Introduction to Apache Kafka

Thessaloniki not-only-Java Meetup

Dimitris Kontokostas @Diffbot

About me...

Data geek,

Software engineer

Open source enthusiast,

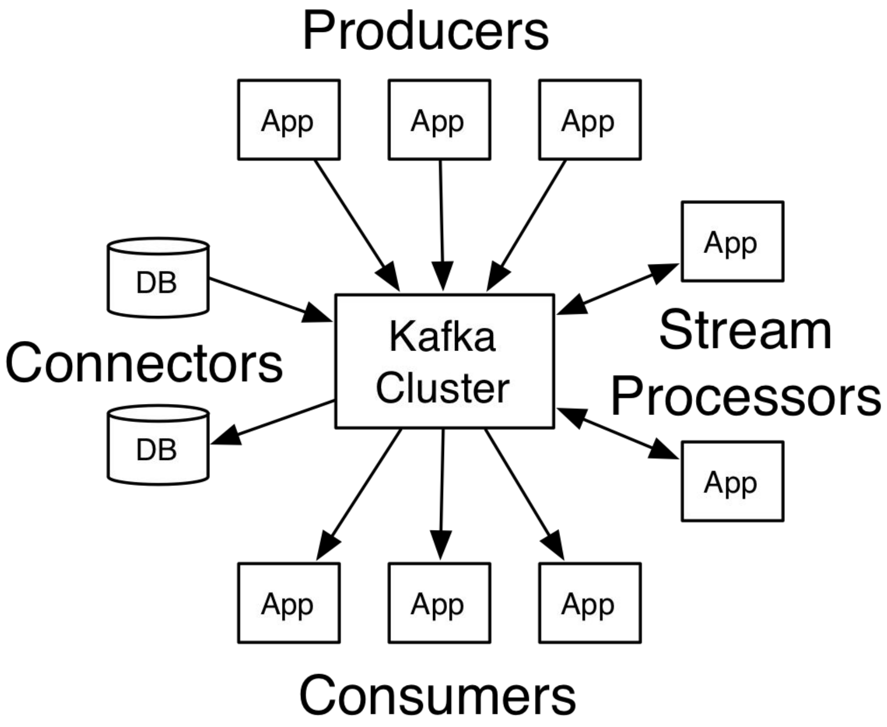
Remote Worker

Currently working @ Diffbot

Summary

- The Log
- Kafka 101
- KStream
- KSQL
- Connectors
- Schemas

Let's keep this interactive...



Kafka is

A distributed streaming platform

A distributed publish-subscribe messaging system/queue

A distributed, immutable event logging data store $^{(*)}$

The Log

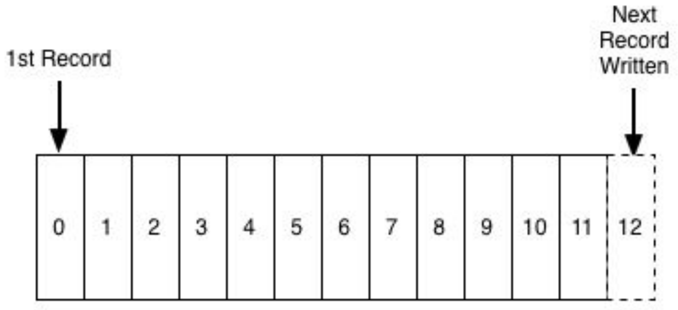
What every software engineer should know about real-time data's unifying abstraction

A great post by Jay Kreps (creator of Kafka)

Parts of the post are used in this presentation

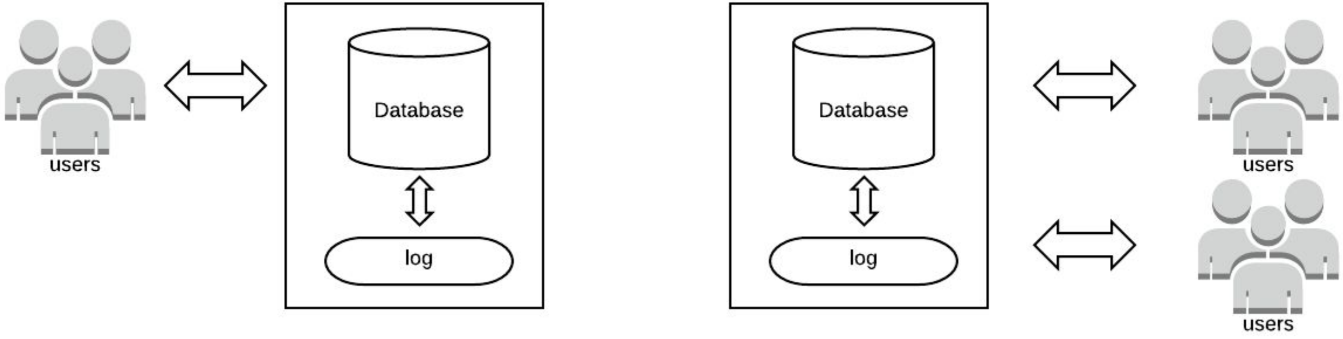
What is a log

- Records **events**
- **What** happened & **when**
- **Append only**, left-to-right
- Timestamps



Logs in databases

- As internal structures to facilitate ACID
- Gradually getting more exposed
- As a data subscription mechanism



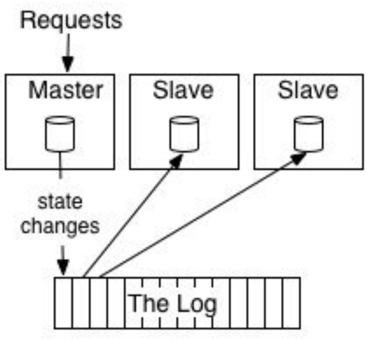
Logs in distributed systems

Ordering

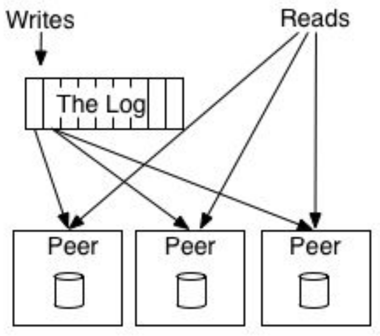
Replication

Active-active vs Active-passive

- E.g. “+5”, “-2”, “*4”
- Vs “0”, “5”, “3”, “12”



Primary-Backup



State-Machine Replication

Events & Tables

Two sides of the same coin

events & tables

```
1. User_add {id:1, email: “joh@doe.com”}

2. User_change {id:1 email: “john@doe.com”}

3. User_add {id:2, email: “mary@doe.com”}

4. User_change {id:2, email: “mary@doe.gr”}

5. User_add {id:3, email: “peter@doe.com”}
```

Table User

id	email
1	?
2	?
3	?

events & tables

1. **User_add** {id:1, email: “joh@doe.com”}
2. **User_change** {id:1 email: “john@doe.com”}
3. **User_add** {id:2, email: “mary@doe.com”}
4. **User_change** {id:2, email: “mary@doe.gr”}
5. **User_add** {id:3, email: “peter@doe.com”}

Table User

id	email
1	john@doe.com
2	mary@doe.gr
3	peter@doe.com

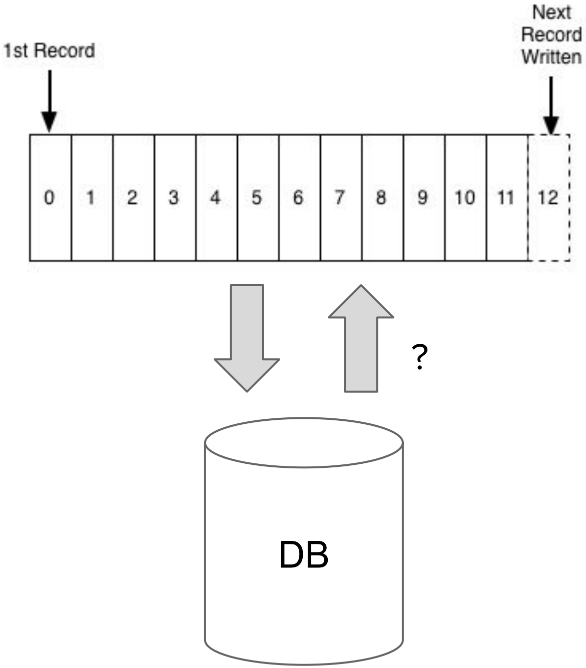
events & tables

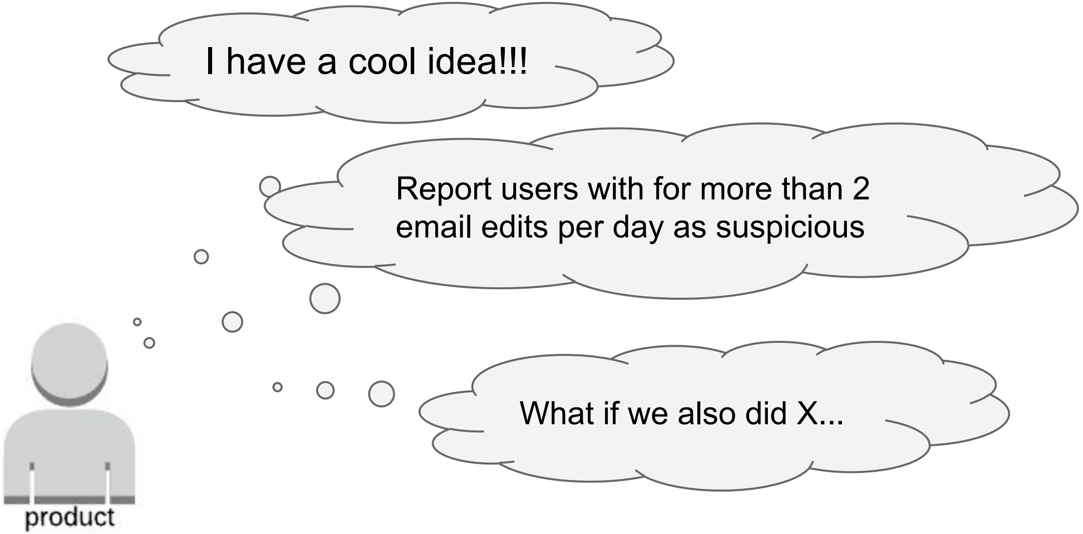
```
1. User_add {id:1, email: “joh@doe.com”}

2. User_change {id:1 email: “john@doe.com”}

3. User_add {id:2, email: “mary@doe.com”}

4. User_change {id:2, email: “mary@doe.gr”}

5. User_add {id:3, email: “peter@doe.com”}

6. User_remove{id:1 }
```

Table User

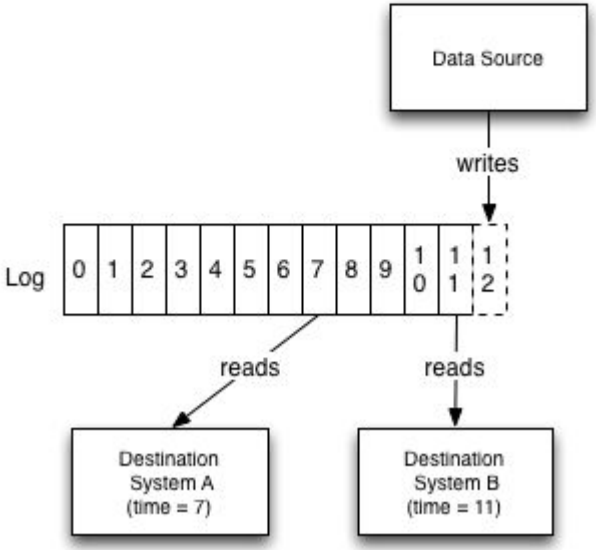
id	email
1	john@doe.com
2	mary@doe.gr
3	peter@doe.com

events & tables




I have a cool idea!!!
Report users with for more than 2 email edits per day as suspicious
What if we also did X...
product

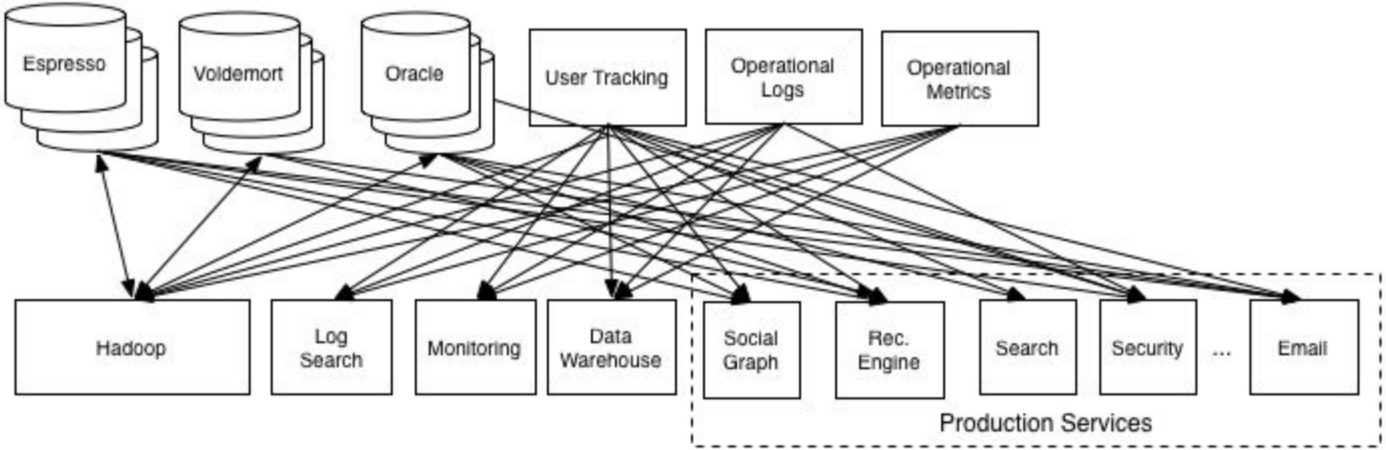
Decoupling from the table


Data Source
writes
Log
0
1
2
3
4
5
6
7
8
9
10
11
12
reads
reads
Destination System A (time = 7)
Destination System B (time = 11)

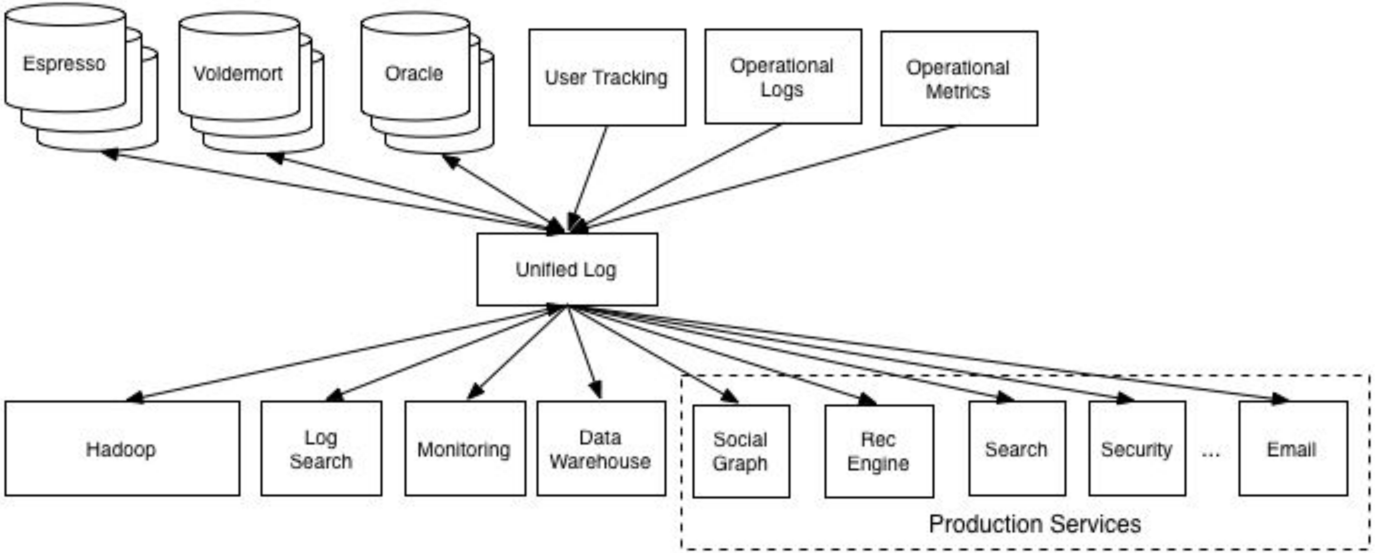
Tables are opinionated views of your data

The later you can defer your opinion, the better...

LinkedIn before Kafka



LinkedIn after Kafka



Kafka Basics

Basic terminology

Kafka maintains feeds of **messages** in categories called **topics**

Kafka Producers are processes that **publish messages** on a kafka topic

Kafka Consumers are processes that **subscribe to topics** and process the feed of published messages

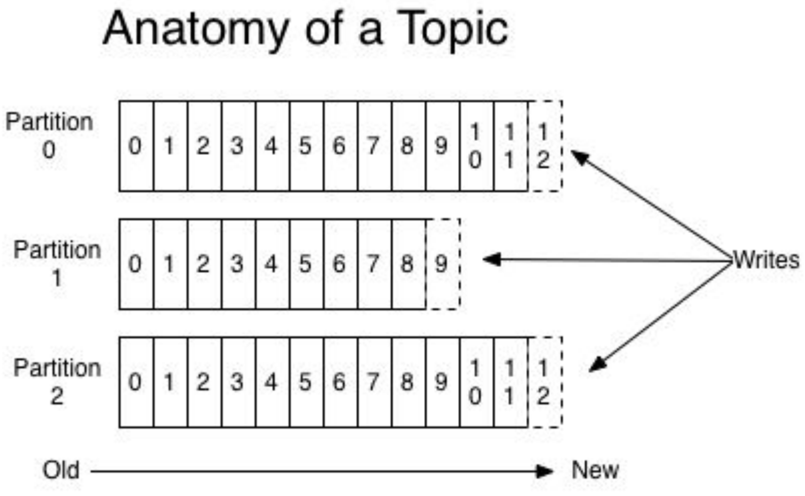
Messages can be in any form, text, json, binary etc.

Kafka brokers are the servers that comprise the **kafka cluster**

Topics & Logs

1 Topic ≈ 1 Partitioned Log

(with configurable retention period)



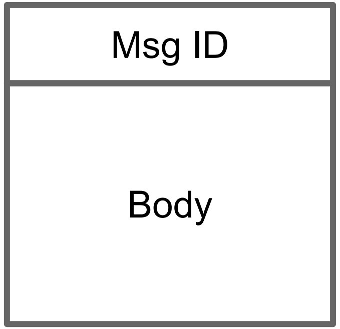
What is the difference with “The Log”?

Ordering…

Kafka Messages

Anything can be a message

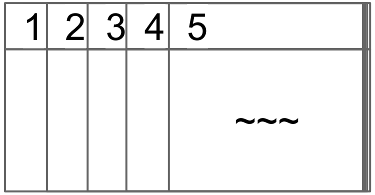
=> Strings, JSON, XML, binary formats



Messages are preferably small (< few Kb)

=> Big messages affect brokers & throughput

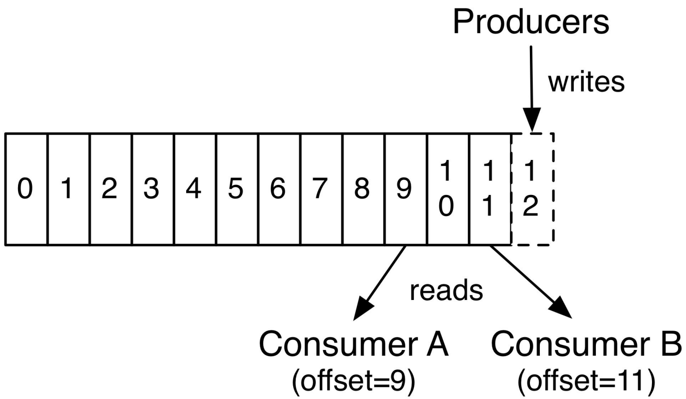
=> Reference big messages as external resources



Message IDs define which partition the message is stored

=> Default behavior / can be manually overwritten

Producers & Consumers



Message Delivery Semantics

=> At least once

=> At most once

=> Exactly once

Multiple (independent) Consumers can read from the same topic

=> Consumers manage their own offset (stored on Kafka)

=> Messages remain on Kafka

Consumer Scaling

Every Consumer has a **Consumer ID**

=> Kafka keeps track of the offets

=> rejoining will continue from latest offset

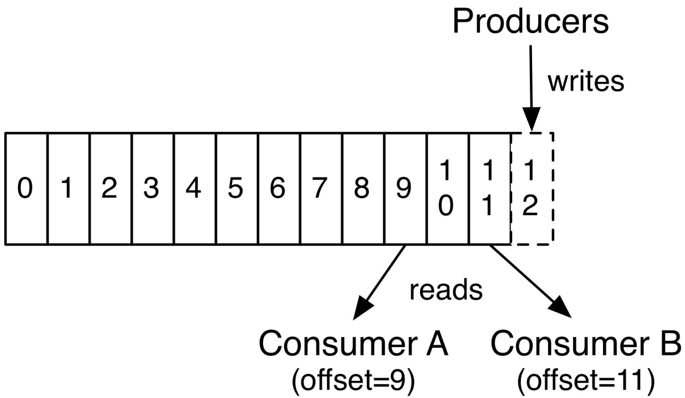
Consumers with the same ID

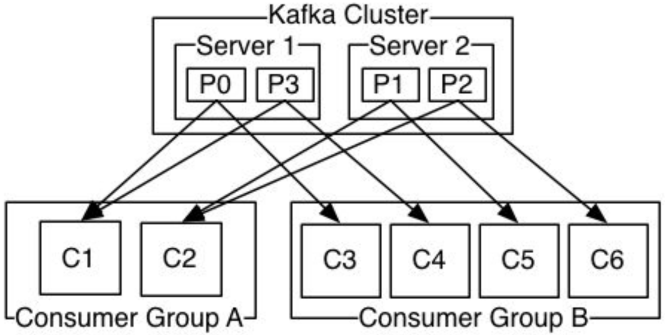
belong to the same **Consumer Group**

Consumers from the same group

read messages from **different partitions**

=> Topic partition size is the hard limit





KStreams/KSQL

Print topic messages (Consumer API)

```
public class KafkaConsumerExample  {
 ...
   static void runConsumer()  throws InterruptedException {
       final Consumer<Long, String> consumer = createConsumer();
       while (true) {

           final ConsumerRecords< Long, String> consumerRecords = consumer.poll();

           consumerRecords.forEach(record -> {
               System.out.printf( "Consumer Record:(%d, %s, %d, %d)\n" ,
                      record.key(), record.value(),record.partition(), record.offset());
           });

           consumer.commitAsync();
       }
       consumer.close();
   }}
```

https://dzone.com/articles/writing-a-kafka-consumer-in-java

Print topic messages (KStream API)

```
static void createWordCountStream(final StreamsBuilder builder) {

   final KStream<String, String> textLines = builder.stream(inputTopic);

   textLines
     .forEach(value -> System.out.printLn(value));
 }
```

KStream API

Abstraction on top of classic Kafka API

Provide a Java Stream-like API on Kafka

=> has a source, sink & stream processors

=> aggregations, windowing, KTables / state

Source processor

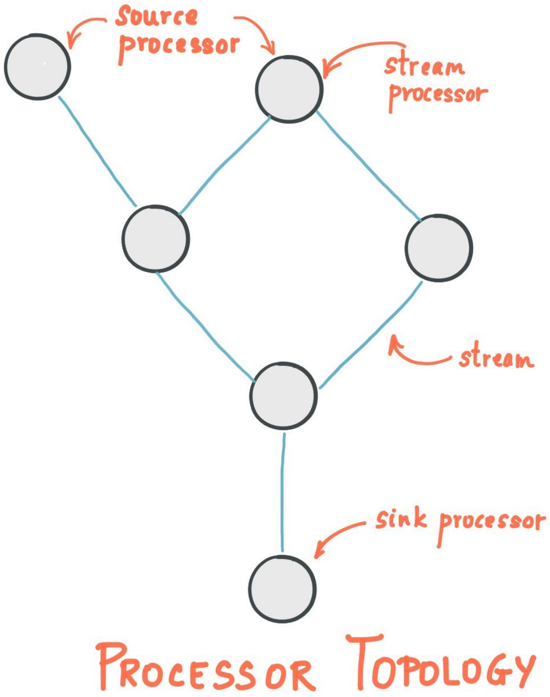
=> consumes on or more topics, forwards downstream

Stream processors

=> apply transformations, aggregations, etc

Sink Processors

=> final processors, stores results on a topic or KTable



KTables

Read-only & fault-tolerant state store

Can be in memory (on consumer) or global (persistent on the cluster)

RocksDB is a default implementation

Example: keep an up-to-date table of `[user id, email]`

Aggregations & Windowing

Example: Users with more than 2 email changes in 24 hours

```
final KTable<Windowed<String>, Long> anomalousUsers = views
   .groupByKey()
   .windowedBy(TimeWindows.of(Duration.ofDays(1)))
   .count()
   .filter((windowedUserId, count) -> count >= 2);

final KStream<String, Long> anomalousUsersForConsole = anomalousUsers
   .toStream()
   .filter((windowedUserId, count) -> count != null)
   .map((windowedUserId, count) -> new KeyValue<>(windowedUserId.toString(), count));

anomalousUsersForConsole.to("AnomalousUsers", Produced.with(stringSerde, longSerde));
```

KSQL

```
CREATE TABLE users_view AS
    SELECT * FROM USERS;



CREATE STREAM suspicious_users AS
  SELECT id,  count(*)
  FROM USERS
  WINDOW SESSION (1 DAY)
  GROUP BY id;
  HAVING count(*) >= 2;
```

Disclaimer: most probably has typos !!!

Connectors & Schema Registries

Big library of source & sink Kafka connectors

=> S3, most DBs, ES, HDFS, other streaming frameworks

=> e.g. write this topic to MySQL table X

=> Kafka Connect API for writing a custom one

Schemas always change, not only in RDBMS :)

=> Message body freedom has a cost

=> Schema registries come to the… hmm… help…

=> data validation, de/serialization, backwards compatibility

Summary

Logs are everywhere

Logs help us defer data schema decision

Kafka is an event logging data store

Has a great ecosystem of libraries & integrations

Thank you for your attention

Questions?