Using MongoDB with Kafka

Percona Live Online
20-21 October 20202

Antonios Giannopoulos

Senior Database Administrator

rackspace technology

Pedro Albuquerque

Principal Database Engineer

TransferWise

Agenda

- Definitions
- Use cases
- Using MongoDB as a source
- Using MongoDB as a sink
- Real world use case: Transferwise
- MongoDB to Kafka Connectors
- Takeaways

What is MongoDB?

- Document-oriented Database
- Flexible JSON-style schema

Use-Cases:

- Pretty much any workload
- After version 4.0/4.2 supports ACID transactions
- Frequent schema changes

What is Apache Kafka?

- Distributed event streaming platform

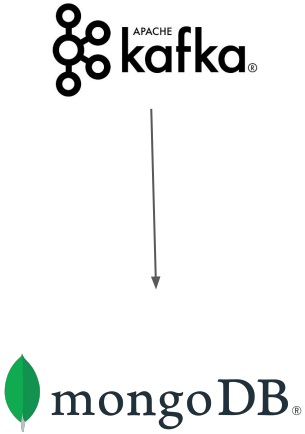
Use-Cases:

- Publish and subscribe to stream of events
- Async RPC-style calls between services
- Log replay
- CQRS and Event Sourcing
- Real-time analytics

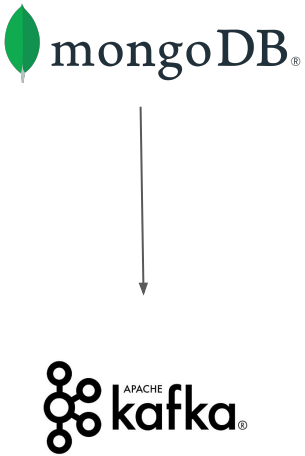
How can they work together?

Use cases - Topologies

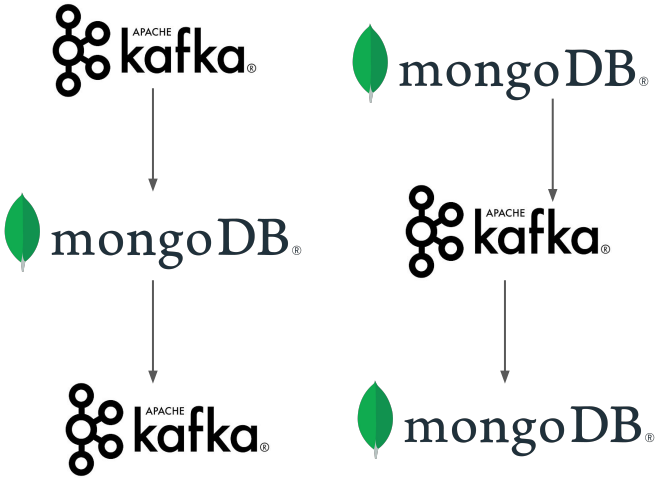
MongoDB as a sink



MongoDB as a source



MongoDB as a source/sink



MongoDB as a Source

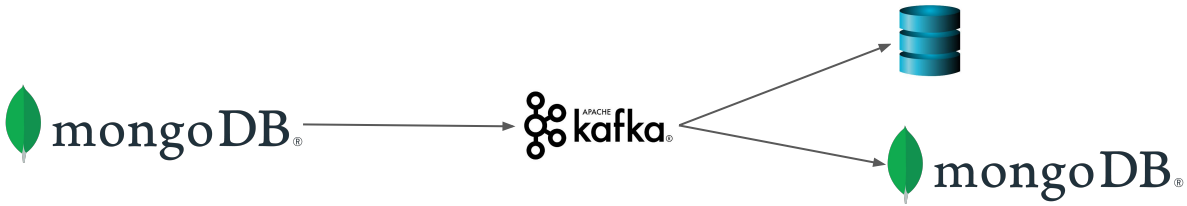
Selective Replication/EL/ETL

MongoDB doesn’t support selective Replication

Oplog or Change Streams (prefered method)

Kafka cluster, with one topic per collection

MongoDB to Kafka connectors



Debezium

Supports both Replica-set and Sharded clusters

Uses the oplog to capture and create events

Selective Replication: [database|collection].[include|exclude].list

EL: field.exclude.list & field.renames

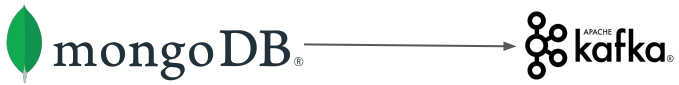
snapshot.mode = initial | never

tasks.max

initial.sync.max.threads

MongoDB Kafka Source Connector

- Supports both Replica-set and Sharded clusters
- Uses MongoDB Change Streams to create events
- Selective Replication:
  - mongodb `db.collection -> db.collection` kafka topic
- Multi-source replication:
  - multiple collections to single kafka topic
- EL: Filter or modify change events with MongoDB aggregation pipeline
- Sync historical data (`copy.existing=true`)
- `copy.existing.max.threads`



MongoDB as a Sink

Throttling

Throttling* (is a forbidden word but) is extremely useful:

- During MongoDB scaling
- Planned or unplanned maintenances
- Unexpected growth events
- Traffic prioritization

The need for throttling: MongoDB 4.2 Flow control

You can configure Flow Control on the Replica-Set level
(Config settings: enableFlowControl, flowControlTargetLagSeconds)

Kafka provides a more flexible “flow control” that you can easily manage

* Throttling may not be suitable for every workloads

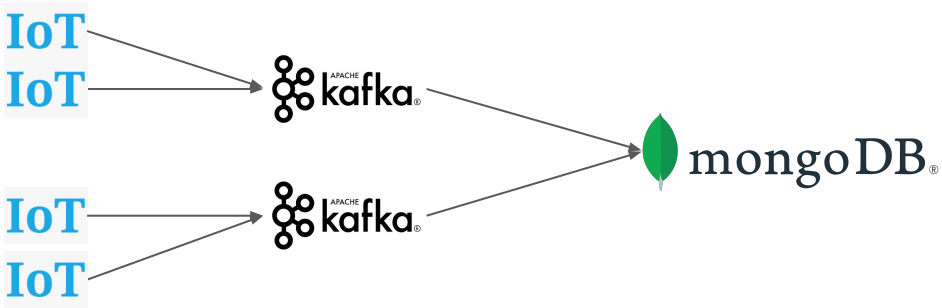
Throttling

The aim is to rate limit **write operations**

Kafka supports higher write throughput & scales faster

Kafka scales:
- Adding partitions
- Add brokers
- Add clusters
- Minimal application changes

MongoDB scales as well:
- Adding shards
- Balancing takes time
- Balancing affects performance



Throttling

Quotas can be applied to (user, client-id), user or client-id groups

producer_byte_rate : The total rate limit for the user's producers without a client-id quota override
consumer_byte_rate : The total rate limit for the user's consumers without a client-id quota override

Static changes: **/config/users/** & **/config/clients** (watch out the override order)

Dynamic changes:

> bin/kafka-configs.sh --bootstrap-server <host>:<port> --describe --entity-type users|clients --entity-name user|client-id

> bin/kafka-configs.sh --bootstrap-server <host>:<port> **--alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048'** --entity-type users|clients --entity-name user|client-id

```
                ¦ ./kafka-configs.sh --bootstrap-server localhost:9092 --describe --user test --all
Configs for user-principal 'test' are consumer_byte_rate=200.0, producer_byte_rate=200.0
                ¦ ./kafka-configs.sh --bootstrap-server localhost:9092 --alter --add-config 'producer_byte_rate=200, consumer_byte_rate=1' --entity-type users --entity-name test
Completed updating config for user test.
                ¦ ./kafka-configs.sh --bootstrap-server localhost:9092 --describe --user test --all
Configs for user-principal 'test' are consumer_byte_rate=1.0, producer_byte_rate=200.0
```

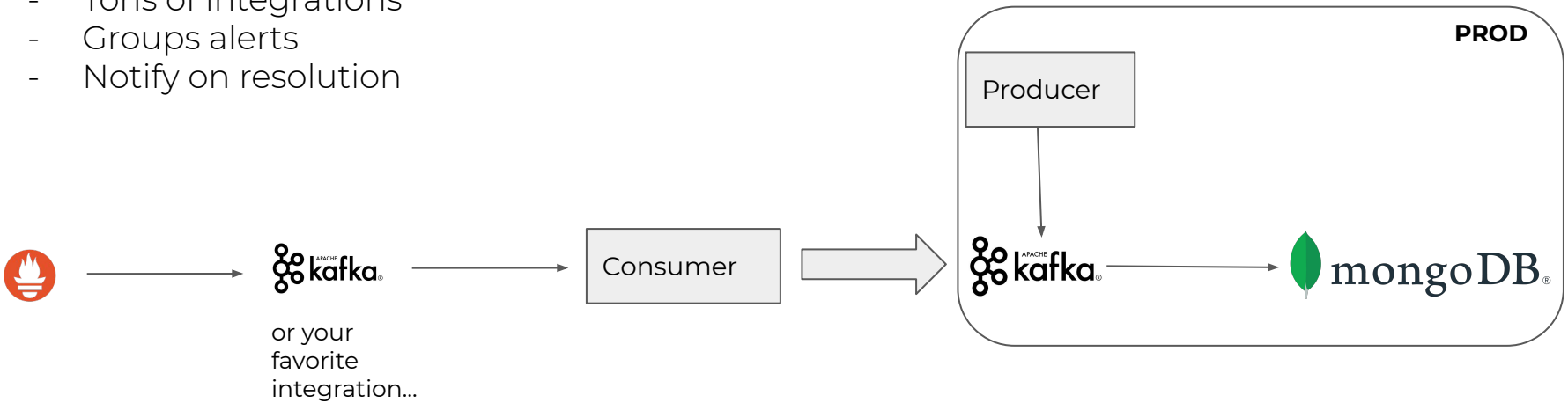
Throttling

Evaluate a MongoDB metric - Read/Write Queues , Latency etc

```
> db.serverStatus().globalLock.currentQueue.writers
0
```

Prometheus Alert Manager

- Tons of integrations
- Groups alerts
- Notify on resolution



Workload isolation

Kafka handles specific workloads better

An successful event website (for example: Percona Live 2020)

- Contains a stream of social media interactions
- Kafka serves the raw stream - all interactions
- MongoDB serves aggregated data - for example top tags

Raw steam is native for Kafka as its a commit-log

MongoDB rich aggregation framework provides aggregated data

Workload isolation

```
from kafka import KafkaClient, KafkaConsumer, TopicPartition

TOPIC="PerconaLive"
START=10

consumer = KafkaConsumer(TOPIC, auto_offset_reset='latest', enable_auto_commit=False)

PARTITIONS = []
for partition in consumer.partitions_for_topic(TOPIC):
    PARTITIONS.append(TopicPartition(TOPIC, partition))

partitions = consumer.end_offsets(PARTITIONS)

for item in partitions:
 print("Printing the last "+str(START)+" for "+str(item))
 START=int(partitions[item]-START)
 consumer = KafkaConsumer()
 partition = TopicPartition(TOPIC,0)
 consumer.assign([partition])
 consumer.seek(partition, START)
 for msg in consumer:
  print(msg)
```

```
Printing the last 10 for TopicPartition(topic=u'PerconaLive', partition=0)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=186, timestamp=1602440651523, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=187, timestamp=1602440651533, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=188, timestamp=1602440651544, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=189, timestamp=1602440651554, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=190, timestamp=1602440651565, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=191, timestamp=1602440651575, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=192, timestamp=1602440651585, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=193, timestamp=1602440651596, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=194, timestamp=1602440651606, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
ConsumerRecord(topic=u'PerconaLive', partition=0, offset=195, timestamp=1602440651617, timestamp_type=0, key='Social', value='This is a post!', headers=[], checksum=None, serialized_key_size=6, serialized_value_size=15, serialized_header_size=-1)
```

Continuous aggregations

Useful for use-cases that raw data are useless (or not very useful)

Kafka streams is your friend - Windowing

Examples:
Meteo stations sending metrics every second
MongoDB serves the min(),max() for every hour

Website statistics - counters
MongoDB gets updated every N seconds with hits summary
MongoDB gets updated with hits per minute/hour

Journal

Data recovery is a usual request in the databases world

Human error, application bugs, hardware failures are some reasons

Kafka can help on partial recovery or point in time recovery

A partial data recovery may require restore of a full backup

Restore changes from a full backup, Replay the changes from Kafka

Journal

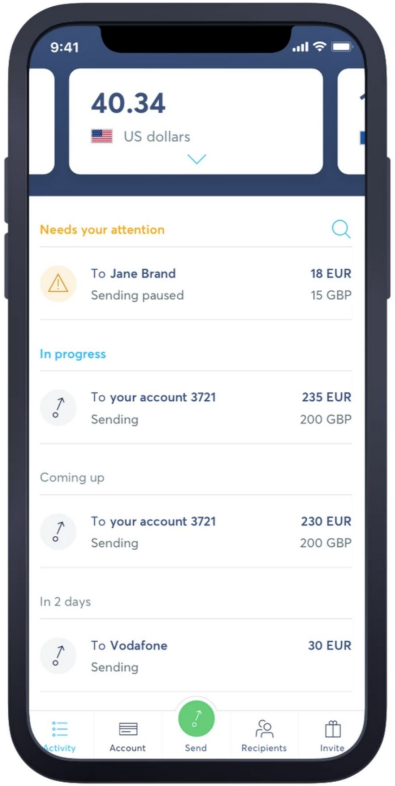
```
> db.users.find()
{ "_id" : "Antonios", "songs" : [ ] }
```

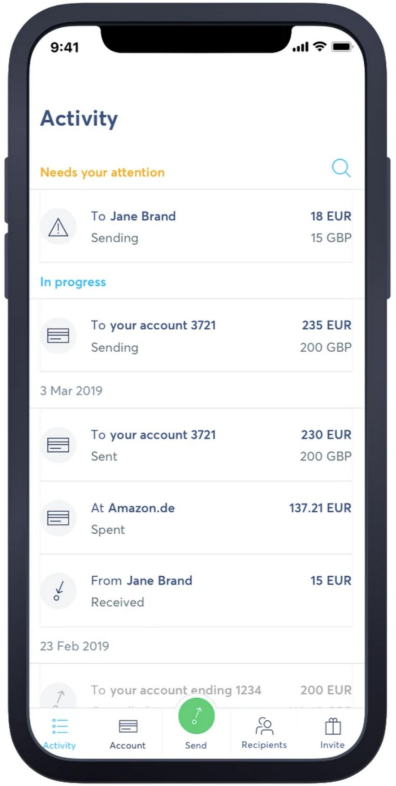
```
from json import dumps
from bson import json_util
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=['localhost:9092'],value_serializer=lambda x: dumps(x).encode('utf-8'))

for i in range(20):
    data = {'songs': 'song'+str(i)}
    print(data)
    producer.send('Plists', key=b'Antonios',value=data)
```

```
from kafka import KafkaClient, KafkaConsumer, TopicPartition
import json
import pymongo
from pymongo import MongoClient

TOPIC="Plists"
client = MongoClient('localhost', 27017)
db = client["kafka"]

consumer = KafkaConsumer(TOPIC, auto_offset_reset='earliest' ,value_deserializer=lambda m: json.loads(m.decode('utf-8')))

for item in consumer:
  name=str(item.key)
  db.users.update_one({"_id": name},{"$addToSet":{"songs":item.value}})
```

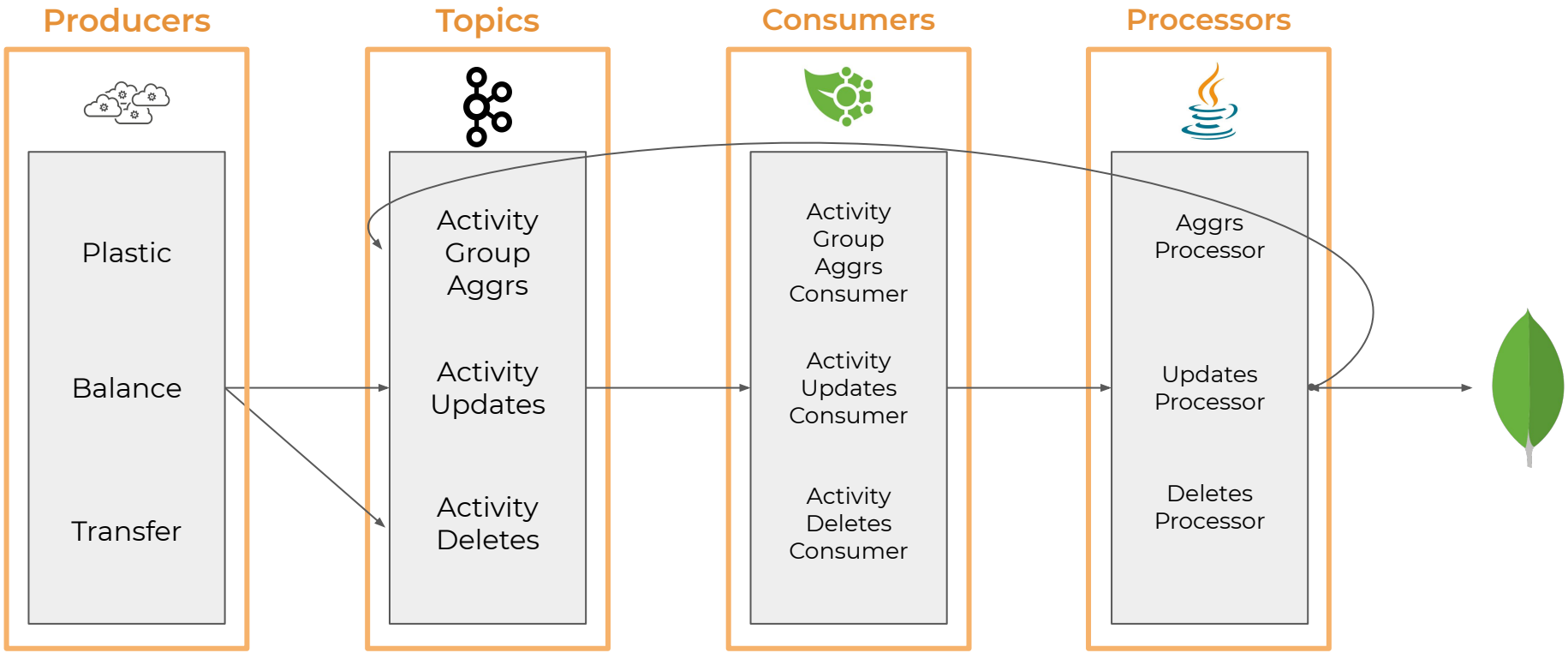
TransferWise: Activity Service

- Customer action
- Many **types**
- Different **status**
- Variety of **categories**
- Repository of all activities
- List of customer's actions
- Activity list
- Ability to search and filter





TransferWise: Activity Service



spring-kafka

Producer configuration

```
private ProducerFactory<Object, Object> producerFactory(KafkaProperties kafkaProperties) {
       return new DefaultKafkaProducerFactory<>(
           Map.of(
               ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getServers(),
               ProducerConfig.CLIENT_ID_CONFIG, kafkaProperties.getClientId(),
               ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, JsonSerializer.class,
               ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class
           )
       );
   }
public KafkaTemplate<Object, Object> kafkaTemplate(KafkaProperties kafkaProperties) {
       return new KafkaTemplate<>(producerFactory(kafkaProperties));
   }
```

spring-kafka

Send message

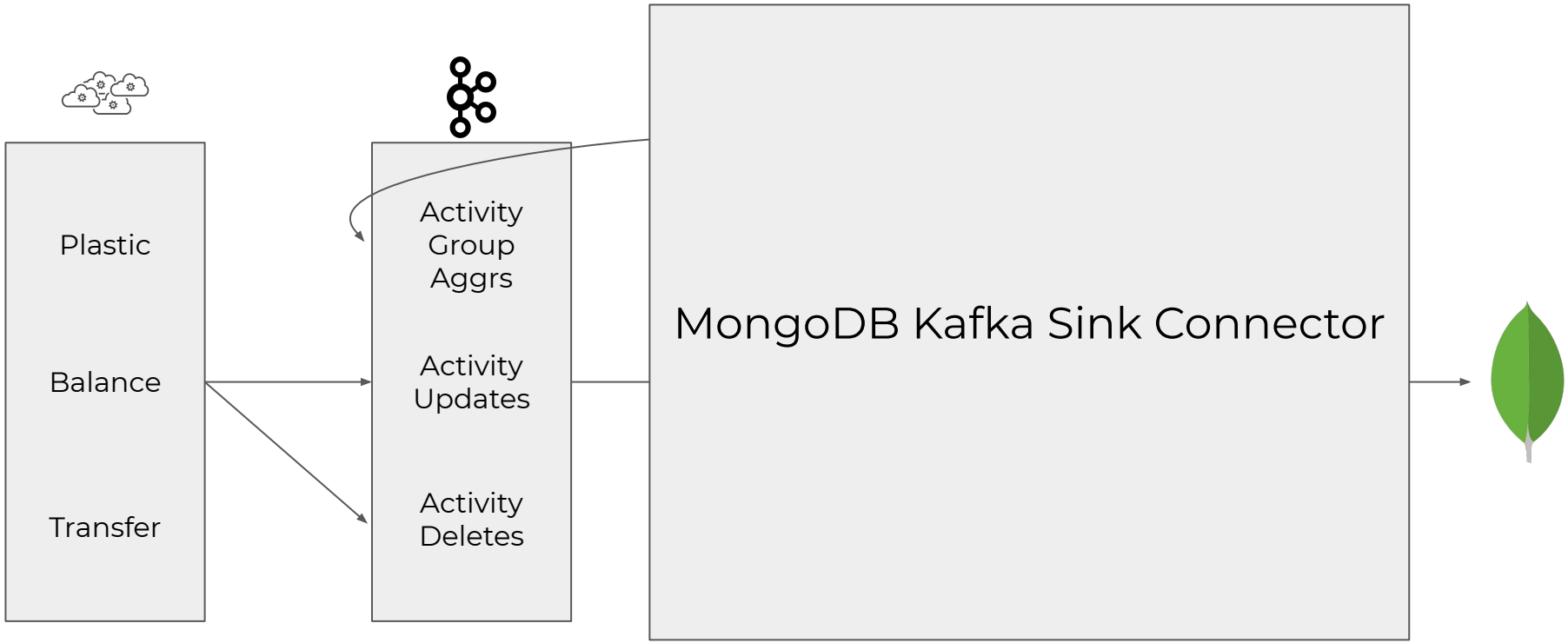
```
public void send(String key, Object value, Runnable successCallback) {
String jsonBody = value.getClass() == String.class ? (String) value : JSON_SERIALIZER.writeAsJson(value);
       kafkaTemplate.send(topic, key, jsonBody)
           .addCallback(new ListenableFutureCallback<>() {
               @Override
               public void onFailure(Throwable ex) {
                   log.error("Failed sending message with key {} to {}", key, topic);
               }
               @Override
               public void onSuccess(SendResult<String, String> result) {
                   successCallback.run();
               }
           });
   }
```

spring-kafka

Consumer configuration

```
@EnableKafka
private ConsumerFactory<String, String> consumerFactory(KafkaProperties kafkaProperties) {
       return new DefaultKafkaConsumerFactory<>(
           Map.of(
               ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getServers(),
               ConsumerConfig.CLIENT_ID_CONFIG, kafkaProperties.getClientId(),
               ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class,
               ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class
           ));}
ConcurrentKafkaListenerContainerFactory<String, String> factory = buildListenerContainerFactory(objectMapper,
kafkaProperties);
       KafkaRetryConfig retryConfig = new KafkaRetryConfig(KafkaProducerFactory.kafkaTemplate(kafkaProperties));

@KafkaListener(topics = "${activity-service.kafka.topics.activityUpdates}", containerFactory =
ActivityUpdatesKafkaListenersConfig.ACTIVITY_UPDATES_KAFKA_LISTENER_FACTORY)
```

TransferWise: Activity Service



MongoDB Kafka Sink connector: Configuration

```
name=mongodb-sink-example
topics=topicA,topicB
connector.class=com.mongodb.kafka.connect.MongoSinkConnector
tasks.max=1

# Specific global MongoDB Sink Connector configuration
connection.uri=mongodb://mongod1:27017,mongod2:27017,mongod3:27017
database=perconalive
collection=slides
```

MongoDB Kafka Sink connector: Configuration

```
# Message types
key.converter=io.confluent.connect.avro.AvroConverter
key.converter.schema.registry.url=http://localhost:8081
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.schema.registry.url=http://localhost:8081
```

MongoDB Kafka Sink connector: Configuration

```
## Document manipulation settings
[key|value].projection.type=AllowList
[key|value].projection.list=name,age,address.post_code

## Id Strategy
document.id.strategy=com.mongodb.kafka.connect.sink.processor.id.strategy.BsonOidStrategy
post.processor.chain=com.mongodb.kafka.connect.sink.processor.DocumentIdAdder
```

MongoDB Kafka Sink connector: Configuration

```
## Dead letter queue
errors.tolerance=all
errors.log.enable=true
errors.log.include.messages=true
errors.deadletterqueue.topic.name=perconalive.deadletterqueue
errors.deadletterqueue.context.headers.enable=true
```

Recap/Takeaways

There are tons of use-cases for MongoDB & Kafka

We described couple of use-cases

- Selective replication/ETL
- Throttling/Journaling/Workload Isolation

Kafka has a rich ecosystem that can expand the use-cases

Connectors is your friend, but you can build your own connector

Large orgs like TransferWise use MongoDB & Kafka for complex projects

- Thank you!!! -

- Q&A -

Big thanks to:
John Moore, Principal Engineer @Eventador
Diego Furtado, Senior Software Engineer @TransferWise
for their guidance