Apache Kafka

Trong-Hop Do

Kafka – A distributed event streaming flatform

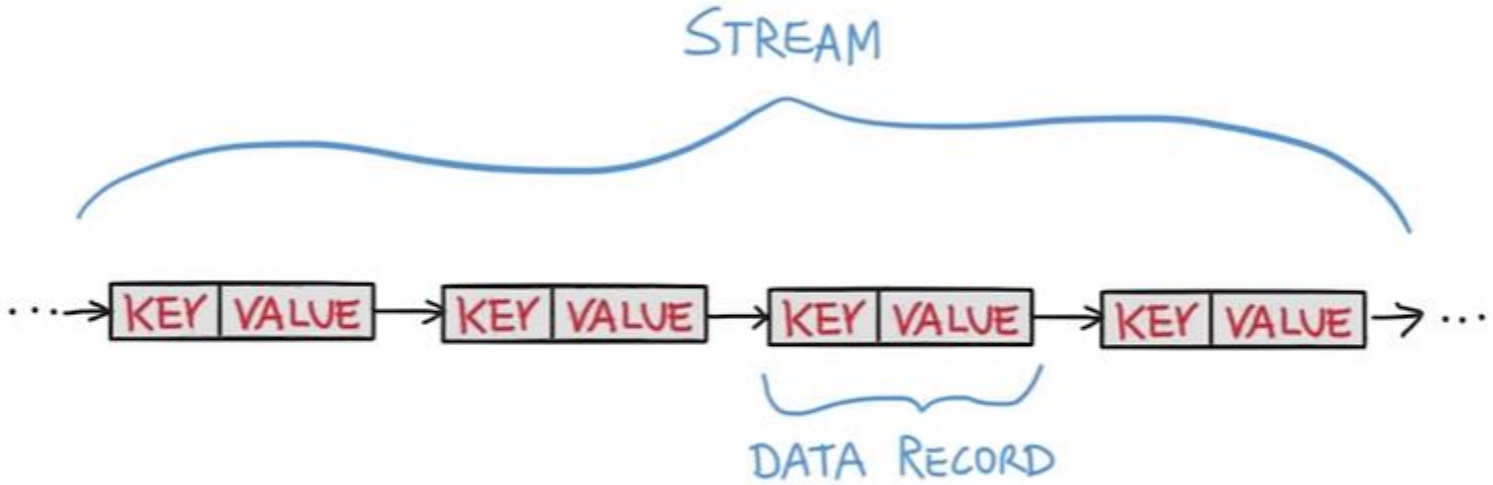
What is event streaming?

What can I use event streaming for?

- To process payments and financial transactions in real-time, such as in stock exchanges, banks, and insurances.
- To track and monitor cars, trucks, fleets, and shipments in real-time, such as in logistics and the automotive industry.
- To continuously capture and analyze sensor data from IoT devices or other equipment, such as in factories and wind parks.
- To collect and immediately react to customer interactions and orders, such as in retail, the hotel and travel industry, and mobile applications.
- To monitor patients in hospital care and predict changes in condition to ensure timely treatment in emergencies.
- To connect, store, and make available data produced by different divisions of a company.
- To serve as the foundation for data platforms, event-driven architectures, and microservices.

What is a stream?

- **Think of a stream as an unbounded, continuous real-time flow of records**
  - You don't need to explicitly request new records, you just receive them
- **Records are key-value pairs**



Motivation

The Shift to Event-driven Systems has Already Begun...

From a static snapshot...

Occasional call to a friend

Daily news reports

...to a continuous stream of events

A constant feed about the activities of all your friends

Real time news feeds, accessible online anytime, anywhere

Motivation

This leads us to...

Single platform to connect everyone to every event

Real-time stream of events

All events stored for historical view

Motivation

Successful Digital Businesses are Inherently Event-driven

Born cloud-native...

Social Networks

Enabling Event Sharing

Streaming Provider

On-demand Digital Content

Traditional ones that adapt...

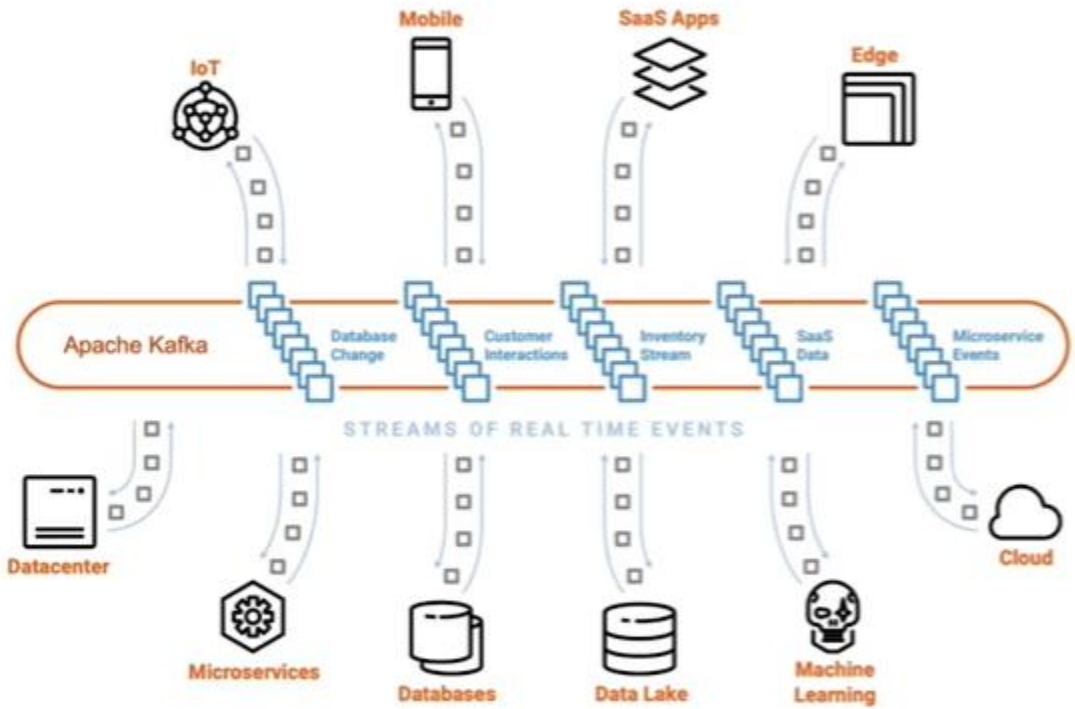
Newspaper

Provide a single Source of Truth

Credit Card Payments

Microservices Architecture

Motivation



Apache Kafka®: the De-facto Standard for Real-Time Event Streaming

- Global-scale
- Real-time
- Persistent Storage
- Stream Processing

Motivation

Thousands of Companies Worldwide trust Kafka for their Journey towards "**Event-driven**"

Travel

6 of top 10

Global banks

7 of top 10

Insurance

8 of top 10

Telecom

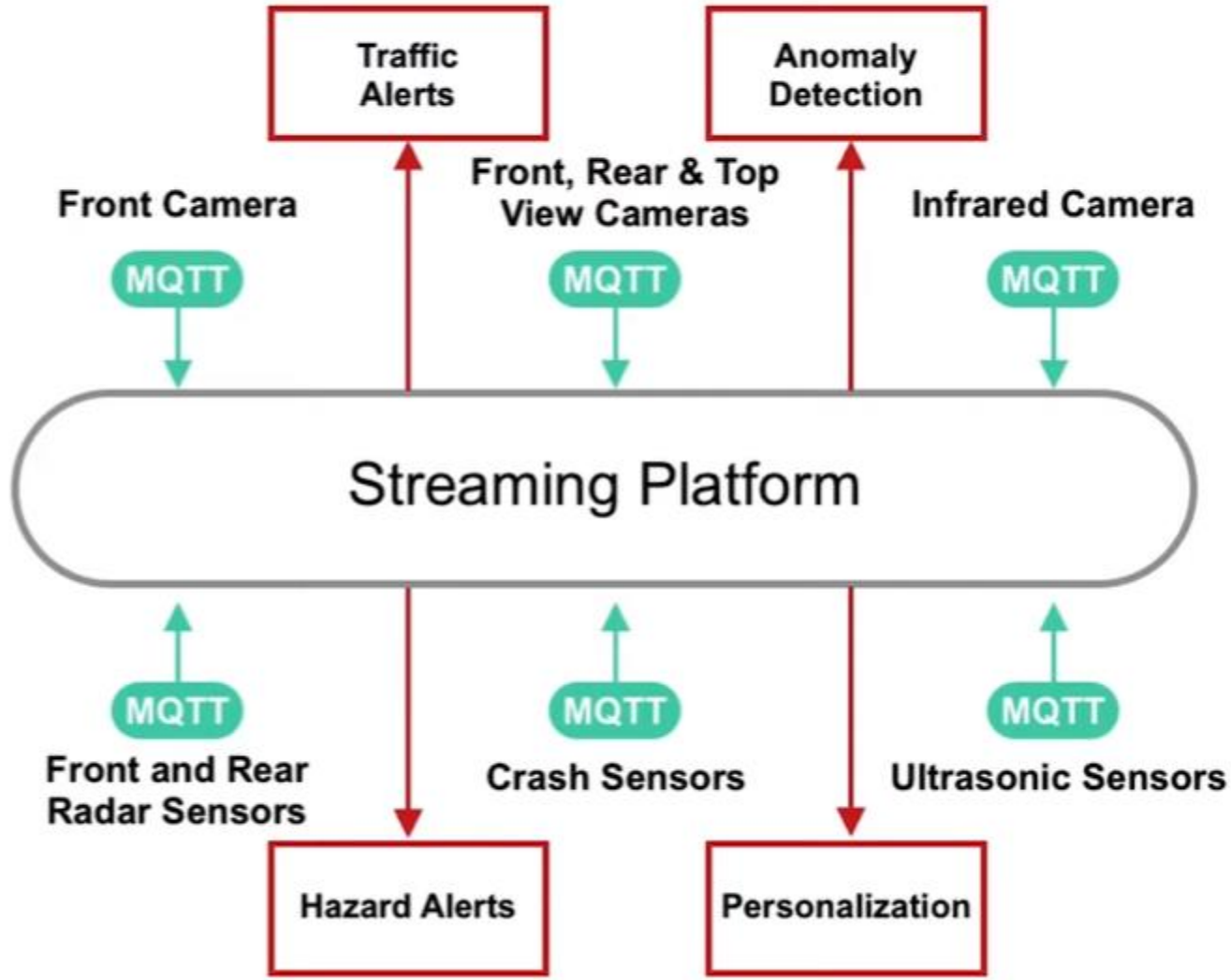
9 of top 10

Real-time Fraud Detection

- Act in real-time
- Detect fraud
- Minimize risk
- Improve customer experience

Automotive

The Future of the Automotive Industry is a Real Time Data Cluster



Real-time e-Commerce

Rewards Program

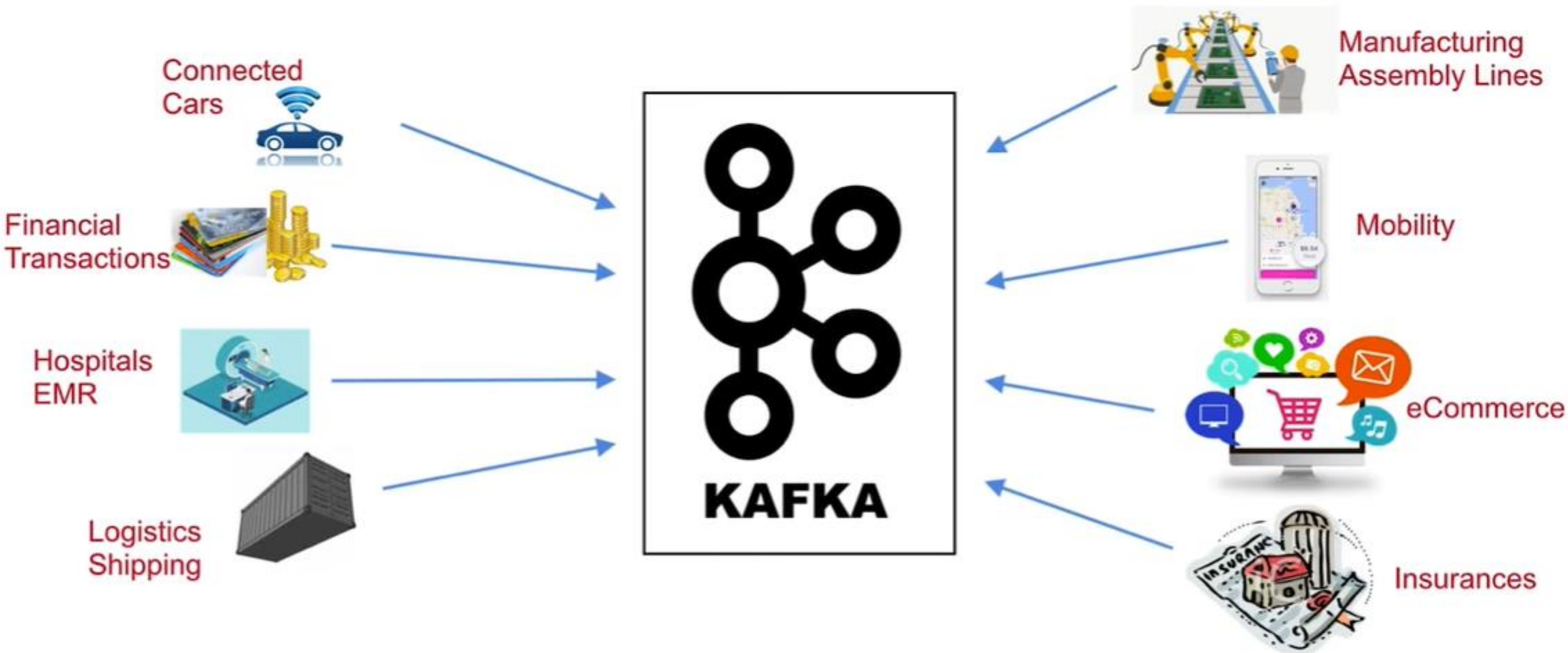
- Onboarding new merchants faster
- Increased speed at which mobile applications are delivered to customers
- Enabled a full 360 view of customers
- Enhanced performance and monitoring
- Projected savings of millions of dollars

Health Care

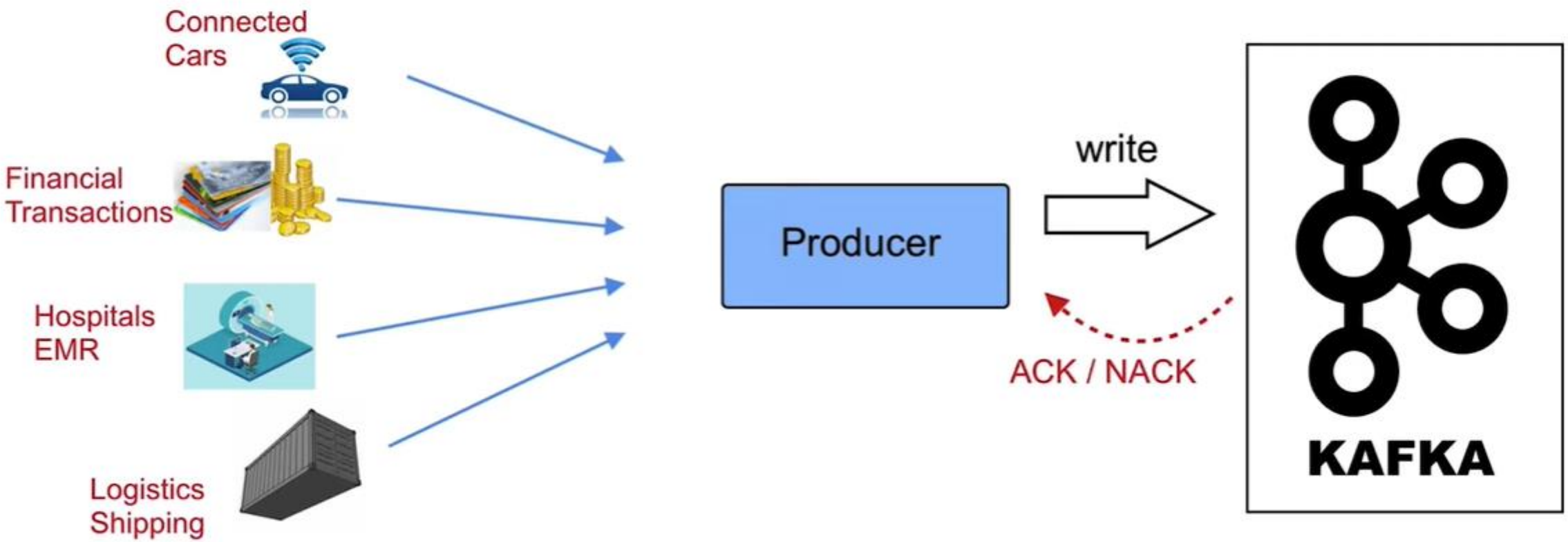
- Microservices

- Internet of Things

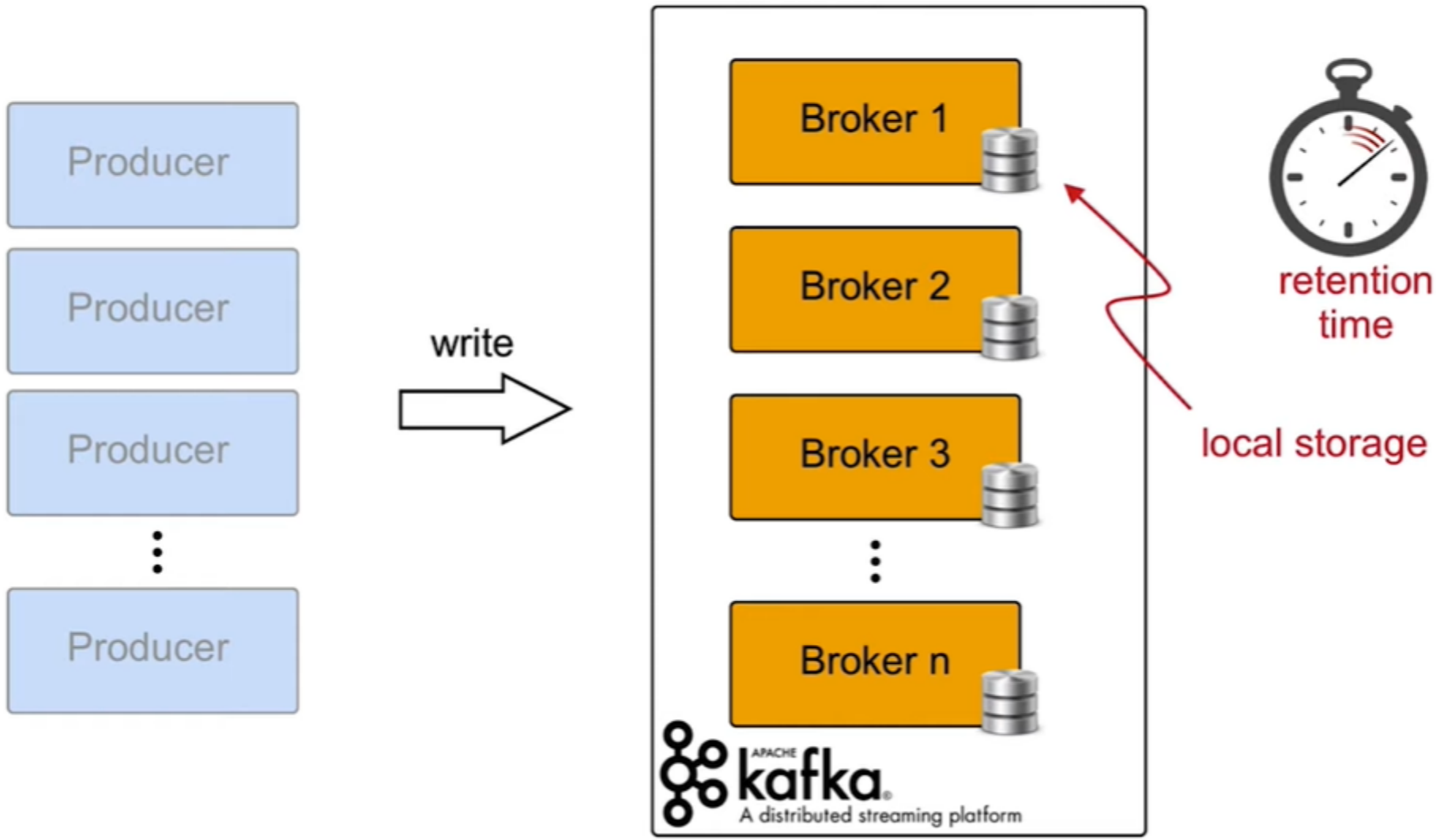
The World Produces Data



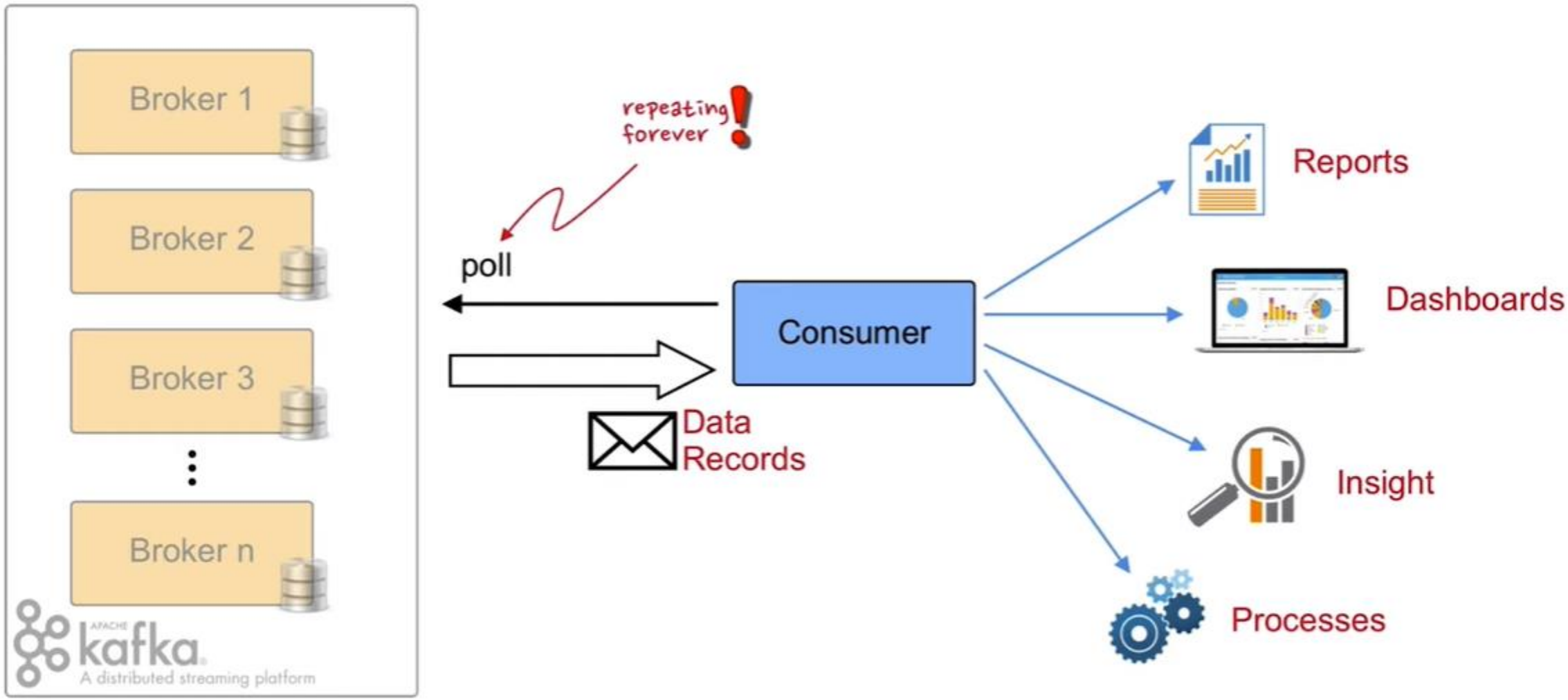
Producers


Connected Cars
Financial Transactions
Hospitals EMR
Logistics Shipping
Producer
write
ACK / NACK
KAFKA

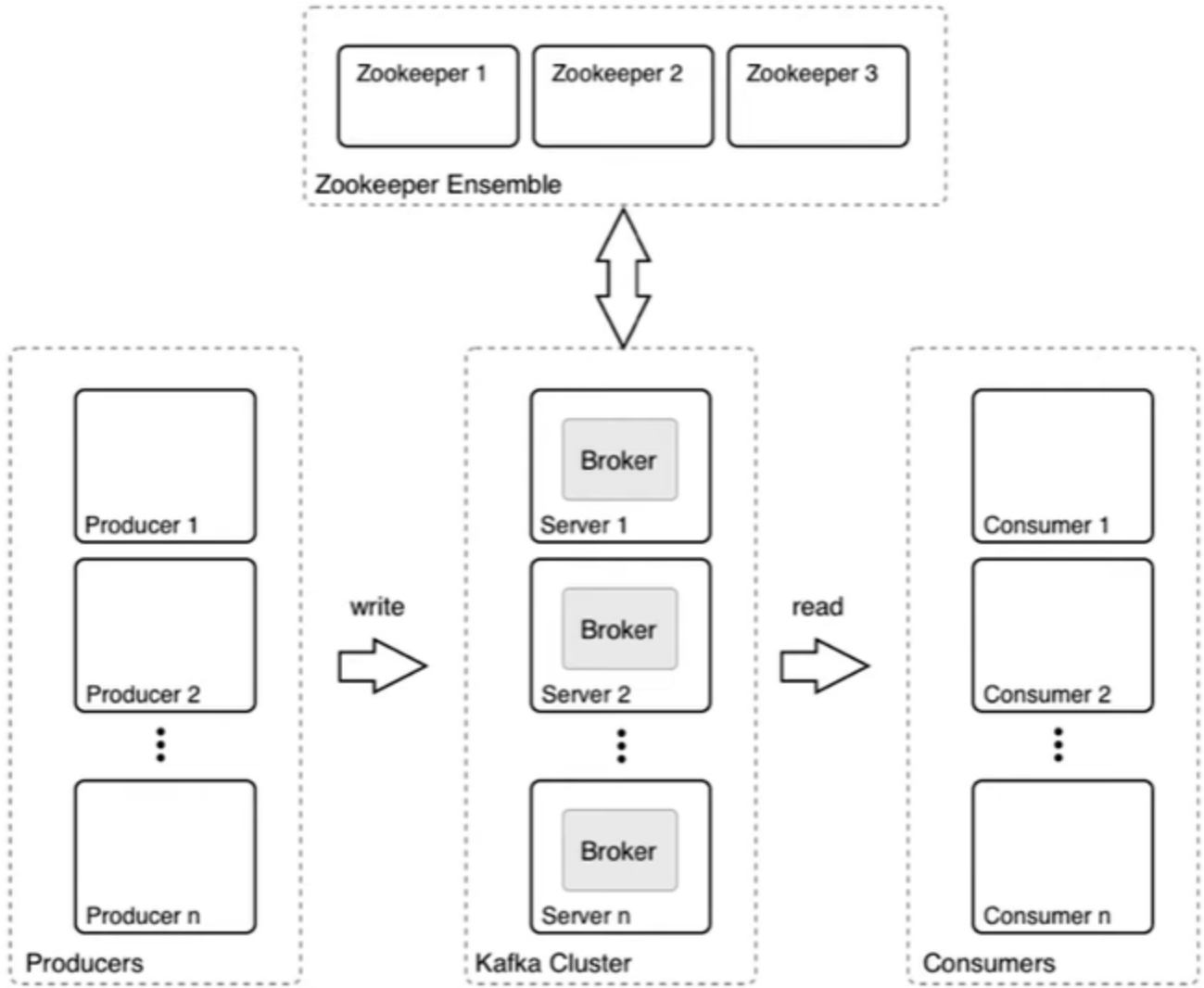
Kafka Brokers



Consumers

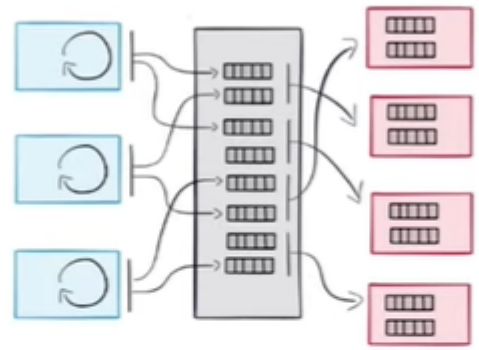

Broker 1
Broker 2
Broker 3
Broker n
APACHE kafka
A distributed streaming platform
repeating forever
poll
Consumer
Data Records
Reports
Dashboards
Insight
Processes

Architecture

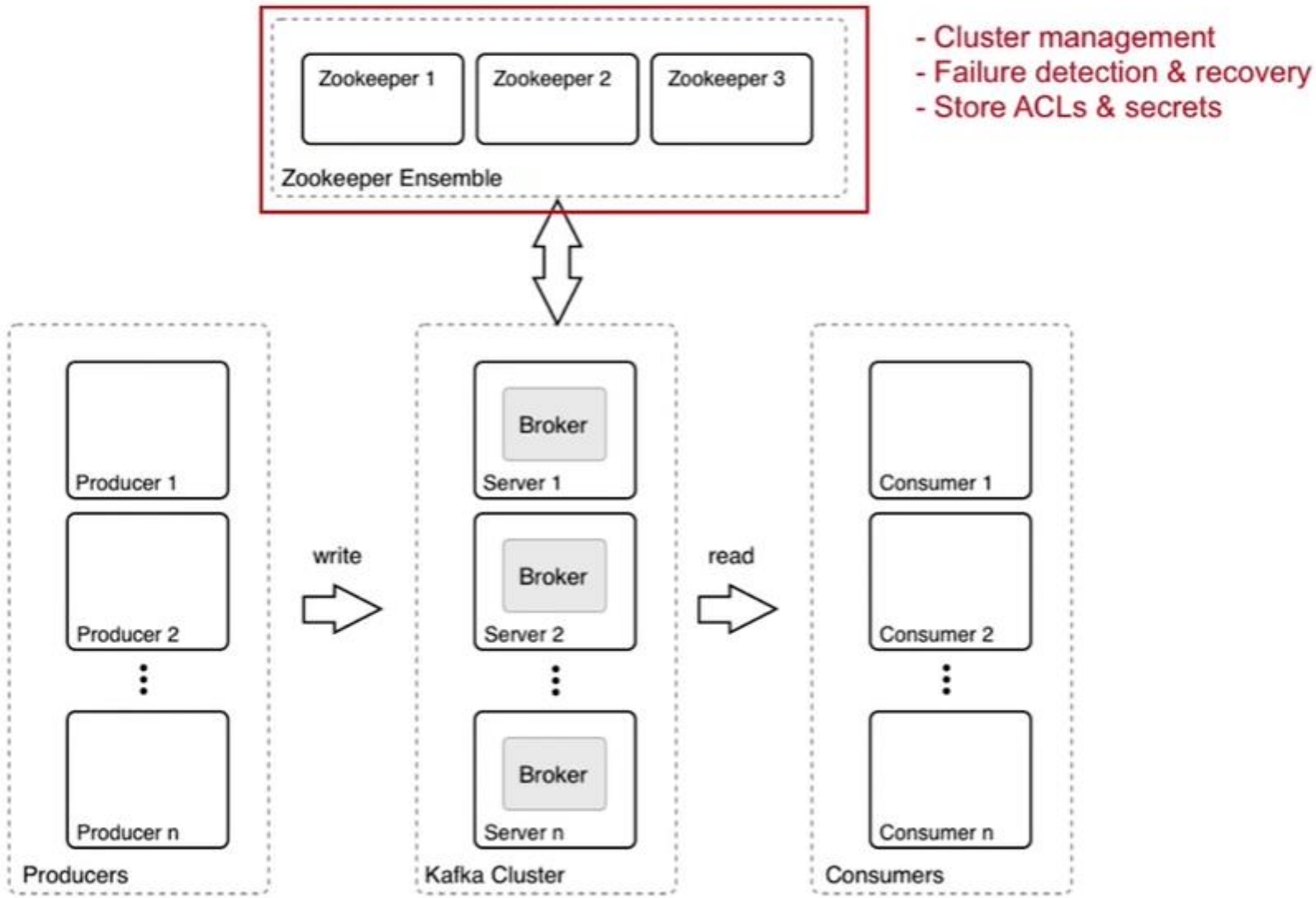


Decoupling Producers and Consumers

- Producers and Consumers are decoupled
- Slow Consumers do not affect Producers
- Add Consumers without affecting Producers
- Failure of Consumer does not affect System

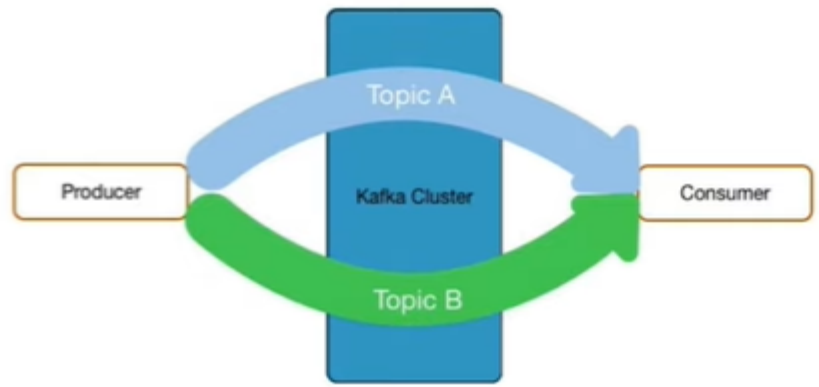


How Kafka Uses ZooKeeper

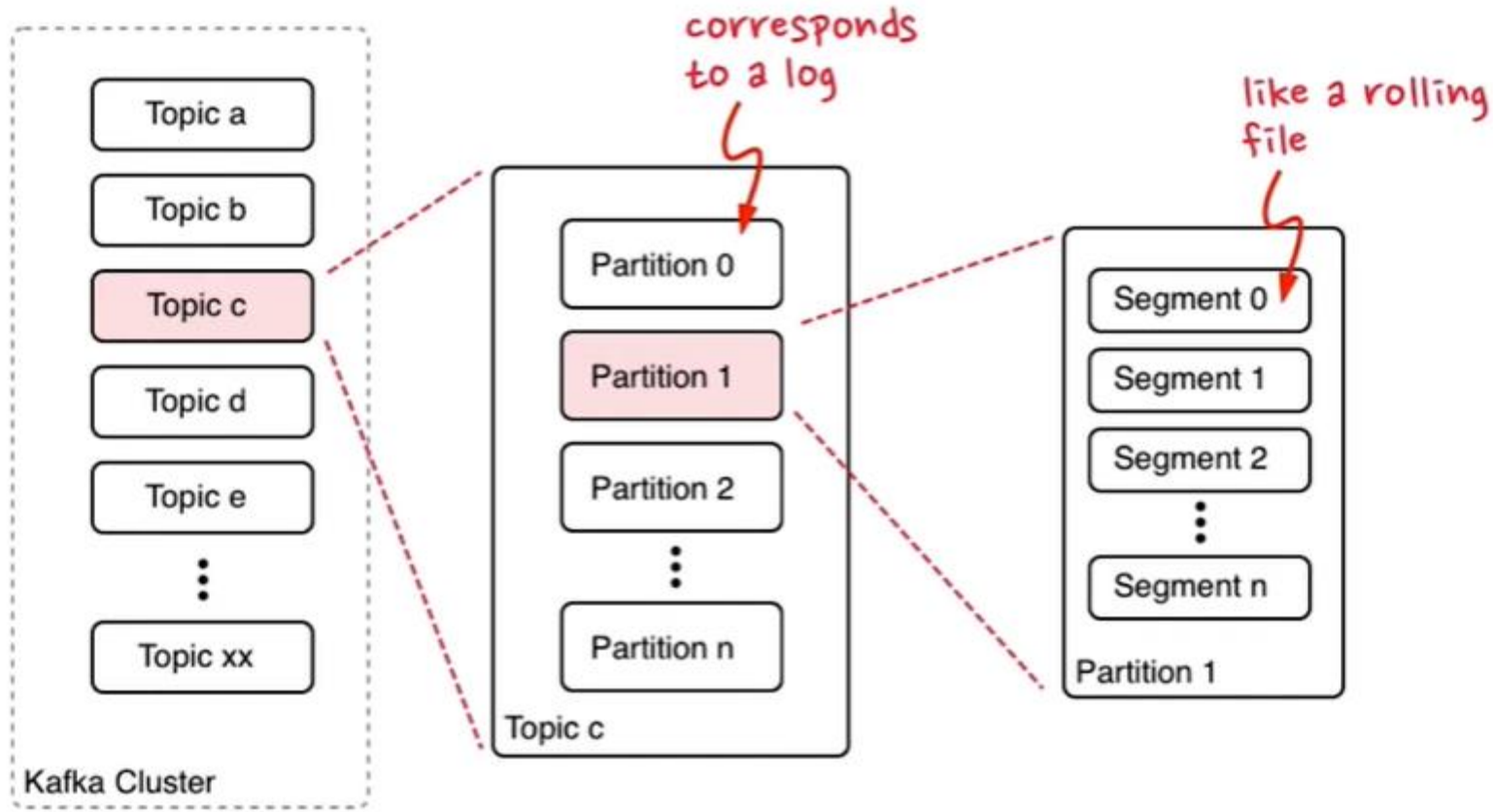


Topics

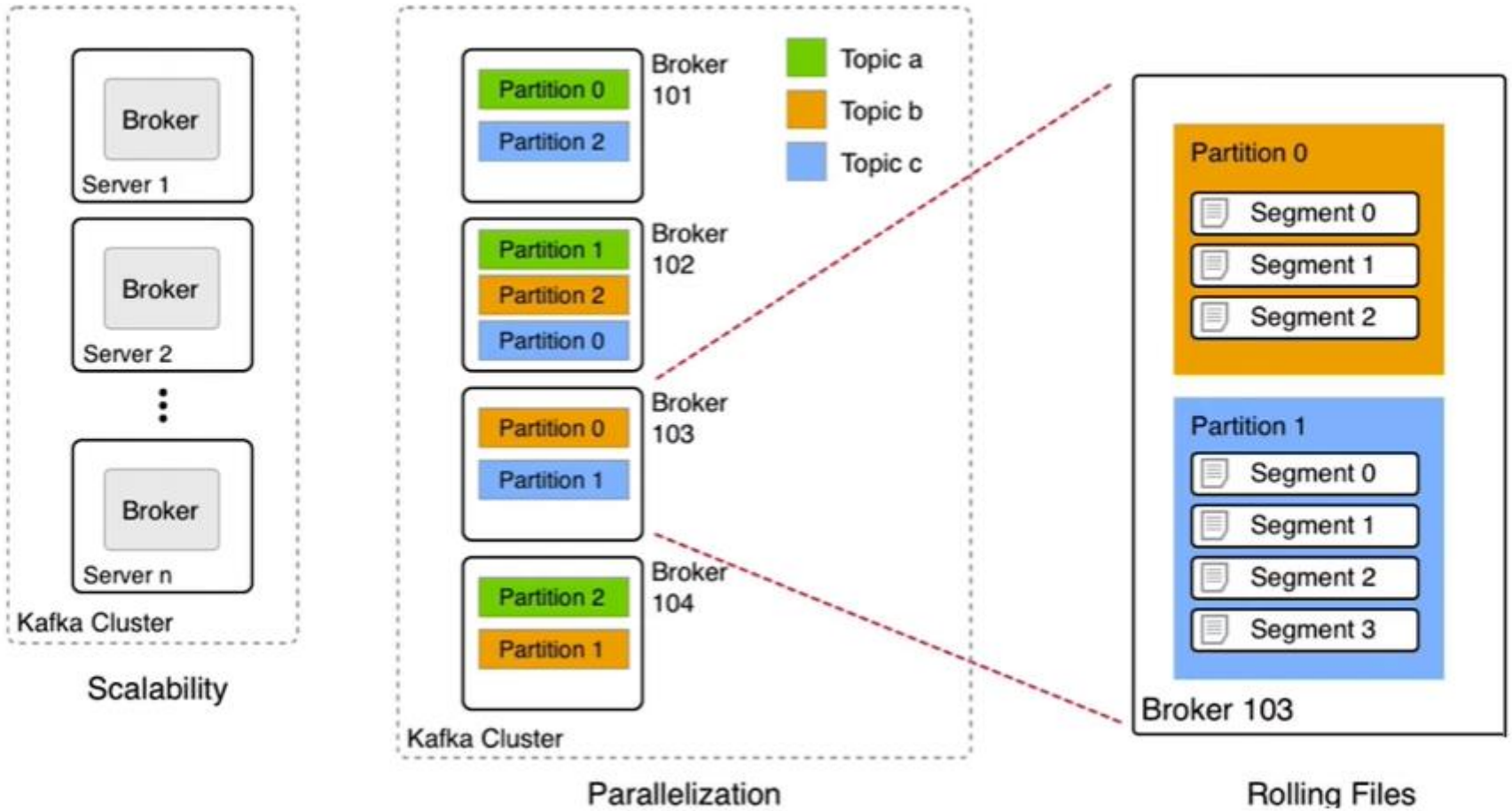
- **Topics:** Streams of "related" Messages in Kafka
  - Is a **Logical Representation**
  - **Categorizes Messages** into Groups
- Developers define Topics
- Producer ←→ Topic: N to N Relation
- Unlimited Number of Topics



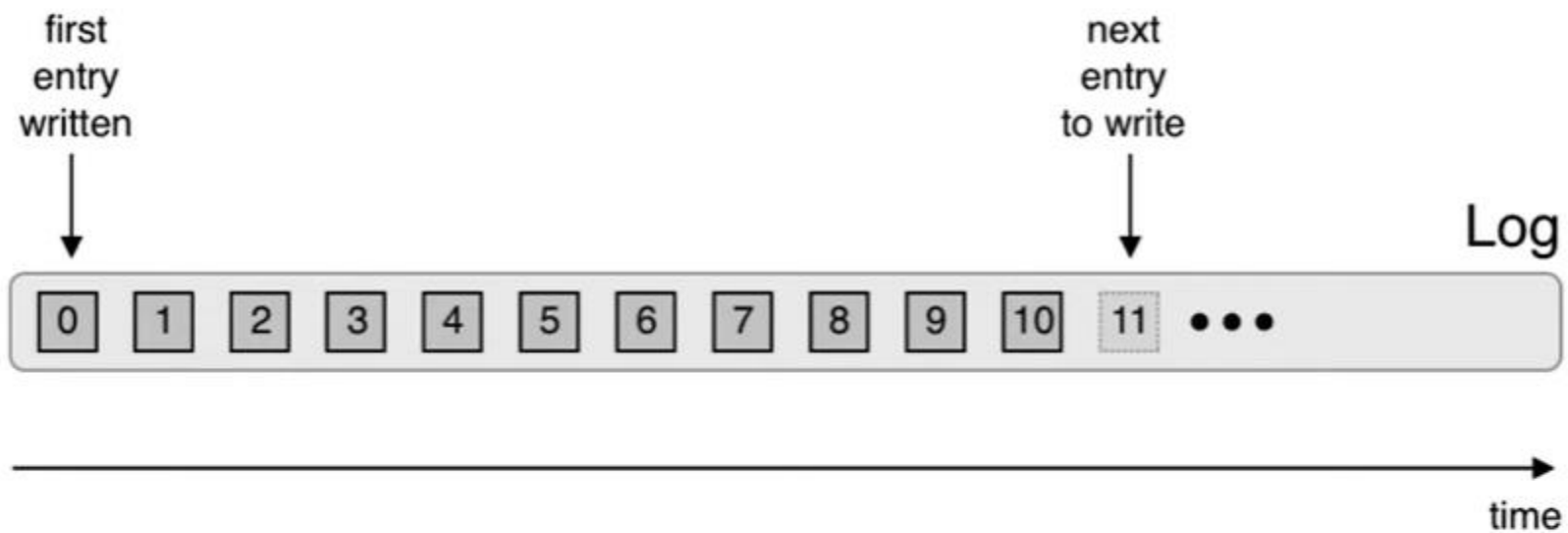
Topics, Partitions and Segments



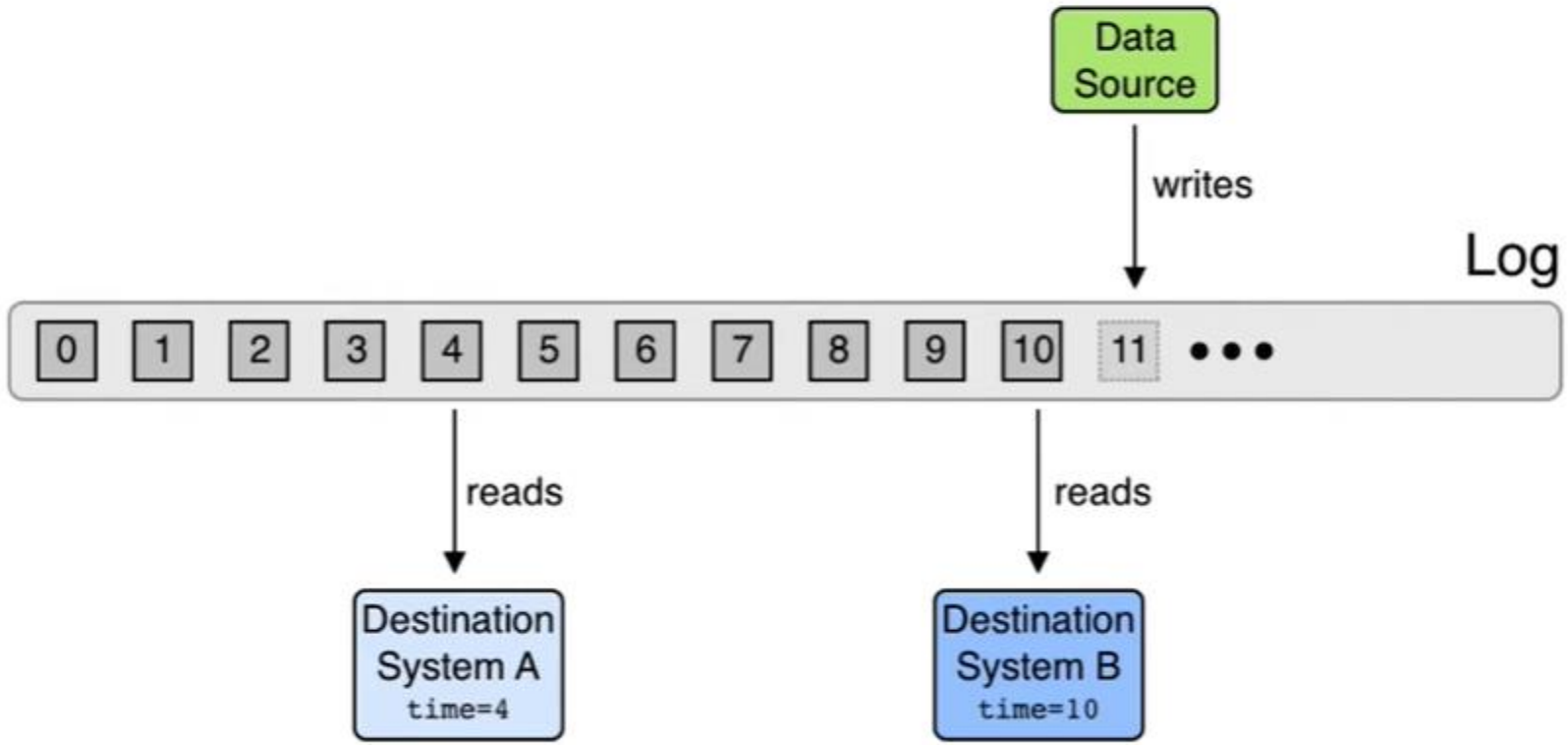
Topics, Partitions and Segments



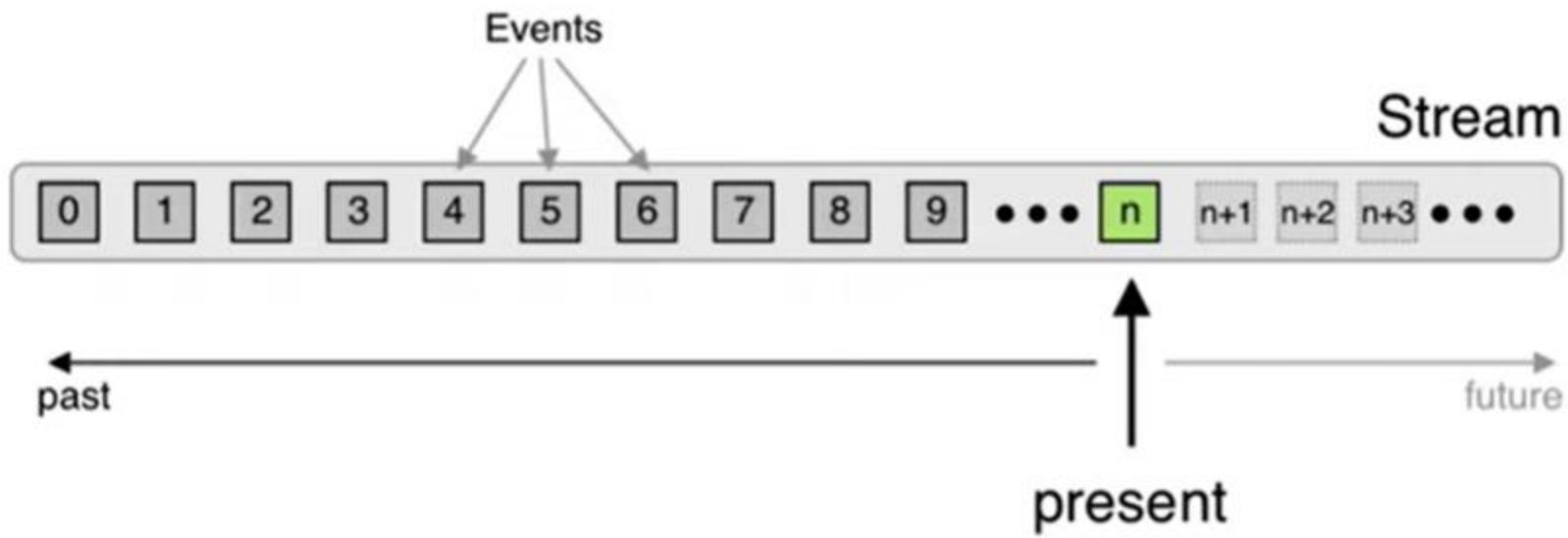
The Log


first
entry
written
next
entry
to write
Log
0
1
2
3
4
5
6
7
8
9
10
11
time

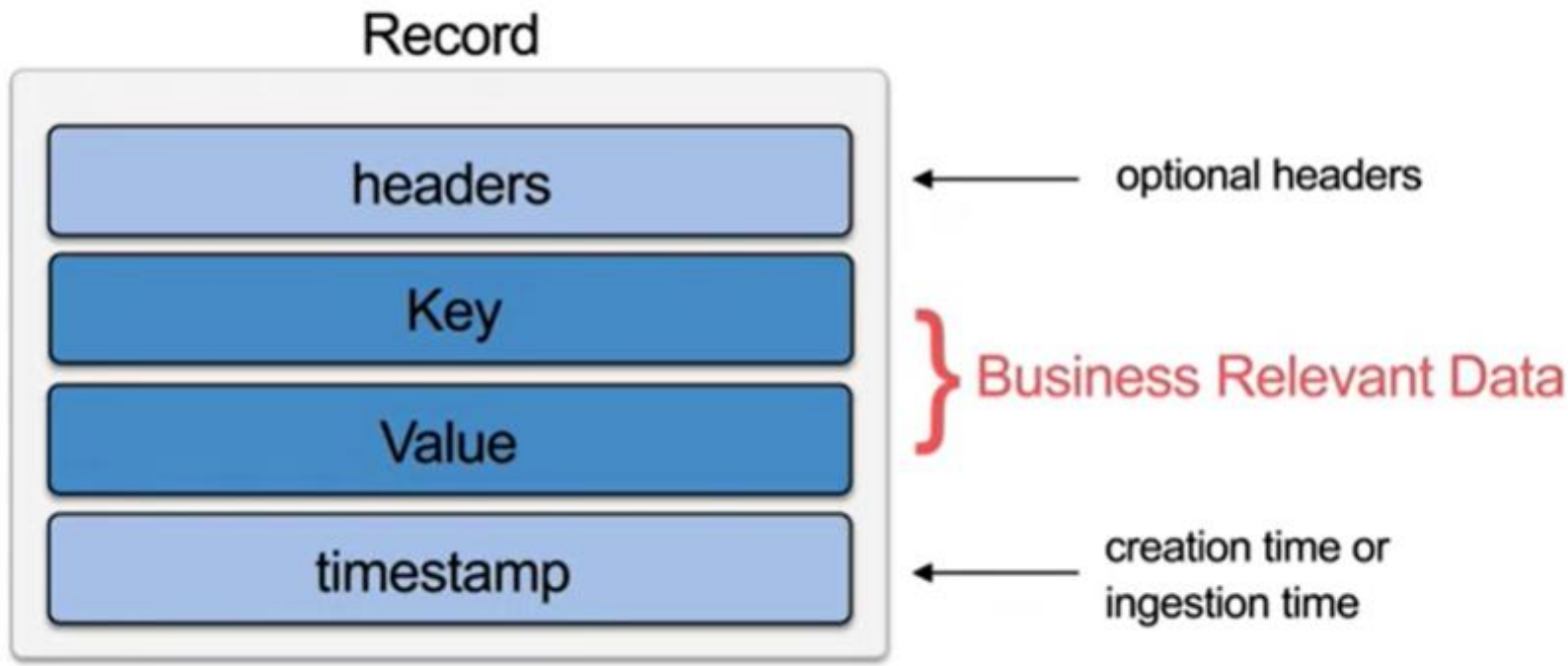
Log Structured Data Flow



The Stream



Data Elements

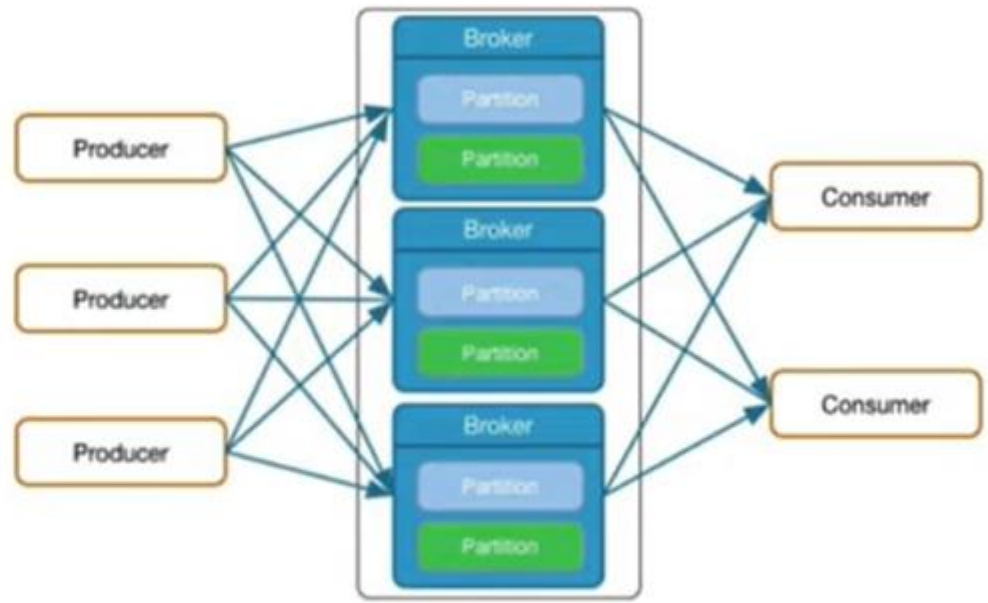


Brokers Manage Partitions

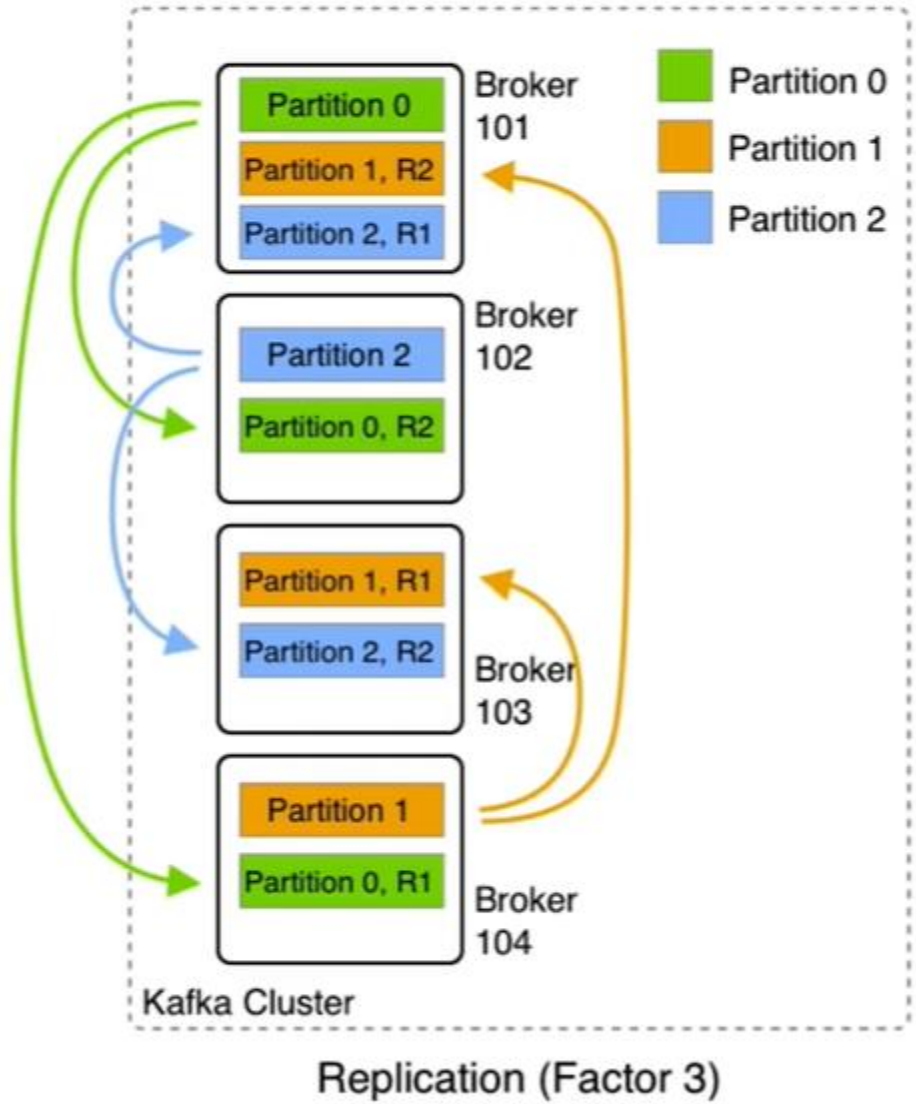
- Messages of Topic spread across Partitions
- Partitions spread across Brokers
- Each Broker handles many Partitions
- Each Partition stored on Broker's disk
- Partition: 1..n `log` files
- Each message in Log identified by *Offset*
- Configurable Retention Policy

Broker Basics

- Producer sends Messages to Brokers
- Brokers receive and store Messages
- A Kafka Cluster can have many Brokers
- Each Broker manages multiple Partitions



Broker Replication



Replication (Factor 3)

Producer Basics

- Producers write Data as Messages
- Can be written in any language
  - Native: Java, C/C++, Python, Go, .NET, JMS
  - More Languages by Community
  - REST Proxy for any unsupported Language
- Command Line Producer Tool

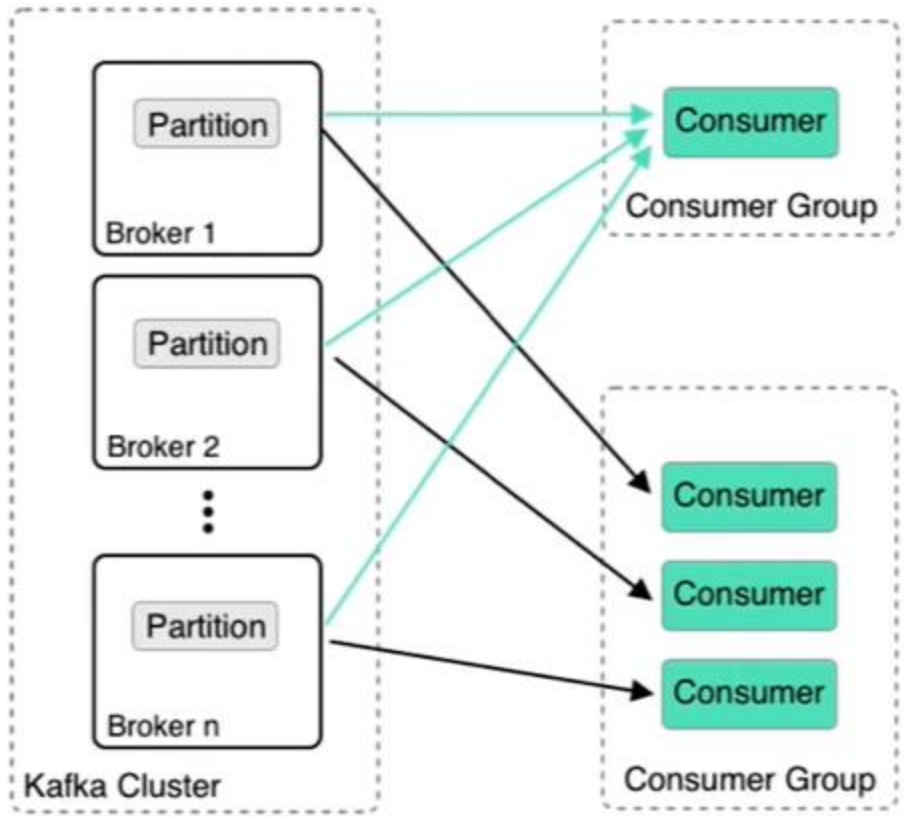
Load Balancing and Semantic Partitioning

- Producers use a Partitioning Strategy to assign each Message to a Partition
- Two Purposes:
  - Load Balancing
  - Semantic Partitioning
- Partitioning Strategy specified by Producer
  - Default Strategy: `hash(key) % number_of_partitions`
  - No Key → Round-Robin
- Custom Partitioner possible

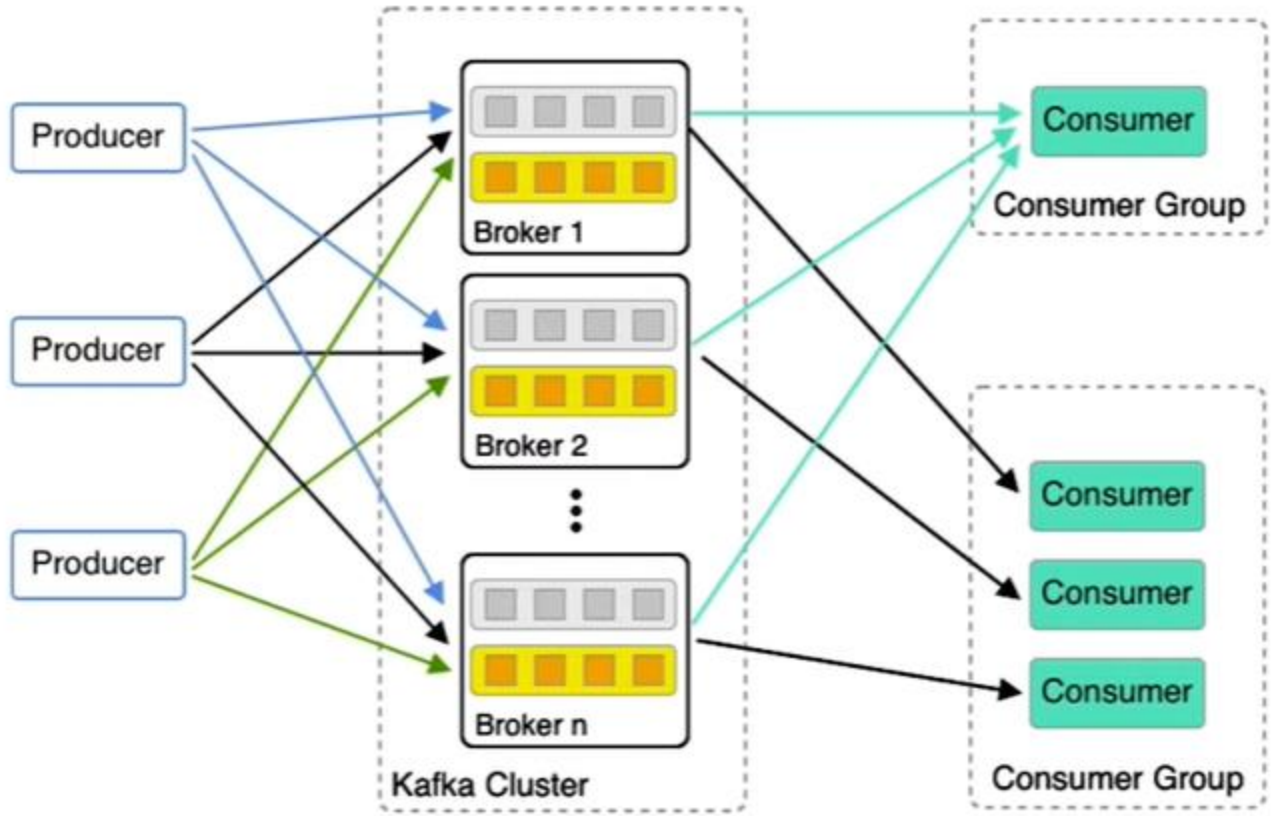
Consumer Basics

- Consumers **pull** messages from 1...n topics
- New inflowing messages are automatically retrieved
- Consumer offset
  - keeps track of the last message read
  - is stored in special topic
- CLI tools exist to read from cluster

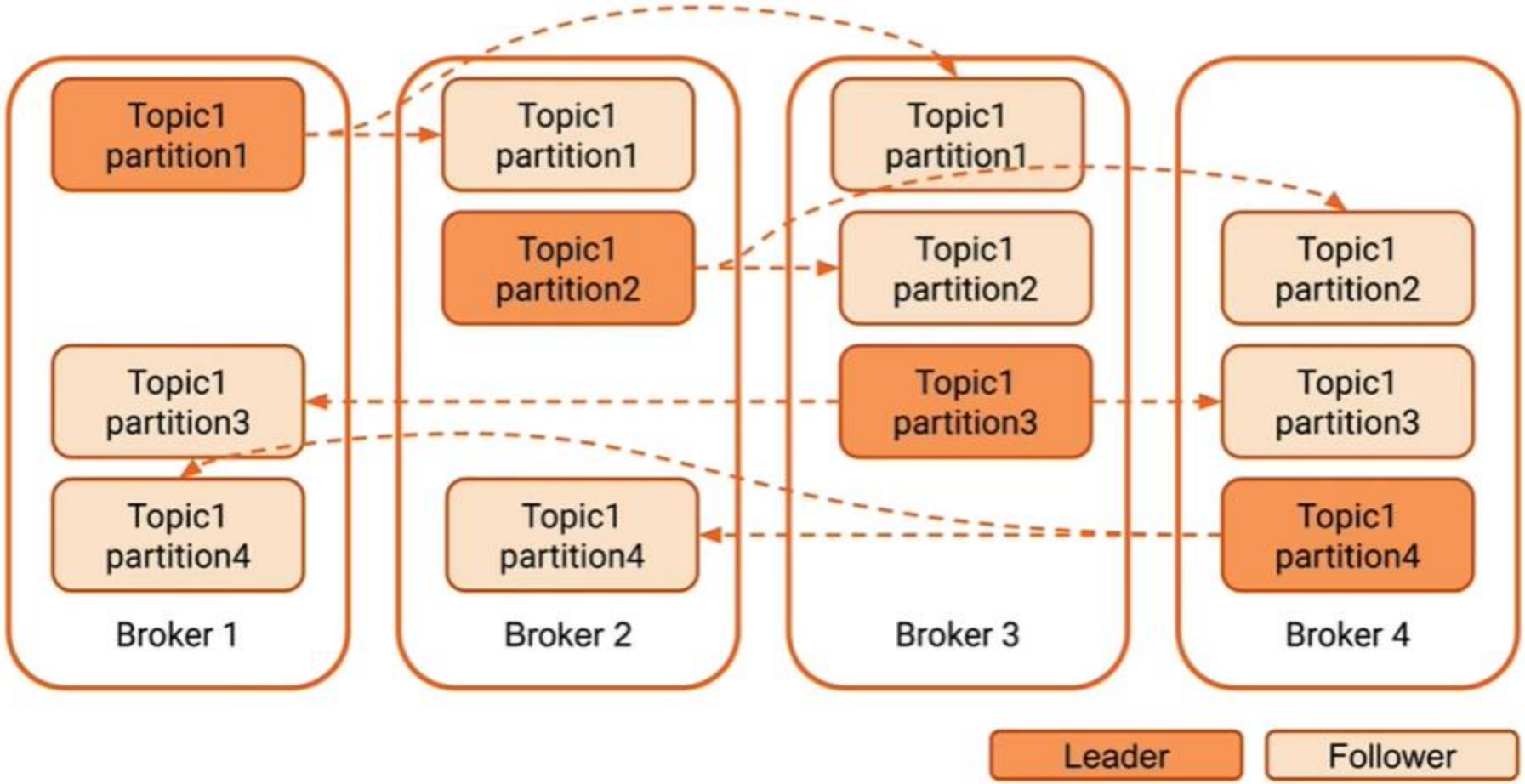
Distributed Consumption



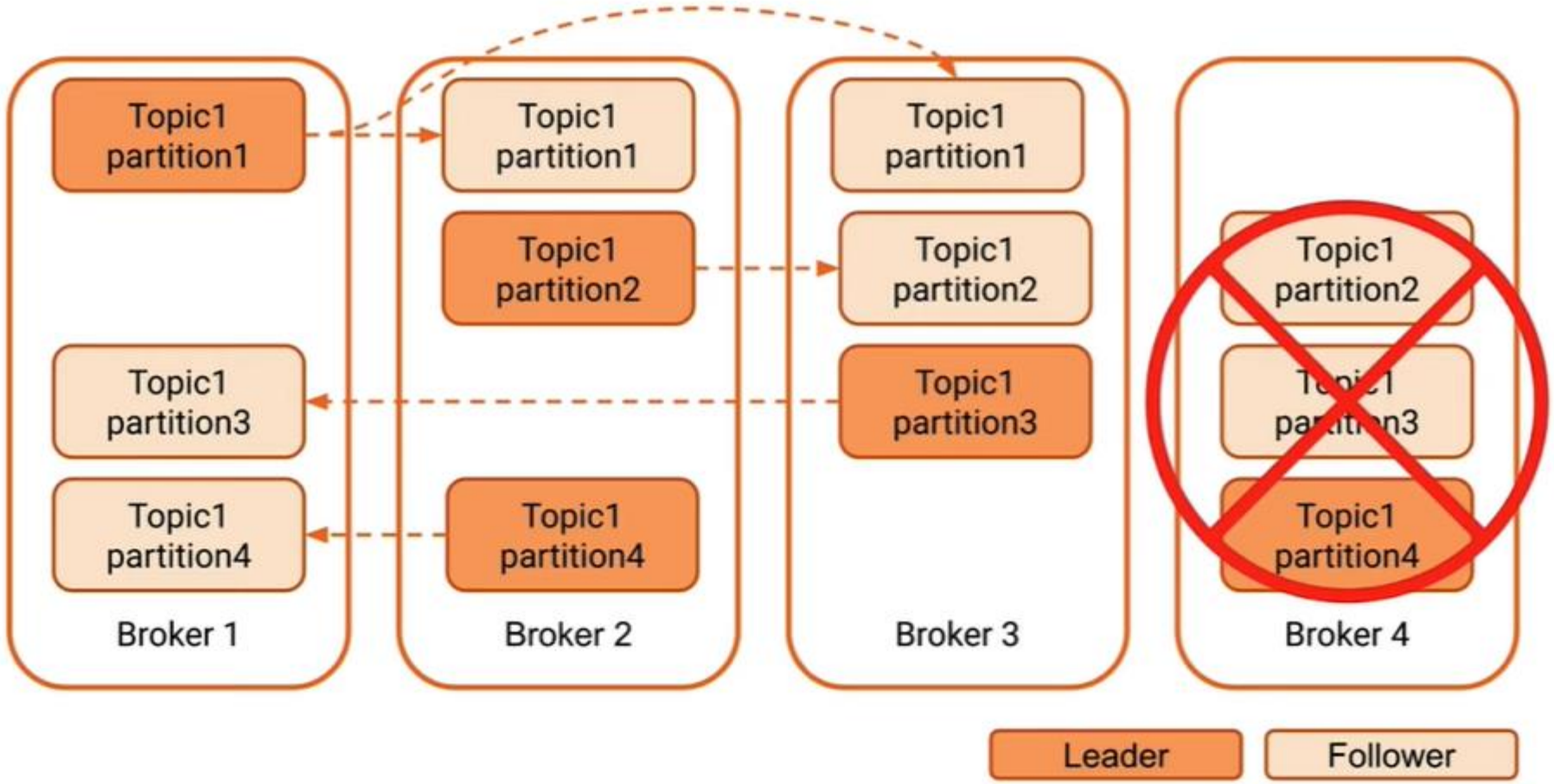
Scalable Data Pipeline



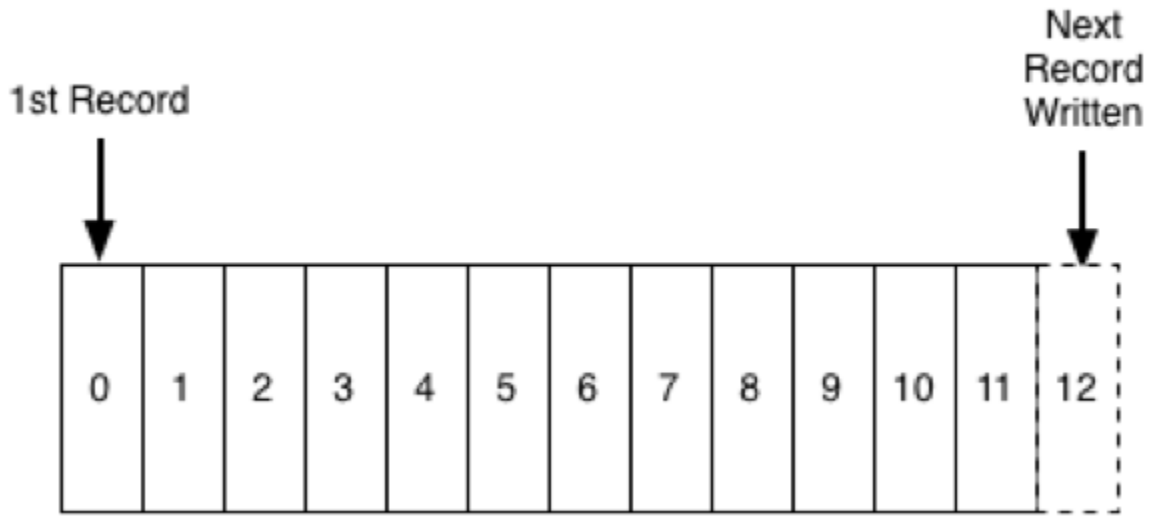
Partition Leadership & Replication



Partition Leadership & Replication



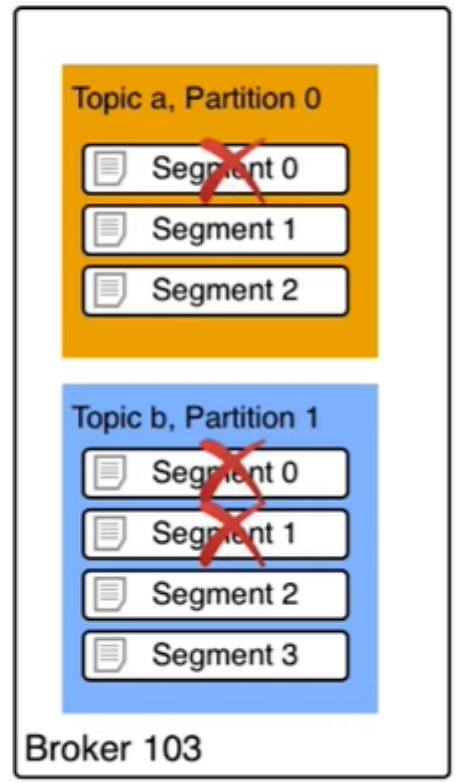
Store data in Kafka?



Data Retention Policy

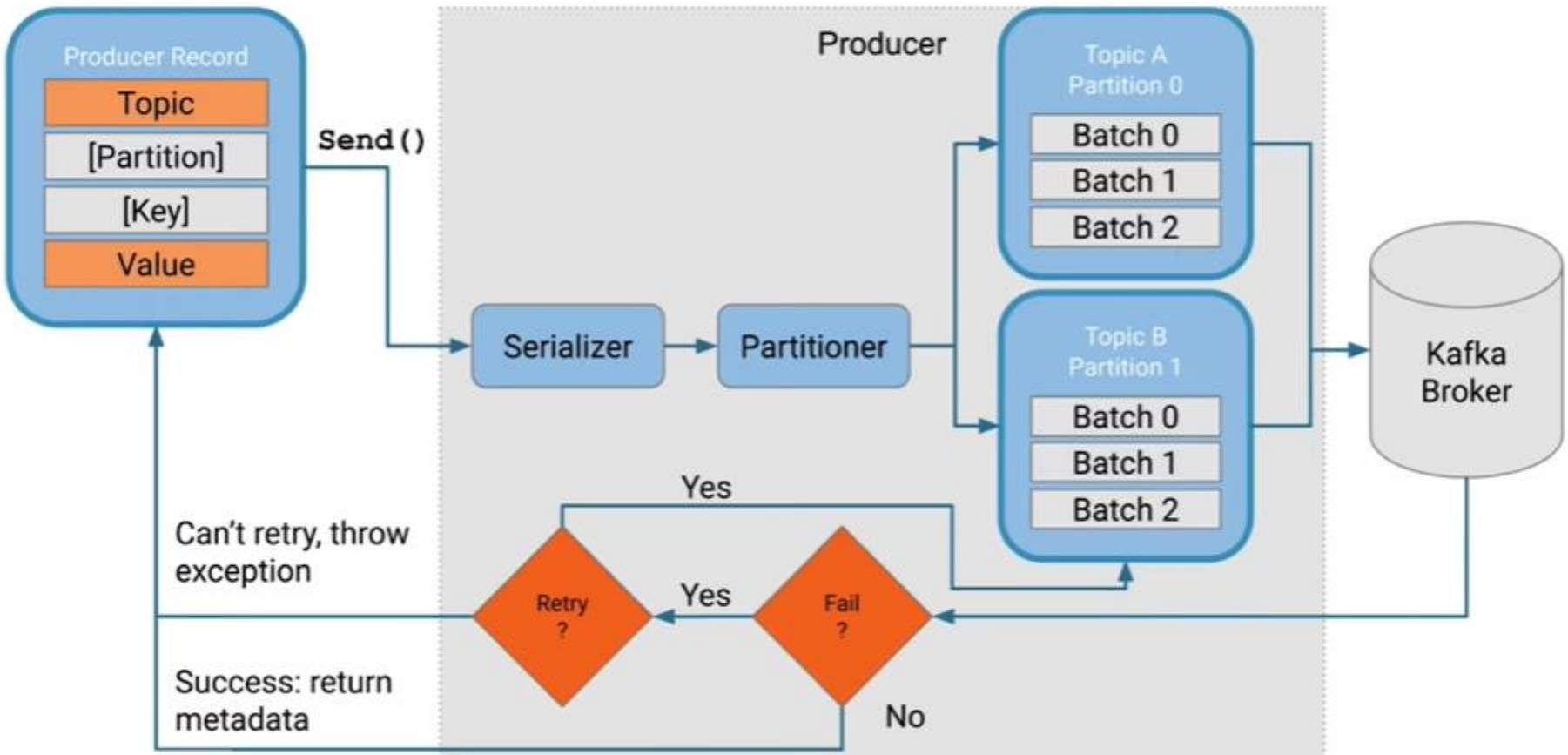
How long do I want or can I store my data?

- How long (default: `1 week`)
- Set **globally** or **per topic**
- Business decision
- Cost factor
- Compliance factor → GDPR

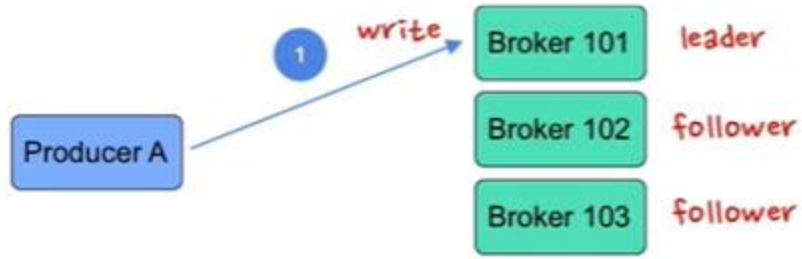


Retention Policy

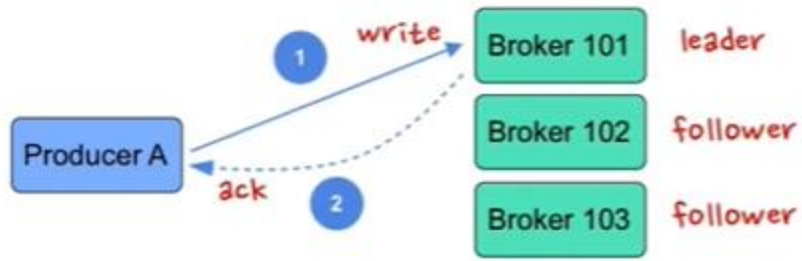
Producer Design



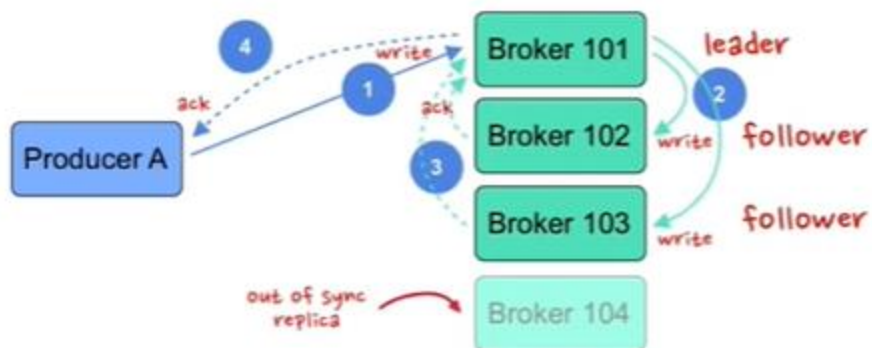
Producer Guarantees



Acks 0 (NONE)

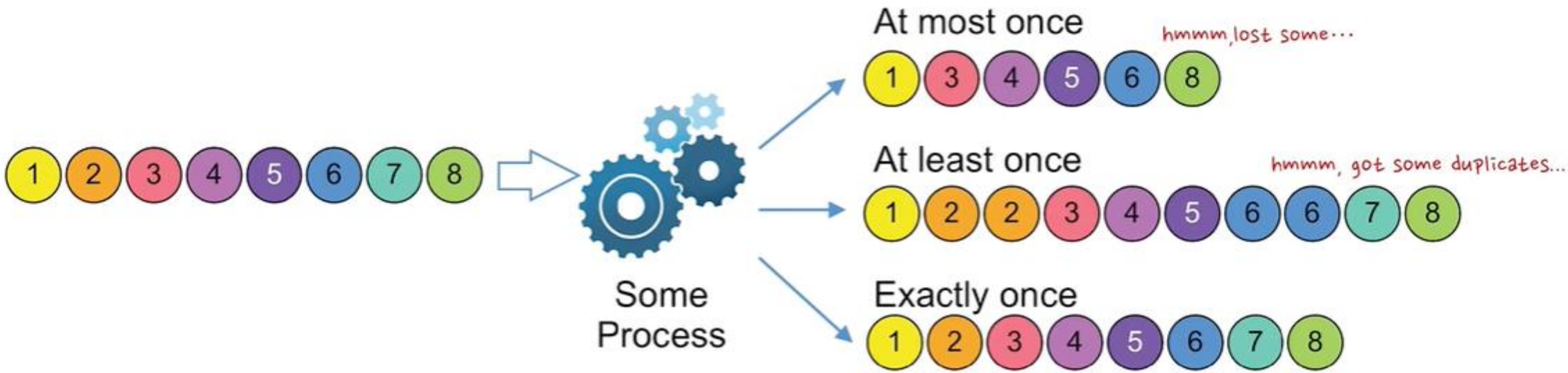


Acks 1 (LEADER)

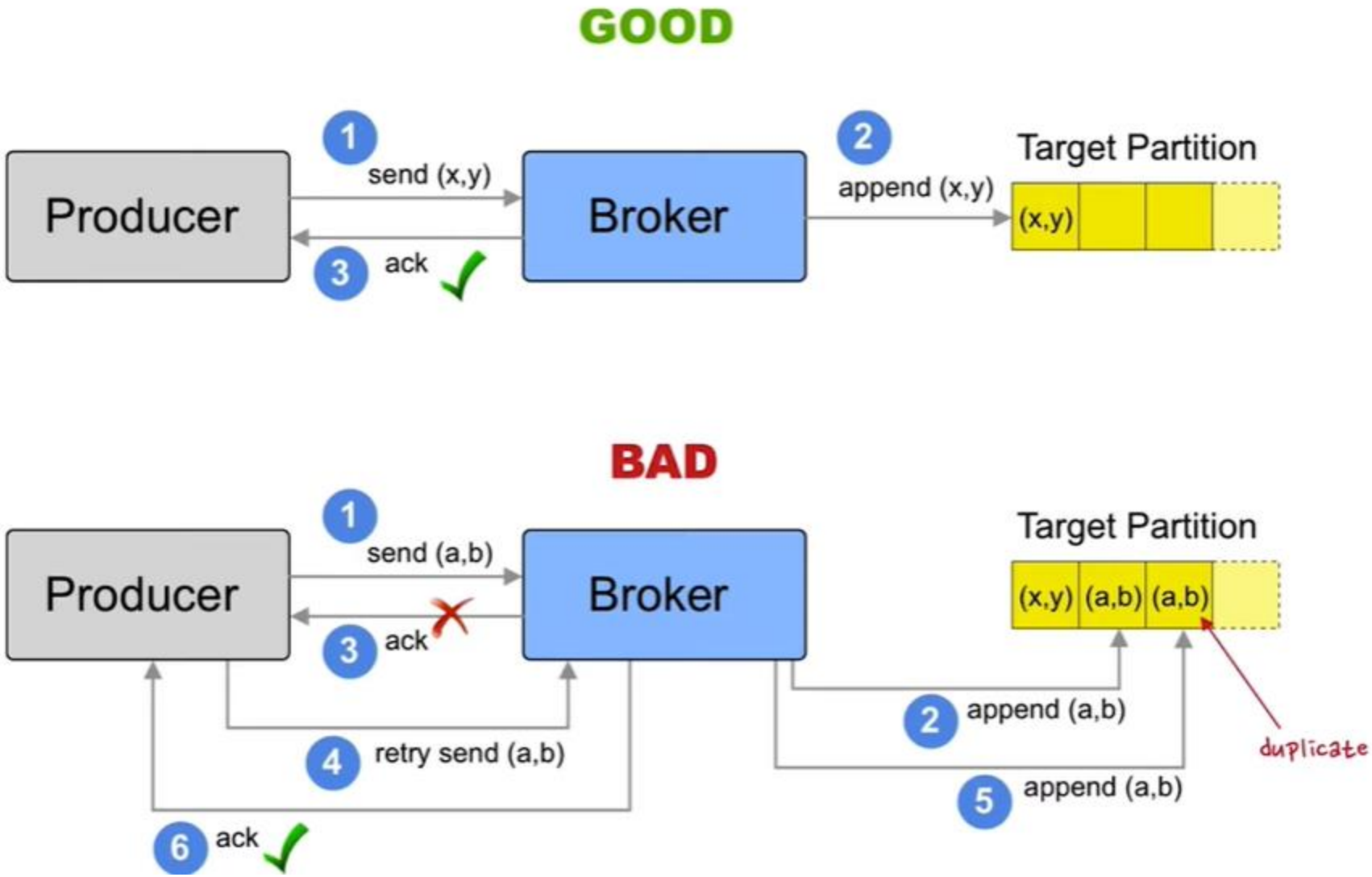


Acks -1 (ALL)

Delivery Guarantees



Idempotent Producers



Exactly Once Semantics

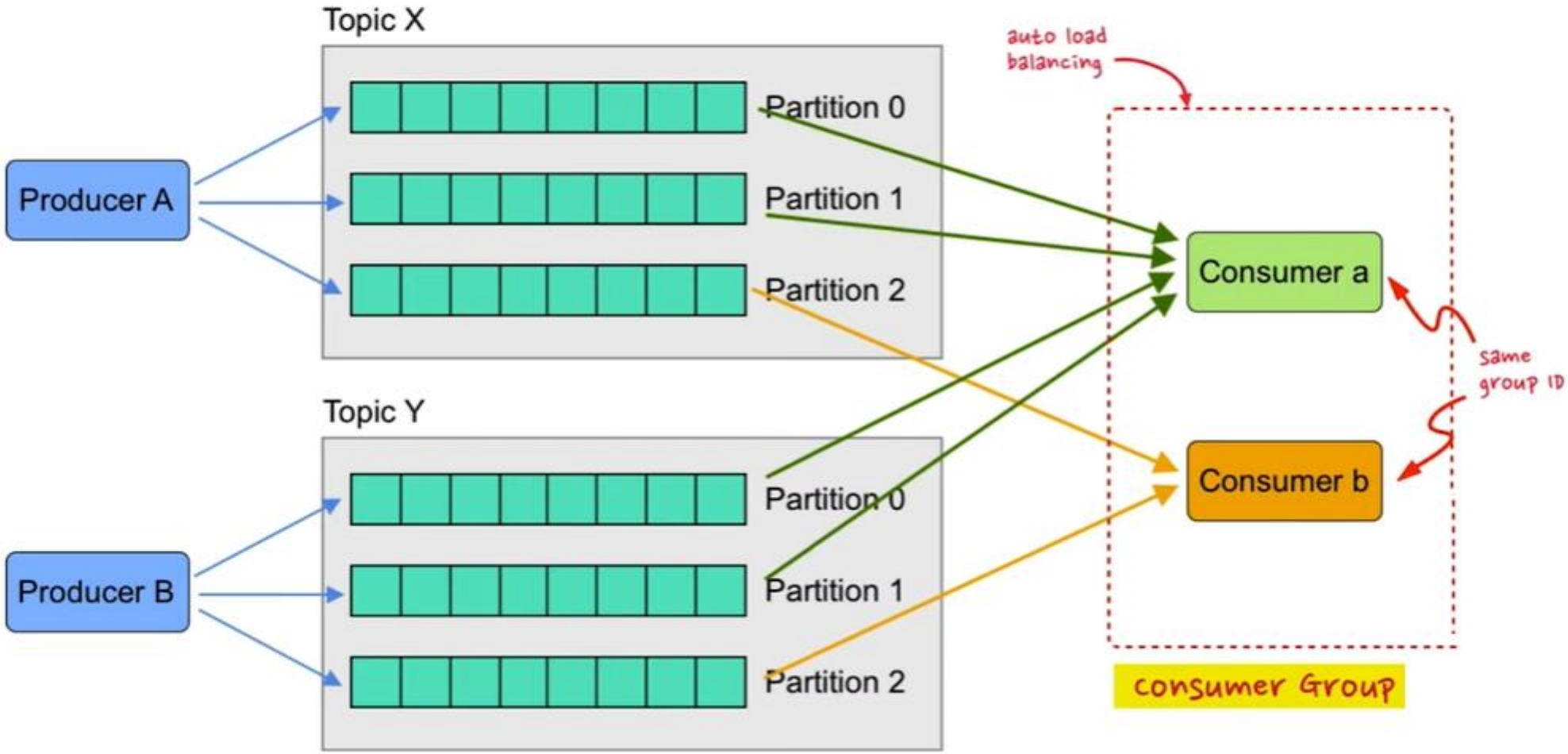
What?

- Strong **transactional guarantees** for Kafka
- Prevents clients from processing duplicate messages
- Handles failures gracefully

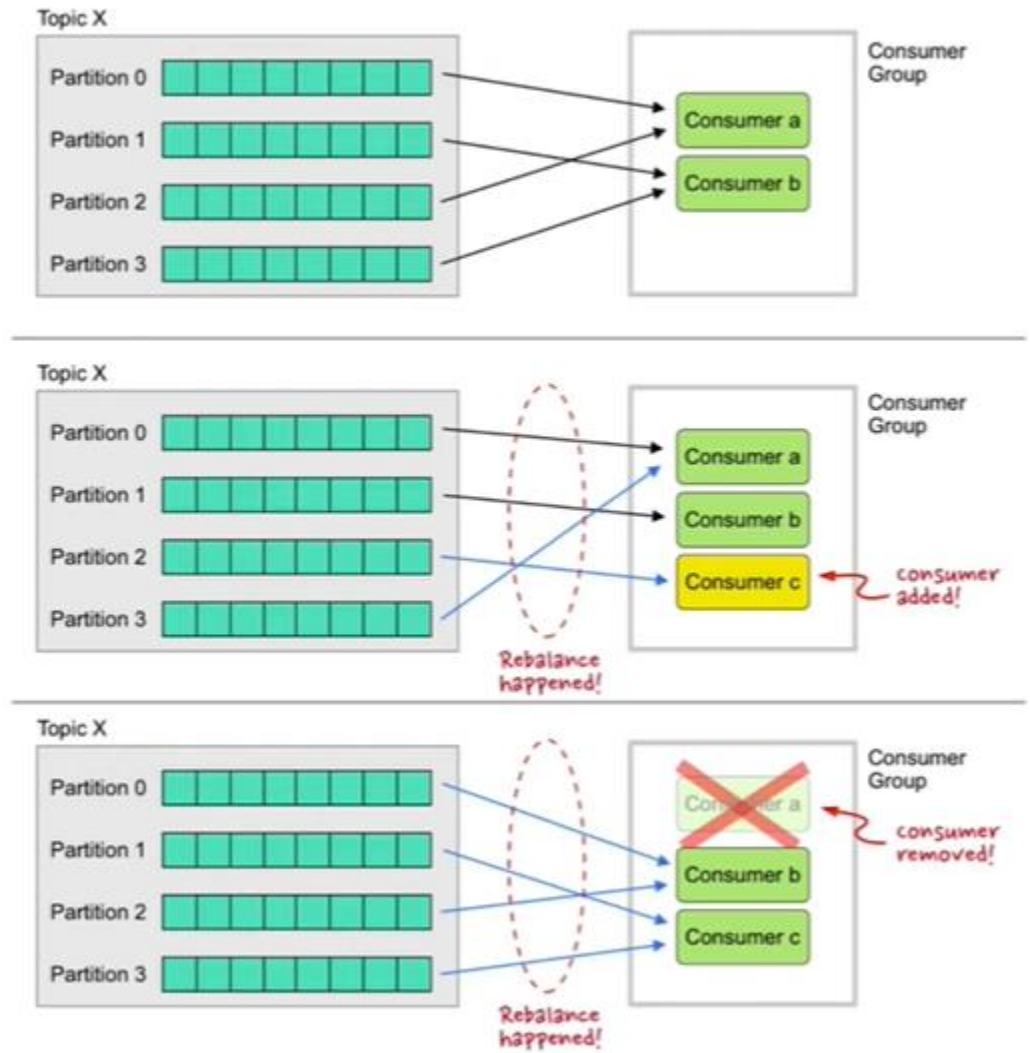
Use Cases

- Tracking ad views
- Processing financial transactions
- Stream processing

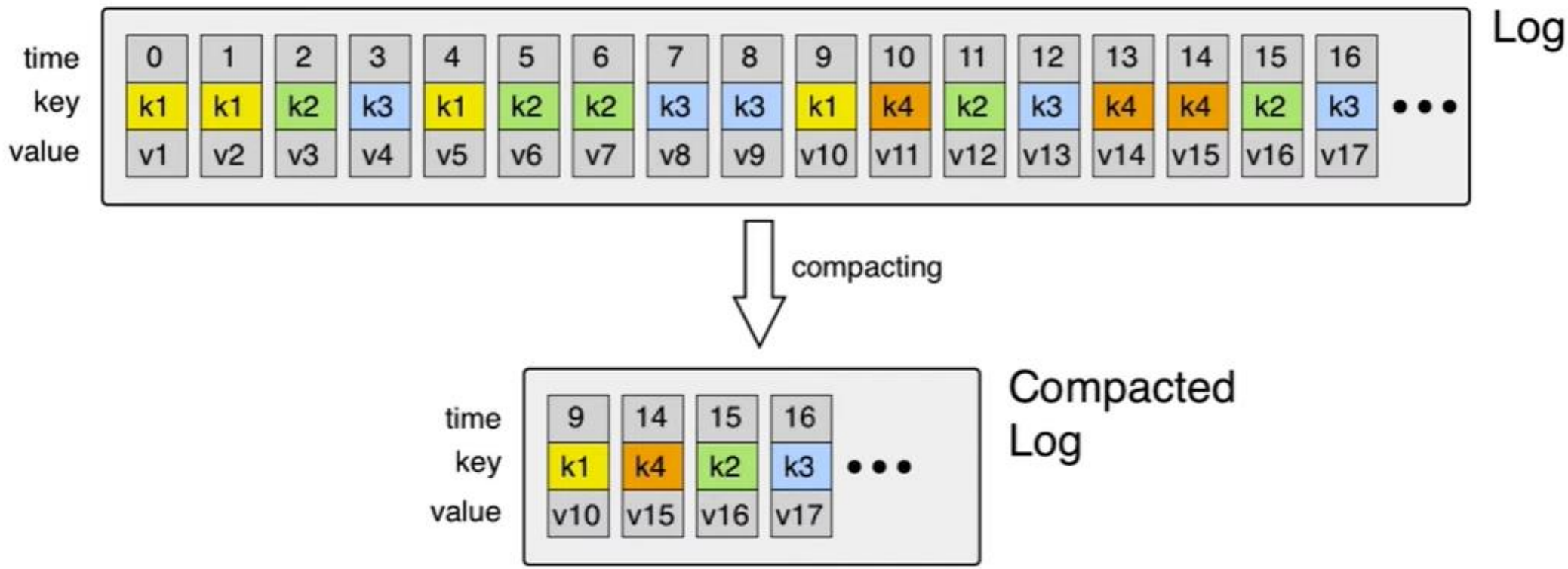
Consumer Groups



Consumer Rebalances

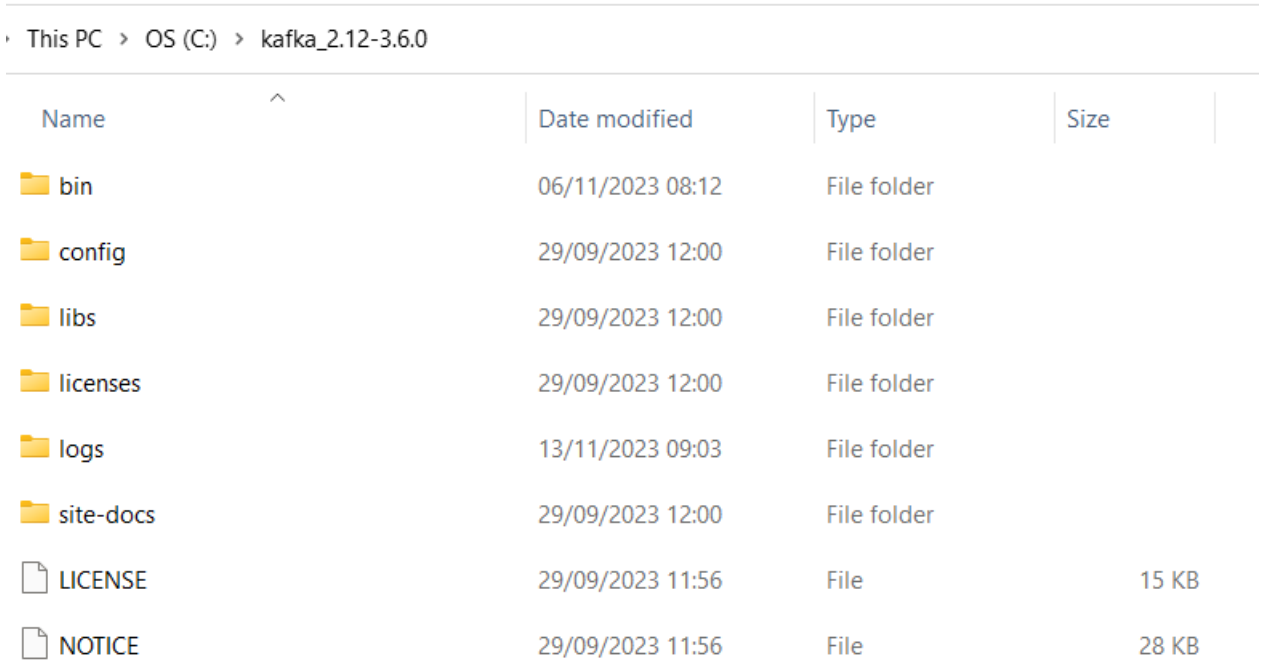


Compacted Topics



Tutorial 1: Kafka installation on Window

- Download Kafka (Binary) from https://kafka.apache.org/
- Unzip the download file move it to C:\ drive



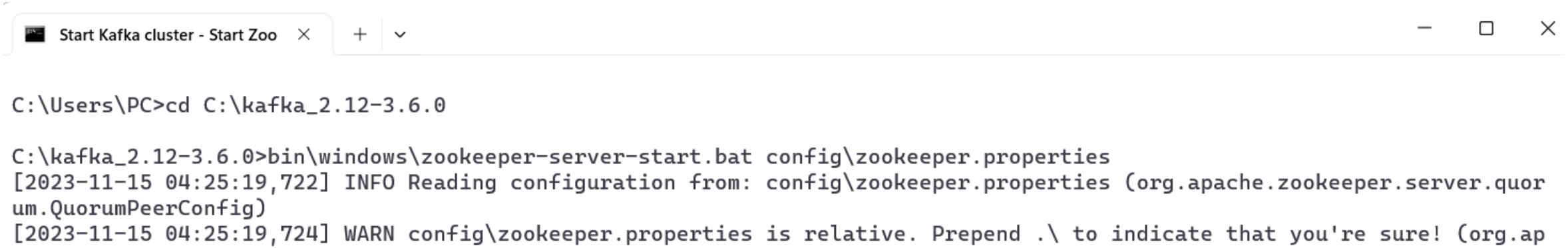
3.6.0

- Released Oct 10, 2023
- Release Notes
- Source download: kafka-3.6.0-src.tgz (asc, sha512)
- Binary downloads:
  - Scala 2.12 - kafka_2.12-3.6.0.tgz (asc, sha512)
  - Scala 2.13 - kafka_2.13-3.6.0.tgz (asc, sha512)

Tutorial 2: Run Apache Kafka on Windows

- Start the Kafka cluster
  - Run the following command to start ZooKeeper:

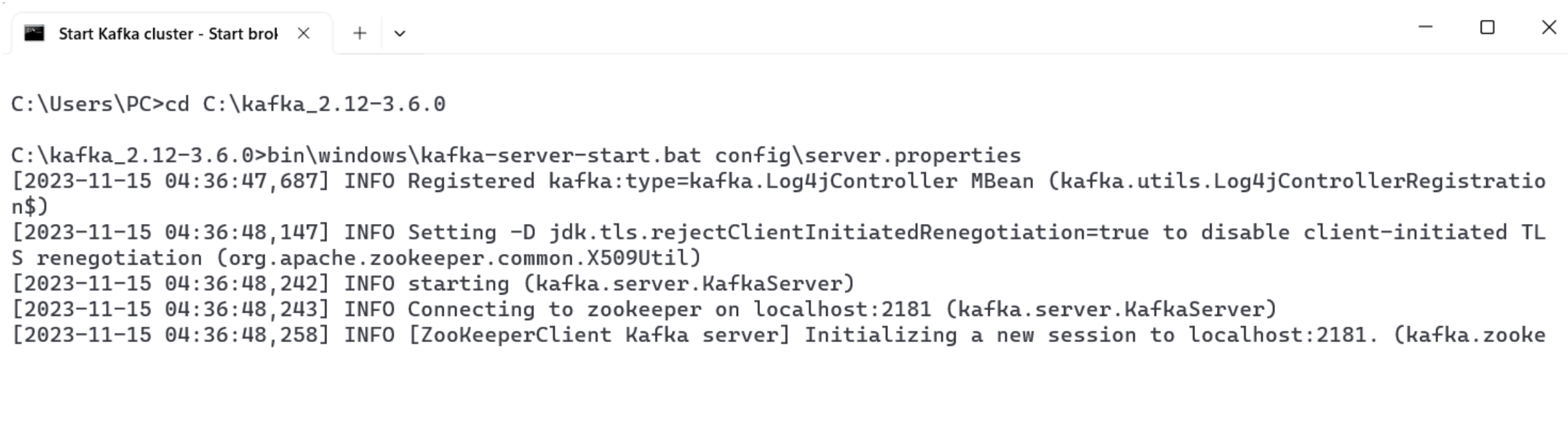
```
cd C:\kafka_2.12-3.6.0
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
```



Run Apache Kafka on Windows

- Start the Kafka cluster
  - Run the following command to start the Kafka broker:

```
cd C:\kafka_2.12-3.6.0
bin\windows\kafka-server-start.bat config\server.properties
```



Run Apache Kafka on Windows

- Produce and consume some messages
    - Run the kafka-topics command to create a Kafka topic named TestTopic

```
bin\windows\kafka-topics.bat --create --topic TestTopic --bootstrap-server localhost:9092
```

    - Let’s create another topic named NewTopic

```
bin\windows\kafka-topics.bat --create --topic NewTopic --bootstrap-server localhost:9092
```

    - Let’s show list of created topics

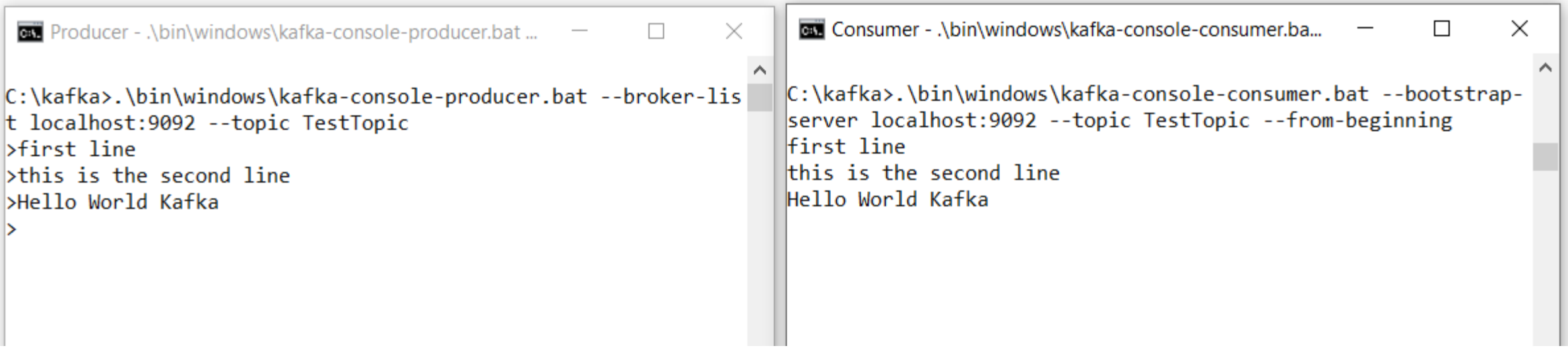
```
bin\windows\kafka-topics.bat --list --bootstrap-server localhost:9092
```

Run Apache Kafka on Windows

- Produce and consume some messages
    - Run the producer and consumer on separate Command Prompt:

bin\windows\kafka-console-producer.bat --topic TestTopic --bootstrap-server localhost:9092

bin\windows\kafka-console-consumer.bat --topic TestTopic --from-beginning --bootstrap-server localhost:9092

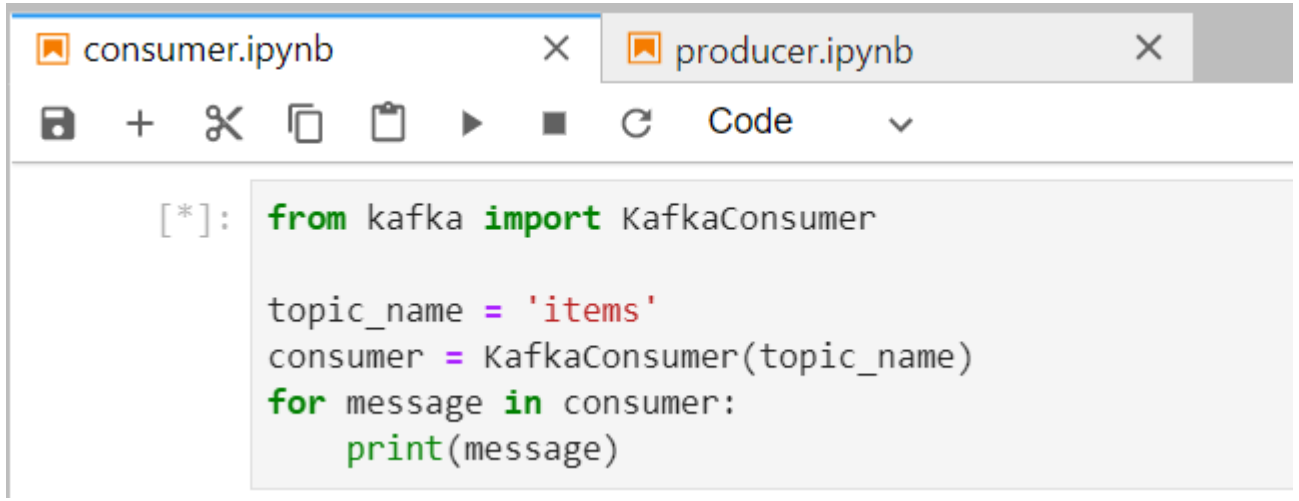


Tutorial 3: Kafka Python client

- https://kafka-python.readthedocs.io/en/master/index.html
- Install Kafka-Python
  - pip install kafka-python
- Start Zookeeper server and Kafka broker
  - Zookeeper is running default on localhost:2181 and Kafka on localhost:9092

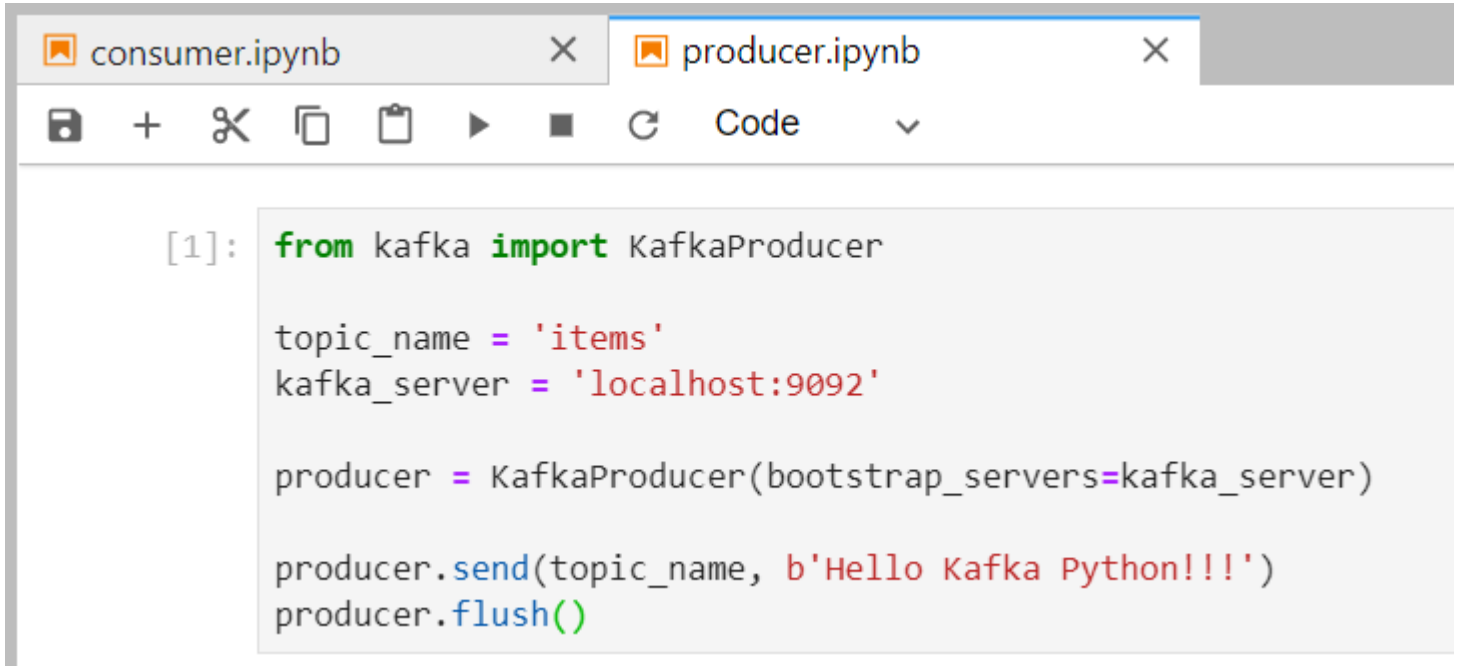
Kafka-Python

- Run consumer code



Kafka-Python

- Run producer code



Kafka-Python

- Check the result

consumer.ipynb producer.ipynb

Code Python 3

```
[*]: from kafka import KafkaConsumer

topic_name = 'items'
consumer = KafkaConsumer(topic_name)
for message in consumer:
    print(message)
```

```
ConsumerRecord(topic='items', partition=0, offset=0, timestamp=1630946348692, timestamp_type=0, key=None, value=b'Hello Kafka P
ython!!!', headers=[], checksum=None, serialized_key_size=-1, serialized_value_size=21, serialized_header_size=-1)
```

Tutorial 4: Run Kafka on Colab

- Download Kafka and unzip

```
!curl -sSOL https://downloads.apache.org/kafka/3.3.1/kafka_2.13-3.3.1.tgz
!tar -xzf kafka_2.13-3.3.1.tgz
```

- Start zookeeper server and kafka server

```
!./kafka_2.13-3.3.1/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-3.3.1/config/zookeeper.properties
!./kafka_2.13-3.3.1/bin/kafka-server-start.sh -daemon ./kafka_2.13-3.3.1/config/server.properties
```

- Create a topic

```
!./kafka_2.13-3.3.1/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic TestTopic
```

Run Kafka on Colab

- Describe the created topic

```
!./kafka_2.13-3.3.1/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic TestTopic
```

- Write some event in the topic

```
!./kafka_2.13-3.3.1/bin/kafka-console-producer.sh --topic TestTopic --bootstrap-server 127.0.0.1:9092
```

- Read the event

```
!./kafka_2.13-3.3.1/bin/kafka-console-consumer.sh --topic TestTopic --from-beginning --bootstrap-server 127.0.0.1:9092
```

Run Kafka on Colab

- You can run cells sequentially and get the result (not really streaming)

```
[12] !./kafka_2.13-3.3.1/bin/kafka-console-producer.sh --topic TestTopic --bootstrap-server 127.0.0.1:9092
```

```
>hello
>second line
>
```

```
[13] !./kafka_2.13-3.3.1/bin/kafka-console-consumer.sh --topic TestTopic --from-beginning --bootstrap-server 127.0.0.1:9092
```

```
test input from kafka producer
hello
second line
```

Run Kafka on Colab

- Or you can run producer and consumer parallelly in different terminals

```
[15] !pip install colab-xterm
     %load_ext colabxterm
```

- Open terminal using Xterm and run consumer (it will be empty at first)

```
[21] %xterm
```

```
Launching Xterm...
```

```
/content# ./kafka_2.13-3.3.1/bin/kafka-console-consumer.sh --topic TestTopic --from-beginning --bootstrap-server 127.0.0.1:9092
second line in terminal
third line in terminal
fourd line in terminal
fifth line in terminal
```

- Open terminal using Xterm and run producer, write some lines and they will appear on the consumer's terminal

```
[20] %xterm
```

```
Launching Xterm...
```

```
/content# ./kafka_2.13-3.3.1/bin/kafka-console-producer.sh --topic TestTopic --bootstrap-server 127.0.0.1:9092
>second line in terminal
>third line in terminal
fourd line in terminal
>fifth line in terminal
>
```

Run Kafka on Colab

- Use kafka-python on Colab

```
[30] !pip install kafka-python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kafka-python in /usr/local/lib/python3.7/dist-packages (2.0.2)
```

```
[27] from kafka import KafkaProducer

topic_name = 'TestTopic'
kafka_server = 'localhost:9092'

producer = KafkaProducer(bootstrap_servers=kafka_server)

producer.send(topic_name, b'Hello Kafka Python!!!')
producer.flush()
```

```
from kafka import KafkaConsumer

topic_name = 'TestTopic'
consumer = KafkaConsumer(topic_name,bootstrap_servers='127.0.0.1:9092',auto_offset_reset='earliest')
print(consumer)
for message in consumer:
    print(message)
```

```
WARNING:kafka.coordinator.consumer:group_id is None: disabling auto-commit.
<kafka.consumer.group.KafkaConsumer object at 0x7fb44a7e2350>
ConsumerRecord(topic='TestTopic', partition=0, offset=0, timestamp=1664896978186, timestamp_type=0, key=None, value=b'ww', headers=[], checksum=N
ConsumerRecord(topic='TestTopic', partition=0, offset=1, timestamp=1664896985895, timestamp_type=0, key=None, value=b'cc', headers=[], checksum=N
```

Tutorial 5: Test Kafka and Spark Structure Streaming on Colab

- Start kafka

```
[ ]  !curl -sSOL https://downloads.apache.org/kafka/3.3.1/kafka_2.13-3.3.1.tgz
     !tar -xzf kafka_2.13-3.3.1.tgz
```

```
[ ]  !./kafka_2.13-3.3.1/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-3.3.1/config/zookeeper.properties
     !./kafka_2.13-3.3.1/bin/kafka-server-start.sh -daemon ./kafka_2.13-3.3.1/config/server.properties
```

- Install PySpark

```
#currently, 3.3.0 is the latest version. However, you still need to specify this.
!pip install pyspark==3.3.0

from pyspark.sql import SparkSession
scala_version = '2.13'
spark_version = '3.3.0'
packages = [  f'org.apache.spark:spark-sql-kafka-0-10_{scala_version}:{spark_version}' , 'org.apache.kafka:kafka-clients:3.3.1' ]
spark = SparkSession.builder.master("local").appName("kafka-example").config("spark.jars.packages", ",".join(packages)).getOrCreate()
spark
```

- Install kafka-python

```
!pip install kafka-python


from kafka import KafkaProducer
from json import dumps
topic_name = 'Number'
kafka_server = 'localhost:9092'
producer = KafkaProducer(bootstrap_servers=kafka_server,value_serializer = lambda x:dumps(x).encode('utf-8'))
for e in range(1000):
    data = {'number' : e}
    producer.send(topic_name, value=data)
producer.flush()
```

- You can test if the topic is sent sucessfully

```
[15] !./kafka_2.13-3.3.1/bin/kafka-console-consumer.sh --topic Number --from-beginning --bootstrap-server 127.0.0.1:9092
{"number": 937}
{"number": 938}
{"number": 939}
{"number": 940}
{"number": 941}
{"number": 942}
{"number": 943}
```

- Create datafram from Kafka topic

```
producer.flush()

kafkaDf = spark.read.format("kafka")\
  .option("kafka.bootstrap.servers", "localhost:9092")\
  .option("subscribe", topic_name)\
  .option("startingOffsets", "earliest")\
  .load()

kafkaDf.show()
```

- Show the dataframe in a formatted way

```
from pyspark.sql.functions import col, concat, lit

kafkaDf.select(
    concat(col("topic"), lit(':'),
col("partition").cast("string")).alias("topic_partition"),col("offset"),col("value").cast("string")
).show()
```

```
+---------------+------+-------------+
|topic_partition|offset|        value|
+---------------+------+-------------+
|       Number:0|     0| {"number": 0}|
|       Number:0|     1| {"number": 1}|
|       Number:0|     2| {"number": 2}|
|       Number:0|     3| {"number": 3}|
|       Number:0|     4| {"number": 4}|
|       Number:0|     5| {"number": 5}|
|       Number:0|     6| {"number": 6}|
```

Tutorial 6: Test Kafka and Spark Structure Streaming on **Local**

- Step1: Start Kafka cluster using Terminal

- Step 2: Run KafkaProducer in Jupyter Notebook

```
from kafka import KafkaProducer
from json import dumps
from time import sleep

topic_name = 'RandomNumber'
kafka_server = 'localhost:9092'

producer = KafkaProducer(bootstrap_servers=kafka_server,value_serializer = lambda x:dumps(x).encode('utf-8'))

for e in range(1000):
    data = {'number' : e}
    producer.send(topic_name, value=data)
    print(str(data) + " sent")
    sleep(5)

producer.flush()
```

- Open another Jupyter Notebook

[1]:
```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession

scala_version = '2.12'  # your scala version
spark_version = '3.0.1' # your spark version
packages = [
    f'org.apache.spark:spark-sql-kafka-0-10_{scala_version}:{spark_version}',
    'org.apache.kafka:kafka-clients:2.8.0' #your kafka version
]
spark = SparkSession.builder.master("local").appName("kafka-example").config("spark.jars.packages", ",".join(packages)).getOrCreate()
spark
```

[1]:

SparkSession - in-memory

SparkContext

Spark UI

Version	`v3.0.1`
Master	`local`
AppName	`kafka-example`

- You will reading data from Kafka in two ways:
  - Batch query
  - Streaming query
- See more at https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html

Creating a Kafka Source for Batch Queries

- Create dataframe from Kafka data

```
topic_name = 'RandomNumber'

kafka_server = 'localhost:9092'

kafkaDf = spark.read.format("kafka").option("kafka.bootstrap.servers", kafka_server).option("subscribe", topic_name).option("startingOffsets", "earliest").load()
```

- Show data (converting dataframe to pandas for cleaner view of data)

```
kafkaDf.toPandas()
```

	key	value	topic	partition	offset	timestamp	timestampType
0	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	0	2022-10-05 15:18:37.301	0
1	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	1	2022-10-05 15:18:42.314	0
2	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	2	2022-10-05 15:18:47.327	0
3	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	3	2022-10-05 15:18:52.341	0
4	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	4	2022-10-05 15:18:57.352	0
5	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	5	2022-10-05 15:19:02.363	0
6	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	6	2022-10-05 15:19:07.376	0
7	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	7	2022-10-05 15:19:12.395	0
8	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	8	2022-10-05 15:19:17.411	0
9	None	[123, 34, 110, 117, 109, 98, 101, 114, 34, 58,...	RandomNumber	0	9	2022-10-05 15:19:22.417	0

- Show streaming data using for loop

```
batchDF = kafkaDf.select(col('topic'),col('offset'),col('value').cast('string').substr(12,1).alias('rand_number'))
from time import sleep
from IPython.display import display, clear_output
for x in range(0, 2000):
    try:
        print("Showing live view refreshed every 5 seconds")
        print(f"Seconds passed: {x*5}")
        display(batchDF.toPandas())
        sleep(5)
        clear_output(wait=True)
    except KeyboardInterrupt:
        print("break")
        break
print("Live view ended...")
```

```
Showing live view refreshed every 5 seconds
Seconds passed: 10
```

	topic	offset	rand_number
0	RandomNumber	0	2
1	RandomNumber	1	4
2	RandomNumber	2	7
3	RandomNumber	3	7
4	RandomNumber	4	3
5	RandomNumber	5	7
6	RandomNumber	6	6

- Perform some data aggregation and show live results

```
batchCountDF = batchDF.groupBy('rand_number').count()
for x in range(0, 2000):
    try:
        print("Showing live view refreshed every 5 seconds")
        print(f"Seconds passed: {x*5}")
        display(batchCountDF.toPandas())
        sleep(5)
        clear_output(wait=True)
    except KeyboardInterrupt:
        print("break")
        break
print("Live view ended...")
```

```
Showing live view refreshed every 5 seconds
Seconds passed: 5
```

	rand_number	count
0	7	5
1	3	2
2	8	1
3	0	1
4	5	1
5	6	3
6	9	1
7	1	1
8	4	1
9	2	1

```
break
Live view ended...
```

Creating a Kafka Source for Streaming Queries

- Create Streaming dataframe from Kafka

```
streamRawDf = spark.readStream.format("kafka").option("kafka.bootstrap.servers", kafka_server).option("subscribe", topic_name).load()
streamDF = streamRawDf.select(col('topic'),col('offset'),col('value').cast('string').substr(12,1).alias('rand_number'))
checkEvenDF = streamDF.withColumn('Is_Even',col('rand_number').cast('int') % 2 == 0 )
```

- Write stream

```
from random import randint
randNum=str(randint(0,10000))
q1name = "queryNumber"+randNum
q2name = "queryCheckEven"+randNum

stream_writer1 = (streamDF.writeStream.queryName(q1name).trigger(processingTime="5 seconds").outputMode("append").format("memory"))
stream_writer2 = (checkEvenDF.writeStream.queryName(q2name).trigger(processingTime="5 seconds").outputMode("append").format("memory"))

query1 = stream_writer1.start()
query2 = stream_writer2.start()
```

- View streaming result

```
for x in range(0, 2000):
    try:
        print("Showing live view refreshed every 5 seconds")
        print(f"Seconds passed: {x*5}")
        result1 = spark.sql(f"SELECT * from {query1.name}")
        result2 = spark.sql(f"SELECT * from {query2.name}")
        display(result1.toPandas())
        display(result2.toPandas())
        sleep(5)
        clear_output(wait=True)
    except KeyboardInterrupt:
        print("break")
        break
print("Live view ended...")
```

```
Showing live view refreshed every 5 seconds
Seconds passed: 20
```

	topic	offset	rand_number
0	RandomNumber	21	0
1	RandomNumber	22	8
2	RandomNumber	23	2
3	RandomNumber	24	1

	topic	offset	rand_number	Is_Even
0	RandomNumber	21	0	True
1	RandomNumber	22	8	True
2	RandomNumber	23	2	True
3	RandomNumber	24	1	False

```
break
Live view ended...
```

Tutorial 7: Kafka and MongoDB on Window

number_producer.ipynb ● number_consumer.ipynb ×

Code

[*]:
```
from time import sleep
from json import dumps
from kafka import KafkaProducer

topic_name = 'numtest'
kafka_server = 'localhost:9092'

producer = KafkaProducer(bootstrap_servers=kafka_server,value_serializer = lambda x:dumps(x).encode('utf-8'))

for e in range(1000):
    data = {'number' : e}
    producer.send(topic_name, value=data)
    sleep(5)
```

[ ]:

number_producer.ipynb | number_consumer.ipynb

Code | Python 3

[*]:
```
from kafka import KafkaConsumer
from pymongo import MongoClient
from json import loads

topic_name = 'numtest'
consumer = KafkaConsumer(
    topic_name,
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group',
    value_deserializer=lambda x: loads(x.decode('utf-8')))

client = MongoClient('localhost:27017')
collection = client.numtest.numtest

for message in consumer:
    message = message.value
    collection.insert_one(message)
    print('{} added to {}'.format(message, collection))
```

```
{'number': 0, '_id': ObjectId('61365919e8fecaa7b75f57e0')} added to Collection(Database(MongoClient(hos
t=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'numtest'), 'numtest')
{'number': 1, '_id': ObjectId('61365919e8fecaa7b75f57e1')} added to Collection(Database(MongoClient(hos
t=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'numtest'), 'numtest')
{'number': 2, '_id': ObjectId('61365919e8fecaa7b75f57e2')} added to Collection(Database(MongoClient(hos
t=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'numtest'), 'numtest')
{'number': 3, '_id': ObjectId('61365919e8fecaa7b75f57e3')} added to Collection(Database(MongoClient(hos
t=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'numtest'), 'numtest')
```

MongoDB Compass - localhost:27017/numtest.numtest

Connect View Collection Help

Local

4 DBS 2 COLLECTIONS

FAVORITE

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 5.0.2 Community

Filter your data

- admin
- config
- local
- numtest
  - numtest

numtest.numtest
Documents

numtest.numtest

DOCUMENTS 34 TOTAL SIZE 1.1KB AVG. SIZE 34B

INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents | Aggregations | Schema | Explain Plan | Indexes | Validation

FILTER { field: 'value' } OPTIONS FIND RESET

ADD DATA VIEW

Displaying documents 1 - 20 of 34 REFRESH

```
_id: ObjectId("61365919e8fecaa7b75f57e0")
number: 0

_id: ObjectId("61365919e8fecaa7b75f57e1")
number: 1

_id: ObjectId("61365919e8fecaa7b75f57e2")
number: 2

_id: ObjectId("61365919e8fecaa7b75f57e3")
number: 3

_id: ObjectId("61365919e8fecaa7b75f57e4")
number: 4

_id: ObjectId("61365919e8fecaa7b75f57e5")
number: 5
```

Tutorial 8

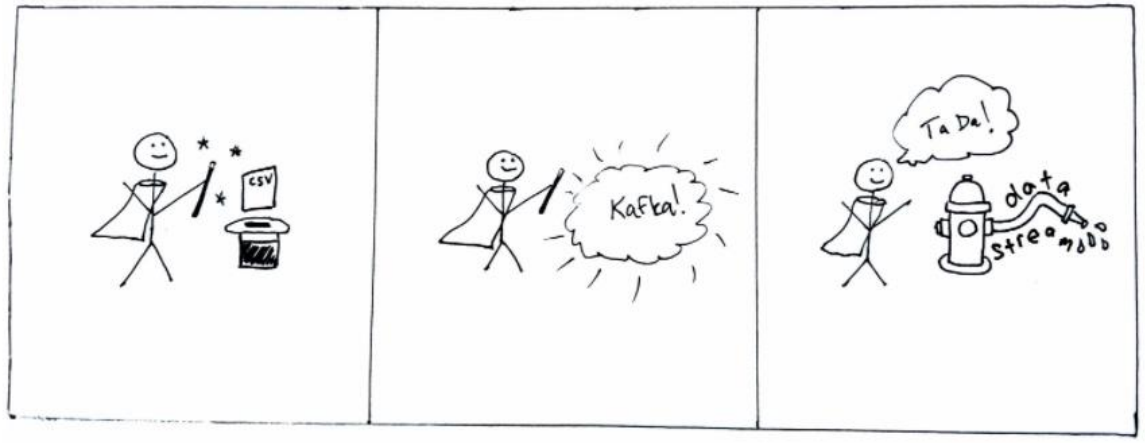
https://towardsdatascience.com/make-a-mock-real-time-stream-of-data-with-python-and-kafka-7e5e23123582

towardsdatascience.com/make-a-mock-real-time-stream-of-data-with-python-and-kafka-7e5e23123582

Make a mock “real-time” data stream with Python and Kafka

A Dockerized tutorial with everything you need to turn a .csv file of timestamped data into a Kafka stream

Maria Patterson Aug 30 · 5 min read ★



Tutorial 8: streaming from CSV

- sendStream.py

```
while True:

    try:

        if firstline is True:
            line1 = next(rdr, None)
            timestamp, value = line1[0], float(line1[1])
            # Convert csv columns to key value pair
            result = {}
            result[timestamp] = value
            # Convert dict to json as message format
            jresult = json.dumps(result)
            firstline = False

            producer.produce(topic, key=p_key, value=jresult, callback=acked)

        else:
            line = next(rdr, None)
            d1 = parse(timestamp)
            d2 = parse(line[0])
            diff = ((d2 - d1).total_seconds())/args.speed
            time.sleep(diff)
            timestamp, value = line[0], float(line[1])
            result = {}
            result[timestamp] = value
            jresult = json.dumps(result)

            producer.produce(topic, key=p_key, value=jresult, callback=acked)

        producer.flush()
```

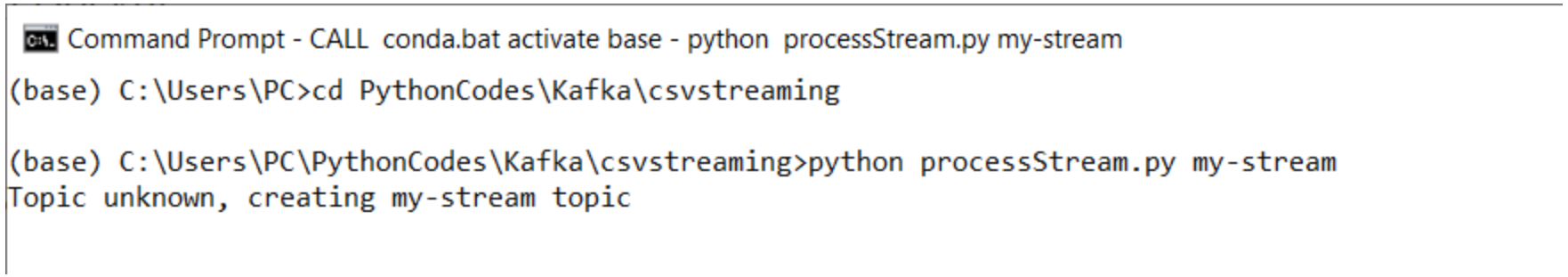
Tutorial 8: streaming from CSV

- processStream.py

```
def main():
    parser = argparse.ArgumentParser(description=__doc__)
    parser.add_argument('topic', type=str, help='Name of the Kafka topic to stream.')

    args = parser.parse_args()

    conf = {'bootstrap.servers': 'localhost:9092','default.topic.config': {'auto.offset.reset': 'smallest'},
            'group.id': socket.gethostname()}

    consumer = Consumer(conf)

    running = True

    try:
        while running:
            consumer.subscribe([args.topic])

            msg = consumer.poll(1)
```

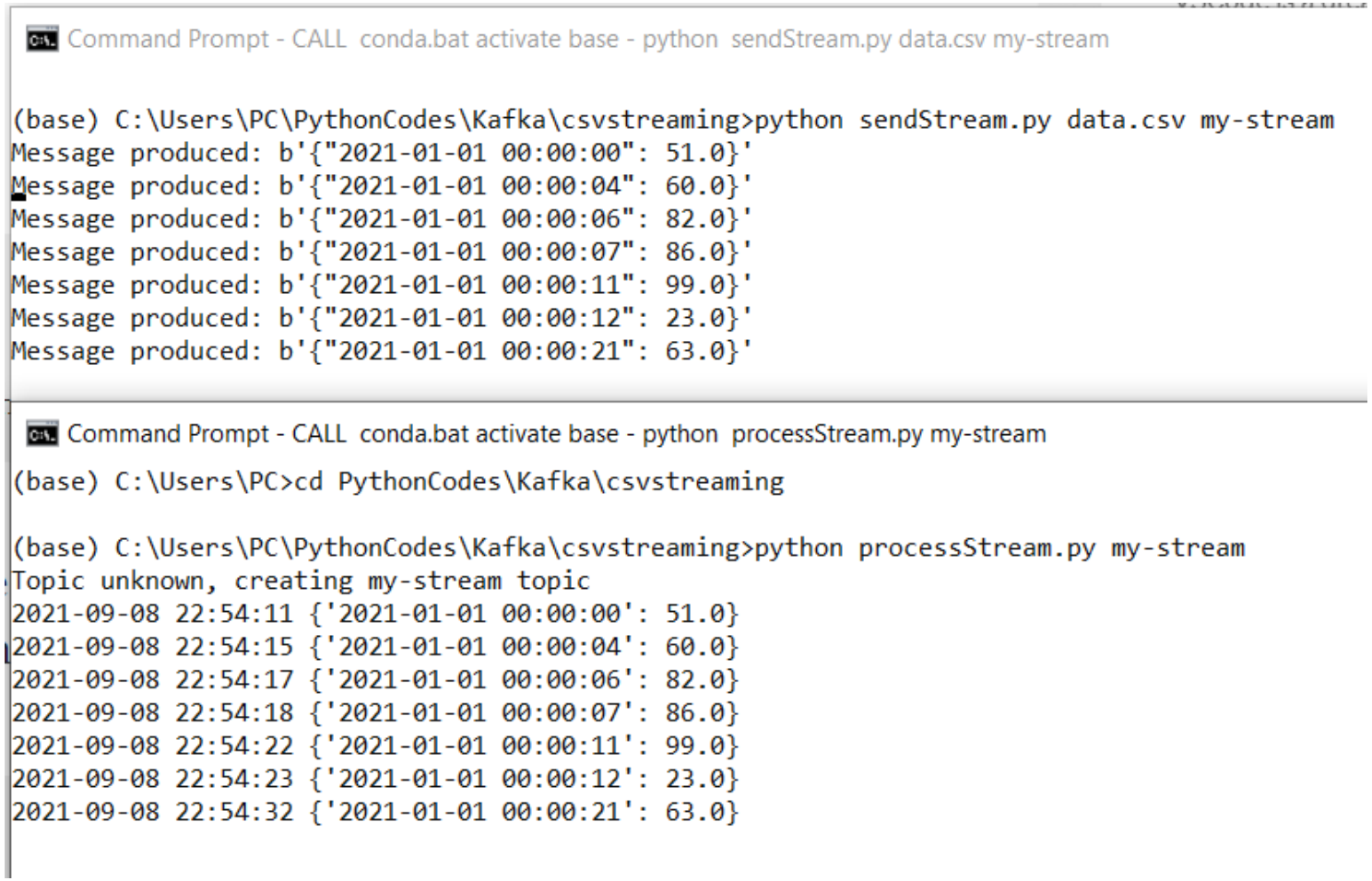
Tutorial 8: streaming from CSV

Start the consumer



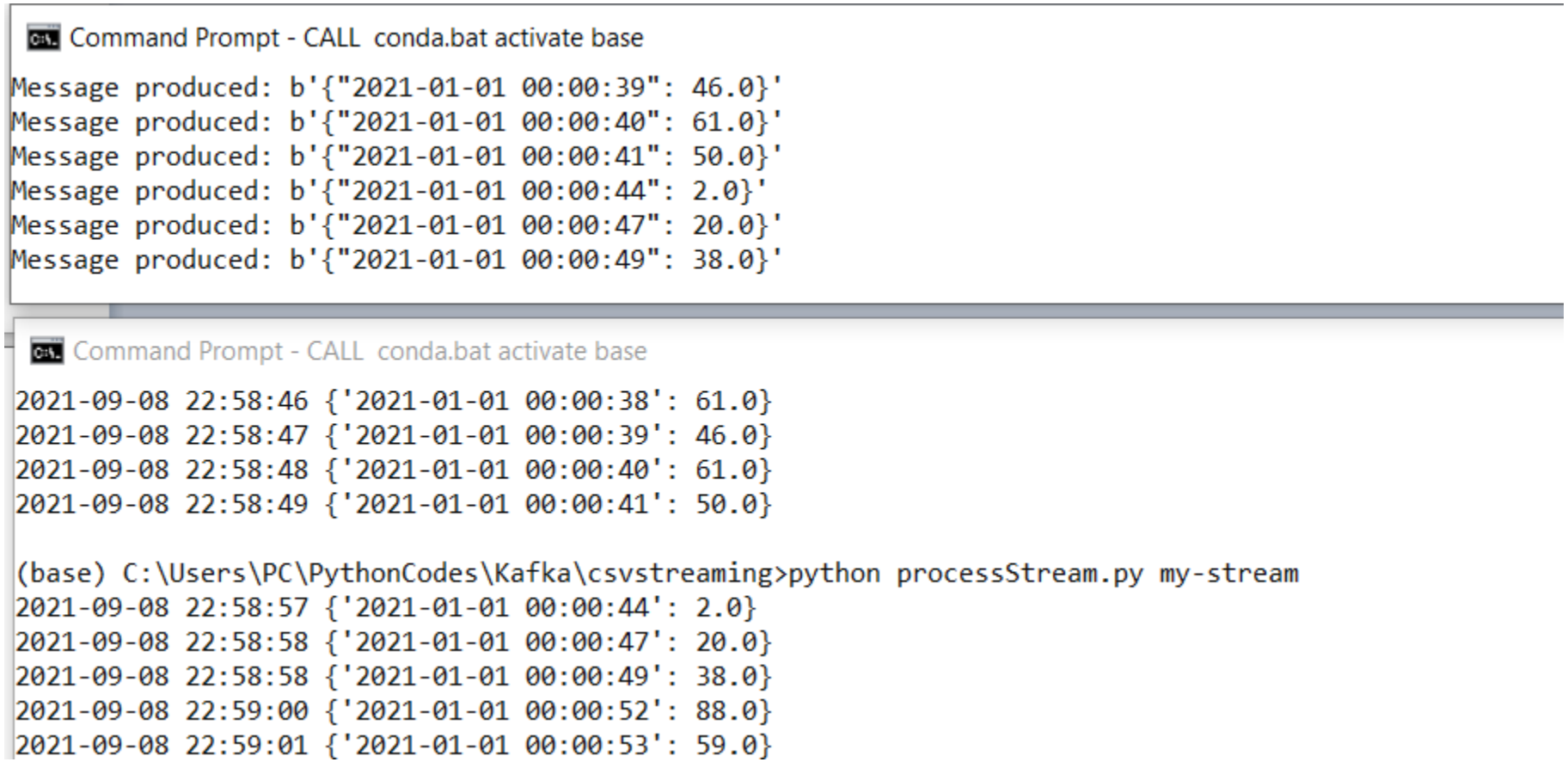
Tutorial 8: streaming from CSV

Start the producer

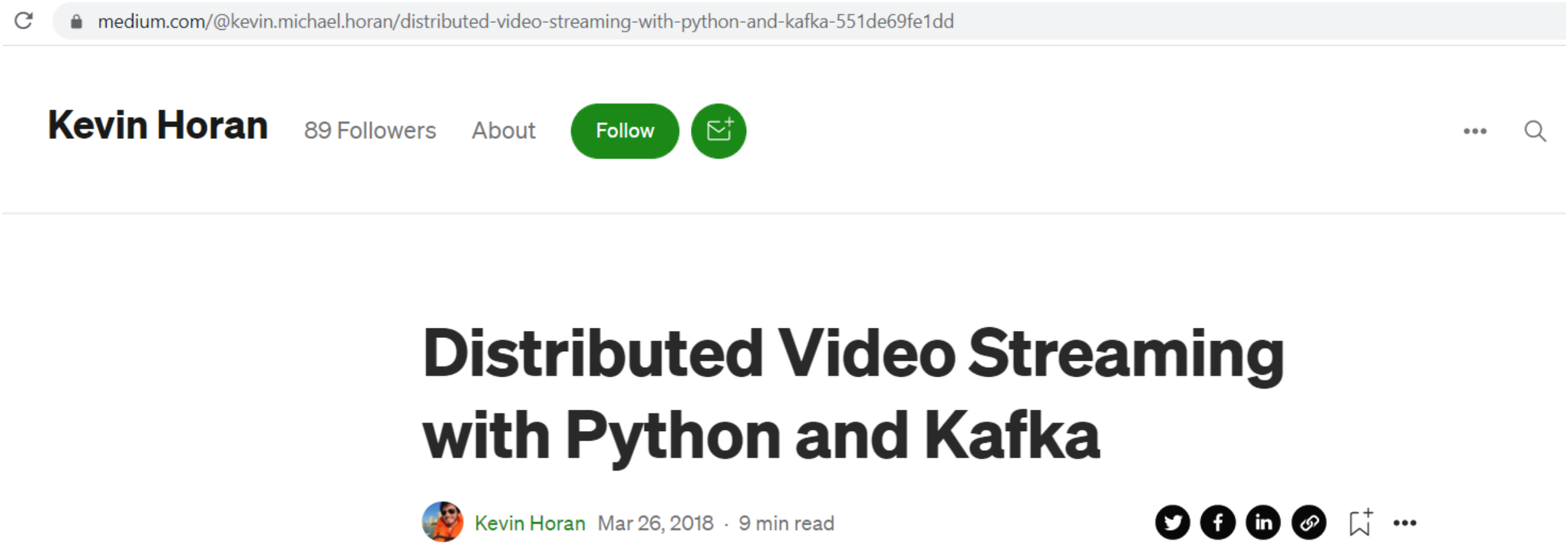


Tutorial 8: streaming from CSV

If you terminate the consumer and then restart it, the streaming will be resumed from where it stop



Tutorial 9



https://medium.com/@kevin.michael.horan/distributed-video-streaming-with-python-and-kafka-551de69fe1dd

Tutorial 9: Video streaming using Kafka

- Producer.py

```
def publish_video(video_file):

    # Start up producer
    producer = KafkaProducer(bootstrap_servers='localhost:9092')

    # Open file
    video = cv2.VideoCapture(video_file)

    while(video.isOpened()):
        success, frame = video.read()

        # Convert image to png
        ret, buffer = cv2.imencode('.jpg', frame)

        # Convert to bytes and send to kafka
        producer.send(topic, buffer.tobytes())

        time.sleep(0.2)
```

Tutorial 9: Video streaming using Kafka

- Producer.py

```
def publish_camera():

    # Start up producer
    producer = KafkaProducer(bootstrap_servers='localhost:9092')

    camera = cv2.VideoCapture(0)
    try:
        while(True):
            success, frame = camera.read()

            ret, buffer = cv2.imencode('.jpg', frame)
            producer.send(topic, buffer.tobytes())

            time.sleep(0.2)

    except:
        print("\nExiting.")
        sys.exit(1)
```

Tutorial 9: Video streaming using Kafka

- Producer.py

```
if __name__ == '__main__':
    """
    Producer will publish to Kafka Server a video file given as a system arg.
    Otherwise it will default by streaming webcam feed.
    """
    if(len(sys.argv) > 1):
        video_path = sys.argv[1]
        publish_video(video_path)
    else:
        print("publishing feed!")
        publish_camera()
```

Tutorial 9: Video streaming using Kafka

- consumer.py

```
# Fire up the Kafka Consumer
topic = "distributed-video1"

consumer = KafkaConsumer(topic,bootstrap_servers=['localhost:9092'])

# Set the consumer in a Flask App
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video_feed', methods=['GET'])
def video_feed():
    return Response( get_video_stream(), mimetype='multipart/x-mixed-replace; boundary=frame')

def get_video_stream():
    for msg in consumer:
        yield (b'--frame\r\n' b'Content-Type: image/jpg\r\n\r\n' + msg.value + b'\r\n\r\n')

if __name__ == '__main__':
    app.run(debug=True)
```

Tutorial 9: Video streaming using Kafka

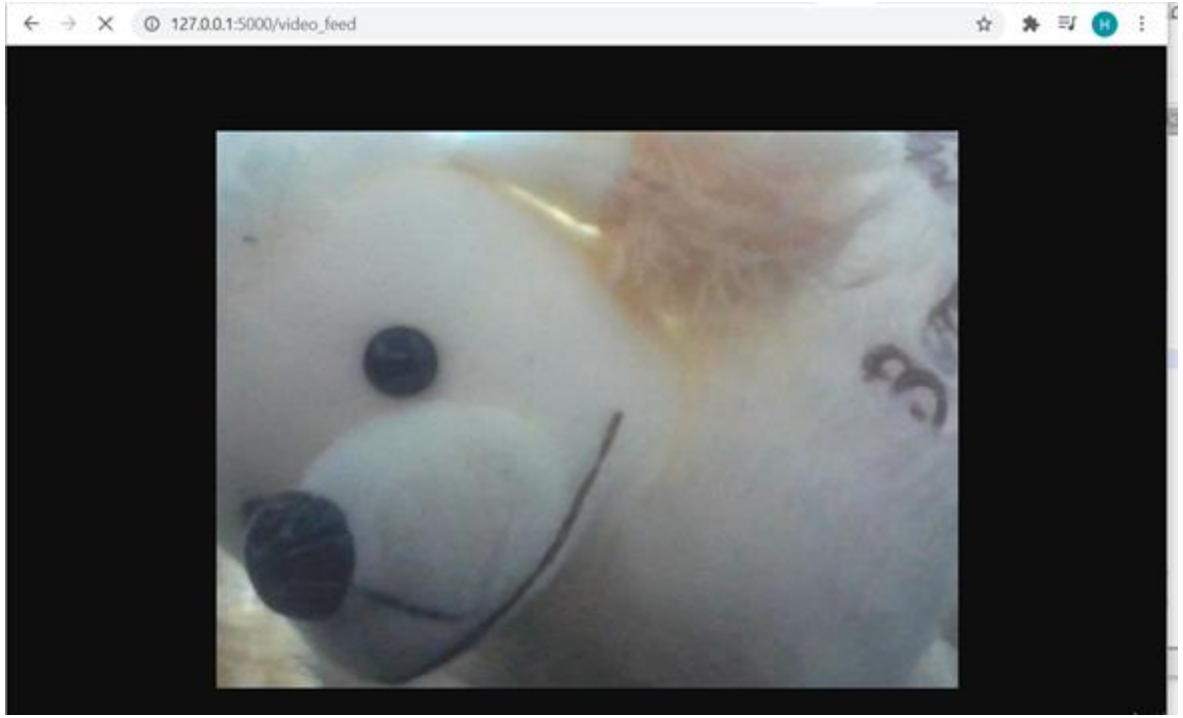
- Run consumer.py

```
(base) C:\Users\PC\PythonCodes\Kafka\videostreaming>python consumer.py
 * Serving Flask app "consumer" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 244-614-898
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Tutorial 9: Video streaming using Kafka

- Stream video from webcam

```
(base) C:\Users\PC\PythonCodes\Kafka\videostreaming>python producer.py
publishing feed!
```



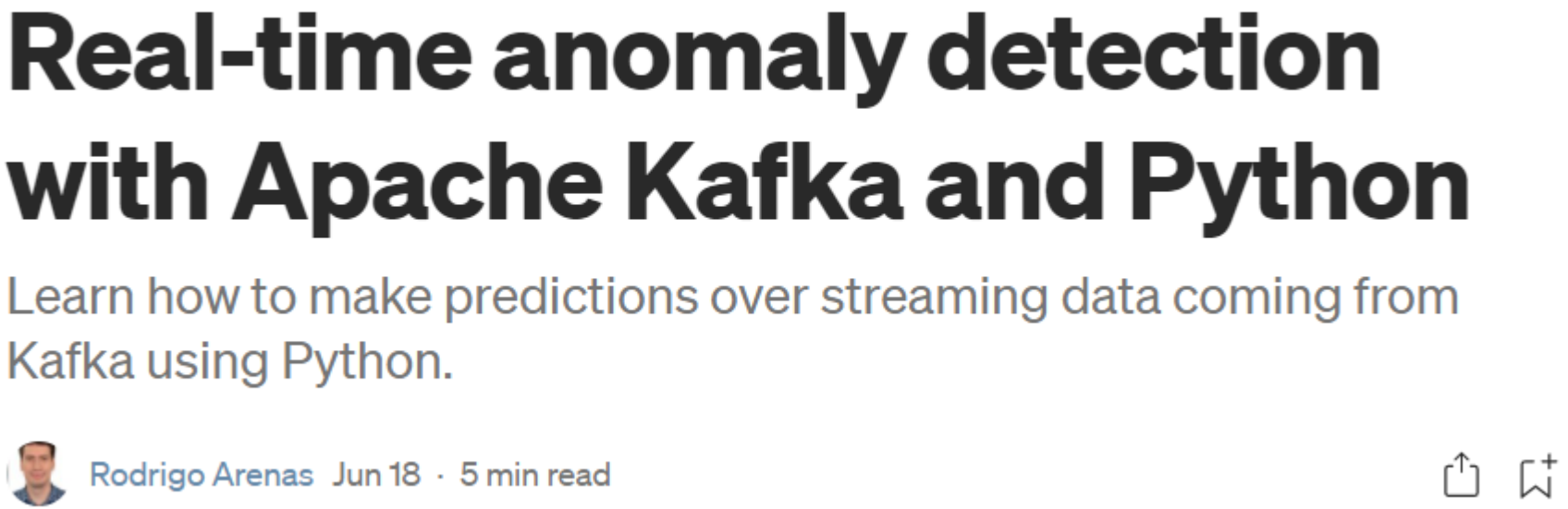
Tutorial 9: Video streaming using Kafka

- Stream a video entitled Countdow1.mp4

```
(base) C:\Users\PC\PythonCodes\Kafka\videostreaming>python producer.py videos/Countdown1.mp4
publishing video...
```

Tutorial 10

https://towardsdatascience.com/real-time-anomaly-detection-with-apache-kafka-and-python-3a40281c01c9



Tutorial 10: real-time anomaly detection

```
C:\kafka>.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic transaction
Created topic transaction.

C:\kafka>.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic anomalies
Created topic anomalies.
```

Tutorial 10: real-time anomaly detection

- Producer.py

```
if producer is not None:
    while True:
        # Generate some abnormal observations
        if random.random() <= OUTLIERS_GENERATION_PROBABILITY:
            X_test = np.random.uniform(low=-4, high=4, size=(1, 2))
        else:
            X = 0.3 * np.random.randn(1, 2)
            X_test = (X + np.random.choice(a=[2, -2], size=1, p=[0.5, 0.5]))

        X_test = np.round(X_test, 3).tolist()

        current_time = datetime.utcnow().isoformat()

        record = {"id": _id, "data": X_test, "current_time": current_time}
        record = json.dumps(record).encode("utf-8")
        print('produce message')
        print(record)
        producer.produce(topic=TRANSACTIONS_TOPIC,
                         value=record)
        producer.flush()
        _id += 1
        time.sleep(DELAY)
```

Tutorial 10: real-time anomaly detection

- train.py

```python
import random
from joblib import dump

import numpy as np
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(42)

# Generate train data
X = 0.3 * rng.randn(500, 2)
X_train = np.r_[X + 2, X - 2]
X_train = np.round(X_train, 3)

# fit the model
clf = IsolationForest(n_estimators=50, max_samples=500, random_state=rng, contamination=0.01)
clf.fit(X_train)

dump(clf, './isolation_forest.joblib')
print('finished training')
```

Tutorial 10: real-time anomaly detection

- detector.py

```
while True:
    message = consumer.poll(timeout=50)
    if message is None:
        continue
    if message.error():
        logging.error("Consumer error: {}".format(message.error()))
        continue

    # Message that came from producer
    record = json.loads(message.value().decode('utf-8'))
    data = record["data"]
    print(data)
    prediction = clf.predict(data)
    if prediction[0] == 1:
        print('Normal')


    # If an anomaly comes in, send it to anomalies topic
    if prediction[0] == -1:
        print('Abnormal')
        score = clf.score_samples(data)
        record["score"] = np.round(score, 3).tolist()

        _id = str(record["id"])
        record = json.dumps(record).encode("utf-8")

        producer.produce(topic=ANOMALIES_TOPIC,
                         value=record)
        producer.flush()
        print(record)
        print('Alert sent!')
    # consumer.commit() # Uncomment to process all messages, not just new ones
```

Tutorial 10: real-time anomaly detection

```
(base) C:\Users\PC\PythonCodes\Kafka\abnomaldetection>python train.py
finished training
```

Tutorial 10: real-time anomaly detection

```
(base) C:\Users\PC\PythonCodes\Kafka\abnomaldetection>python producer.py
produce message
b'{"id": 0, "data": [[2.098, 2.606]], "current_time": "2021-09-09T15:03:21.134716"}'
produce message
b'{"id": 1, "data": [[2.159, 2.785]], "current_time": "2021-09-09T15:03:22.177714"}'
produce message
b'{"id": 2, "data": [[-1.014, 3.463]], "current_time": "2021-09-09T15:03:23.204923"}'
produce message
b'{"id": 3, "data": [[2.102, -3.905]], "current_time": "2021-09-09T15:03:24.223482"}'
produce message
b'{"id": 4, "data": [[1.2, 1.708]], "current_time": "2021-09-09T15:03:25.252457"}'
produce message
```

Tutorial 10: real-time anomaly detection

```
C:\kafka>.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic transactions
{"id": 42, "data": [[2.149, 2.192]], "current_time": "2021-09-09T15:04:04.194174"}
{"id": 43, "data": [[-1.713, -2.004]], "current_time": "2021-09-09T15:04:05.213219"}
{"id": 44, "data": [[2.128, 2.349]], "current_time": "2021-09-09T15:04:06.244390"}
{"id": 45, "data": [[-2.026, -2.136]], "current_time": "2021-09-09T15:04:07.266721"}
{"id": 46, "data": [[1.473, 1.82]], "current_time": "2021-09-09T15:04:08.293584"}
{"id": 47, "data": [[-1.583, -1.949]], "current_time": "2021-09-09T15:04:09.325670"}
{"id": 48, "data": [[1.394, -1.956]], "current_time": "2021-09-09T15:04:10.354479"}
{"id": 49, "data": [[-2.348, -1.901]], "current_time": "2021-09-09T15:04:11.384494"}
```

Tutorial 10: real-time anomaly detection

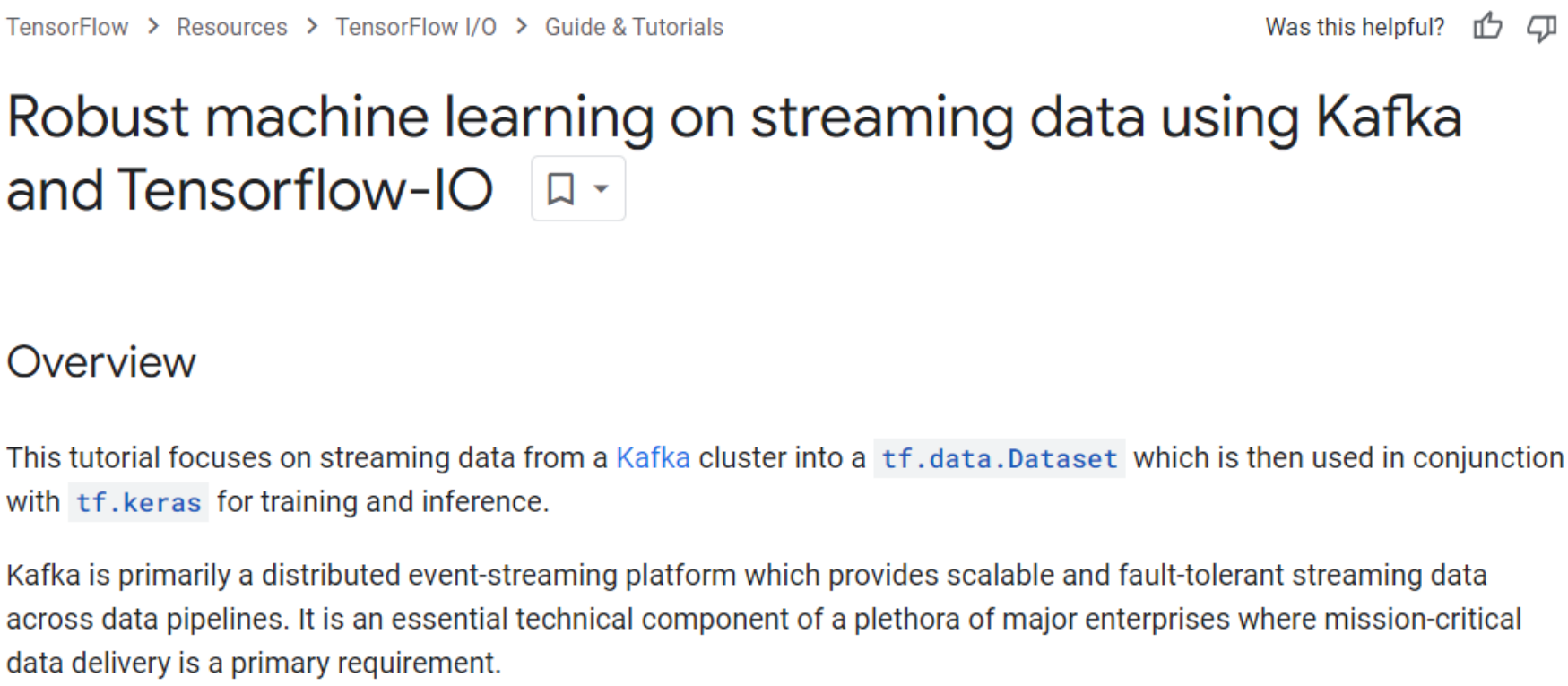
```
(base) C:\Users\PC\PythonCodes\Kafka\abnomaldetection>python detector.py
[[-1.915, 1.71]]
Abnormal
b'{"id": 725, "data": [[-1.915, 1.71]], "current_time": "2021-09-09T14:01:07.985942", "score": [-0.711]}'
Alert sent
[[2.516, 1.881]]
Normal
[[-1.472, -2.328]]
Normal
[[2.597, 1.787]]
Normal
```

Tutorial 10: real-time anomaly detection

```
C:\kafka>.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic anomalies
{"id": 181, "data": [[-2.883, -3.107]], "current_time": "2021-09-09T15:06:26.751928", "score": [-0.733]}
{"id": 184, "data": [[-0.804, 1.628]], "current_time": "2021-09-09T15:06:29.830897", "score": [-0.699]}
{"id": 192, "data": [[-0.899, -3.483]], "current_time": "2021-09-09T15:06:38.020883", "score": [-0.711]}
```

Tutorial 11: Tensorflow-IO and Kafka

https://www.tensorflow.org/io/tutorials/kafka



```
!pip install tensorflow-io==0.25.0
!pip install kafka-python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow-io==0.25.0 in /usr/local/lib/python3.7/dist-packages (0.25.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem==0.25.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow-io==0.25.0) (0.25.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kafka-python in /usr/local/lib/python3.7/dist-packages (2.0.2)
```

Import packages

```
import os
from datetime import datetime
import time
import threading
import json
from kafka import KafkaProducer
from kafka.errors import KafkaError
from sklearn.model_selection import train_test_split
import pandas as pd
import tensorflow as tf
import tensorflow_io as tfio
```

Download and setup Kafka and Zookeeper instances

For demo purposes, the following instances are setup locally:

- Kafka (Brokers: 127.0.0.1:9092)
- Zookeeper (Node: 127.0.0.1:2181)

```
[ ]  !curl -sSOL https://downloads.apache.org/kafka/3.3.1/kafka_2.13-3.3.1.tgz
     !tar -xzf kafka_2.13-3.3.1.tgz
```

Using the default configurations (provided by Apache Kafka) for spinning up the instances.

```
[ ]  !./kafka_2.13-3.3.1/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-3.3.1/config/zookeeper.properties
     !./kafka_2.13-3.3.1/bin/kafka-server-start.sh -daemon ./kafka_2.13-3.3.1/config/server.properties
     !echo "Waiting for 10 secs until kafka and zookeeper services are up and running"
     !sleep 10
```

```
Waiting for 10 secs until kafka and zookeeper services are up and running
```

Create the kafka topics with the following specs:

- susy-train: partitions=1, replication-factor=1
- susy-test: partitions=2, replication-factor=1

```
[ ] !./kafka_2.13-3.3.1/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic susy-train
    !./kafka_2.13-3.3.1/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 2 --topic susy-test
```

```
Error while executing topic command : Topic 'susy-train' already exists.
[2022-10-04 15:41:03,437] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'susy-train' already exists.
 (kafka.admin.TopicCommand$)
Error while executing topic command : Topic 'susy-test' already exists.
[2022-10-04 15:41:05,951] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'susy-test' already exists.
 (kafka.admin.TopicCommand$)
```

- Just follow https://www.tensorflow.org/io/tutorials/kafka

Evaluate the performance of the batch training model on the test data

```
[ ]  res = model.evaluate(test_ds)
     print("test loss, test acc:", res)
```

```
1250/1250 [==============================] - 16s 12ms/step - loss: 0.4328 - accuracy: 0.7994
test loss, test acc: [0.43278080224990845, 0.7993500232696533]
```

```
[ ]  def decode_kafka_online_item(raw_message, raw_key):
       message = tf.io.decode_csv(raw_message, [[0.0] for i in range(NUM_COLUMNS)])
       key = tf.strings.to_number(raw_key)
       return (message, key)

     for mini_ds in online_train_ds:
       mini_ds = mini_ds.shuffle(buffer_size=32)
       mini_ds = mini_ds.map(decode_kafka_online_item)
       mini_ds = mini_ds.batch(32)
       if len(mini_ds) > 0:
         model.fit(mini_ds, epochs=3)
```

```
Epoch 1/3
313/313 [==============================] - 1s 4ms/step - loss: 0.4366 - accuracy: 0.7990
Epoch 2/3
313/313 [==============================] - 2s 6ms/step - loss: 0.4329 - accuracy: 0.8000
Epoch 3/3
313/313 [==============================] - 1s 4ms/step - loss: 0.4305 - accuracy: 0.8023
```

Tutorial 12: Spotify Recommendation System

https://www.analyticsvidhya.com/blog/2021/06/spotify-recommendation-system-using-pyspark-and-kafka-streaming/

Link Spotify dataset new

Spotify Recommendation System using Pyspark and Kafka streaming

Siddharth1698 — June 23, 2021

Advanced Data Engineering Python

Tutorial 13: Order book simulation

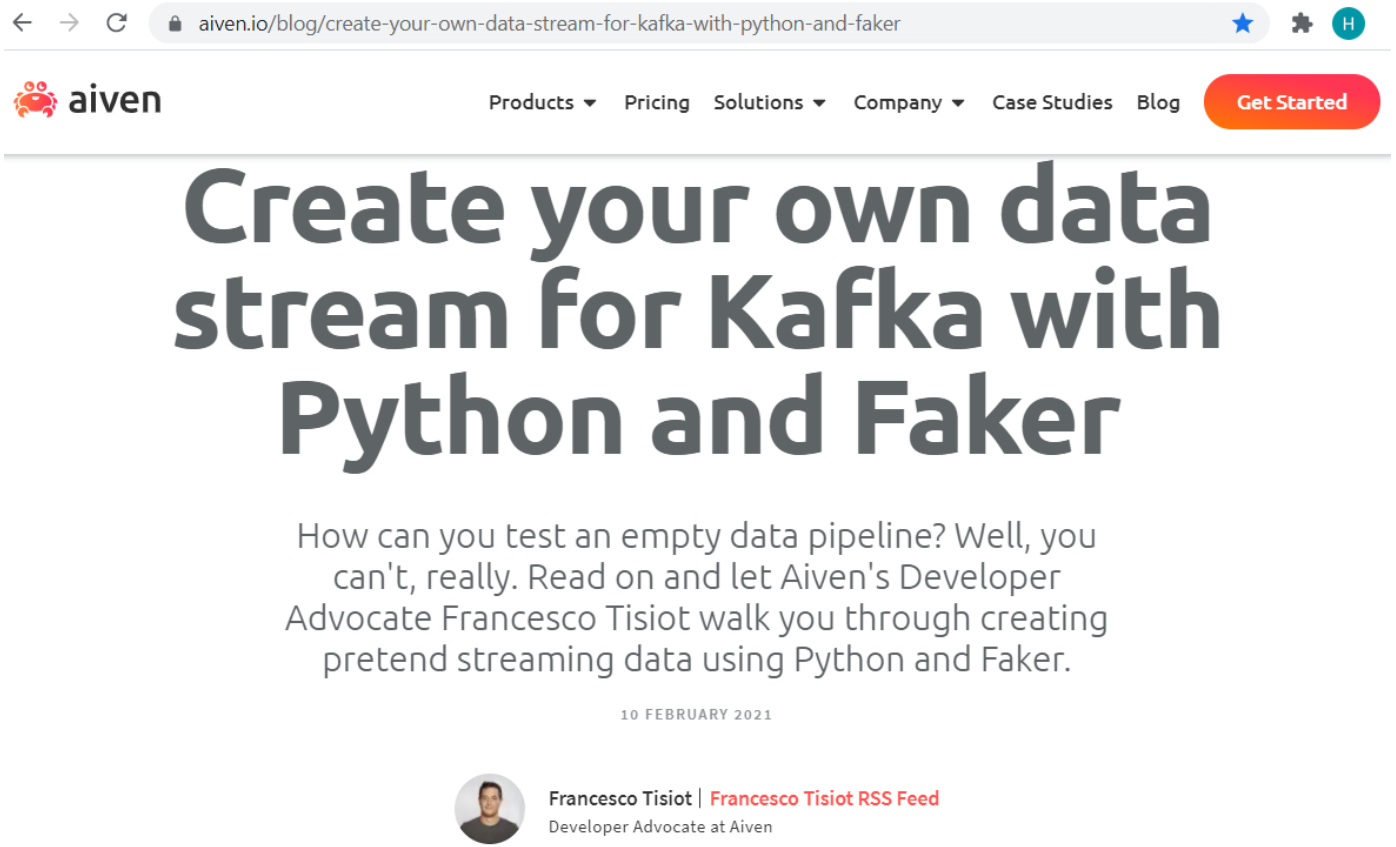
https://github.com/rongpenl/order-book-simulation

https://rongpeng.notion.site

Order book simulation with Kafka and Streamlit

Tutorial 14: Create your own data stream

https://aiven.io/blog/create-your-own-data-stream-for-kafka-with-python-and-faker



Tutorial 15: Bigmart sale prediction

- Dataset: https://www.kaggle.com/datasets/brijbhushannanda1979/bigmart-sales-data
- Use train set to train some simple prediction model using Spark MLlib
- Stream data from test set to Kafka server (remember to set the time interval)
- Create Spark streaming dataframe from Kafka and apply the trained model to get the real-time prediction

```
Showing live view refreshed every 3 seconds
Seconds passed: 150
+---------------+-----------+---------------+--------+----------+---------------------------+
|Item_Identifier|Item_Weight|Item_Visibility|Item_MRP|Outlet_Age|Item_Outlet_Sales_Prediction|
+---------------+-----------+---------------+--------+----------+---------------------------+
|          FDW58|      20.75|    0.007564836|107.8622|        24|          1628.3373282574423|
|          FDW14|        8.3|    0.038427677| 87.3198|        16|          1553.3888202285195|
|          NCN55|       14.6|    0.099574908|241.7538|        25|          420.90160117513597|
|          FDQ58|      7.315|    0.015388393| 155.034|        16|          2574.9108782350113|
|          FDY38|        8.3|    0.118599314|  234.23|        38|          6087.5964602566455|
|          FDW58|      20.75|    0.007564836|107.8622|        24|          1628.3373282574423|
|          FDW14|        8.3|    0.038427677| 87.3198|        16|          1553.3888202285195|
|          NCN55|       14.6|    0.099574908|241.7538|        25|          420.90160117513597|
|          FDQ58|      7.315|    0.015388393| 155.034|        16|          2574.9108782350113|
|          FDY38|        8.3|    0.118599314|  234.23|        38|          6087.5964602566455|
|          FDH56|        9.8|    0.063817206|117.1492|        26|          2004.2270443218151|
|          FDL48|      19.35|    0.082601537| 50.1034|        14|           89.64992285945802|
|          FDC48|        8.3|    0.015782495| 81.0592|        38|           1797.400735516933|
|          FDN33|      6.305|    0.123365446| 95.7436|        21|          1514.5830372463292|
|          FDA36|      5.985|    0.005698435|186.8924|        16|           3305.455936646477|
|          FDT44|       16.6|    0.103569075|118.3466|        16|          2013.9487249985405|
|          FDQ56|       6.59|     0.10581147| 85.3908|        21|          1504.5158649216487|
|          NCC54|        8.3|    0.171079215|240.4196|        38|           509.4295601878332|
|          FDU11|      4.785|    0.092737611|122.3098|        24|           2011.757531506747|
|          DRL59|      16.75|    0.021206464| 52.0298|        36|           762.8046850541455|
+---------------+-----------+---------------+--------+----------+---------------------------+
only showing top 20 rows

Live view ended...
```