aiven

KSQL - Introduction

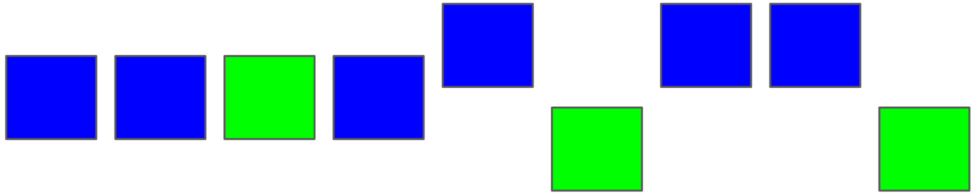
Heikki Nousiainen

2019-01-16

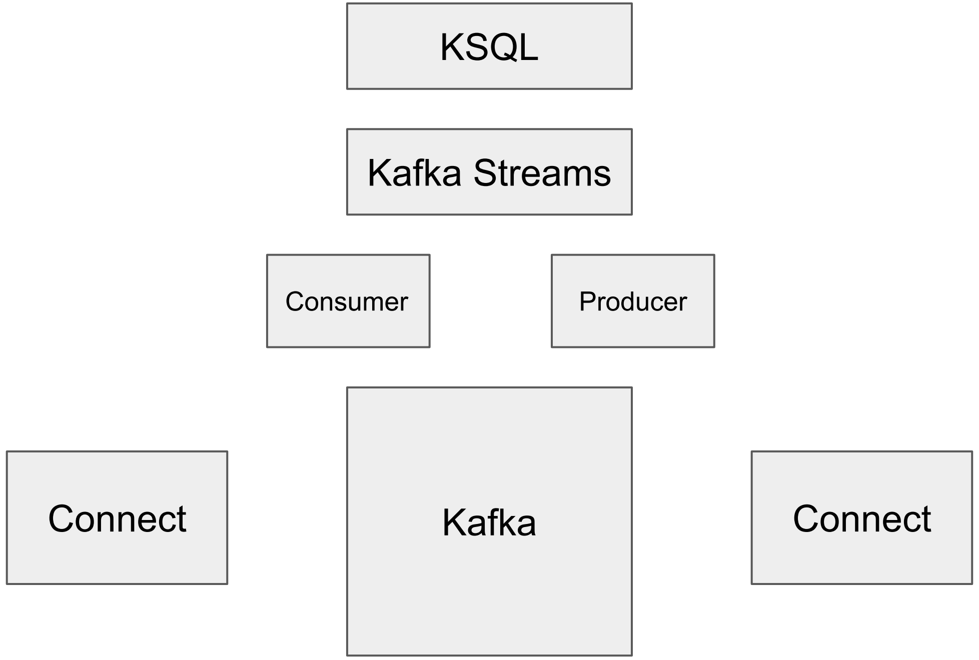
Stream Processing

- Unbounded, continuous stream of events
- Real-time vs. (micro-)batching
- Handle high volume & velocity data

- Filtering & Enrichment
- Validation & Transformation
- Dehydration & Hydration
- Routing, Splitting and Joining



Kafka Ecosystem



KSQL

- An engine for running streaming jobs on Kafka data
- Queries are defined in **SQL**-like syntax and run within **KSQL-server** instance
- Record at a time, real-time processing
- Input from and output to Kafka topics

- First release in August 2018, developing rapidly.

KSQL

- Queries and posted to KSQL server over REST interface
- KSQL commandline client

Why KSQL?

- Convenient, no need to write application code
- No dependency on a specific language runtime
- Continuous queries operate without needing to run / schedule application runtime
- Very fast development cycles

Why Streams / Consumer/Producer?

- KSQL might not offer the level of controls than Streams / Consumer/Producer
- KSQL has access to Kafka topic data only, e.g. other systems
- KSQL is somewhat tricky to extend with custom functions

- KSQL created topologies tend to use a lot of intermediate topics; data reshuffling may amplify network utilization quite a bit

KSQL basics

Define a **stream** for input:

```
CREATE STREAM pageviews
        (status INTEGER,
                url STRING,
                user STRING)
        WITH
        (KAFKA_TOPIC='pageviews',
        VALUE_FORMAT='JSON');
```

KSQL basics

Query a **stream**:

```
SELECT status, user, ‘staging_’ || url FROM pageviews
      WHERE status = 200;
```

Filter + transform

KSQL basics

Create a new **stream** & **topic**:

```
CREATE STREAM pageview_errors
        WITH
                (KAFKA_TOPIC = 'pageview_errors',
                VALUE_FORMAT='JSON')
        AS
                SELECT url, status, user
                        FROM pageviews
                        WHERE status <> 200
                        PARTITION BY url;
```

Background query

KSQL basics

Check what’s running:

```
SHOW STREAMS;

SHOW QUERIES;

DESCRIBE pageviews;
```

Streams & tables

Key	Value
a	blue
b	green
a	violet
b	yellow

Key	Value
a	violet
b	yellow

KSQL

Repartition:

```
CREATE STREAM pageviews_by_user
        AS
                SELECT user, url, status
                        FROM pageviews
                        PARTITION BY user;
```

KSQL

Aggregations:

```
CREATE TABLE pageviews_per_user
        WITH
                (KAFKA_TOPIC = 'pageviews_per_user',
                VALUE_FORMAT='JSON')
        AS
                SELECT user, count(*) as count
                        FROM pageviews
                        GROUP BY user;
```

Aggregation functions: max, min, count, ...

KSQL

Windowed aggregations:

```
CREATE TABLE pageviews_per_minute
        WITH
                (KAFKA_TOPIC = 'pageviews_per_minute',
                VALUE_FORMAT='JSON')
        AS
                SELECT url, count(*)
                        FROM pageviews
                        WINDOW TUMBLING (SIZE 1 MINUTE)
                        GROUP BY url;
```

Windowing: tumbling, hopping, session

KSQL

Joins:

```
CREATE STREAM pageviews_with_user_stats
      AS
            SELECT p.url, ppu.count
            FROM pageviews_by_user p
            LEFT JOIN pageviews_per_user ppu
                  ON p.user = ppu.user;
```

Joins: left, inner, outer

stream - stream (windowed), stream - table (lookup)

stream-stream: event correlation/attribution, stream-table: data enrichment

KSQL

Joins:

```
CREATE STREAM feedback_with_pages
        AS
                SELECT pbu.user, pbu.url, pbu.status
                FROM feedback_by_user fbu
                INNER JOIN pageviews_by_user pbu
                        WITHIN 1 MINUTE
                        ON fbu.user = pbu.user;
```

Joins: left, inner, outer

stream - stream (windowed), stream - table (lookup)

stream-stream: event correlation/attribution, stream-table: data enrichment

KSQL Benefits

- Quick to develop with - no code needed
- REST Can be called from any language
- Kafka Streams backs state & progress into Kafka
- Distributed operation: work distribution, fault recovery
- Scale by number of instances
- Trivially runs on cloud & containers

Links & resources

- https://docs.confluent.io/current/ksql/docs/tutorials/
- https://github.com/confluentinc/ksql
- https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html

Epilog

- KSQL license changes / Dec 2019: no longer Open Source
- We’re looking at Flink ourselves, and likely present our experiences in the coming meetups.