Adventures with Kafka

Marko Štrukelj & Aleš Justin
OpenBlend

JavaCro'19

Agenda

- Kafka intro
- What we learned
  - Kafka API
  - Kafka on GCP / Kubernetes

Kafka?

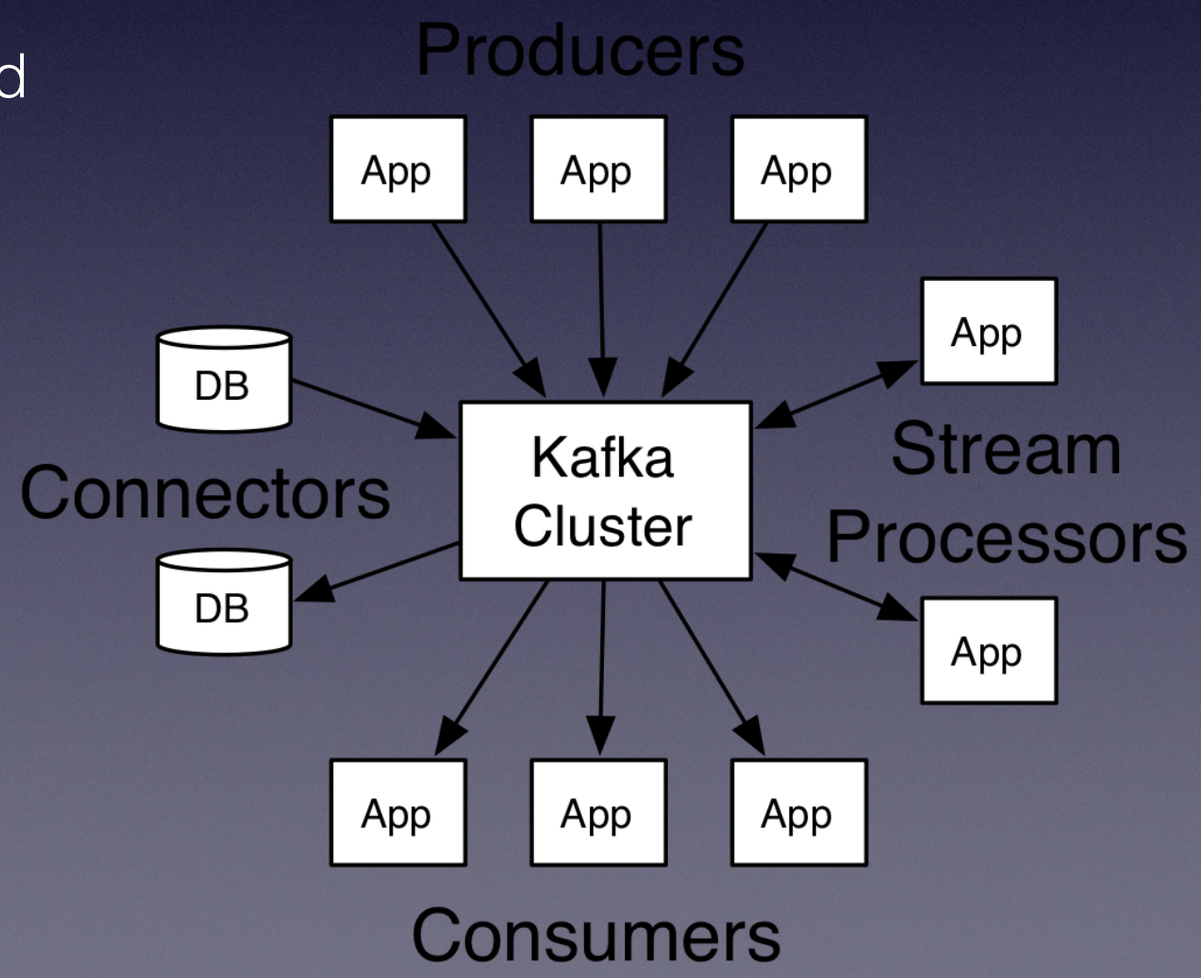
Kafka® is used **for** building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.

Kafka?

- Publish & Subscribe
  - Like a messaging service
- Process
  - Distributed, fault-tolerant events handling
    - On the clients
- Store
  - Distributed, replicated, fault-tolerant storage
    - On the brokers

Kafka?

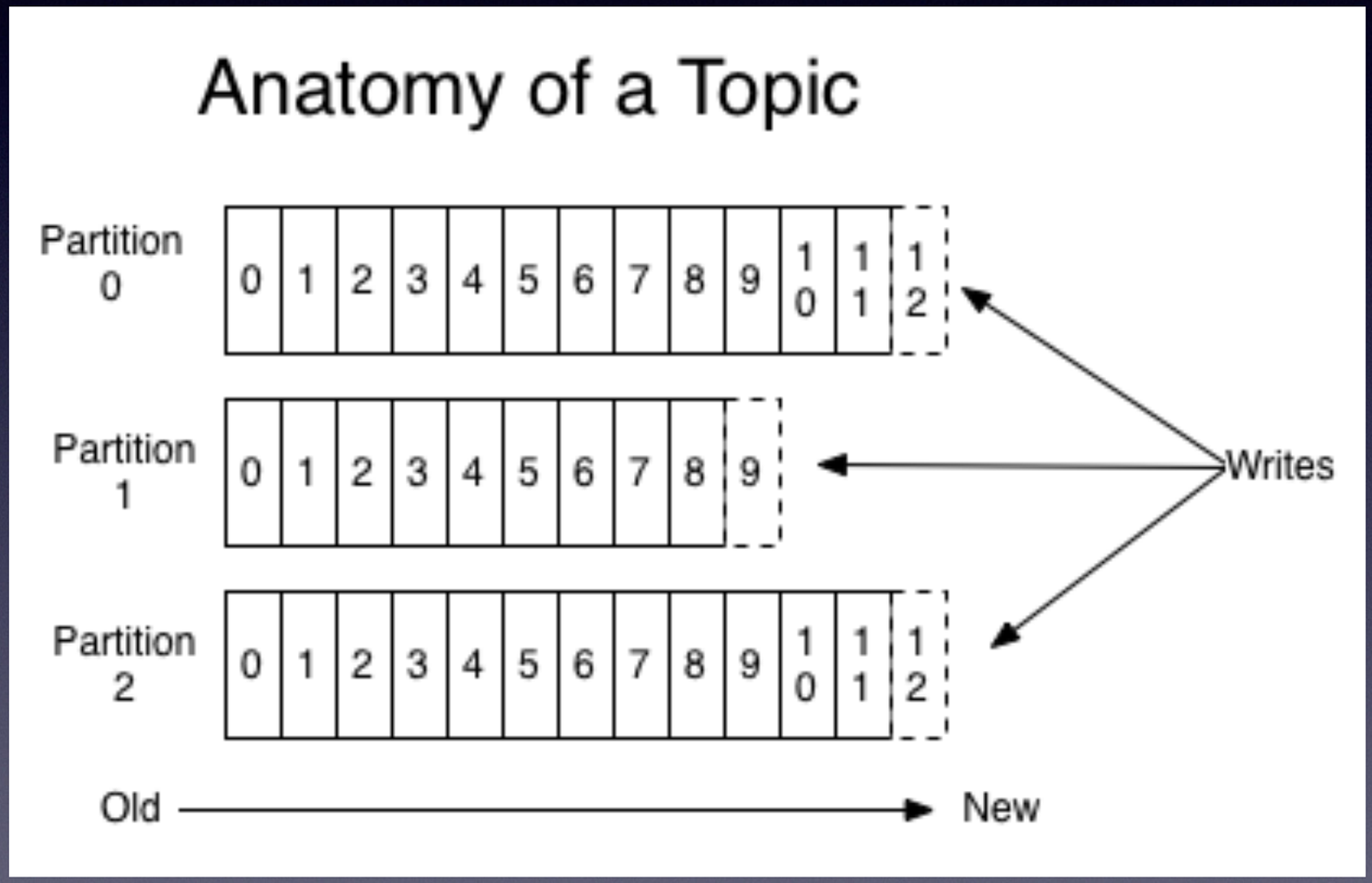
- Concepts
  - Cluster of Kafka servers aka brokers (ZooKeeper + Kafka)
  - Storing streams of records in topics
    - <key, value, timestamp> = record
- Core APIs
  - Producer
  - Consumer
  - Streams
  - Connector (*)



Kafka?

- Communication
  - Client <—> Server
  - Simple, high-performance, language agnostic TCP protocol
  - Versioned and back-compatible protocol
    - Java client by default, support for other languages also available
- Topics (stream of records)
  - Named
  - Partitioned (ordered, immutable sequence — offset / partition)
  - Multi-subscriber (0, 1, many)
  - Retention period

Kafka?


Anatomy of a Topic
Partition 0
0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1
0 1 2 3 4 5 6 7 8 9
Partition 2
0 1 2 3 4 5 6 7 8 9 10 11 12
Writes
Old
New

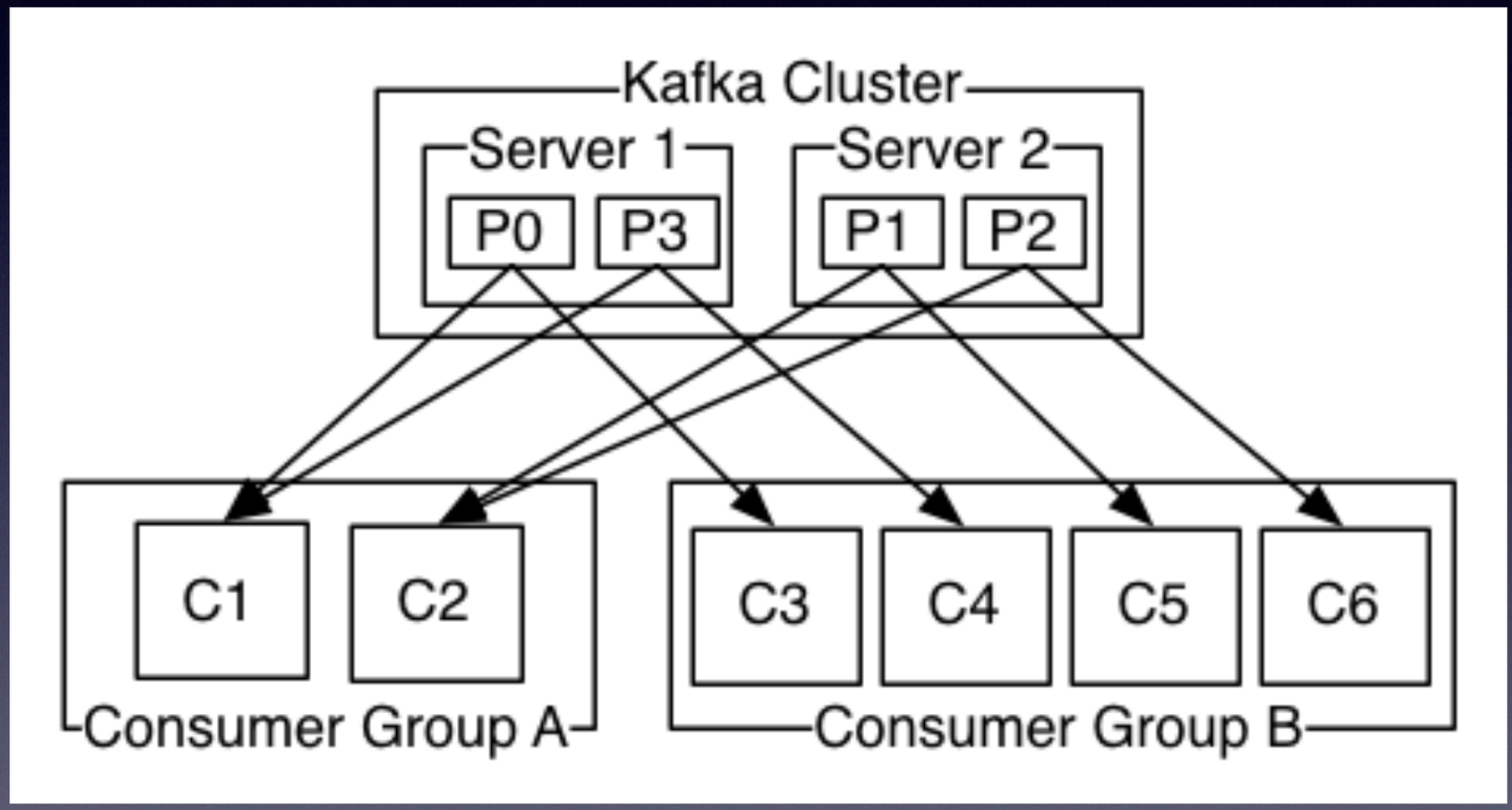
Kafka?

- Producers
  - record target: <topic, partition>
  - partition = hash(<key>) % (#of partitions)
- Produced Record(s) distribution and persistence (from producers to brokers)
  - Leaders (for each <topic, partition>) vs. Followers
- Consumers
  - Each consumer is a single thread of consumption of records from subscribed topic(s)
  - There can be multiple consumers subscribed to same topic(s)
  - May be grouped into consumer groups (consumer group name)

Kafka?

- Consumed Record(s) distribution (from brokers to consumers)
  - Each consumer group receives a separate copy of all records from subscribed topic(s) (pub-sub)
  - Each consumer in a consumer group receives records from a disjunctive subset of partitions from subscribed topic(s) (queue)
  - Ordering is guaranteed only within <topic, partition>
  - Auto-rebalance -- dynamic (re)assignment of subsets of <topic, partition>(s) among "live" consumers of a consumer group

Consumer Groups



Kafka?

- Storage System
  - Guaranteed / ack
  - Same perf (50KB vs 50TB)
    - Read position (offset)
- Stream Processing
  - Streams API — hides complexity
  - Built on core primitives

Trivia

- Kafka was developed in 2010 at LinkedIn. LinkedIn was facing a problem of low latency ingestion of a large amount of data from the website into a lambda architecture which would be able to process events in real-time.

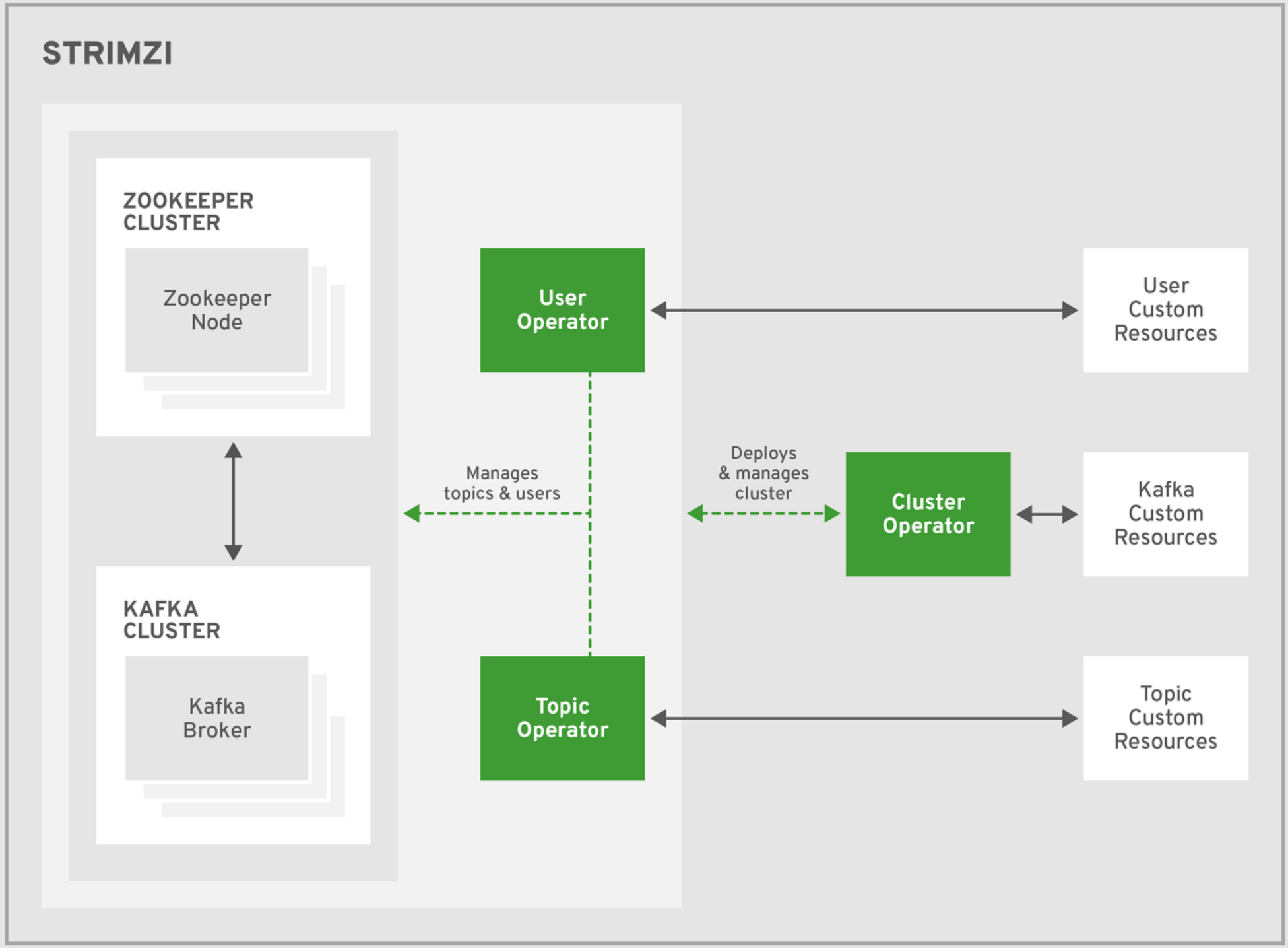
- In 2017, Pintrest built and open-sourced DoctorKafka, a Kafka operations automation service to perform partition reassignment during broker failure for operation automation.

Our use case

- Microservices app
- SpringBoot based
- Running on GoogleCloudPlatform / Kubernetes
- Using Strimzi to run Kafka servers / brokers

Strimzi

- Kubernetes operator for Kafka
- Provides easy deployment of ZooKeeper and Kafka clusters using StatefulSets
- Automatically applies cluster configuration changes
- Provides automatic rebalancing when cluster configuration changes



OPENSHIFT CONTAINER PLATFORM / KUBERNETES

Strimzi

- Running one distributed system (Kafka brokers) that depends on another distributed system (ZooKeeper cluster) on top of a distributed system (Kubernetes) on top of virtual networking and using mounted network volumes for files (instead of local disks).

- What could possibly go wrong :)

Lessons learned!

- Spring-Kafka — yes or no?
    - Started with “yes”, ended with “no”
- Reason(s)?
    - Started - as always :-) - as a bug fix
    - Specific needs -> custom code

“No” Spring-Kafka

- Producer
  - Async-wrapper
  - Recreating on fatal errors (some fatal responses from broker(s) require re-establishing the producer)
  - Result as CompletableFuture
  - What about tx?
    - Long-tx span + send on commit == slow
    - Rather used a chain of Futures + compensate

“No” Spring-Kafka

- Consumer / listener
  - It’s actually a client doing polling all the time
- With exponential re-try on failure (till max delay)
- Proper thread / concurrency handling
  - Dynamically re-assign topics / partitions
  - Dynamically re-size consumers

Lessons learned!

- Serialization
  - Everything is ProtoBuf
- Tracing
  - Wrappers - producer / consumer
  - Write / read to/from record headers

Lessons learned!

- Streams
  - Window-ing is easy and very useful
- Consider (ReadOnly)KeyValueStore over DB
  - Distributed lookup (per key) over gRPC
    - gRPC has streaming ;-)

Lessons learned!

- Cluster configuration
  - For production use 3 ZooKeeper instances
  - For production use 3 Kafka instances or more but make it an odd number

Lessons learned!

- Topic configuration
    - For fault tolerance and robustness use 3 replicas and a minimum in-sync replicas of 2

Strimzi related issues

- Broker restart requires client restarts because a new broker gets a new IP address (Kubernetes stateful sets behaviour)
- One day everything just stops working. There is a problem accessing log files. Turns out when you’re in the cloud and mounting network volumes things can go wrong at your cloud provider. There was a filesystem corruption on the persistent volume.

Strimzi related issues

- Killing Kubernetes brokers results in unclean shutdown (leaves index files corrupted)
- Corrupted indexes are automatically rebuilt on restart - may take a long time (several hours if you have many topics / partitions with a lot of records).
- Override liveness probe for Kafka pods - set initialDelaySeconds to a big value, effectively turning it off.

Strimzi related issues

- Asymetric network failures
- 5 brokers
  - Brokers 1,2,3,4 complain they can’t connect to broker 5
  - Broker 5 is oblivious of any issues, its partitions are not available to clients (service not available!)

Configuration issues

- Topic retention
  - By default your records get deleted if older than a week
    - When creating a topic set retention period explicitly
- Max message size
  - There is message size limit on brokers fetching replicas from other brokers. Your cluster can get stuck on too big messages that can’t be replicated.

Key Lessons

- Kafka Cluster can suffer a failure
- Test your application with realistic data - in terms of numbers of messages, sizes of messages, data ingestion rates.
- Give testing phase enough time. Some issues you'll see for the first time only after months of test usage.
- Consider using Mirror Maker to keep another cluster fully in-sync so you can switch over if needed to quickly recover from the outage.

Ping?

- ales.justin@openblend.org
- @alesj

Q & A