Streaming ETL with Apache Kafka

Szymon Chojnacki,
Enterprise Architect
LPP SA

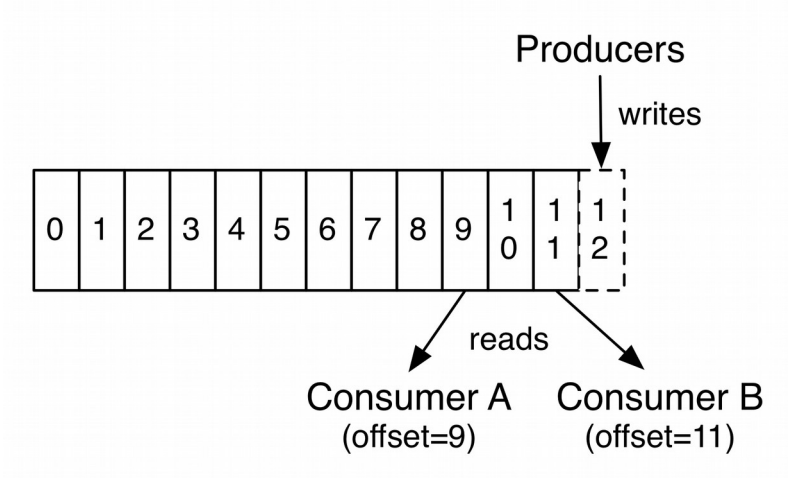
PyData Gdańsk 14.XI.2018

LPP

Agenda

1. **Apache Kafka**
2. **Stream processing model**
3. **ETL Design patterns**
4. **Demo of KSQL**

Apache Kafka is a distributed streaming platform

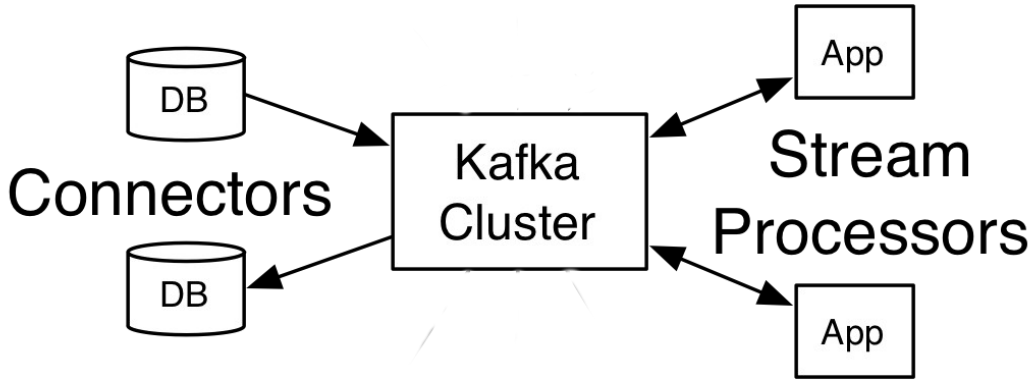


Used to build streaming data pipelines, real-time streaming applications.

Source: https://kafka.apache.org/intro

Stream processing model

1. **Event streams are ordered**
2. **Immutable data records**
3. **Event streams are replayable**



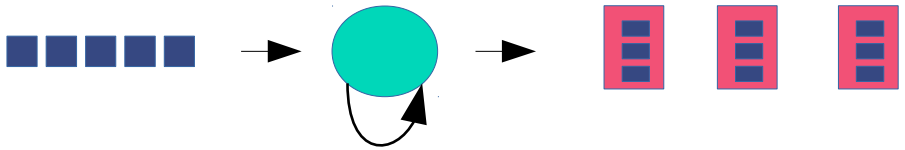
Other streaming platforms: **Spark, Flink, Storm, Samza, Pub/Sub, Kinesis**

Source: Kafka The Definitive Guide, Real-time data and stream processing at scale. Neha Narkhede, Gwen Shapira & Todd Palino

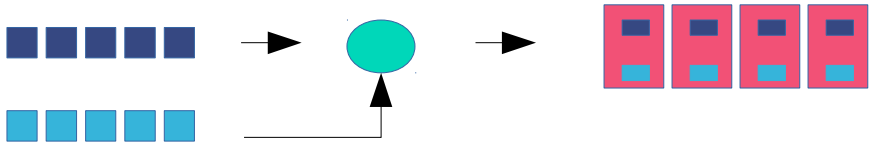
ETL Design patterns

1. **Single event processing**

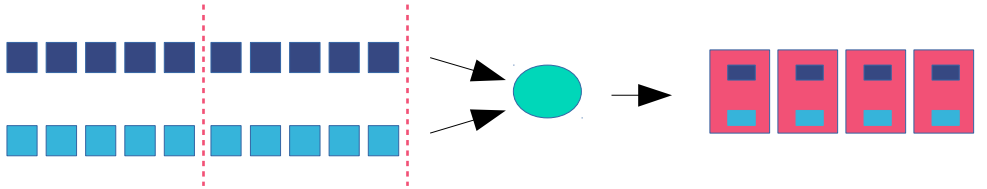
2. **Processing with local state**



3. **Stream - Table Join**



4. **Streaming Join**



5. **Insert Into**

Source: Kafka The Definitive Guide, Real-time data and stream processing at scale. Neha Narkhede, Gwen Shapira & Todd Palino

Start Confluent Platform CLI

```
pydata $ confluent start
This CLI is intended for development only, not for production
https://docs.confluent.io/current/cli/index.html

Using CONFLUENT_CURRENT: /tmp/confluent.Z1Lzp0jQ
Starting zookeeper
zookeeper is [UP]
Starting kafka
kafka is [UP]
Starting schema-registry
schema-registry is [UP]
Starting kafka-rest
kafka-rest is [UP]
Starting connect
connect is [UP]
Starting ksql-server
ksql-server is [UP]
Starting control-center
control-center is [UP]
pydata $
```

Note: In the demo we only use following components: zookeeper, broker and ksql server

Download from: https://www.confluent.io/download/

Generate users

```
pydata $ cat user.avro
{
        "namespace": "streams",
        "name": "user",
        "type": "record",
        "fields": [
                {"name": "userid", "type": {
                    "type": "long",
                    "arg.properties": {
                       "iteration": { "start": 1, "step": 1}
                    }
                }},
                {"name": "country", "type": {
                    "type": "string",
                    "arg.properties": {
                       "regex": "PL|IT|FR"
                    }
                }}
        ]
}
```

1

2

```
pydata $ ksql-datagen schema=user.avro \
> format=delimited \
> topic=user \
> key=userid \
> iterations=10 \
> maxInterval=1500
Outputting 10 to user
[2018-10-25 10:43:05,440] INFO AvroDataConfig values:
        schemas.cache.config = 1
        enhanced.avro.schema.support = false
        ne    eta.data = true
 (io.co   en   nnect.avro.AvroDataConfig:179)
1 --> ([ 1 | 'FR' ]) ts:1540456985740
2 --> ([ 2 | 'PL' ]) ts:1540456985761
3 --> ([ 3 | 'IT' ]) ts:1540456986156
4 --> ([ 4 | 'IT' ]) ts:1540456987140
5 --> ([ 5 | 'PL' ]) ts:1540456988354
6 --> ([ 6 | 'IT' ]) ts:1540456988801
7 --> ([ 7 | 'FR' ]) ts:1540456990119
8 --> ([ 8 | 'PL' ]) ts:1540456991613
9 --> ([ 9 | 'PL' ]) ts:1540456992184
10 --> ([ 10 | 'IT' ]) ts:1540456992190
pydata $
```

1

2

Generate purchases

```
pydata $ cat buy.avro
{
        "namespace": "streams",
        "name": "buy",
        "type": "record",
        "fields": [
                {"name": "buyid", "type": {
                    "type": "long",
                    "arg.properties": {
                       "iteration": { "start": 1, "step": 1}
                    }
                }},
                {"name": "userid", "type": {
                    "type": "string",
                    "arg.properties": {
                        "regex": "[1-5]"
                    }
                }},
                {"name": "amount", "type": {
                    "type": "string",
                    "arg.properties": {
                        "regex": "[1-9][0-9]\\.[0-9][0-9]"
                    }
                }}

        ]
}
```

1 2 3

```
pydata $ ksql-datagen schema=buy.avro format=delimited topic=buy
key=buyid iterations=5 maxInterval=1500
Outputting 5 to buy
[2018-10-25 10:50:46,751] INFO AvroDataConfig values:
        schemas.cache.config = 1
        enhanced.avro.schema.support = false
        [illegible] = true
 (io.confluent.connect.avro.AvroDataConfig:179)
1 --> ([ 1 | '5' | '83.93' ]) ts:1540457447046
2 --> ([ 2 | '3' | '39.94' ]) ts:1540457447250
3 --> ([ 3 | '5' | '76.70' ]) ts:1540457448600
4 --> ([ 4 | '3' | '91.59' ]) ts:1540457449805
5 --> ([ 5 | '4' | '46.13' ]) ts:1540457451195
pydata $
```

Start KSQL Client

```
pydata $ ksql

            ===========================================
            =        _  __ _____  ____  _             =
            =       | |/ // ____|/ __ \| |            =
            =       | ' /| (___ | |  | | |            =
            =       |  <  \___ \| |  | | |            =
            =       | . \ ____) | |__| | |____        =
            =       |_|\_\_____/ \___\_\______|       =
            =                                         =
            =  Streaming SQL Engine for Apache Kafka® =
            ===========================================

Copyright 2017-2018 Confluent Inc.

CLI v5.0.0, Server v5.0.0 located at http://localhost:8088

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql> SET 'auto.offset.reset'='earliest';
Successfully changed local property 'auto.offset.reset' from 'null' to 'earliest'
ksql> SET 'ksql.sink.partitions'='1';
ksql>
ksql>
ksql> show topics;
```

KSQL client's binary is in <confluent-path>/bin

Step 1/7 – Create KStreams

Users stream with 10 events

```
ksql> CREATE STREAM user (userid BIGINT, country VARCHAR) \
WITH (kafka_topic='user', value_format='delimited');

 Message
----------------
 Stream created
----------------
ksql>
ksql> select * from user;
1540456985740 | 1 | 1 | FR
1540456985761 | 2 | 2 | PL
1540456986156 | 3 | 3 | IT
1540456987140 | 4 | 4 | IT
1540456988354 | 5 | 5 | PL
1540456988801 | 6 | 6 | IT
1540456990119 | 7 | 7 | FR
1540456991613 | 8 | 8 | PL
1540456992184 | 9 | 9 | PL
1540456992190 | 10 | 10 | IT
^CQuery terminated
```

Transactions stream with 5 events

```
ksql> CREATE STREAM buy (buyid BIGINT, userid VARCHAR, amount VARCHAR) \
WITH (kafka_topic='buy', value_format='delimited');

 Message
----------------
 Stream created
----------------
ksql>
ksql> select * from buy;
1540457447046 | 1 | 1 | 5 | 83.93
1540457447250 | 2 | 2 | 3 | 39.94
1540457448600 | 3 | 3 | 5 | 76.70
1540457449805 | 4 | 4 | 3 | 91.59
1540457451195 | 5 | 5 | 4 | 46.13
^CQuery terminated
```

Step 2/7 – Join Stream-Stream

Join gives output for 1 hours window

```
ksql> select buy.userid, amount, country FROM buy JOIN user \
WITHIN 1 HOURS ON buy.userid=user.userid;
5 | 83.93 | PL
3 | 39.94 | IT
5 | 76.70 | PL
3 | 91.59 | IT
4 | 46.13 | IT
^CQuery terminated
```

There is no output for 1 minutes window

```
ksql> select buy.userid, amount, country FROM buy JOIN user \
WITHIN 1 MINUTES ON buy.userid=user.userid;
... no output ...
```

Step 3/7 – Join Stream-Stream more data

1. Generate more transactions

```
pydata $ ksql-datagen schema=buy.avro format=delimited \
topic=buy key=buyid iterations=5
...
1 --> ([ 1 | '4' | '32.36' ]) ts:1540469855275
2 --> ([ 2 | '1' | '66.39' ]) ts:1540469855527
3 --> ([ 3 | '1' | '71.47' ]) ts:1540469856260
4 --> ([ 4 | '2' | '16.68' ]) ts:1540469857181
5 --> ([ 5 | '4' | '74.28' ]) ts:1540469857523
```

2. Still no output

```
ksql> select buy.userid, amount, country FROM buy JOIN user \
WITHIN 1 MINUTES ON buy.userid=user.userid;
... no output ...
```

3. Generate more users

```
pydata $ ksql-datagen schema=user.avro format=delimited \
topic=user key=userid iterations=10 maxInterval=1500
...
1 --> ([ 1 | 'IT' ]) ts:1540469872219
2 --> ([ 2 | 'PL' ]) ts:1540469872594
3 --> ([ 3 | 'IT' ]) ts:1540469872883
4 --> ([ 4 | 'IT' ]) ts:1540469873202
5 --> ([ 5 | 'FR' ]) ts:1540469874481
6 --> ([ 6 | 'IT' ]) ts:1540469875786
7 --> ([ 7 | 'PL' ]) ts:1540469876256
8 --> ([ 8 | 'PL' ]) ts:1540469877652
9 --> ([ 9 | 'PL' ]) ts:1540469878735
10 --> ([ 10 | 'FR' ]) ts:1540469879006
```

```
ksql> select buy.userid, amount, country FROM \
buy JOIN user WITHIN 1 MINUTES ON buy.userid=user.userid;
1 | 66.39 | IT
1 | 71.47 | IT
2 | 16.68 | PL
4 | 32.36 | IT
4 | 74.28 | IT
^CQuery terminated
```

Step 4/7 – Create KTable

Content of user stream

```
ksql> select * from user;
1540456985740 | 1 | 1 | FR
1540456985761 | 2 | 2 | PL
1540456986156 | 3 | 3 | IT
1540456987140 | 4 | 4 | IT
1540456988354 | 5 | 5 | PL
1540456988801 | 6 | 6 | IT
1540456990119 | 7 | 7 | FR
1540456991613 | 8 | 8 | PL
1540456992184 | 9 | 9 | PL
1540456992190 | 10 | 10 | IT
1540469872219 | 1 | 1 | IT
1540469872594 | 2 | 2 | PL
1540469872883 | 3 | 3 | IT
1540469873202 | 4 | 4 | IT
1540469874481 | 5 | 5 | FR
1540469875786 | 6 | 6 | IT
1540469876256 | 7 | 7 | PL
1540469877652 | 8 | 8 | PL
1540469878735 | 9 | 9 | PL
1540469879006 | 10 | 10 | FR
^CQuery terminated
ksql>
```

Create KTable user

```
ksql> CREATE TABLE table_user (userid BIGINT, country VARCHAR) \
WITH (kafka_topic='user', value_format='delimited', key='userid');
```

Content of KTable

```
ksql> select * from table_user;
1540469872219 | 1 | 1 | IT
1540469872594 | 2 | 2 | PL
1540469872883 | 3 | 3 | IT
1540469873202 | 4 | 4 | IT
1540469874481 | 5 | 5 | FR
1540469875786 | 6 | 6 | IT
1540469876256 | 7 | 7 | PL
1540469877652 | 8 | 8 | PL
1540469878735 | 9 | 9 | PL
1540469879006 | 10 | 10 | FR
^CQuery terminated
```

Step 5/7 – Join Stream with KTable

New transaction is joined with most recent record from KTable.
No need for windowing.

```
ksql> create stream buyuser as select buy.userid, amount, country \
FROM buy JOIN table_user ON buy.userid=table_user.userid;
...
ksql> select * from buyuser limit 10;
1540457447046 | 5 | 5 | 83.93 | FR
1540457447250 | 3 | 3 | 39.94 | IT
1540457448600 | 5 | 5 | 76.70 | FR
1540457449805 | 3 | 3 | 91.59 | IT
1540457451195 | 4 | 4 | 46.13 | IT
1540469855275 | 4 | 4 | 32.36 | IT
1540469855527 | 1 | 1 | 66.39 | IT
1540469856260 | 1 | 1 | 71.47 | IT
1540469857181 | 2 | 2 | 16.68 | PL
1540469857523 | 4 | 4 | 74.28 | IT
Limit Reached
Query terminated
ksql>
```

Step 6/7 Aggregate by country

Group by statement

```
ksql> select country, SUM(CAST(amount AS DOUBLE)) as suma \
FROM buyuser GROUP BY country;
FR | 160.63
PL | 16.68
IT | 422.15999999999997
```

Generate one more transaction

```
pydata $ ksql-datagen schema=buy.avro format=delimited topic=buy \
key=buyid iterations=1
...
1 --> ([ 1 | '5' | '53.20' ]) ts:1540470833300
pydata $
```

Stream gets updated

```
ksql> select country, SUM(CAST(amount AS DOUBLE)) as suma \
FROM buyuser GROUP BY country;
FR | 160.63
PL | 16.68
IT | 422.15999999999997
FR | 213.82999999999998
^CQuery terminated
```

Step 7/7 Aggregate with windows

Flow of transactions

```
pydata $ ksql-datagen schema=buy.avro \
format=delimited topic=buy key=buyid
...
1 --> ([ 1 | '1' | '47.13' ]) ts:1540470971470
2 --> ([ 2 | '2' | '89.83' ]) ts:1540470971618
3 --> ([ 3 | '4' | '48.36' ]) ts:1540470971945
4 --> ([ 4 | '2' | '11.23' ]) ts:1540470972323
5 --> ([ 5 | '3' | '59.50' ]) ts:1540470972353
6 --> ([ 6 | '1' | '36.31' ]) ts:1540470972496
7 --> ([ 7 | '1' | '97.36' ]) ts:1540470972789
8 --> ([ 8 | '1' | '90.59' ]) ts:1540470972928
9 --> ([ 9 | '4' | '40.31' ]) ts:1540470973071
10 --> ([ 10 | '2' | '81.61' ]) ts:1540470973424
11 --> ([ 11 | '3' | '31.67' ]) ts:1540470973586
12 --> ([ 12 | '4' | '96.19' ]) ts:1540470973712
13 --> ([ 13 | '2' | '10.06' ]) ts:1540470973877
14 --> ([ 14 | '3' | '62.00' ]) ts:1540470974196
15 --> ([ 15 | '5' | '53.12' ]) ts:1540470974337
16 --> ([ 16 | '5' | '11.27' ]) ts:1540470974537
```

Flow of aggregates

```
ksql> select country, SUM(CAST(amount AS DOUBLE)) as suma \
FROM buyuser \
WINDOW TUMBLING (SIZE 10 SECONDS) GROUP BY country;
FR | 160.63
IT | 131.53
IT | 46.13
PL | 16.68
IT | 244.5
FR | 53.2
IT | 47.13
PL | 89.83
PL | 182.67000000000002
IT | 547.4200000000001
PL | 246.91000000000003
IT | 710.6600000000001
FR | 84.28999999999999
PL | 309.06
FR | 164.76
IT | 823.28
FR | 267.44
IT | 1106.3200000000002
```

LPP RESERVED CROPP house MOHITO sinsay