CLOUDERA

Streaming Analytics with FlinkSQL

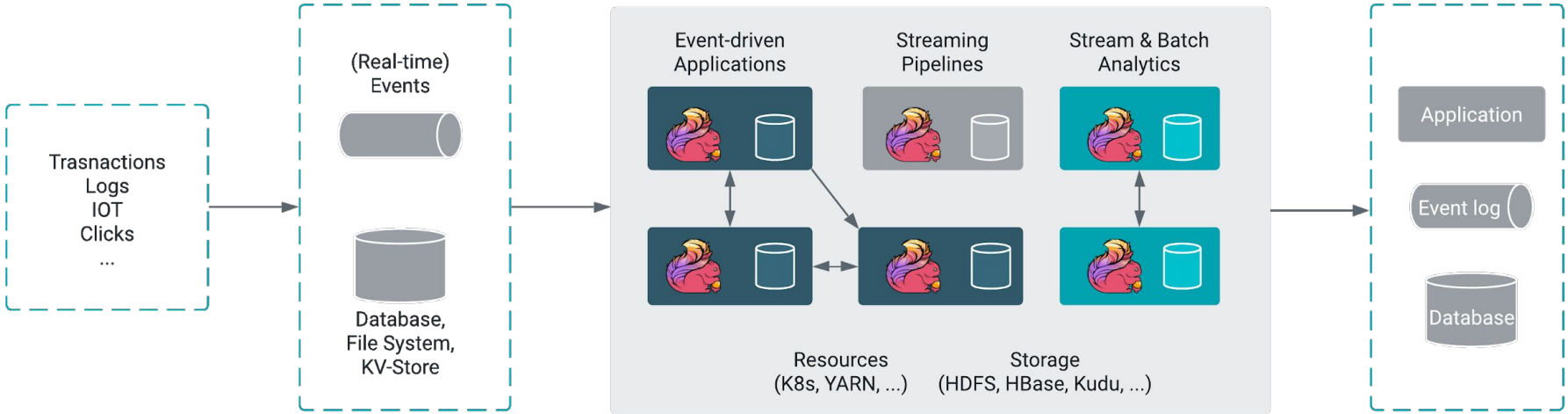
Marton Balassi
Engineering Lead, Streaming Analytics

About me

@MartonBalassi, mbalassi@cloudera.com

- My claim to fame is that I have written the first line of code of the Apache Flink Streaming API with Gyula Fora in 2013
- Apache Flink PMC member since 2014
- Worked for Ververica (data Artisans)
- Spent 3 years at Cloudera as a (Senior) Solutions Architect, worked with ~50 customers
- Currently leading the Streaming Analytics (Apache Flink) Engineering team at Cloudera

Flink is a Distributed Data Processing System

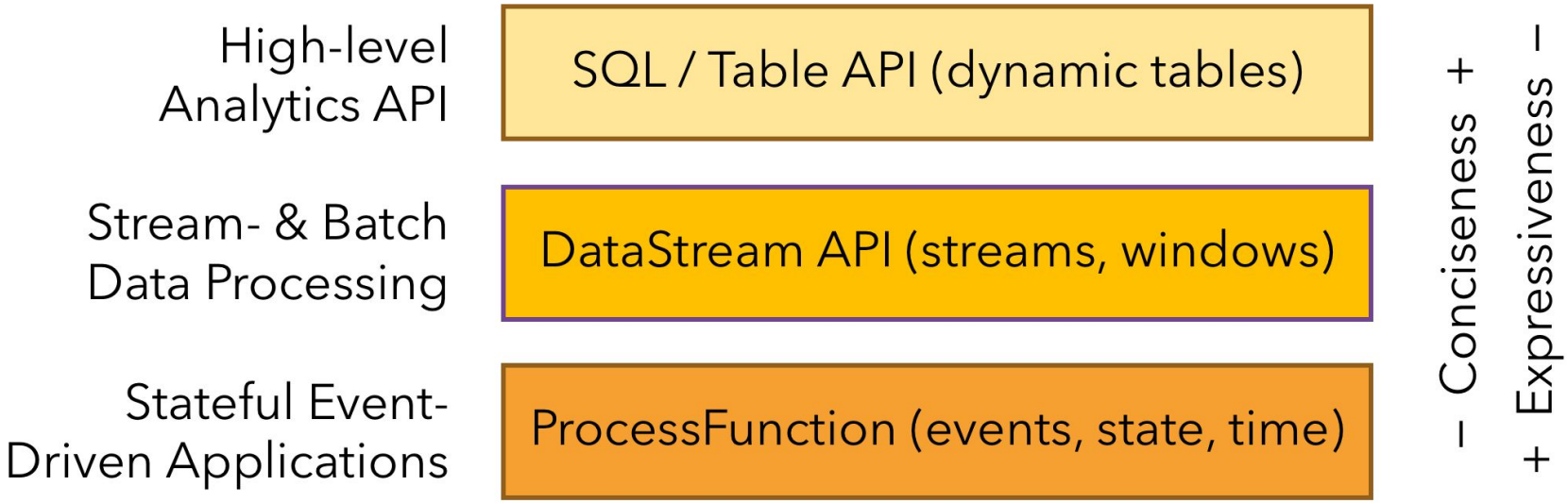


Consistency, Scale, Ecosystem

- Flexible and expressive APIs
- Guaranteed correctness
  - Exactly-once state consistency
  - Event-time semantics
- In-memory processing at massive scale
  - Runs on 100000s of cores
  - Manages 100s TBs of state
- Flexible deployments and large ecosystem
  - Kubernetes, YARN, Docker, HDFS, Kafka, HBase, Kudu, S3, Kinesis...

APIs

Layered APIs



SQL & Table API

- Unified APIs for streaming data and data at rest
  - Run the same query on batch and streaming data
  - ANSI SQL: No stream-specific syntax or semantics!
  - Many common stream analytics use cases supported

Count clicks per user and session (defined by 30 min. gap of inactivity).

```
SELECT
  userId,
  COUNT(*) AS cnt
  SESSION_START(clicktime, INTERVAL '30' MINUTE)
FROM clicks
GROUP BY
  SESSION(clicktime, INTERVAL '30' MINUTE),
  userId
```

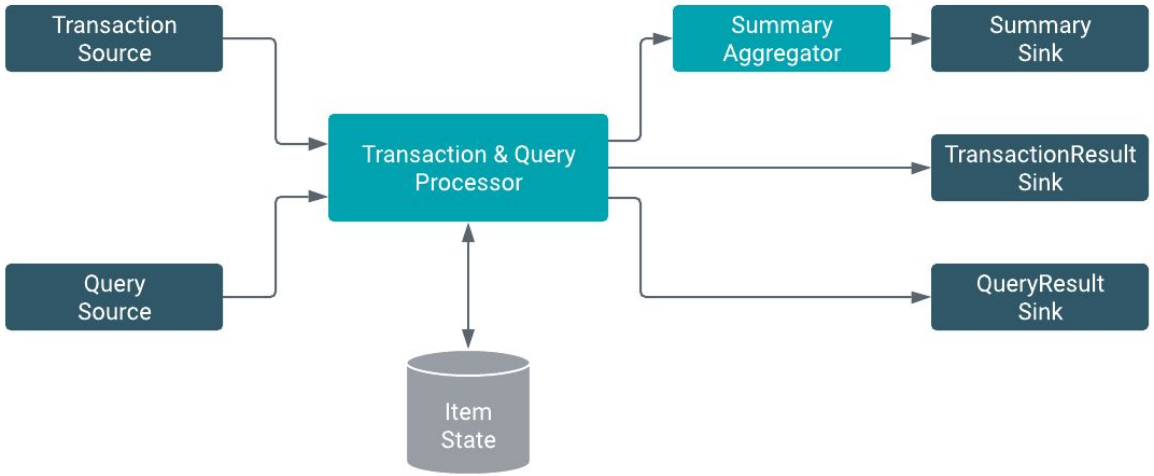
DataStream API

```
// a stream of website clicks
DataStream<Click> clicks = ...

DataStream<Tuple2<String, Long>> result = clicks
  // project clicks to userId and add a 1 for counting
  .map(
    // define function by implementing the MapFunction interface.
    new MapFunction<Click, Tuple2<String, Long>>() {
      @Override
      public Tuple2<String, Long> map(Click click) {
        return Tuple2.of(click.userId, 1L);
      }
    })
  // key by userId (field 0)
  .keyBy(0)
  // define session window with 30 minute gap
  .window(EventTimeSessionWindows.withGap(Time.minutes(30L)))
  // count clicks per session. Define function as lambda function.
  .reduce((a, b) -> Tuple2.of(a.f0, a.f1 + b.f1));
```

Count clicks per user and session (defined by 30 min. gap of inactivity).
Same as the previous SQL query.

Demo

Cloudera Flink Tutorials



https://github.com/cloudera/flink-tutorials

Next steps

Next steps for FlinkSQL

At least the ones I am most excited about :-)

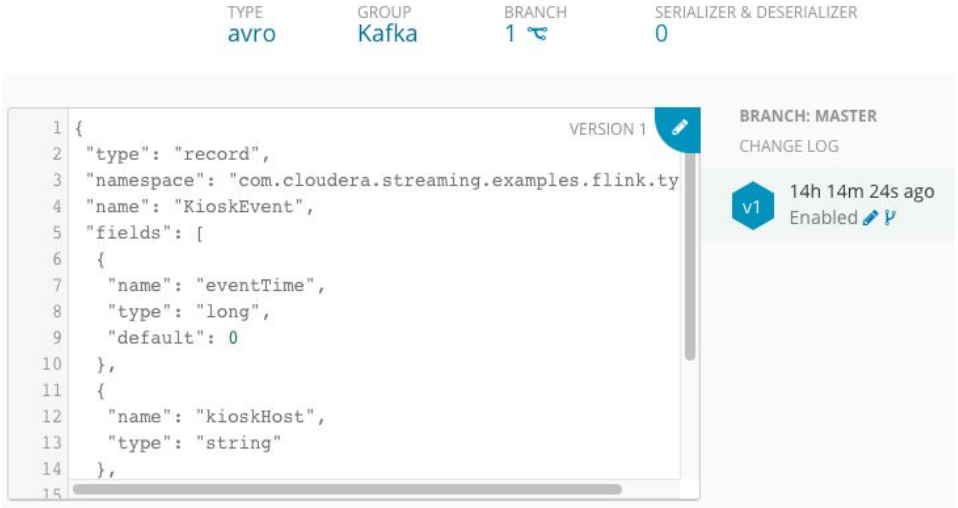
- REST endpoint (Logical equivalent for JDBC in streaming)
- Visual editor
- Unified catalog integration
  - Hive
  - Schema Registry
  - ...

Road to StreamSQL

Democratizing Stream Processing

Let us compute the number of errors per host for each 10 minute window

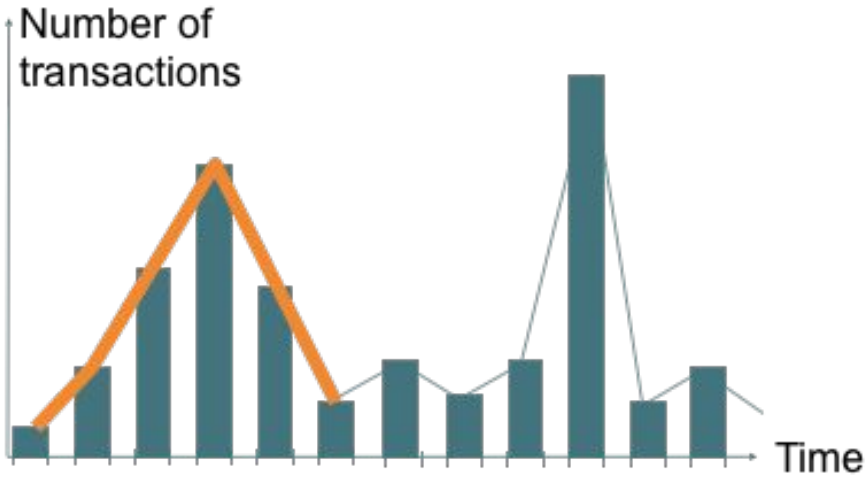
```
USE CATALOG REGISTRY;

SELECT
  TUMBLE_END(eventTime, INTERVAL '10' MINUTES)
    AS windowEnd,
  kioskHost,
  COUNT(*) AS numErrors,
  FIRST_VALUE(event) AS sample
FROM
  kiosk_events_raw_sfo
WHERE
  event LIKE '%ERROR%'
GROUP BY
  kioskHost,
  TUMBLE_END(eventTime, INTERVAL '10' MINUTES);
```



SQL API extensions

Let's detect peak transaction times in each area via Flink StreamSQL for our ATMs.

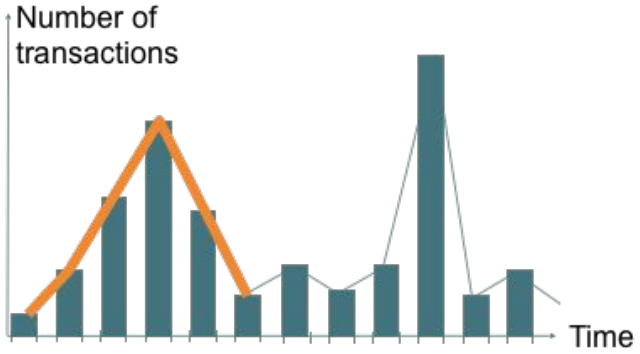
```
table Transactions

transactionId:          BIGINT
atmId:                  BIGINT
lon:                    DOUBLE
lat:                    DOUBLE
transactionTime:        TIMESTAMP
```



Helper View: Statistics per Area

```
CREATE VIEW TransactionsInArea AS
SELECT
    toAreaId(lat, lon) AS area,
    COUNT(DISTINCT transactionId) AS transactionCount,
    TUMBLE_ROWTIME(transactionTime, INTERVAL '30' MINUTE) AS rowTime,
    TUMBLE_START(transactionTime, INTERVAL '30' MINUTE) AS startTime,
    TUMBLE_END(transactionTime, INTERVAL '30' MINUTE) AS endTime
FROM
    Transactions
GROUP BY
    toAreaId(lat, lon),
    TUMBLE(transactionTime, INTERVAL '30' MINUTE)
```

Detect Peak Hours - V shape




```
SELECT * FROM TransactionsInArea MATCH_RECOGNIZE(
    PARTITION BY area ORDER BY rowTime
    MEASURES
        FIRST(UP.startTime) as peakStart,
        LAST(DOWN.endTime) AS peakEnd,
        SUM(UP.transactionCount) + SUM(DOWN.transactionCount) AS transactionSum
    AFTER MATCH SKIP PAST LAST ROW
    PATTERN (UP{4,} DOWN{2,} E)
    DEFINE
        UP AS UP.transactionCount > LAST(UP.transactionCount, 1) OR
            LAST(UP.transactionCount, 1) IS NULL,
        DOWN AS DOWN.transactionCount < LAST(DOWN.transactionCount, 1) OR
            LAST(DOWN.transactionCount, 1) IS NULL,
        E AS E.rideCount > LAST(DOWN.transactionCount)
)
```