

Building stream processing applications with Apache Kafka® using KSQL

@rmoff

#KafkaSummit

STREAM

PROCESSING

PROCESSING

STREAM

PROCESSING

a

STREAM

of **EVENTS**

STREAMS ARE
of EVENTS

EVERYWHERE

A Customer Experience



A Sale



A Sensor Reading

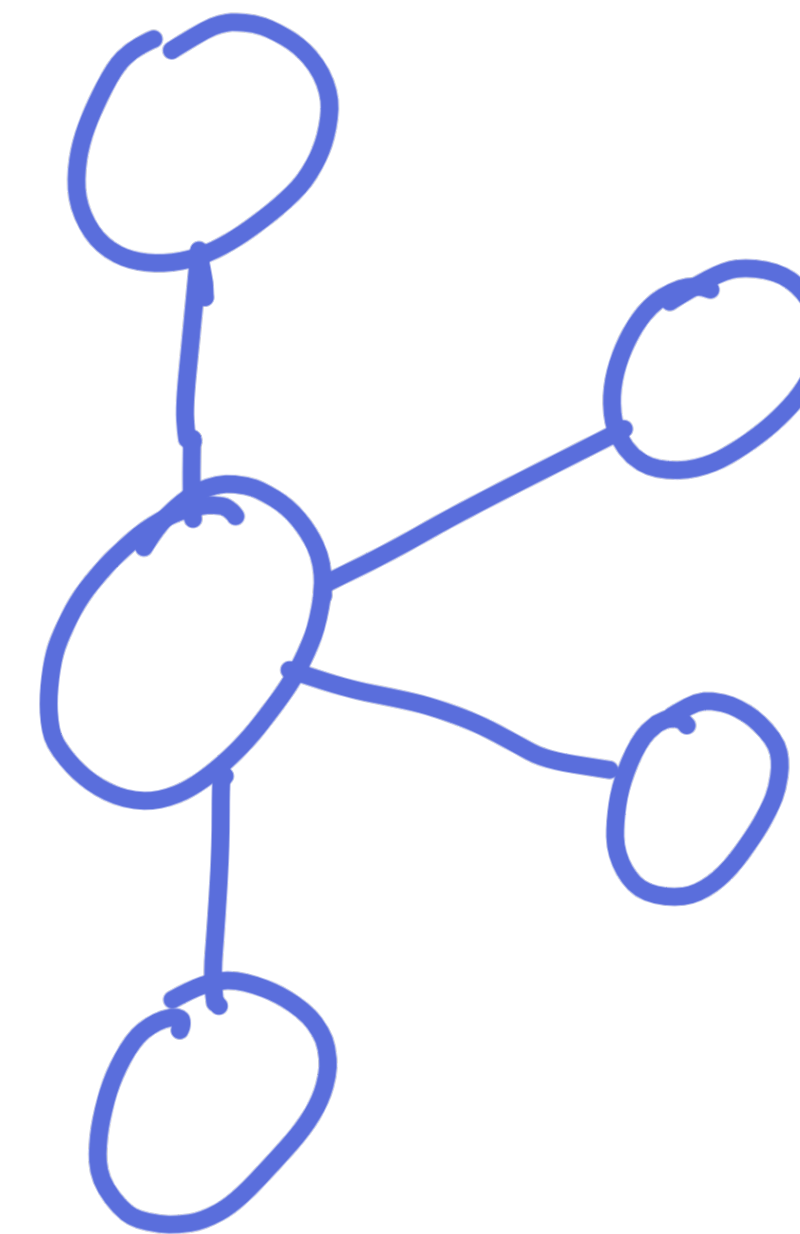
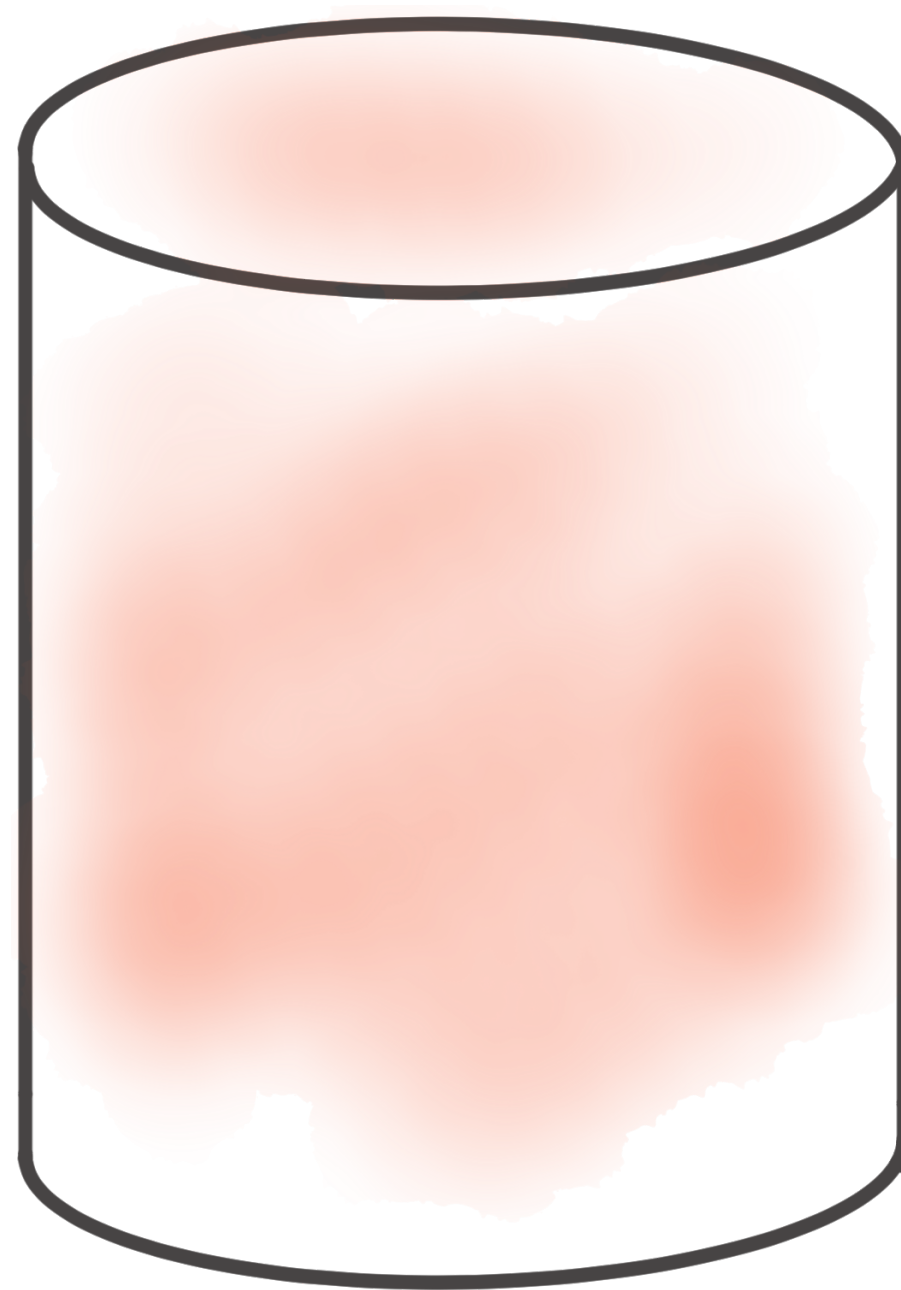


An Application Log Entry



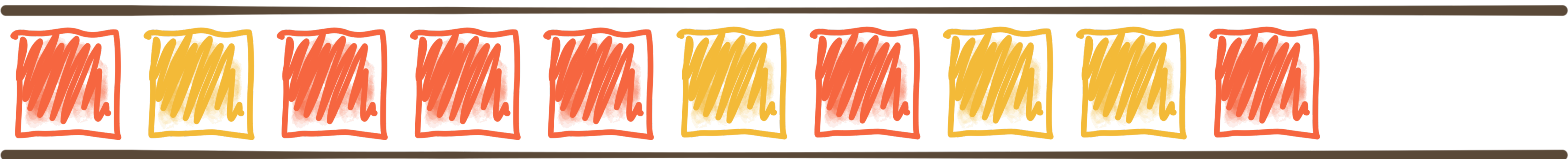
Databases





Immutable event log

Streams of events



Time →

Stream Processing with KSQL



Stream: widgets



Stream: widgets_red

Stream Processing with KSQL



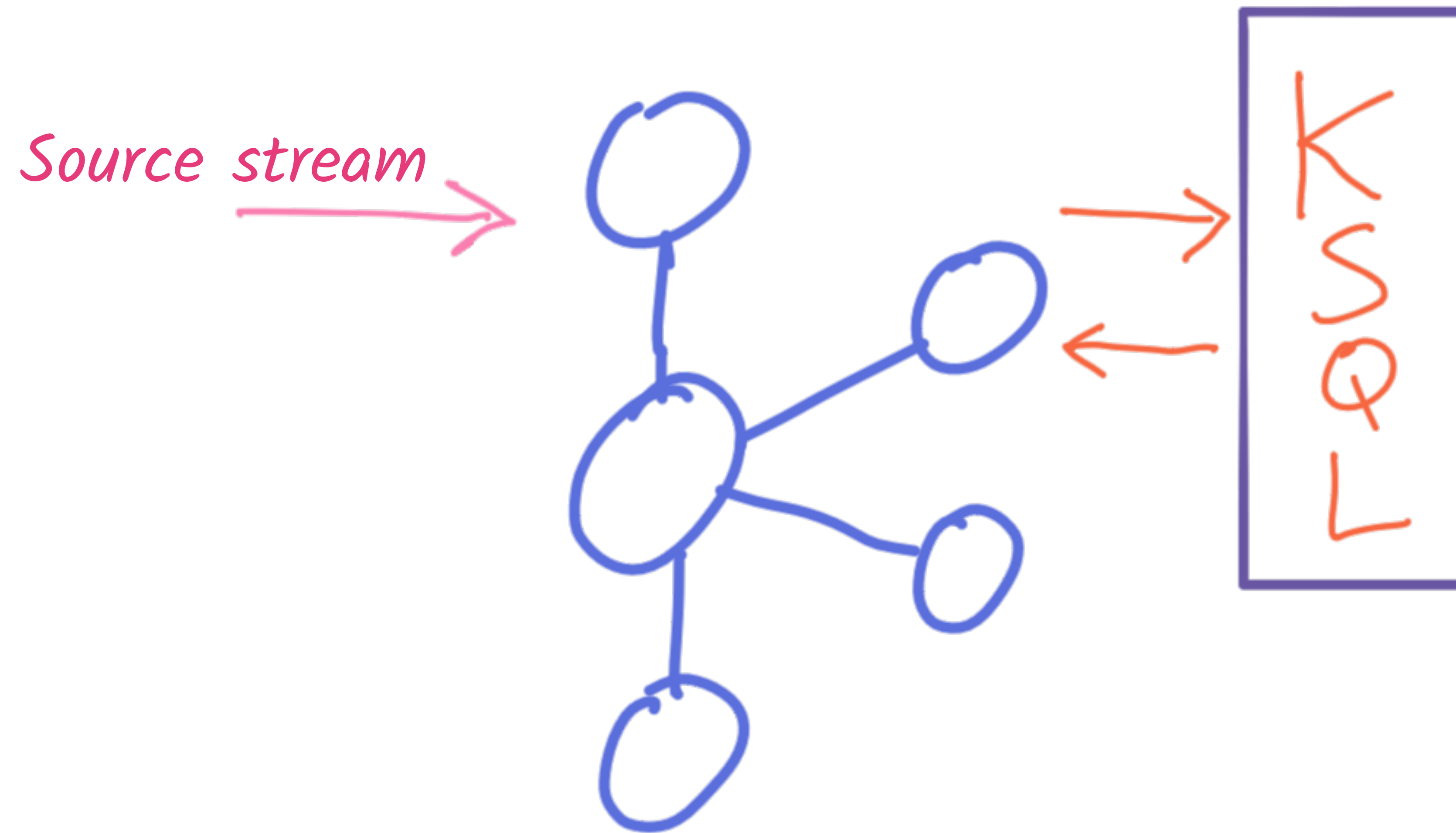
KSQL

↓

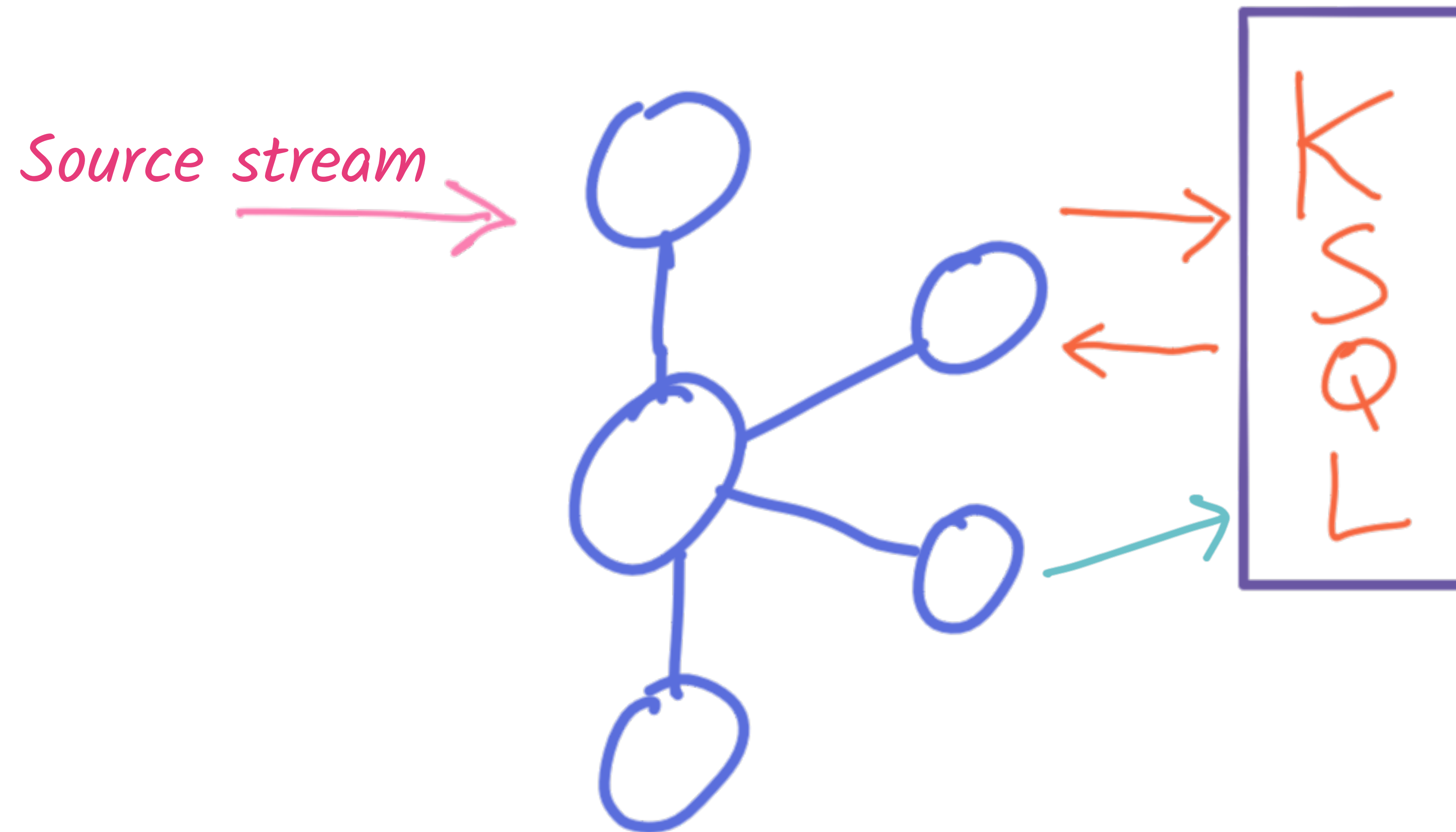
```
CREATE STREAM widgets_red AS
SELECT * FROM widgets
WHERE colour='RED';
```



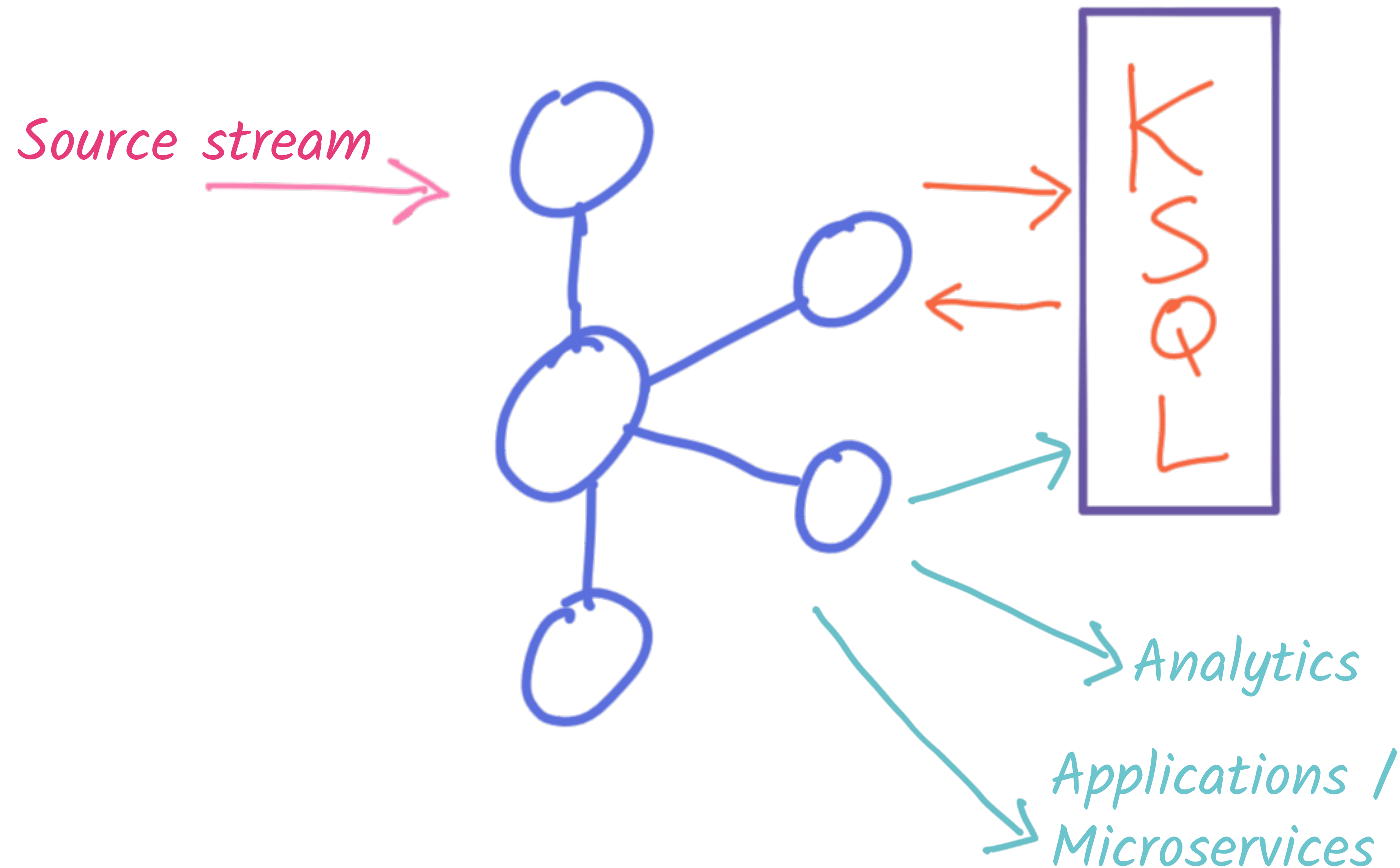
Stream Processing with KSQL



Stream Processing with KSQL



Stream Processing with KSQL



ksql in action



<https://rmoff.dev/kssf19-ksql-code>





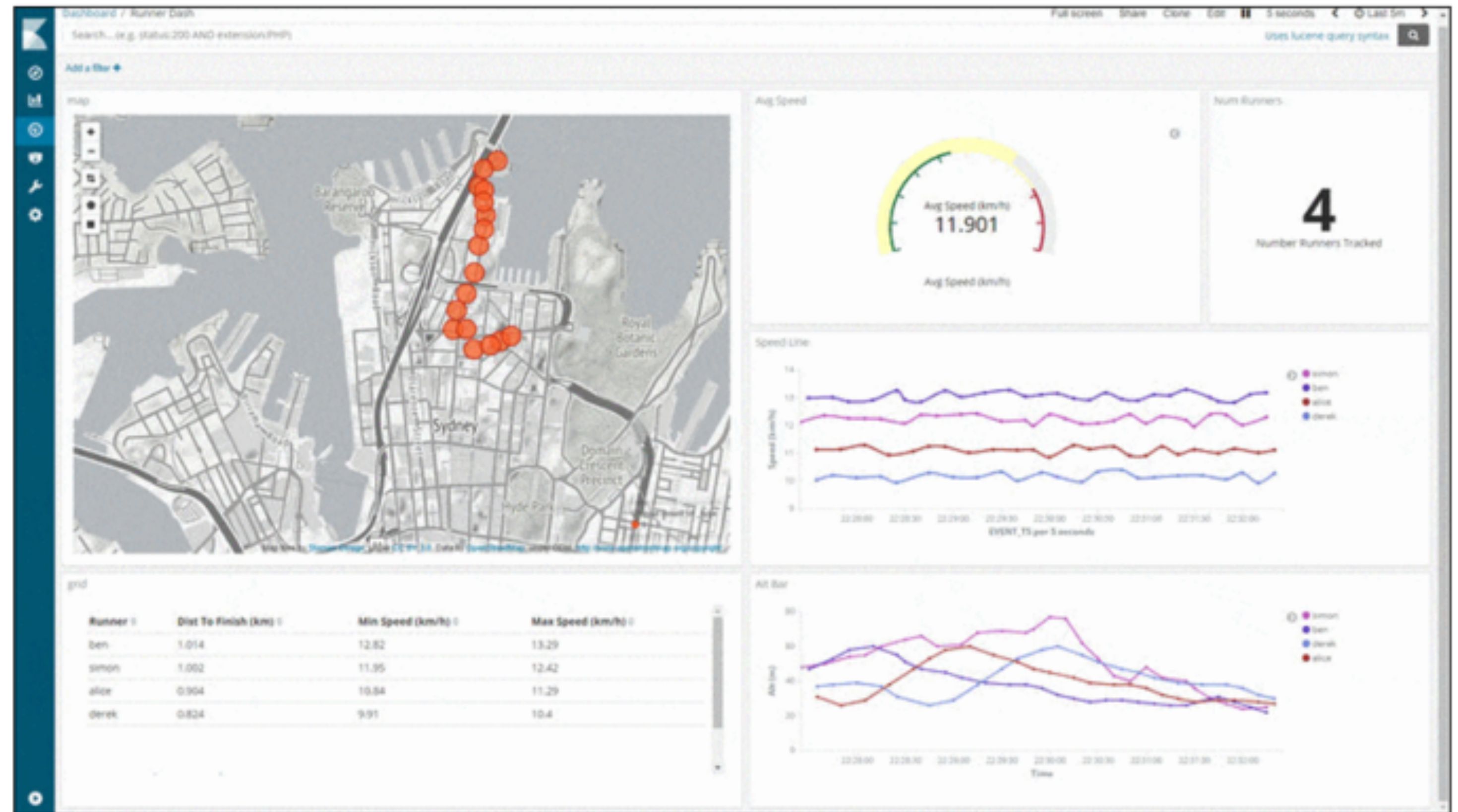
Did I beat Ben? Race tracking with Kafka KSQL, MQTT & Kibana



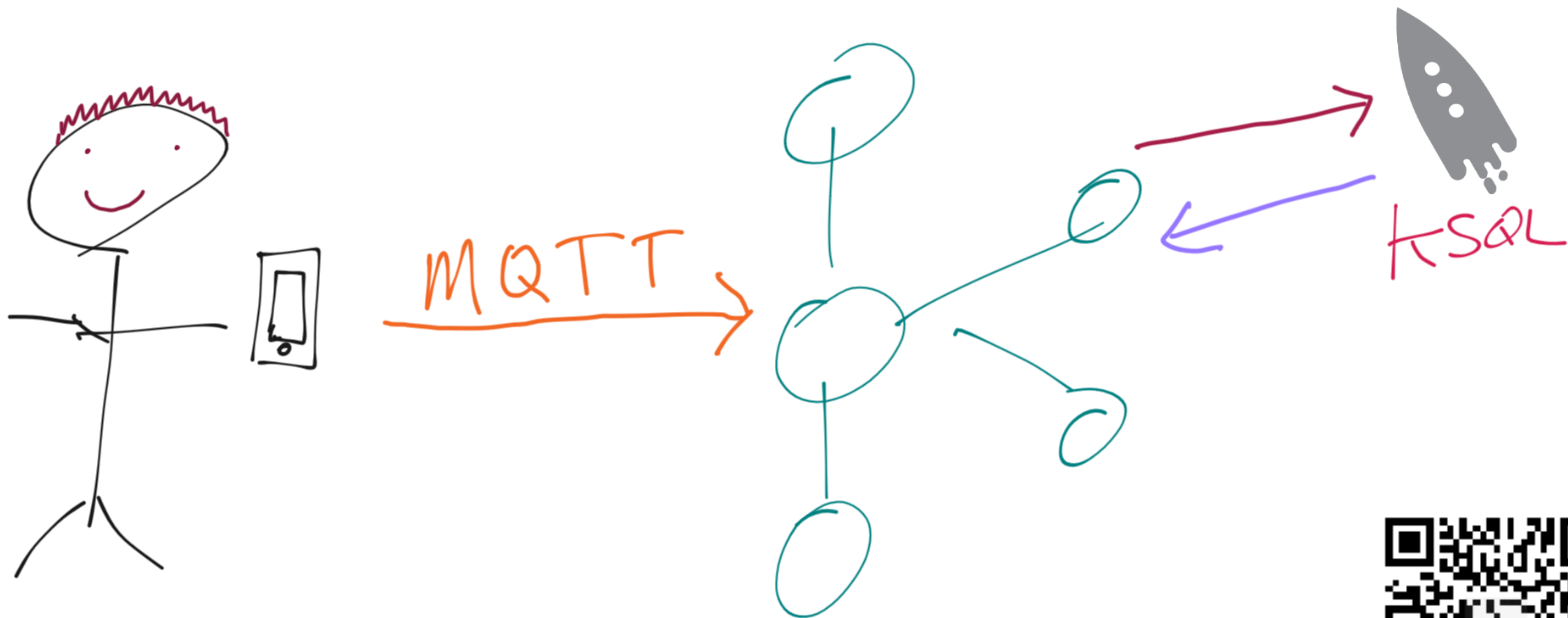
Simon Aubury [Follow](#)

May 28 · 4 min read

Live race tracking with MQTT and Kafka stream processing and visualisations with Kibana. Can I run a



Kibana dashboard — showing live race analytics



<https://rmoff.dev/kssf19-ksql-code>



DEMO

[**https://rmoff.dev/kssf19-ksql-code**](https://rmoff.dev/kssf19-ksql-code)



Code!

@rmoff #KafkaSummit

The screenshot shows the GitHub interface for the repository `confluentinc/demo-scene`. The repository has 51 Unwatch, 190 Stars, and 100 Forks. The main content area shows a description: "Scripts and samples to support Confluent Platform talks. May be rough around the edges. For automated tutorials and QA'd code, see <https://github.com/confluentinc/examples/>". Below this are tabs for `kafka`, `ksql`, `elasticsearch`, `syslog`, `mysql`, `debezium`, `kafka-streams`, and `kafka-connect`, along with a `Manage topics` link. A progress bar shows 721 commits, 24 branches, 0 packages, 0 releases, and 13 contributors. At the bottom, a list of recent commits is shown, with the latest commit by `rmoff` titled "Fix KSQL statement" dated 4 days ago. The commit list includes folders like `build-a-streaming-pipeline`, `ccloud-cube-demo`, `community-components-only`, `connect-5.3-improvements`, `connect-deepdive`, and `connect-error-handling`.

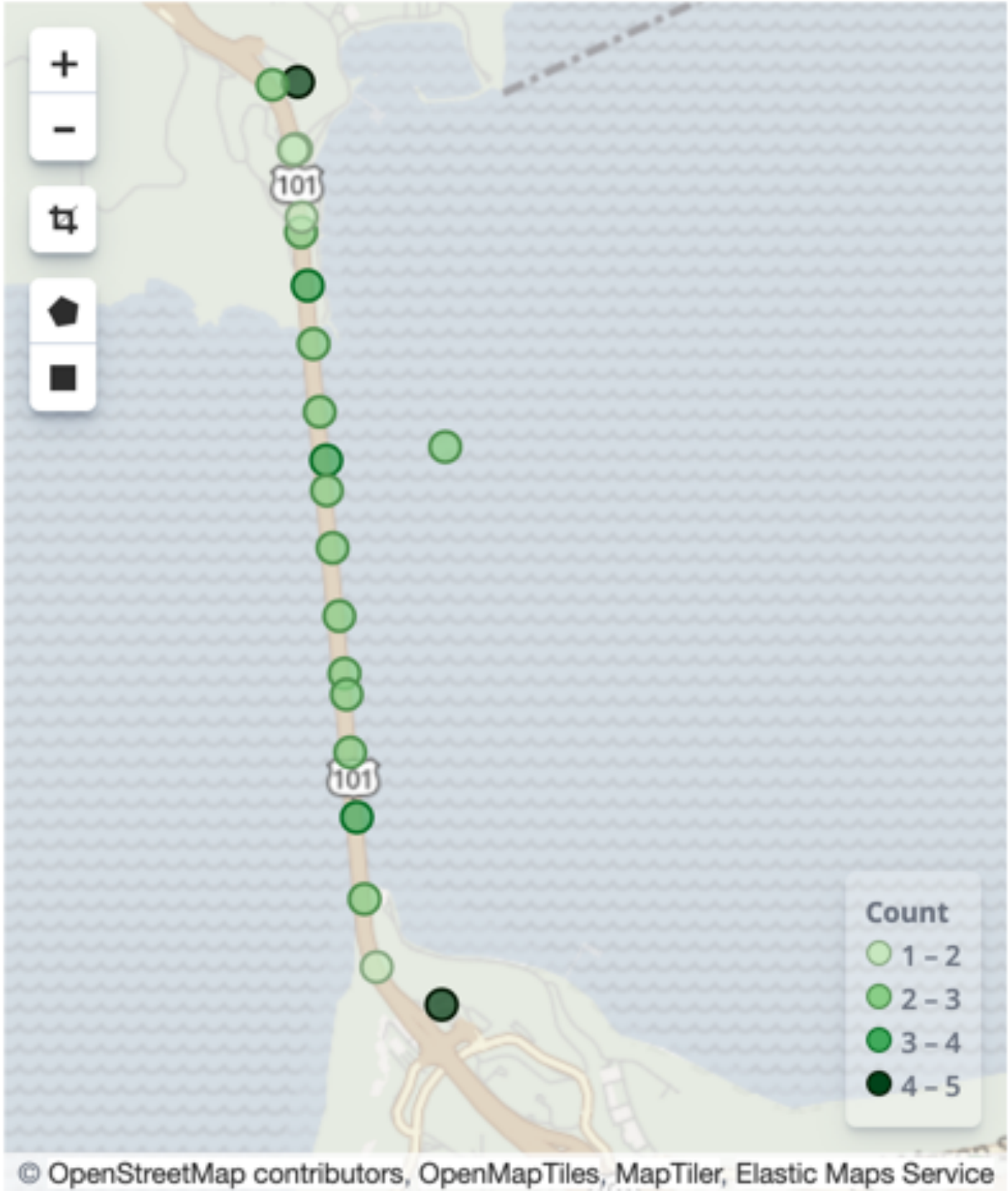
<https://rmoff.dev/kssf19-ksql-code>



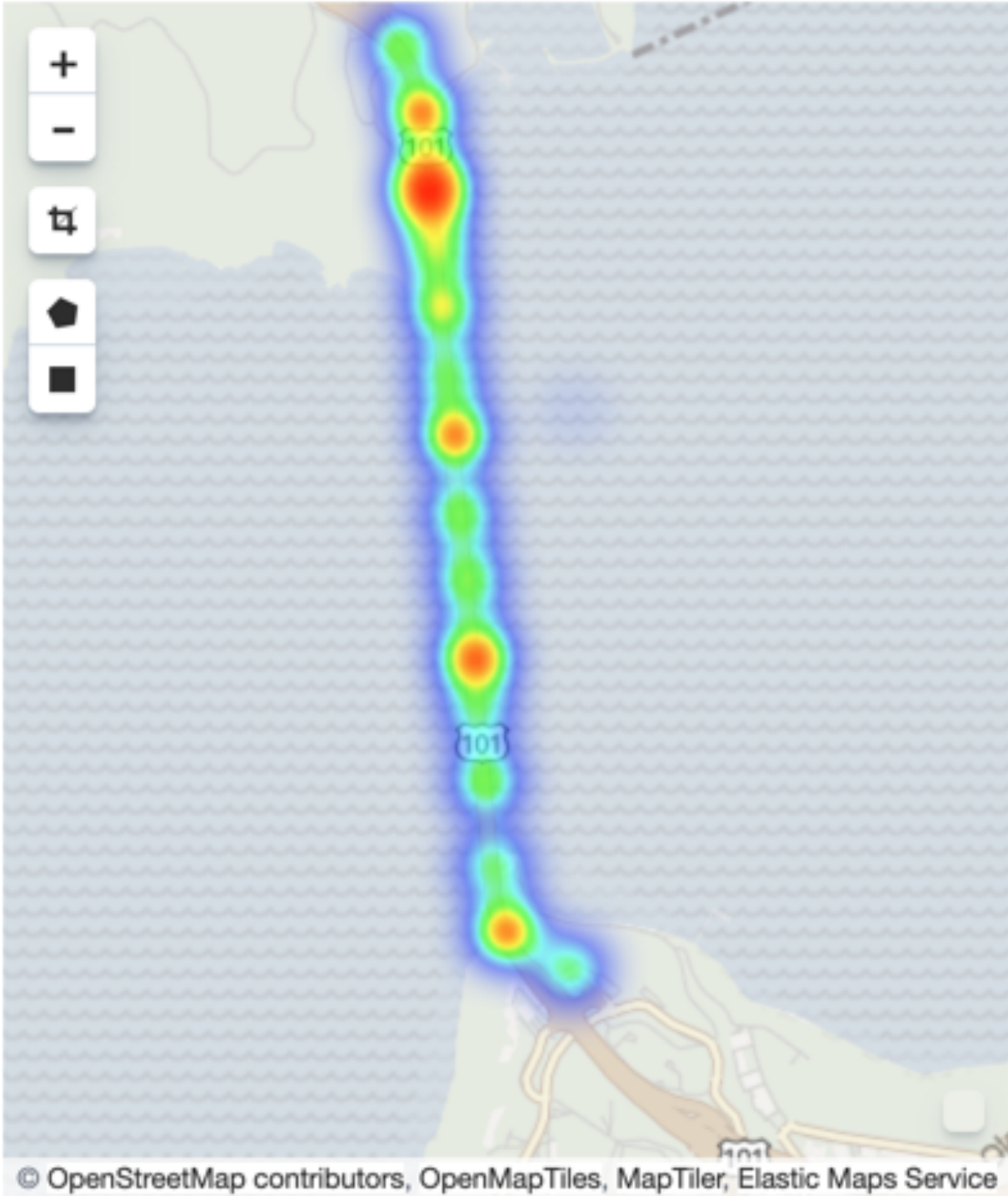
MQTT + Kafka + KSQL + Elastic = ❤️

@rmoff #KafkaSummit

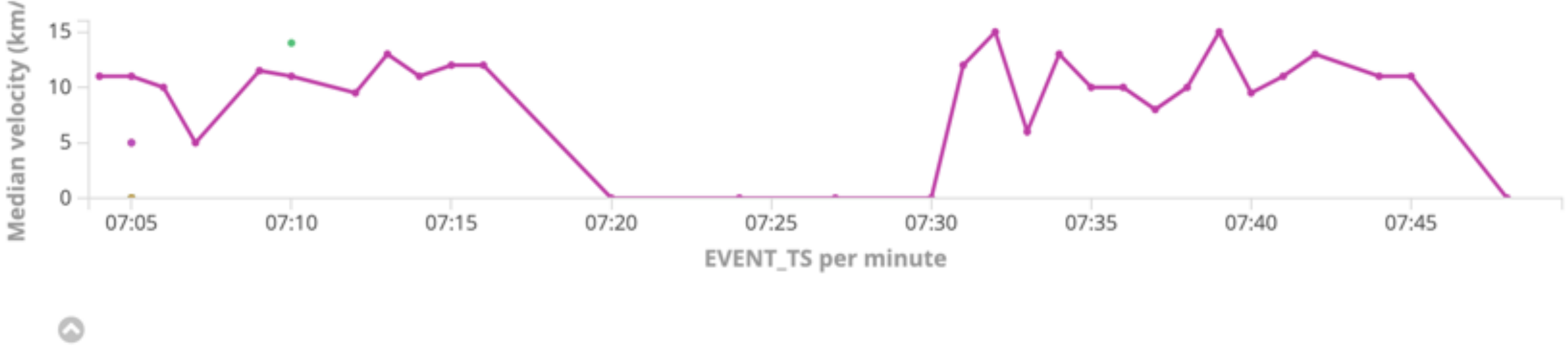
Map



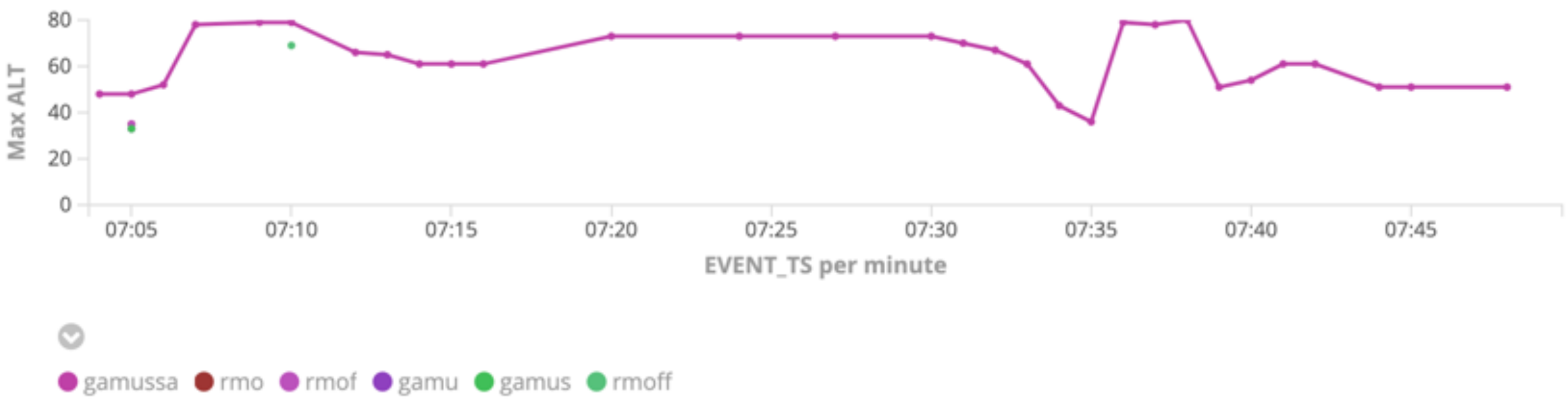
Velocity map



Speed



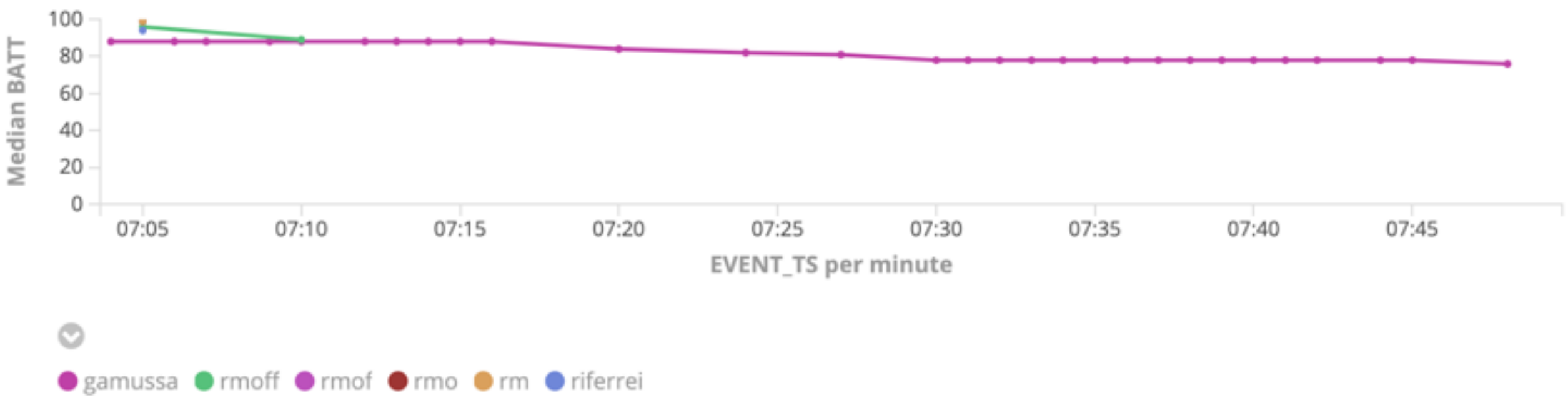
Altitude

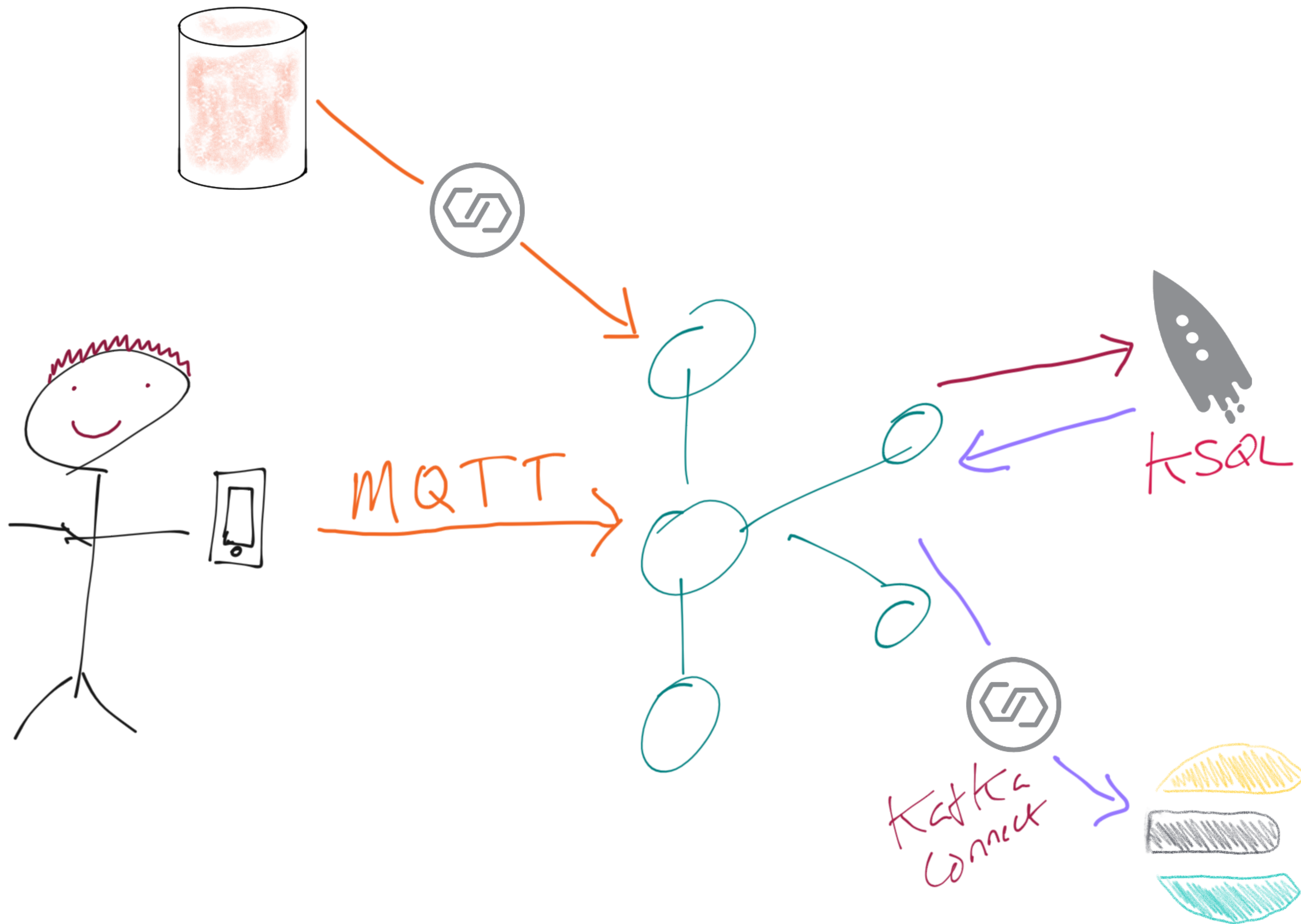


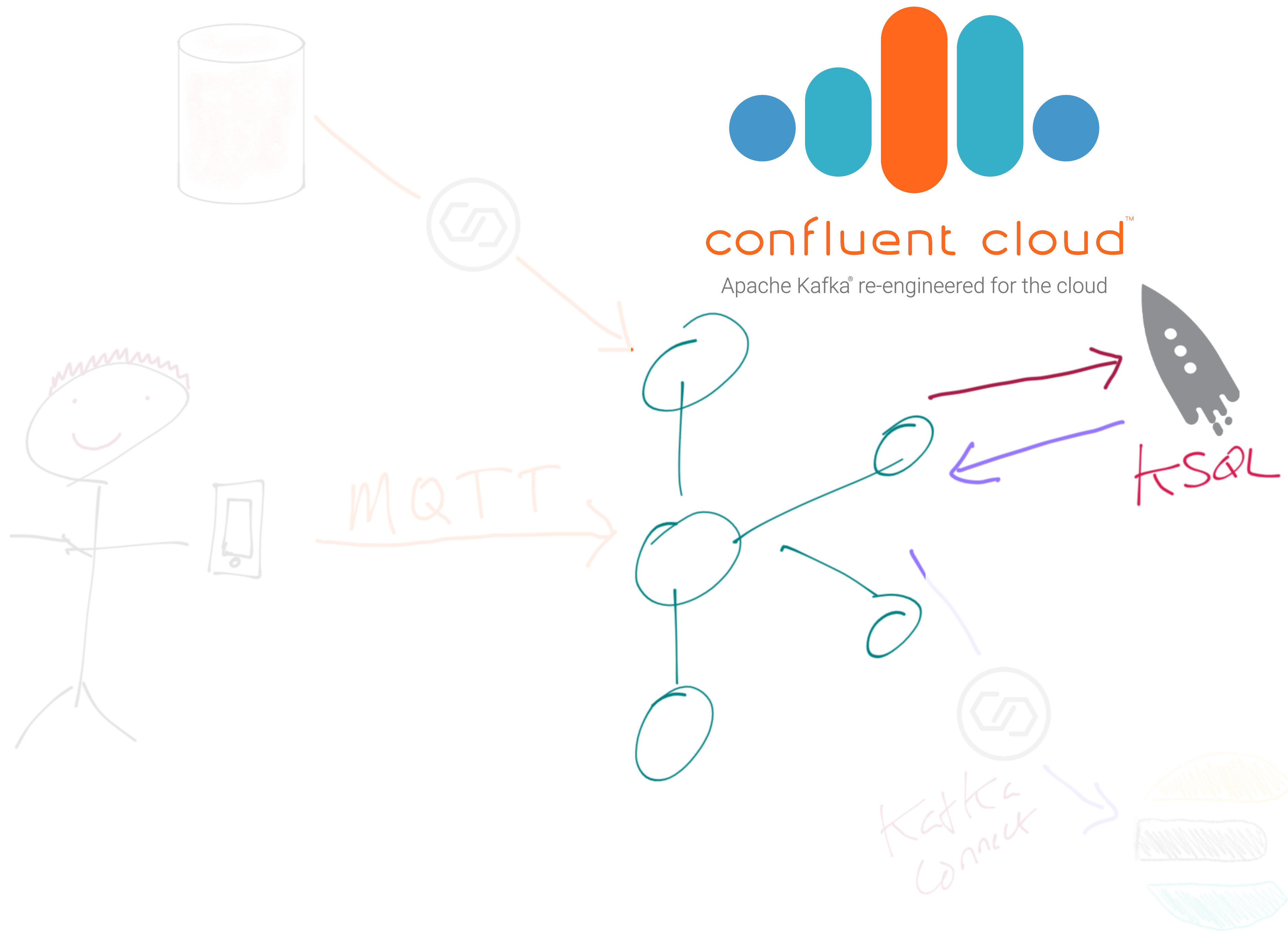
Location

1-50 of 59 < >								
Time	WHO	TID	P	CONN	BS	ALT	BATT	VEL
October 1st 2019, 07:48:47.523	gamussa	VG	100.969	m	1	48	76	0
October 1st 2019, 07:48:47.283	gamussa	VG	100.969	m	1	48	76	0
October 1st 2019, 07:48:21.059	gamussa	VG	100.972	m	1	51	76	11

Battery









confluent cloud

Fully Managed Kafka as a Service

<http://confluent.cloud/signup>

Interacting with KSQL



KSQL - Confluent Control Center

confluent

CO

Cluster 1

Query 1

1

2

3

4

5

6

SELECT

TIMESTAMPSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss') AS TS,

ORDERID,

ITEMID,

ORDERUNITS,

ADDRESS->STREET

FROM ORDERS ;

Add query properties

Running...

Run

Stop

Data structure

STREAM

Total messages

594

Messages/sec

3.94

Total message bytes

40896

Message fields

• TS

• ORDERID

• ITEMID

• ORDERUNITS

Editor

Filter by keyword

TS	ORDERID	ITEMID	ORDERUNITS	ADDRESS_STREET
2019-09-16 08:05:46	4905	Item_0	11	640 Fair Oaks Junction
2019-09-16 08:05:46	4904	Item_0	4	58 Meadow Vale Trail
2019-09-16 08:05:46	4903	Item_36	14	957 Parkside Court
2019-09-16 08:05:45	4902	Item_0	16	13210 Rieder Hill
2019-09-16 08:05:45	4901	Item_5	8	1 Park Meadow Junction
2019-09-16 08:05:45	4900	Item_22	10	2 Columbus Court

Flow

Streams

Tables

Running queries

KSQL - CLI

```
CLI v5.2.2, Server v5.2.2 located at http://ksql-server:8088
```

```
Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!
```

```
ksql>
```

```
ksql> CREATE STREAM ORDERS_NO_ADDRESS_DATA AS
```

```
> SELECT TIMESTAMPTOSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss') AS ORDER_TIMESTAMP,  
>         ORDERID,  
>         ITEMID,  
>         ORDERUNITS  
> FROM ORDERS;
```

```
Message
```

```
-----  
Stream created and running  
-----
```

```
ksql> select * from ORDERS_NO_ADDRESS_DATA;
```

```
1562059702636 | 0 | 2019-07-02 09:28:22 | 0 | Item_48 | 16  
1562059703535 | 0 | 2019-07-02 09:28:23 | 0 | Item_45 | 16  
1562059703638 | 1 | 2019-07-02 09:28:23 | 1 | Item_18 | 15  
1562059703804 | 2 | 2019-07-02 09:28:23 | 2 | Item_11 | 2  
1562059703826 | 2 | 2019-07-02 09:28:23 | 2 | Item_0 | 6
```


KSQL - REST API

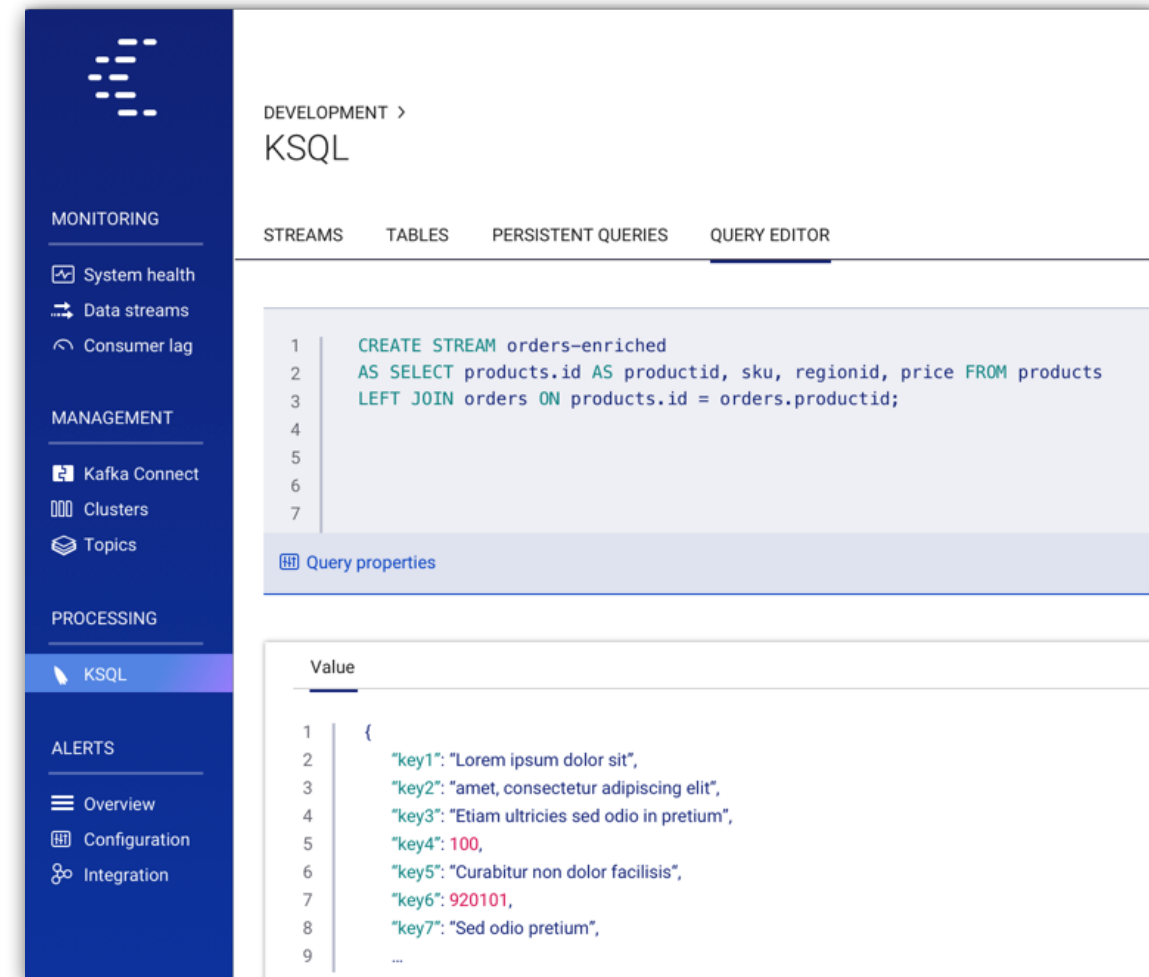
```
$ echo '{"ksql":"SELECT ORDERID, ITEMID, ADDRESS FROM ORDERS LIMIT 5;", "streamsProperties":  
      "ksql.streams.auto.offset.reset": "earliest"}}' | \  
http http://localhost:8088/query  
HTTP/1.1 200 OK  
Content-Encoding: gzip  
Content-Type: application/json  
Date: Tue, 02 Jul 2019 12:46:25 GMT  
Server: Jetty(9.4.14.v20181114)  
Transfer-Encoding: chunked  
Vary: Accept-Encoding, User-Agent  
  
{  
  "row": {  
    "columns": [0, "Item_0", {"STREET": "377 Maryland Place", "CITY": "Beaumont", "STATE": "Texas"},  
    "terminal": false  
  },  
  "row": {  
    "columns": [0, "Item_0", {"STREET": "072 Butternut Lane", "CITY": "Grand Junction", "STATE": "Colorado",  
    "sage": null, "terminal": false  
  },  
  "row": {  
    "columns": [1, "Item_0", {"STREET": "703 Hoffman Place", "CITY": "Mountain View", "STATE": "California",  
    "sage": null, "terminal": false  
  },  
  "row": {  
    "columns": [2, "Item_0", {"STREET": "0 Dorton Circle", "CITY": "Brooklyn", "STATE": "New York"},  
    "terminal": false  
  },  
  "row": {  
    "columns": [3, "Item_0", {"STREET": "404 Mayer Park", "CITY": "Lubbock", "STATE": "Texas"}],  
    "terminal": false  
  },  
  "row": null, "errorMessage": null, "finalMessage": "Limit Reached", "terminal": true  
}
```


ksql operations and deployment



KSQL in Development and Production

Interactive KSQL
for development and testing

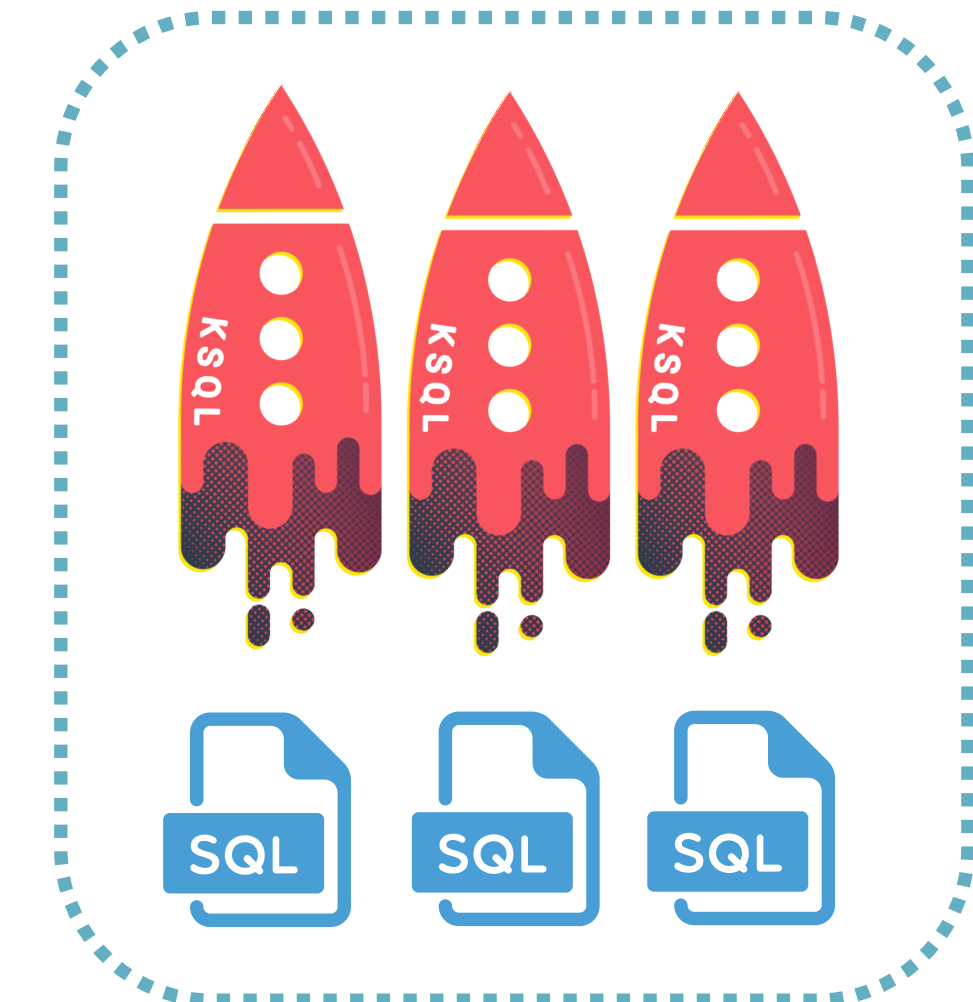


REST

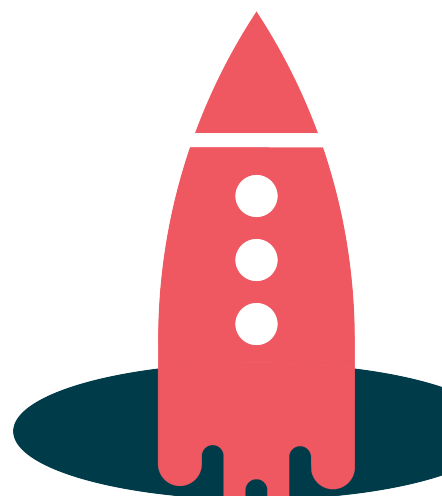


Desired KSQL queries
have been identified

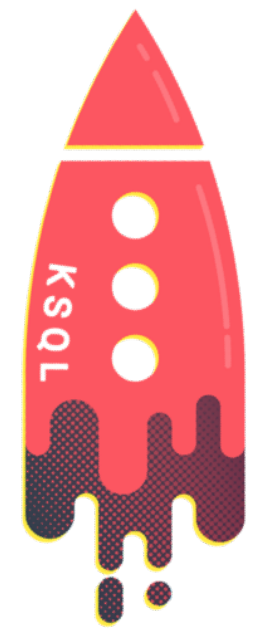
Headless KSQL
for Production



"Hmm, let me try
out this idea..."



How to run KSQL



KSQL Server
(JVM process)

DEB, RPM, ZIP, TAR downloads

<http://confluent.io/ksql>

Docker images

[confluentinc/cp-ksql-server](#)

[confluentinc/cp-ksql-cli](#)



Physical



docker



kubernetes



openstack®

vmware®



Microsoft
Azure



Google Cloud Platform



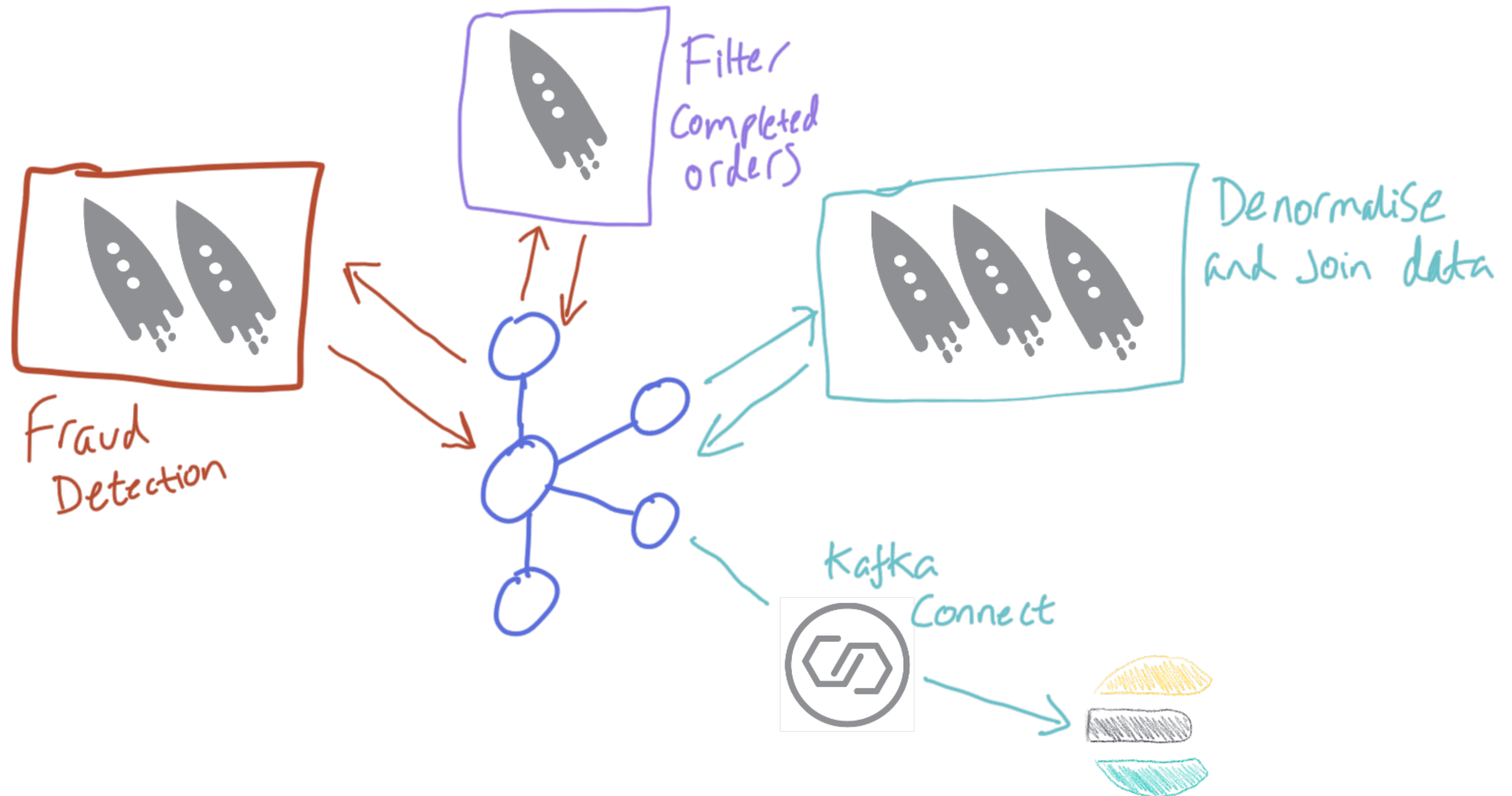
amazon
web services

...and many more...

ONE DOES NOT SIMPLY RUN

A SINGLE KSQL CLUSTER

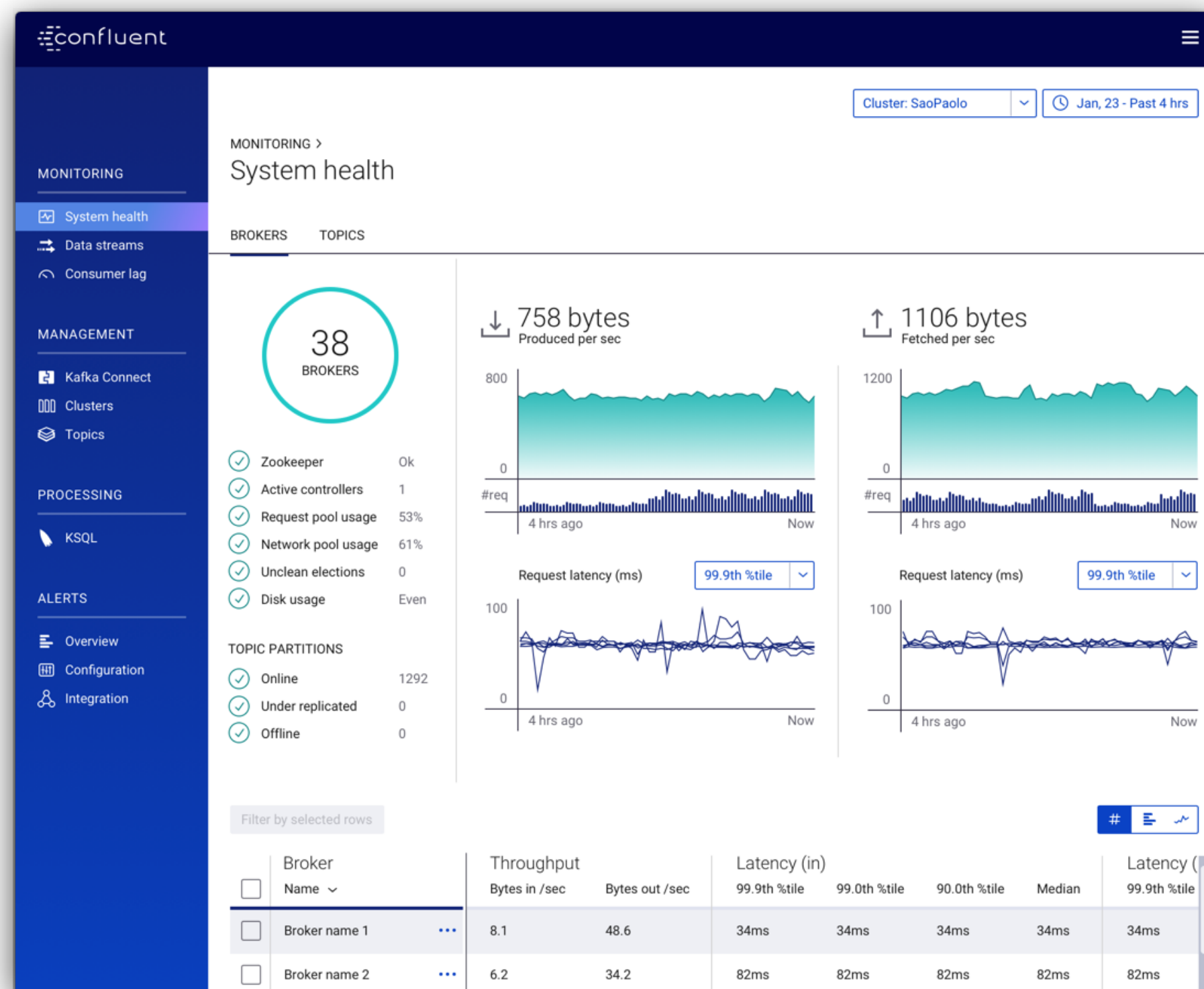
Think *Applications*, not database instances



Monitoring KSQL

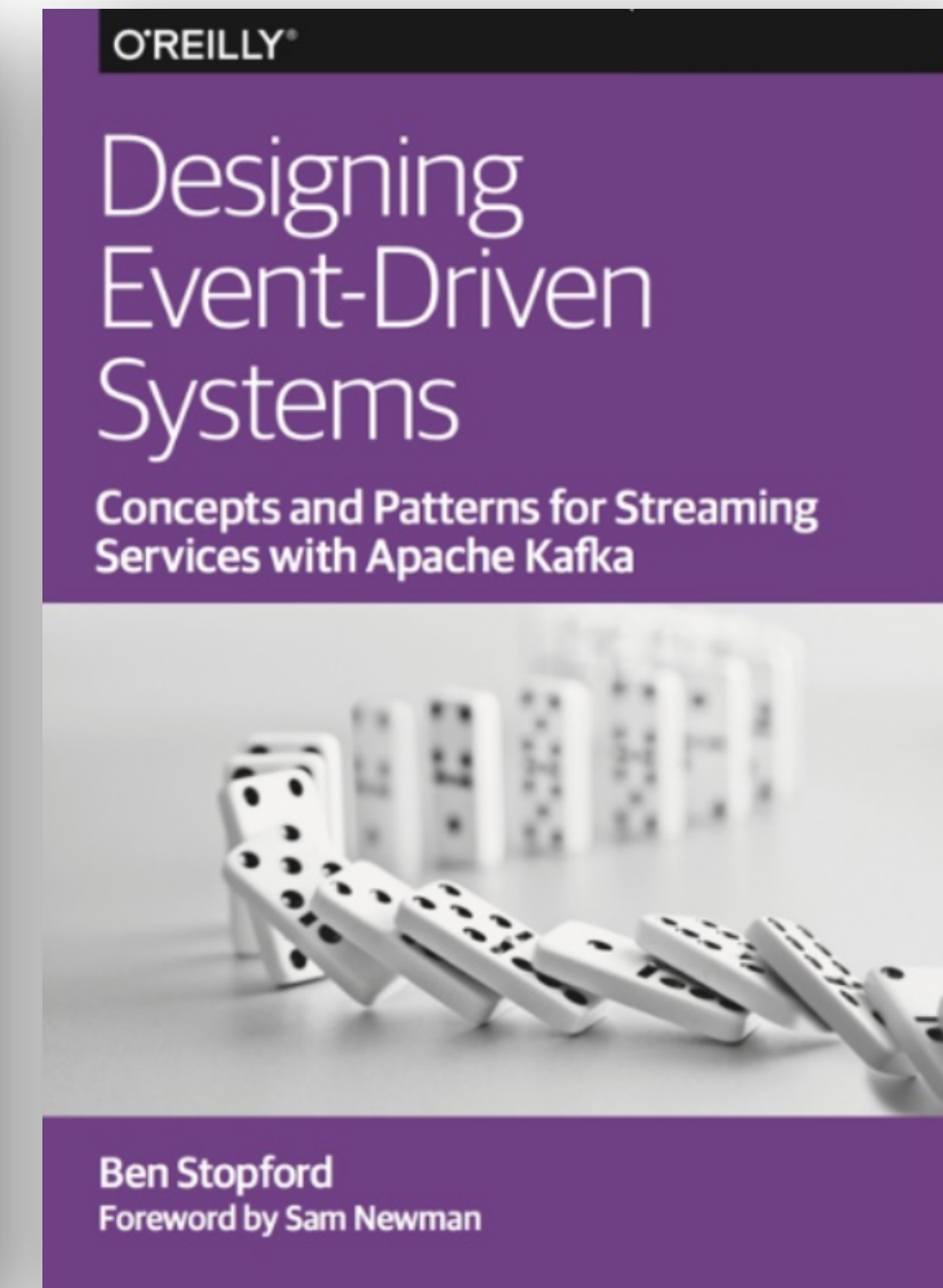
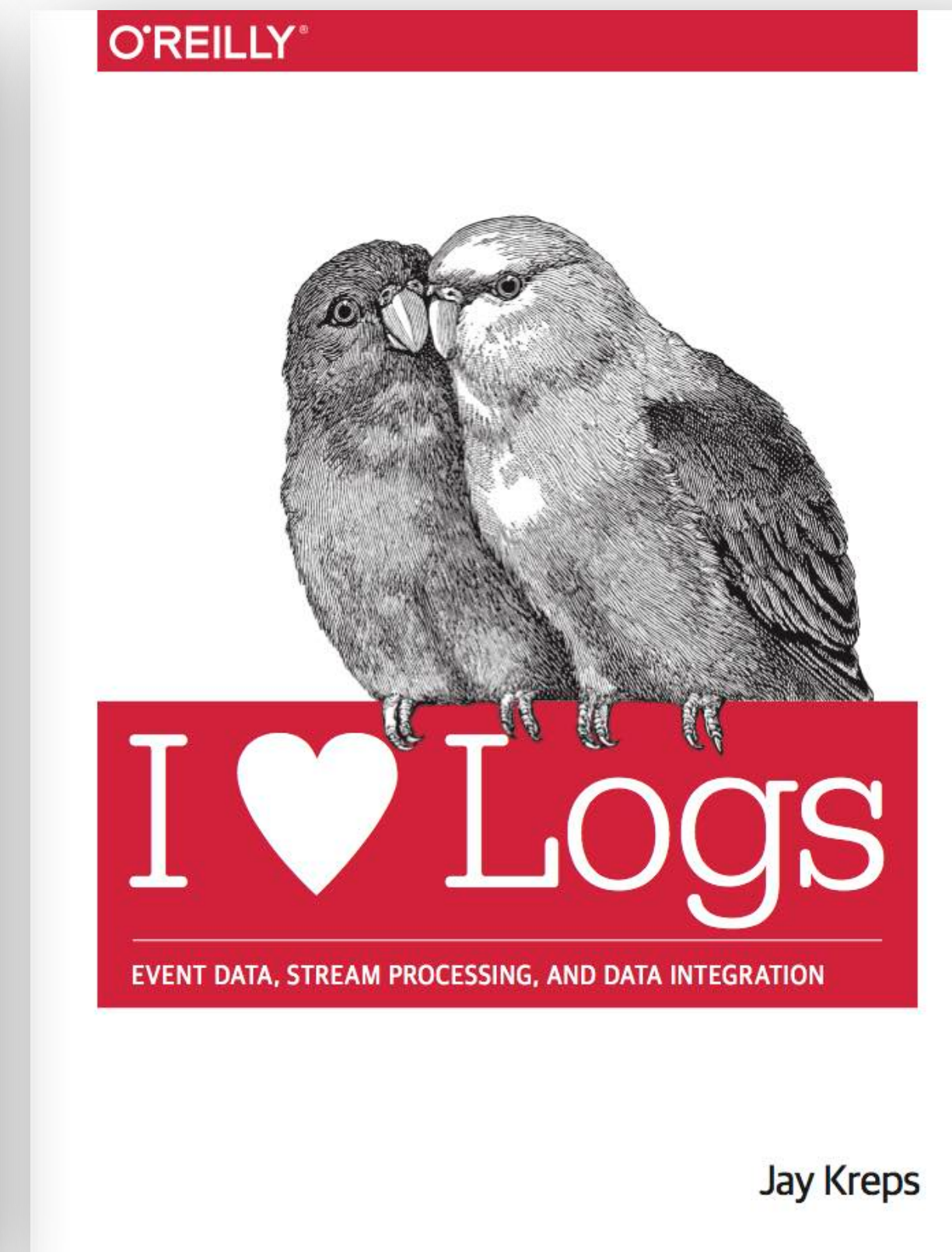
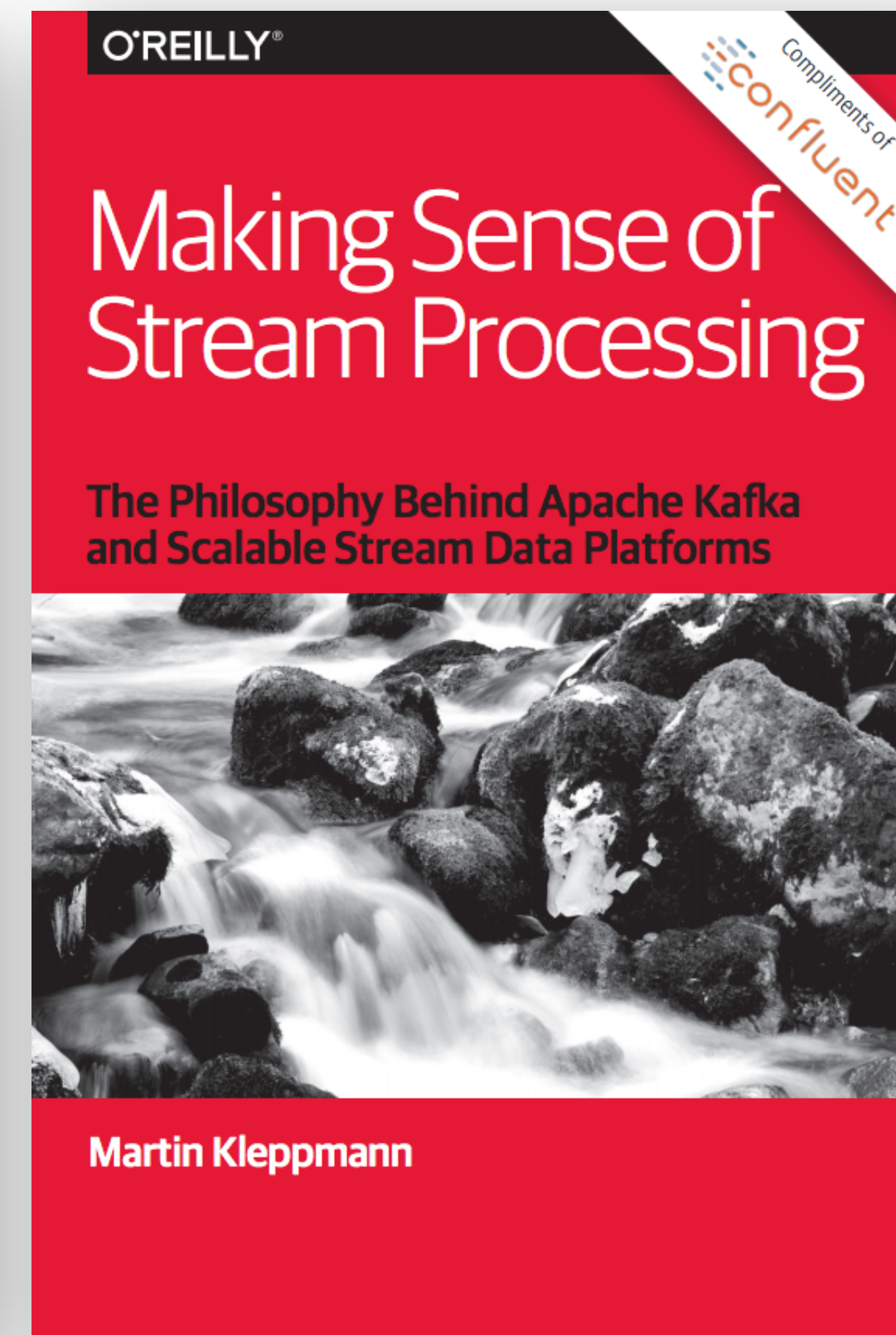
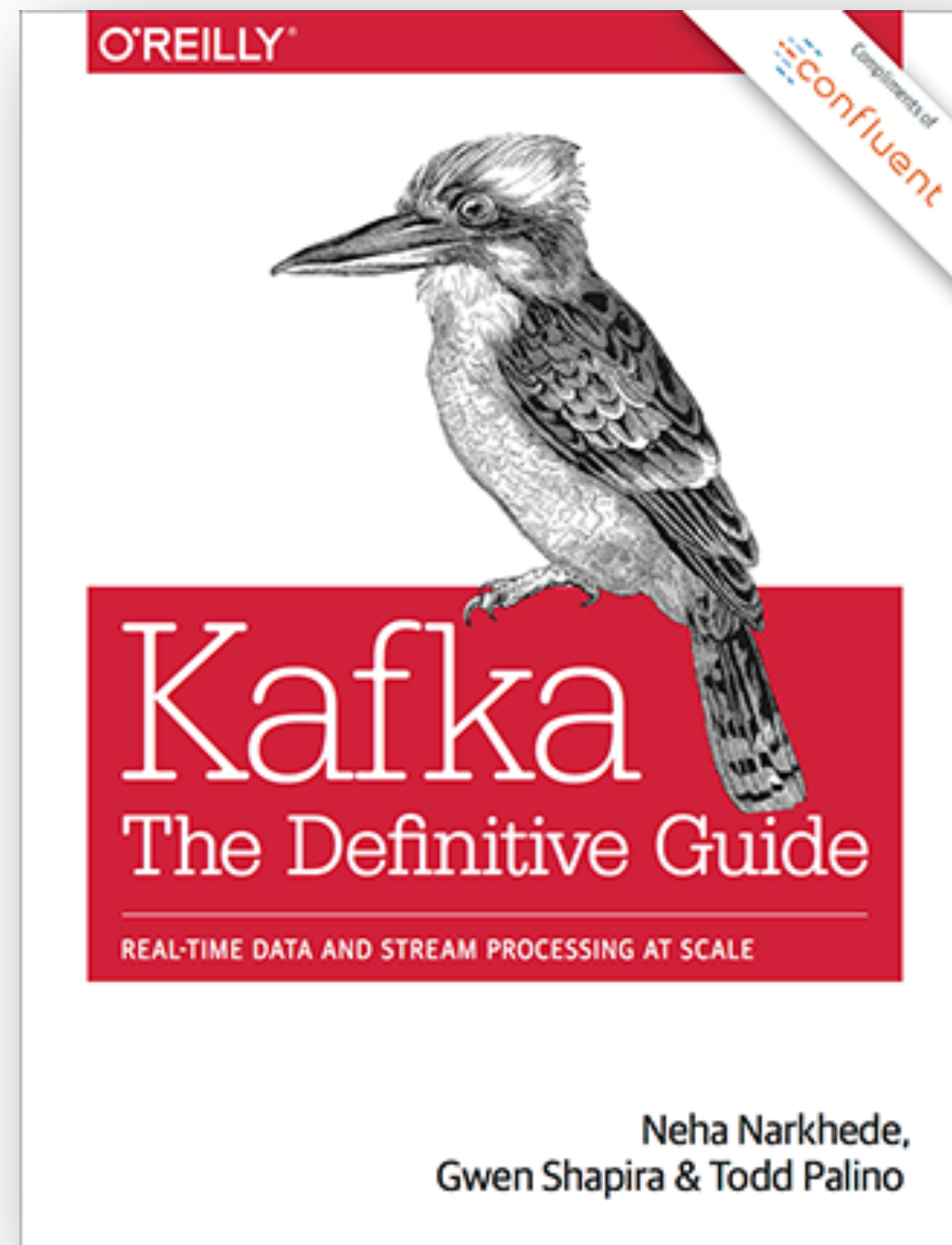
Confluent Control Center

JMX



<https://www.confluent.io/blog/troubleshooting-ksql-part-2>

<http://cnfl.io/book-bundle>



@rmoff

#KafkaSummit

#EOF



Join the Confluent Community Slack group at <http://cnfl.io/slack>

<https://talks.rmoff.net>

Related Talks

- The Changing Face of ETL:
Event-Driven Architectures for Data Engineers

-  [Slides](#)
-  [Recording](#)

- ATM Fraud detection with Kafka and KSQL

-  [Slides](#)
-  [Code](#)
-  [Recording](#)

- Embrace the Anarchy: Apache Kafka's Role in Modern Data Architectures

-  [Slides](#)
-  [Recording](#)

- Apache Kafka and KSQL in Action : Let's Build a Streaming Data Pipeline!

-  [Slides](#)
-  [Code](#)
-  [Recording](#)

- No More Silos: Integrating Databases and Apache Kafka

-  [Slides](#)
-  [Code \(MySQL\)](#)
-  [Code \(Oracle\)](#)
-  [Recording](#)

**Bonus
content!**

ksql in action



Filtering with KSQL

NY CA CA NY **ORDERS**

Filtering with KSQL

NY CA CA NY **ORDERS**

KSQL ↓

```
CREATE STREAM ORDERS_NY AS
  SELECT *
    FROM ORDERS
   WHERE ADDRESS->STATE='New York';
```

Filtering with KSQL

NY CA CA NY **ORDERS**

KSQL ↓

```
CREATE STREAM ORDERS_NY AS  
  SELECT *  
    FROM ORDERS  
   WHERE ADDRESS->STATE='New York';
```

NY NY **ORDERS_NY**

Schema manipulation with KSQL



ORDERS

```
{ "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }  
}
```


Schema manipulation with KSQL



ORDERS

KSQL



```
CREATE STREAM ORDERS_NO_ADDRESS_DATA AS  
  SELECT ORDERTIME, ORDERID, ITEMID, ORDERUNITS  
  FROM ORDERS;
```

```
{ "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }  
}
```

Schema manipulation with KSQL



ORDERS

```
{ "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }}
```

KSQL



```
CREATE STREAM ORDERS_NO_ADDRESS_DATA AS  
  SELECT TIMESTAMPSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss')  
        AS ORDER_TIMESTAMP,  
        ORDERID, ITEMID, ORDERUNITS  
  FROM ORDERS;
```



ORDERS_NO_ADDRESS_DATA

```
{ "order_ts": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5  
}
```

Schema manipulation with KSQL



```
{  
  "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }  
}
```


Schema manipulation with KSQL



KSQL

```
CREATE STREAM ORDERS_FLAT AS
```

```
  SELECT [...]
```

```
    ADDRESS->STREET AS ADDRESS_STREET,
```

```
    ADDRESS->CITY AS ADDRESS_CITY,
```

```
    ADDRESS->STATE AS ADDRESS_STATE
```

```
  FROM ORDERS;
```

```
{  
  "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }  
}
```

Schema manipulation with KSQL



ORDERS

KSQL

```
CREATE STREAM ORDERS_FLAT AS
```

```
  SELECT [...]
```

```
    ADDRESS->STREET AS ADDRESS_STREET,
```

```
    ADDRESS->CITY AS ADDRESS_CITY,
```

```
    ADDRESS->STATE AS ADDRESS_STATE
```

```
  FROM ORDERS;
```



ORDERS_FLAT

```
{  
  "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address": {  
    "street": "243 Utah Way",  
    "city": "Orange",  
    "state": "California"  
  }  
}
```

```
{"ordertime": 1560070133853,  
 "orderid": 67,  
 "itemid": "Item_9",  
 "orderunits": 5,  
 "address-street": "243 Utah Way",  
 "address-city": "Orange",  
 "address-state": "California"}
```

Reserialising data with KSQL



ORDERS

```
{"ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address-street": "243 Utah Way",  
  "address-city": "Orange",  
  "address-state": "California"}
```


Reserialising data with KSQL



ORDERS

```
{"ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address-street": "243 Utah Way",  
  "address-city": "Orange",  
  "address-state": "California"}
```

KSQL



```
CREATE STREAM ORDERS_CSV  
  WITH (VALUE_FORMAT='DELIMITED') AS  
  SELECT * FROM ORDERS_FLAT;
```

Reserialising data with KSQL



ORDERS

```
{ "ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5,  
  "address-street": "243 Utah Way",  
  "address-city": "Orange",  
  "address-state": "California" }
```

KSQL



```
CREATE STREAM ORDERS_CSV  
  WITH (VALUE_FORMAT='DELIMITED') AS  
  SELECT * FROM ORDERS_FLAT;
```



ORDERS_CSV

```
1560045914101,24644,Item_0,1,43078 De  
1560047305664,24643,Item_29,3,209 Mon  
1560057079799,24642,Item_38,18,3 Autu  
1560088652051,24647,Item_6,6,82893 Ar  
1560105559145,24648,Item_0,12,45896 W  
1560108336441,24646,Item_33,4,272 Hef  
1560123862235,24641,Item_15,16,0 Dort  
1560124799053,24645,Item_12,1,71 Knut
```

Lookups and Joins with KSQL

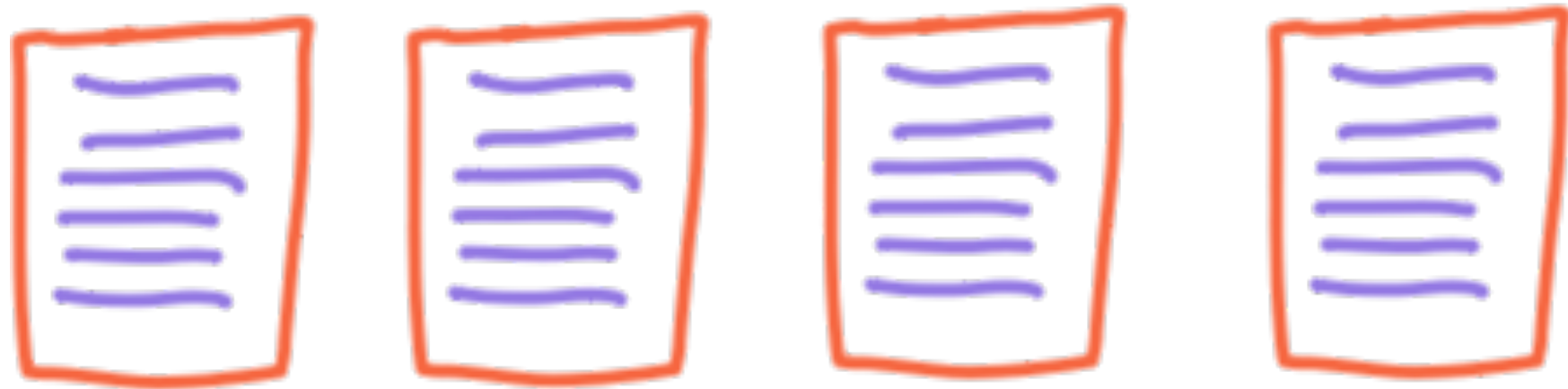


```
{"ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5}
```


Lookups and Joins with KSQL

Lookup

ITEMS



ORDERS

```
{  
  "id": "Item_9",  
  "make": "Boyle-McDermott",  
  "model": "Apiaceae",  
  "unit_cost": 19.9  
}  
  
{"ordertime": 1560070133853,  
  "orderid": 67,  
  "itemid": "Item_9",  
  "orderunits": 5}
```

Lookups and Joins with KSQL

Lookup

ITEMS



ORDERS

KSQL



```
CREATE STREAM ORDERS_ENRICHED AS
SELECT O.*, I.*,
       O.ORDERUNITS * I.UNIT_COST
       AS TOTAL_ORDER_VALUE,
FROM ORDERS O
      INNER JOIN ITEMS I
      ON O.ITEMID = I.ID ;
```

```
{
  "id": "Item_9",
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
  "unit_cost": 19.9
}
{"ordertime": 1560070133853,
 "orderid": 67,
 "itemid": "Item_9",
 "orderunits": 5}
```

Lookups and Joins with KSQL

Lookup

ITEMS

ORDERS

```
CREATE STREAM ORDERS_ENRICHED AS
SELECT O.*, I.*,
       O.ORDERUNITS * I.UNIT_COST
       AS TOTAL_ORDER_VALUE,
FROM ORDERS O
      INNER JOIN ITEMS I
      ON O.ITEMID = I.ID ;
```

ORDERS_ENRICHED

```
{
  "id": "Item_9",
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
  "unit_cost": 19.9
}
{"ordertime": 1560070133853,
 "orderid": 67,
 "itemid": "Item_9",
 "orderunits": 5}
```

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "make": "Boyle-McDermott",
  "model": "Apiaceae",
  "unit_cost": 19.9,
  "total_order_value": 99.5
}
```


Connecting to other systems with Kafka Connect

Lookup

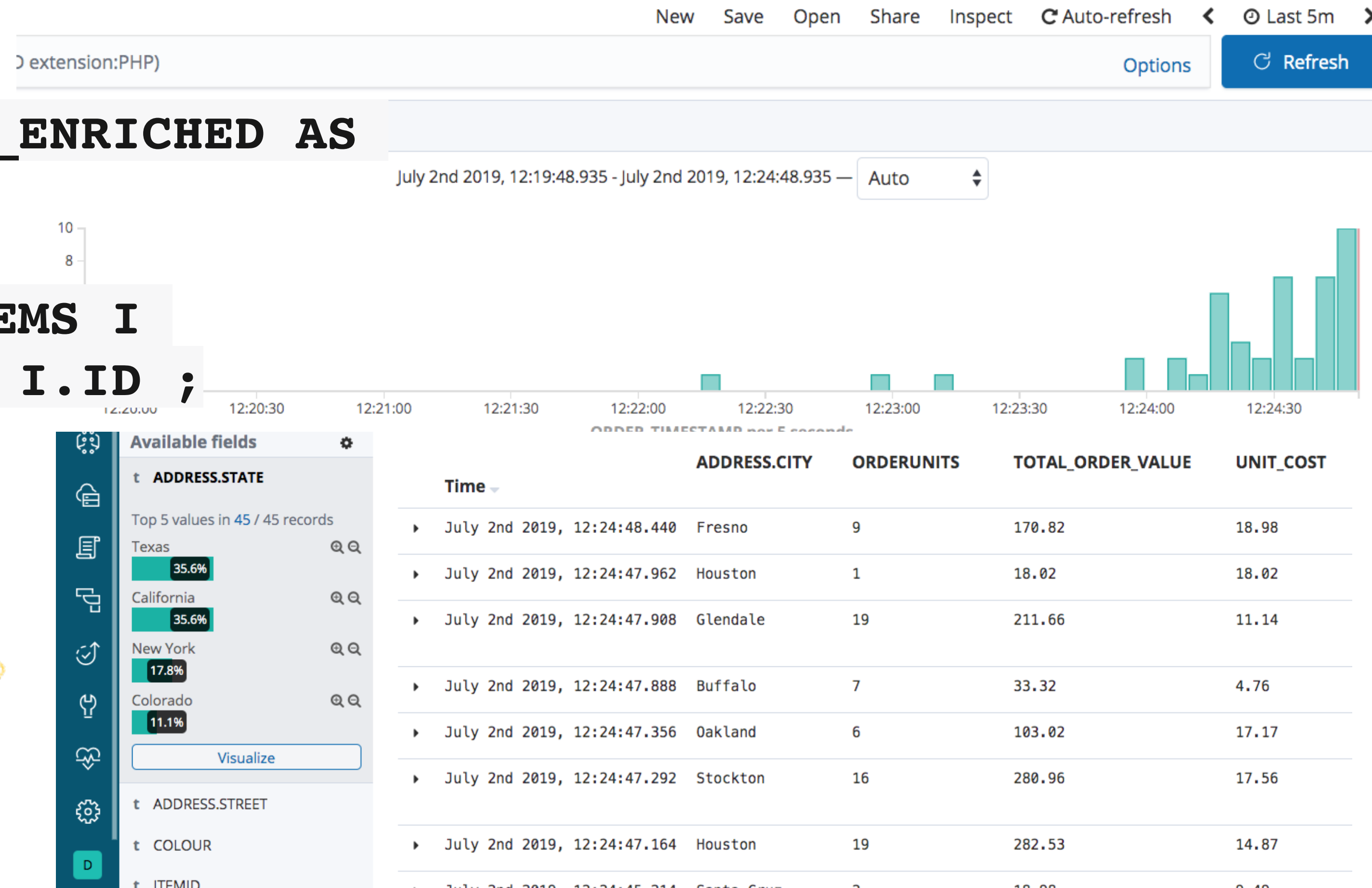


KSQL

```
CREATE STREAM ORDERS_ENRICHED AS
SELECT [...]
FROM ORDERS O
INNER JOIN ITEMS I
ON O.ITEMID = I.ID ;
```



Kafka Connect



Stateful Aggregation with KSQL



Stateful Aggregation with KSQL



**SELECT MAKE, COUNT(*) AS ORDER_COUNT
FROM ORDERS_ENRICHED
GROUP BY MAKE;**

Stateful Aggregation with KSQL



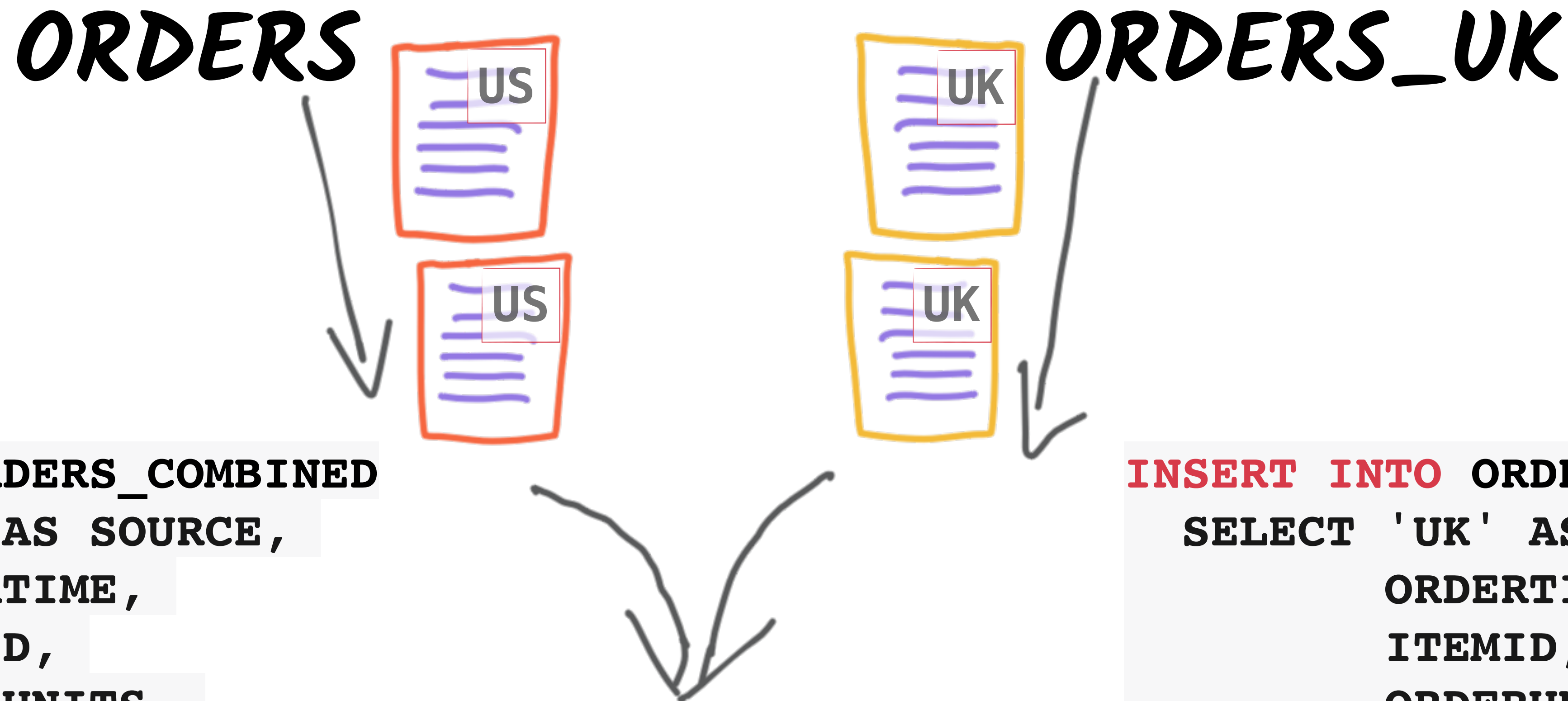
**SELECT MAKE, COUNT(*) AS ORDER_COUNT
FROM ORDERS_ENRICHED
GROUP BY MAKE;**

COUNT=4

Transform data with KSQL - merge streams



Transform data with KSQL - merge streams

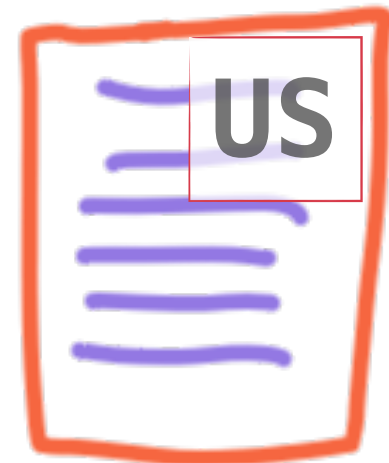


```
INSERT INTO ORDERS_COMBINED  
  SELECT 'US' AS SOURCE,  
         ORDERTIME,  
         ITEMID,  
         ORDERUNITS,  
         ADDRESS  
  FROM ORDERS;
```

```
INSERT INTO ORDERS_COMBINED  
  SELECT 'UK' AS SOURCE,  
         ORDERTIME,  
         ITEMID,  
         ORDERUNITS,  
         ADDRESS  
  FROM ORDERS_UK;
```


Transform data with KSQL - merge streams

ORDERS



ORDERS_UK

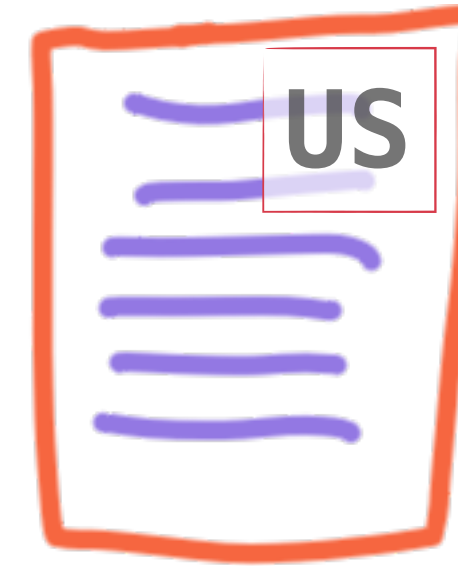
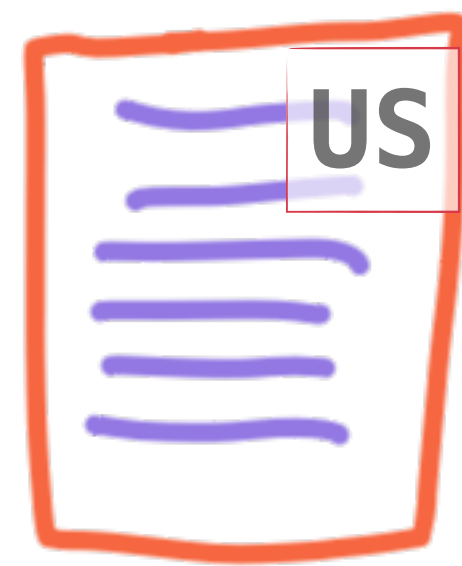
```
INSERT INTO ORDERS_COMBINED
SELECT 'US' AS SOURCE,
       ORDERTIME,
       ITEMID,
       ORDERUNITS,
       ADDRESS
FROM ORDERS;
```

```
INSERT INTO ORDERS_COMBINED
SELECT 'UK' AS SOURCE,
       ORDERTIME,
       ITEMID,
       ORDERUNITS,
       ADDRESS
FROM ORDERS_UK;
```



ORDERS_COMBINED

Transform data with KSQL - split streams



ORDERS_COMBINED

Transform data with KSQL - split streams

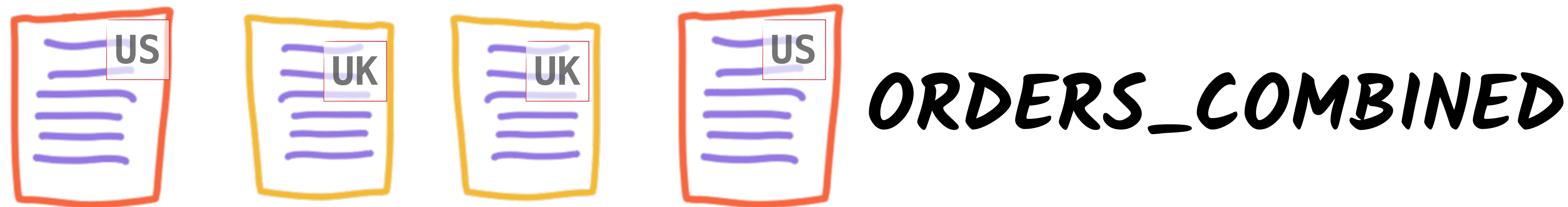


ORDERS_COMBINED

```
CREATE STREAM ORDERS_US AS  
SELECT *  
FROM ORDERS_COMBINED  
WHERE SOURCE = 'US' ;
```

```
CREATE STREAM ORDERS_UK AS  
SELECT *  
FROM ORDERS_COMBINED  
WHERE SOURCE = 'UK' ;
```


Transform data with KSQL - split streams



```
CREATE STREAM ORDERS_US AS  
SELECT *  
FROM ORDERS_COMBINED  
WHERE SOURCE = 'US' ;
```

```
CREATE STREAM ORDERS_UK AS  
SELECT *  
FROM ORDERS_COMBINED  
WHERE SOURCE = 'UK' ;
```

