Select Star:

Flink SQL for Pulsar Folks

Marta Paes (@morsapaes)
Developer Advocate

ververica

Why Pulsar + Flink?

😨

J. Doe • 1st 2w •••

Whenever I see #ApacheFlink mentioned my brain builds an image of #ApacheSpark. After all these years under influence of Spark, that's not a surprise. Same with #ApachePulsar and #ApacheKafka (after years with Kafka). That made me think if I'm the only one who considers Flink x Pulsar combo an attempt to "dethrone" Spark x Kafka? And they are all from The Apache Software Foundation and fully open source! ✊

Like | Reply

Why Pulsar + Flink?

Why Pulsar + Flink?

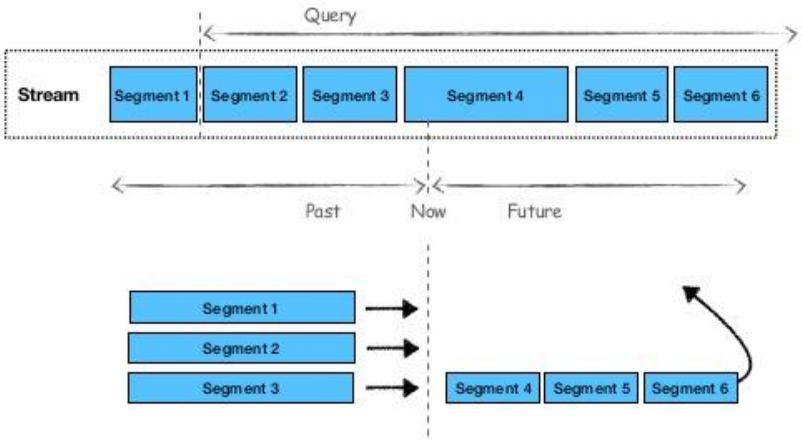
"Batch as a special case of streaming"

"Stream as a unified view on data"

Pulsar: Unified Storage

"Stream as a unified view on data"

- Pub/Sub messaging layer (Streaming)
- Durable storage layer (Batch)



Credit: StreamNative

Unified Storage
(Segments / Pub/Sub)

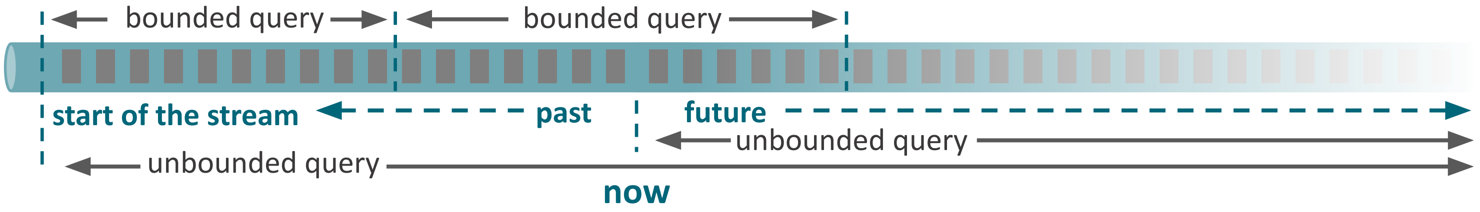
Learn more: Apache Pulsar as One Storage System for Both Real-time and Historical Data Analysis

Flink: Unified Processing Engine

"Batch as a special case of streaming"

- Reuse code and logic across batch and stream processing
- Ensure consistent semantics between processing modes
- Simplify operations
- Power applications mixing historic and real-time data

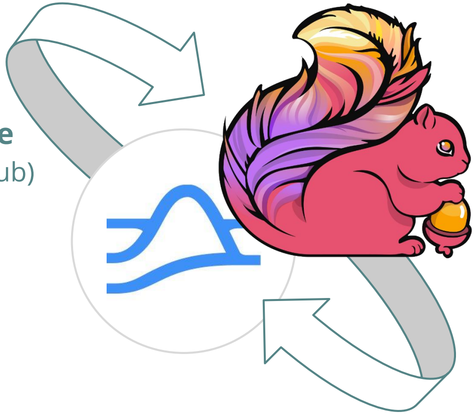
Unified Processing Engine
(Batch / Streaming)



Learn more: [Flink Ahead: What Comes After Unified Batch and Streaming?]

A Unified Data Stack

Unified Storage
(Segments / Pub/Sub)



Unified Processing Engine
(Batch / Streaming)

Flink is broad

Streaming Analytics & ML

SQL, PyFlink, Tables

Stateful Stream Processing

Streams, State, Time

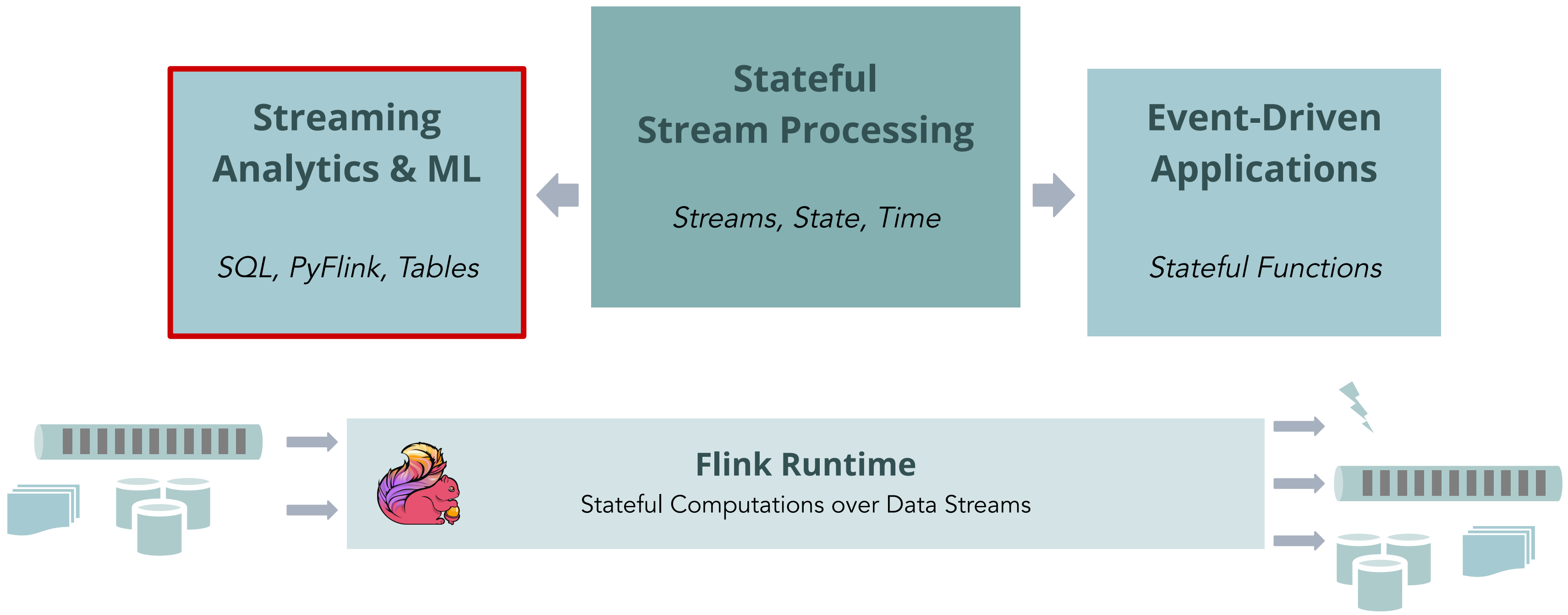
Event-Driven Applications

Stateful Functions

Flink Runtime

Stateful Computations over Data Streams

Flink is broad



Use Cases

More high-level or domain-specific use cases can be modeled with SQL/Python and dynamic tables.

- Focus on business logic, not implementation
- Mixed workloads (batch and streaming)
- Maximize **developer speed** and **autonomy**

Examples

	Uber	criteo
Unified Online/Offline Model Training	E2E Streaming Analytics Pipelines	ML Feature Generation

Flink SQL

"Everyone knows SQL, right?"

```
SELECT user_id, COUNT(url) AS cnt
FROM clicks
GROUP BY user_id;
```

This is standard SQL (ANSI SQL)

Flink SQL

"Everyone knows SQL, right?"

```
SELECT user_id, COUNT(url) AS cnt
FROM clicks
GROUP BY user_id;
```

This is ~~standard SQL (ANSI SQL)~~

also Flink SQL

A Regular SQL Engine

Take a snapshot when the query starts

user	cTime	url
Mary	12:00:00	https://...
Bob	12:00:00	https://...
Mary	12:00:02	https://...
Liz	12:00:03	https://...

A row that was added after the query was started is not considered

```
SELECT user_id,
       COUNT(url) AS cnt
FROM clicks
GROUP BY user_id;
```

The query terminates

A final result is produced

user	cnt
Mary	2
Bob	1

A Streaming SQL Engine

Ingest all changes as they happen

Continuously update the result

user	cTime	url
Mary	12:00:00	https://...
Bob	12:00:00	https://...
Mary	12:00:02	https://...
Liz	12:00:03	https://...

```sql
SELECT user_id,
       COUNT(url) AS cnt
FROM clicks
GROUP BY user_id;
```

user	cnt
Mary	2
Bob	1
Liz	1

The result is identical to the one-time query (at this point)

Flink SQL in a Nutshell

- Standard SQL syntax and semantics (i.e. not a “SQL-flavor”)
- Unified APIs for batch and streaming
- Support for **advanced time handling** and **operations** (e.g. CDC, pattern matching)

Execution

Streaming | **Batch**

TPC-DS Coverage

UDF Support

Java | **Python**

Scala

Native Connectors

Apache Kafka

FileSystems

Kinesis

JDBC

HBase

Elasticsearch

+

Formats

Debezium

JSON

CSV

Apache orc

Parquet

AVRO

Data Catalogs

HIVE Metastore

Postgres (JDBC)

For an overview of supported operations, check the Flink documentation: [Table API&SQL / SQL / Queries]

Pulsar Integration Over Time

Streaming Source/Sink Connectors

Table Sink Connector

Flink 1.6+

2018

Read more on Pulsar support in Flink 1.6+: [When Flink & Pulsar Come Together]

Pulsar Integration Over Time

Streaming Source/Sink Connectors

Table Sink Connector

Flink 1.9+

Flink 1.6+
2018

Pulsar Schema + Flink Catalog Integration

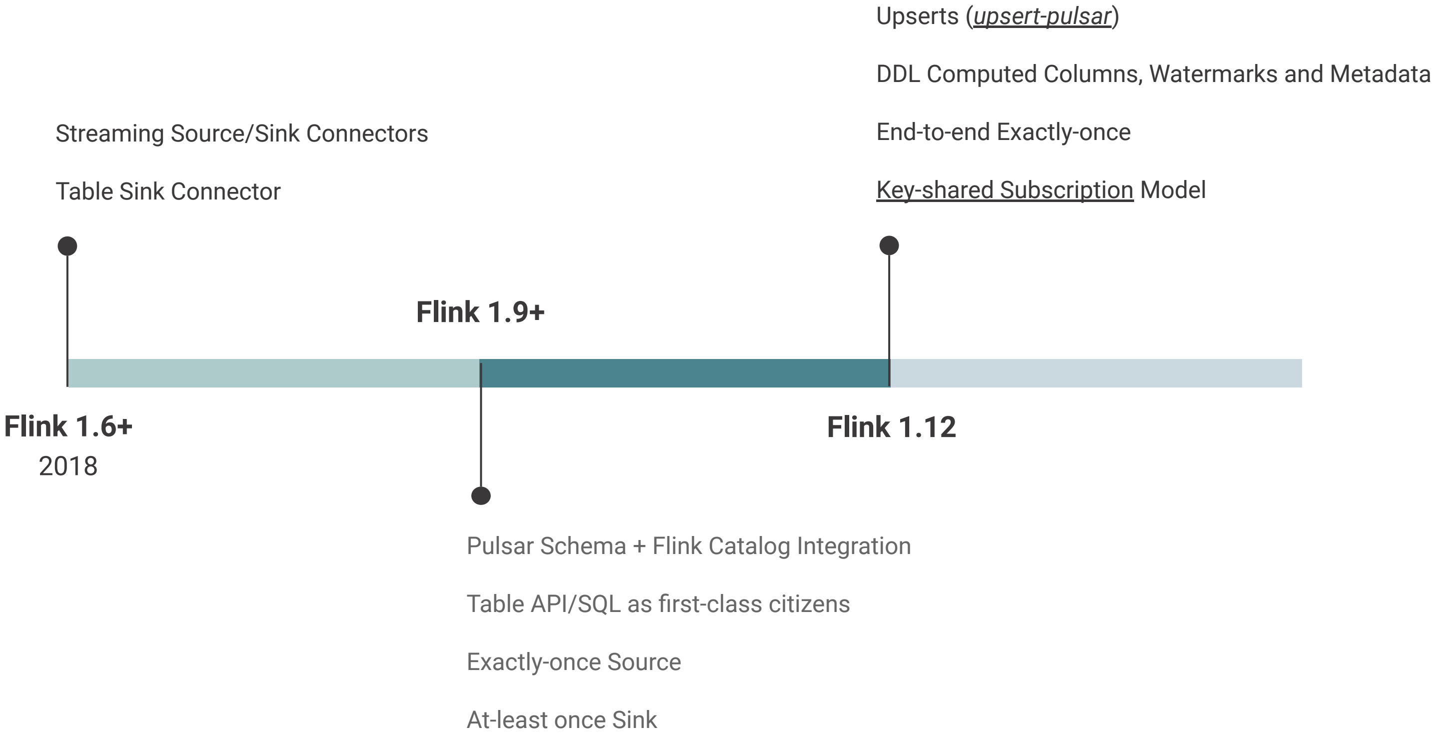
Table API/SQL as first-class citizens

Exactly-once Source

At-least once Sink

Read more on Pulsar support in Flink 1.9+: [How to Query Pulsar Streams using Apache Flink]

Pulsar Integration Over Time



Read more on Pulsar support in Flink 1.12+: What's New in the Pulsar Flink Connector 2.7.0?

Pulsar Integration Over Time

Flink 1.6+
2018

Streaming Source/Sink Connectors

Table Sink Connector

Flink 1.9+

Pulsar Schema + Flink Catalog Integration

Table API/SQL as first-class citizens

Exactly-once Source

At-least once Sink

Flink 1.12

Upserts (*upsert-pulsar*)

DDL Computed Columns, Watermarks and Metadata

End-to-end Exactly-once

Key-shared Subscription Model

Flink 1.13 *
Apr/May'21

Contribution to the Apache Flink repository [FLINK-20726]

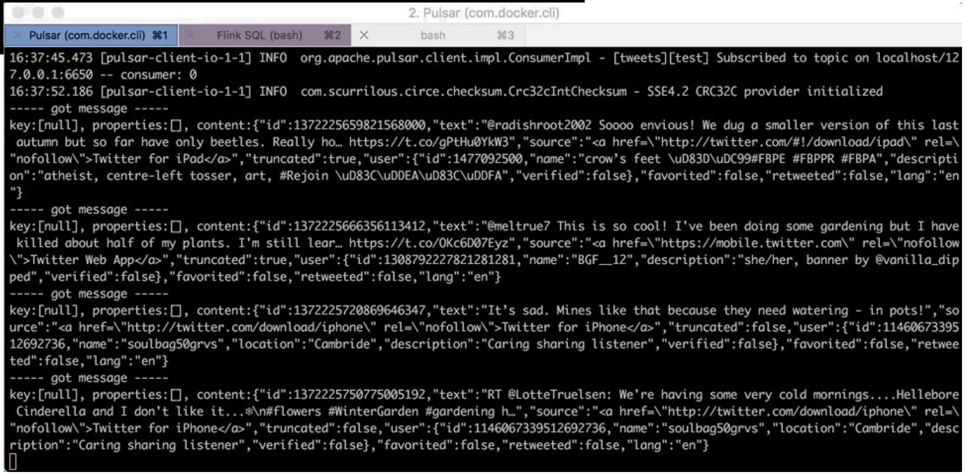
Port connector to the new, unified Source API (FLIP-27)

* For upcoming improvements to Flink SQL as a whole, check this Wiki page

How does this look like, **in practice**?

DEMO

1. Use the Twitter Firehose built-in connector to consume tweets about gardening 🌿 into a Pulsar topic (`tweets`).

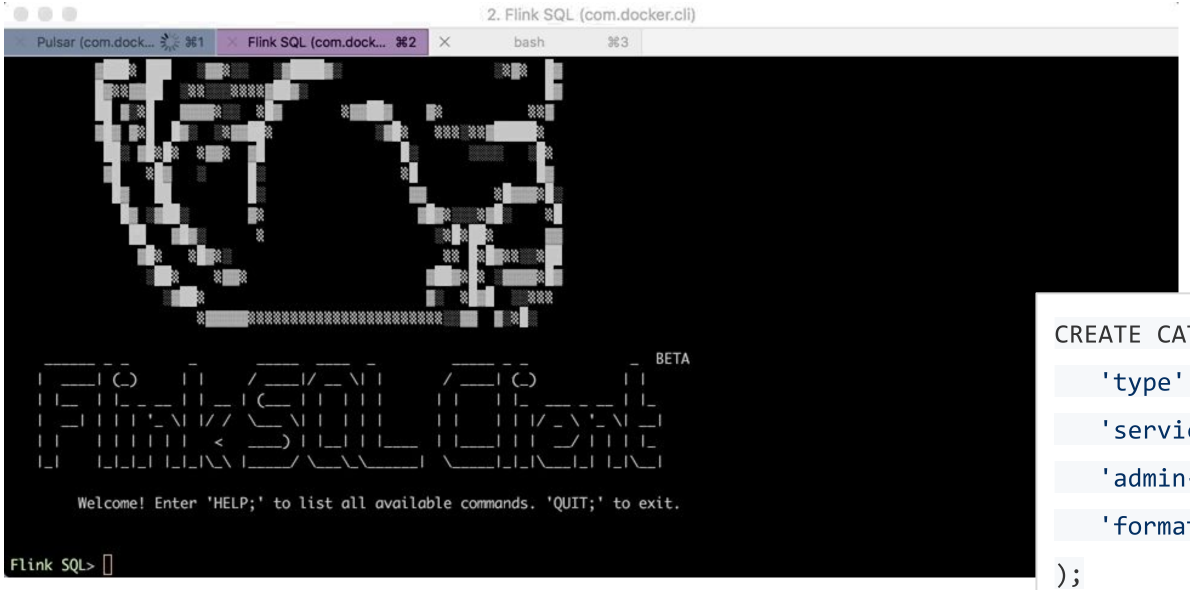


Follow along with this example: https://github.com/morsapaes/flink-sql-pulsar

DEMO

2. Start the Flink SQL Client and use a Pulsar catalog to access the topic directly as a table in Flink.

SQL Client



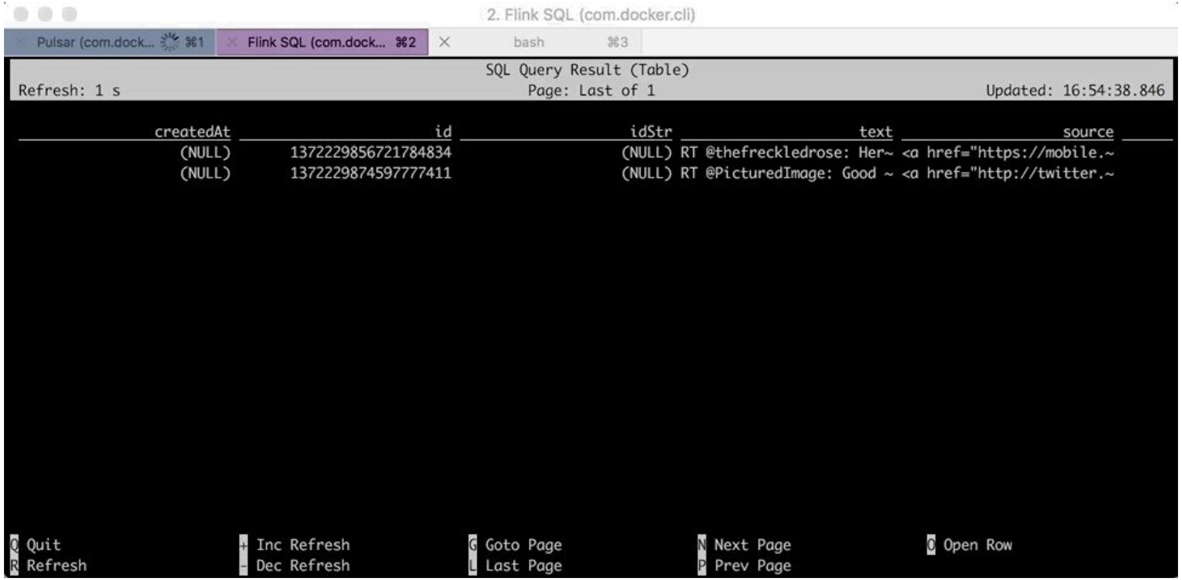
Catalog DDL

```
CREATE CATALOG pulsar WITH (
   'type' = 'pulsar',
   'service-url' = 'pulsar://pulsar:6650',
   'admin-url' = 'http://pulsar:8080',
   'format' = 'json'
);
```

DEMO

2.1. You can query the `tweets` topic off-the-bat using a simple SELECT statement — it's treated as a Flink table!

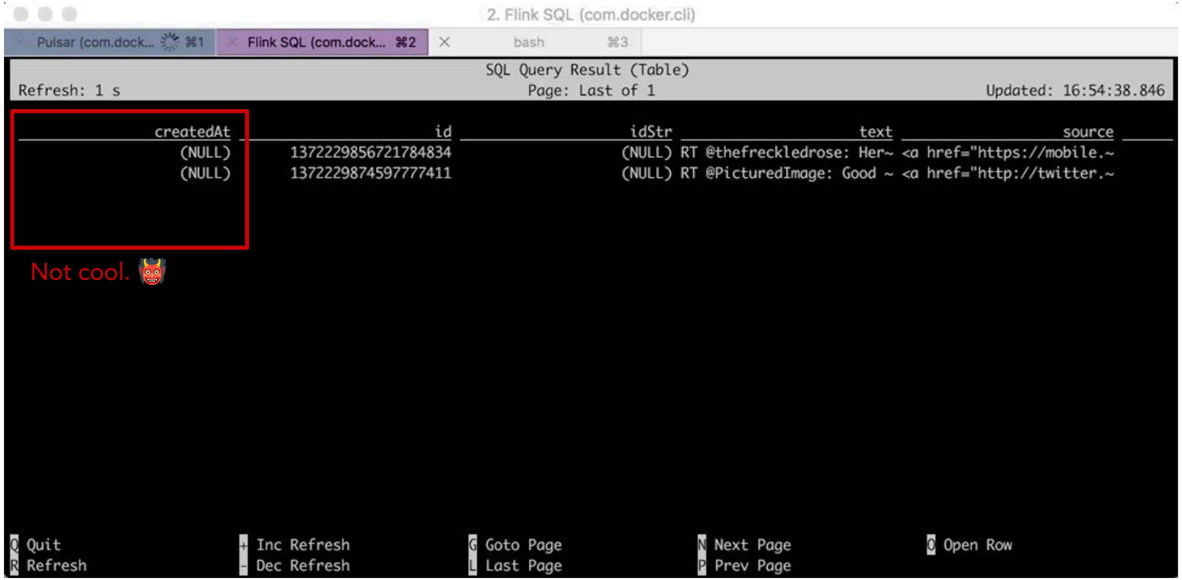
SQL Client



DEMO

2.2. But then you find out that most Firehose events have a `null` createdTime. What now?

SQL Client



DEMO

3. One way to get a relevant timestamp is to use Pulsar metadata to get the `publishTime` (i.e. ingestion time).

Source Table DDL

```
CREATE TABLE pulsar_tweets (
      publishTime TIMESTAMP(3) METADATA,
      WATERMARK FOR publishTime AS publishTime - INTERVAL '5' SECOND
) WITH (
  'connector' = 'pulsar',
  'topic' = 'persistent://public/default/tweets',
  'value.format' = 'json',
  'service-url' = 'pulsar://pulsar:6650',
  'admin-url' = 'http://pulsar:8080',
  'scan.startup.mode' = 'earliest-offset'
 )
LIKE tweets;
```

Read and use Pulsar message metadata

Define the source connector (Pulsar)

Derive schema from the original topic

Pulsar Flink connector repo: https://github.com/streamnative/pulsar-flink

DEMO

4. Perform a simple windowed aggregation (count), and insert results into a new pulsar topic (tweets_agg).

Sink Table DDL

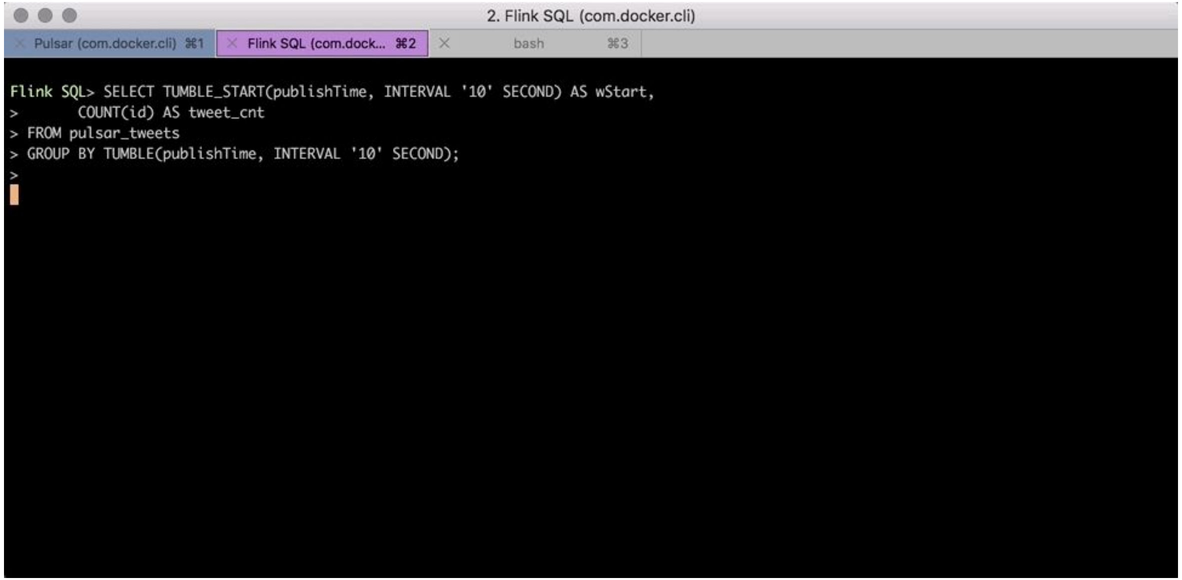
```
CREATE TABLE pulsar_tweets_agg (
      tmstmp TIMESTAMP(3),
      tweet_cnt BIGINT
) WITH (
  'connector'='pulsar',
  'topic'='persistent://public/default/tweets_agg',
  'value.format'='json',
  'service-url'='pulsar://pulsar:6650',
  'admin-url'='http://pulsar:8080'
 );
```

Continuous SQL Query

```
INSERT INTO pulsar_tweets_agg
SELECT TUMBLE_START(publishTime, INTERVAL '10' SECOND) AS
wStart,
       COUNT(id) AS tweet_cnt
FROM pulsar_tweets
GROUP BY TUMBLE(publishTime, INTERVAL '10' SECOND);
```

DEMO

5. We'll get a count of the # of tweets in windows of 10 seconds (based on event time!).



There’s a lot more to it!

Check out the [Flink SQL Cookbook](), where we share hands-on examples, patterns, and use cases for Flink SQL.

09 Maintaining Materialized Views with Change Data Capture (CDC) and Debezium

Flink Version 1.11+

💡 This example will show how you can use Flink SQL and Debezium to m

In the world of analytics, databases are still mostly seen as static sources
waiting to be queried. The reality is that most of the data stored in these
so...why not *stream* it?

Change Data Capture (CDC) allows you to do just that: track and propaga
Ahead-Log in Postgres) to downstream consumers. Debezium is a popula
Connector and **2)** a set of "standalone" Flink CDC Connectors.

Let's get to it!

In this example, you'll monitor a table with insurance claim data related to
aggregated **materialized view** that is **incrementally updated** with the lat
deploying Debezium, Kafka and Kafka Connect in this repository.

Pre-requisites

05 Real Time Star Schema Denormalization (N-Way Join)

💡 In this recipe, we will de-normalize a simple star schema with an n-way temporal table join.

Star schemas are a popular way of normalizing data within a data warehouse. At the center of a star schema is a **fact table** whose rows
contain metrics, measurements, and other facts about the world. Surrounding fact tables
metadata useful for enriching facts when computing queries.
You are running a small data warehouse for a railroad company which consists of a fact tal
(`stations` , `booking_channels` , and `passengers`). All inserts to the fact table, and all up
Apache Kafka. Records in the fact table are interpreted as inserts only, and so the table is
(`connector` = `kafka`);. In contrast, the records in the dimensional tables are upserts bas
Kafka connector (`connector` = `upsert-kafka`).

With Flink SQL you can now easily join all dimensions to our fact table using a 5-way temp
arbitrary table (left input/probe site) and correlate each row to the corresponding row’s rel
input/build side). Flink uses the SQL syntax of `FOR SYSTEM_TIME AS OF` to perform this op
consistent, reproducible results when joining a fact table with more (slowly) changing dim
is joined to its corresponding value of each dimension based on when the event occurred

Script

```
CREATE TEMPORARY TABLE passengers (
```

08 Detecting patterns with MATCH_RECOGNIZE

💡 This example will show how you can use Flink SQL to detect patterns in a stream of events with `MATCH_RECOGNIZE` .

A common (but historically complex) task in SQL day-to-day work is to identify meaningful sequences of events in a data set — also known as Complex Event Processing (CEP). This becomes even more relevant when dealing with streaming data, as you want to react quickly to known patterns or changing trends to deliver up-to-date business insights. In Flink SQL, you can easily perform this kind of tasks using the standard SQL clause `MATCH_RECOGNIZE` .

Breaking down MATCH_RECOGNIZE

In this example, you want to find users that downgraded their service subscription from one of the premium tiers (`type IN ('premium','platinum')`) to the basic tier.

Input

The input argument of `MATCH_RECOGNIZE` will be a row pattern table based on `subscriptions` . As a first step, logical partitioning and ordering must be applied to the input row pattern table to ensure that event processing is correct and deterministic:

```
PARTITION BY user_id
ORDER BY proc_time
```

Output

Thank you!

Follow me on Twitter: @morsapaes

Learn more about Flink: https://flink.apache.org/

ververica