ksqlDB

Emanuele Della Valle

prof @ Politecnico di Milano

Why use ksqlDB?

Why use ksqlDB?

Too many cooks in the kitchen.

Why use ksqlDB?

Divide and conquer

Why use ksqlDB?
Cut from the same mold

Why use ksqlDB?

Reunited

Key operations

Key operations

A running example about riders

Key operations

Capture events

```
CREATE SOURCE CONNECTOR riders WITH (
  'connector.class' = 'JdbcSourceConnector',
  'connection.url'  = 'jdbc:postgresql://...',
  'topic.prefix'    = 'rider',
  'table.whitelist' = 'riderLocations, profiles',
  'key'             = 'profile_id',
  ...);
```

Key operations

Perform continuous transformations

```
CREATE STREAM locations AS
  SELECT rideId, latitude, longitude,
         GEO_DISTANCE(latitude, longitude,
                      dstLatitude, dstLongitude
                      ) AS kmToDst
  FROM ridersLocation
  EMIT CHANGES;
```

Enrichment

Key operations

Create materialized views

```
CREATE TABLE activePromotions AS
  SELECT rideId,
         qualifyPromotion(kmToDst) AS promotion
  FROM locations
  GROUP BY rideId
  EMIT CHANGES;
```

User Defined Aggs

Key operations

Serve lookups against materialized views

```
SELECT rideId, promotion
FROM activePromotions
WHERE rideId = '6fd0fcdb';
```

Concepts

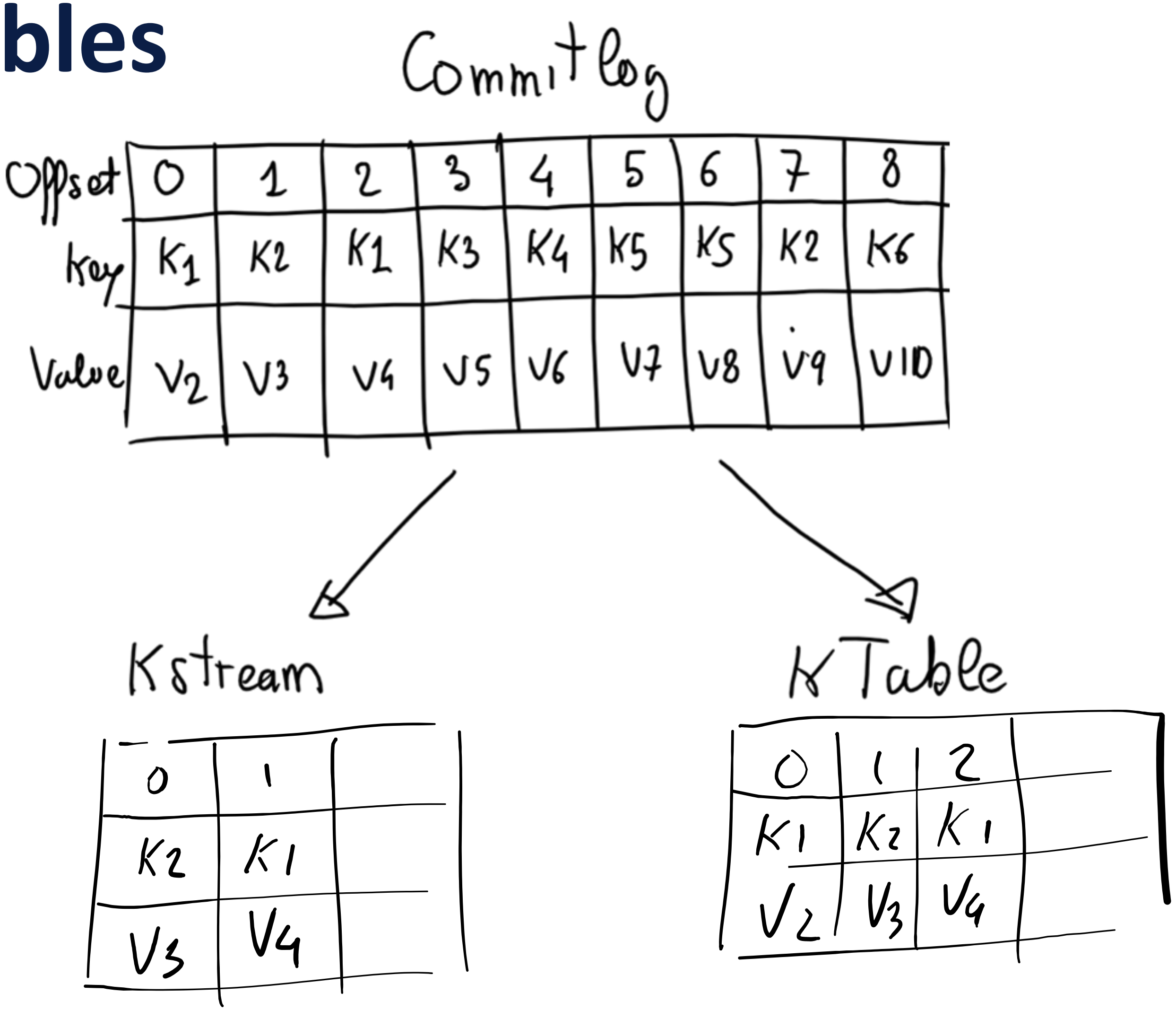
Concepts
Basics

- **Event**: the fundamental unit of data in stream processing
- **Stream**: an *immutable*, *append-only** collection of events that represents a series of historical facts
- **Table** (a.k.a. **materialized view**): a *mutable* collection of events. They let you represent the latest version of each value per key.
- **Queries**: how you *transform, filter, aggregate*, and *join* collections together to derive new collections or materialized views that are incrementally updated in real-time as new events arrive.

Concepts

Topics vs. Streams vs. Tables

- topics can have two origins in Kafka
  - *independent observations*
  - *change logs* captured from a DB
- *independent observations* can be captured in a **stream**
- *change logs* can be captured in a **stream** but can also in a time-varying **table** (a.k.a. **materialized views**)



Commit log

Offset	0	1	2	3	4	5	6	7	8
Key	K1	K2	K1	K3	K4	K5	K5	K2	K6
Value	V2	V3	V4	V5	V6	V7	V8	V9	V10

Kstream

0	1	
K2	K1	
V3	V4	

KTable

0	1	2	
K1	K2	K1	
V2	V3	V4	

Concepts
Push queries

- let you **subscribe** to a query's result as it changes in real-time
- When new events arrive, push queries emit refinements, which allow **reacting** to new information
- Fit for **asynchronous** application **flows**

```
SELECT riderId, latitude, longitude
FROM Locations
WHERE rider = '6fd0fcdb'
EMIT CHANGES;
```

Concepts
Pull queries

- **Fetch** the current state of a materialized view
- Incrementally updated as new events arrive
- fit for **request/response flows**

```
SELECT riderId, latitude, longitude
FROM currentCarLocations
WHERE ROWKEY = '6fd0fcdb';
```

How it works

How it works
ksqlDB Architecture

How it works
ksqlDB vs Kafka Streams 1/3

How it works
ksqlDB vs Kafka Streams 2/3

- For example, the following KSQL query ...

create at registication time

```
CREATE STREAM fraudulent_payments AS
SELECT fraudProbability(data)
FROM payments
WHERE fraudProbability(data) > 0.8
EMIT CHANGES;
```

filter

UDF

src

How it works
ksqlDB vs Kafka Streams 3/3

... is equivalent to the following Scala code

```
bject FraudFilteringApplication extends App {
  val builder: StreamsBuilder = new StreamsBuilder()
  val fraudulentPayments: KStream[String, Payment] = builder
    .stream[String, Payment]("payments")
    .filter((_ ,payment) => payment.fraudProbability > 0.8)
  fraudulentPayments.to("fraudulent-payments-topic")
  val config = new java.util.Properties
  config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")
  config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092")
  val streams: KafkaStreams = new KafkaStreams(builder.build(), config)
  streams.start()
}
```

UDF

How it works

Differences Between ksqlDB and Kafka Streams

What	ksqlDB	Kafka Streams
You write	~ SQL	scala / java
Console	✓	✗
REST API	✓	✗
Runtime	✓ KsqlDB Server	JVM

How it works

Deployment Modes

How it works

Fault tolerance and scale-out

Notes

- For each query there is a src topic (with its partitions, say 3)
- Each ksqlDB server creates a consumer referring to the consumer group of the query
- As far as there a partitions to assign, Kafka assign them to the consumers created by the ksqlDB Serves
  - Up to 3 in the example
- Any other consumer created by the ksqlDB Serves will remain in idle state waiting for a fault or for a scale out operation

How it works

Query Lifecycle

Walking through basic tutorials

Walking through basic tutorials

Materialized Views

The hard way

The ksqlDB way

Walking through basic tutorials

Streaming ETL

The hard way

The ksqlDB way

Learn m ore

- https://ksqldb.io/
- https://ksqldb.io/examples.html
- https://github.com/confluentinc/ksql
- https://twitter.com/ksqldb

Thank you for your attention!

Questions?

ksqlDB

Emanuele Della Valle

prof @ Politecnico di Milano