Confluent Kafka and KSQL:
Streaming Data Pipelines Made Easy

Kairo Tavares
kairo.tavares@hpe.com

Goals

How **Confluent Kafka Platform** can solve problems in company

What are **streaming data pipelines** and what are its challenges

How **KSQL** can make easy your streaming data pipeline

Agenda

– Kafka 101

– Confluent Kafka

– Streaming Data Pipeline

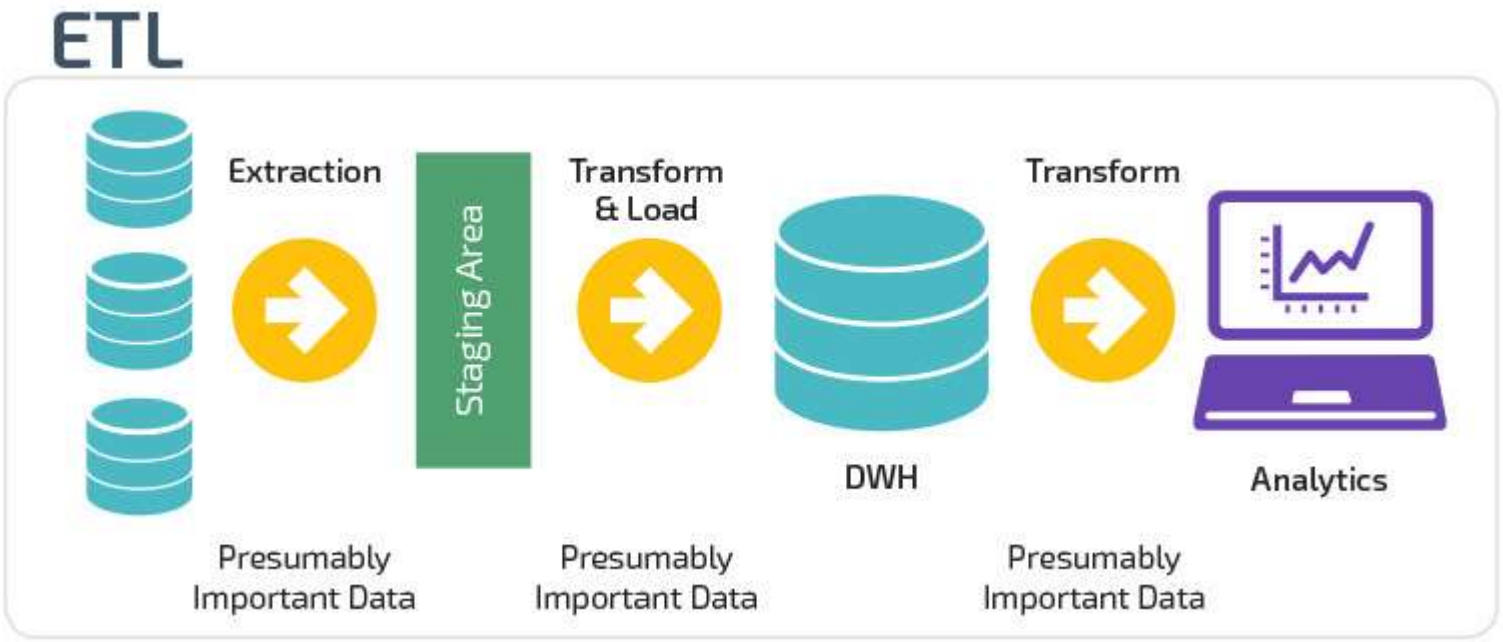
– KSQL

– Demo

Kafka 101

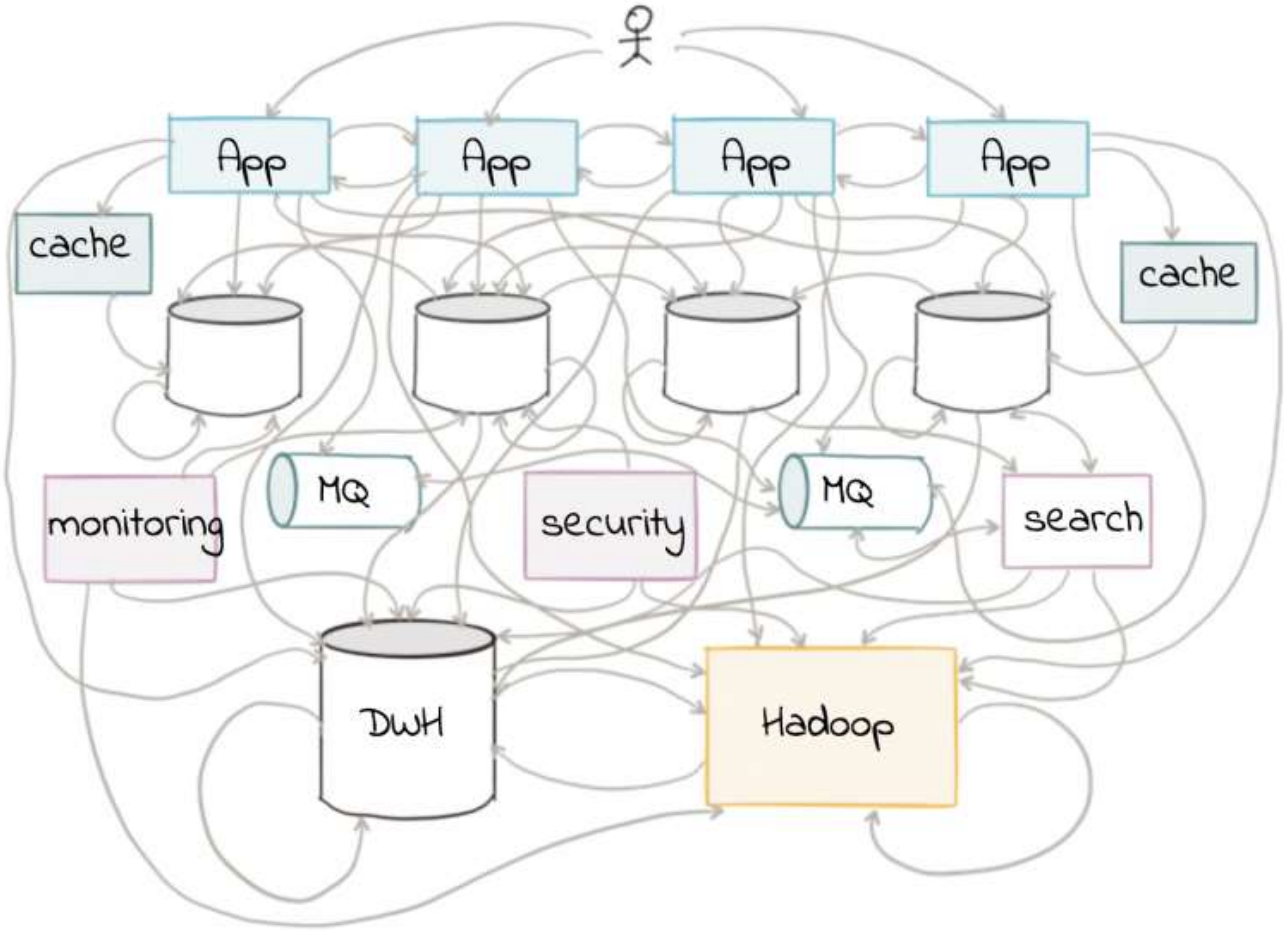
Common problems that we face

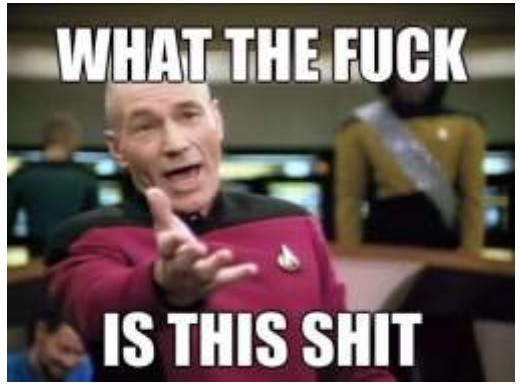
Extract - Transform – Load (ETL)



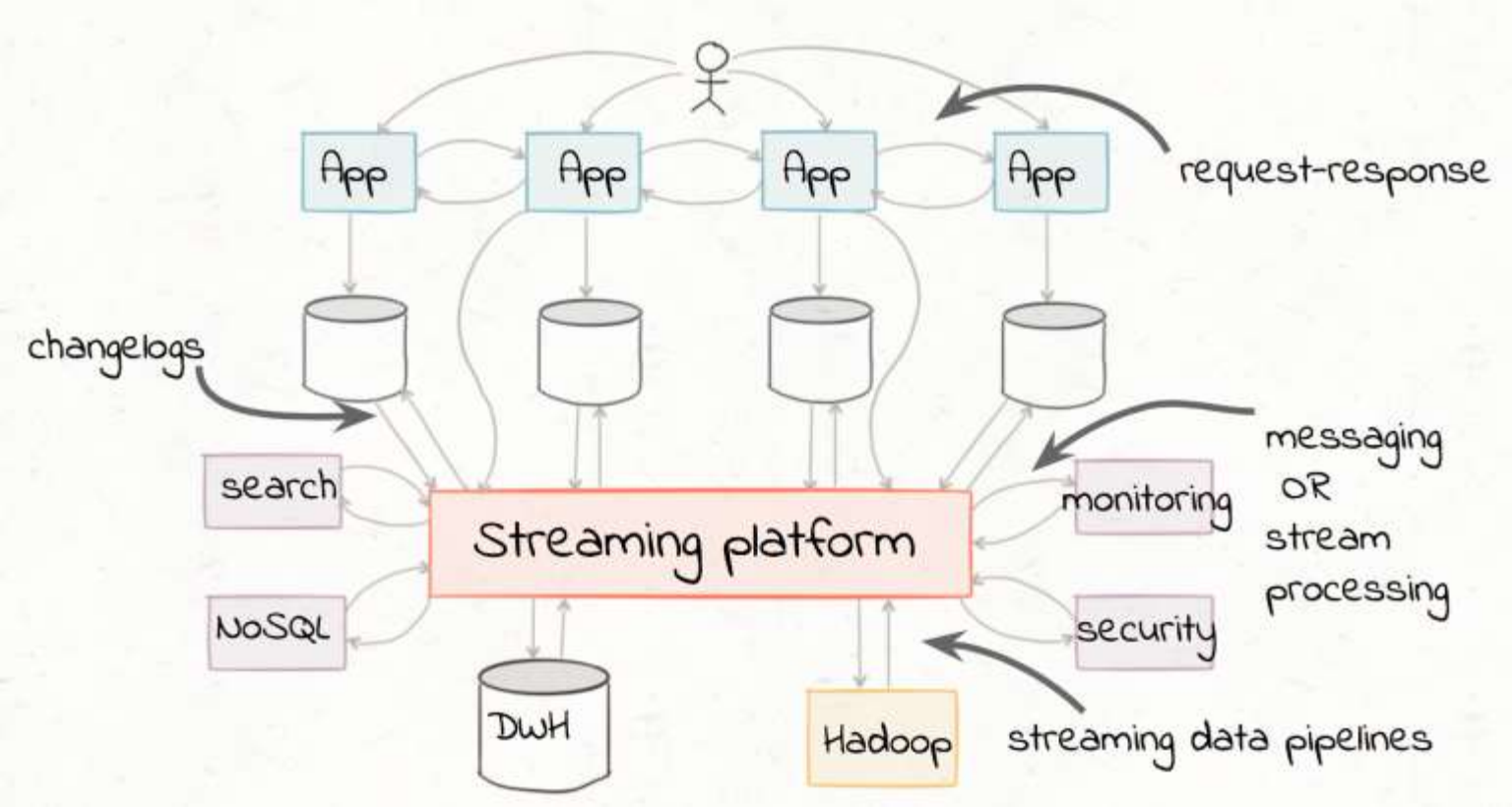
Common problems that we face

Microservices





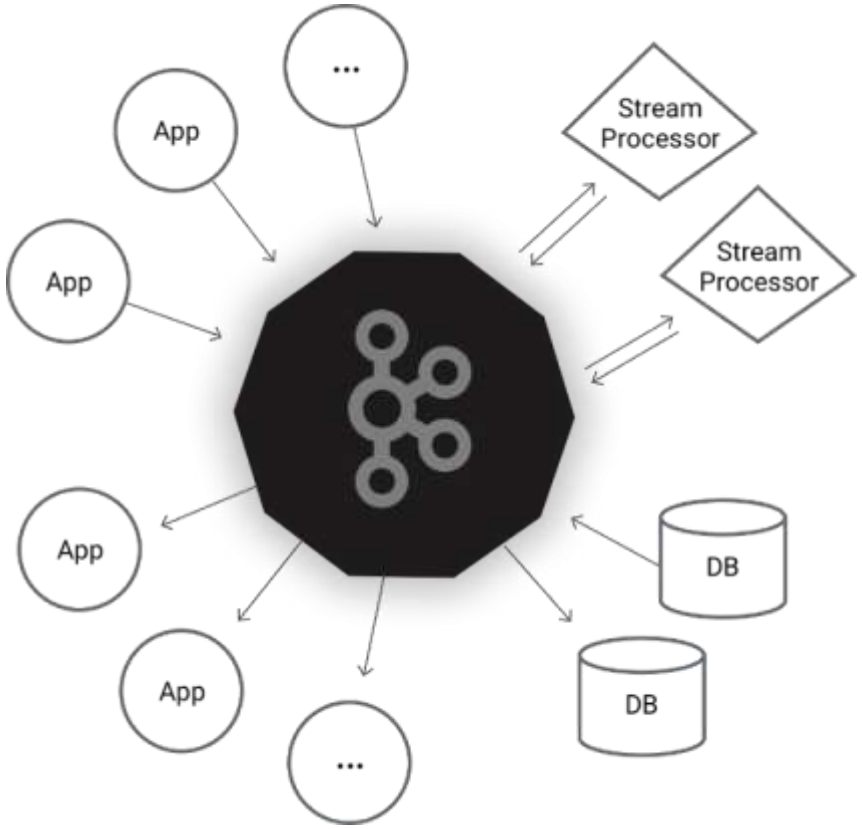
Lets organized it



Kafka 101

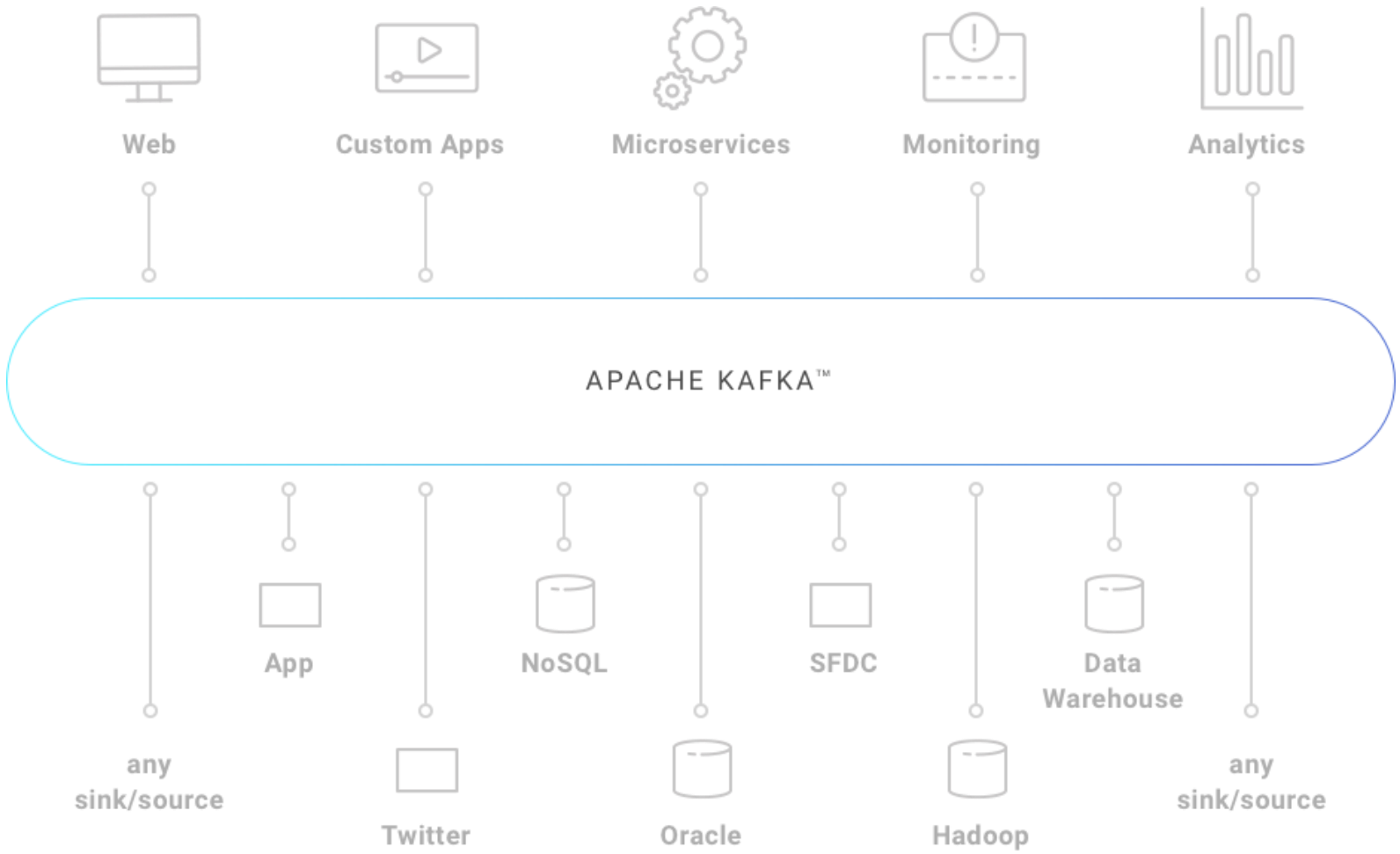
Apache Kafka

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



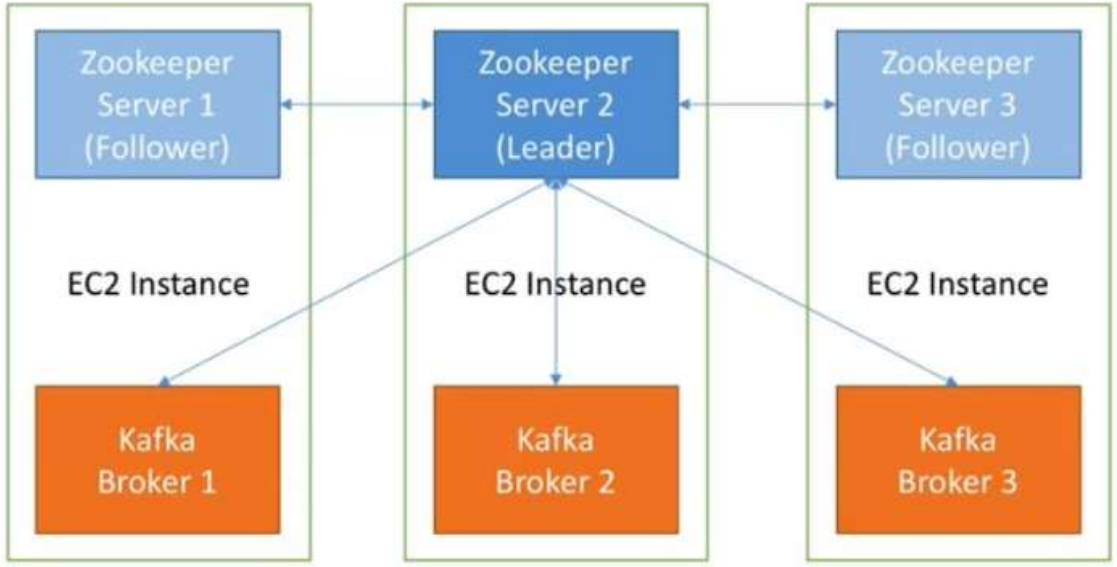
https://kafka.apache.org/

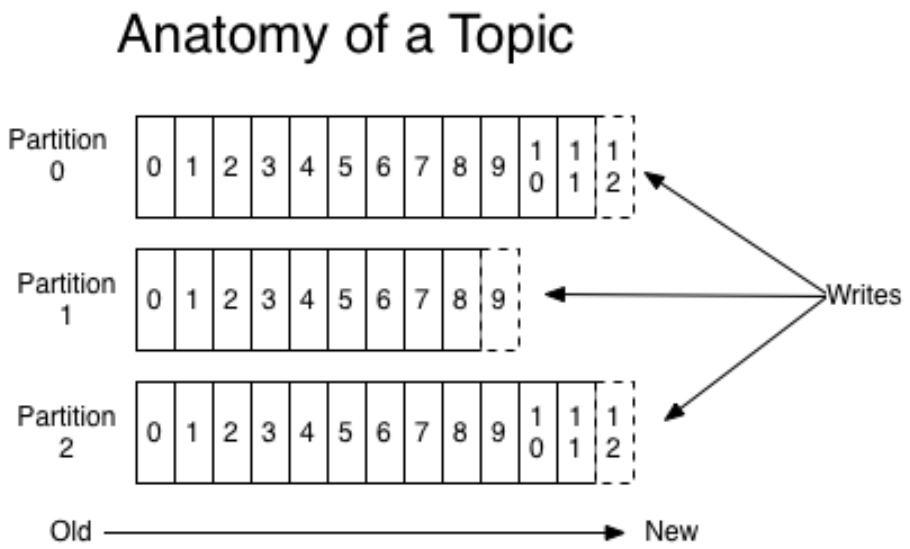
Kafka 101



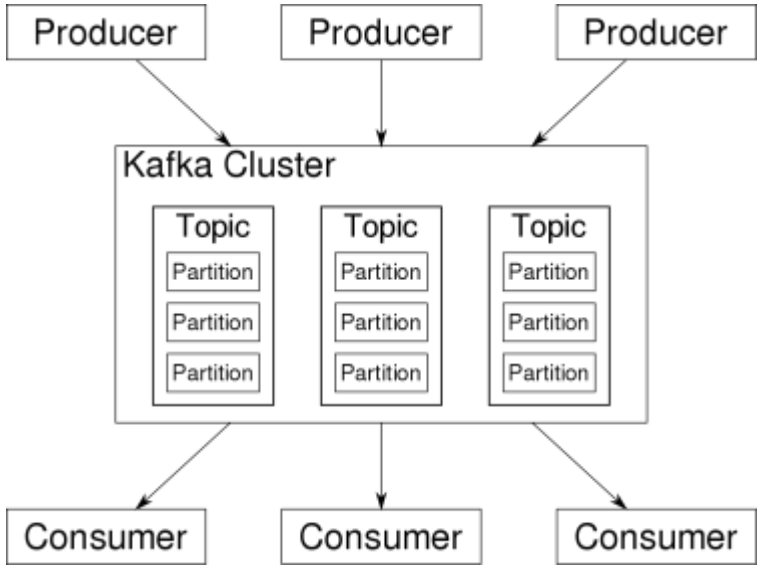
Hewlett Packard Enterprise

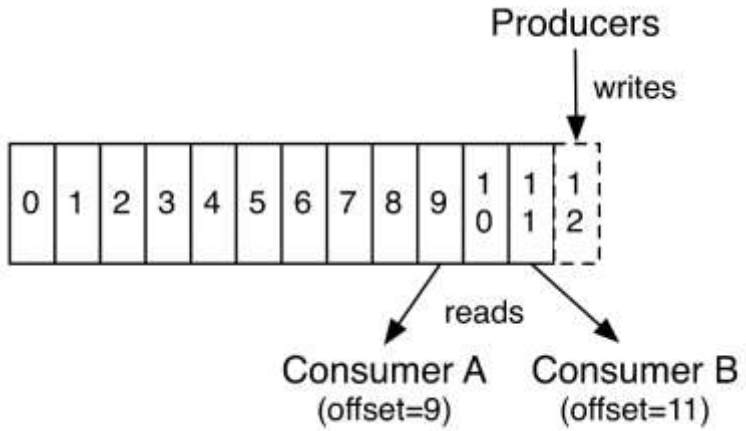
Kafka 101
Producing Data



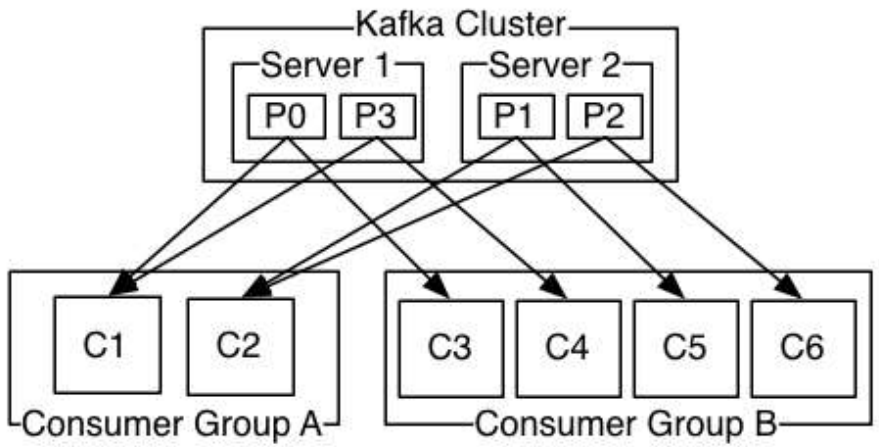


Kafka 101
Consuming Data





Simple Consumer



Consumer Groups

Confluent Platform

Confluent Platform

Founded by the **team** that built **Apache Kafka®**, Confluent builds an event **streaming platform** that enables companies to easily access data as **real-time streams**

Development and Connectivity

Stream Processing

Management and Operations

https://www.confluent.io/

Hewlett Packard Enterprise

Confluent Platform

Deployment and Connectivity

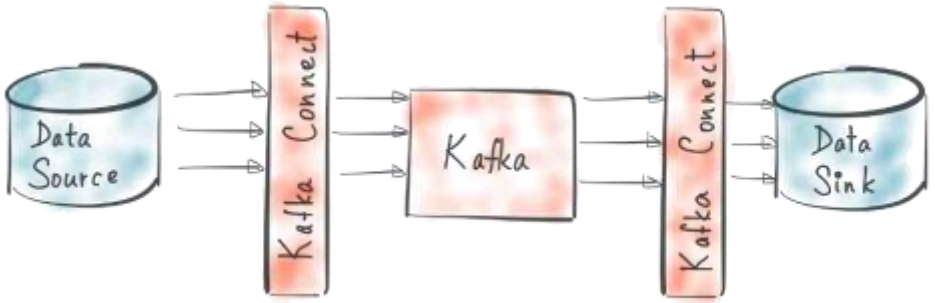
– **Kafka Connect & Connectors**

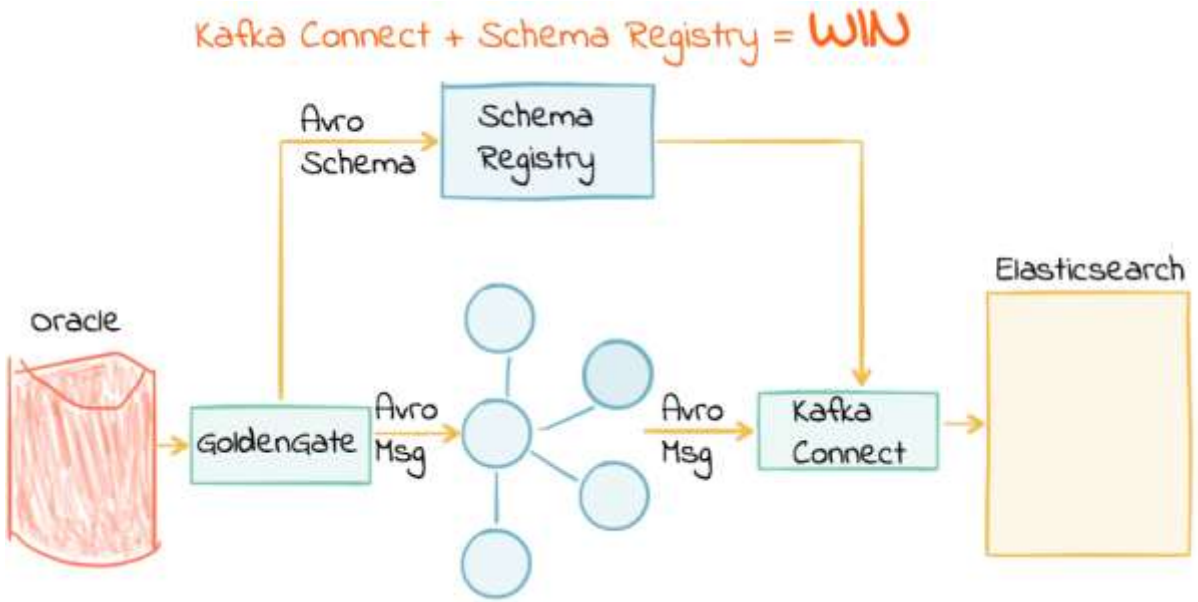
– **Schema Registry**

– **Kafka Clients**

– **REST Proxy**

– **MQTT Proxy**





Confluent Platform

Avro Format

– Open Source Data Format

– Choice for a number of reasons:
  – **Direct mapping** to and from JSON
  – **Compact format**
  – **Very fast** to serialize and deserialize
  – **Support** to several programming languages
  – It has a rich, **extensible schema language** defined in pure JSON
  – It has the best notion of **compatibility** for evolving your data over time
  – Built-in **documentation**

One of the critical features of Avro is the ability to define a schema for your data. For example an event that represents the sale of a product might look like this:

```
{
  "time": 1424849130111,
  "customer_id": 1234,
  "product_id": 5678,
  "quantity":3,
  "payment_type": "mastercard"
}
```

It might have a schema like this that defines these five fields:

```
{
  "type": "record",
  "doc":"This event records the sale of a product",
  "name": "ProductSaleEvent",
  "fields" : [
    {"name":"time", "type":"long", "doc":"The time of the purchase"},
    {"name":"customer_id", "type":"long", "doc":"The customer"},
    {"name":"product_id", "type":"long", "doc":"The product"},
    {"name":"quantity", "type":"int"},
    {"name":"payment",
     "type":{"type":"enum",
             "name":"payment_types",
             "symbols":["cash","mastercard","visa"]},
     "doc":"The method of payment"}
  ]
}
```

Confluent Platform

Management and Operations

- **Control Center**
  - Manage key operations and monitor the health and performance of Kafka clusters and data streams with curated dashboards directly from a GUI.
- **Replicator**
  - Replicate Kafka topics across data centers and public clouds to ensure disaster recovery and build distributed data pipelines.
- **Auto Data Balancer**
  - Optimize your resource utilization by invoking a rack-aware algorithm that automatically rebalances partitions across a Kafka cluster.
- **Security Controls**
  - Enable pass-through client credentials from REST Proxy / Schema Registry to Kafka broker. Map AD/LDAP groups to Kafka ACLs.
- **Operator**
  - Automate deployment of the complete Confluent Platform, including Kafka, as a cloud-native application on Kubernetes.

Confluent Platform

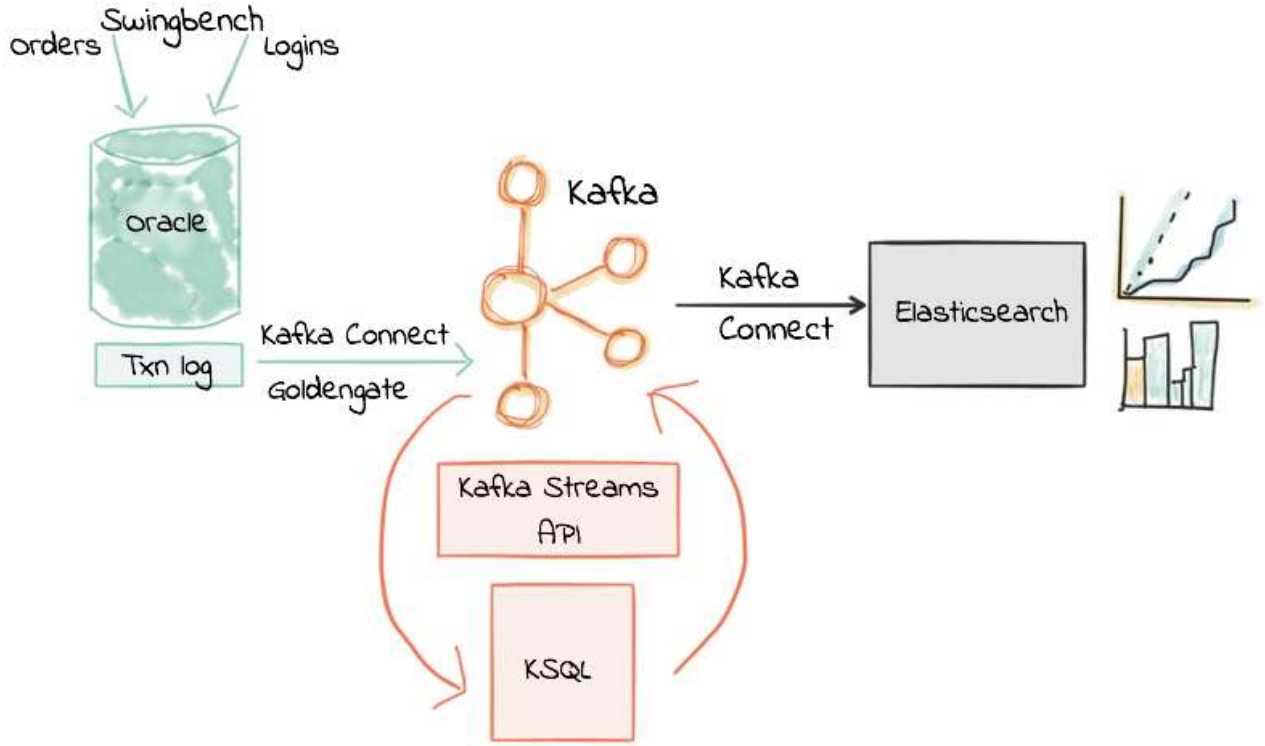
Stream Processing

– **Kafka Streams (Library)**

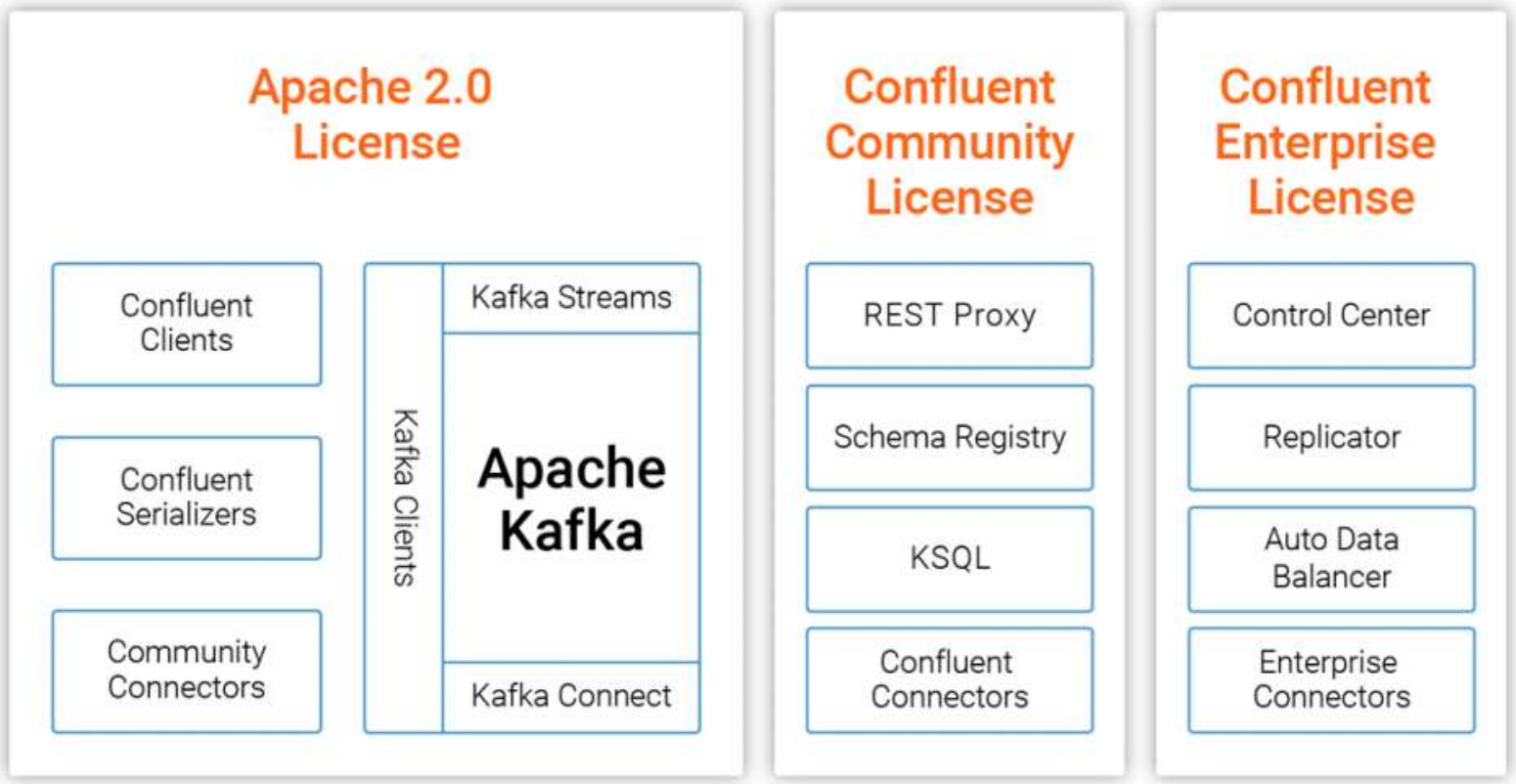
  – Build mission-critical real-time applications with a simple Java library and a Kafka cluster - no additional framework or cluster needed.

– **KSQL (Service)**

  – Build real-time stream processing applications against Apache Kafka using simple SQL-like semantics.



Confluent Platform
License



Streaming Data Pipeline

Streaming Data Pipeline

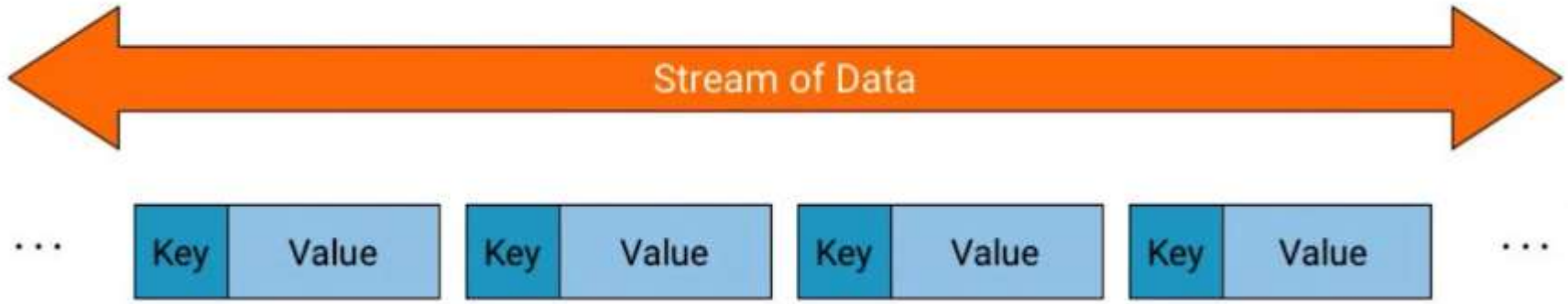
But what is a stream?

– A stream is an **unbounded**, continuous flow of records

– Data is **real-time**

– **Immutable** events

– Records are **key-value** pairs (Kafka)



Streaming Data Pipeline

Stream Processing

Per-record **millisecond delay**

Data **filtering**

Data **transformation** and **conversions**

Data **enrichment** with joins

Data **manipulation** with scalar functions

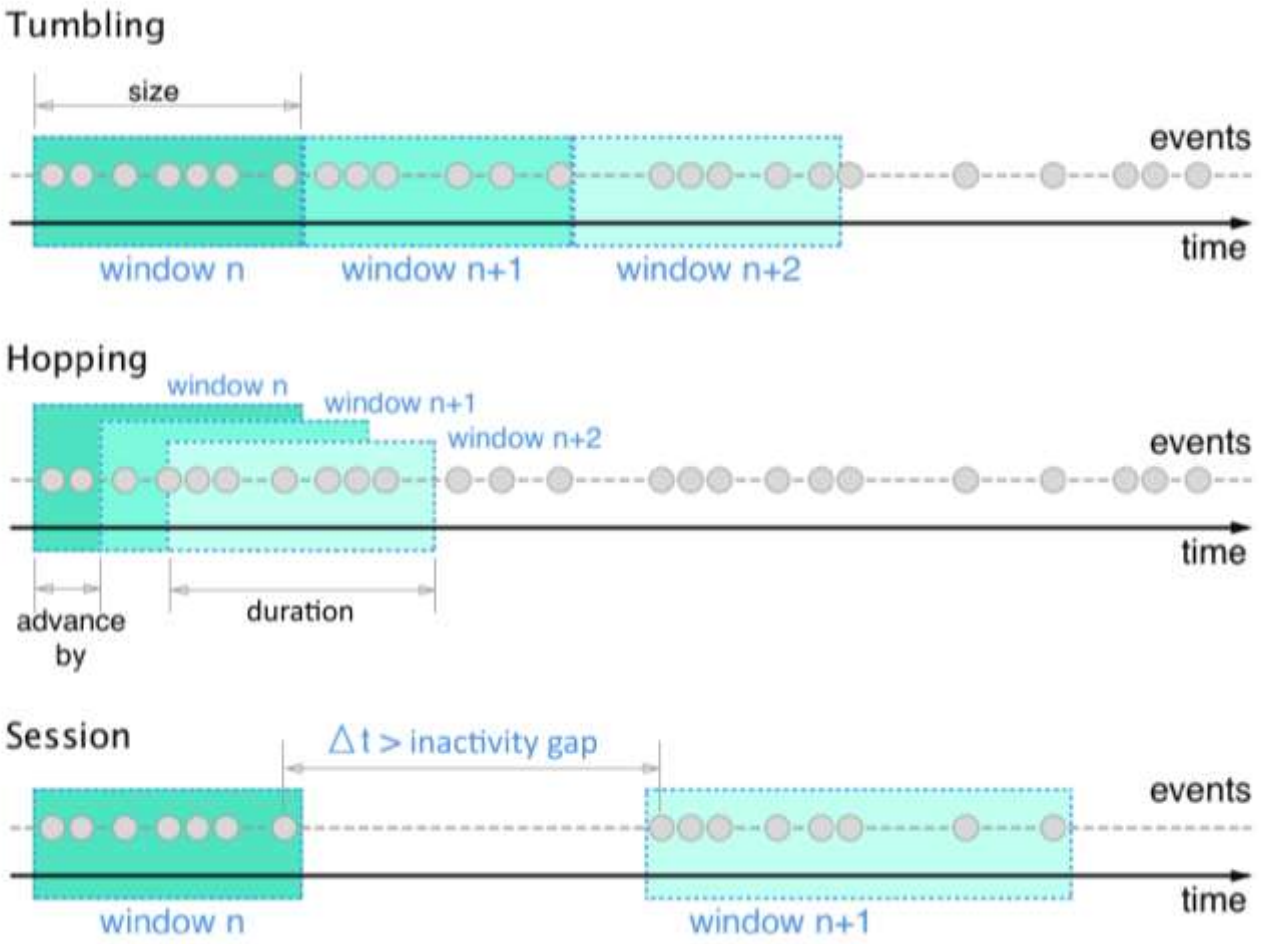
Stateful processing

Data **Aggregation**

Windowing processing

Streaming Data Pipeline

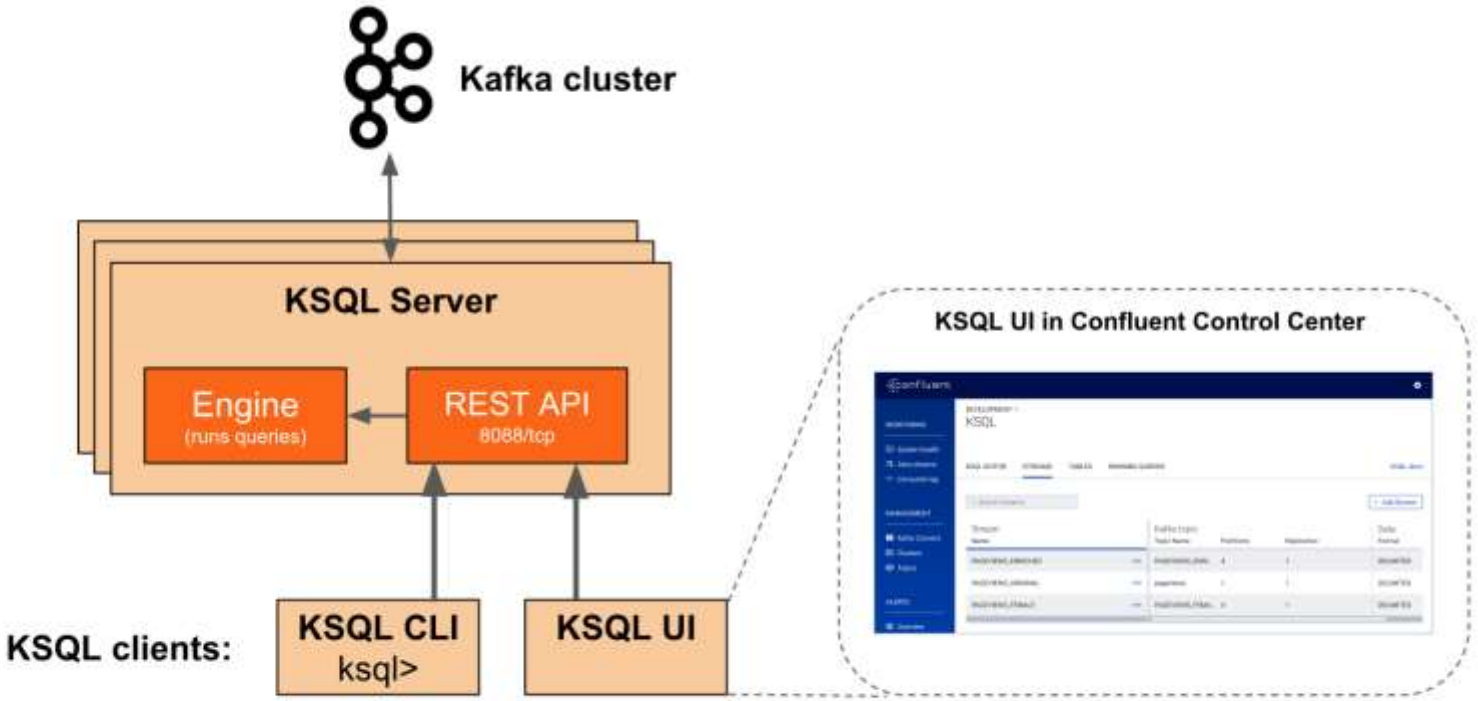
Windowing



KSQL

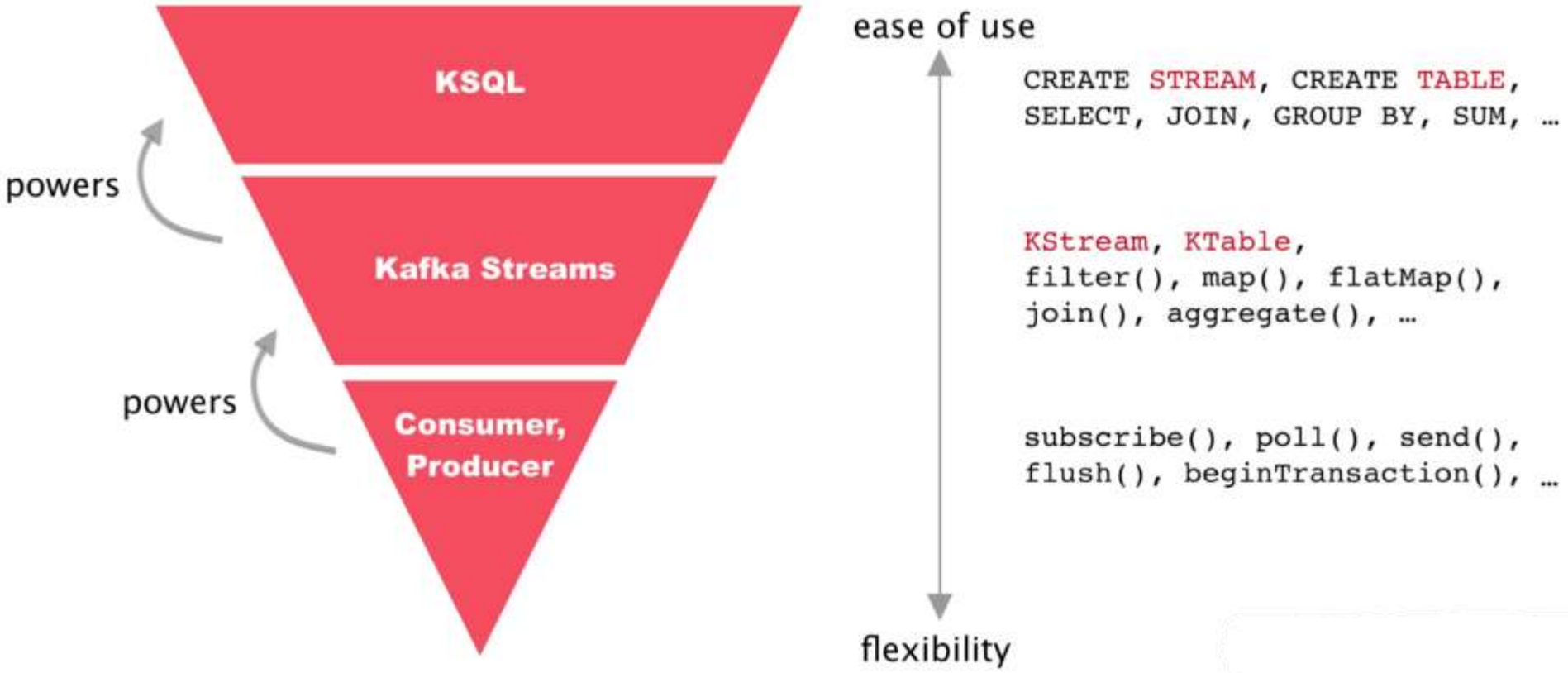
KSQL

Architecture and components



https://docs.confluent.io/current/ksql/docs/concepts/ksql-architecture.html

KSQL
Why KSQL?



KSQL

Kafka Streams Library vs KSQL

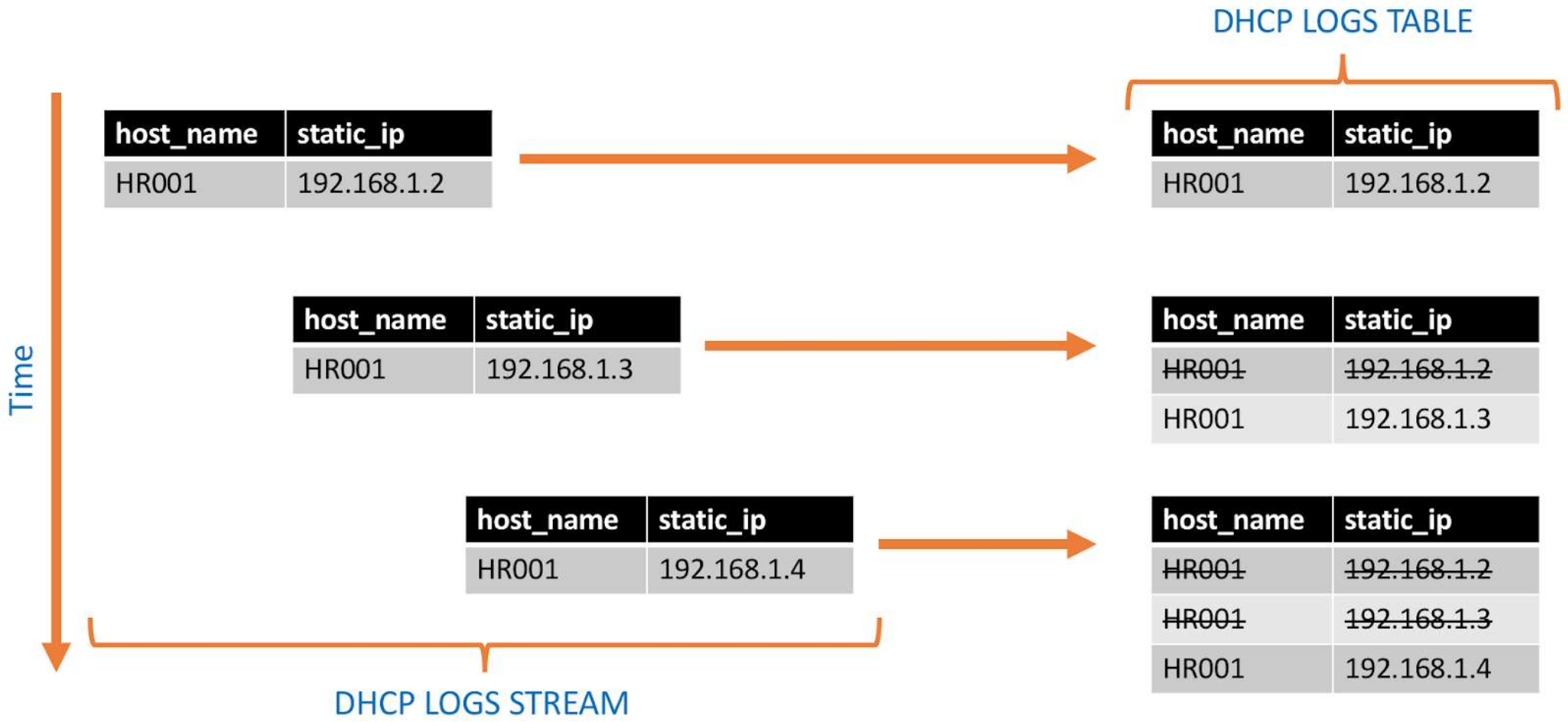
Differences	KSQL	Kafka Streams
You write:	KSQL statements	JVM applications
Graphical UI	Yes, in Confluent Control Center and Confluent Cloud	No
Console	Yes	No
Data formats	Avro, JSON, CSV	Any data format, including Avro, JSON, CSV, Protobuf, XML
REST API included	Yes	No, but you can implement your own
Runtime included	Yes, the KSQL server	Applications run as standard JVM processes
Queryable state	No	Yes

```
CREATE STREAM fraudulent_payments AS
 SELECT fraudProbability(data) FROM payments
 WHERE fraudProbability(data) > 0.8;
```

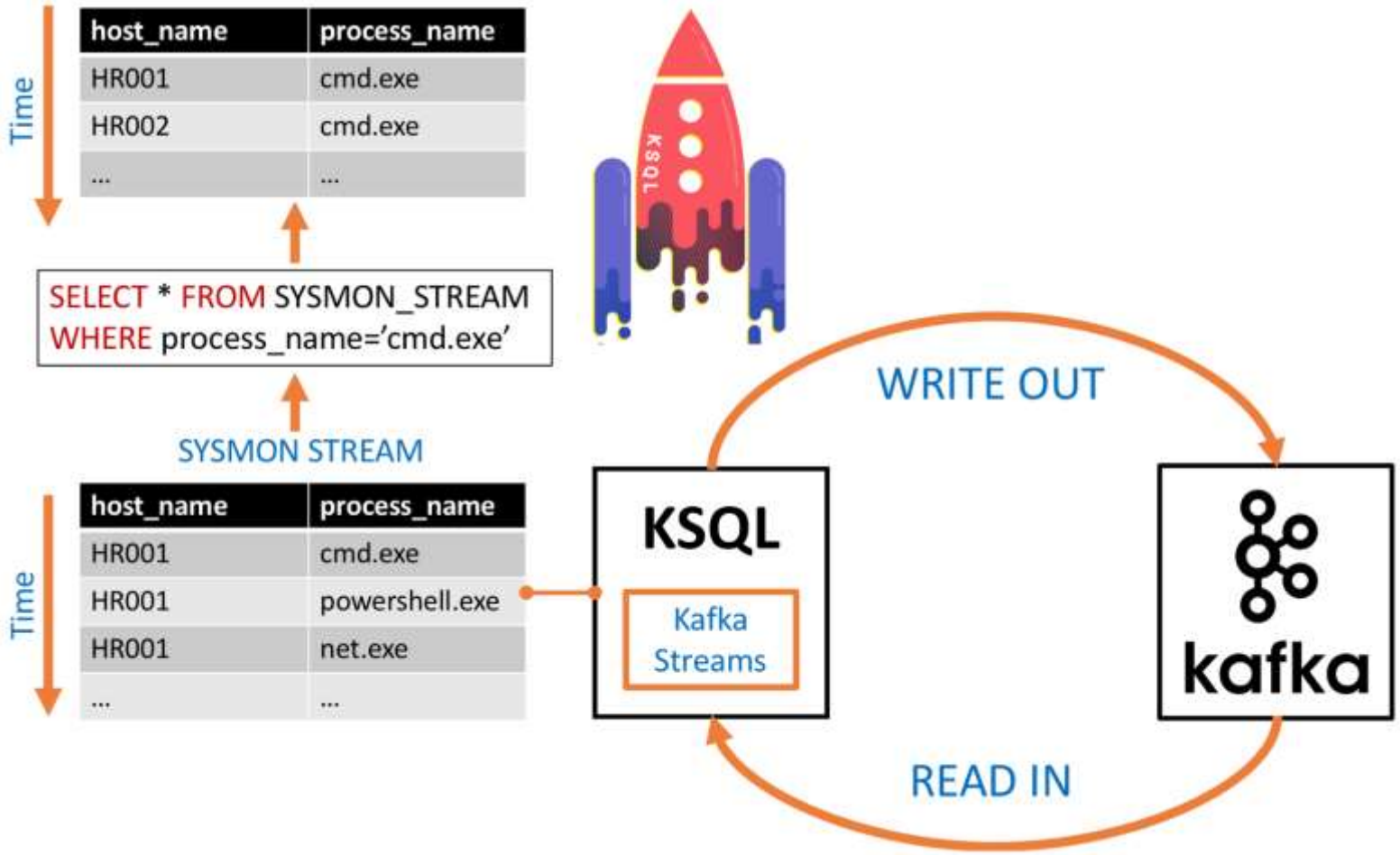
```
// Example fraud-detection logic using the Kafka Streams API.
object FraudFilteringApplication extends App {

  val builder: StreamsBuilder = new StreamsBuilder()
  val fraudulentPayments: KStream[String, Payment] = builder
    .stream[String, Payment]("payments-kafka-topic")
    .filter((_ ,payment) => payment.fraudProbability > 0.8)
  fraudulentPayments.to("fraudulent-payments-topic")

  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092")

  val streams: KafkaStreams = new KafkaStreams(builder.build(), config)
  streams.start()
}
```

KSQL

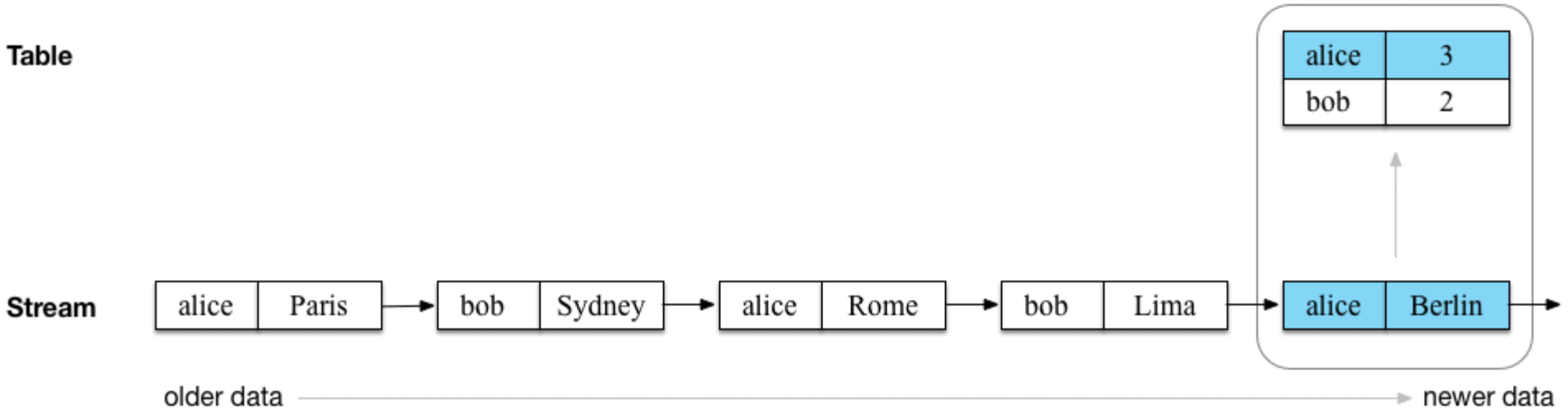
Streams vs Tables



KSQL
Filtering



KSQL
Aggregation




```
CREATE TABLE pageviews_per_region AS
  SELECT regionid,
         COUNT(*)
  FROM pageviews
  GROUP BY regionid;
```

KSQL

Aggregation Functions

Function	Example	Input Type
COLLECT_LIST	`COLLECT_LIST(col1)`	Stream, Table
COLLECT_SET	`COLLECT_SET(col1)`	Stream
COUNT	`COUNT(col1)`, `COUNT(*)`	Stream, Table
HISTOGRAM	`HISTOGRAM(col1)`	Stream, Table
MAX	`MAX(col1)`	Stream
MIN	`MIN(col1)`	Stream
SUM	`SUM(col1)`	Stream, Table
TOPK	`TOPK(col1, k)`	Stream
TOPKDISTINCT	`TOPKDISTINCT(col1, k)`	Stream
WindowStart	`WindowStart()`	Stream Table
WindowEnd	`WindowEnd()`	Stream Table

KSQL
Joins

JOIN Sources	Description
Stream-Stream	Stream-Stream joins are always time-windowed joins and support INNER, LEFT OUTER, and FULL OUTER joins
Stream-Table	Stream-table joins are always non-time-windowed joins and support INNER and LEFT joins
Table-Table	Table-table joins are always non-time-windowed joins and support INNER, LEFT OUTER, and FULL OUTER joins

Stream

host_name	static_ip
HR001	192.168.1.2
HR002	192.168.1.4
HR003	192.168.1.6

Table

host_name	user_name
HR005	cbrown
HR002	prodriguez
HR006	nschnider

INNER JOIN =

host_name	static_ip	host_name	user_name
HR002	192.168.1.4	HR002	prodriguez

Hewlett Packard Enterprise

KSQL

Click Stream - ETL

Stream
(clickstream)
{userid, page, action, usage_time}

Table
(users)
{user_id, level, gender, age}

```
CREATE STREAM vip_actions AS
SELECT userid, page, action
FROM clickstream c
LEFT JOIN users u ON c.userid = u.user_id
WHERE u.level = 'Platinum';
```

KSQL

Credit Card Fraud – Anomaly detection

```
CREATE TABLE possible_fraud AS
SELECT card_number, count(*)
FROM authorization_attempts
WINDOW TUMBLING (SIZE 5 SECONDS)
GROUP BY card_number
HAVING count(*) > 3;
```

KSQL

Error Monitoring

```
CREATE TABLE error_counts AS
SELECT error_code, count(*)
FROM monitoring_stream
WINDOW TUMBLING (SIZE 1 MINUTE)
WHERE  type = 'ERROR'
GROUP BY error_code;
```

KSQL
Arithmetic Operations

- Arithmetic (`+,-,/,*,%`) The usual arithmetic operators may be applied to numeric types (INT, BIGINT, DOUBLE)

```
SELECT LEN(FIRST_NAME) + LEN(LAST_NAME) AS NAME_LENGTH FROM USERS;
```

- Concatenation (`+,||`) The concatenation operator can be used to concatenate STRING values.

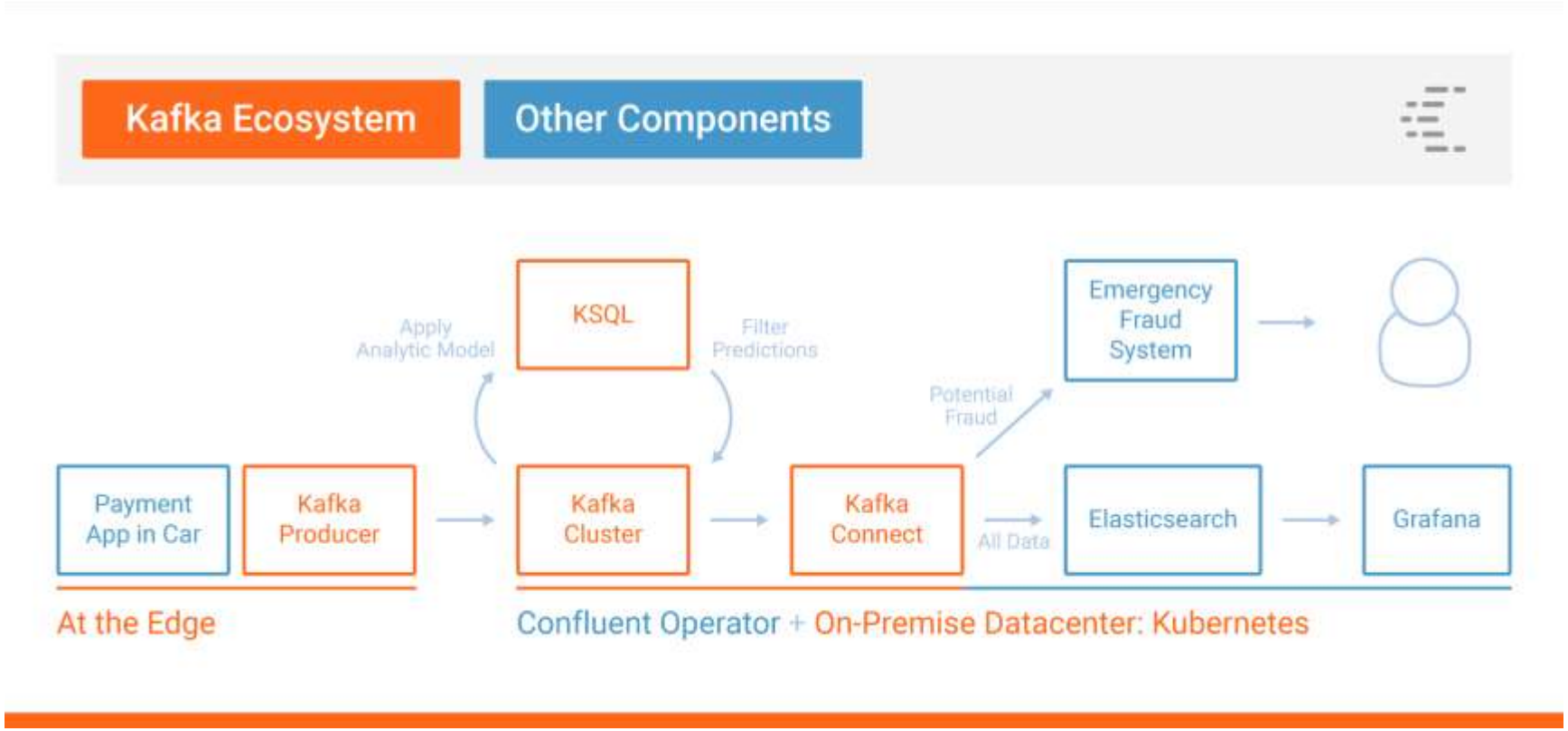
```
SELECT FIRST_NAME + LAST_NAME AS FULL_NAME FROM USERS;
```

- You can use the `+` operator for multi-part concatenation, for example:

```
SELECT TIMESTAMPTOSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss') +
        ': :heavy_exclamation_mark: On ' +
        HOST +
        ' there were ' +
        CAST(INVALID_LOGIN_COUNT AS VARCHAR) +
        ' attempts in the last minute (threshold is >=4)'
FROM INVALID_USERS_LOGINS_PER_HOST
WHERE INVALID_LOGIN_COUNT>=4;
```

KSQL

Machine Learning Prediction




```
CREATE STREAM AnomalyDetectionWithFilter
SELECT rowtime, eventid, anomaly(sensorinput) AS Anomaly
FROM carsensor
WHERE anomaly(sensorinput) > 5;
```

Demo

https://docs.confluent.io/current/quickstart/ce-docker-quickstart.html

Thanks

kairo.tavares@hpe.com