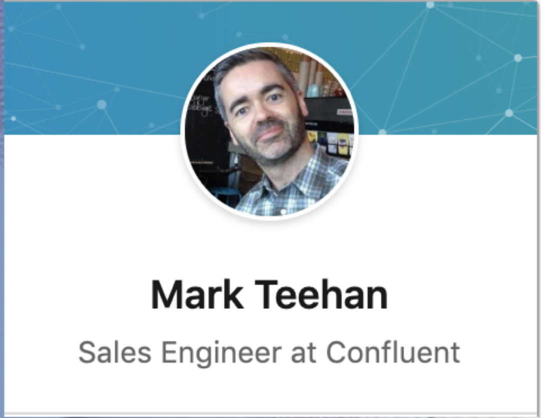
kafka KRU



Mark Teehan

Sales Engineer at Confluent

SEOUL - 18. OCT. 2019

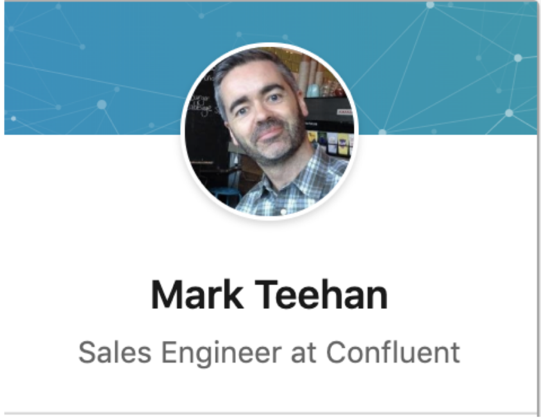
KSQL and Kafka Streams

When to use which, and when to use both

Mark Teehan
Sales Engineer
Confluent

Agenda

- KSQL and Kafka Streams in 3 minutes
- Example Use Cases
- Similarities & Differences
- Guidance



Mark Teehan

Sales Engineer at Confluent

Duration: ~40m

Author Credit: Dr. Michael Noll, Confluent

KSQL and Kafka Streams in 3 minutes

In a nutshell

KSQL

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

kafka
streams

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

Hello, Streaming World

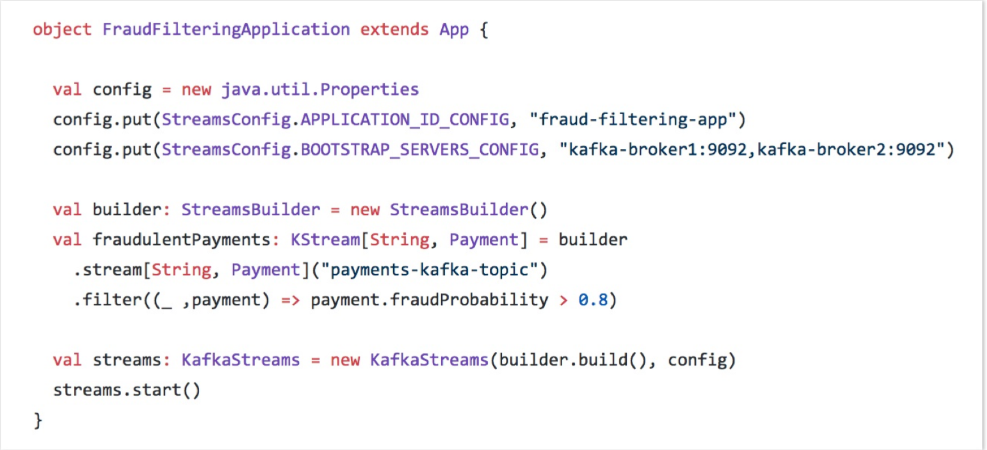
KSQL

```
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, if you want to.

kafka streams



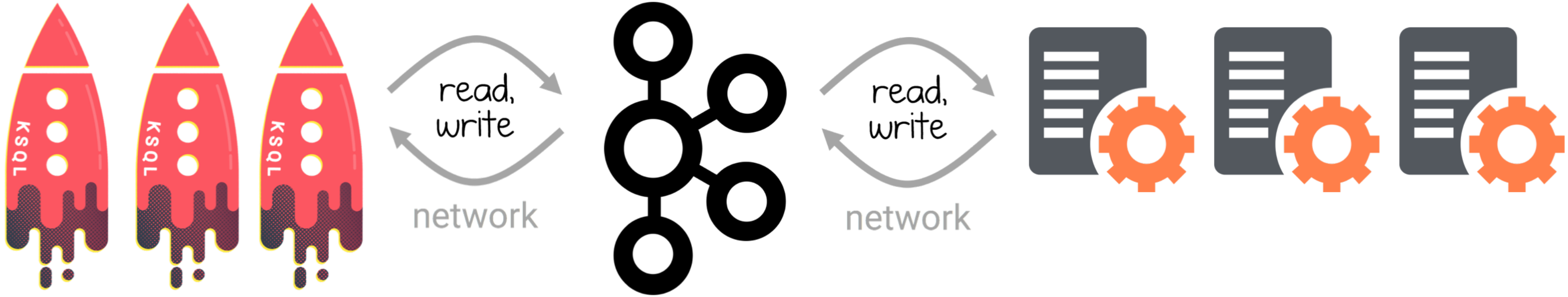
```
object FraudFilteringApplication extends App {

  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()
}
```

Interaction with Kafka

KSQL
(processing)

Kafka
(data)

JVM application
with Kafka Streams (processing)



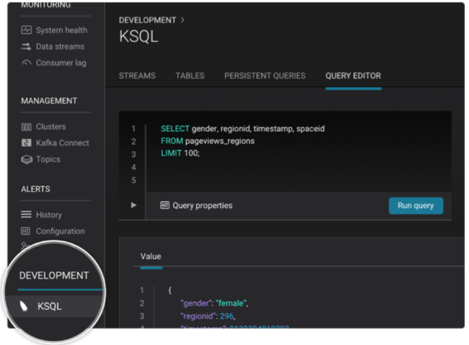
read, write
network

read, write
network

Does not run on Kafka brokers

Does not run on Kafka brokers

KSQL can be used interactively + programmatically



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

```
{
    "gender": "female",
    "regionid": 296,
```

1 UI

```
ksql>
```

2 CLI

```
POST /query
```

3 REST

4 Headless

Example Use Cases

(focus on KSQL)

KSQL for Data Exploration

An easy way to inspect your data in Kafka

```
SHOW TOPICS;
```

```
PRINT 'my-topic' FROM BEGINNING;
```

```
SELECT page, user_id, status, bytes
 FROM clickstream
 WHERE user_agent LIKE 'Mozilla/5.0%';
```

KSQL for Data Transformation

Quickly make derivations of existing data in Kafka

```
CREATE STREAM clicks_by_user_id
   WITH (PARTITIONS=6,
         TIMESTAMP='view_time’
         VALUE_FORMAT='JSON') AS
   SELECT * FROM clickstream
   PARTITION BY user_id;
```

1 Change number of partitions

2 Convert data to JSON

3 Repartition the data

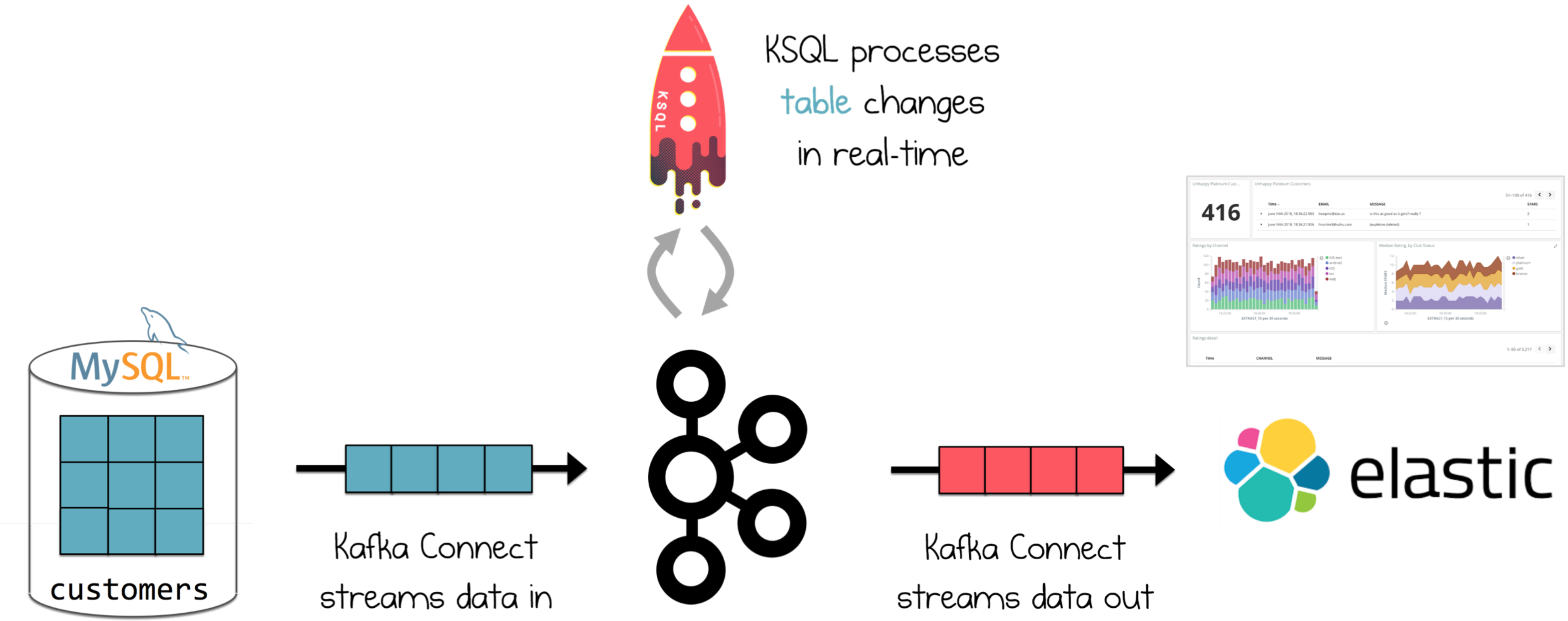
KSQL for Real-Time, Streaming ETL

Filter, cleanse, process data while it is in motion

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

1 Pick only VIP users

Example: CDC from DB via Kafka to Elastic



KSQL processes table changes in real-time

Kafka Connect streams data in

Kafka Connect streams data out

Unhappy Platinum Cust...

416

Unhappy Platinum Customers

51–100 of 416

Time	EMAIL	MESSAGE	STARS
June 14th 2018, 18:36:22.983	ltoopinc@icio.us	is this as good as it gets? really ?	2
June 14th 2018, 18:36:21.926	hrumke3@sohu.com	(expletive deleted)	1

Ratings detail

1–50 of 3,217

Time	CHANNEL	MESSAGE

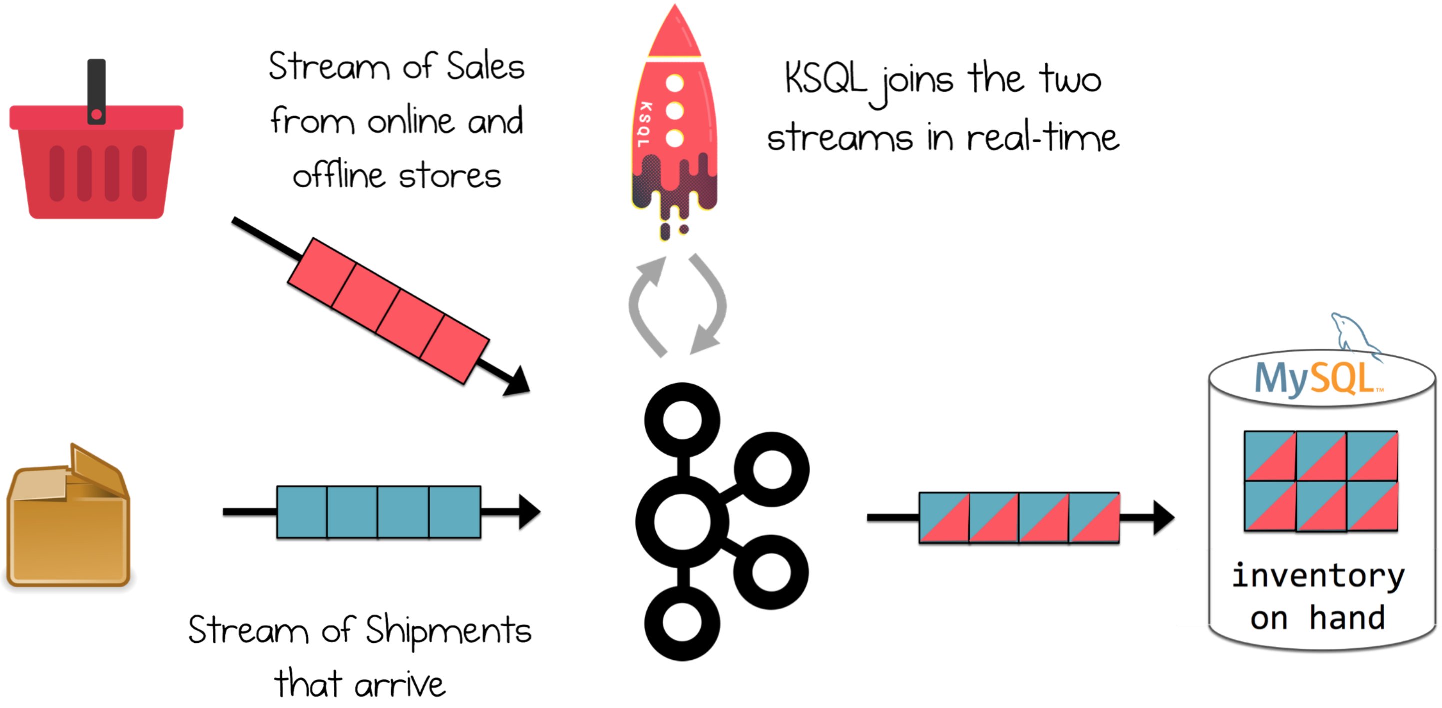
KSQL for Real-time Data Enrichment

Join data from a variety of sources to see the full picture

```
CREATE STREAM enriched_payments AS
   SELECT payment_id, c.country, total
   FROM payments_stream p
   LEFT JOIN customers_table c
          ON p.user_id = c.user_id;
```

1 Stream-Table Join

Example: Retail



Stream of Sales from online and offline stores

KSQL joins the two streams in real-time

Stream of Shipments that arrive

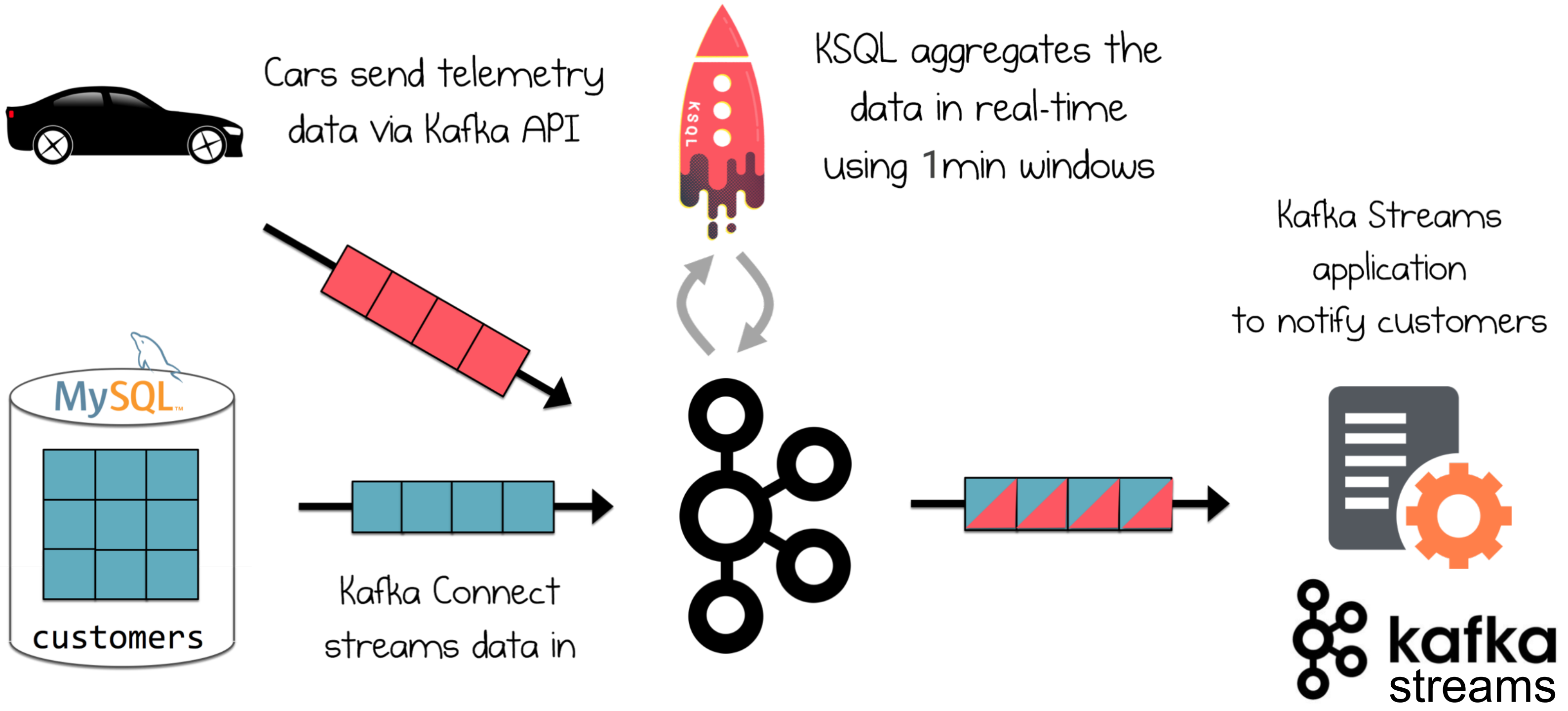
KSQL for Real-Time Monitoring

Derive insights from events (IoT, sensors, etc.) and turn them into actions

```
CREATE TABLE failing_vehicles AS
   SELECT vehicle, COUNT(*)
   FROM vehicle_monitoring_stream
   WINDOW TUMBLING (SIZE 1 MINUTE)
   WHERE event_type = 'ERROR'
   GROUP BY vehicle
   HAVING COUNT(*) >= 5;
```

1 Now we know to alert, and whom

Example: IoT, Automotive, Connected Cars



Cars send telemetry data via Kafka API

KSQL aggregates the data in real-time using 1min windows

Kafka Streams application to notify customers

Kafka Connect streams data in

KSQL for Anomaly Detection

Aggregate data to identify patterns and anomalies in real-time

```
CREATE TABLE possible_fraud AS
   SELECT card_number, COUNT(*)
   FROM authorization_attempts
   WINDOW TUMBLING (SIZE 30 SECONDS)
   GROUP BY card_number
   HAVING COUNT(*) > 3;
```

1 Aggregate data

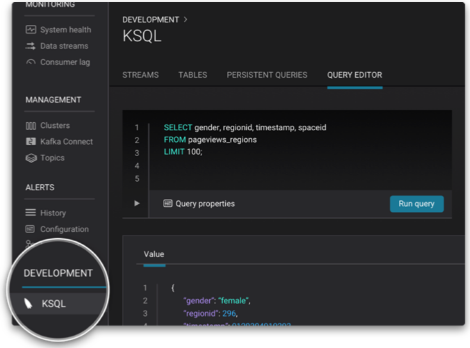
2 … per 30-sec windows

Workflow Comparison

Typical developer interaction

KSQL

kafka streams



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

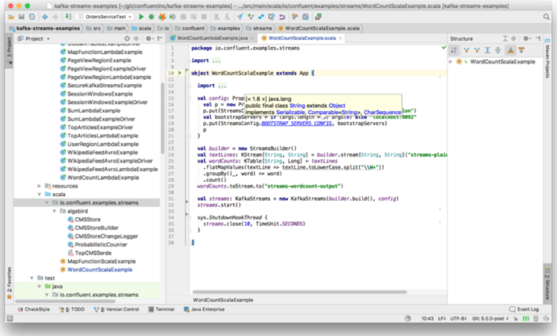
```
{
    "gender": "female",
    "regionid": 296,
```

write KSQL queries

view results in real-time

write code in Java or Scala

recompile, then run/test your app



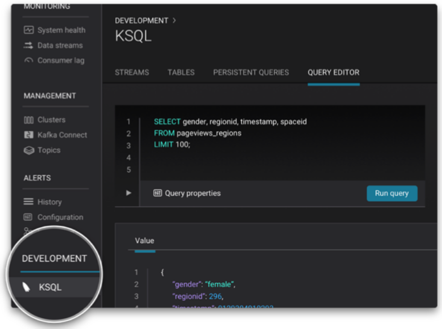
```
package io.confluent.examples.streams

import ...

object WordCountScalaExample extends App {

  import ...

  val config: Prop
    val p = new Pr
    p.put(StreamsC                                                     ion")
    val bootstrapServers = if (args.length > 0) args(0) else "localhost:9092"
    p.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers)
    p
  }

  val builder = new StreamsBuilder()
  val textLines: KStream[String, String] = builder.stream[String, String]("streams-plai
  val wordCounts: KTable[String, Long] = textLines
    .flatMapValues(textLine => textLine.toLowerCase.split("\\W+"))
    .groupBy((_, word) => word)
    .count()
  wordCounts.toStream.to("streams-wordcount-output")

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

  sys.ShutdownHookThread {
    streams.close(10, TimeUnit.SECONDS)
  }

}
```

```
[< 1.8 >] java.lang
public final class String extends Object
implements Serializable, Comparable<String>, CharSequence
```

KSQL: typical workflow from development to production

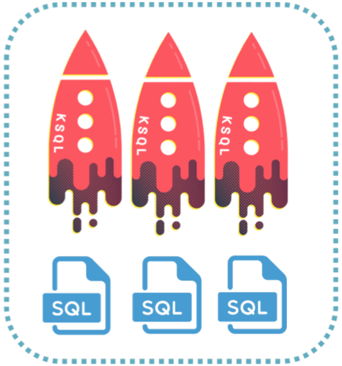
Interactive KSQL
for development

Headless KSQL
in production



```
SELECT gender, regionid, timestamp, spaceid
FROM pageviews_regions
LIMIT 100;
```

```
{
    "gender": "female",
    "regionid": 296,
```



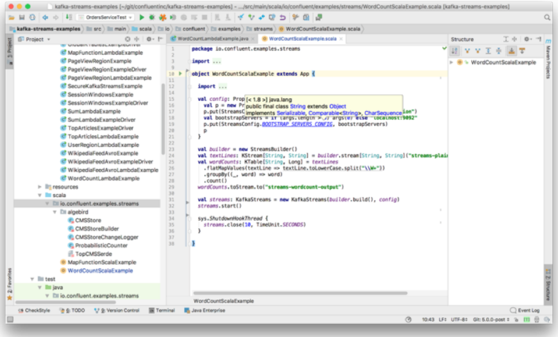
develop your application
and its queries

deploy & run application

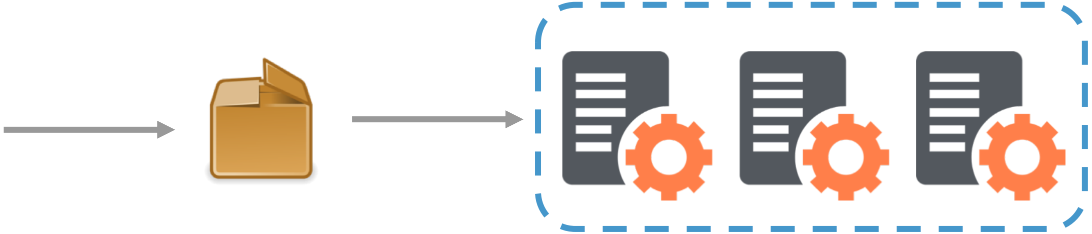
Kafka Streams: typical workflow from development to production

Local development and testing with Java/Scala IDE

Production



```
package io.confluent.examples.streams

import ...

object WordCountScalaExample extends App {

  import ...

  val config: Prop [< 1.8 >] java.lang
    val p = new Pr public final class String extends Object
    p.put(StreamsC implements Serializable, Comparable<String>, CharSequence ion")
    val bootstrapServers = if (args.length > 0) args(0) else "localhost:9092"
    p.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers)
    p
  }

  val builder = new StreamsBuilder()
  val textLines: KStream[String, String] = builder.stream[String, String]("streams-plai
  val wordCounts: KTable[String, Long] = textLines
    .flatMapValues(textLine => textLine.toLowerCase.split("\\W+"))
    .groupBy((_, word) => word)
    .count()
  wordCounts.toStream.to("streams-wordcount-output")

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

  sys.ShutdownHookThread {
    streams.close(10, TimeUnit.SECONDS)
  }

}
```



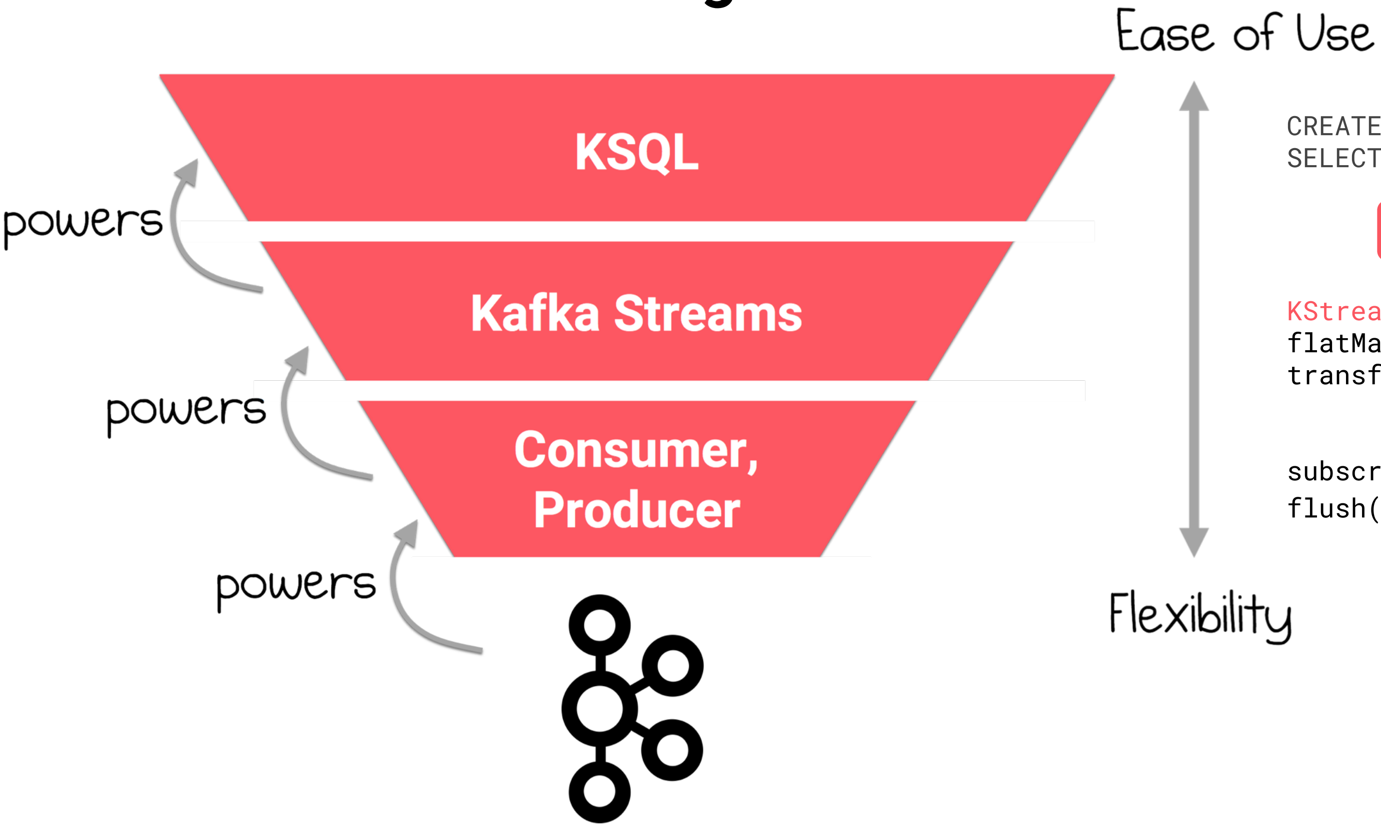
develop your application

build & package the Java/Scala application

deploy & run application

Similarities

Shoulders of Streaming Giants



Ease of Use

KSQL

powers

Kafka Streams

powers

Consumer, Producer

powers

Flexibility

```
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, …
```

KSQL UDFs

```
KStream, KTable, filter(), map(),
flatMap(), join(), aggregate(),
transform(), …
```

```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

Similarities of KSQL & Kafka Streams

Enterprise Support

Open Source

Runs Everywhere

Elastic, Scalable, Fault-tolerant

Kafka Security Integration

Powerful Processing incl. Filters, Transforms, Joins, Aggregations, Windowing

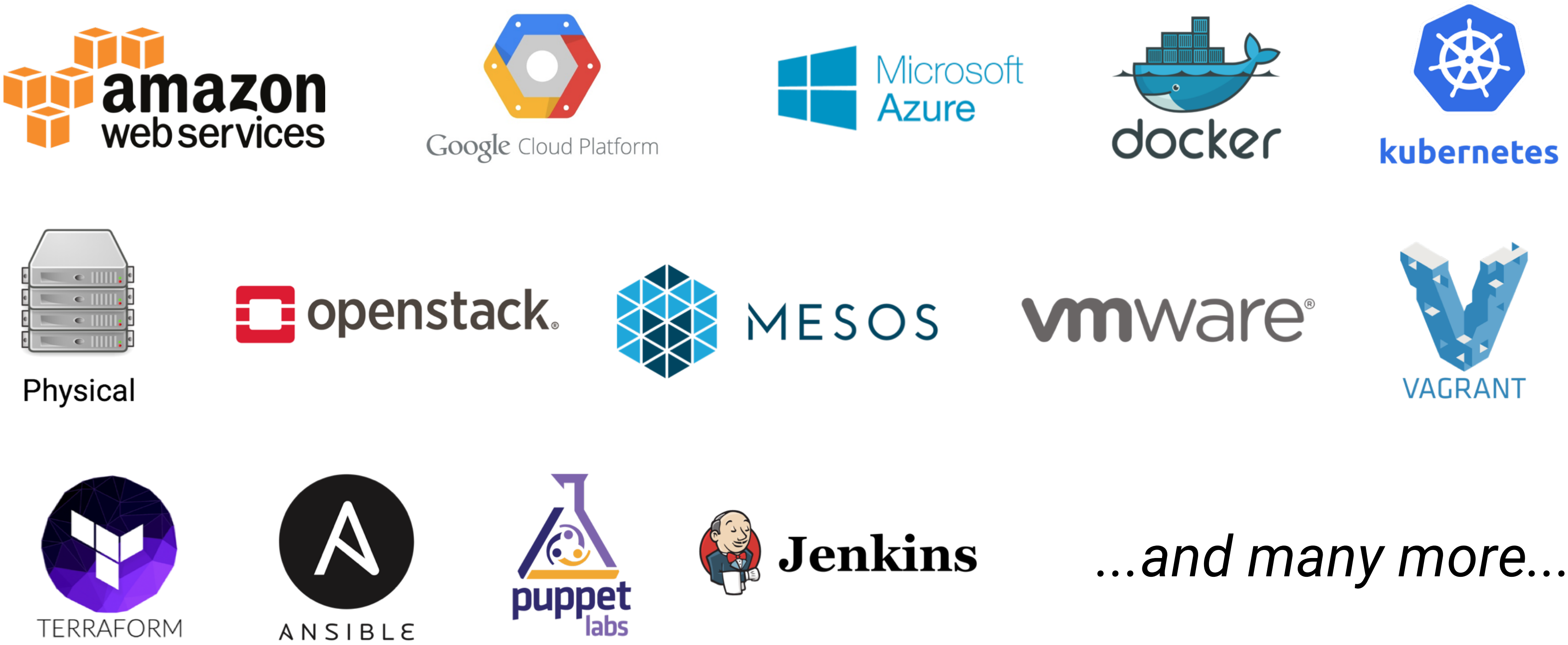
Supports Streams and Tables

Exactly-Once Processing

Event-Time Processing

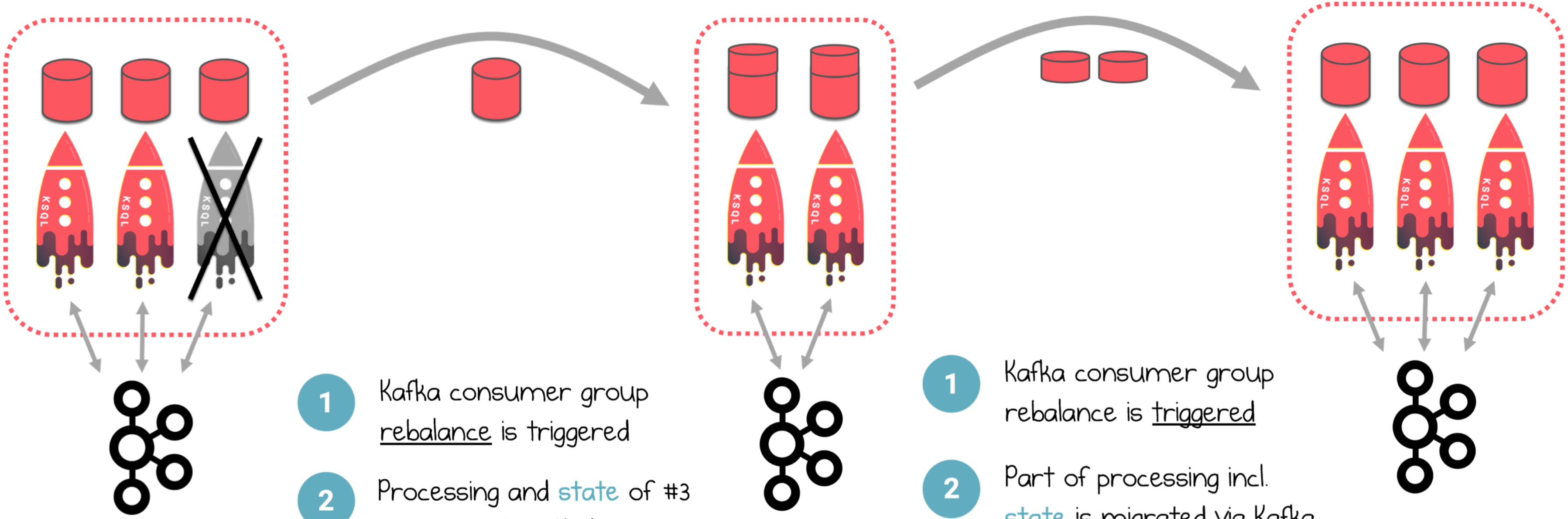
Can Be Used Together

Runs Everywhere, Integrates Smoothly with What You Have



...and many more...

Fault-Tolerance, powered by Kafka (here: KSQL)



#3 died so #1 and #2 take over

1. Kafka consumer group rebalance is triggered
2. Processing and state of #3 is migrated via Kafka to remaining servers #1 + #2

#3 is back so the work is split again

1. Kafka consumer group rebalance is triggered
2. Part of processing incl. state is migrated via Kafka from #1 + #2 to server #3

Differences

Differences

	KSQL	kafka streams
You write...	**KSQL statements**	**JVM applications**
UI included for human interaction	**Yes**, in Confluent Enterprise	No
CLI included for human interaction	**Yes**	No
Data formats	Avro, JSON, CSV (today)	**Any data format**, including Avro, JSON, CSV, Protobuf, XML
REST API included	**Yes**	No, but you can DIY
Runtime included	**Yes**, the KSQL server	**Not needed**, applications run as standard JVM processes
Queryable state	Not yet	**Yes**

Guidance

KSQL

kafka streams

And remember: They can also be used together!

Start with KSQL when...

- New to streaming and Kafka
- To quicken and broaden the adoption & value of Kafka in your organization
- Prefer an interactive experience with UI and CLI
- Prefer SQL to writing code in Java or Scala
- Use cases include enriching data; joining data sources; filtering, transforming, and masking data; identifying anomalous events
- Use case is naturally expressible through SQL, with optional help from User Defined Functions as “get out jail free” card
- Want the power of Kafka Streams but you are not on the JVM: use the KSQL REST API from Python, Go, C#, JavaScript, shell

Start with Kafka Streams when...

- Prefer writing and deploying JVM applications like Java and Scala; e.g. due to people skills, tech environment
- Use case is not naturally expressible through SQL, e.g. finite state machines
- Building microservices
- Must integrate with external services, or use 3rd-party libraries (but KSQL UDFs may help)
- To customize or fine-tune a use case, e.g. with Kafka Streams’ Processor API; examples: custom join variants, probabilistic counting at very large scale with Count-Min Sketch
- Need for queryable state, which is not yet supported by KSQL

KSQL is usually not yet a good fit for:
BI reports & ad-hoc querying, queries with random access patterns (because no indexes, no native JDBC)

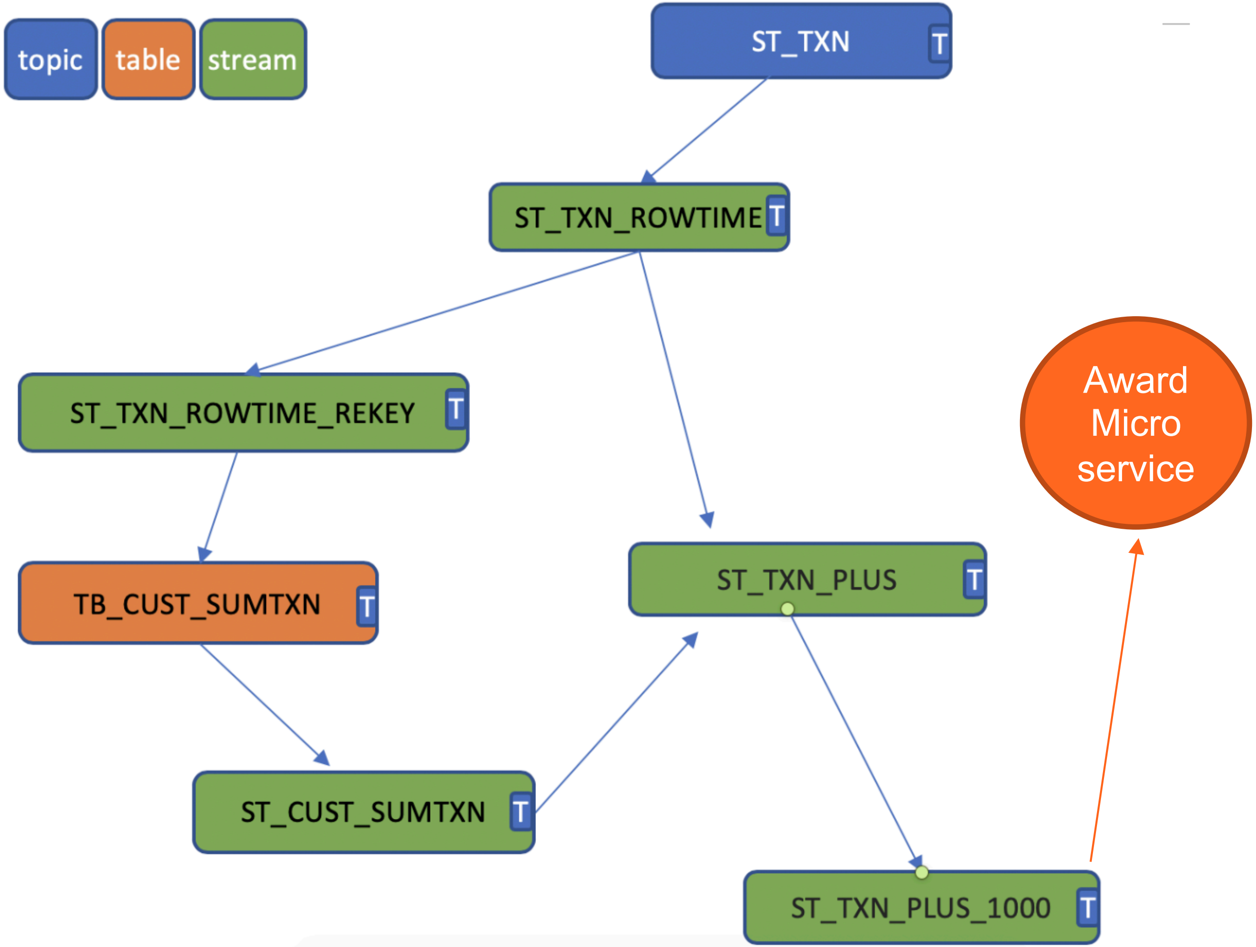
STREAM PROCESSING COOKBOOK

KSQL Recipes

Credit Card promotion:

Monitor the first 3 million customers that spend $1000 using a new credit card

Based on Confluent kSQL Recipe “Inline Streaming Aggregation”



STREAM PROCESSING COOKBOOK

KSQL Recipes

```
CREATE STREAM ST_TXN
WITH
(kafka_topic='CC_TXN',
value_format='AVRO');
```

```
CREATE STREAM ST_TXN_ROWTIME
AS SELECT CUSTID, THISAMOUNT,
TXNDATE, ROWTIME as C_ROWTIME
FROM ST_TXN;
```

```
CREATE STREAM
ST_TXN_ROWTIME_REKEY AS
SELECT * FROM ST_TXN_ROWTIME
PARTITION BY CUSTID;
```

```
CREATE TABLE TB_CUST_SUMTXN AS
SELECT CUSTID as C_CUSTID
, cast(count(*) as bigint)      as C_COUNTTXN
, SUM(cast(THISAMOUNT as DOUBLE)) as C_SUMTXN
, max(UNIQ)                          as MAX_TS
FROM ST_TXN_ROWTIME_REKEY
GROUP BY CUSTID;
```

```
CREATE STREAM ST_CUST_SUMTXN  WITH
(KAFKA_TOPIC='TB_CUST_SUMTXN',
VALUE_FORMAT='AVRO');
```

```
CREATE STREAM ST_TXN_PLUS AS
SELECT
CUSTID, THISAMOUNT
,cast(C_SUMTXN as INT)    as C_SUMTXN
,cast(C_COUNTTXN as INT) as C_COUNTTXN
FROM ST_TXN_ROWTIME_KEY
JOIN ST_CUST_SUMTXN WITHIN 60 MINUTES
ON (CUSTID = C_CUSTID)
WHERE C_ROWTIME = MAX_TS;
```

```
CREATE STREAM ST_TXN_PLUS_1000
AS
SELECT * FROM ST_TXN_PLUS
 WHERE (C_SUMTXN > 1000)
   AND (C_SUMTXN - THISAMOUNT) <=1000;
```

Confluent Community - What next?

Join the Confluent Community Slack Channel

About 10,000 Kafkateers are collaborating every single day on the Confluent Community Slack channel!

cnfl.io/community-slack

The Confluent Community Catalyst Program

NOMINATE YOURSELF OR A PEER AT

CONFLUENT.IO/NOMINATE

Subscribe to the Confluent blog

Get frequent updates from key names in Apache Kafka® on best practices, product updates & more!

cnfl.io/read

Apache, Apache Kafka, Kafka and the Kafka logo are trademarks of the Apache Software Foundation. The Apache Software Foundation has no affiliation with and does not endorse the materials provided at this event.

Until the end of 2019 Confluent are giving new users $50 of free usage per month for their first 3 months

Here’s advice on how to use this promotion to try Confluent Cloud for free!

Sign up for a Confluent Cloud account

Please bear in mind that you will be required to enter credit card information but will not be charged unless you go over the $50 usage in any of the first 3 months or if you don’t cancel your subscription before the end of your promotion.

You won’t be charged if you don’t go over the limit!

Get the benefits of Confluent Cloud, but keep an eye on your your account making sure that you have enough remaining free credits available for the rest of your subscription month!!

Cancel before the 3 months end If you don’t want to continue past the promotion

If you fail to cancel within your first three months you will start being charged full price. To cancel, immediately stop all streaming and storing data in Confluent Cloud and email cloud-support@confluent.io

bit.ly/**TryConfluentCloud**

Available on

Google Cloud Platform

Microsoft Azure

aws

THANK YOU !

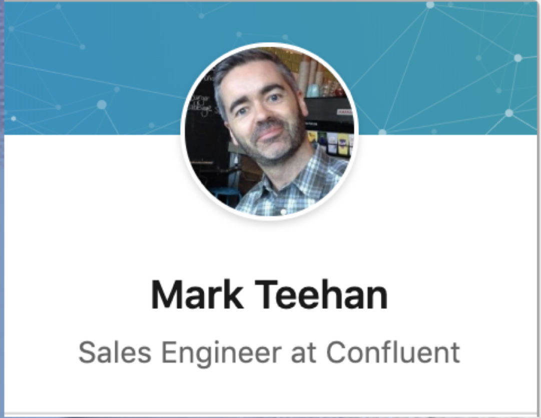
Learn more:

confluent.io/download

confluent.io/product/ksql/

confluent cloud

confluent.io/confluent-cloud/



Mark Teehan

Sales Engineer at Confluent

kafka KRU