioning
- Writes messages to “leader” for a partition
- Acknowledge :
  - No ack
  - Ack on message written to “leader”
  - Ack on message also replicated to “in-sync” replicas

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)

Producers & Consumers

Writing/Reading to/from leaders



Consumer: partitions assignment

Available approaches

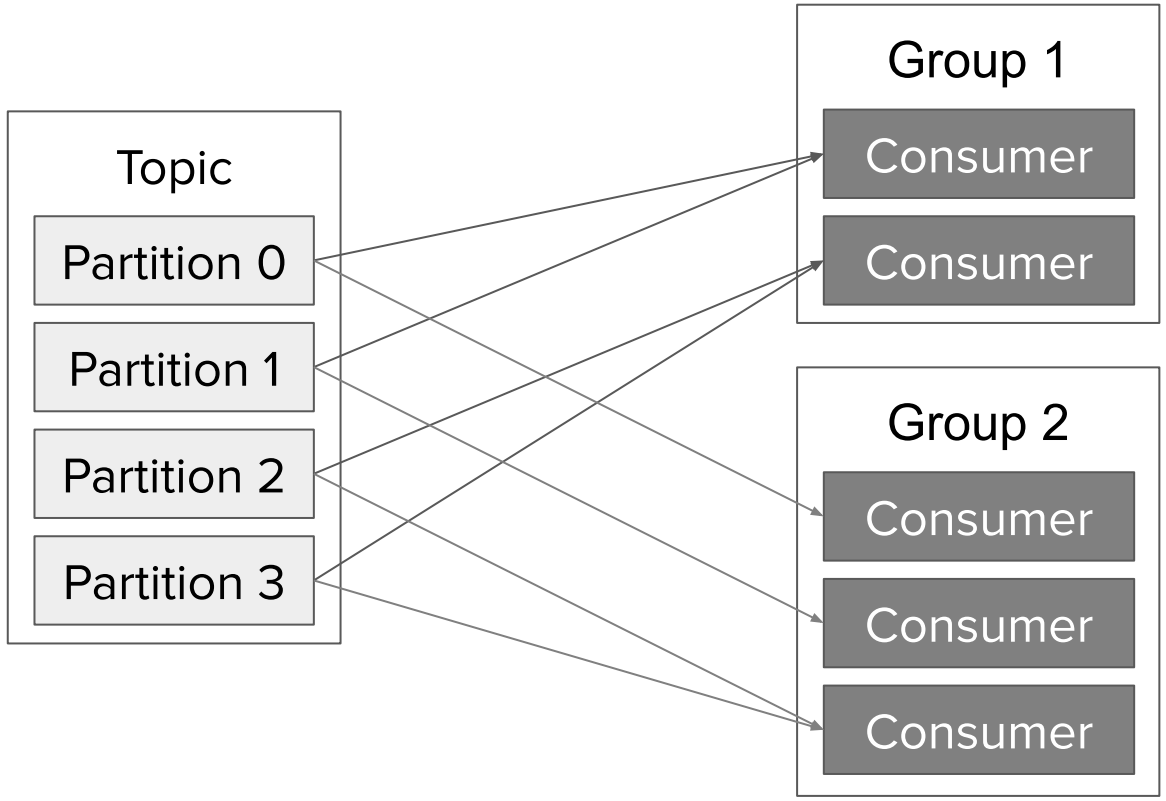
- The consumer asks for a specific partition (assign)
  - An application using one or more consumers has to handle such assignment on its own, the scaling as well
- The consumer is part of a “consumer group”
  - Consumer groups are an easier way to scale up consumption
  - One of the consumers, as “group 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)

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

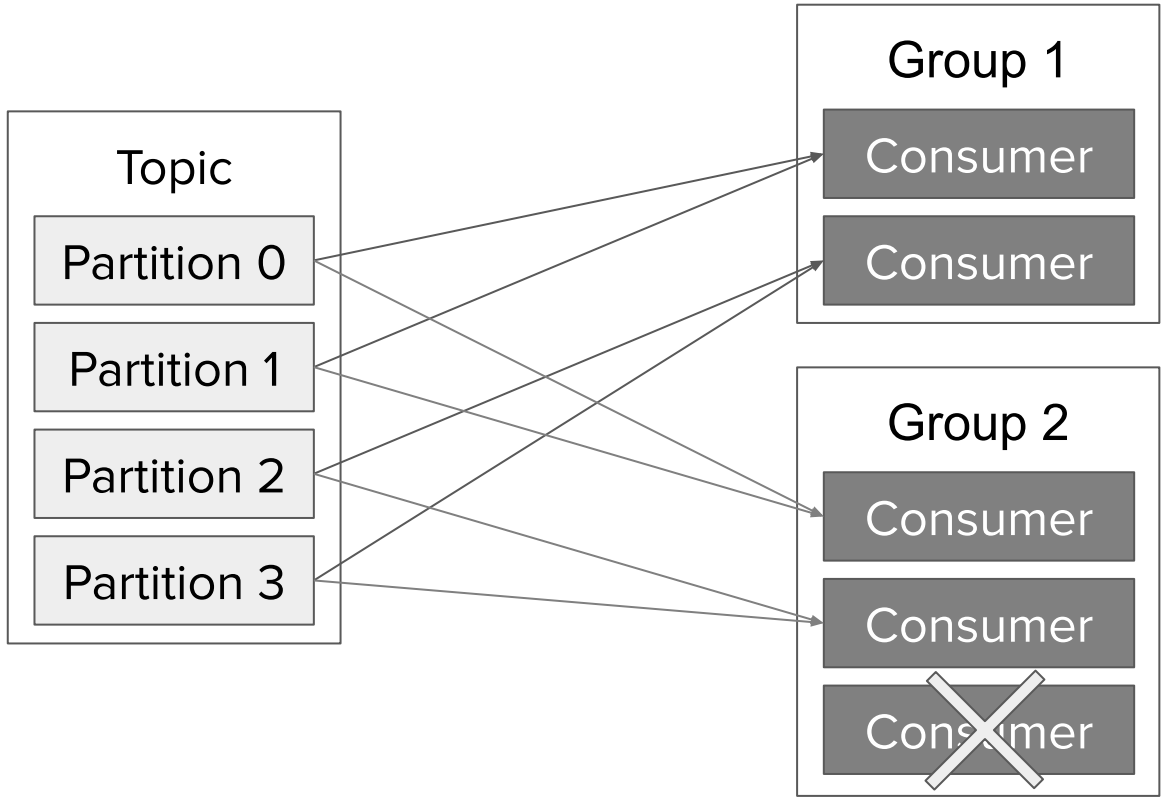
Consumer Groups

Partitions assignment



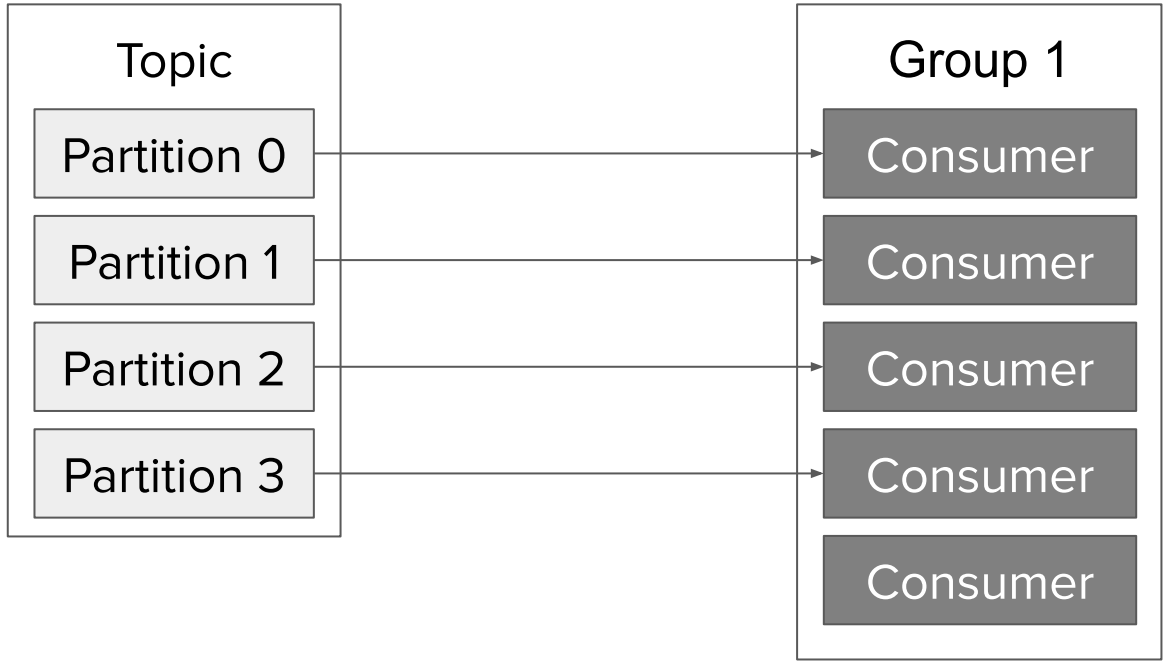
Consumer Groups

Rebalancing



Consumer Groups

Max parallelism & Idle consumer



Kafka + Kubernetes

AMQ STREAMS

When Kafka meets Kubernetes….

OPENSHIFT

AMQ STREAMS

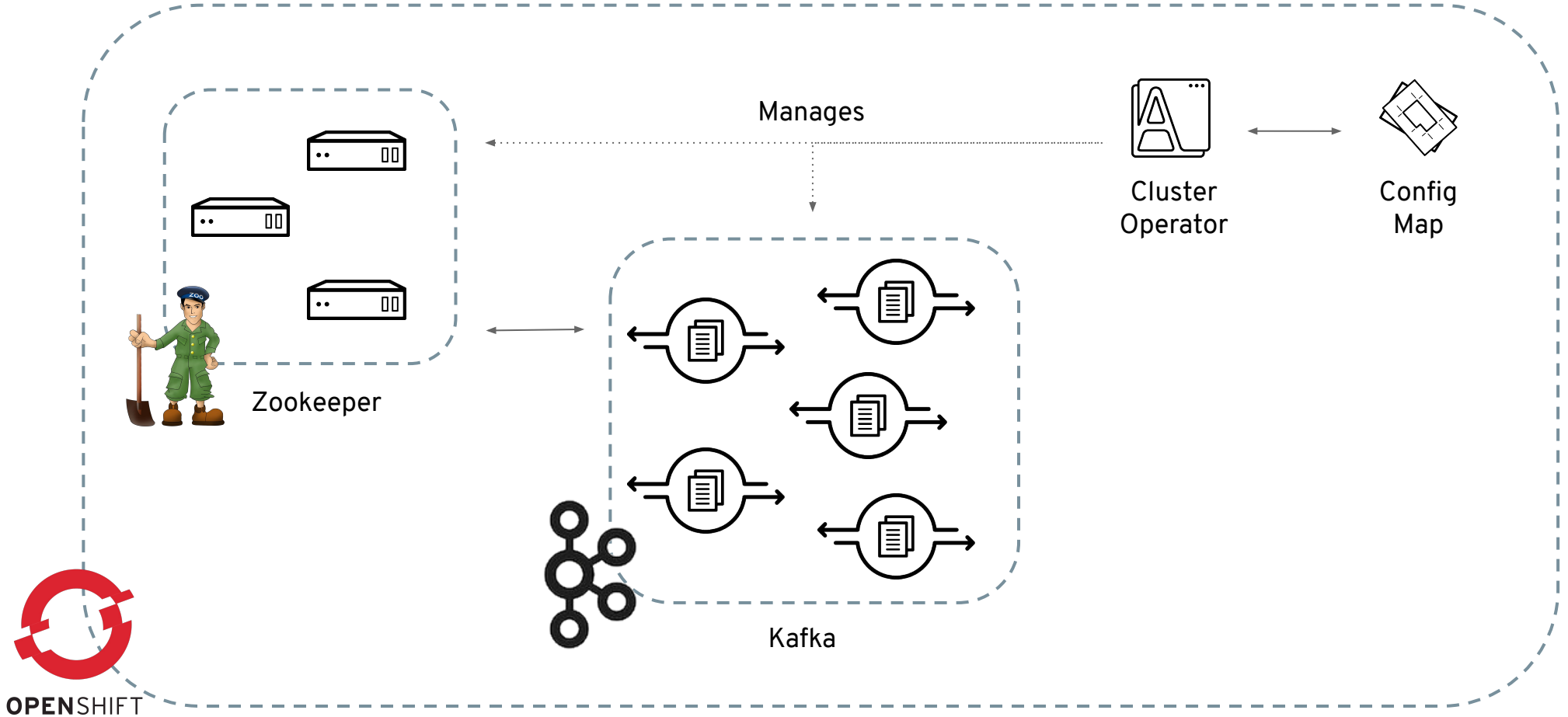
- Easy scalability
  - Running Kafka on bare metal has a high bar
    (ops competency, physical servers, scaling up/down, etc.)
- Automation
  - Configuration as code and automated ops via Operators
  - Tedious ops actions like rolling updates and software upgrades are greatly simplified
- High availability
  - Restoration of Kafka nodes by rescheduling pods in the event of failure
- Messaging use cases are often latency sensitive
  - Can provision cluster/topics as the same time as the application

OPENSHIFT

STRIMZI

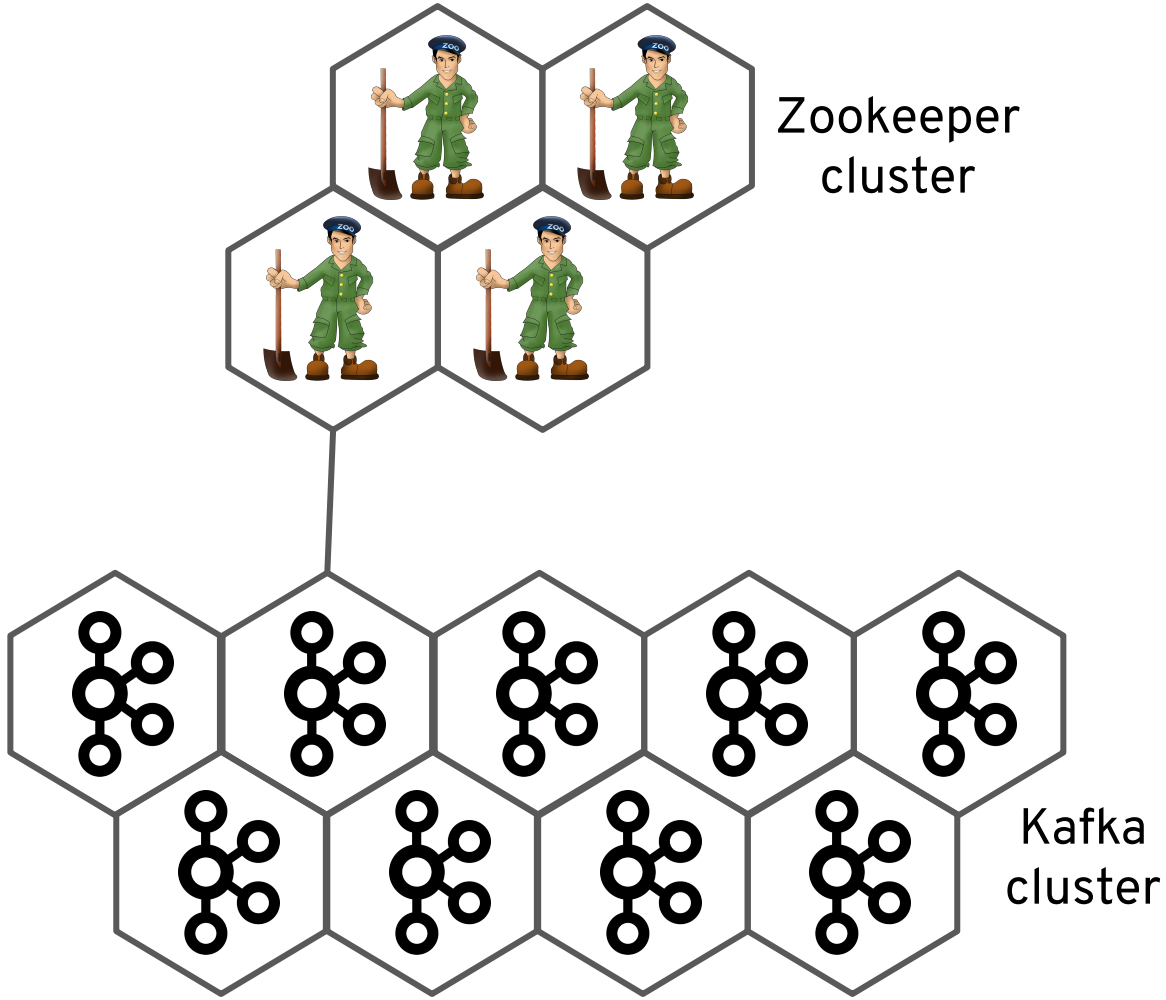
Cluster Operator

Creating and managing Apache Kafka clusters

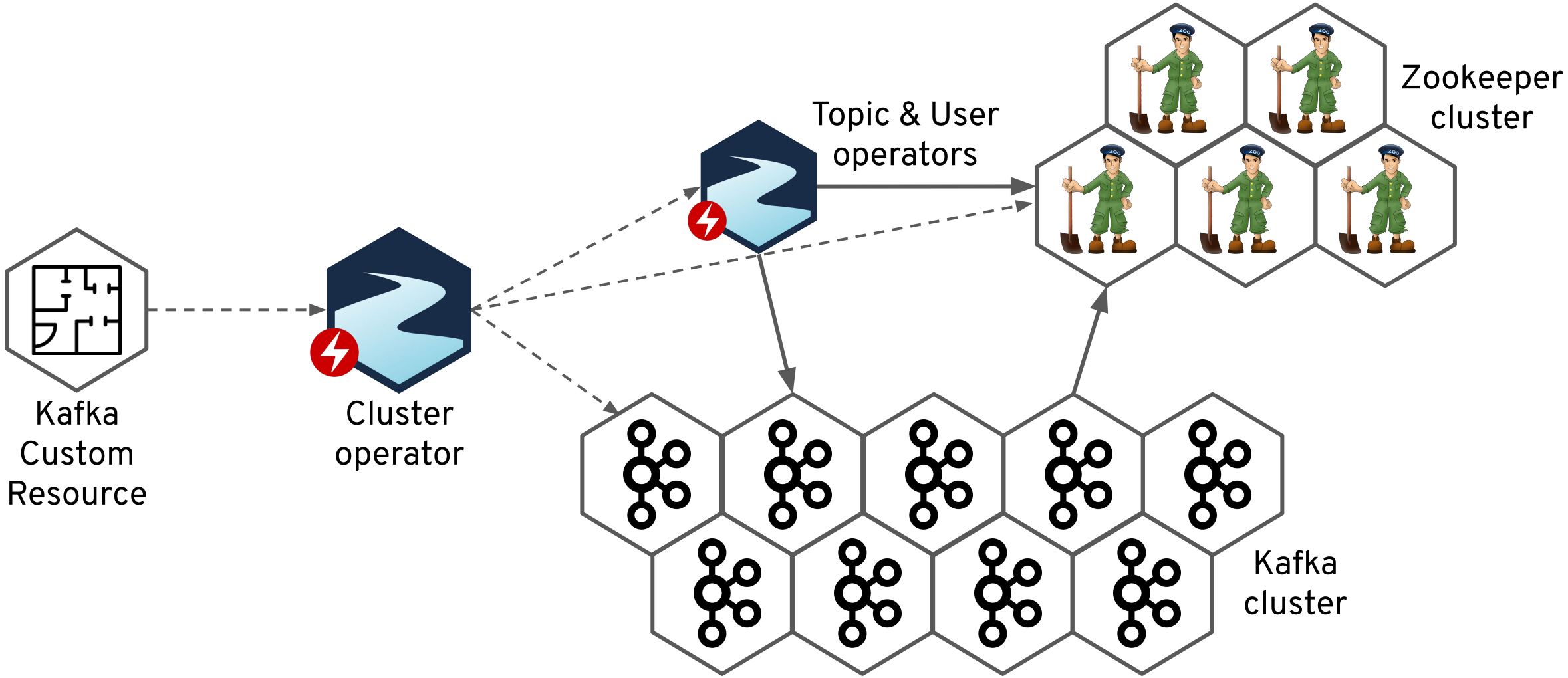


DEPLOYING A CLUSTER

TRADITIONAL APPROACH



DEPLOYING A CLUSTER



Demo

Thanks!


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat