IPPON

Digital . Technologies . Hosting

Meet Kafka
“the distributed log”

kafka

BWM- 27 octobre 2017 - Lyon

Florian Garcia

Software engineer at Ippon

fgarcia@ippon.fr

@lm_flog

- Backend lover
- Zipkin committer in spare time
- Interested in
  - Distributed tracing
  - Kafka
  - Docker
- Speaker

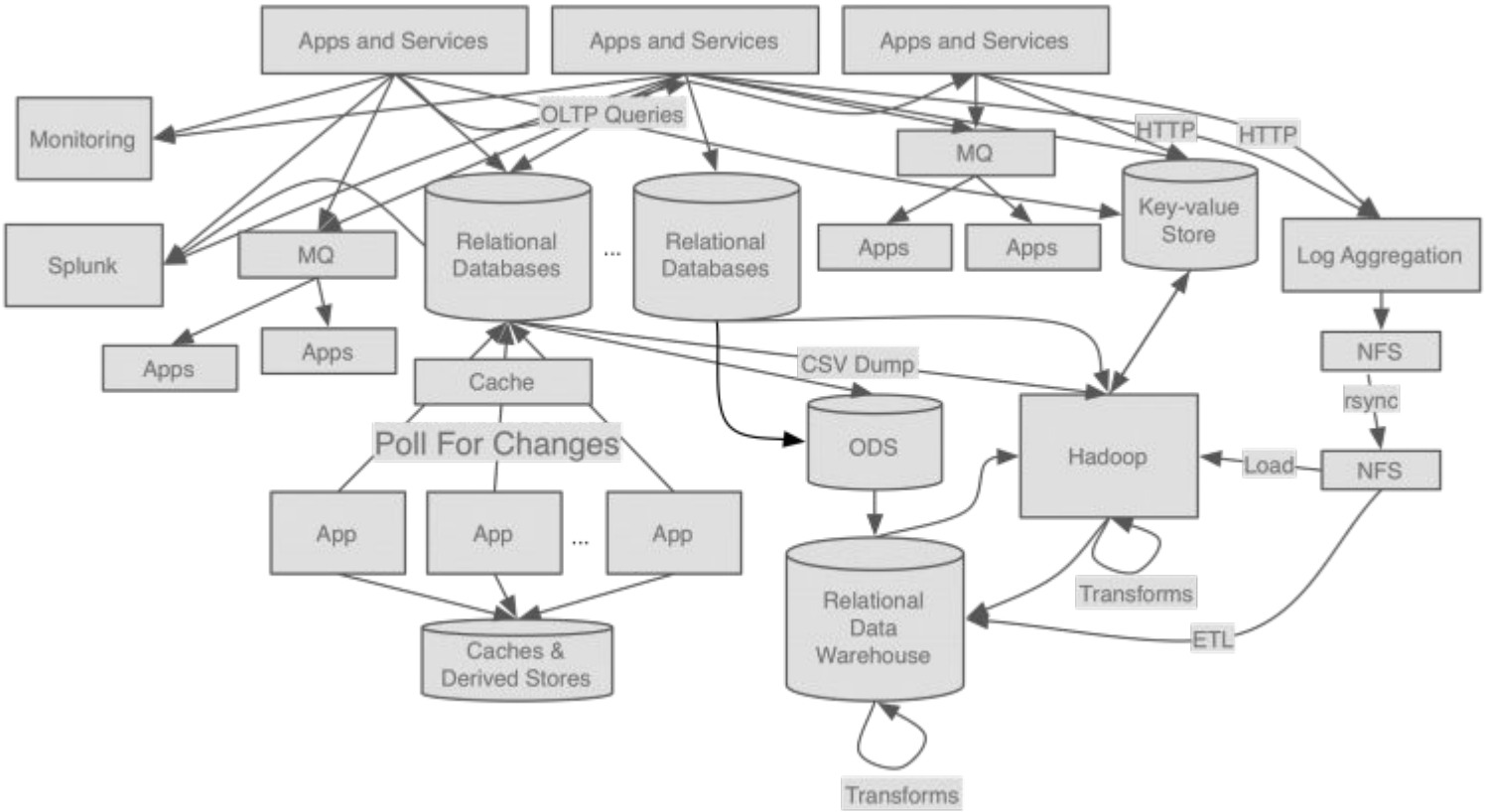
Ippon Technologies © 2017

Kafka

- Kafka who are you?
- Kafka use cases
- Kafka 101
- Kafka Streams
- Kafka from the trenches

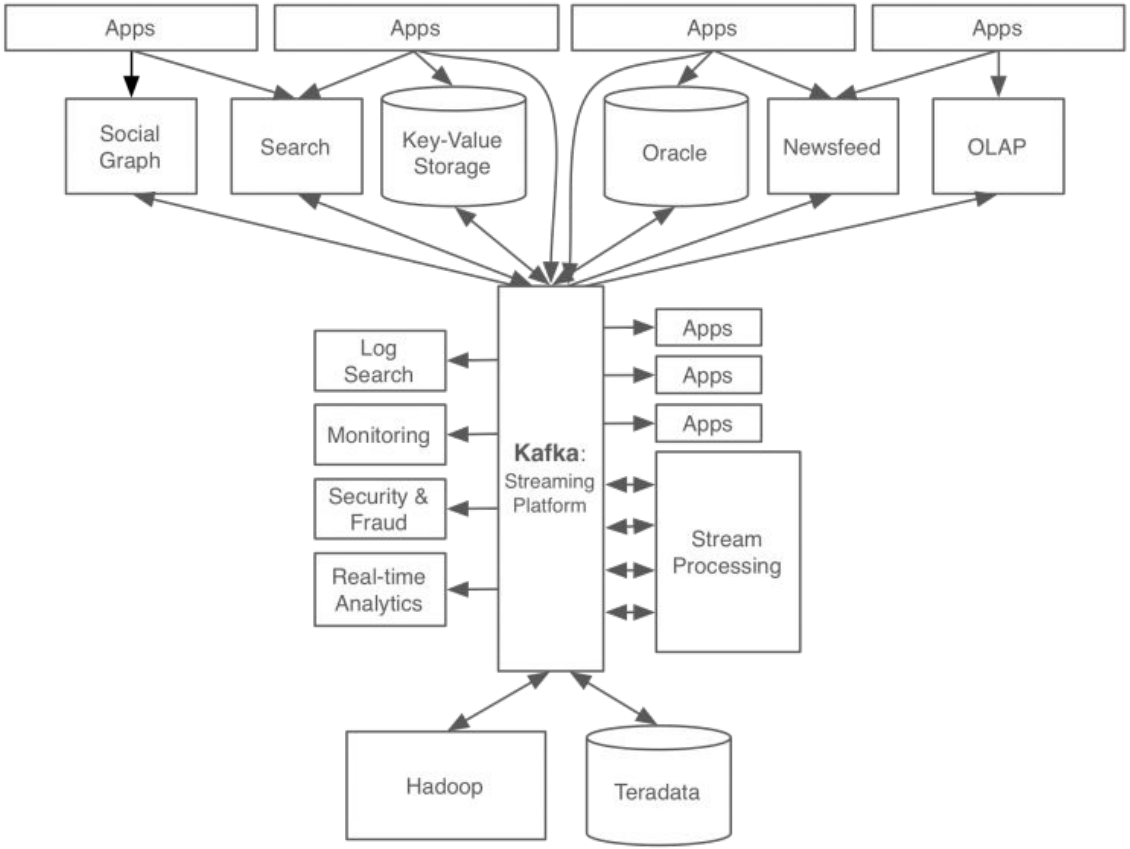
Kafka, who are you?

Linkedin before Kafka (pre 2010)



Ippon Technologies © 2017

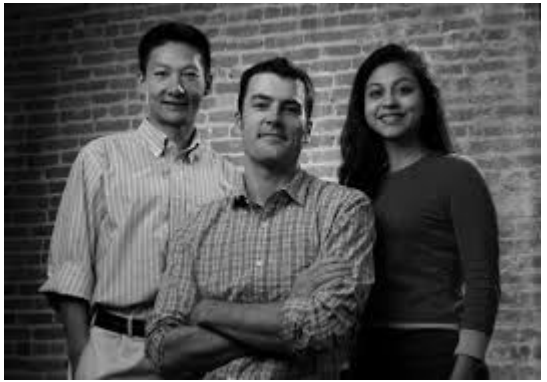
Linkedin after Kafka



Ippon Technologies © 2017

What about today ?

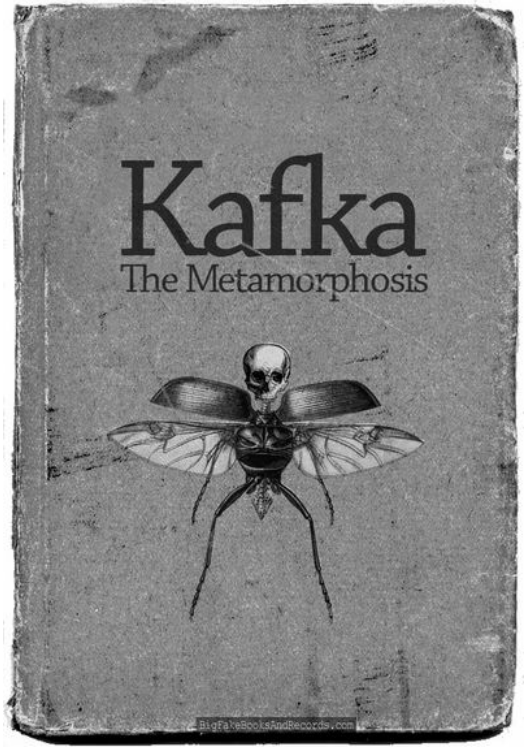
- Open sourced in 2011
- Now a top level Apache project
  - Very active
  - Current release 0.11.0.X
- Spinoff company Confluent created in 2014
  - Jay Kreps, Neda Narkhede and Jun Rao
  - Ships the Confluent platform
  - Recently raised $50M in a round C funding



Ippon Technologies © 2017

The metamorphosis

- Building a Kafka ecosystem
  - Kafka Connect
  - Schema Registry
  - Integrations
- A lot of new features regularly added
- From distributed log to a streaming platform
- End to end exactly once processing !



Ippon Technologies © 2017

Glossary

- Broker ⇔ a server running Kafka
- Topic ⇔ category where records are published
- Partition ⇔ shard of a topic
- Producer ⇔ client writing to Kafka
- Consumer ⇔ client reading from Kafka

Ippon Technologies © 2017

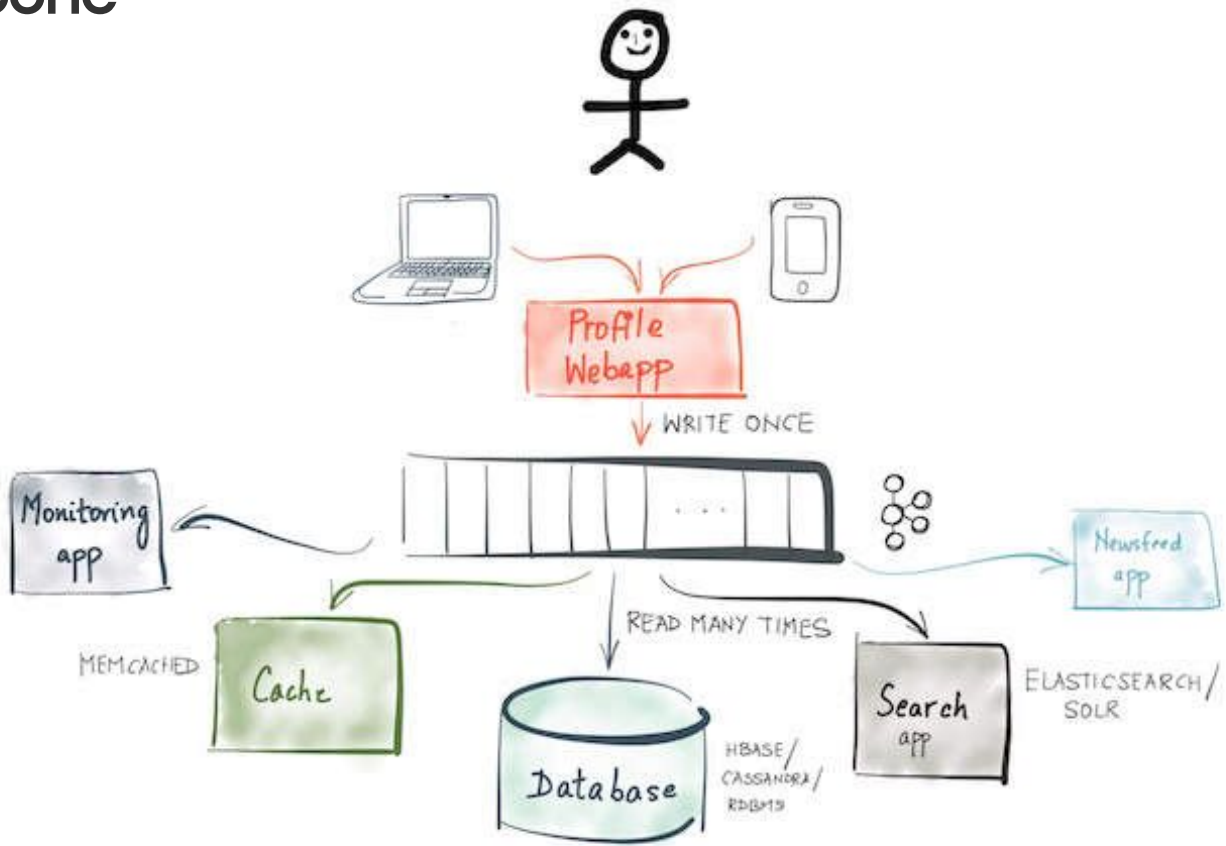
Kafka Use Cases

Data deluge

- User activity
- Server info
- Messaging
- Business data consolidation
- Event sourcing

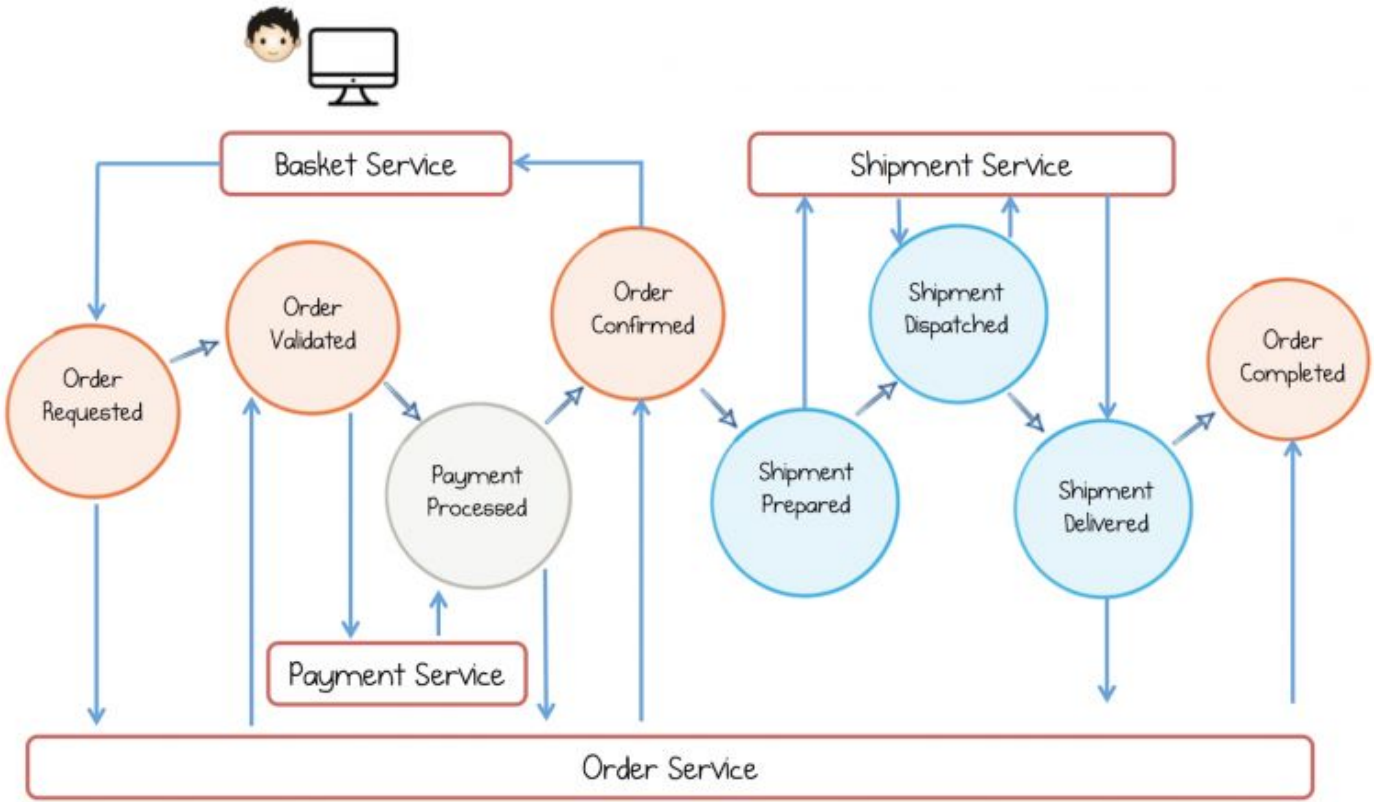
Ippon Technologies © 2017

As a backbone



Ippon Technologies © 2017

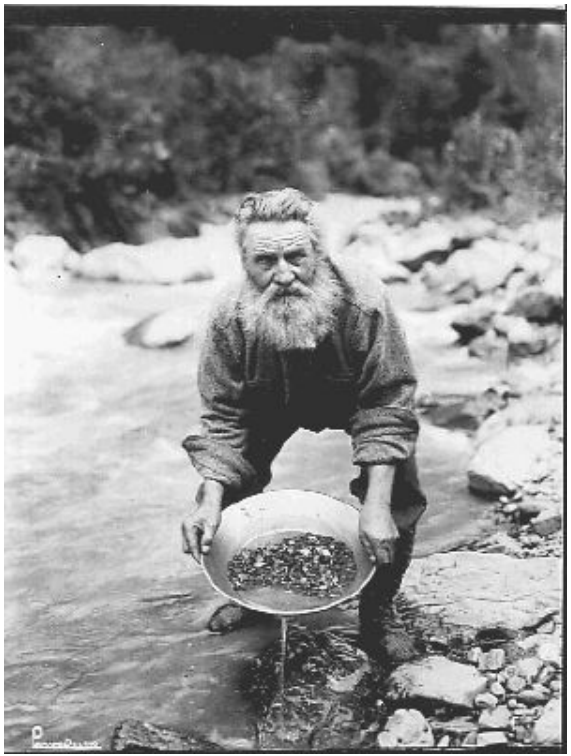
For event driven architectures



Ippon Technologies © 2017

Streaming !

- Process data as soon as they are available !
- Powerful
- Easy to use
- Real time
  - decision
  - data consolidation
  - monitoring
  - fraud detection
  - ...



Ippon Technologies © 2017

But wait, I already bought an ESB

- ESB
  - complex logic
  - routing
  - processing
- Smart endpoints, dumb pipes !
- Kafka justs transports the data
- Get rid of the ESB before you need to sell your children

Ippon Technologies © 2017

But wait, I already use RabbitMQ

- RabbitMQ
  - Broker centric
  - Complex routing
  - Mature (Erlang)
  - Standardized (AMQP)
- Kafka
  - Consumer / producer centric
  - Ordered (per partition)
  - Higher performance
  - Backed by confluent

Ippon Technologies © 2017

Who uses Kafka ?













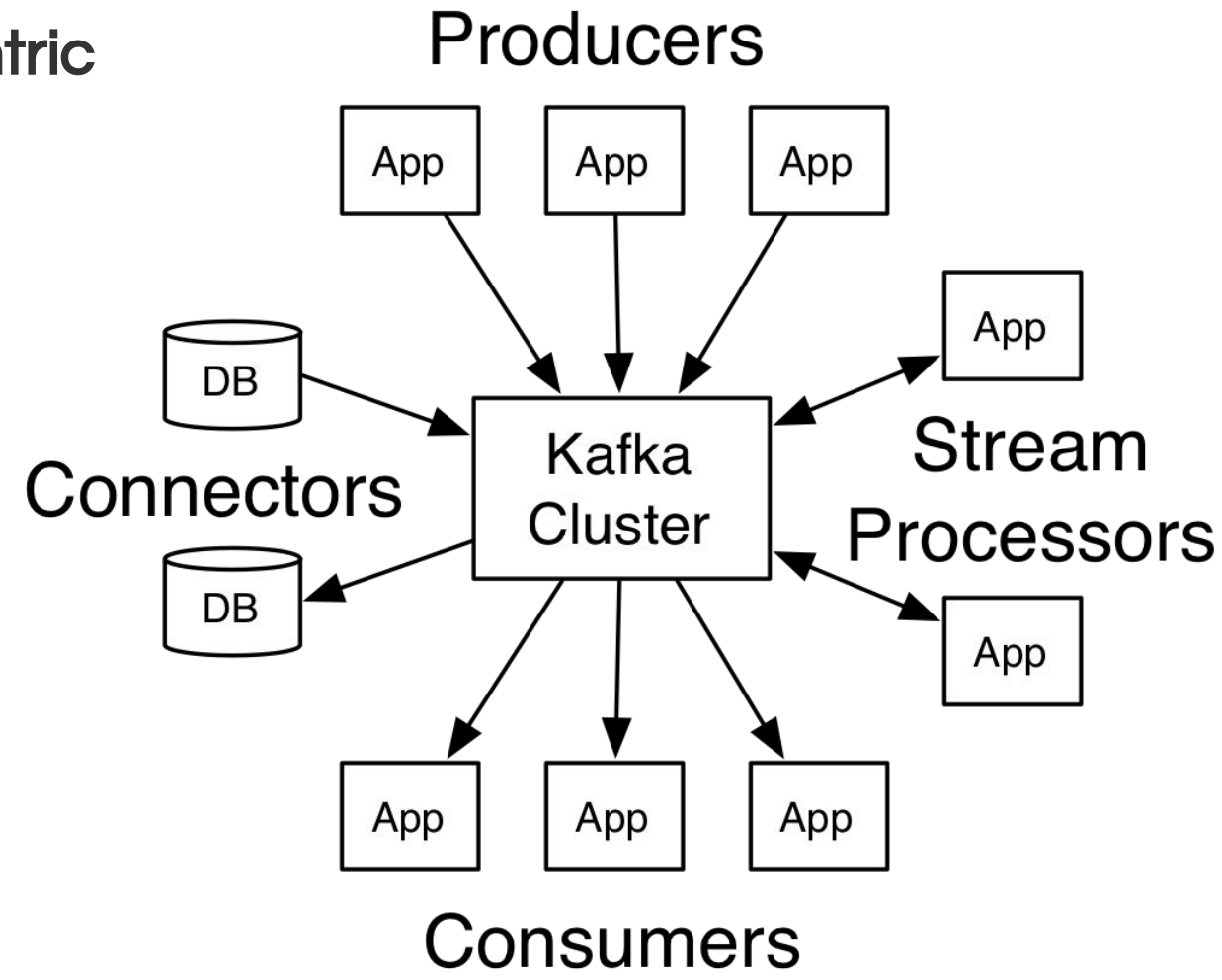



https://kafka.apache.org/powered-by

Ippon Technologies © 2017

Kafka 101

Kafka centric



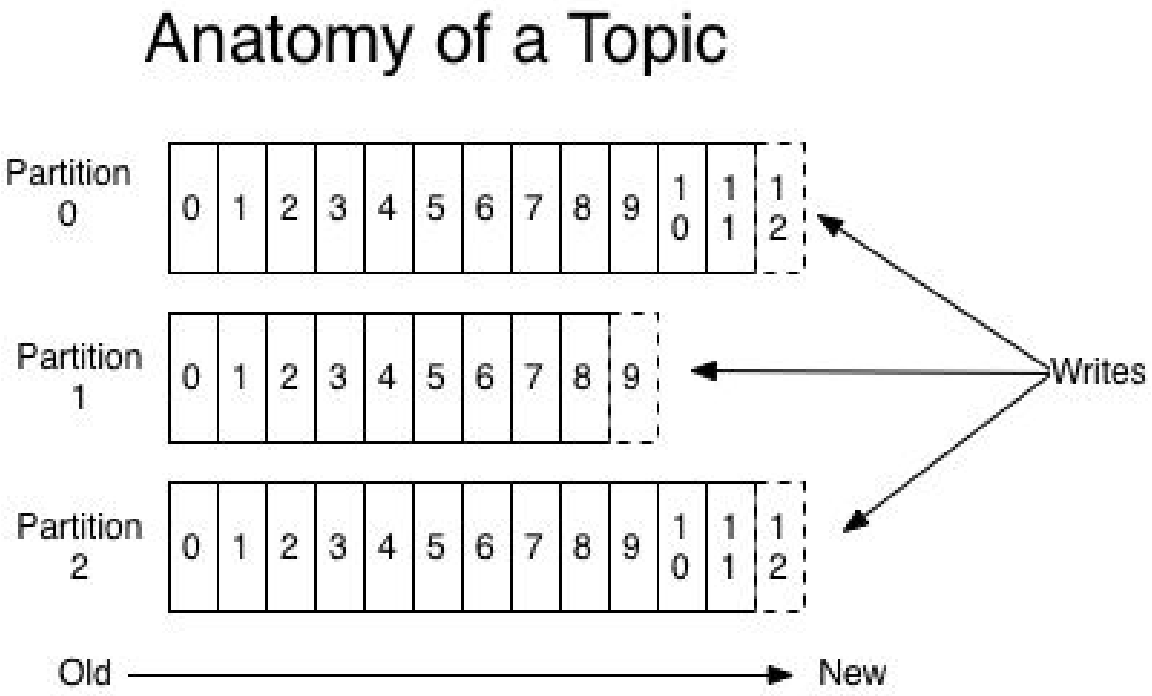
Ippon Technologies © 2017

Brokers

- A running Kafka service
- Role
  - Read / write on disk
  - Consumer / producer interface
  - Group balancing, leader election
- Uses Zookeeper for coordination

Ippon Technologies © 2017

Topics and partitions



Ippon Technologies © 2017

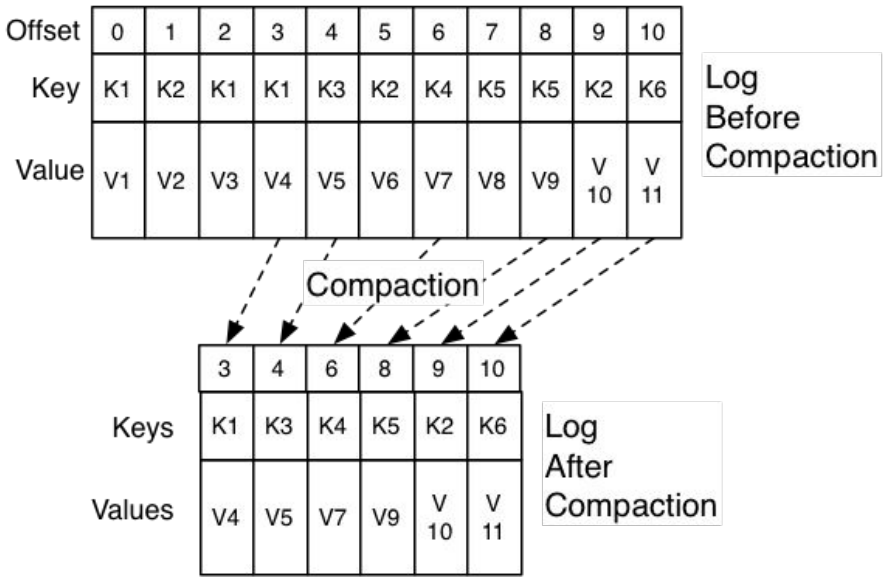
Message content

- **RecordBatch**
  - FirstOffset & LastOffsetDelta
  - FirstTimestamp & MaxTimestamp
  - MagicByte => 2 for version > 0.11.X
  - Records
- **Record**
  - Timestamp & Offset deltas
  - Key
  - Value
  - Headers

Ippon Technologies © 2017

Compaction

- Keep only the last log for a given key
- Happens after a new segment creation by the log cleaner



Ippon Technologies © 2017

Kafka guarantees

Understanding Streaming Semantics

At most once	At least once	Exactly once
Message pulled once	Message pulled one or more times; processed each time	Message pulled one or more times; processed once
May or may not be received	Receipt guaranteed	Receipt guaranteed
No duplicates	Likely duplicates	No duplicates
Possible missing data	No missing data	No missing data

5

MEMSQL

Ippon Technologies © 2017

Kafka Clients

- Low level libraries
  - Java
  - C/C++ (librdkafka)
- Built on top of librdkafka
  - C#
  - Go
  - Haskell
  - Python
  - ...

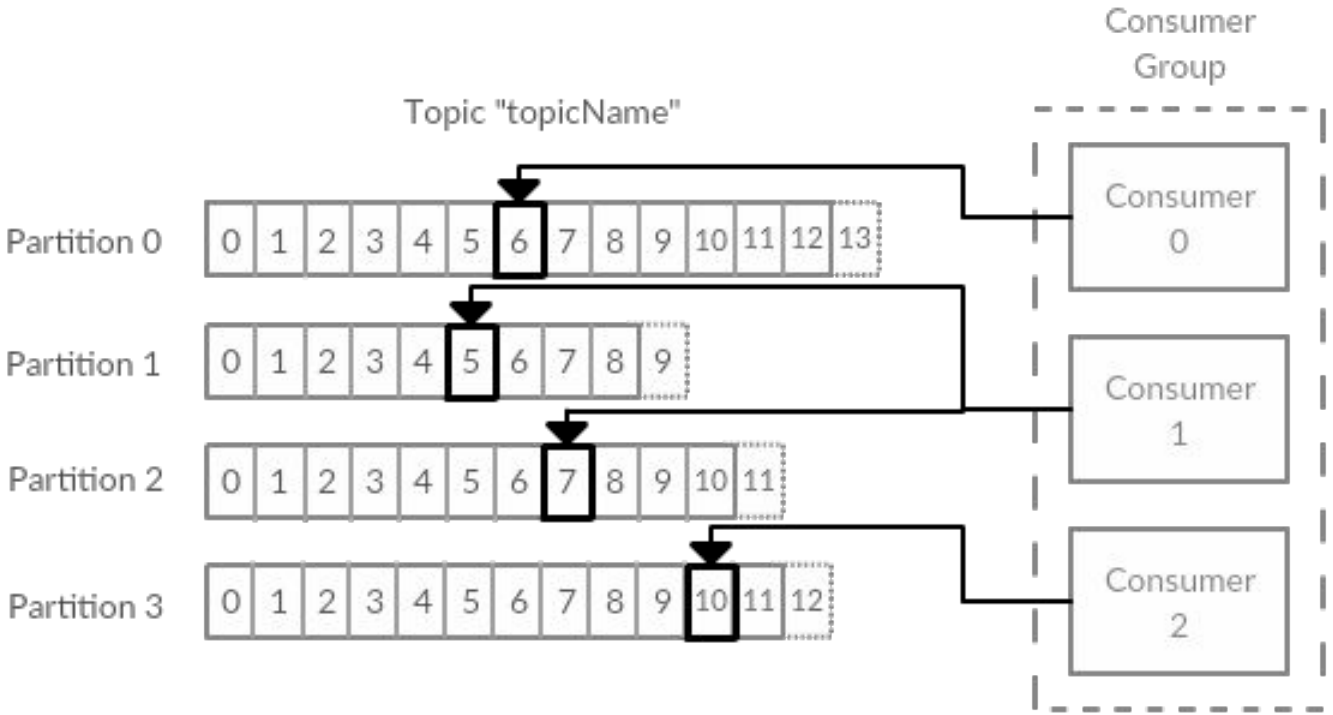
Ippon Technologies © 2017

Producing

- Send data to topics
- Binary content
- The key define the partition
- Send asynchronously
  - buffer pending records
  - batch records for efficiency
- Retry in case of failure

Ippon Technologies © 2017

Consuming



Ippon Technologies © 2017

Dataset - [french opendata]

Ippon Technologies © 2016

Example : producer (settings)

```
private static Properties buildKafkaProps(boolean withTransaction) {
    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092");
    props.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

    // Transaction support
    if (withTransaction) {
        props.put("transactional.id", "myAppId");
    }
    // Exactly once
    props.put("enable.idempotence", true);
    return props;
}
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Example : producer (without transactions)

```
try (KafkaProducer<Integer, String> kafkaProducer = new KafkaProducer<>(props)) {
    kafkaProducer.send(
            new ProducerRecord<>(
                    EFFECTIFS_TOPIC,
                    effectif.getYear(), // Shard by year
                    jsonMapper.writeValueAsString(effectif)));
}
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Example producer (with transactions)

```
try (KafkaProducer<Integer, String> kafkaProducer = new KafkaProducer<>(props)) {
    kafkaProducer.initTransactions();

    kafkaProducer.beginTransaction();
    kafkaProducer.send(
            new ProducerRecord<>(
                    EFFECTIFS_TOPIC,
                    effectif.getYear(), // Shard by year
                    jsonMapper.writeValueAsString(effectif)));
    kafkaProducer.commitTransaction();
}
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Exemple : consumer

```
private static Properties buildKafkaProps() {
    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092");
    props.put("key.deserializer", "org.apache.kafka.common.serialization.IntegerDeserializer");
    props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    // Consumer group ID
    props.put("group.id", "basic");

    // Enabling transaction support
    props.put("isolation.level", "read_committed");
    return props;
}
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Exemple : consumer

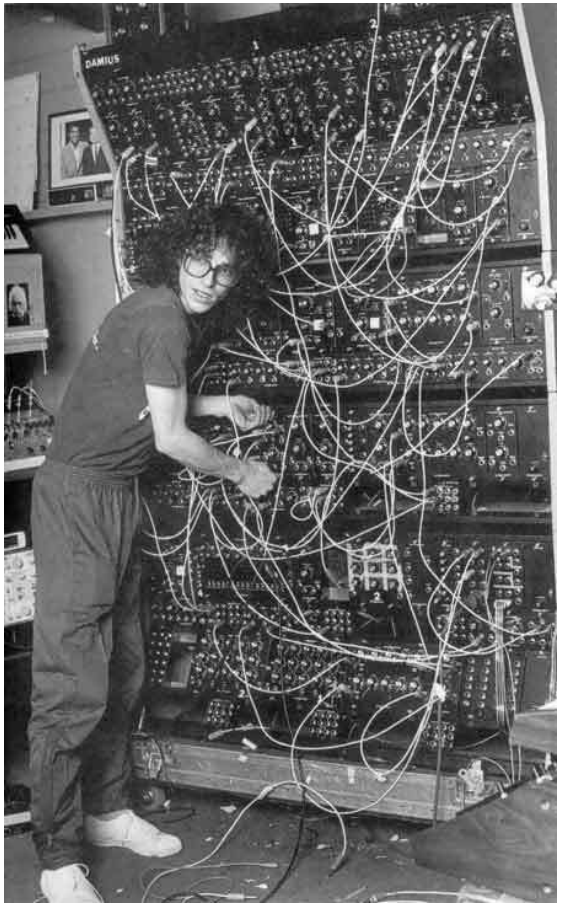
```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Collections.singleton(EFFECTIFS_TOPIC));
while (running) {
    ConsumerRecords<String, String> records = consumer.poll( timeout: 100);
    for (ConsumerRecord<String, String> record : records) {
            Effectif effectif = jsonMapper.readValue(record.value(), Effectif.class);
    }
}
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Kafka Connect

- Standardizes integration with other systems
  - Community connectors
  - Custom connectors
- Working modes
  - Distributed
  - Standalone
- Automatic offset management
- Distributed and scalable by default



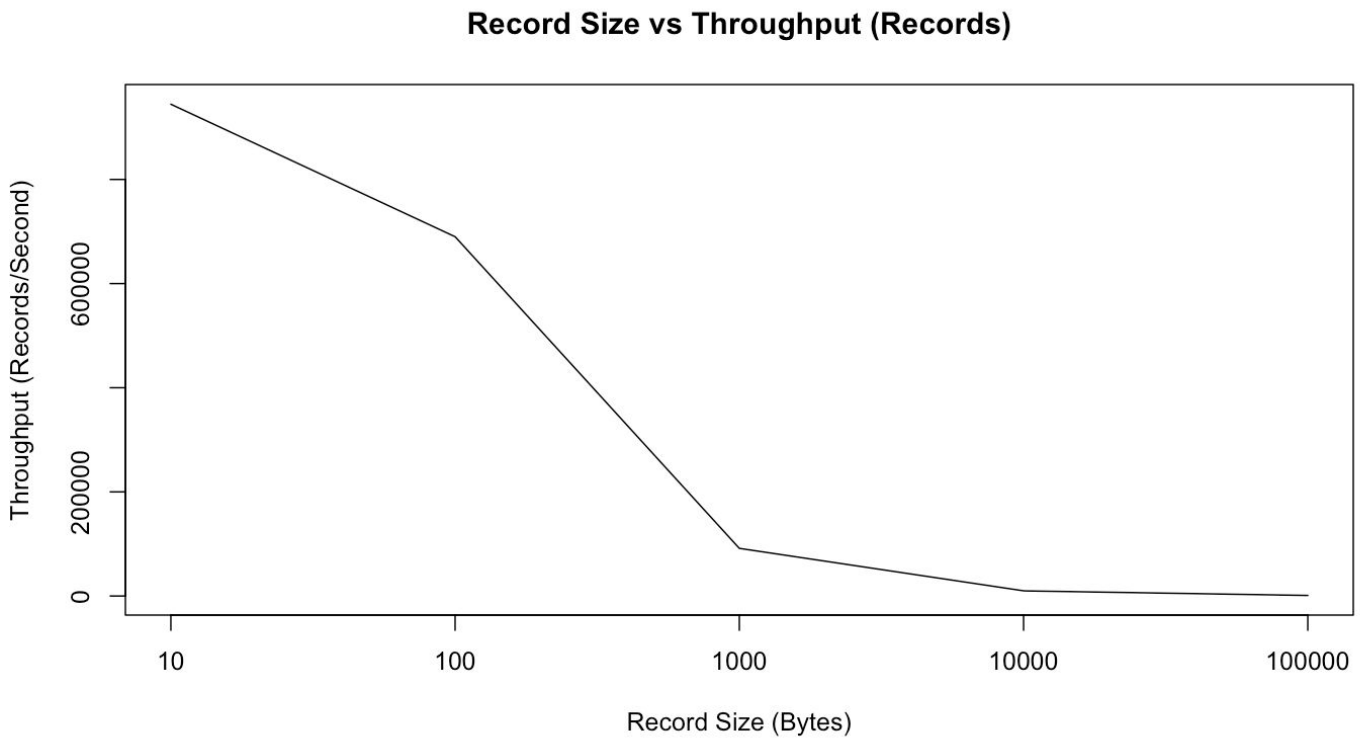
Ippon Technologies © 2017

Confluent platform additions

- Schema Registry
- REST Proxy
- Connector REST interface ($)
- Additional connectors ($)
- Kafka replicator ($)
- Support ($)

Ippon Technologies © 2017

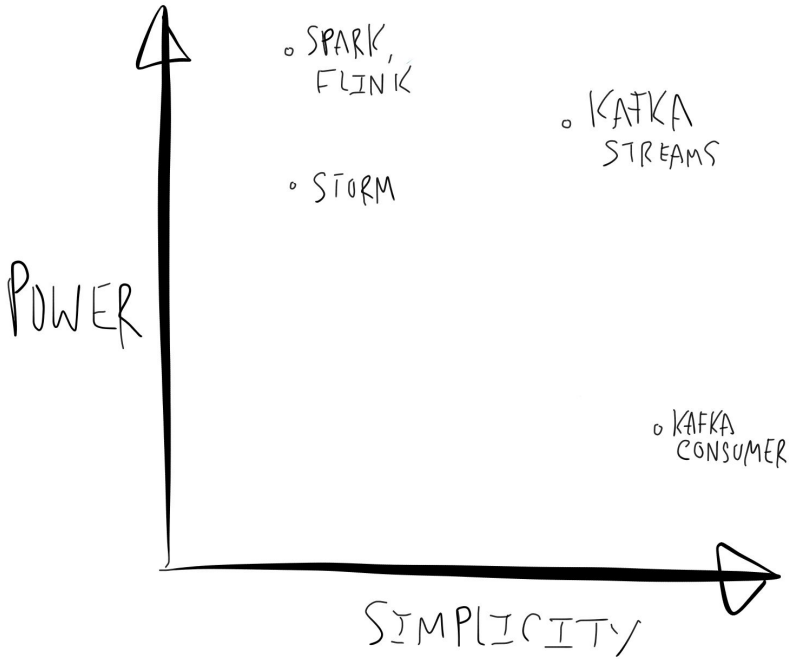
Performance



Ippon Technologies © 2017

Kafka Streams

The goal



Source www.confluent.io

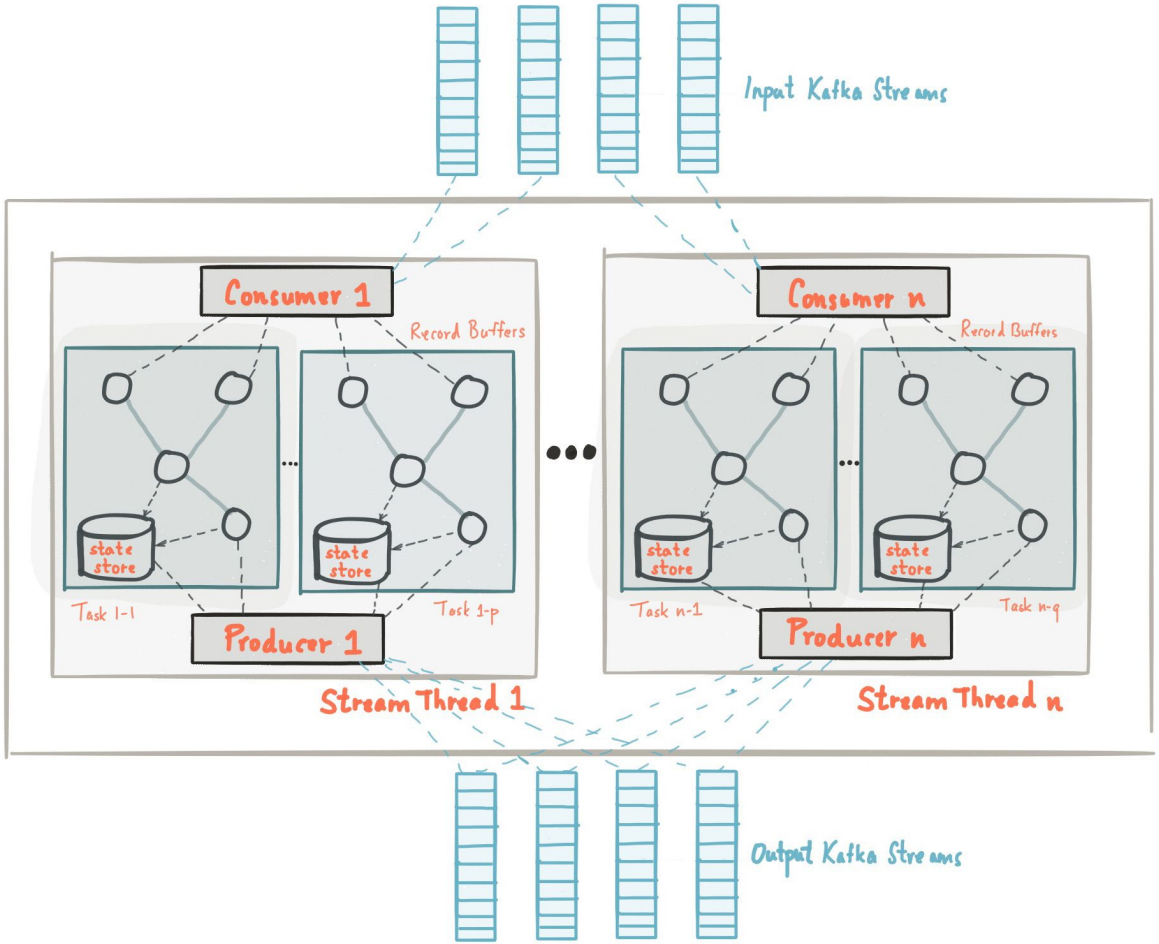
Ippon Technologies © 2017

Kafka streams

- Available since version 0.10.0
- A simple and lightweight client library
- Event-at-a-time processing
- Millisecond latency
- Stateful processing
- Exactly once processing !

Ippon Technologies © 2017

Anatomy



Source: Kafka documentation

Ippon Technologies © 2017

KStream

- An abstraction of a record stream
  - Each data is an “insert”
  - No record replaces an existing row with the same key

- Example
  1. ("Blend", 1)
  2. ("Blend", 3)
  3. Sum(“Blend”) = ?
  4. Sum(“Blend”) = 4

Ippon Technologies © 2017

KTable

- An abstraction of a changelog stream
  - Each data is an “upsert”
  - Based on the record key

- Example
  1. ("Blend", 1)
  2. ("Blend", 3)
  3. Sum(“Blend”) = ?
  4. Sum(“Blend”) = 3

Ippon Technologies © 2017

For developers

- Java only
- kafka-stream dependency
- A high level API
  - Use java 8 lambdas
  - Map, filter, join, ...
- A low level API
  - Add and connect processors
  - Interact directly with state stores

Ippon Technologies © 2017

Operations

- Stateless transformations
  - map
  - groupBy
  - filter
- Stateful transformations
  - Windowing
  - Joins
  - Aggregations

Ippon Technologies © 2017

Example

```
KTable<Integer, Integer> studentsPerYear = kStreamBuilder
        .stream(Serdes.Integer(), effectifSerde, inputTopic)
        .filter((year, effectif) -> !"TOTAL".equals(effectif.getGroup()))
        .filter((year, effectif) -> "Commune".equals(effectif.getGeographicLevel()))
        .map((year, effectif) -> new KeyValue<>(year, effectif.getStudentCount()))
        .groupByKey(Serdes.Integer(), Serdes.Integer())
        .aggregate(
                () -> 0,
                (aggKey, value, aggregate) -> aggregate + value,
                Serdes.Integer(),
                s: "studentsPerYear");

studentsPerYear.to(Serdes.Integer(), Serdes.Integer(), s: "studentsPerYearTopic");
```

https://github.com/lmFlog/kafka-playground

Ippon Technologies © 2017

Example

```
KTable<String, Integer> studentsPerCommune2014 = new KStreamBuilder()
        .stream(Serdes.Integer(), effectifSerde, inputTopic)
        .filter((year, effectif) -> !"TOTAL".equals(effectif.getGroup()))
        .filter((year, effectif) -> "Commune".equals(effectif.getGeographicLevel()))
        .filter((year, effectif) -> year == 2014)
        .groupBy((year, effectif) -> effectif.getGeographicUnit(), Serdes.String(), effectifSerde)
        .aggregate(
                () -> 0,
                ((aggKey, value, aggregate) ->
                        aggregate + (value.getStudentCount() != null ? value.getStudentCount() : 0)),
                Serdes.Integer(),
                 s: "studentsPerCommune2014");
```

```
studentsPerCommune2014
        .leftJoin(studentsPerCommune2015, this::calculateEvolution)
        .toStream()
        .print(Serdes.String(), Serdes.String(),  s: "Evolution 2014 2015");
```


Ippon Technologies © 2017

Start streaming

```
private static Properties getProperties() {
    Properties settings = new Properties();
    settings.put(StreamsConfig.APPLICATION_ID_CONFIG, STREAM_APPLICATION_NAME);
    settings.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    settings.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    settings.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());

    settings.put(StreamsConfig.PROCESSING_GUARANTEE_CONFIG, StreamsConfig.EXACTLY_ONCE);
    return settings;
}
```

```
StreamsConfig config = new StreamsConfig(getProperties());
KafkaStreams kafkaStreams = new KafkaStreams(kStreamBuilder, config);
kafkaStreams.start();
```


Ippon Technologies © 2017

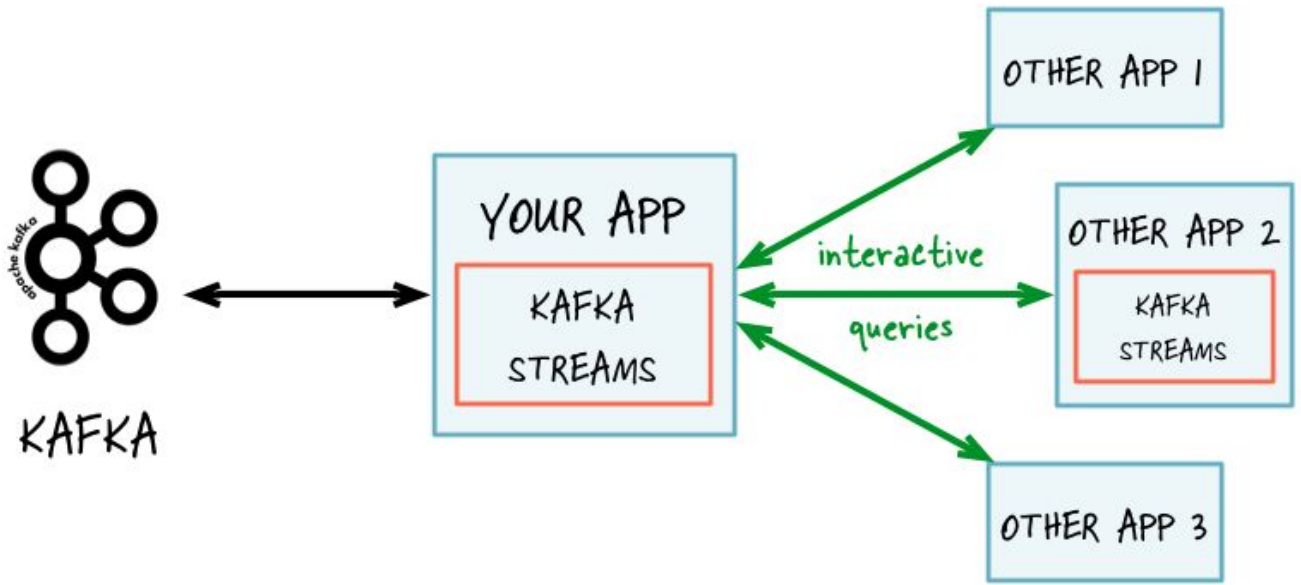
Under the hood, the “state store”

- Store / Query data
- Automatic with count / aggregations / windowing
- Use rocksDB by default
- Global state spread across stream applications
- Also resistant to failure
  - State can be rebuilt from a specific topic
  - Compacted so small footprint

Ippon Technologies © 2017

Bonus : Interactive queries

1. Capture business events in Kafka
2. Process the events with Kafka Streams
3. With interactive queries, other apps can directly query the latest results



Ippon Technologies © 2017

Bonus : KSQL

- Manipulate streaming data with SQL
- Experimental

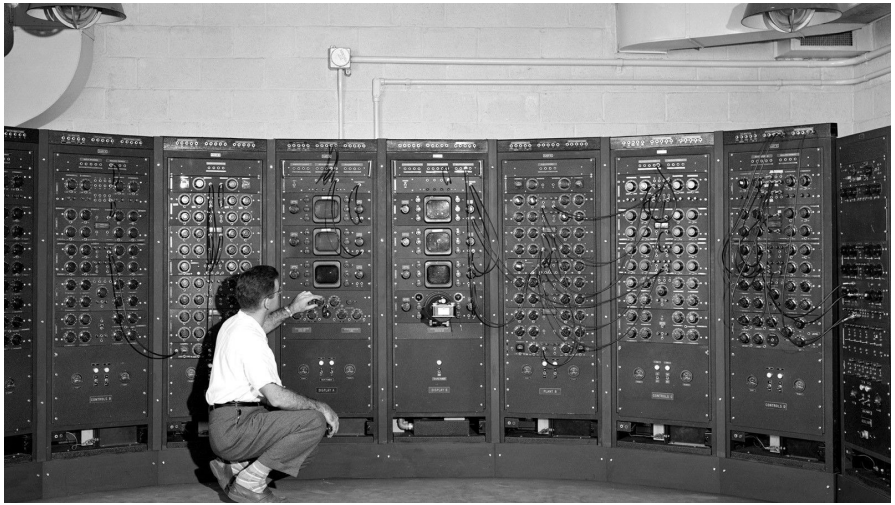
```
CREATE STREAM pageview (viewtime bigint, pageid varchar, userid varchar)
    WITH (value_format = 'json', kafka_topic='pageview_topic_json');

CREATE TABLE users (registertime bigint, userid varchar, regionid varchar, gender varchar)
    WITH (value_format = 'json', kafka_topic='user_topic_json');

-- Find the pageviews from reagion with id ending in _8 and _9 from the female pageview
CREATE STREAM enrichedpv_female_r8 AS
    SELECT *
    FROM enrichedpv_female
    WHERE regionid LIKE '%_8' OR regionid LIKE '%_9';
```


Ippon Technologies © 2017

Kafka from the trenches

Basic setup

- Java 8
- RAM
- Unix systems
- Separate drives for
  - Operating System
  - Kafka logs
  - Zookeeper (or on another cluster)



Ippon Technologies © 2017

Monitoring

- Not trivial
  - Cluster status
  - Topic health / lag
- JMX based
  - Replication
  - Broker status
  - Consumer / producer throughput
- Monitoring for Kafka and Zookeeper

Ippon Technologies © 2017

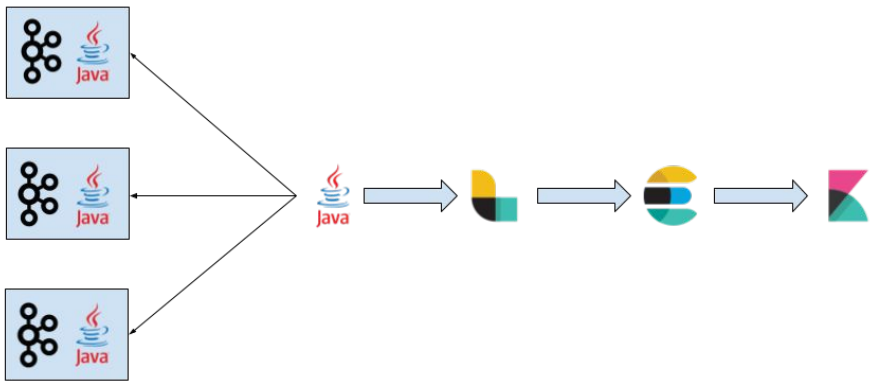
Tools

- **Monitoring:**
  - Confluent Control Center ($)
  - SPM ($)
  - Burrow
- **Management**
  - Kafka Manager
  - Kafkat

Ippon Technologies © 2017

Monitoring : Build your own

- Java monitoring agent on each broker
  - JMX collect and simple transform
  - Expose through HTTP
- Logstash for data ingestion
- Query through Elasticsearch
- Visualize in Kibana



Ippon Technologies © 2017

Conclusion

10 commandments

1. Be kind with Zookeeper
2. Exactly once you will produce
3. Big messages you shall not push
4. Tune producer & consumer settings you will
5. Size your topics
6. Compress if you need to
7. Compaction you will consider
8. Monitor and do not overreact
9. You shall stream
10. Not a silver bullet you shall consider it

Ippon Technologies © 2017

Ressources

- ## Examples
  - ➔ https://github.com/ImFlog/kafka-playground
  - ➔ https://github.com/confluentinc/examples
- ## Documentation
  - ➔ http://docs.confluent.io/
  - ➔ https://kafka.apache.org/
- ## Blogs / articles
  - ➔ https://www.confluent.io/blog
  - ➔ https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
  - ➔ https://cwiki.apache.org/confluence/display/KAFKA/KIP-98+-+Exactly+Once+Delivery+and+Transactional+Messaging
  - ➔ https://zcox.wordpress.com/2017/01/16/updating-materialized-views-and-caches-using-kafka/
  - ➔ https://technology.amis.nl/2017/02/12/apache-kafka-streams-running-top-n-grouped-by-dimension-from-and-to-kafka-topic/

Ippon Technologies © 2017

PARIS
BORDEAUX
NANTES
LYON
MARRAKECH
WASHINGTON DC
NEW-YORK
RICHMOND
MELBOURNE

contact@ippon.fr
www.ippon.fr - www.ippon-hosting.com - www.ippon-digital.fr
@ippontech
-
01 46 12 48 48