
# Kafka Streams and Protobuf

2017-02-04, Brussels, Belgium

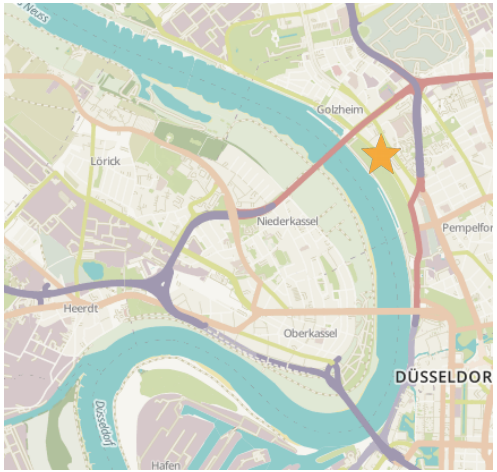
Clemens Valiente


## Clemens Valiente

Senior Data Engineer
trivago Düsseldorf


Originally a mathematician
Studied at Uni Erlangen
At trivago for 5 years


Email: clemens.valiente@trivago.com
de.linkedin.com/in/clemensvaliente

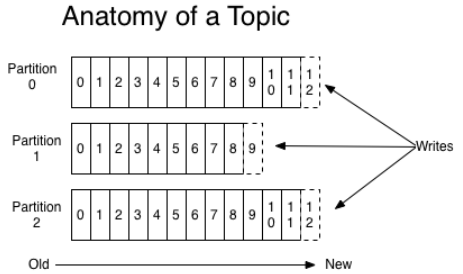



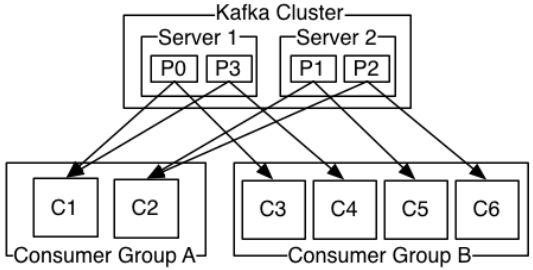
# Agenda

1. Kafka – quick introduction
2. Kafka Streams Concepts
3. Google Protocol Buffers
4. Stream processing with Kafka Streams and protobuf

# Apache Kafka (very quick intro)

- Distributed streaming platform
- Fault tolerant, replicated
- Consumer group responsible for rebalancing and scaling

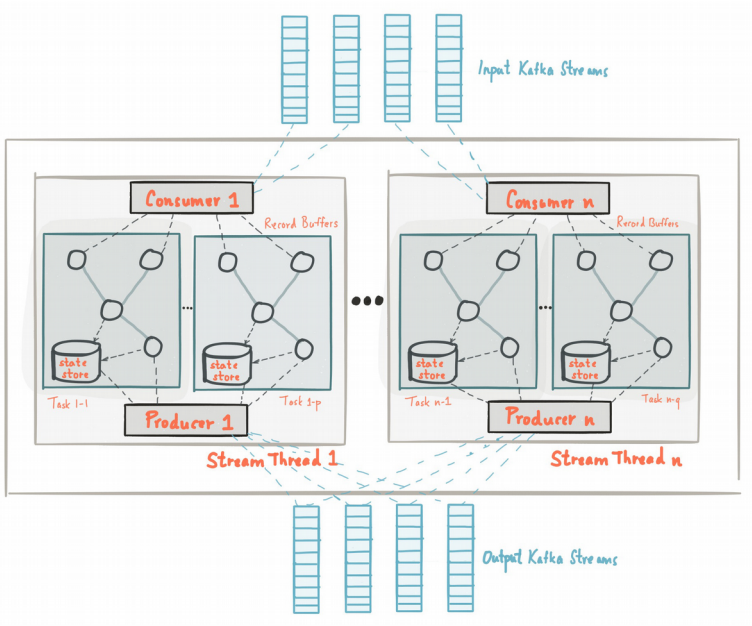



https://kafka.apache.org/intro

# Apache Kafka Streams

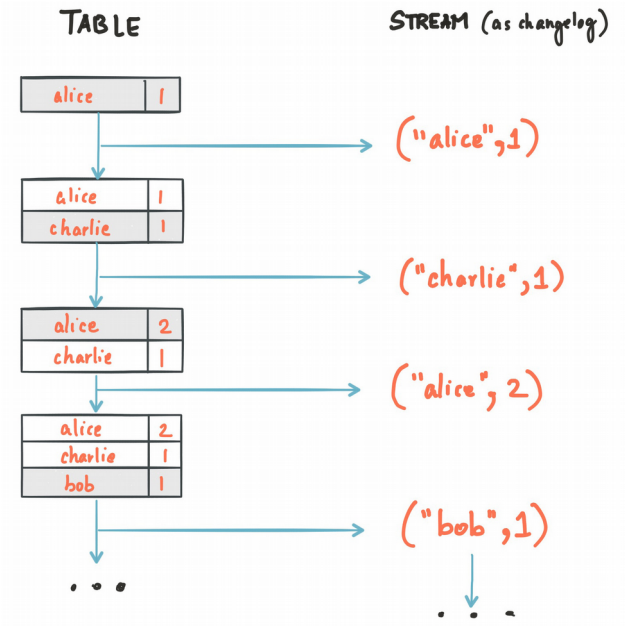
- Small library instead of huge framework, no further external dependencies
- Simple java application: build fatJar and run (on yarn, Mesos, Docker, Kubernetes...)
- Uses consumer group logic for elastic scaling, fault tolerance and distributing workload
- Ideal for a microservice architecture




http://docs.confluent.io/3.1.2/streams/architecture.html#streams-architecture

# Apache Kafka Streams – KTables and KStreams

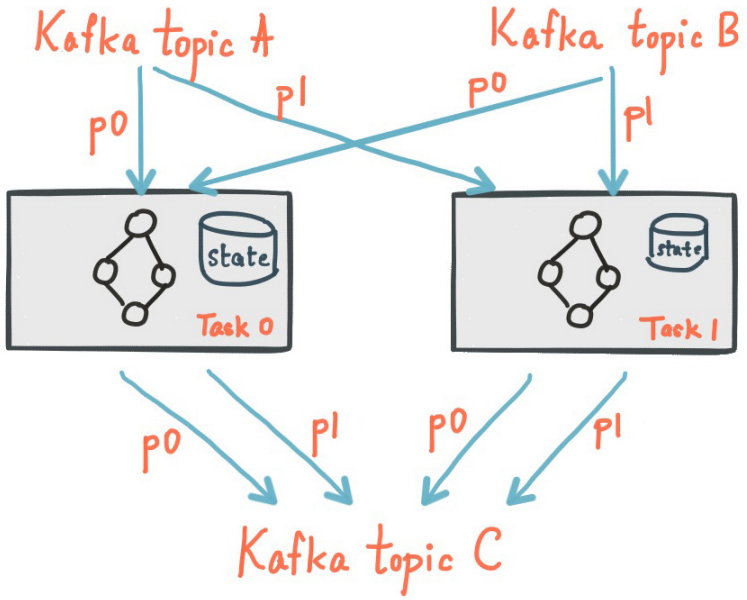
- Streams are infinite event logs
- Tables are finite with updates/deletes/inserts on the key
- Both can be represented as a kafka topic (table changelog)
- Join via key (e.g. customer event stream with customer table)




http://docs.confluent.io/3.1.2/streams/architecture.html#streams-architecture

# Apache Kafka Streams – local and global state

- State of a stream task is stored locally in a statestore, e.g. RocksDB (by default)
- Writes to this statestore are also replicated to a Kafka topic as a changelog
- On rebalancing or recovery, new tasks rebuild the state from the Kafka topic
- You can even query the local state




http://docs.confluent.io/3.1.2/streams/architecture.html#streams-architecture

# Kafka and Avro

- Several parts of the Confluent platform and Kafka assume you are using avro:
- Kafka Connect
- Schema Registry

=> Natural choice

- You always need to have the correct schema in the correct version to be able to read your messages reliably
- If you don’t have a schema, all your data is just byte garbage

# Google Protocol Buffers

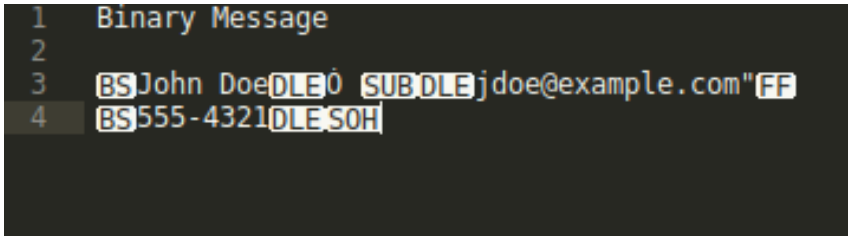
- Similar to Avro in a lot of its features
  - Binary encoded
  - Defined schema
  - Support for lots of languages
- But: Can also read messages with different schema and no schema at all!

Person.proto

```
message Person {
   required string name = 1;
   required int32 id = 2;
   optional string email = 3;

   enum PhoneType {
      MOBILE = 0;
      HOME = 1;
      WORK = 2;
   }

   message PhoneNumber {
      required string number = 1;
      optional PhoneType type = 2 [default = HOME];
   }

   repeated PhoneNumber phones = 4;
}
```

https://developers.google.com/protocol-buffers/docs/javatutorial

# Example Person




**Decoded Message with Schema**

```
name: "John Doe"
id: 1234
email: "jdoe@example.com"
phones {
  number: "555-4321"
  type: HOME
}
```

**Decoded Message with old Schema**

```
name: "John Doe"
id: 1234
phones {
  number: "555-4321"
  type: HOME
}
3: "jdoe@example.com"
```

**Decoded Message without Schema**

```
1: "John Doe"
2: 1234
3: "jdoe@example.com"
4 {
  1: "555-4321"
  2: 1
}
```

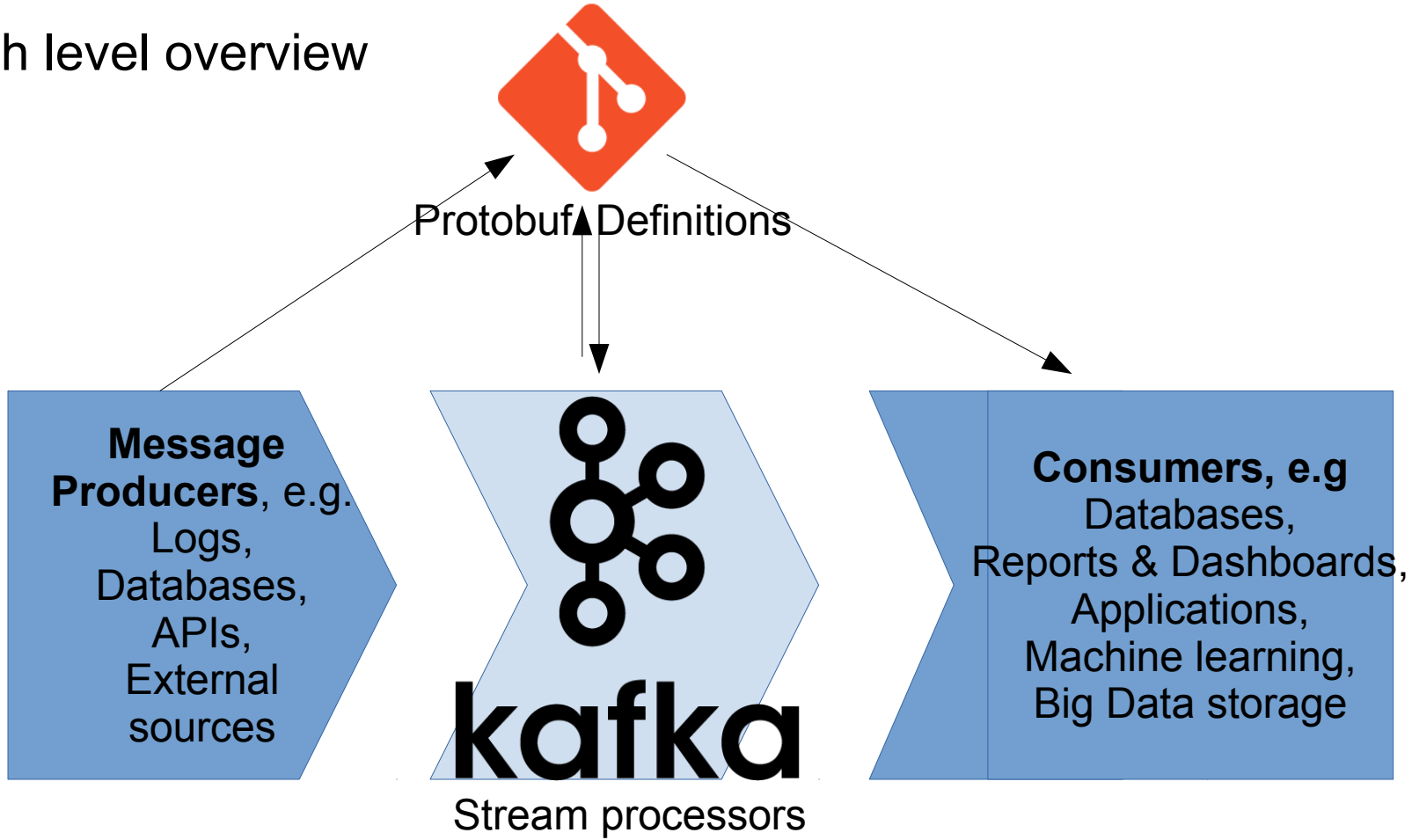
# The magic of unknown fields

If you build a new message from an existing one, all unknown fields will be copied over and included in the new message

```
this.unknownFields = builder.getUnknownFields();
```

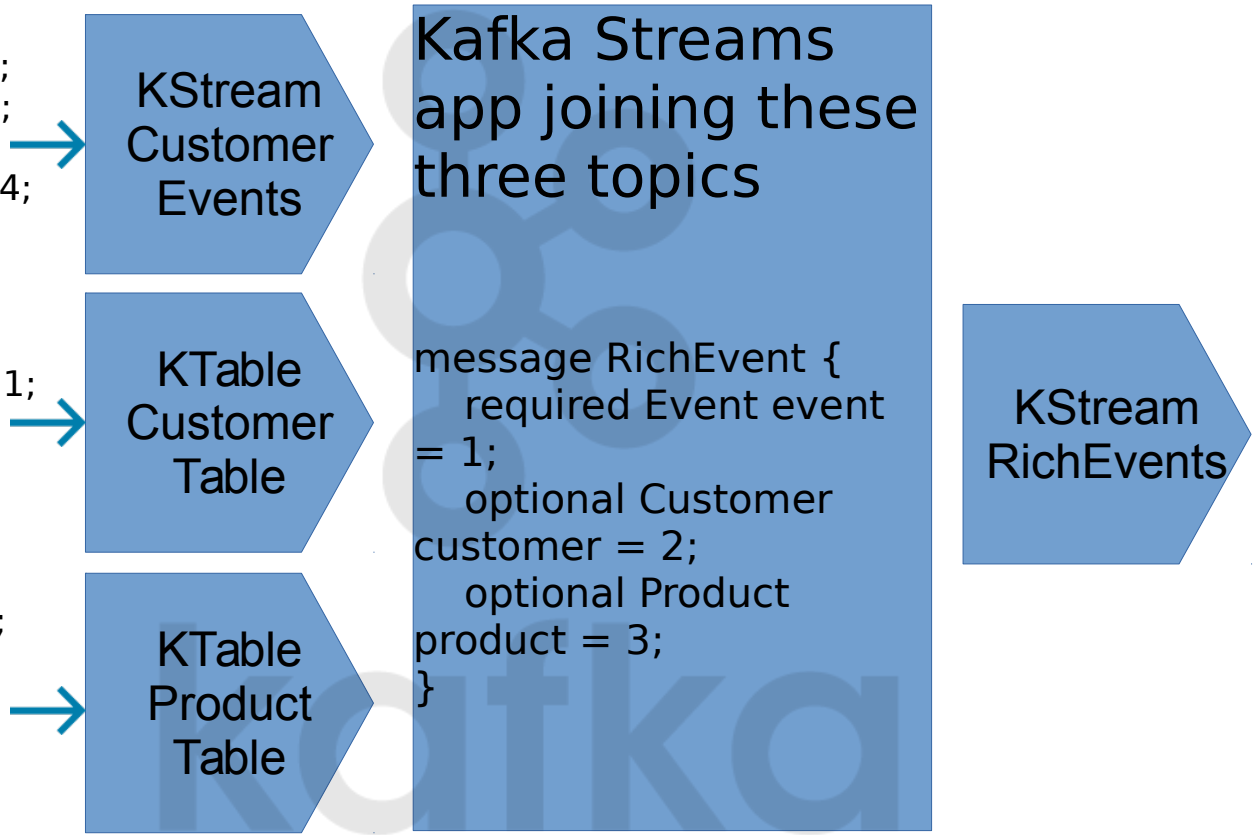
Even if your application doesn’t have the correct schema, it will not break the original content

# High level overview



# Tying it all together

```
message Event {
    required int64 timestamp = 1;
    required string session_id = 2;
    required string type = 3;
    optional int32 customer_id = 4;
    optional int32 product_id = 5;
}

message Customer {
    required int32 customer_id = 1;
    optional string name = 2;
    optional string email = 3;
}

message Product {
    required int32 product_id = 1;
    optional string name = 2;
    optional float price = 3;
}
```

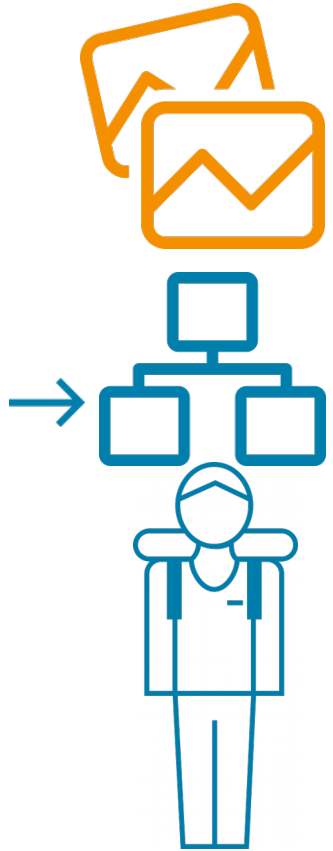


# Tying it all together

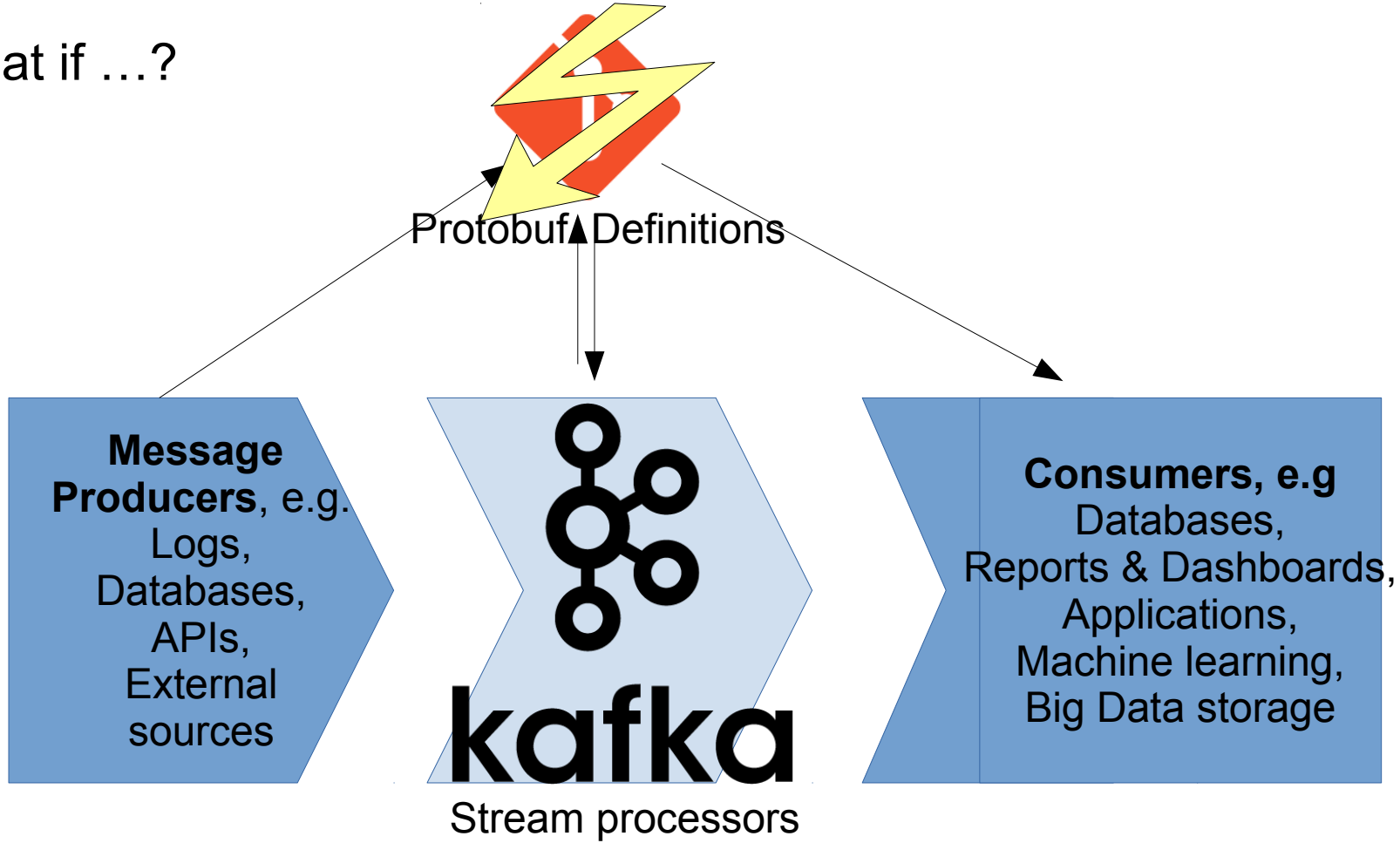
```
message RichEvent {
    required Event event = 1;
    optional Customer
customer = 2;
    optional Product product =
3;
}
```


Kafka Streams app collecting into sessions

```
message Session {
    repeated RichEvents
richevents = 1;
}
```

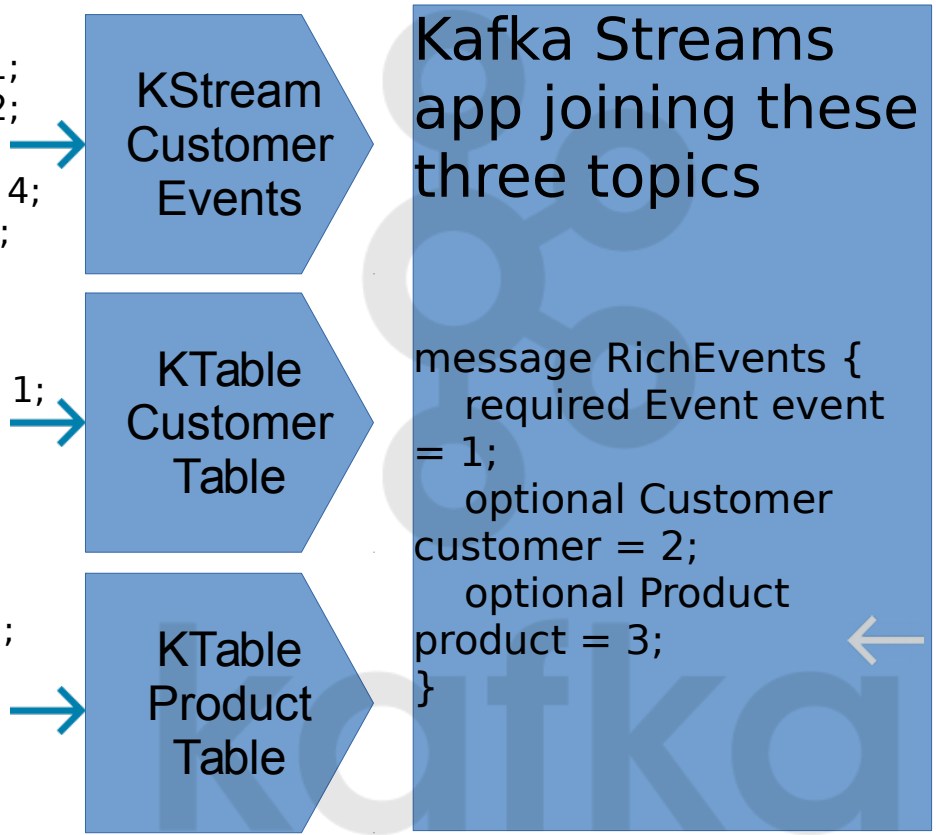


What if …?


Protobuf Definitions
**Message Producers**, e.g.
Logs,
Databases,
APIs,
External sources
kafka
Stream processors
**Consumers, e.g**
Databases,
Reports & Dashboards,
Applications,
Machine learning,
Big Data storage

# Adding a new field

```
message Event {
    required int64 timestamp = 1;
    required string session_id = 2;
    required string type = 3;
    optional int32 customer_id = 4;
    optional int32 product_id = 5;
}

message Customer {
    required int32 customer_id = 1;
    optional string name = 2;
    optional string email = 3;
}

message Product {
    required int32 product_id = 1;
    optional string name = 2;
    optional float price = 3;
    optional string colour = 4;
}
```



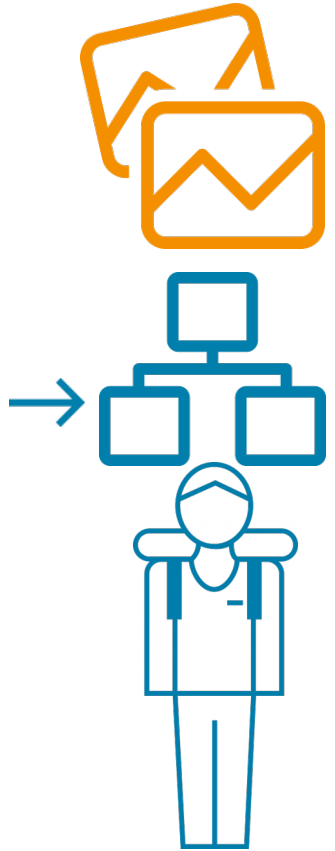
# Adding a new field

```
message RichEvent {
    required Event event = 1;
    optional Customer
customer = 2;
    optional Product product =
3;
}
```


Kafka Streams app collecting into sessions

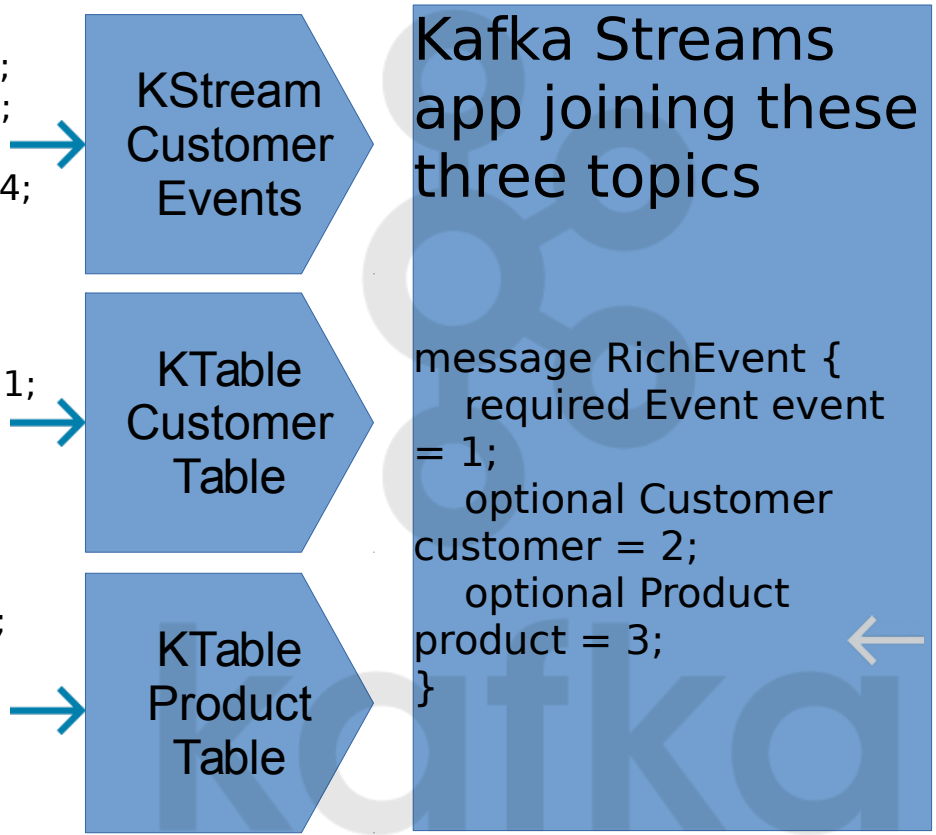
```
message Session {
    repeated RichEvent
richevent = 1;
}
```

**Contains Products with unknown Fields**



# Removing a field

```
message Event {
    required int64 timestamp = 1;
    required string session_id = 2;
    required string type = 3;
    optional int32 customer_id = 4;
    optional int32 product_id = 5;
}

message Customer {
    required int32 customer_id = 1;
    optional string name = 2;
    optional string email = 3;
}

message Product {
    required int32 product_id = 1;
    optional string name = 2;
    optional float price = 3;
}
```



# Summary

- Decouples development processes
  - Teams can move at their own speed
  - No strict alignment for releases necessary
- “hands-off” data engineering: Only actual producers and consumers need to align on new information, pipeline in between runs uninterrupted

# Downsides of google protobuf

- No dynamic schema generation
  - Make sure to stick to your field ids and don’t reuse them!
  - More consideration needed and “handcrafting” schemas
- Less implementations than avro around Kafka & Hadoop
- Also less users
- Google wants to remove unknown fields in Protobuf 3
  - https://github.com/google/protobuf/issues/272
- Slightly bigger than avro

trivago

# Thank you!

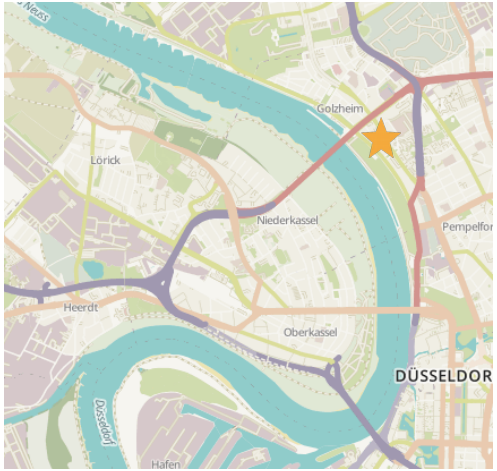
Questions and comments?


**Clemens Valiente**

Senior Data Engineer
trivago Düsseldorf


Originally a mathematician
Studied at Uni Erlangen
At trivago for 5 years


Email: clemens.valiente@trivago.com
de.linkedin.com/in/clemensvaliente

trivago

- Thanks to Jan Filipiak for his brainpower behind most projects, giving me the opportunity to present them
- Additional resources (trivago Open Source):
- https://github.com/trivago/gollum A n:m message multiplexer written in Go
- https://github.com/trivago/triava TriavaCache, JSR107 compliant cache