TÉCNICO LISBOA

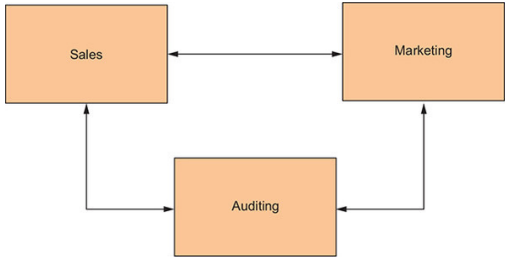
Enterprise Integration

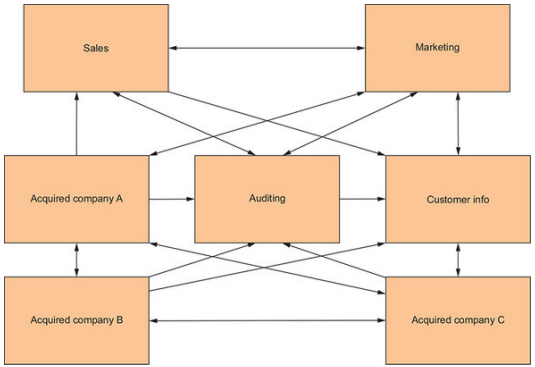
Messaging systems

Prof. Sérgio Guerreiro
Sergio.guerreiro@tecnico.ulisboa.pt

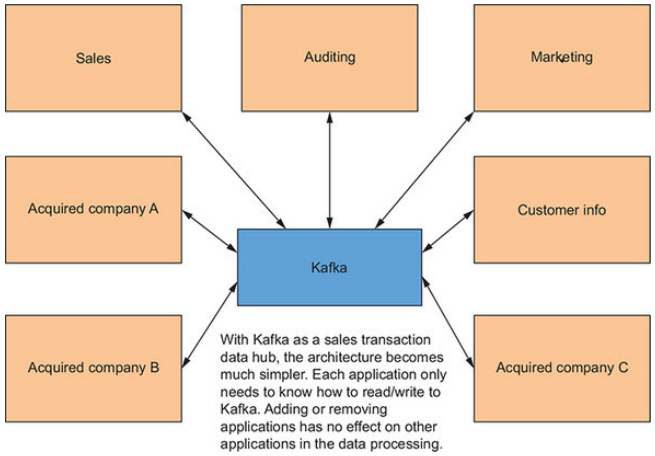
Department of Computer Science and Engineering
Instituto Superior Técnico / Universidade de Lisboa
INESC-ID
URL: http://www.inesc-id.pt
Rua Alves Redol, 9
1000-029 Lisboa
Portugal

The problem, then Publish and Subscribe solution





- The original data architecture for ZMart was simple enough to have information flowing to and from each source of information.
- With more applications being added over time, connecting all these information sources has become complex.
- Using Kafka as a sales transaction hub simplifies the ZMart data architecture significantly. Now each machine doesn't need to know about every other source of information. All they need to know is how to read from and write to Kafka.



Bejeck, B. (2018). *Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API*. Simon and Schuster.

Publish/Subscribe Messaging

- Publish/subscribe (**pub/sub**) messaging is a pattern that is characterized by the sender (**publisher**) of a piece of data (**message**) not specifically directing it to a receiver. Instead, the publisher classifies the message somehow, and that receiver (**subscriber**) subscribes to receive certain classes of messages.
- Pub/sub systems often have a broker, a central point where messages are published, to facilitate this pattern.

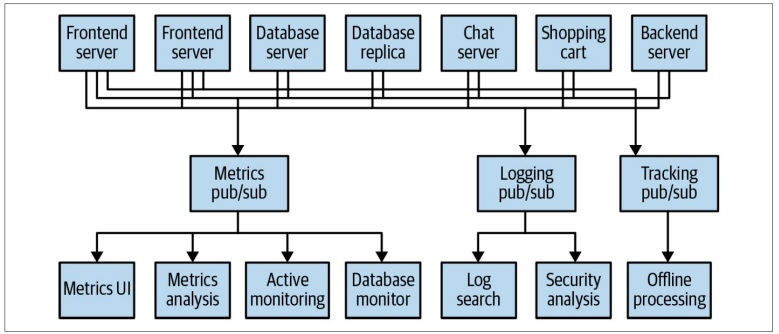


Figure 1-4. Multiple publish/subscribe systems

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Limitations of the synchronous communication

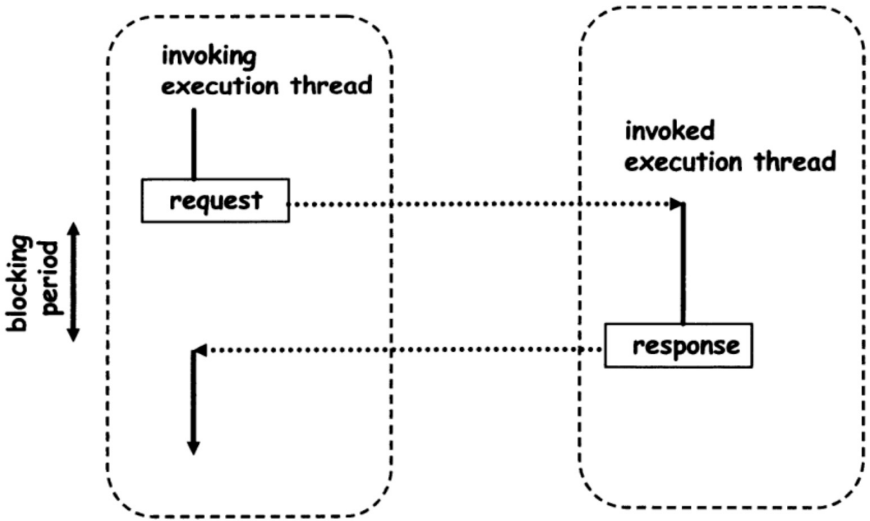


Fig. 1.14. A synchronous call requires the requester to block until the response arrives

- Both applications must be running simultaneously – time coupling
- Blocking period exist
- Fault on communication
- Fault on threads
- Different receiver/production capacity rate => message lose

Alonso, G., Casati, F., Kuno, H., Machiraju, V., Alonso, G., Casati, F., ... & Machiraju, V. (2004). *Web services* (pp. 123-149). Springer Berlin Heidelberg

Asynchronous communication

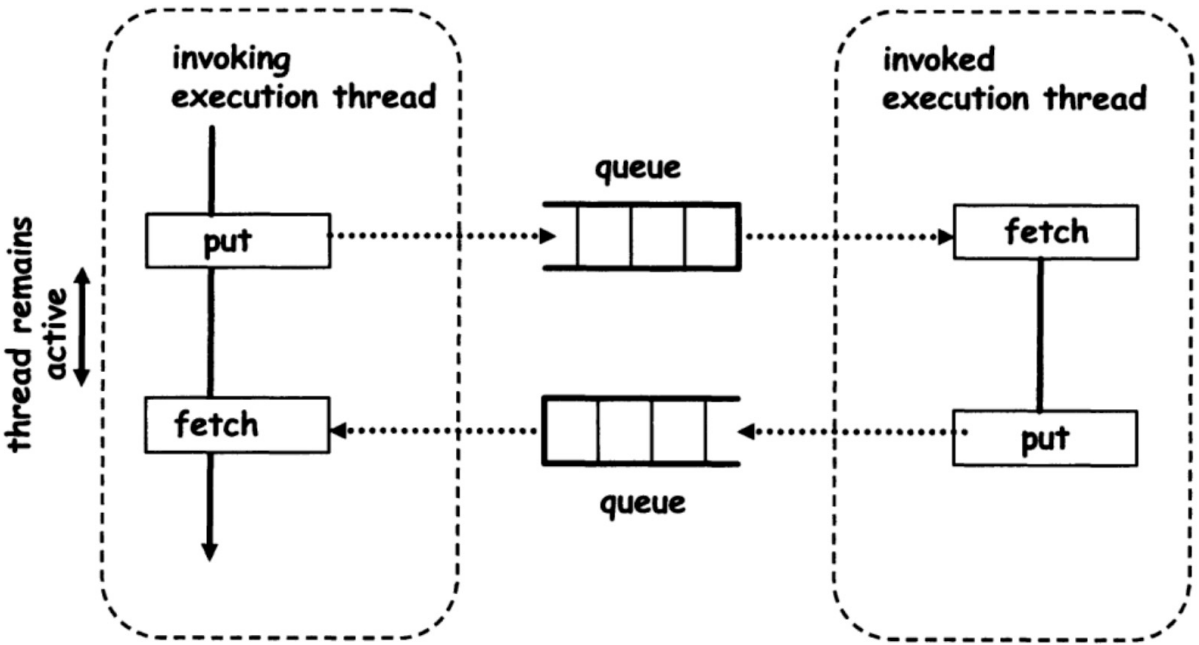
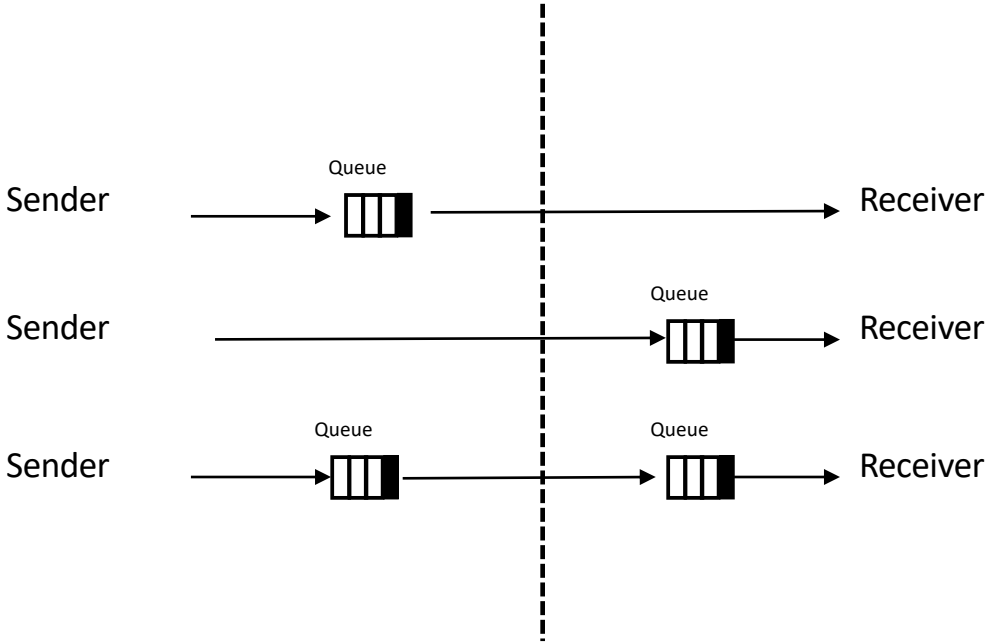


Fig. 1.15. An asynchronous call performed through queues allows the caller to continue working while the request is processed

Alonso, G., Casati, F., Kuno, H., Machiraju, V., Alonso, G., Casati, F., ... & Machiraju, V. (2004). *Web services* (pp. 123-149). Springer Berlin Heidelberg

Solutions for Asynchronous communication with messages queues

Other communication models designs are possible:



Time decoupling *vs.* asynchronous

- Time decoupling and asynchronous communications are different concepts

- Time coupling implies that both the sender and receiver of the information on an integration must be running simultaneously, or the communication fails.
  - Where, the service invocation may be synchronous or asynchronous

- In time decoupling there is no requirement that both are up and running, tolerating transient crash faults of the server

- **Obviously in time decoupling only an asynchronous communication pattern makes sense. Therefore, to have time decoupling, we need some form to persist messages**

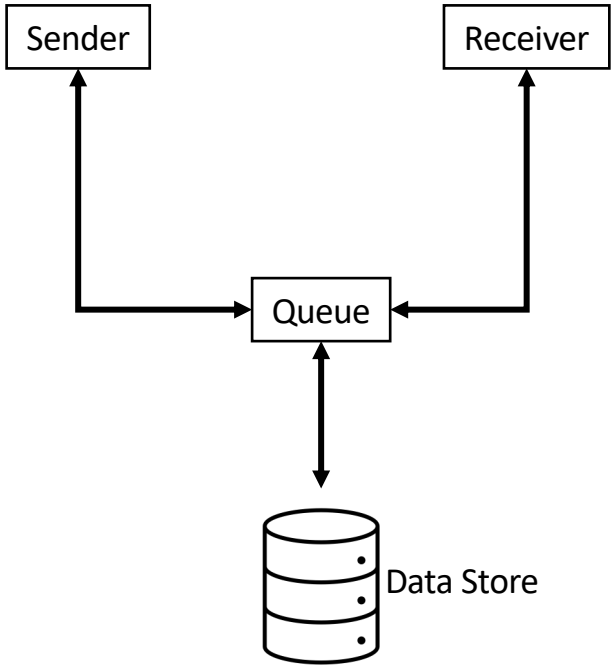
Message Oriented Middleware (MOM)

Advantages

- The capability of persistently storing messages (Store forward):
  - Buffering messages decouples the rate of production and the rate of consumption, unblocking the sender
- Allows to cope with the unavailability of the Service.
- Tolerates temporary crash faults of the service
- Act as a broker to distribute messages accordingly to different routing patterns

Disadvantages

- Asynchronous communication implies a less intuitive programming model (like event programming) than the request-response paradigm
- Message queues need to be supported and managed with additional support and investment costs



MOM key concepts

Channel

- Corresponds to message queues with unique name.
- Guarantees persistence.
- Point-to-point – dedicated channel to deliver unique message
- pub/sub – a copy of a message for each client

MOM key concepts

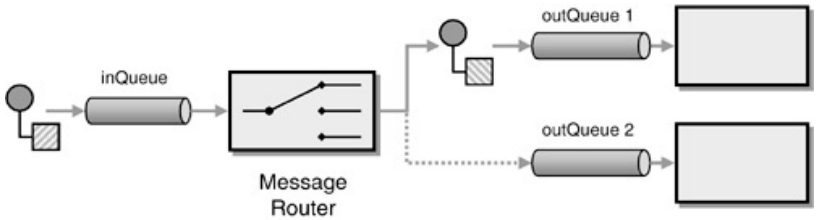
Message

- Package of data that is passed through a channel
- Could be: document, event, image, video, etc…
- To allow interoperability the same schema is required, otherwise message need to be converted

MOM key concepts

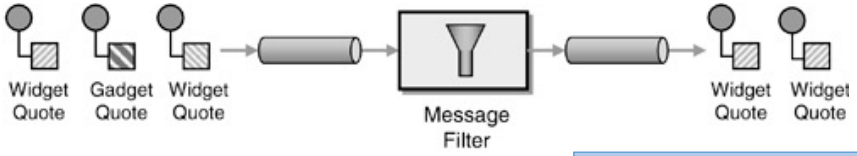
Router

- Is the component in the messaging system that decides the destination of a message
- Different messages may be handled by different applications
- The routing decision is handled in the messaging system



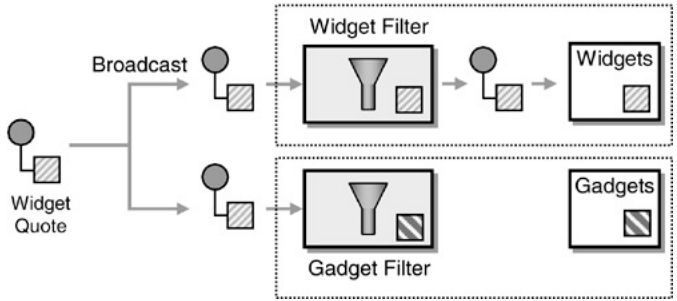
MOM key concepts

Other **routing** mechanisms

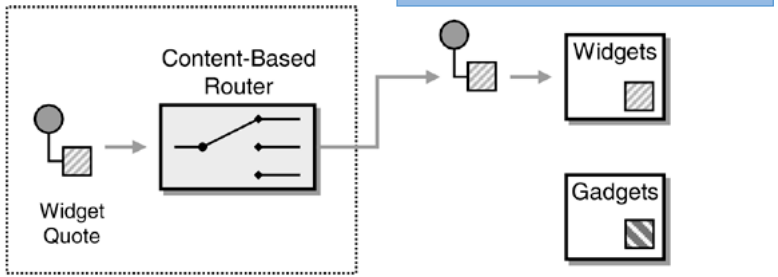


Message-type filter

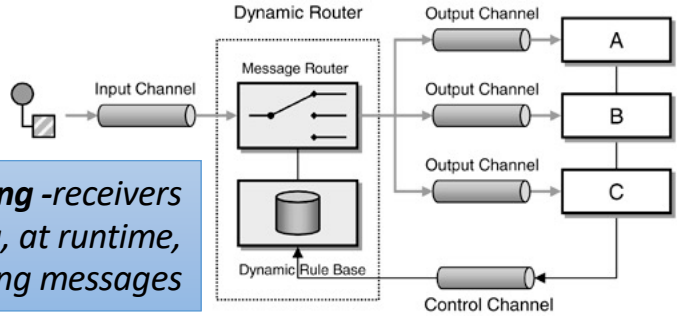
Broadcast routing



Content-based routing



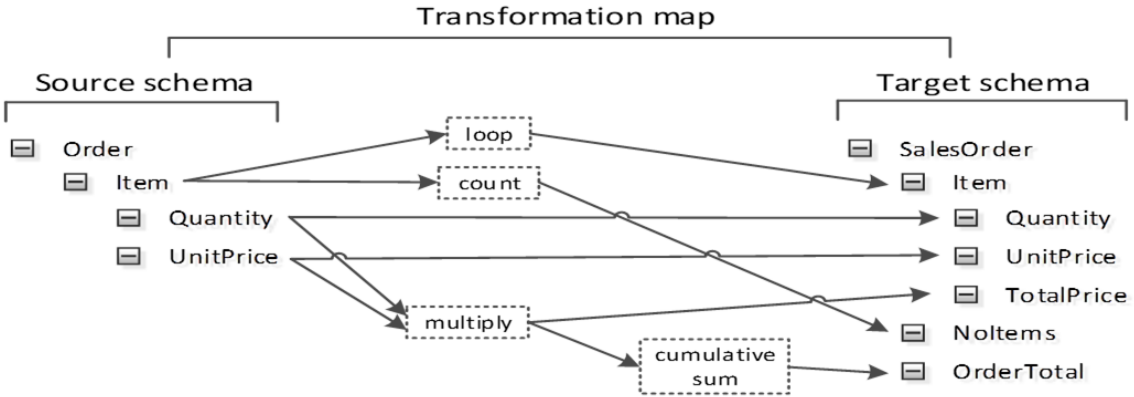
Dynamic routing *-receivers configuring, at runtime, rules for routing messages*



MOM key concepts

Translator provides the ability to convert message content from one structure into another

A transformation map defines the translations required:



MOM key concepts

Endpoint is the software component that abstracts the sender and receiver application from the messaging system.

For sending messages: invoke the send method with the desired mode (*e.g.*, fire-and-forget, synchronous or asynchronous)

For receiving messages:

- **Pooling**: receiver checking in the messaging system if there are any new message available
- **Callback**: receiver is notified by the messaging system when any new message is available

Advanced Message Queuing Protocol (AMQP)

Historically, there was a lack of standards governing the use of message-oriented middleware. Most of the major vendors have their own implementations, each with its own application programming interface (API) and management tools.

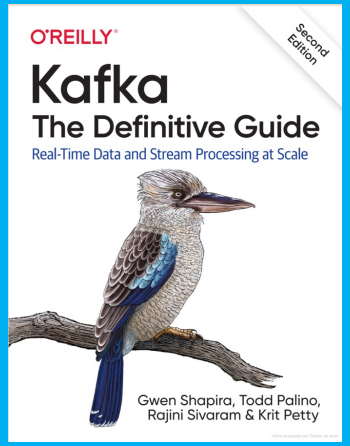
The Advanced Message Queuing Protocol (AMQP) is an approved OASIS and ISO standard that defines the protocol and formats used between participating application components, https://www.amqp.org/

AMQP specifies flexible routing schemes, including point-to-point, fan-out, publish/subscribe, and request-response, transaction management, queuing, distribution, security, management, clustering, federation and heterogeneous multi-platform support.

Example of AMQP implementation: https://www.rabbitmq.com/

TÉCNICO LISBOA

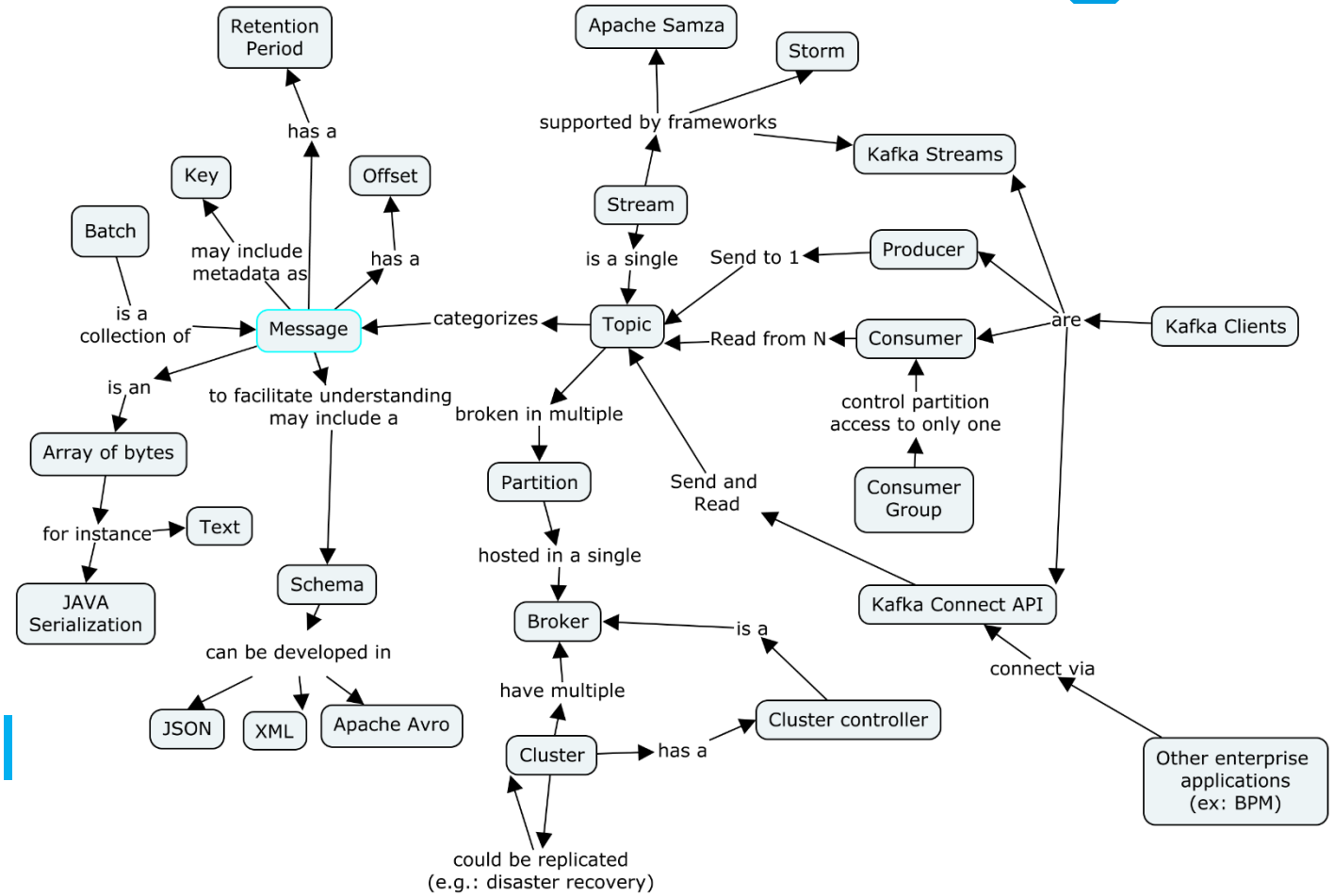
Kafka



Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

TÉCNICO LISBOA

Kafka Conceptual Map



What Does Kafka Do?

Learn about the fundamentals of Kafka, event streaming, and the surrounding ecosystem. Click on an element to find out more.

Event Streaming

Create, import, share streams of events like payments, orders, and database changes in milliseconds, at scale. Store them as long as you need. Replay and reprocess historical data like a time machine.

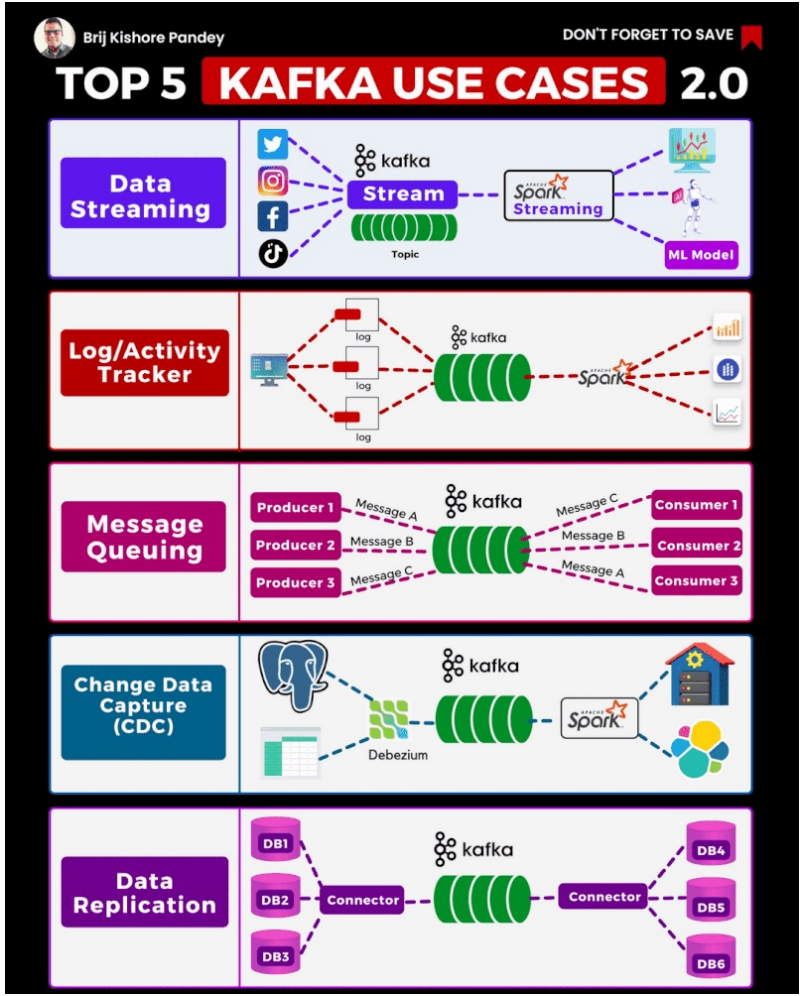
Continuous Real-time Processing

Build lightweight, elastic applications and microservices that respond immediately to events and that scale during live operations. Process, join, and analyze streams and tables of data in real-time, 24x7.

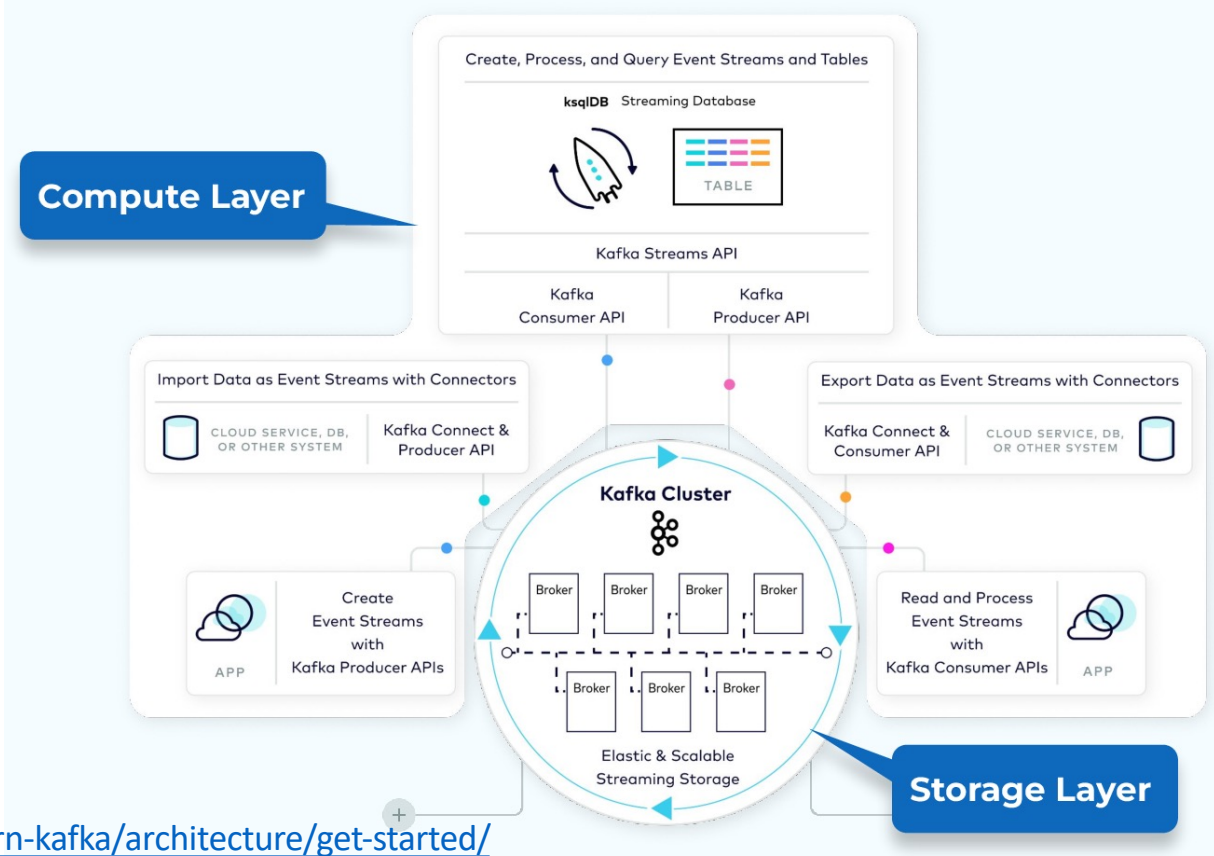
Your Data, Everywhere

Use one platform to set data in motion across your entire enterprise. Connect systems, data centers, and clouds—all with the same trusted technology. The largest production deployments handle trillions of events per day.

https://developer.confluent.io/#kafka-internals


Brij Kishore Pandey
DON'T FORGET TO SAVE
TOP 5 KAFKA USE CASES 2.0
Data Streaming
kafka
Stream
Topic
Spark Streaming
ML Model
Log/Activity Tracker
log
log
log
kafka
Spark
Message Queuing
Producer 1
Producer 2
Producer 3
Message A
Message B
Message C
kafka
Message C
Message B
Message A
Consumer 1
Consumer 2
Consumer 3
Change Data Capture (CDC)
Debezium
kafka
Spark
Data Replication
DB1
DB2
DB3
Connector
kafka
Connector
DB4
DB5
DB6

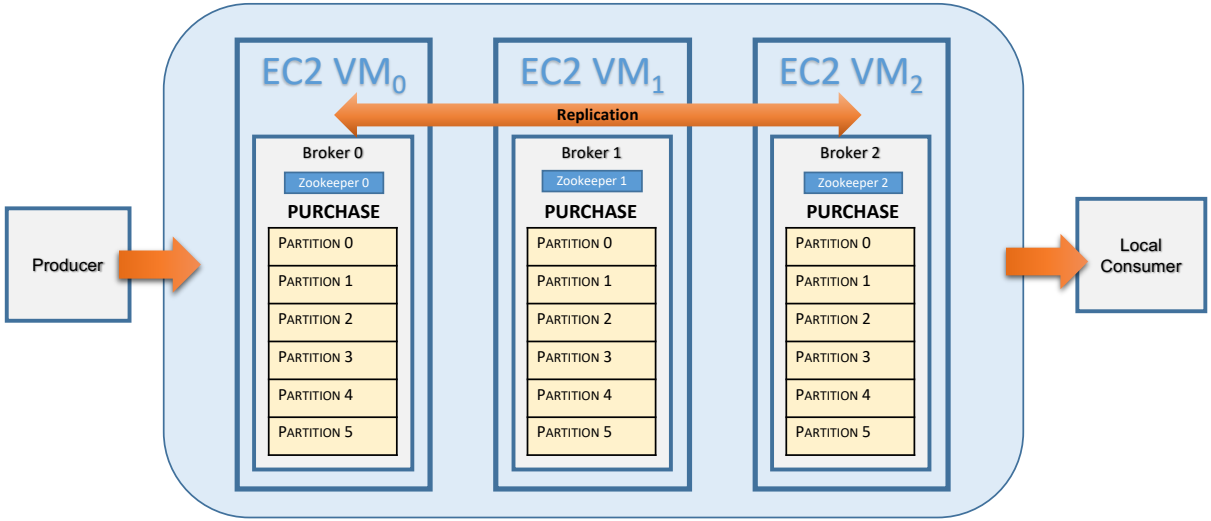
Kafka Architecture



https://developer.confluent.io/learn-kafka/architecture/get-started/

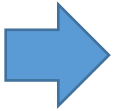
Kafka and Zookeper

- ZooKeeper is reliable storage running on a cluster of nodes
- Kafka uses Apache ZooKeeper to maintain the list of the brokers and all the information related to partitions, offsets, *etc.*
- ZooKeeper can guarantee that the local replicas never diverge



MOM versus Kafka

- MOMs provide:
  - Reliable communication in presence of transient faults on consumers
  - Buffering between producers and consumers
  - Publish/subscription and point to point channels



- Kafka adds with:
  - High ingestion rate of messages
  - Distributes architecture tolerating faults from brokers
  - Disk retention policies

- MOMs normally provide
  - Message transformation
  - Transactional messages
  - Automatic indexes for reading
  - Dynamic routing of messages

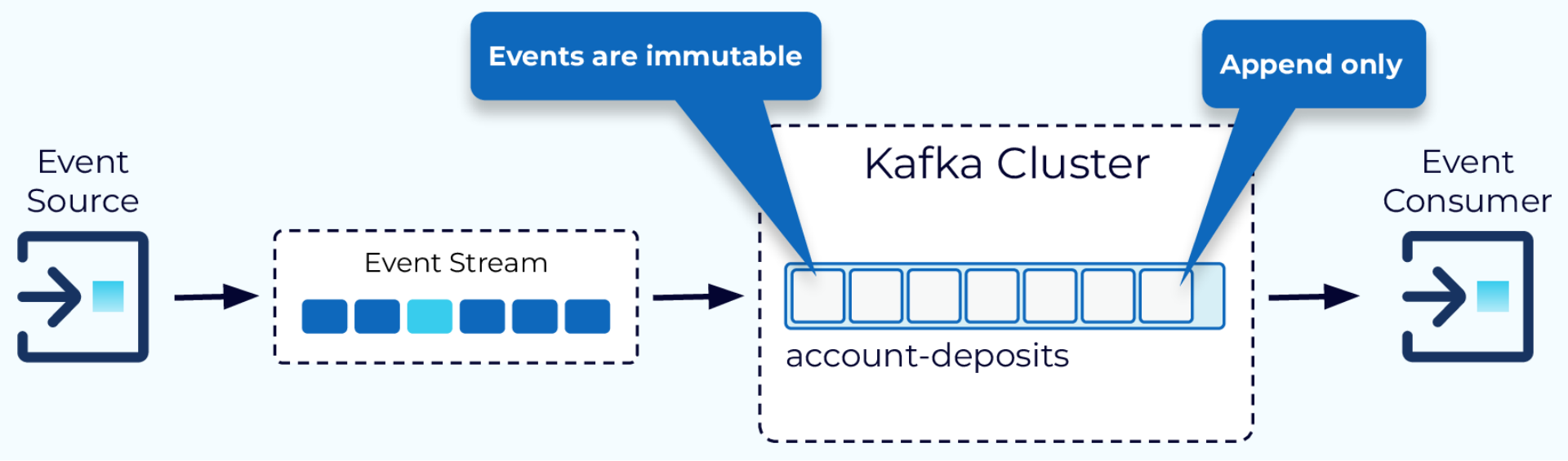
- Kafka does not provide any of these features

Commit log

- …a commit log is designed to provide a **durable record of all transactions** so that they can be replayed to consistently build the state of a system
- Similarly, data within Kafka is stored durably, **in order**, and can be **read deterministically**
- In addition, the data can be **distributed** within the system to provide **additional protections against failures**, as well as significant opportunities for **scaling performance**

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Immutable events

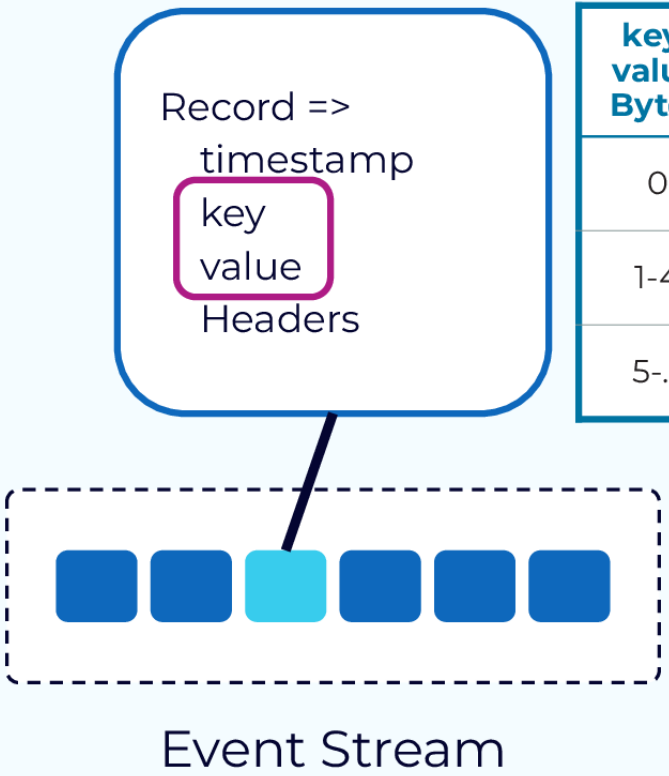


Kafka Message

- …A message is simply an array of bytes as far as Kafka is concerned, so the data contained within it **does not have a specific format or meaning to Kafka**
- If you are approaching Kafka from a database background, you can think of this as similar to a *row* or a *record*
- A message can have an optional piece of metadata, which is referred to as a **key**. The key is also a byte array and, as with the message, has no specific meaning to Kafka
- **Keys are used when messages are to be written to partitions in a more controlled manner**…

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Each Kafka message = <Key,Value>



key/ value Bytes	Area	Description
0	Magic Byte	Confluent serialization format version number; currently always 0.
1-4	Schema ID	4-byte schema ID as returned by Schema Registry.
5-...	Data	Serialized data for the specified schema format.

TÉCNICO LISBOA

Kafka messages in batches

- For efficiency, **messages are written into Kafka in batches**.
- A **batch** is just a collection of messages, all of which are being produced to the **same topic and partition**.
- An individual round trip across the network for each message would result in excessive overhead, and collecting messages together into a batch reduces this.
- Of course, this is a trade-off between latency and throughput: **the larger the batches, the more messages that can be handled per unit of time**, but the longer it takes an individual message to propagate.
- Batches are also typically compressed, providing more efficient data transfer and storage at the cost of some processing power

Kafka Topic and partitions

- Messages in Kafka are categorized into **topics**. The closest analogies for a topic are a database *table* or a *folder* in a filesystem.
- Topics are additionally **broken down into a number of partitions**. Going back to the *“commit log”* description, a partition is a single log.
- Messages are written to it in an append-only fashion and are read in order from beginning to end.
- Note that as a topic typically has multiple partitions, there is **no guarantee of message ordering across the entire topic, just within a single partition**

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Topic and partitions

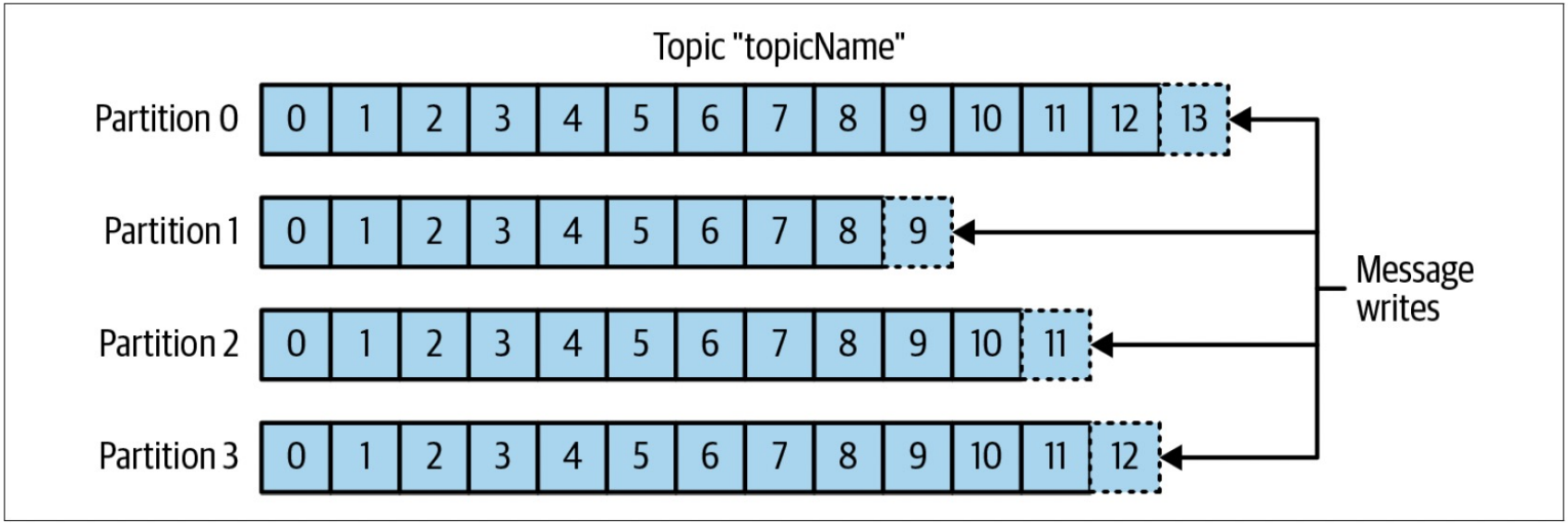


Figure 1-5. Representation of a topic with multiple partitions

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Log directory

Figure 2.6. The logs directory is the base storage for messages. Each directory under /logs represents a topic partition. Filenames within the directory start with the name of the topic, followed by an underscore, which is followed by a partition number.

The logs directory is configured in the root at /logs.

/logs

/logs/topicA_0	topicA has one partition.
/logs/topicB_0	topicB has three partitions.
/logs/topicB_1	
/logs/topicB_2	

Bejeck, B. (2018). Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API. Simon and Schuster.

Listing the count of partitions for a specific topic

TÉCNICO LISBOA

/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server *<YourIP_or_DNS>*:9092 --topic clicks

```
Topic:clicks    PartitionCount:8        ReplicationFactor:1     Configs:
        Topic: clicks   Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 2    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 3    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 4    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 5    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 6    Leader: 0       Replicas: 0     Isr: 0
        Topic: clicks   Partition: 7    Leader: 0       Replicas: 0     Isr: 0
```

Partition Number = hash(key) % #Partitions(topic)

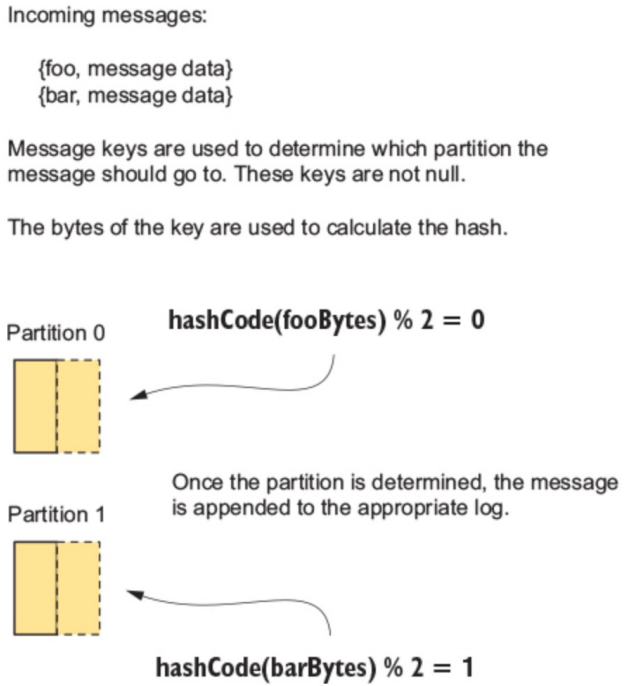
- **Keys are used when messages are to be written to partitions in a more controlled manner….**
- The simplest such scheme is to generate a consistent hash of the key and then select the partition number for that message by taking the result of the hash modulo the total number of partitions in the topic.
- This ensures that messages with the same key are always written to the same partition (*provided that the partition count does not change*)

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Partition Number = hash(key) % #Partitions(topic)

TÉCNICO LISBOA

Figure 2.8. "foo" is sent to partition 0, and "bar" is sent to partition 1. You obtain the partition by hashing the bytes of the key, modulus the number of partitions.



Bejeck, B. (2018). Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API. Simon and Schuster.

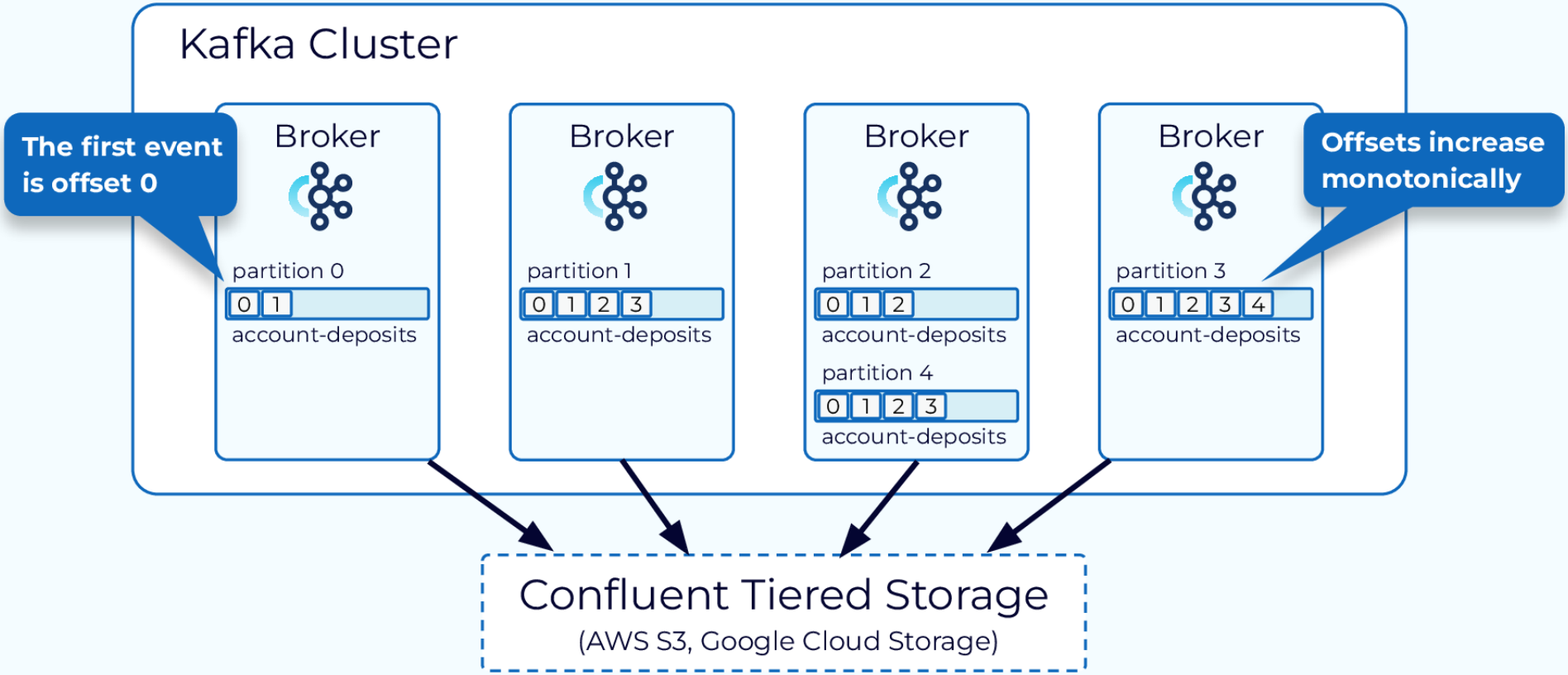
TÉCNICO LISBOA

Message Offset

- The offset—an integer value that continually increases—is another piece of metadata that Kafka adds to each message as it is produced

- **Each message in a given partition has a unique offset**, and the following message has a greater offset (though not necessarily monotonically greater)

Message offset increasing monotonically



Kafka Producer

- Producers create **new messages**
- **A message will be produced to a specific topic**. By default, the producer will **balance** messages over all partitions of **a topic evenly**. In some cases, the producer will direct messages to specific partitions. This is typically done using the message key and a partitioner that will generate a hash of the key and map it to a specific partition
- This ensures that all messages produced with a given key will get written to the same partition. *The producer could also use a custom partitioner that follows other business rules for mapping messages to partitions*

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Producer sending message to the broker

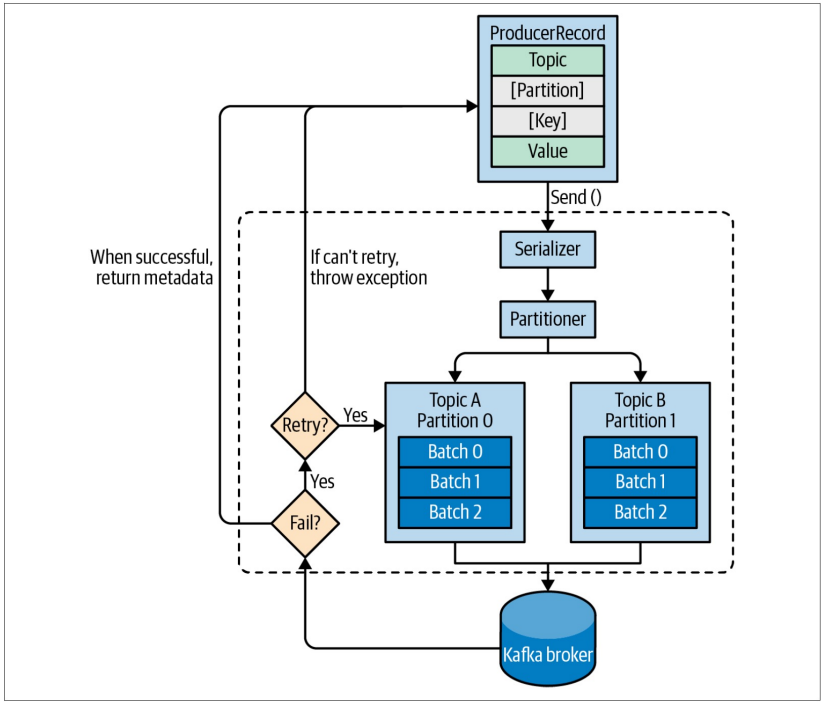


Figure 3-1. High-level overview of Kafka producer components

Sending policies

- **Fire-and-forget**

Sending a message to the server and do not really care if it arrives successfully or not. Most of the time, it will arrive successfully, since Kafka is highly available, and the producer will retry automatically. However, some messages will get lost using this method.

- **Synchronous send**

Sending a message and testing the acknowledge: the `send()` method returns a future object, and the sender can call `get()` to wait on the future and see if send was successful.

- **Asynchronous send**

Calling the `send()` method with a callback function, which gets triggered when it receives a response from the Kafka broker. Whenever the synchronous response take too long, or no error processing need to be done.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

TÉCNICO LISBOA

JAVA Source code for fire & forget sending policy

```java
import java.util.Properties;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.LongSerializer;
import org.apache.kafka.common.serialization.StringSerializer;
public class ClientProducerKafka {
    Run | Debug
    public static void main(String[] args) {

        Properties kafkaProps = new Properties();
        kafkaProps.put(key: "bootstrap.servers", value: "YOURAWSIP:9092");
        kafkaProps.put(key: "key.serializer",value: "org.apache.kafka.common.serialization.StringSerializer");
        kafkaProps.put(key: "value.serializer",value: "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(kafkaProps);

        //Fire-and-forget
        System.out.println(x: "Fire-and-forget starting...");
        ProducerRecord<String, String> record =
                new ProducerRecord<>("NAMEOFTOPIC", "KEY", "MESSAGE");
        try {  producer.send(record); }
        catch (Exception e) {   e.printStackTrace();}
        System.out.println(x: "Fire-and-forget stopped.");
    }
}
```

TÉCNICO LISBOA

```java
import java.util.Properties;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.LongSerializer;
import org.apache.kafka.common.serialization.StringSerializer;
public class ClientProducerKafka {
    Run | Debug
    public static void main(String[] args) {

        Properties kafkaProps = new Properties();
        kafkaProps.put(key: "bootstrap.servers", value: "YOURAWSIP:9092");
        kafkaProps.put(key: "key.serializer",value: "org.apache.kafka.common.serialization.StringSerializer");
        kafkaProps.put(key: "value.serializer",value: "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(kafkaProps);

        //Synchronous send
        System.out.println(x: "Synchronous send starting...");
        ProducerRecord<String, String> record2 =
                    new ProducerRecord<>("NAMEOFTOPIC", "KEY", "MESSAGE2");
        try {   System.out.println("Result get =" + producer.send(record2).get());  }
              catch (Exception e) {e.printStackTrace();    }
        System.out.println(x: "Synchronous stopped.");

    }
}
```

JAVA Source code for synchronous sending policy

```
private class DemoProducerCallback implements Callback { ❶
    @Override
    public void onCompletion(RecordMetadata recordMetadata, Exception e) {
        if (e != null) {
            e.printStackTrace(); ❷
        }
    }
}

ProducerRecord<String, String> record =
    new ProducerRecord<>("CustomerCountry", "Biomedical Materials", "USA"); ❸
producer.send(record, new DemoProducerCallback()); ❹
```

❶ To use callbacks, you need a class that implements the `org.apache.kafka.clients.producer.Callback` interface, which has a single function—`onCompletion()`.

❷ If Kafka returned an error, `onCompletion()` will have a nonnull exception. Here we "handle" it by printing, but production code will probably have more robust error handling functions.

❸ The records are the same as before.

❹ And we pass a `Callback` object along when sending the record.

JAVA
Source code
for
asynchronous
sending
policy

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Consumer

- Consumers **read messages**
- The consumer **subscribes to one or more topics** and reads the messages in the **order** in which they were produced **to each partition**.
- The consumer keeps track of which messages it has already consumed by **keeping track of the offset of messages**
- By storing the next possible offset for each partition a consumer can stop and restart without losing its place.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Consumer Group

- **Consumers** work as **part of a consumer group**, which is one or more consumers that work together to consume a topic. The group ensures that **each partition is only consumed by one member**.
- In this way, consumers can **horizontally scale** to consume topics with a large number of messages. Additionally, **if a single consumer fails**, the remaining members of the group will **reassign the partitions being consumed** to take over for the missing member: **Rebalance**

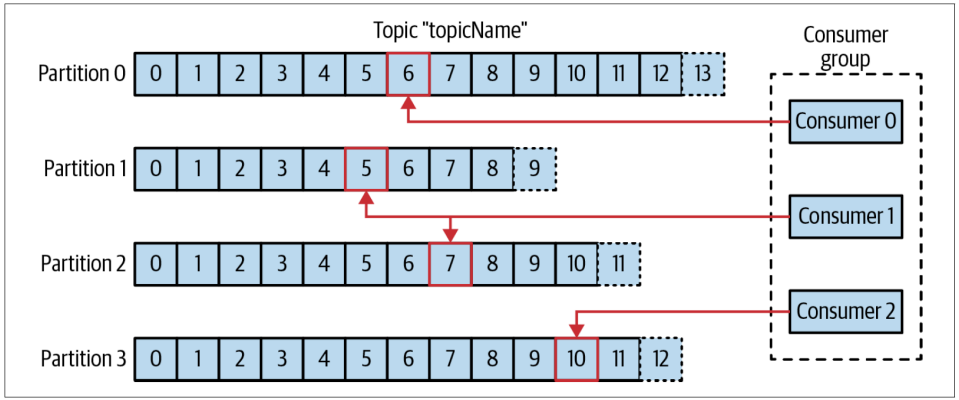


Figure 1-6. A consumer group reading from a topic

Here are three consumers in a single group consuming a topic. Two of the consumers are working from one partition each, while the third consumer is working from two partitions.

The mapping of a consumer to a partition is often called *ownership* of the partition by the consumer.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

On Kafka Consumer Group rebalance

- On rebalancing, Kafka needs to know the offset where the actual consumers were reading

- How to communicate the offset to Kafka?
  - Automatic commit:

    ```
    enable.auto.commit=true;
    ```

    then every five seconds the consumer will commit the largest offset your client receive from poll().

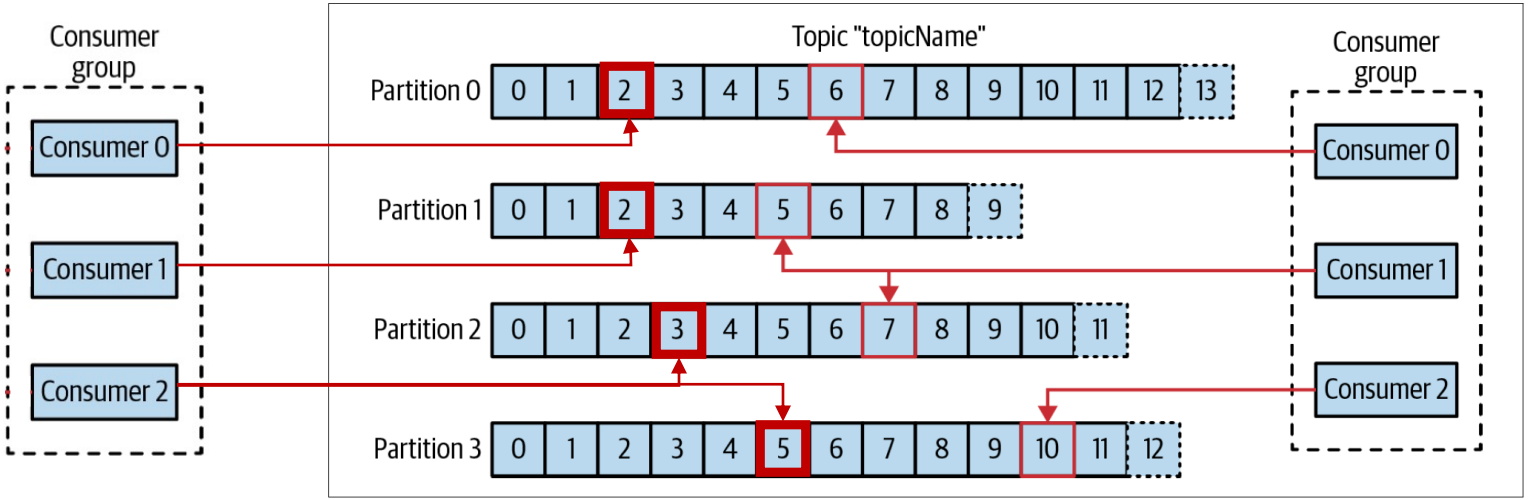
  - Or, explicitly Commit current Offset

```
try
{
      consumer.commitSync();
}
catch (CommitFailedException e){log.error("commit failed",e)}
```

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Consumer Group

- Each consumer group have its **own offset**
- Consumer groups remember the offsets where they left off
- *Example, 2 consumer groups for the same topic with different offsets. Both consumer groups are consuming the same messages:*



Listing the consumer-ids from a specific consumer group

sudo /usr/local/kafka/bin/kafka-consumer-groups.sh --bootstrap-server *<YourIP_or_DNS>*:9092 -describe -group console-consumer-28356

```
TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID                                     HOST            CLIENT-ID
clicks          1          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          4          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          7          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          0          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          5          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          2          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          3          -               1               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
clicks          6          -               0               -               consumer-1-4924ed20-7fc8-46e4-909a-35c71d5a79fc /194.210.61.144 consumer-1
```

TÉCNICO LISBOA

JAVA Source code for consuming from a topic

Polling mechanism

```java
import java.util.Collections;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
public class ClientConsumerKafka {
    Run | Debug
    public static void main(String[] args) {
        Properties props = new Properties();
        props.put(key: "bootstrap.servers", value: "yourAWSIP:9092");
        props.put(key: "group.id", value: "CountryCounter");
        props.put(key: "key.deserializer", value: "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(key: "value.deserializer", value: "org.apache.kafka.common.serialization.StringDeserializer");
        KafkaConsumer < String, String > consumer = new KafkaConsumer < String, String > (props);
        consumer.subscribe(Collections.singletonList(o: "NAMEOFTOPIC"));
        try {
            while (true) {
                /*  consumers must keep polling Kafka or they will be considered
                    dead and the partitions they are consuming will be handed to another
                    consumer in the group to continue consuming. The parameter we pass, poll(),
                    is a timeout interval and controls how long poll() will block if data is not available
                    in the consumer buffer. If this is set to 0, poll() will return immediately;
                    otherwise, it will wait for the specified number of milliseconds for data to arrive
                    from the broker. */
                ConsumerRecords < String, String > records = consumer.poll(100);
                for (ConsumerRecord < String, String > record: records) {
                    System.out.printf(format: "topic = %s, partition = %s, offset = %d,customer = %s, country = %s\n",
                                      record.topic(), record.partition(), record.offset(), record.key(), record.value());
                }
            }
        } finally {
            consumer.close();
        }
    }
}
```

On Kafka Partitions and Consumer Groups

- From a performance point of view, it is the number of **partitions** that matters.
- But since each topic in Kafka has at least one partition, if you have N topics, you inevitably have at least N partitions
- However, keep in mind that, partitions have costs (latency, more memory, more file descriptors…)
- Create a **new consumer group** for each application that needs all the messages from **one or more topics**
- Add consumers to an existing consumer group to scale reading and processing messages from topics. Obviously, it is **bounded by the number of partitions in the topic**
- The reads from a single partition maintain the order in which the messages were produced but within a **multiple partition Topic global order is not guaranteed**

Kafka Broker

- A single Kafka server is called a **broker**.
- The broker receives messages from producers, assigns offsets to them, and writes the messages to storage on disk.
- It also services consumers, responding to fetch requests for partitions and responding with the messages that have been published.
- Depending on the specific hardware and its performance characteristics, a single broker can easily **handle thousands of partitions and millions of messages per second**.
- A broker can define a **retention period** for data which can be configured by each Topic: a *given period* and a *given size*

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

A simple Kafka Cluster

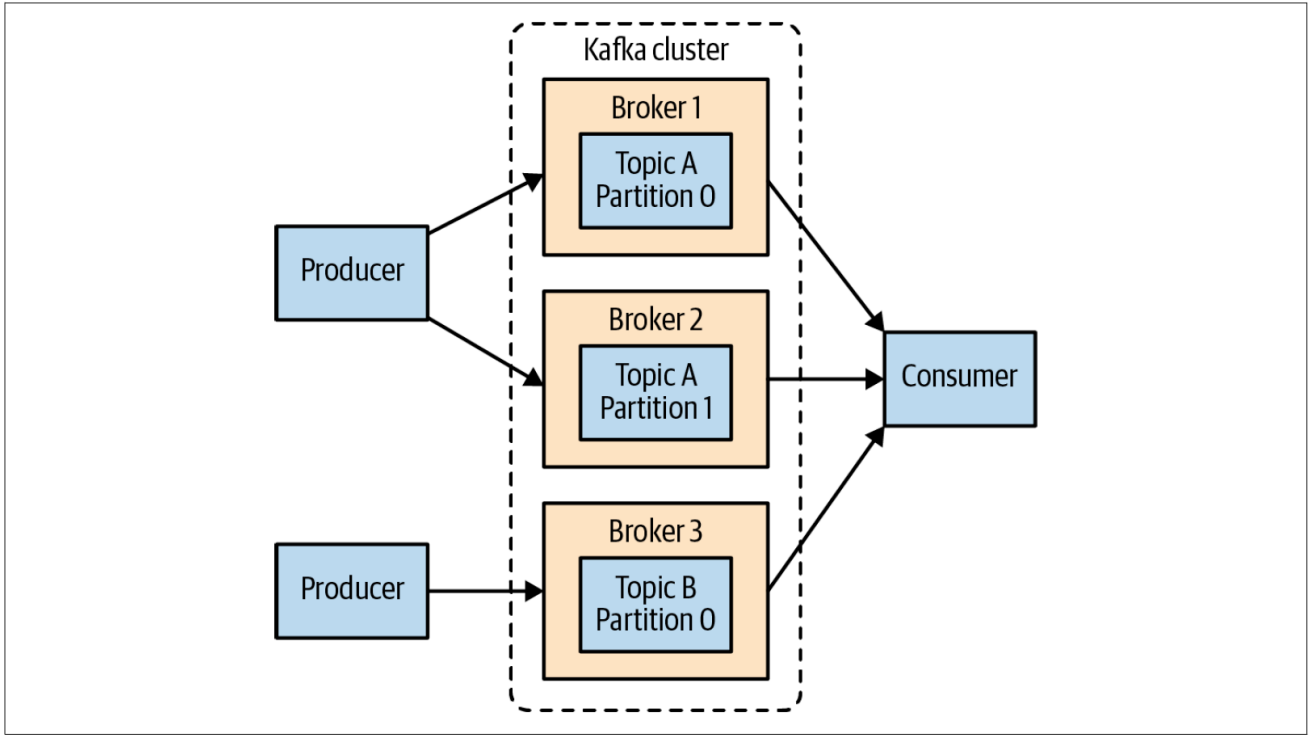


Figure 2-2. A simple Kafka cluster

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Cluster

- Kafka brokers are designed to operate as part of a cluster.
- Within a cluster of brokers, one broker will also function as the **cluster controller** (**elected** automatically from the live members of the cluster).
- The controller is responsible for administrative operations, including assigning partitions to brokers and monitoring for broker failures.
- A partition is owned by a single broker in the cluster, and that broker is called the **leader of the partition**. A replicated partition is assigned to additional brokers, called **followers** of the partition (*next figure).*
- Replication provides **redundancy** of messages in the partition, such that one of the followers can take over leadership if there is a broker failure. All producers must connect to the leader in order to publish messages, but consumers may fetch from either the leader or one of the followers.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Replication of partitions in a cluster

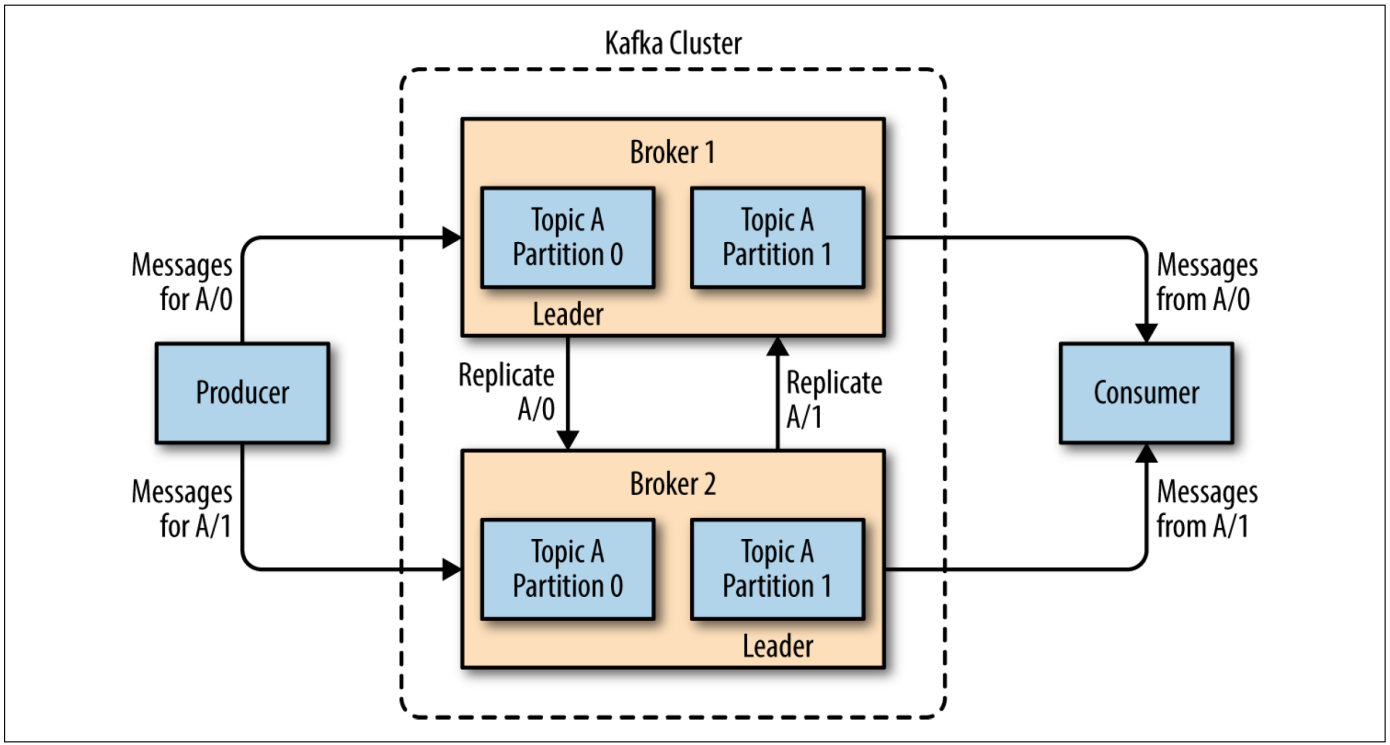
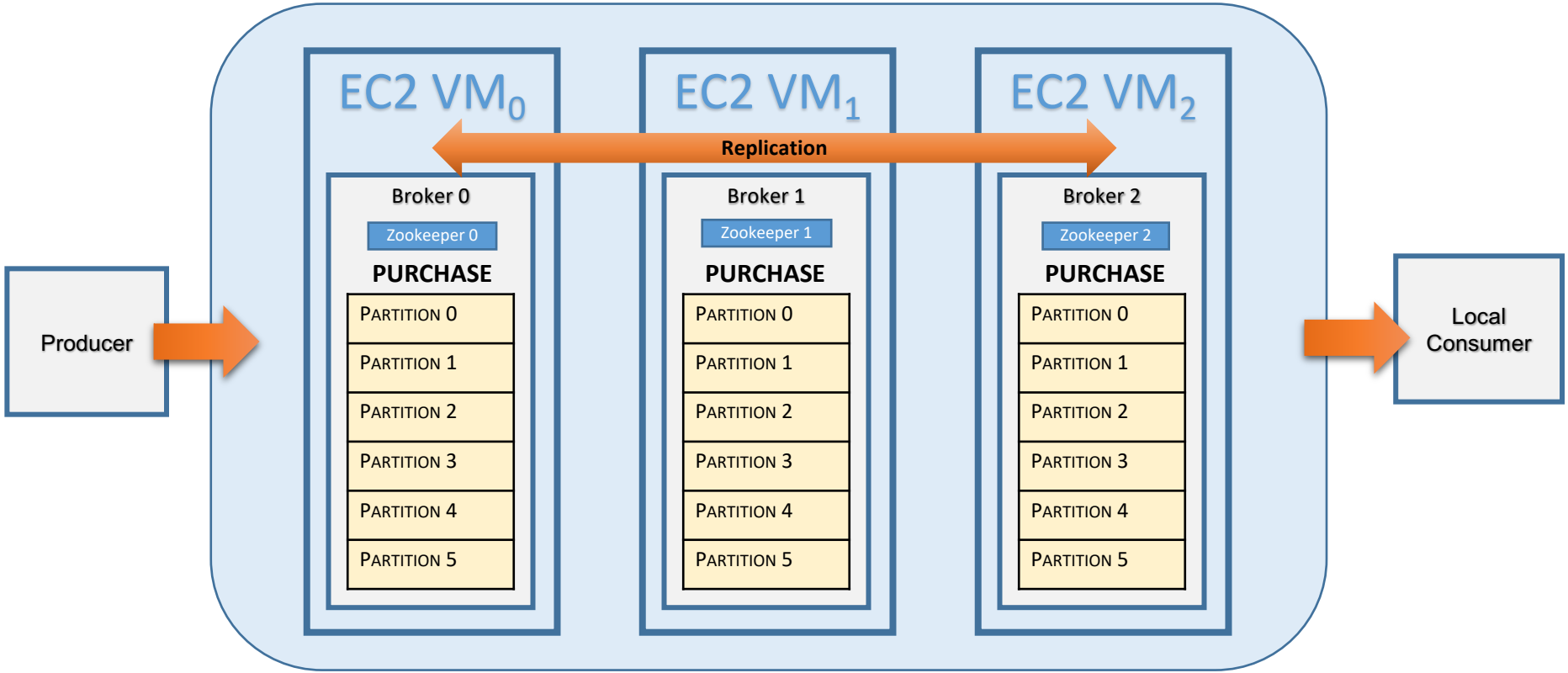


Figure 1-7. Replication of partitions in a cluster

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

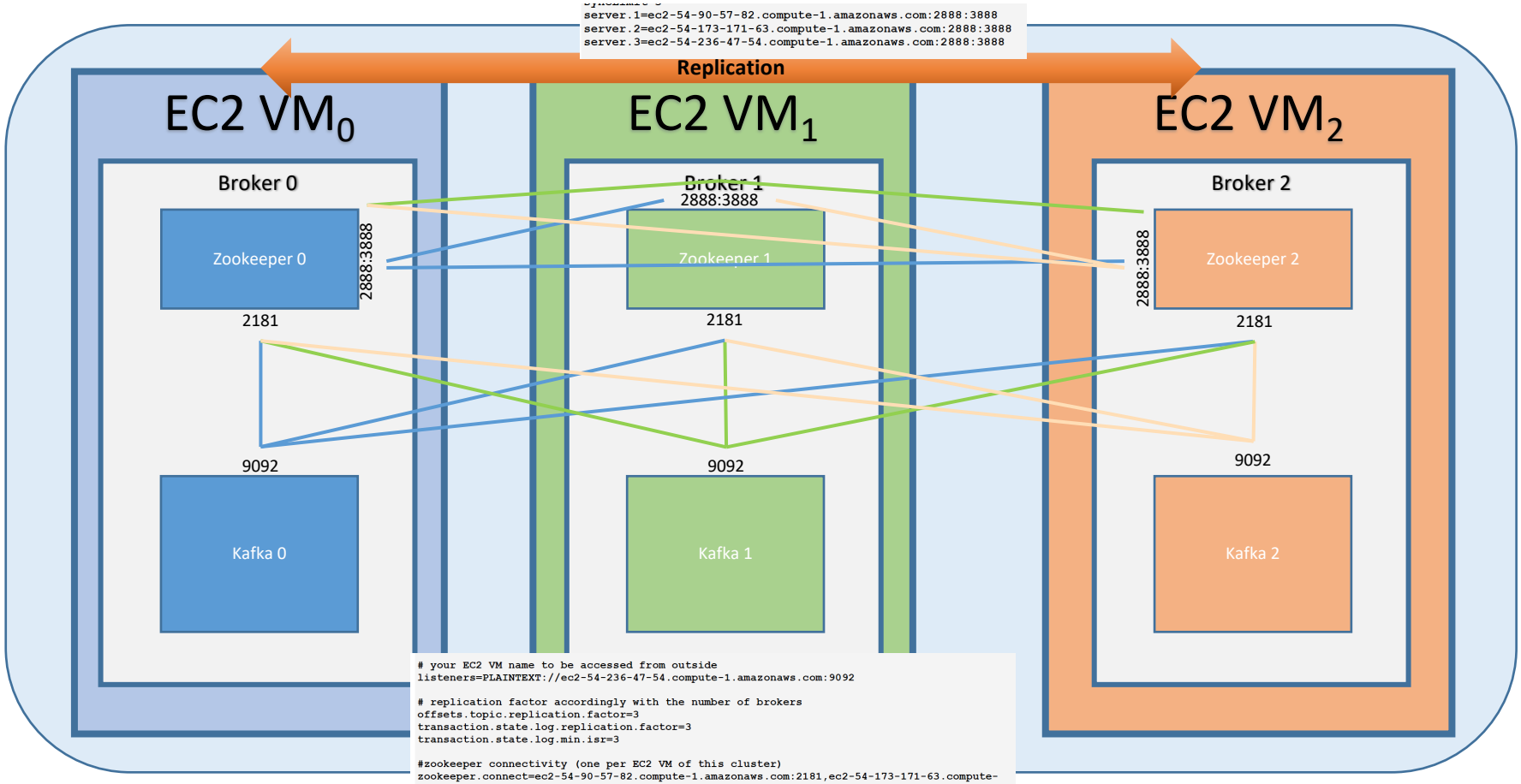
Kafka Cluster deployment with 3 brokers

TÉCNICO LISBOA



Implementation of a Kafka Cluster

TÉCNICO LISBOA



TÉCNICO LISBOA

SE

in kafka is it possible to consume a message that is not propagated to all brokers?

In Apache Kafka, a message is typically written to a topic and then replicated across multiple brokers for fault tolerance and high availability. By default, a message needs to be written to all in-sync replicas (ISRs) of a partition before it can be considered committed and available for consumption.

However, it is possible to configure Kafka to allow consumers to read messages that have not been fully replicated to all brokers. This is called the "unclean leader election" mode, which is disabled by default for data consistency reasons.

When unclean leader election is enabled, Kafka may allow a replica that is not fully up-to-date to become the leader of a partition. In this case, the leader may serve messages that have not yet been replicated to all brokers, and consumers may be able to consume these messages from the leader before they are propagated to all replicas.

It's worth noting that enabling unclean leader election can compromise data consistency and increase the risk of data loss, so it should be used with caution and only in certain scenarios where data consistency can be relaxed.

Broker failure test

Starting a cluster with 2 brokers

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server <YourIP_or_DNS>:9092 --topic events

Topic:events    PartitionCount:4        ReplicationFactor:2     Configs:
        Topic: events   Partition: 0    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: events   Partition: 1    Leader: 1       Replicas: 1,0   Isr: 1,0
        Topic: events   Partition: 2    Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: events   Partition: 3    Leader: 1       Replicas: 1,0   Isr: 1,0
```

in-sync replicas (ISRs)

Stopping broker 0

```
/usr/local/kafka/bin/kafka-topics.sh --describe --bootstrap-server <YourIP_or_DNS>:9092 --topic events

Topic:events    PartitionCount:4        ReplicationFactor:2     Configs:
        Topic: events   Partition: 0    Leader: 1       Replicas: 0,1   Isr: 1
        Topic: events   Partition: 1    Leader: 1       Replicas: 1,0   Isr: 1
        Topic: events   Partition: 2    Leader: 1       Replicas: 0,1   Isr: 1
        Topic: events   Partition: 3    Leader: 1       Replicas: 1,0   Isr: 1
```

TÉCNICO LISBOA

Best size for Kafka Cluster?

- **Criteria 1**: Data storage available on each kafka broker versus overall storage need?
  - E.g.: retain 10TB and each broker store 2TB = cluster with 5 brokers

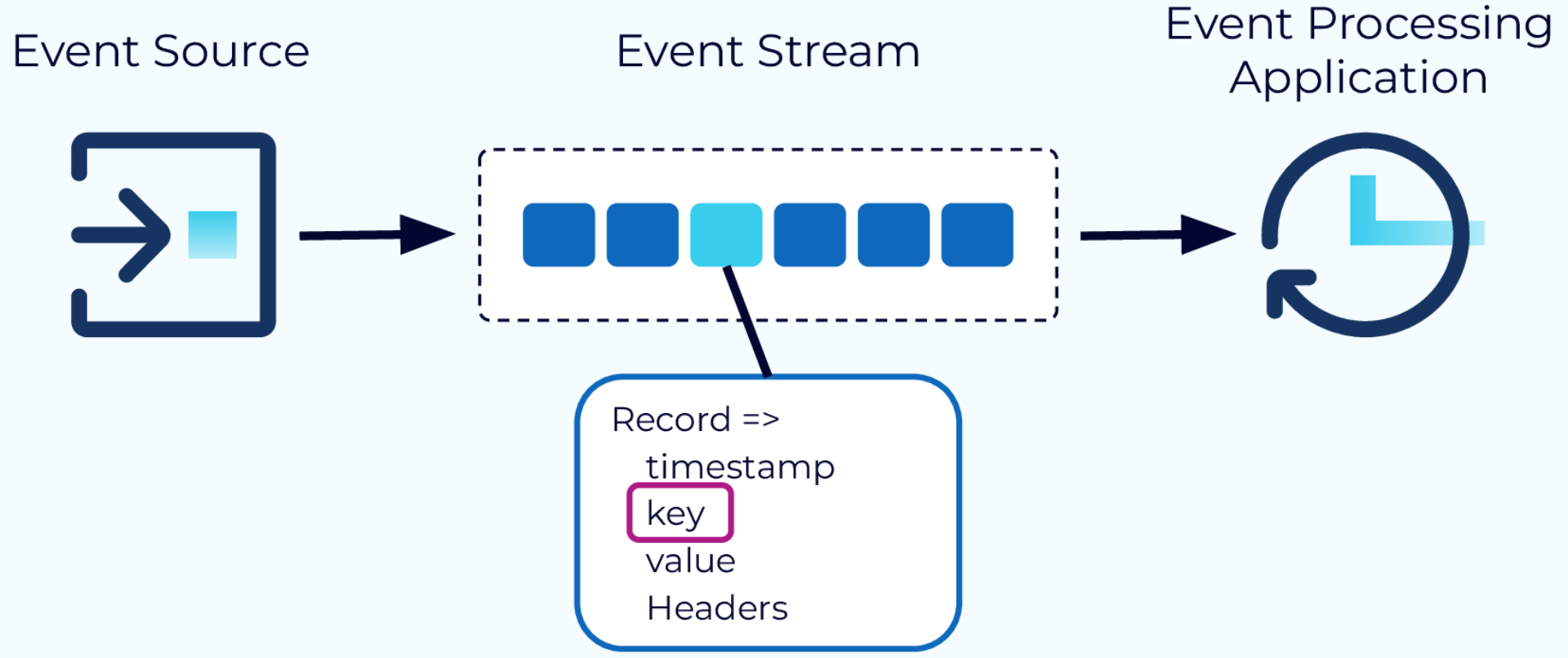
- **Criteria 2**: Handling requests capacity?
  - E.g.: what is the capacity of the network interfaces, and can they handle the client traffic if there are multiple consumers of the data or if the traffic is not consistent over the retention period of the data. Consider the bursts of traffic during peak times

TÉCNICO LISBOA

How many Kafka partions?

- Producers are typically much faster than consumers, then always calculate for consumers' expectations
- What is the maximum throughput expected to achieve when consuming from a single partition?
  - Considering, always one consumer reading from a partition, so if you know that your slower consumer writes the data to a database that never handles more than 50 MB per second from each thread writing to it, then you know you are **limited to 50MB throughput** when consuming from a partition
- If you are sending messages to partitions based on keys, adding partitions later **can be very challenging**, so calculate throughput based on your expected future usage, not the current usage
- Avoid overestimating, as each partition uses memory and other resources on the broker and will increase the time for leader elections

Event Stream Processing



Kafka Streams

- What is a data stream (also called an event stream or streaming data)?
- First and foremost, a data stream is an abstraction representing an **unbounded dataset**.
- Unbounded means infinite and ever growing. The dataset is unbounded because over time, **new records keep arriving**

- Note that this simple model (a stream of events) can be used to represent just about every business activity we care to analyze. We can look at a stream of credit card transactions, stock trades, package deliveries, network events going through a switch, events reported by sensors in manufacturing equipment, emails sent, moves in a game, etc. The list of examples is endless because pretty much everything can be seen as a sequence of events.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Streams properties

- **Unbounded**
- Event streams are **ordered**
- **Immutable** data records
- Event streams are **replayable** - to correct errors, try new methods of analysis, or perform audits

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Kafka Streams processing

- Continuous and nonblocking: Stream processing fills the gap between the request-response world, where we wait for events that take two milliseconds to process, and the batch processing world, where data is processed once a day and takes eight hours to complete.
- Most business processes don’t require an immediate response within milliseconds but can’t wait for the next day either. Most business processes happen continuously, and as long as the business reports are updated continuously and the line of business apps can continuously respond, the processing can proceed without anyone waiting for a specific response within milliseconds.
- Business processes such as alerting on suspicious credit transactions or network activity, adjusting prices in real-time based on supply and demand, or tracking deliveries of packages are all a natural fit for continuous but nonblocking processing.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

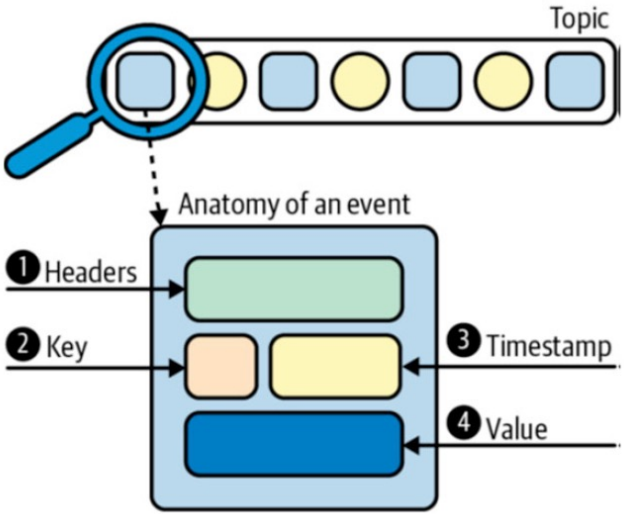
Topology

- A stream processing application includes **one or more processing topologies**.
- A processing topology starts with **one or more source streams** that are passed through a graph of stream processors connected through event streams, until results are written to one or more sink streams.
- Each stream processor is a computational step applied to the stream of events in order to transform the events.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Time

- **Event time** - This is the time the events we are tracking occurred and the record was created
- **Log append time** - This is the time the event arrived at the Kafka broker and was stored there, also called ingestion time
- **Processing time** - This is the time at which a stream processing application received the event in order to perform some calculation



Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

State

- **To keep track of more information**— how many events of each type did we see this hour, all events that require joining, sums, averages, etc.

- **Local or internal state** - State that is accessible only by a specific instance of the stream processing application. This state is usually maintained and managed with an embedded, in-memory database running within the application.
  - The advantage of local state is that it is extremely fast.
  - The disadvantage is that we are limited to the amount of memory available
- External state - State that is maintained in an external data store, often a NoSQL system like Cassandra.
  - The advantages of an external state are its virtually unlimited size and the fact that it can be accessed from multiple instances of the application or even from different applications.
  - The downside is the extra latency and complexity introduced with an additional system, as well as availability—the application needs to handle the possibility that the external system is not available

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Table -> Stream

- To convert a table to a stream, we need to capture the changes that modify the table.

- Take all those insert, update, and delete events and store them in a stream. Most databases offer change data capture (CDC) solutions for capturing these changes, and there are many Kafka connectors that can pipe those changes into Kafka where they will be available for stream processing.

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Stream -> Table

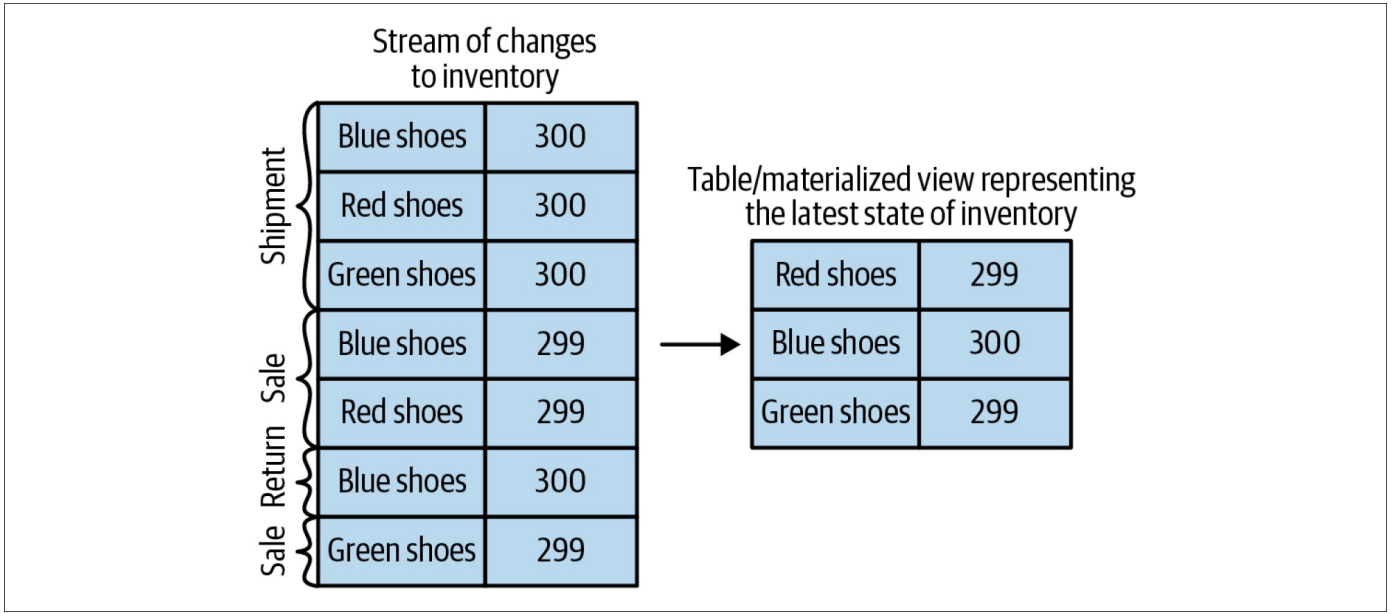
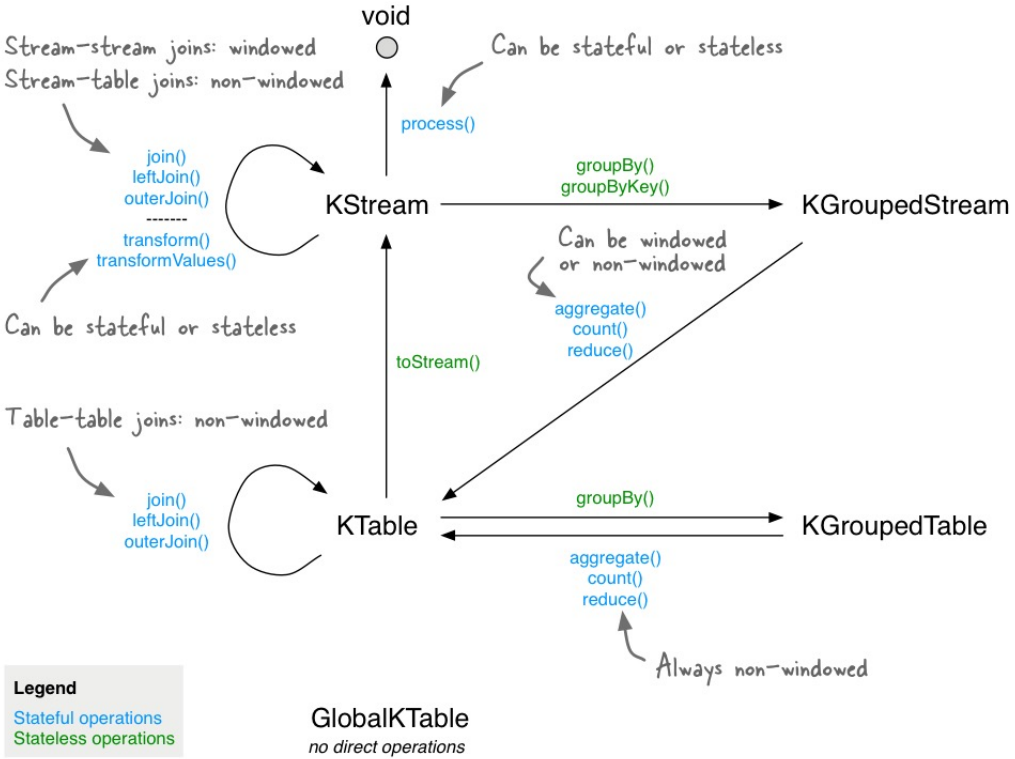


Figure 14-1. Materializing inventory changes

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

KStreams-KTable duality



https://docs.confluent.io/platform/current/streams/developer-guide/dsl-api.html#streams-dsl

Time Windows

Most operations on streams are windowed operations, operating on slices of time: moving averages, top products sold this week, 99th percentile load on the system, *etc.*

When calculating moving averages, we want to know:

- Size of the window
- How often the window moves (advance interval)
- How long the window remains updatable (grace period)

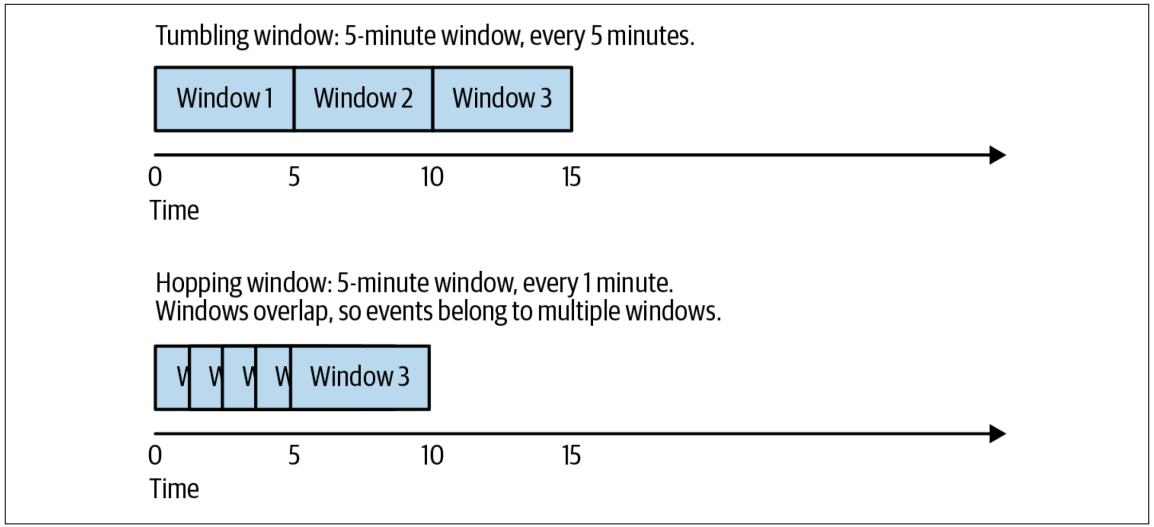


Figure 14-2. Tumbling window versus hopping window

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Stream Processing Design Patterns
Single-Event Processing

handled with a simple producer and consumer

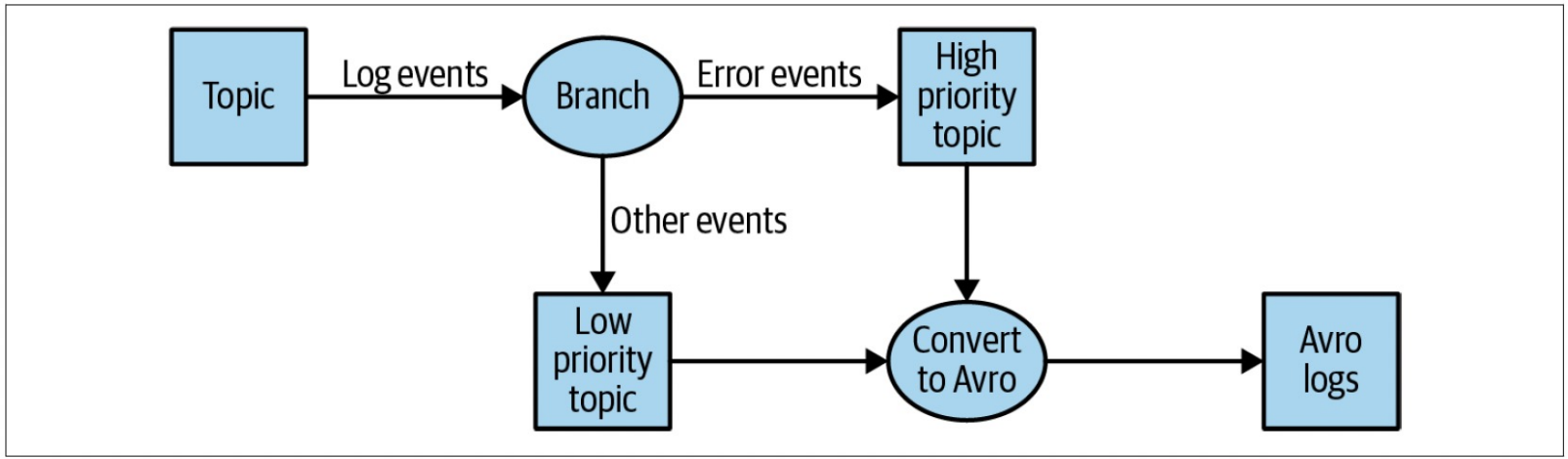


Figure 14-3. Single-event processing topology

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Stream Processing Design Patterns
Processing with Local State

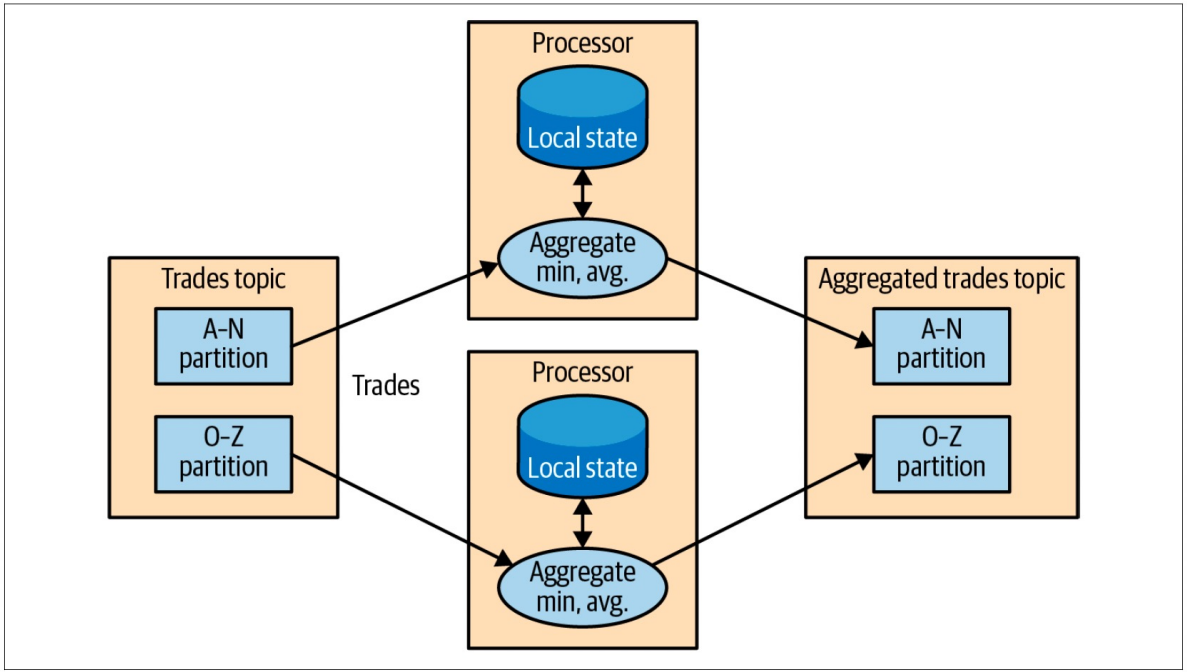


Figure 14-4. Topology for event processing with local state

done using local state (rather than a shared state)

...local state is stored in-memory using embedded RocksDB, which also persists the data to disk for quick recovery after restarts

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

Stream Processing Design Patterns
Multiphase Processing/Repartitioning

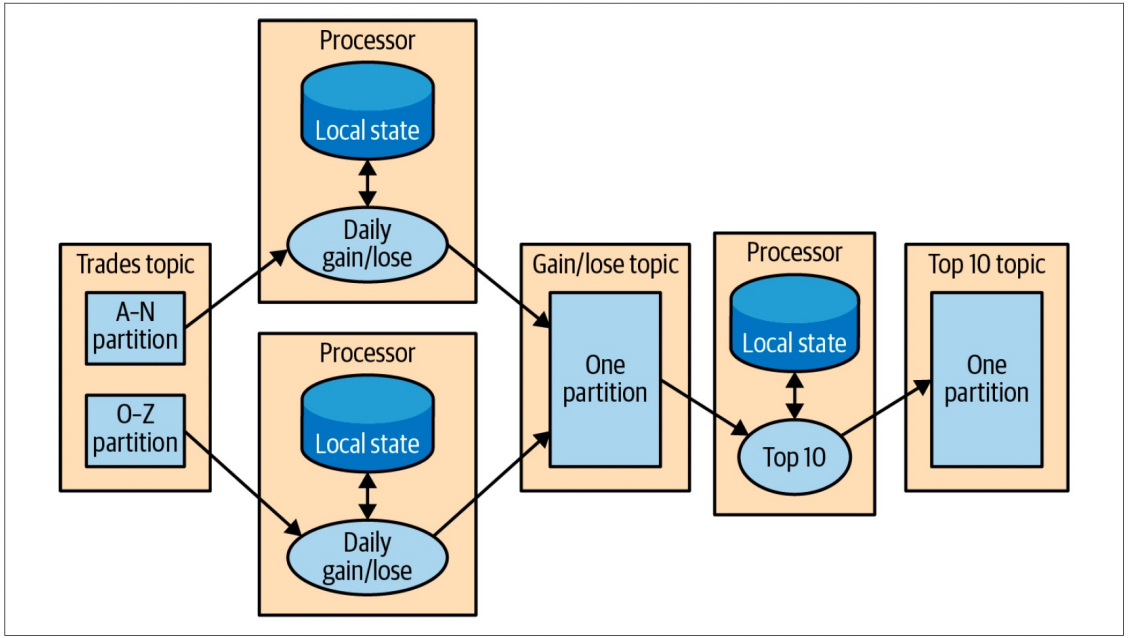


Figure 14-5. Topology that includes both local state and repartitioning steps

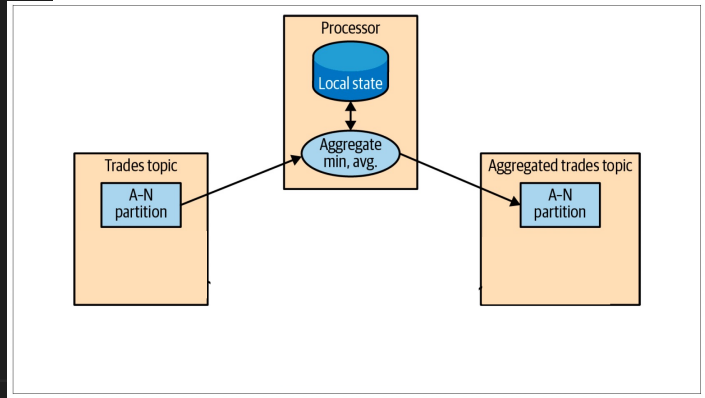
First, we calculate the daily gain/loss for each stock symbol. We can do this on each instance with a local state.
Then we write the results to a new topic with a single partition. This partition will be read by a single application instance that can then find the top 10 stocks for the day.
The second topic, which contains just the daily summary for each stock symbol, is obviously much smaller with significantly less traffic than the topics that contain the trades themselves, and therefore it can be processed by a single instance of the application

Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

```java
package com.shapira.examples.streams.wordcount;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.KeyValue;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.StreamsBuilder;
import java.util.Arrays;
import java.util.Properties;
import java.util.regex.Pattern;

public class WordCountExample {
    Run | Debug
    public static void main(String[] args) throws Exception{
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, value: "wordcount");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, value: "ec2-52-207-190-101.compute-1.amazonaws.com:9092");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, value: "earliest");
        StreamsBuilder builder = new StreamsBuilder();
        KStream<String, String> source = builder.stream("wordcount-input");

        final Pattern pattern = Pattern.compile(regex: "\\W+");
        KStream counts  = source.flatMapValues(value-> Arrays.asList(pattern.split(value.toLowerCase())))
                .map((key, value) -> new KeyValue<Object, Object>(value, value))
                .filter((key, value) -> (!value.equals(obj: "the")))
                .groupByKey()
                .count().mapValues(value->Long.toString(value)).toStream();
        counts.to("wordcount-output");

        KafkaStreams streams = new KafkaStreams(builder.build(), props);

        streams.cleanUp();
        streams.start();
        Thread.sleep(millis: 5000L);
        streams.close();
    }
}
```

WordCount Example

using JAVA lambda



Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2021). Kafka: the definitive guide. " O'Reilly Media, Inc."

TÉCNICO LISBOA

Q&A

TÉCNICO LISBOA