KSQL project

We have out docker-compose file, which has 4 containers.

```
---
version: '2'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.4.1
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-enterprise-kafka:5.4.1
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    ports:
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:9092,PLAINTEXT_HOST://localhost:29092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1

  ksqldb-server:
```

```
    image: confluentinc/ksqldb-server:0.8.1
    hostname: ksqldb-server
    container_name: ksqldb-server
    depends_on:
      - broker
    ports:
      - "8088:8088"
    environment:
      KSQL_LISTENERS: http://0.0.0.0:8088
      KSQL_BOOTSTRAP_SERVERS: broker:9092
      KSQL_KSQL_LOGGING_PROCESSING_STREAM_AUTO_CREATE: "true"
      KSQL_KSQL_LOGGING_PROCESSING_TOPIC_AUTO_CREATE: "true"

  ksqldb-cli:
    image: confluentinc/ksqldb-cli:0.8.1
    container_name: ksqldb-cli
    depends_on:
      - broker
      - ksqldb-server
    entrypoint: /bin/sh
    tty: true
```

Then we create a Topic in Conductor:

Conduktor

Kafka Desktop Client - Beautiful UI

https://www.conduktor.io/

Topic name: event_test

Then, in kotlin, we produce some messages to said topic:

```
import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerRecord
import org.apache.kafka.common.serialization.ByteArraySerializer
import org.apache.kafka.common.serialization.StringSerializer
import java.io.BufferedReader
import java.io.FileReader
import java.io.IOException
import java.util.*

fun main() {
    val properties = Properties()
    properties["bootstrap.servers"] = "localhost:29092"
    properties["key.serializer"] = StringSerializer::class.java
    properties["value.serializer"] = ByteArraySerializer::class.java
    val sampleProducer = Producer<ByteArray>()
    val reader: BufferedReader
    try {
        reader = BufferedReader(
            FileReader(
                "events.json"
            )
        )
        var line = reader.readLine()
```

```
        while (line != null) {
            sampleProducer.produce("event_test", properties,"event: ", line.toByteArray())
            line = reader.readLine()
        }
        reader.close()
    } catch (e: IOException) {
        e.printStackTrace()
    }
    //
}

class Producer<X> {
    fun produce(topicName:String, properties: Properties, key:String , value: X){
        val producerRecord = ProducerRecord(topicName, key, value)
        val kafkaProducer = KafkaProducer<String, X>(properties)
        kafkaProducer.send(producerRecord)
        kafkaProducer.close()
    }
}
```

Next, we connect to the ksql server:

```
docker exec -it ksqldb-cli ksql http://ksqldb-server:8088
```

First, we make the start to earliest

```
set 'auto.offset.reset' = 'earliest';
```

Next, we check if the data is there:

```
print 'event_test';
```

Now we create a stream that knows more about the data:

The Stream needs to know the schema of the json data

```
CREATE STREAM meetup_events_stream(
  utc_offset BIGINT,
  venue STRUCT<
    country VARCHAR,
    city VARCHAR,
    address_1 VARCHAR,
    name VARCHAR,
    lon DOUBLE,
    lat DOUBLE
  >,
  rsvp_limit INTEGER,
  venue_visibility VARCHAR,
  visibility VARCHAR,
  maybe_rsvp_count INTEGER,
  description VARCHAR,
```

```
  mtime BIGINT,
  event_url VARCHAR,
  yes_rsvp_count INTEGER,
  payment_required INTEGER,
  name VARCHAR,
  id VARCHAR,
  time BIGINT,
  "GROUP" STRUCT<
    join_mode VARCHAR,
    country VARCHAR,
    city VARCHAR,
    name VARCHAR,
    group_lon DOUBLE,
    id BIGINT,
    state VARCHAR,
    urlname VARCHAR,
    category STRUCT<
      name VARCHAR,
      id VARCHAR,
      shortname VARCHAR
    >,
    group_photo STRUCT<
      highres_link VARCHAR,
      photo_link VARCHAR,
      photo_id BIGINT,
      thumb_link VARCHAR
    >,
    group_lat DOUBLE
  >,
  status VARCHAR) WITH (KAFKA_TOPIC='event_test', VALUE_FORMAT='JSON');
```

Now we can run queries on the stream:

so our first stream should only the result, where the venue is in GB

Query:

```
SELECT name,"GROUP"->city AS city , "GROUP"->country AS country
FROM meetup_events_stream WHERE "GROUP"->country = 'de' EMIT CHANGES;
```

Stream:

```
CREATE STREAM meetup_events_stream_de AS SELECT name,"GROUP"->city
AS city , "GROUP"->country AS country FROM meetup_events_stream
WHERE "GROUP"->country = 'de';
```

And the second one only where the venue is in London:

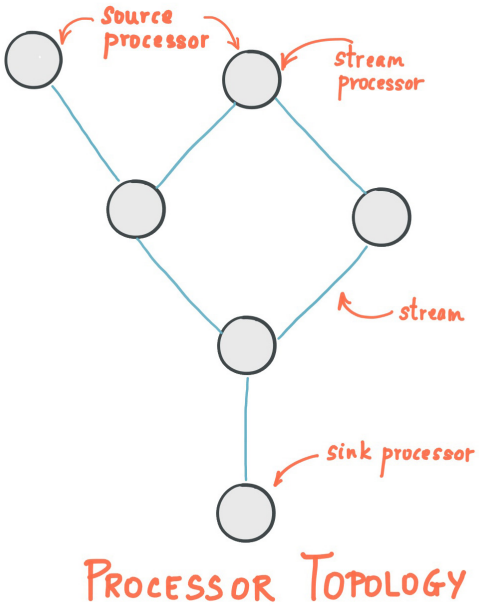
Query:

```
SELECT * FROM meetup_events_stream_de WHERE city = 'Munich' EMIT CHANGES;
```

Stream:

```
CREATE STREAM meetup_events_stream_munich AS SELECT name, city, country
FROM meetup_events_stream_de WHERE city = 'Munich';
```

So we basicly have a processor topology



Our Source is the kotlin producer, then we have two strean processors.

Our Sink can be the python Programm which just prints the query.

But it could also be a dashboard or an app.

Sources:

ksqlDB: The event streaming database purpose-built for stream processing applications.

Learn the basics The guide below demonstrates how to get a minimal ksqlDB environment up and running with the ksqlDB Docker image. When you're ready to learn more, take the next steps with use-case-driven tutorials.

https://ksqldb.io/quickstart.html

Documentation - Confluent Platform

Get Confluent | Sign up for Confluent Cloud or download Confluent Platform

https://docs.confluent.io/current/streams/architecture.html](https://docs.confluent.io/current/streams/architecture.html)