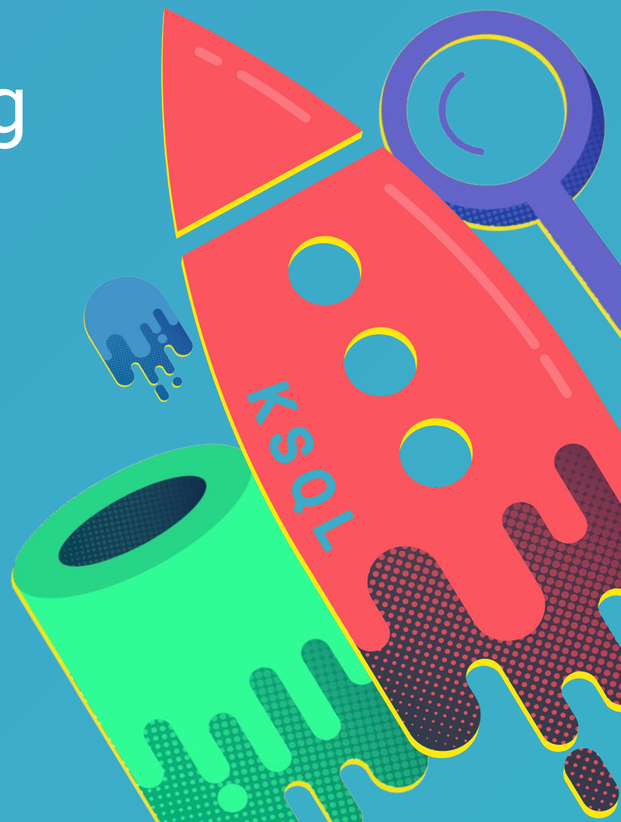
confluent

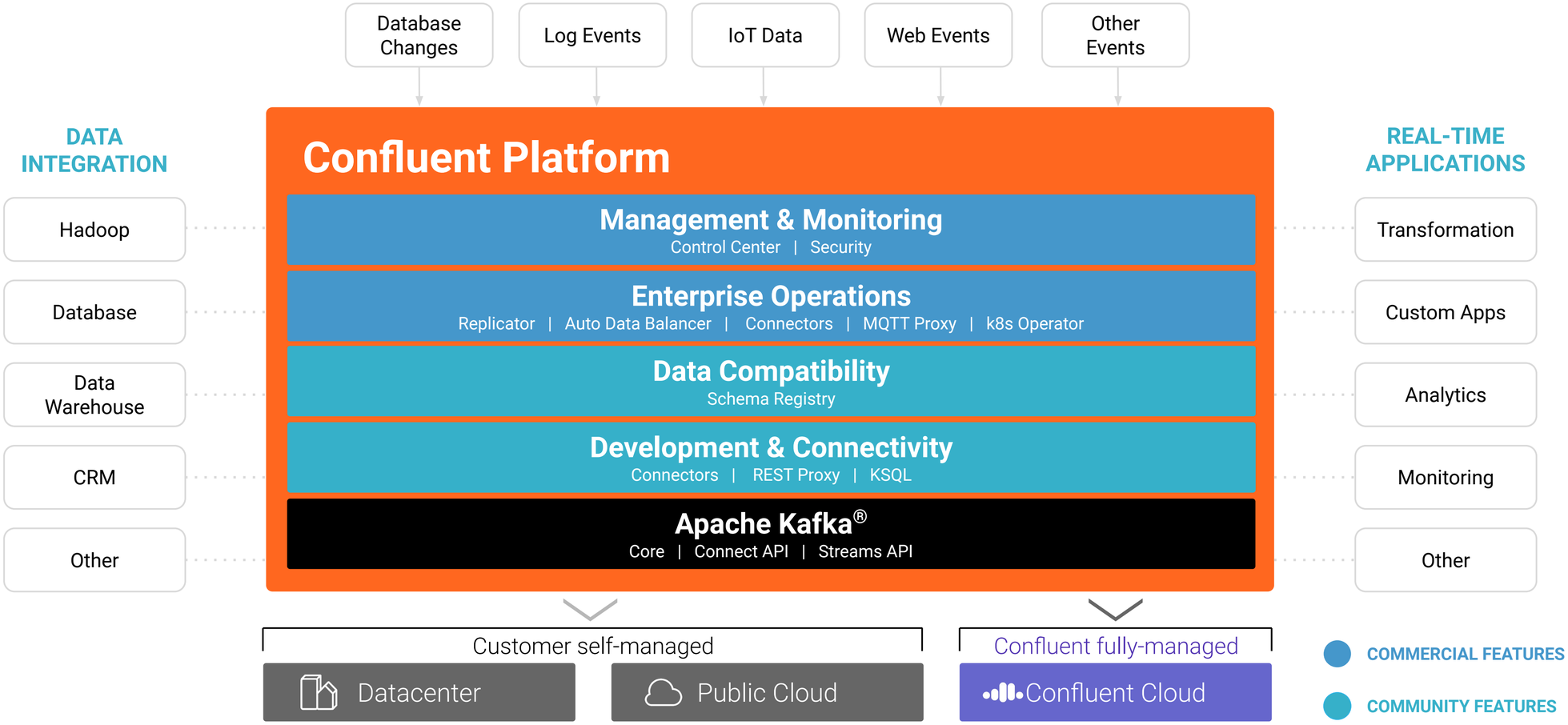
Technical Workshop

KSQL showcases Apache Kafka® stream processing using KSQL

https://github.com/confluentinc/demo-scene/tree/master/ksql-workshop



Complete Set of Development, Operations and Management Capabilities to run Kafka at Scale

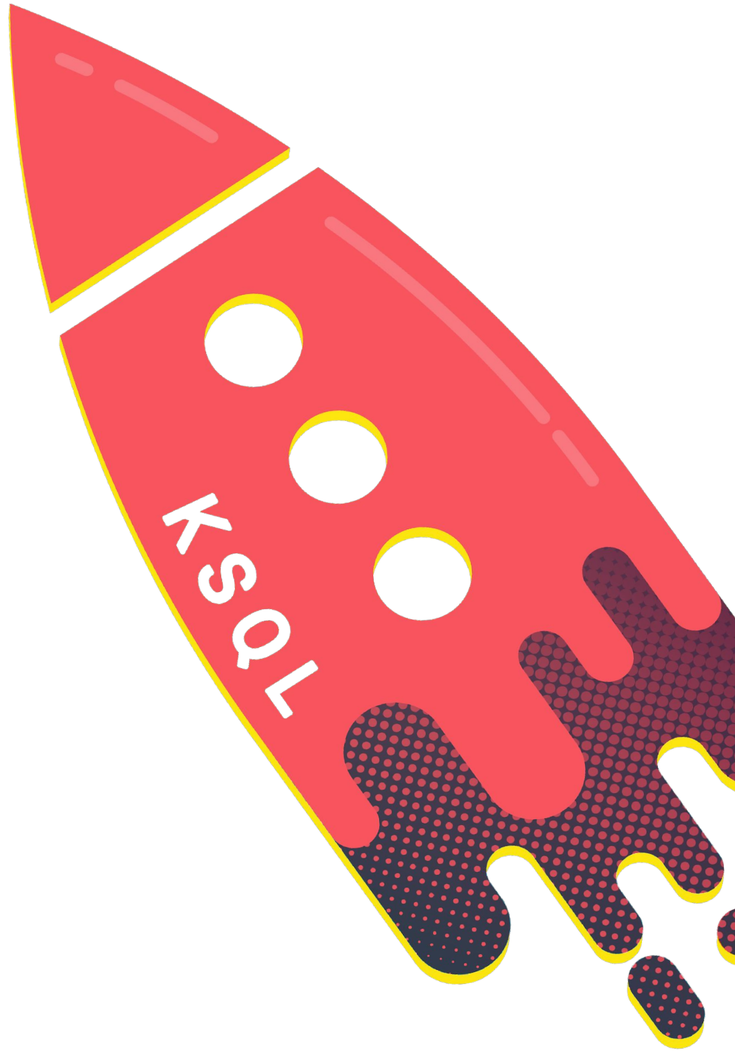


KSQL

is a

Declarative

Stream Processing Language



KSQL

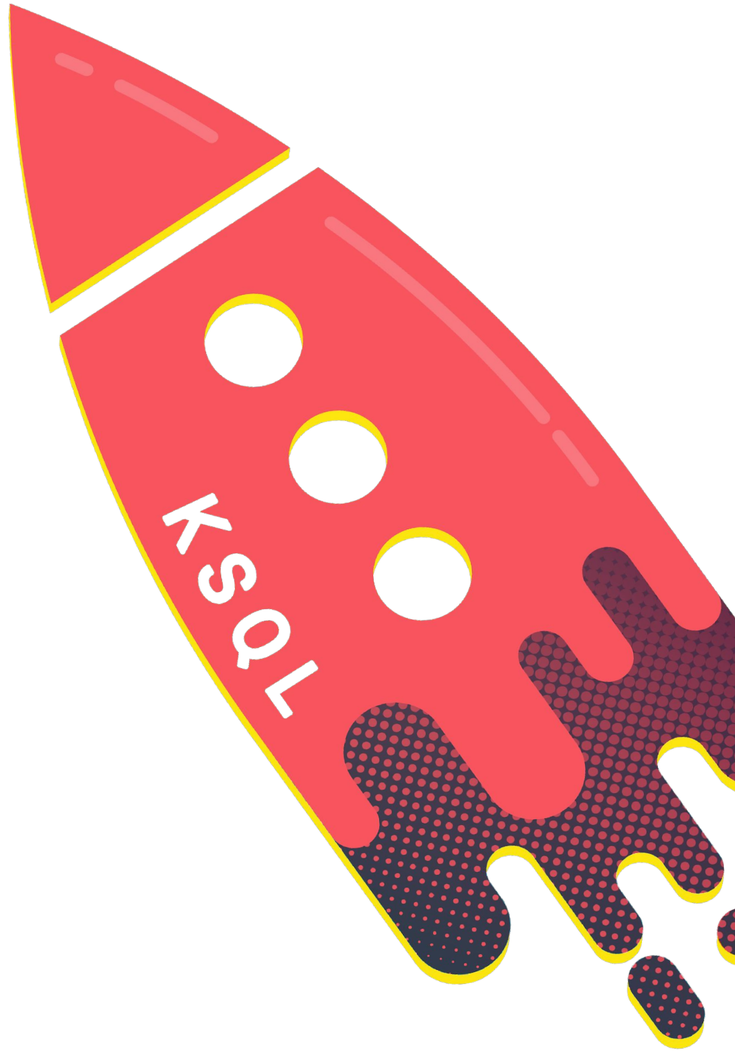
is the

Streaming

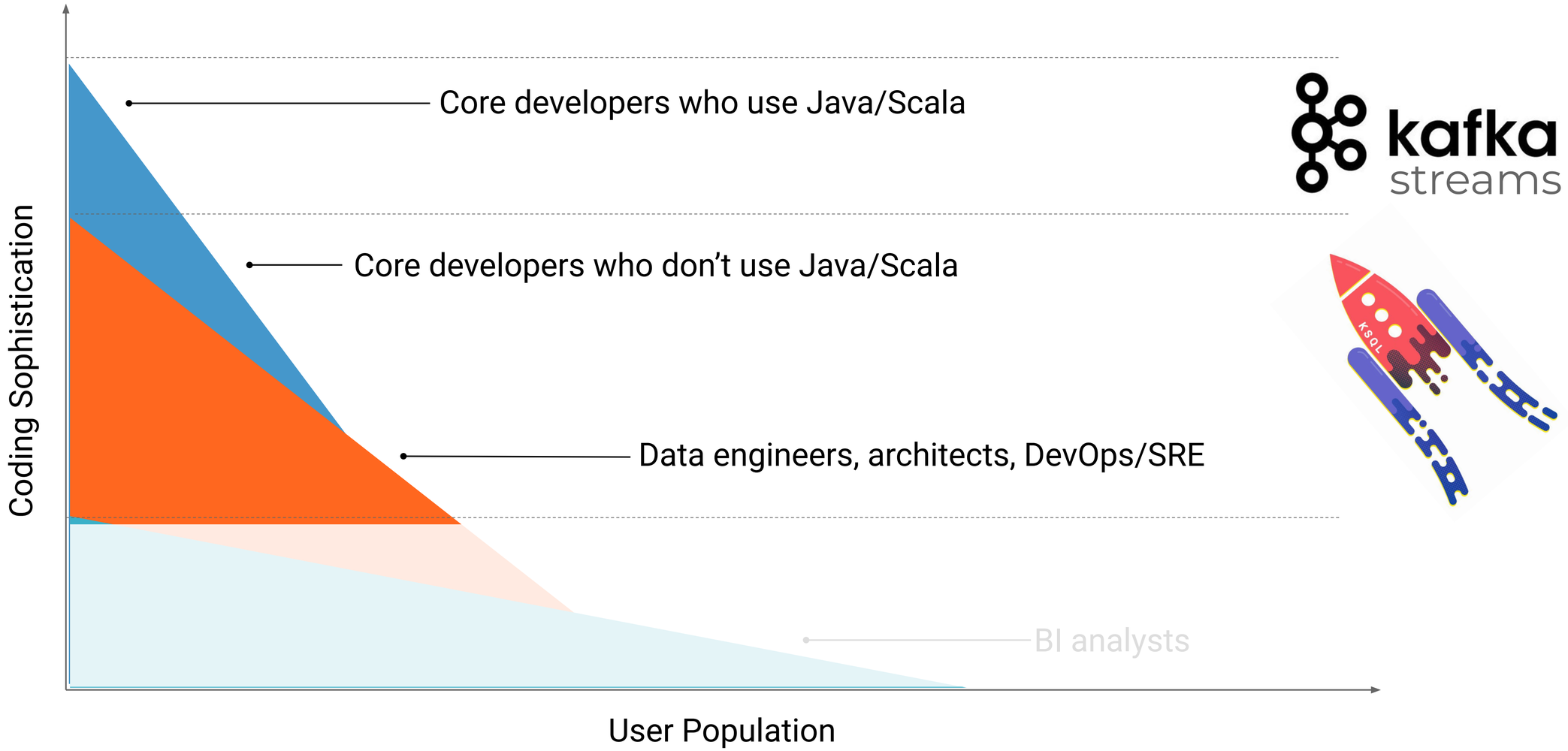
SQL Engine

for

Apache Kafka



Lower the bar to enter the world of streaming



Event Transformation with Stream Processing

Lower the bar to enter the world of streaming

kafka streams

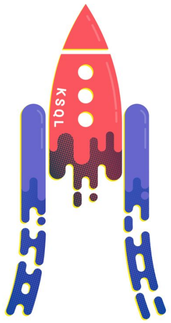
Apache Kafka® library to write real-time applications and microservices in Java and Scala

```
object FraudFilteringApplication extends App {

  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092,kafka-broker2:9092")

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

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



Confluent KSQL

The streaming SQL engine for Apache Kafka® to write real-time applications in SQL

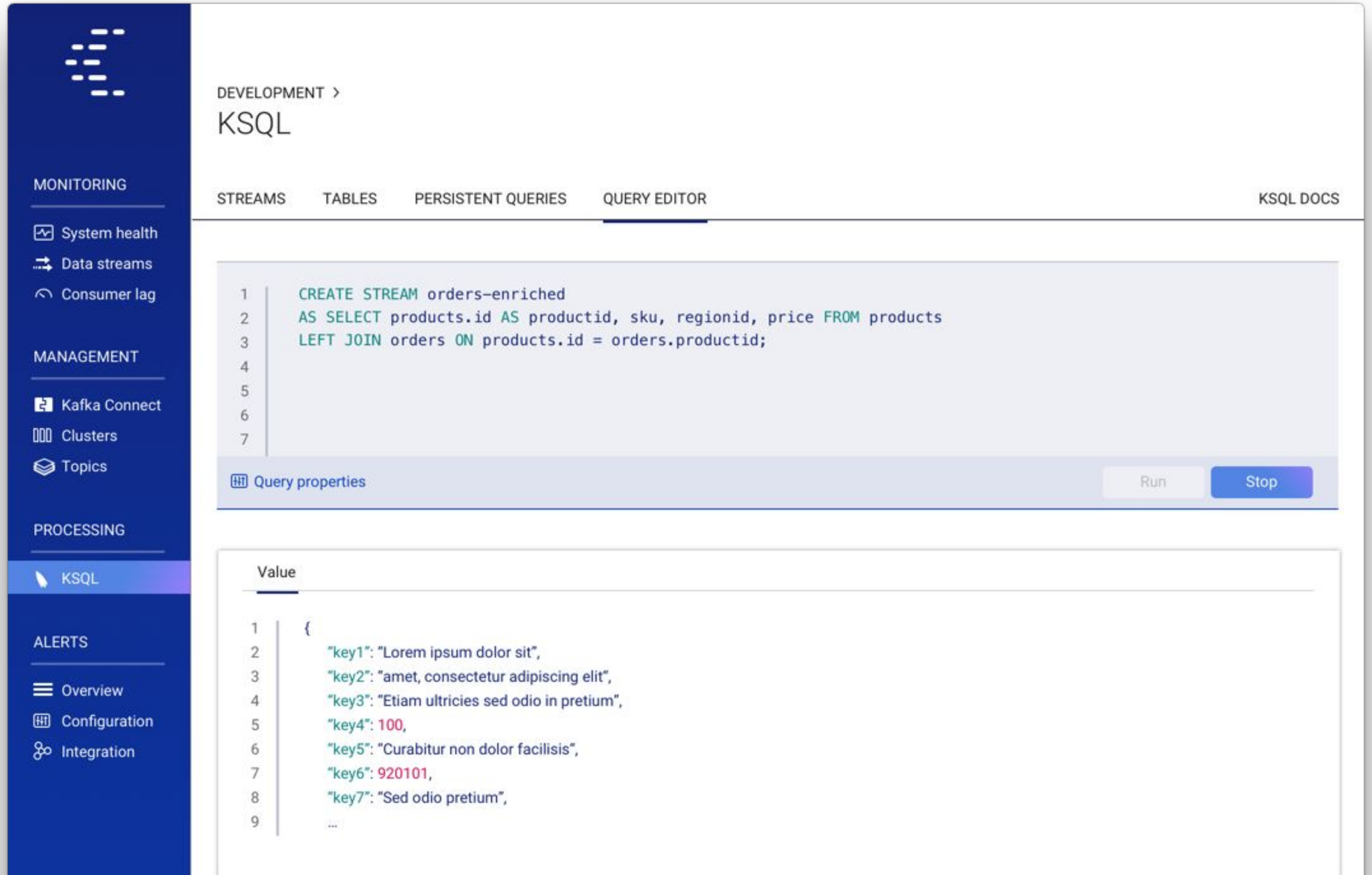
```
CREATE STREAM fraudulent_payments AS
 SELECT * FROM payments
 WHERE fraudProbability > 0.8;
```

You write ***only*** SQL. No Java, Python, or other boilerplate to wrap around it!

But you can create KSQL User Defined Functions in Java

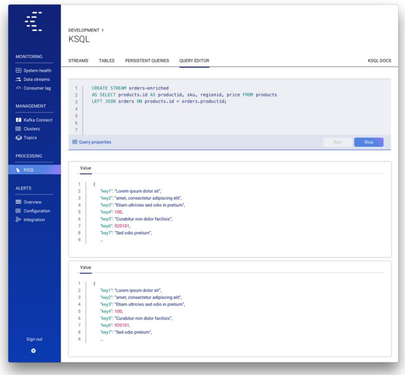
confluent.io/product/ksql

New user experience: interactive stream processing



KSQL can be used interactively + programmatically

1 UI

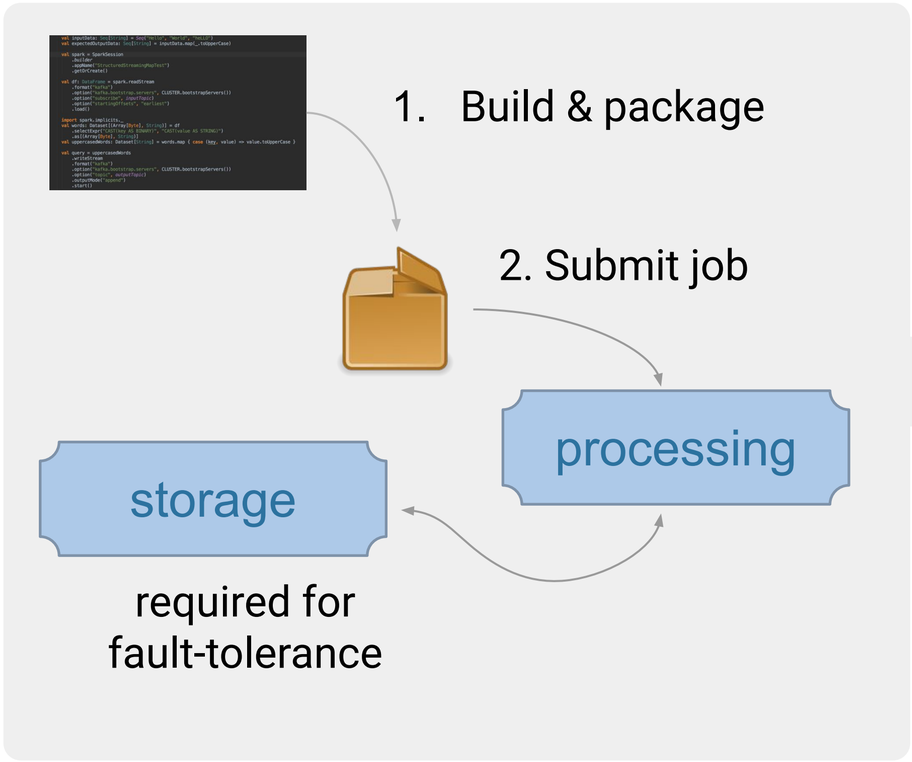


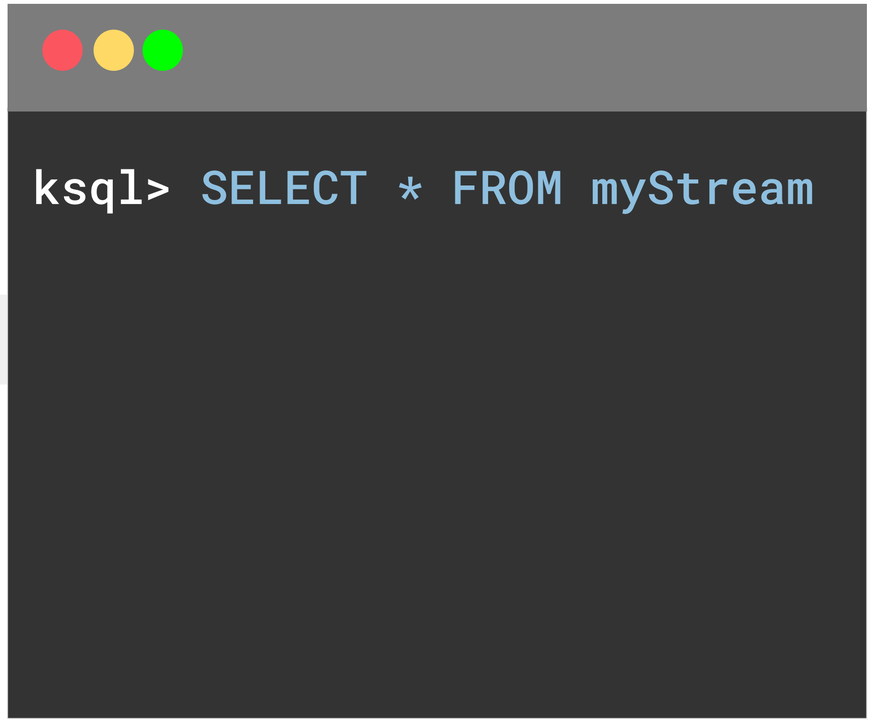
2 CLI

3 REST

4 Headless

All you need is Kafka and KSQL

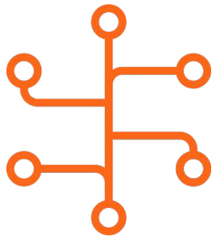




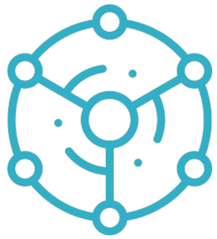
Without KSQL

With KSQL

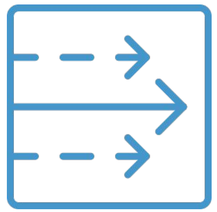
KSQL example use cases



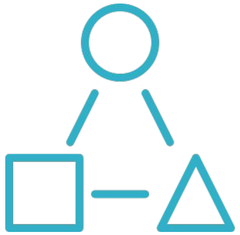
Data exploration



Data enrichment



Streaming ETL

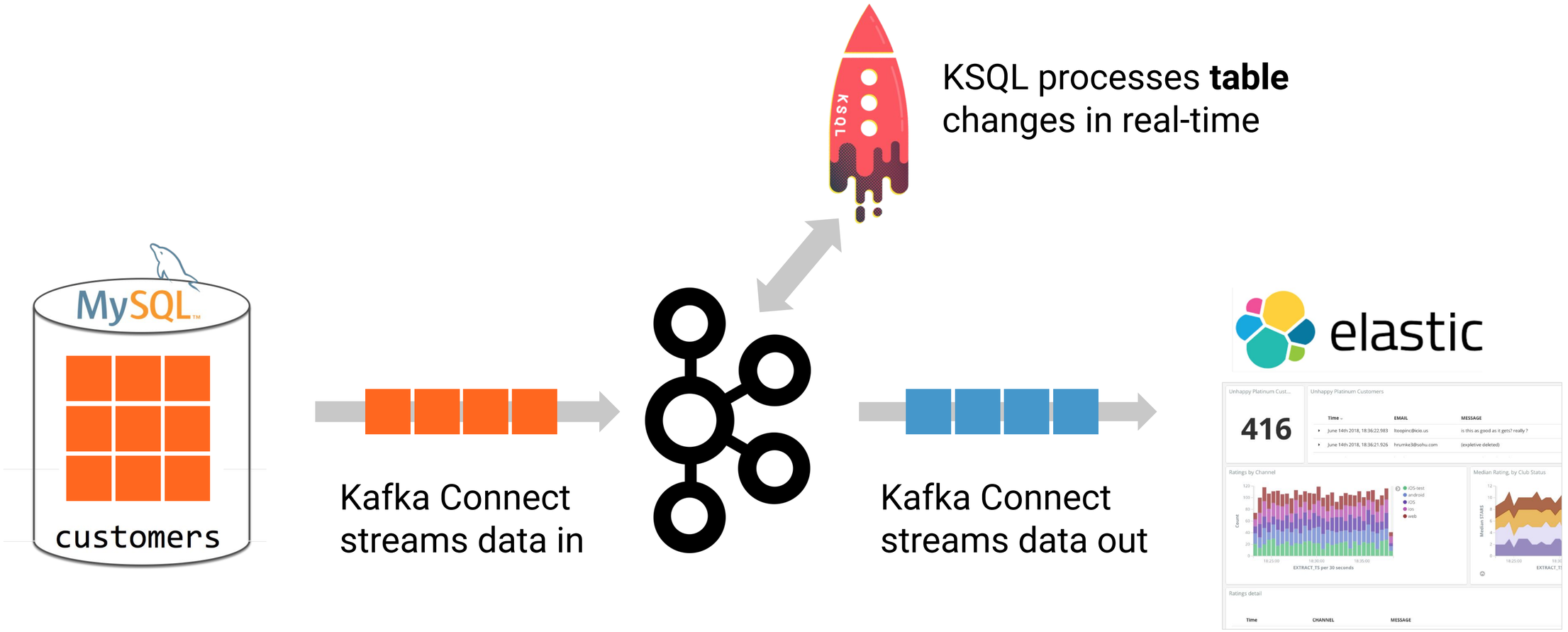


Filter, cleanse, mask

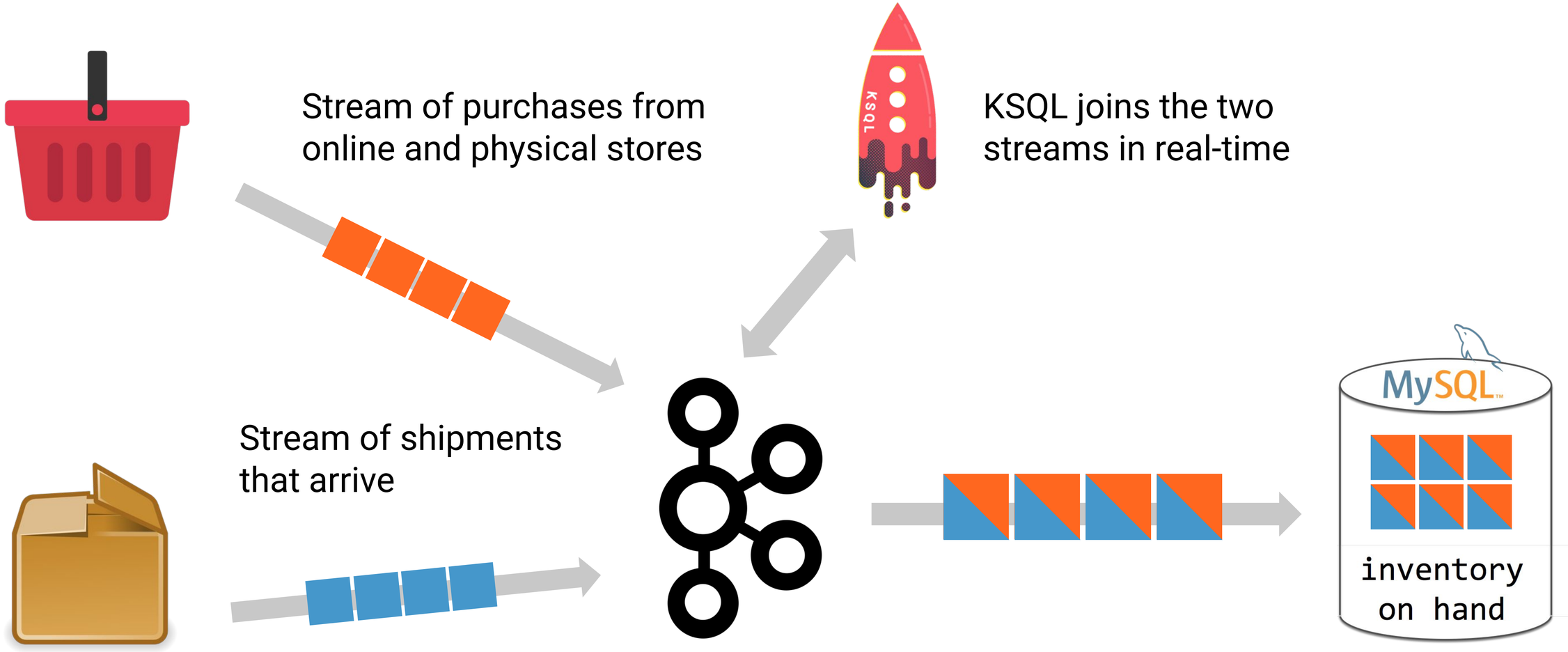
Real-time monitoring

Anomaly detection

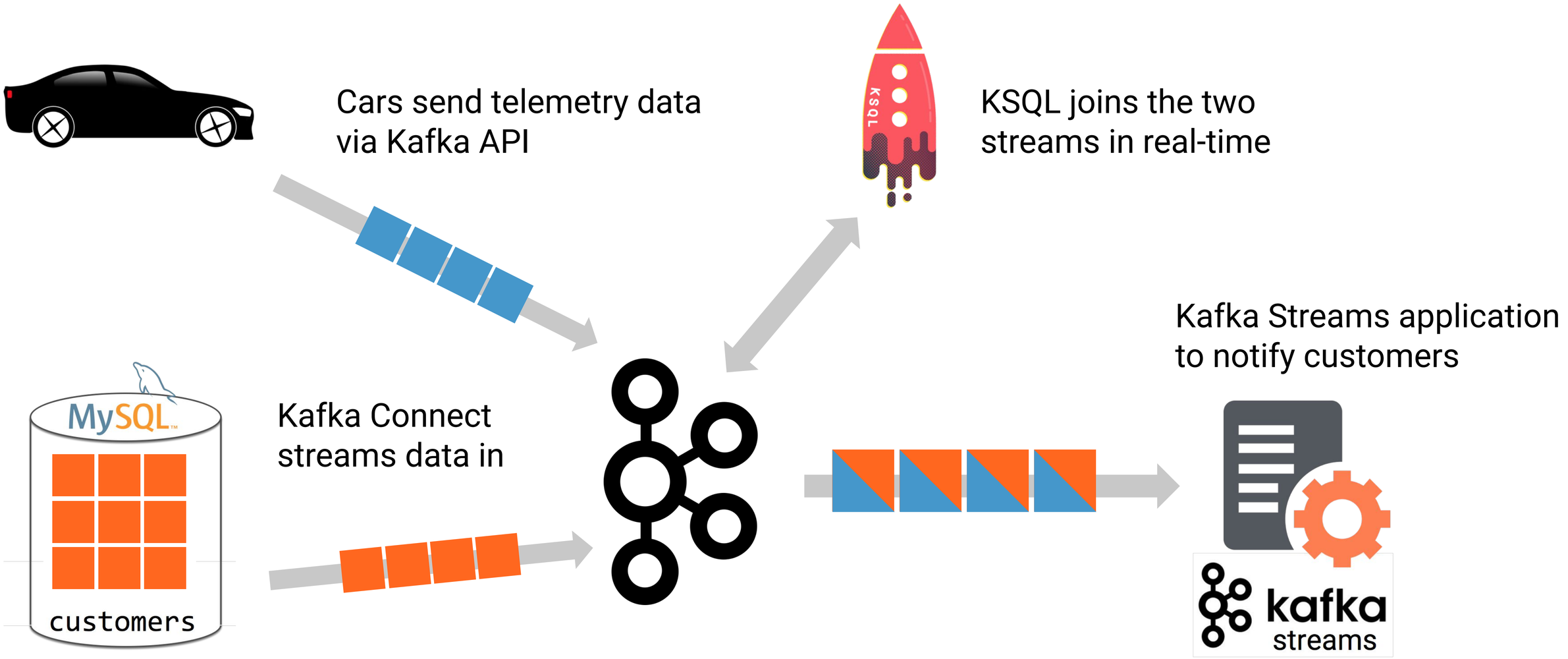
Example: CDC from DB via Kafka to Elastic



Example: Retail



Example: IoT, Automotive, Connected Cars



KSQL for Real-Time Monitoring

- Log data monitoring
- Tracking and alerting
- Syslog data
- Sensor / IoT data
- Application metrics

```
CREATE STREAM syslog_invalid_users AS
  SELECT host, message
  FROM syslog
  WHERE message LIKE '%Invalid user%';
```

http://cnfl.io/syslogs-filtering / http://cnfl.io/syslog-alerting

KSQL for Anomaly Detection

- Identify patterns or anomalies in real-time data, surfaced in milliseconds

```
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 for Streaming ETL

- Joining, filtering, and aggregating streams of event data

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

KSQL for Data Transformation

- Easily make derivations of existing topics

```
CREATE STREAM pageviews_avro
  WITH (PARTITIONS=6,
        VALUE_FORMAT='AVRO') AS
  SELECT * FROM pageviews_json
  PARTITION BY user_id;
```

confluent

How to run KSQL

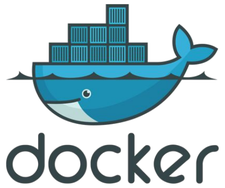
How to run KSQL

KSQL Server
(JVM process)

DEB, RPM, ZIP, TAR downloads
http://confluent.io/ksql

Docker images
confluentinc/cp-ksql-server
confluentinc/cp-ksql-cli

Physical



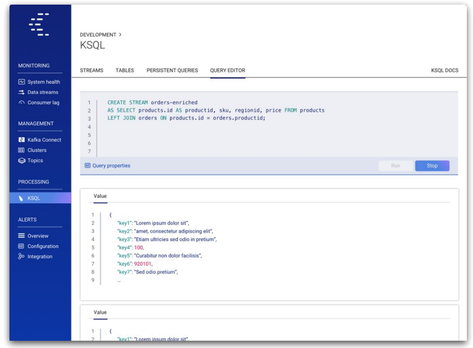
openstack.

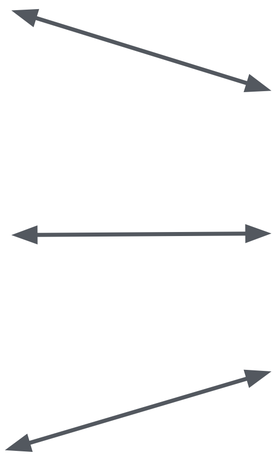
Google Cloud Platform

…and many more…

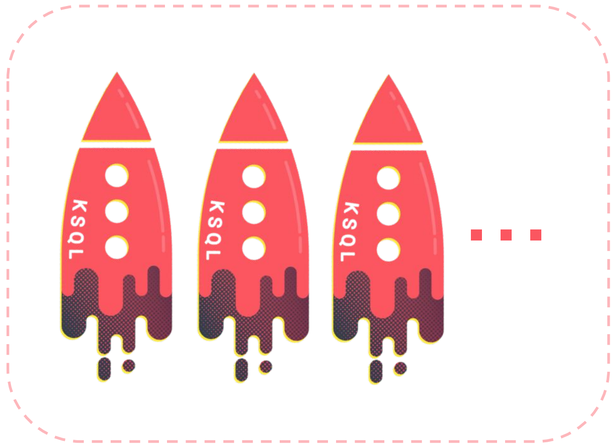
How to run KSQL

#1 Interactive KSQL, for development & testing

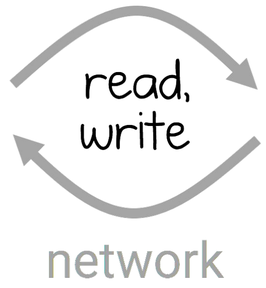




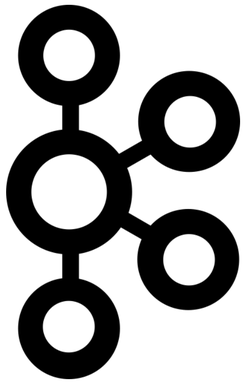
KSQL Cluster
(processing)



KSQL does not run
on Kafka brokers!

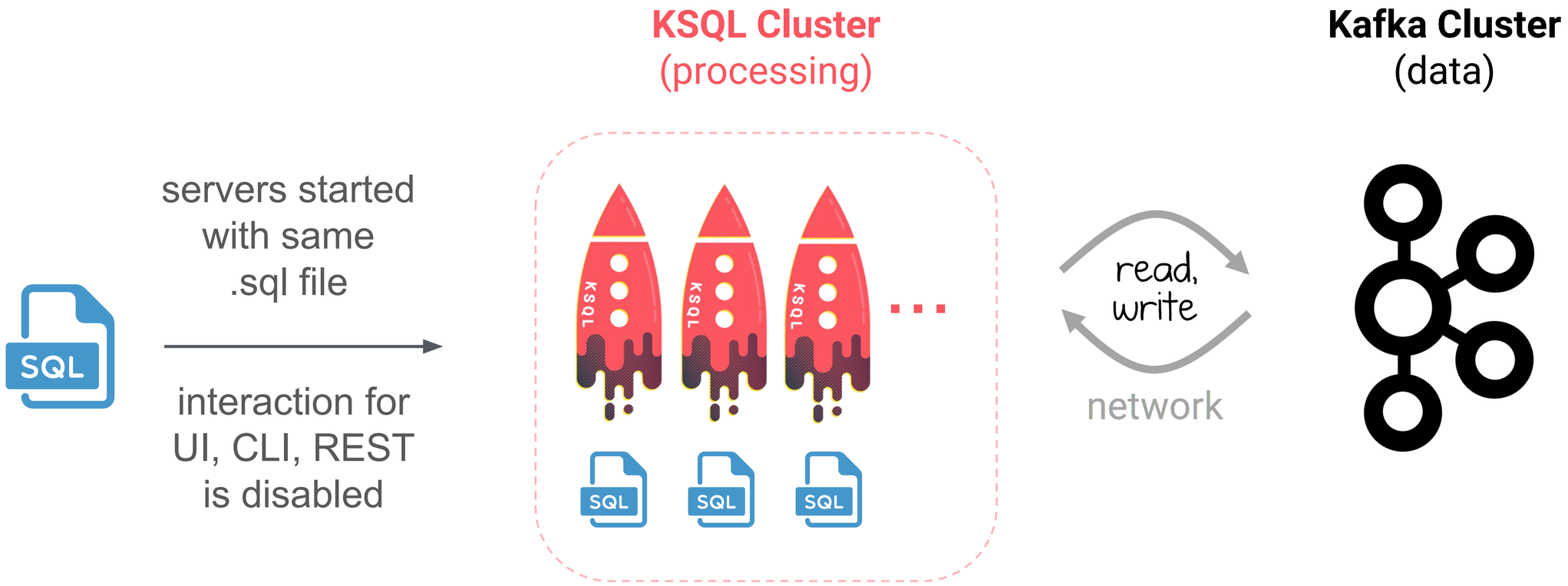


Kafka Cluster
(data)



How to run KSQL

#2 Headless KSQL, for production

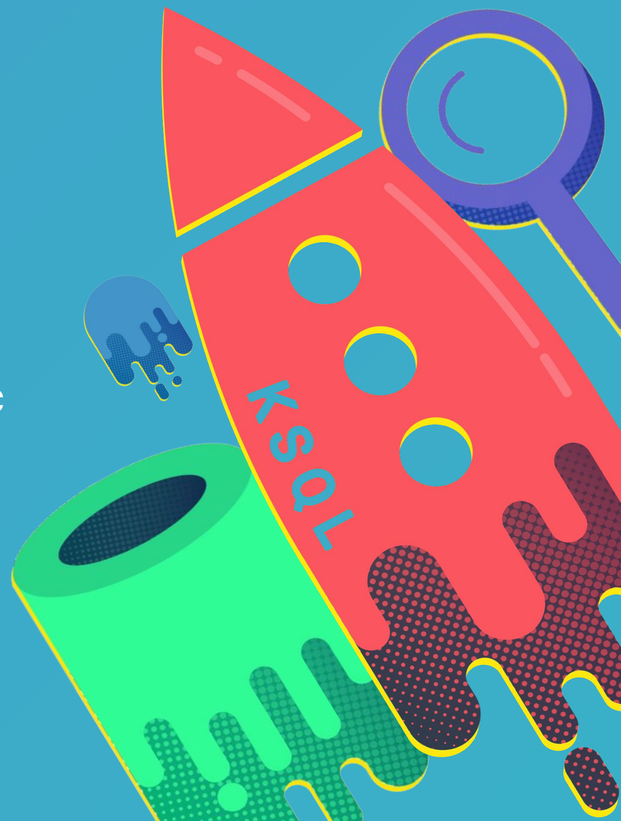


confluent

Ready for the Workshop !?

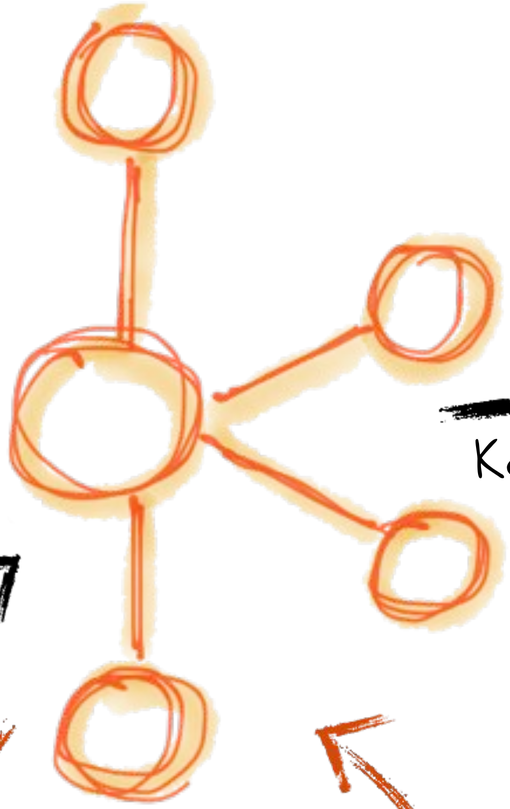
https://github.com/confluentinc/demo-scene/tree/master/ksql-workshop

Pre-requisites for KSQL Workshop :
https://github.com/confluentinc/demo-scene/blob/master/ksql-workshop/pre-requisites.adoc

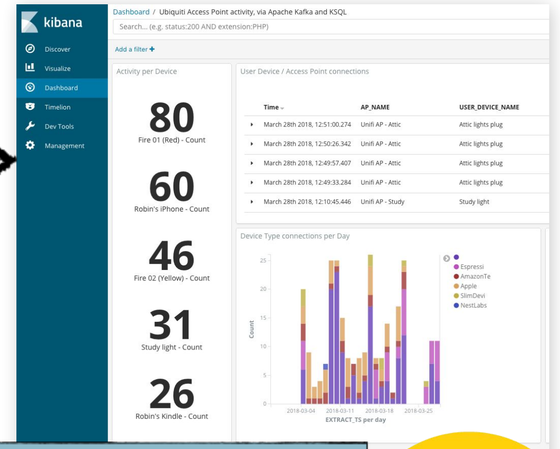


```
{
 "rating_id": 5313,
 "user_id": 3,
 "stars": 4,
 "route_id": 6975,
 "rating_time": 1519304105213,
 "channel": "web",
 "message": "worst. flight. ever. #neveragain"
}
```


Producer API

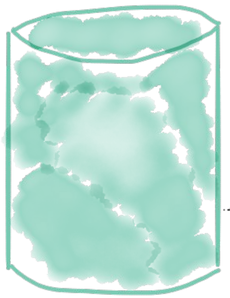


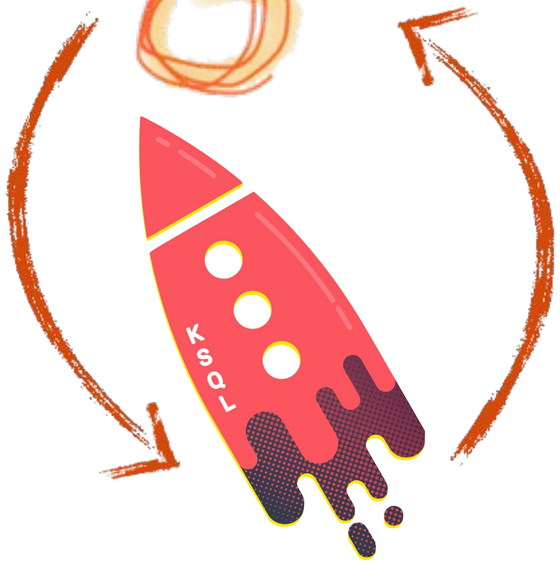
Kafka Connect



Elasticsearch

Kafka Connect






```
{
  "id": 3,
  "first_name": "Merilyn",
  "last_name": "Doughartie",
  "email": "mdoughartie1@dedecms.com",
  "gender": "Female",
  "club_status": "platinum",
  "comments": "none"
}
```

Setup Steps – Sections 1 to 2

Make sure your docker-compose is up and running

```
macbookpro:~/demo-scene/ksql-workshop$ docker-compose up -d
Creating network "ksql-workshop_default" with the default driver
Creating ksql-workshop_zookeeper_1     ... done
Creating ksql-workshop_mysql_1         ... done
Creating ksql-workshop_elasticsearch_1 ... done
Creating ksql-workshop_kafka_1         ... done
Creating ksql-workshop_kibana_1          ... done
Creating ksql-workshop_schema-registry_1 ... done
Creating ksql-workshop_kafkacat_1          ... done
Creating ksql-workshop_ksql-server_1       ... done
Creating ksql-workshop_kafka-connect-cp_1 ... done
Creating ksql-workshop_datagen-ratings_1   ... done
Creating ksql-workshop_connect-debezium_1 ... done
Creating ksql-workshop_ksql-cli_1          ... done

macbookpro:~/demo-scene/ksql-workshop$ docker-compose ps
Name                                Command                                State      Ports
-------------------------------------------------------------------------------------------------------------------------------
ksql-workshop_connect-debezium_1   /docker-entrypoint.sh start      Up      0.0.0.0:8083->8083/tcp, 8778/tcp, 9092/tcp, 9779/tcp
ksql-workshop_datagen-ratings_1    bash -c echo Waiting for K ...   Up
ksql-workshop_elasticsearch_1      /usr/local/bin/docker-entr ...   Up      0.0.0.0:9200->9200/tcp, 9300/tcp
ksql-workshop_kafka-connect-cp_1   /etc/confluent/docker/run        Up      0.0.0.0:18083->18083/tcp, 8083/tcp, 9092/tcp
ksql-workshop_kafka_1              /etc/confluent/docker/run        Up      0.0.0.0:9092->9092/tcp
ksql-workshop_kafkacat_1           sleep infinity                   Up
ksql-workshop_kibana_1             /usr/local/bin/kibana-docker     Up      0.0.0.0:5601->5601/tcp
ksql-workshop_ksql-cli_1           /bin/sh                          Up
ksql-workshop_ksql-server_1        /etc/confluent/docker/run        Up      8088/tcp
ksql-workshop_mysql_1              docker-entrypoint.sh mysqld      Up      3306/tcp, 33060/tcp
ksql-workshop_schema-registry_1    /etc/confluent/docker/run        Up      8081/tcp
ksql-workshop_zookeeper_1          /etc/confluent/docker/run        Up      2181/tcp, 2888/tcp, 3888/tcp
```

Let's Start – Preliminary – Sections 3 to 5

- Start up the full set of container : `macbookpro:~/demo-scene/ksql-workshop$ docker-compose up -d`
  Check docker containers status : `macbookpro:~/demo-scene/ksql-workshop$ docker-compose ps`
- Inspect data in ratings topic with KSQL CLI and kafkacat (https://github.com/edenhill/kafkacat)
  KSQL CLI with :

```
macbookpro:~/demo-scene/ksql-workshop$ docker-compose exec ksql-cli ksql
http://ksql-server:8088
ksql> SHOW TOPICS;
ksql> PRINT 'ratings';
```

- Create a Stream [ratings] from ratings topic

```
ksql> CREATE STREAM ratings WITH (KAFKA_TOPIC='ratings', VALUE_FORMAT='AVRO');
```

- Inspect, query and filter Stream [ratings]

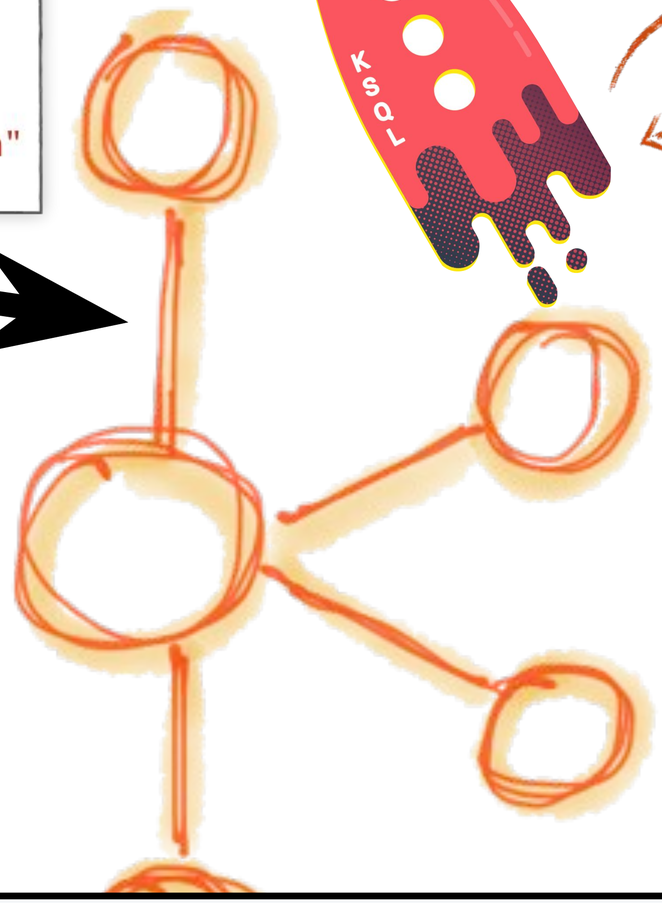
```
ksql> DESCRIBE ratings;
ksql> SELECT * FROM ratings LIMIT 5;
ksql> SELECT USER_ID, STARS, CHANNEL, MESSAGE FROM ratings WHERE STARS <3 AND CHANNEL='iOS'
LIMIT 3;
```

- KSQL Offsets

```
ksql> SET 'auto.offset.reset' = 'earliest';
```

```
{
 "rating_id": 5313,
 "user_id": 3,
 "stars": 4,
 "route_id": 6975,
 "rating_time": 1519304105213,
 "channel": "web",
 "message": "worst. flight. ever. #neveragain"
}
```


Producer API



Filter all ratings where STARS<3

POOR_RATINGS

```
CREATE STREAM POOR_RATINGS AS

SELECT * FROM ratings WHERE STARS <3
```

Let's go w/ KSQL– Sections 6 to 7

- Create a new Stream [POOR_RATINGS] as a filter of initial Stream [ratings]

```
ksql> CREATE STREAM POOR_RATINGS AS SELECT * FROM ratings WHERE STARS <3 AND CHANNEL='iOS';
```

- Inspect & Query Stream [POOR_RATINGS]

`ksql> DESCRIBE POOR_RATINGS;` Additional info with : `ksql> DESCRIBE EXTENDED POOR_RATINGS;`

Query the stream : `ksql> SELECT STARS, CHANNEL, MESSAGE FROM POOR_RATINGS;`

- Open up stream of customers data update from DB instantiating 2 CDC connectors with :

```
macbookpro:~/demo-scene/ksql-workshop$ docker-compose exec connect-debezium bash -c '/scripts/create-mysql-source.sh'
```

  - 2 topics created and fed : asgard.demo.CUSTOMERS & asgard.demo.CUSTOMERS-raw
  - Check results of DB operations into mysql-source-demo-customers-raw topic

    Start My SQL command prompt :

    ```
    macbookpro:~/demo-scene/ksql-workshop$ docker-compose exec mysql bash -c 'mysql -u $MYSQL_USER -p$MYSQL_PASSWORD demo'
    ```

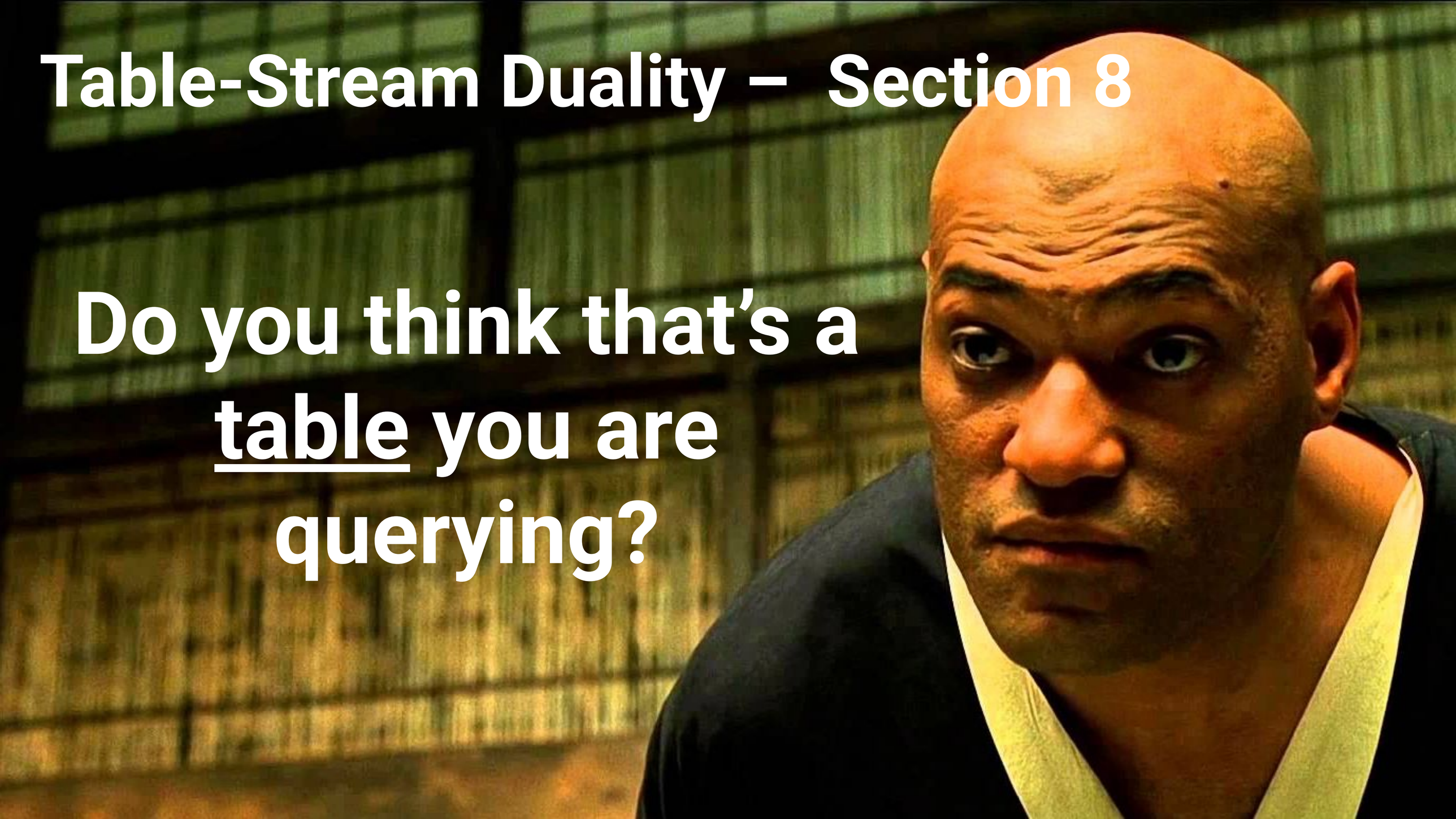
    Insert data in MySQL : `mysql> INSERT INTO CUSTOMERS (ID,FIRST_NAME,LAST_NAME) VALUES (42,'Rick','Astley');`

    Update date in MySQL : `mysql> UPDATE CUSTOMERS SET FIRST_NAME = 'Thomas', LAST_NAME ='Smith' WHERE ID=2;`
  - Inspect data in mysql-source-demo-customers topic with KSQL or kafkacat

    ```
    ksql> PRINT 'asgard.demo.CUSTOMERS' FROM BEGINNING;
    ```
  - Re-Key Topics with KSQL to join the customer data to the ratings

    ```
    ksql> CREATE STREAM CUSTOMERS_SRC WITH (KAFKA_TOPIC='asgard.demo.CUSTOMERS', VALUE_FORMAT='AVRO');
    ksql> SET 'auto.offset.reset' = 'earliest';
    ksql> CREATE STREAM CUSTOMERS_SRC_REKEY WITH (PARTITIONS=1) AS SELECT * FROM CUSTOMERS_SRC PARTITION BY ID;
    ```

Table-Stream Duality – Section 8



The Table Stream Duality – Section 8

Stream

Account ID	Amount
12345	+ €50
12345	+ €25
12345	-€60

Time

Table

Account ID	Balance
12345	€50

Account ID	Balance
12345	€75

Account ID	Balance
12345	€15

The Stream/Table Duality – Section 8

- Now create table [CUSTOMERS] from [CUSTOMERS_SRC_REY] with the right key

```
ksql> CREATE TABLE CUSTOMERS WITH (KAFKA_TOPIC='CUSTOMERS_SRC_REKEY', VALUE_FORMAT ='AVRO', KEY='ID');
```

- Query the table [CUSTOMERS]

```
ksql> SELECT ID, FIRST_NAME, LAST_NAME, EMAIL, CLUB_STATUS FROM CUSTOMERS LIMIT 3;
```

- Both table [CUSTOMERS] and stream [CUSTOMER_SRC_REKEY] are driven from the same Kafka topic, you can check duality with 2 KSQL Sessions :

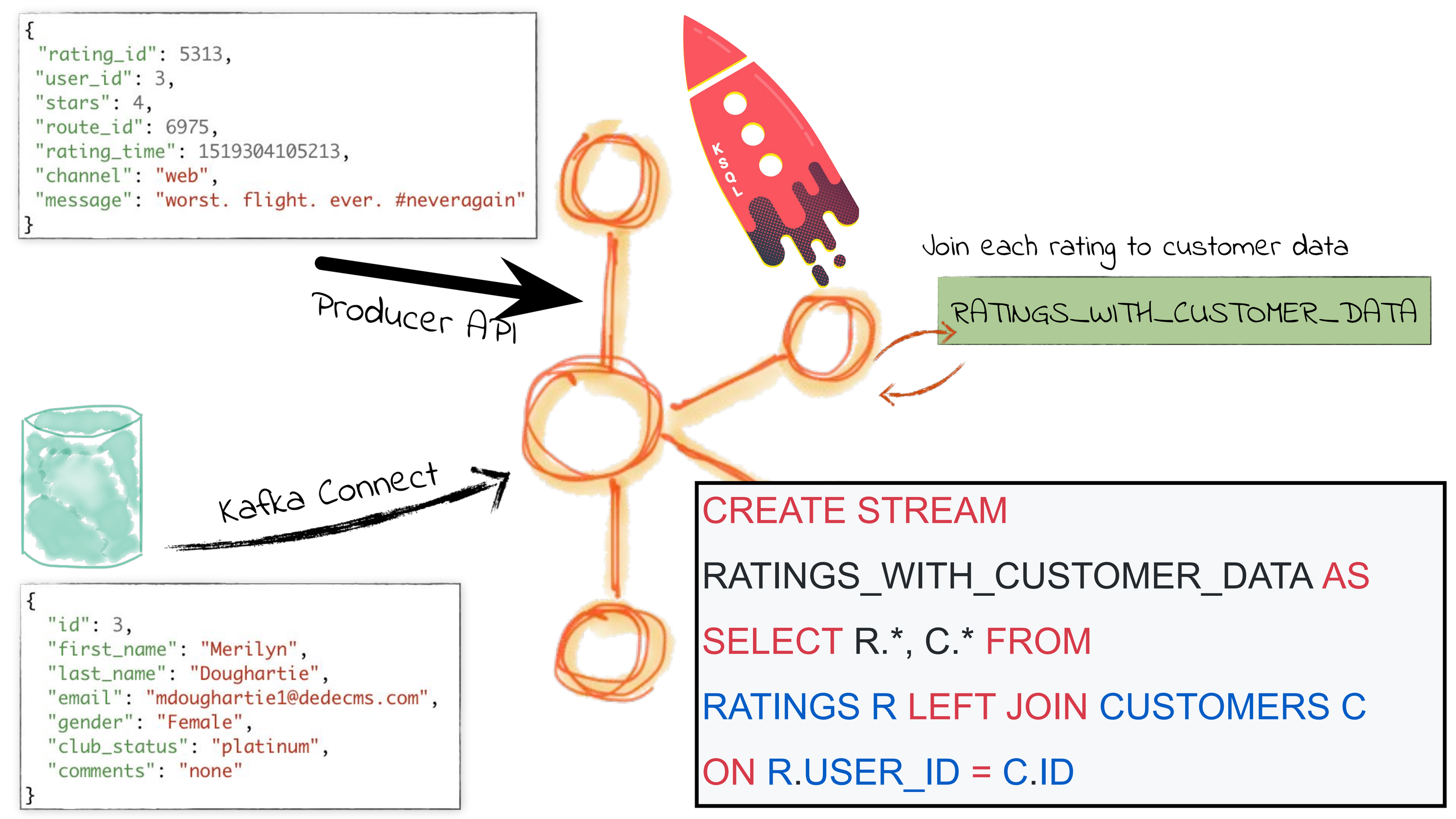
```
macbookpro:~/demo-scene/ksql-workshop$ docker-compose exec ksql-cli ksql http://ksql-server:8088
```

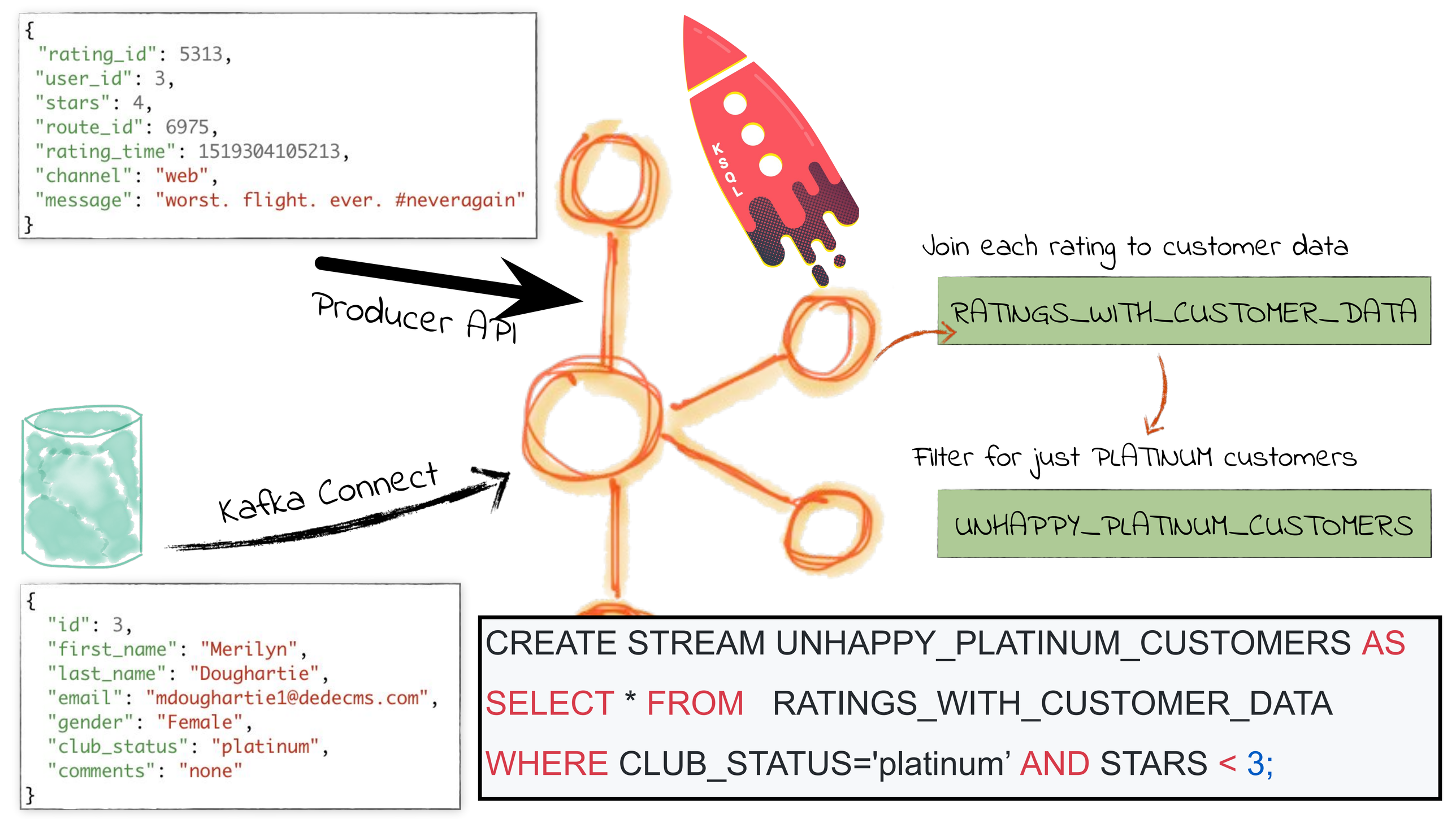
1.
```
ksql> SET 'auto.offset.reset' = 'earliest';
ksql> SELECT ID, FIRST_NAME, LAST_NAME, EMAIL, CLUB_STATUS FROM CUSTOMERS WHERE ID=2;
```

2.
```
ksql> SET 'auto.offset.reset' = 'earliest';
ksql> SELECT ID, FIRST_NAME, LAST_NAME, EMAIL, CLUB_STATUS FROM CUSTOMERS_SRC_REKEY WHERE ID=2;
```

3. Make an update in MySQL in a third session to compare stream and table behavior :

```
macbookpro:~/demo-scene/ksql-workshop$ docker-compose exec mysql bash -c 'mysql -u $MYSQL_USER -p$MYSQL_PASSWORD demo'
mysql> UPDATE CUSTOMERS SET EMAIL='foo@bar.com' WHERE ID=2;
mysql> UPDATE CUSTOMERS SET EMAIL='example@bork.bork.bork.com' WHERE ID=2;
```


{
"rating_id": 5313,
"user_id": 3,
"stars": 4,
"route_id": 6975,
"rating_time": 1519304105213,
"channel": "web",
"message": "worst. flight. ever. #neveragain"
}
Producer API
KSQL
Join each rating to customer data
RATINGS_WITH_CUSTOMER_DATA
Kafka Connect
{
"id": 3,
"first_name": "Merilyn",
"last_name": "Doughartie",
"email": "mdoughartie1@dedecms.com",
"gender": "Female",
"club_status": "platinum",
"comments": "none"
}
CREATE STREAM
RATINGS_WITH_CUSTOMER_DATA AS
SELECT R.*, C.* FROM
RATINGS R LEFT JOIN CUSTOMERS C
ON R.USER_ID = C.ID


{
"rating_id": 5313,
"user_id": 3,
"stars": 4,
"route_id": 6975,
"rating_time": 1519304105213,
"channel": "web",
"message": "worst. flight. ever. #neveragain"
}
Producer API
KSQL
Join each rating to customer data
RATINGS_WITH_CUSTOMER_DATA
Filter for just PLATINUM customers
UNHAPPY_PLATINUM_CUSTOMERS
Kafka Connect
{
"id": 3,
"first_name": "Merilyn",
"last_name": "Doughartie",
"email": "mdoughartie1@dedecms.com",
"gender": "Female",
"club_status": "platinum",
"comments": "none"
}
CREATE STREAM UNHAPPY_PLATINUM_CUSTOMERS AS
SELECT * FROM RATINGS_WITH_CUSTOMER_DATA
WHERE CLUB_STATUS='platinum' AND STARS < 3;

Join Data in KSQL – Section 9

- Now we can join table [CUSTOMERS] with stream [RATINGS]

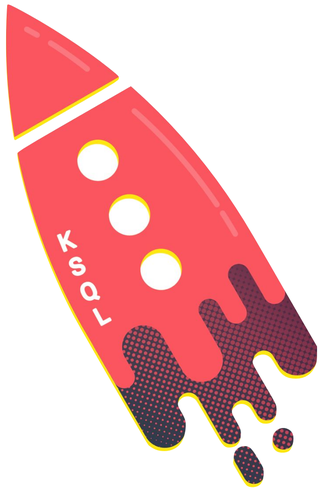
```
ksql> CREATE STREAM RATINGS_WITH_CUSTOMER_DATA WITH (PARTITIONS=1) AS \
SELECT R.RATING_ID, R.CHANNEL, R.STARS, R.MESSAGE, \
    C.ID, C.CLUB_STATUS, C.EMAIL, \
    C.FIRST_NAME, C.LAST_NAME \
FROM RATINGS R \
    INNER JOIN CUSTOMERS C \
      ON R.USER_ID = C.ID;
```

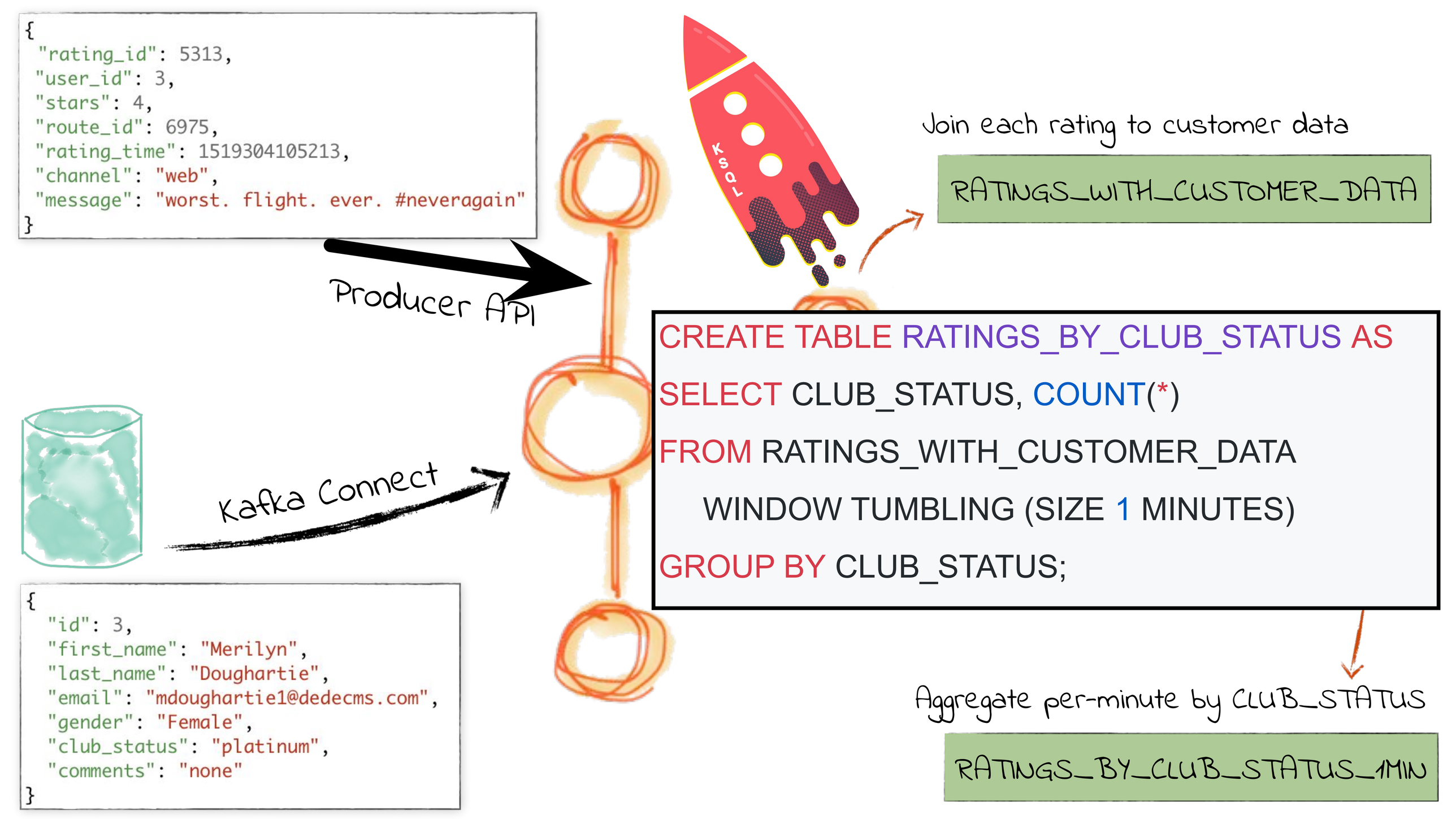
- Create a new stream of poor rating filtering on customer's status

```
ksql> CREATE STREAM UNHAPPY_PLATINUM_CUSTOMERS AS \
SELECT CLUB_STATUS, EMAIL, STARS, MESSAGE \
FROM RATINGS_WITH_CUSTOMER_DATA \
WHERE STARS < 3 \
 AND CLUB_STATUS = 'platinum';
```

- Query the new stream of unhappy customers

```
ksql> SELECT STARS, MESSAGE, EMAIL FROM UNHAPPY_PLATINUM_CUSTOMERS;
```




{
"rating_id": 5313,
"user_id": 3,
"stars": 4,
"route_id": 6975,
"rating_time": 1519304105213,
"channel": "web",
"message": "worst. flight. ever. #neveragain"
}
Producer API
KSQL
Join each rating to customer data
RATINGS_WITH_CUSTOMER_DATA
CREATE TABLE RATINGS_BY_CLUB_STATUS AS
SELECT CLUB_STATUS, COUNT(*)
FROM RATINGS_WITH_CUSTOMER_DATA
WINDOW TUMBLING (SIZE 1 MINUTES)
GROUP BY CLUB_STATUS;
Kafka Connect
{
"id": 3,
"first_name": "Merilyn",
"last_name": "Doughartie",
"email": "mdoughartie1@dedecms.com",
"gender": "Female",
"club_status": "platinum",
"comments": "none"
}
Aggregate per-minute by CLUB_STATUS
RATINGS_BY_CLUB_STATUS_1MIN

Aggregating with KSQL – Sections 10

- Count ratings by customer status each 1 minute

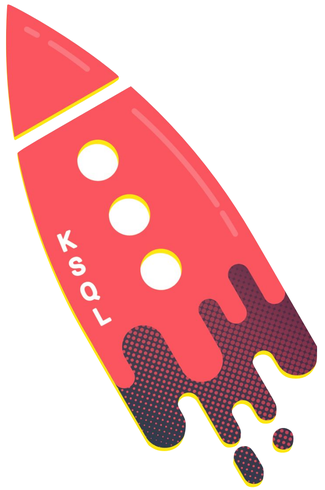
```
ksql> SELECT TIMESTAMPTOSTRING(WindowStart(), 'yyyy-MM-dd HH:mm:ss'), \
 CLUB_STATUS, COUNT(*) AS RATING_COUNT \
FROM RATINGS_WITH_CUSTOMER_DATA \
 WINDOW TUMBLING (SIZE 1 MINUTES) \
GROUP BY CLUB_STATUS;
```

- Create a table to store results of counting aggregate

```
ksql> CREATE TABLE RATINGS_BY_CLUB_STATUS AS \
SELECT WindowStart() AS WINDOW_START_TS, CLUB_STATUS, COUNT(*) AS RATING_COUNT \
FROM RATINGS_WITH_CUSTOMER_DATA \
     WINDOW TUMBLING (SIZE 1 MINUTES) \
GROUP BY CLUB_STATUS;
```

- We can now query and filter this table

```
ksql> SELECT TIMESTAMPTOSTRING(WINDOW_START_TS, 'yyyy-MM-dd HH:mm:ss'), \
 CLUB_STATUS, RATING_COUNT \
FROM RATINGS_BY_CLUB_STATUS \
WHERE CLUB_STATUS='bronze';
```



Optional : Stream data to Elasticsearch – Section 11

Configure Kafka Connect to stream data to Elasticsearch

Assumes that you know how to use Kibana