

---

# *INTRODUCTION TO SQL ON APACHE FLINK®*

TIMO WALTHER, SOFTWARE ENGINEER

FLINK FORWARD BERLIN  
SEPTEMBER 3, 2018

---

# MOTIVATION

# FLINK'S POWERFUL ABSTRACTIONS

Layered abstractions to  
navigate simple to complex use cases

High-level  
Analytics API

Stream- & Batch  
Data Processing

Stateful Event-  
Driven Applications

SQL / Table API (dynamic tables)

DataStream API (streams, windows)

Process Function (events, state, time)

```
SELECT room, TUMBLE_END(rowtime, INTERVAL '1' HOUR), AVG(temp)  
FROM sensors  
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

```
val stats = stream  
.keyBy("sensor")  
.timeWindow(Time.seconds(5))  
.sum((a, b) -> a.add(b))
```

```
def processElement(event: MyEvent, ctx: Context, out: Collector[Result]) = {  
    // work with event and state  
    (event, state.value) match { ... }  
  
    out.collect(...) // emit events  
    state.update(...) // modify state  
  
    // schedule a timer callback  
    ctx.timerService.registerEventTimeTimer(event.timestamp + 500)  
}
```



# DATASTREAM API IS GREAT...

- Very expressive stream processing
  - Transform data, update state, define windows, aggregate, etc.
- Highly customizable windowing logic
  - Assigners, Triggers, Evictors, Lateness
- Asynchronous I/O
  - Improve communication to external systems
- Low-level Operations



# ... BUT IT IS NOT FOR EVERYONE!

- Writing distributed programs is not always easy
  - Stream processing technology spreads rapidly
  - New streaming concepts (time, state, ...)
- Requires knowledge & skill
  - Continuous applications have special requirements
  - Programming experience (Java / Scala)
- Users want to focus on their business logic



# WHY NOT A RELATIONAL API?

- Relational API is declarative
  - User says what is needed, system decides how to compute it
- Queries can be effectively optimized
  - Less black-boxes, well-researched field
- Queries are efficiently executed
  - Let Flink handle state, time, and common mistakes
- “Everybody” knows and uses SQL!



# GOALS

- Easy, declarative, and concise relational API
- Tool for a wide range of use cases
- Unification of batch & streaming with same semantics



---

# TABLE & SQL API

# APACHE FLINK'S RELATIONAL APIs

## ANSI SQL

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

## LINQ-style Table API

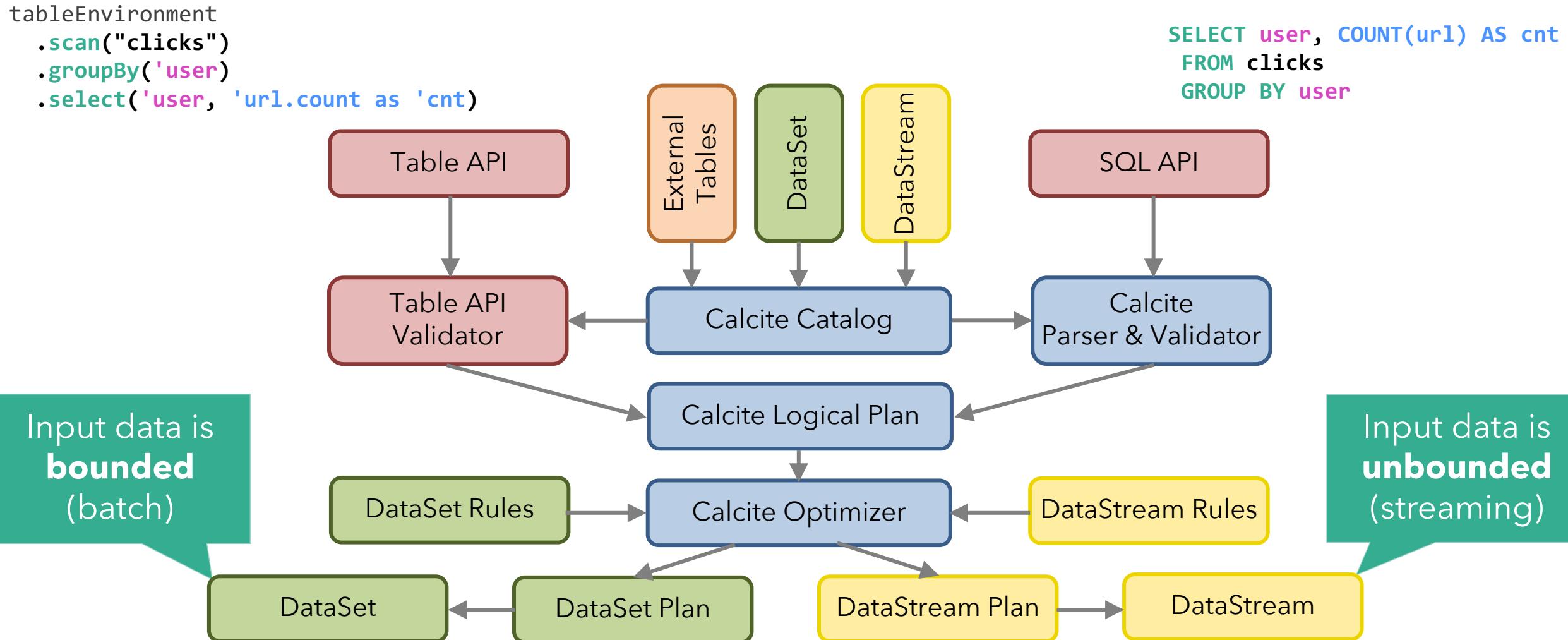
```
tableEnvironment  
.scan("clicks")  
.groupBy('user')  
.select('user', 'url.count as 'cnt')
```

Unified APIs for batch & streaming data

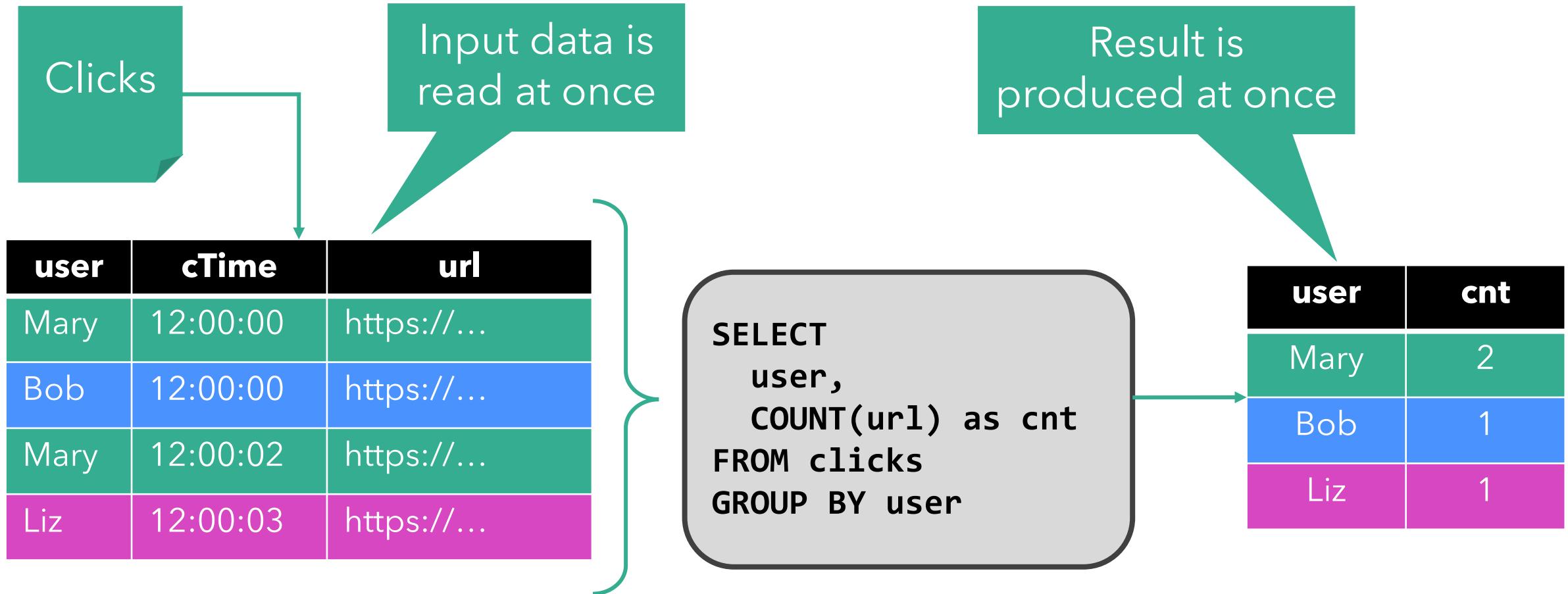
*A query specifies exactly the same result regardless whether its input is static batch data or streaming data.*



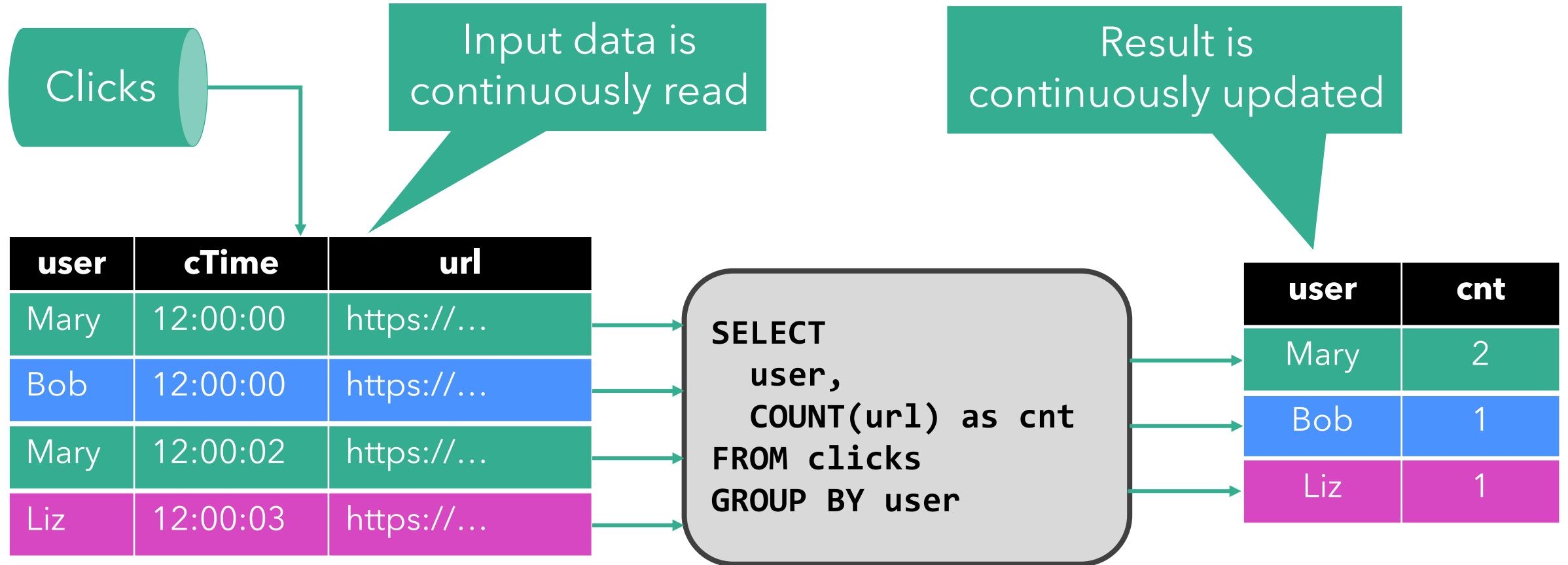
# QUERY TRANSLATION



# WHAT IF “CLICKS” IS A FILE?



# WHAT IF “CLICKS” IS A STREAM?



The result is the same!



---

# USE CASES

# BIG APACHE FLINK SQL USERS

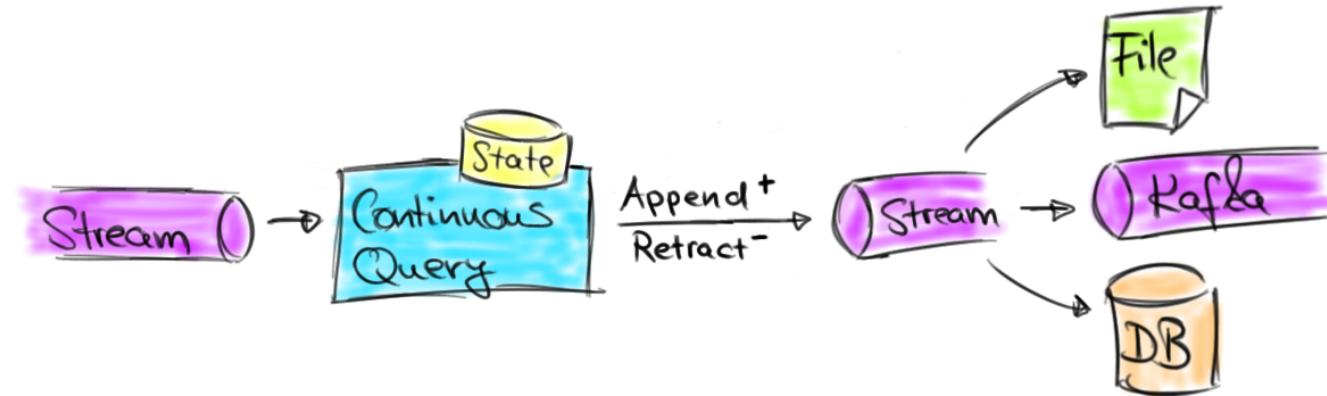


UBER



# DATA PIPELINES

- Transform, aggregate, and move events in real-time
- Low-latency ETL
  - Convert and write streams to file systems, DBMS, K-V stores, indexes, ...
  - Ingest appearing files to produce streams



# DATA PIPELINES

- Support for POJOs, maps, arrays, and other nested types
- Large set of built-in functions (150+)
  - LIKE, EXTRACT, TIMESTAMPADD, FROM\_BASE64, MD5, STDDEV\_POP, AVG, ...
- Support for custom UDFs (scalar, table, aggregate)

See also:

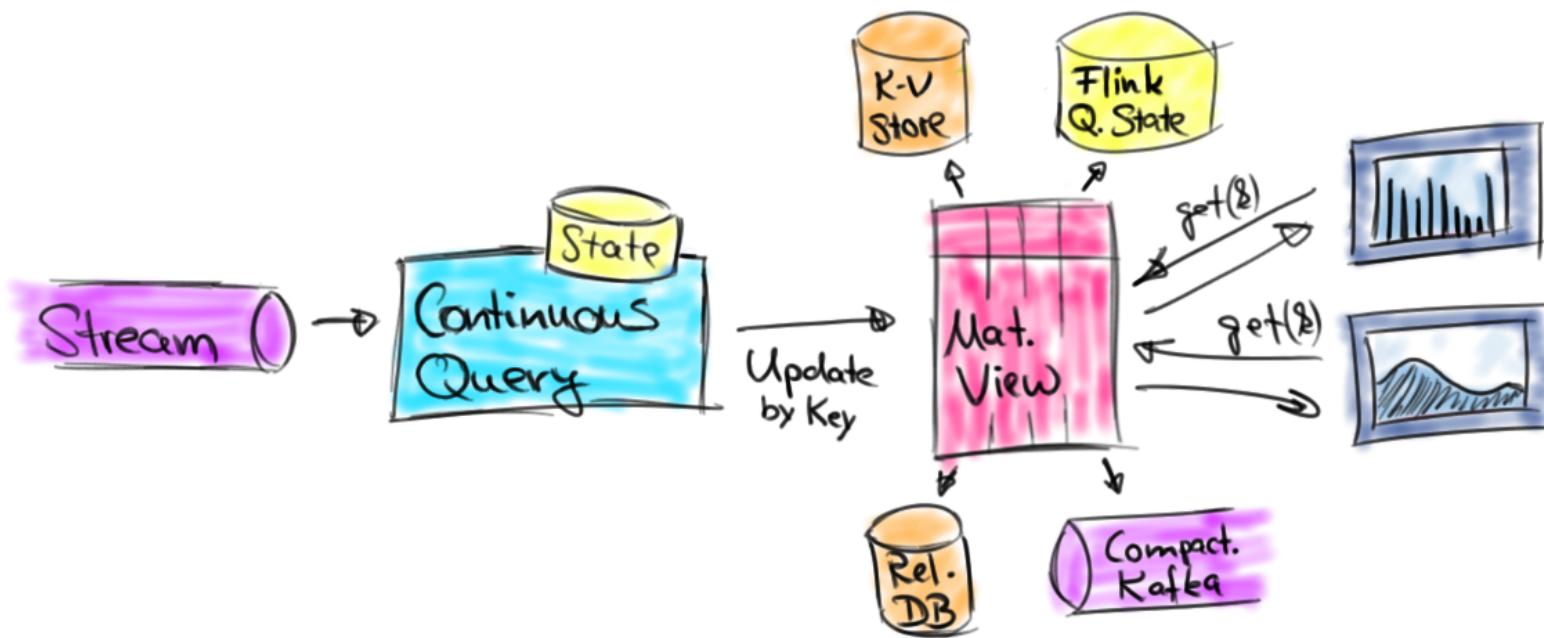
<https://ci.apache.org/projects/flink/flink-docs-master/dev/table/functions.html>

<https://ci.apache.org/projects/flink/flink-docs-master/dev/table/udfs.html>



# STREAM & BATCH ANALYTICS

- Stream & Batch Analytics
  - Run analytical queries over bounded and unbounded data
  - Query and compare historic and real-time data
  - Compute and update data to visualize in real-time



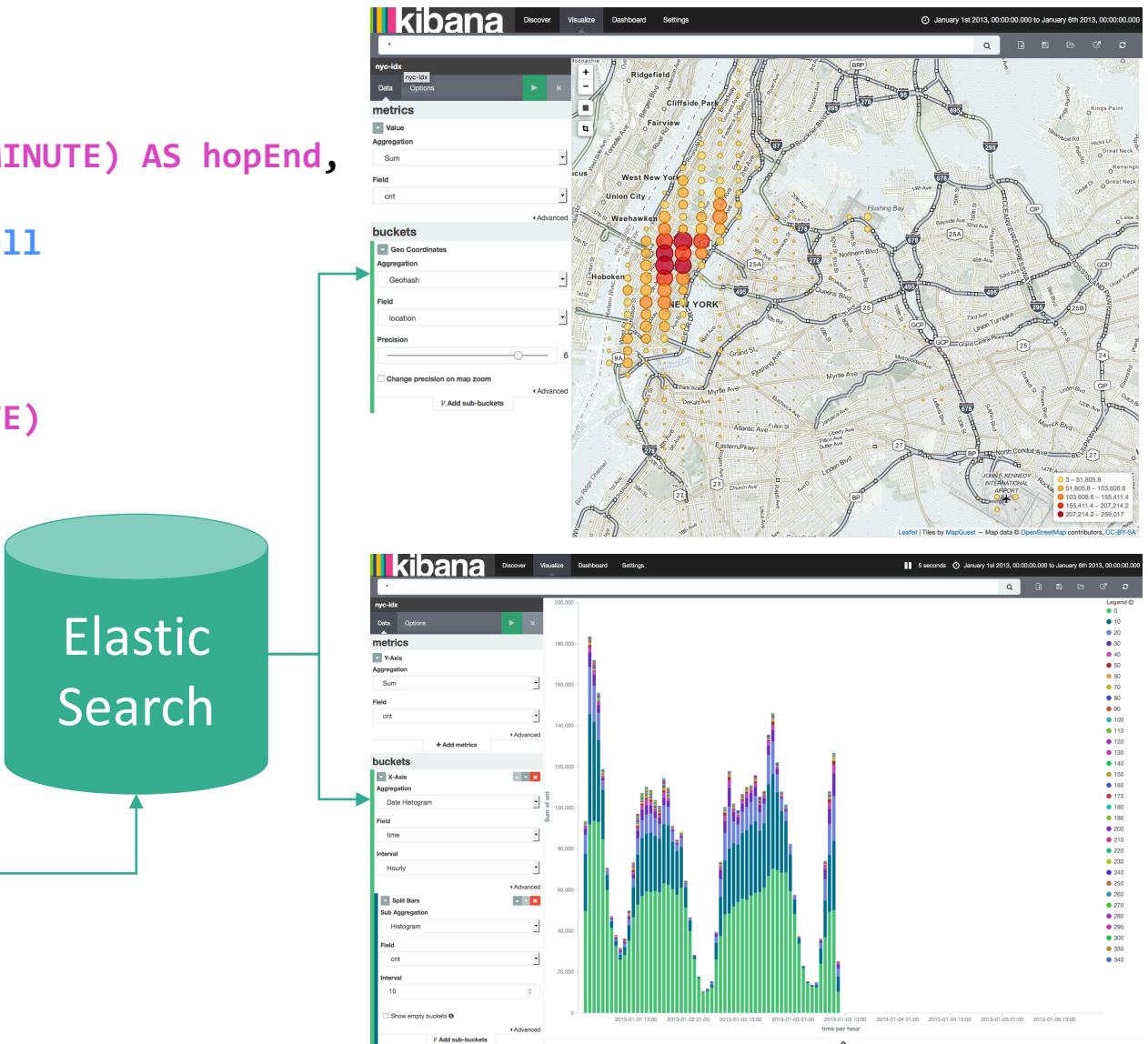
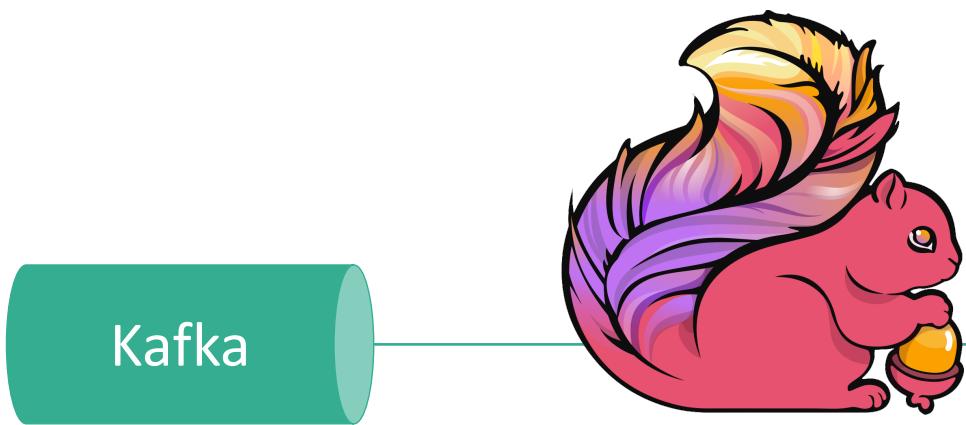
# SQL FEATURE SET IN FLINK 1.6.0

- SELECT FROM WHERE
- GROUP BY / HAVING
  - Non-windowed, TUMBLE, HOP, SESSION windows
- JOIN / IN
  - Windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
  - Non-windowed INNER, LEFT / RIGHT / FULL OUTER JOIN
- [streaming only] OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING
- [batch only] UNION / INTERSECT / EXCEPT / ORDER BY



# BUILDING A DASHBOARD EXAMPLE

```
SELECT cell,  
       isStart,  
       HOP_END(rowtime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE) AS hopEnd,  
       COUNT(*) AS cnt  
FROM (SELECT rowtime, isStart, toCellId(lon, lat) AS cell  
      FROM TaxiRides)  
GROUP BY cell,  
       isStart,  
       HOP(rowtime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE)
```



---

# SQL CLIENT BETA

**dataArtisans**

# INTRODUCTION TO SQL CLIENT

- Newest feature of the Flink SQL family (since Flink 1.5)



The screenshot shows the Flink SQL Client interface. On the left, there's a terminal window with a pixelated Flink logo and the word "BETA". The text "Welcome! Enter HELP to list all available commands. QUIT to exit." is displayed. On the right, there's a table titled "SQL Query Result (Table)" showing taxi trip data. The table has columns: rideId, taxiId, driverId, isStart, lon, lat, and rowtime. The data consists of approximately 5647 rows of taxi trip information from January 1, 2013.

rideId	taxiId	driverId	isStart	lon	lat	rowtime
67284	2013002474	2013002471	true	-73.99392	40.766483	2013-01-01 02:13:00.0
67285	2013000125	2013000125	true	-73.9902	40.73164	2013-01-01 02:13:00.0
67286	2013000543	2013000539	true	-73.988106	40.741186	2013-01-01 02:13:00.0
67287	2013007851	2013007877	true	-73.976776	40.7886	2013-01-01 02:13:00.0
67288	2013005130	2013005127	true	-73.93579	40.749916	2013-01-01 02:13:00.0
67289	2013002462	2013002479	true	-74.00155	40.72886	2013-01-01 02:13:00.0
67290	2013000810	2013000807	true	-73.95563	40.77609	2013-01-01 02:13:00.0
67291	2013008175	2013008171	true	-73.99869	40.74534	2013-01-01 02:13:00.0
67292	2013006354	2013006350	true	-73.99883	40.75019	2013-01-01 02:13:00.0
67293	2013005780	2013005777	true	-74.01012	40.719673	2013-01-01 02:13:00.0
67294	2013000597	2013000594	true	-74.08884	40.738075	2013-01-01 02:13:00.0
67295	2013001758	2013001755	true	-73.97829	40.74584	2013-01-01 02:13:00.0
67296	2013010585	2013010626	true	-73.97319	40.792835	2013-01-01 02:13:00.0
67297	2013007112	2013007108	true	-73.959915	40.77376	2013-01-01 02:13:00.0
67298	2013000600	2013000597	true	-73.970215	40.76242	2013-01-01 02:13:00.0
67299	2013000548	2013000544	true	-73.949066	40.781593	2013-01-01 02:13:00.0
67300	2013006876	2013006872	true	-73.97647	40.75168	2013-01-01 02:13:00.0
67301	2013003876	2013003873	true	-73.97031	40.757233	2013-01-01 02:13:00.0
67302	2013001431	2013001428	true	-73.98575	40.73183	2013-01-01 02:13:00.0
67303	2013001094	2013001091	true	-73.981995	40.772144	2013-01-01 02:13:00.0
67304	2013003286	2013003282	true	-73.97425	40.731556	2013-01-01 02:13:00.0
67305	2013004072	2013004069	true	-73.97144	40.79806	2013-01-01 02:13:00.0
67306	2013001433	2013001430	true	-73.971405	40.755013	2013-01-01 02:13:00.0
67307	2013010091	2013010091	true	-73.993805	40.76684	2013-01-01 02:13:00.0
67308	2013004447	2013004444	true	-73.94932	40.713917	2013-01-01 02:13:00.0
67309	2013009698	2013009697	true	-73.96828	40.762447	2013-01-01 02:13:00.0
67310	2013004806	2013004803	true	-73.96799	40.755592	2013-01-01 02:13:00.0
67311	2013006693	2013006699	true	-74.0827	40.742493	2013-01-01 02:13:00.0
67312	2013002498	2013002495	true	-73.957306	40.76597	2013-01-01 02:13:00.0
67313	2013009517	2013010348	true	-73.98775	40.754333	2013-01-01 02:13:00.0
67314	2013000819	2013000816	true	-73.99196	40.749355	2013-01-01 02:13:00.0



# INTRODUCTION TO SQL CLIENT

- Goal: Flink without a single line of code
  - only SQL and YAML
  - “*drag&drop*” SQL JAR files for connectors and formats
- Build on top of Flink's Table & SQL API
- Useful for prototyping & submission



# SQL CLIENT ENVIRONMENT FILES

- Non-programmatic way of configuring Flink jobs
- Per-session and/or global configuration in YAML
- Configures:
  - Tables from external systems
  - Views defined in SQL
  - User-defined functions
  - Execution properties (e.g. result mode, execution mode)
  - Deployment properties



# SQL CLIENT ENVIRONMENT FILE EXAMPLE

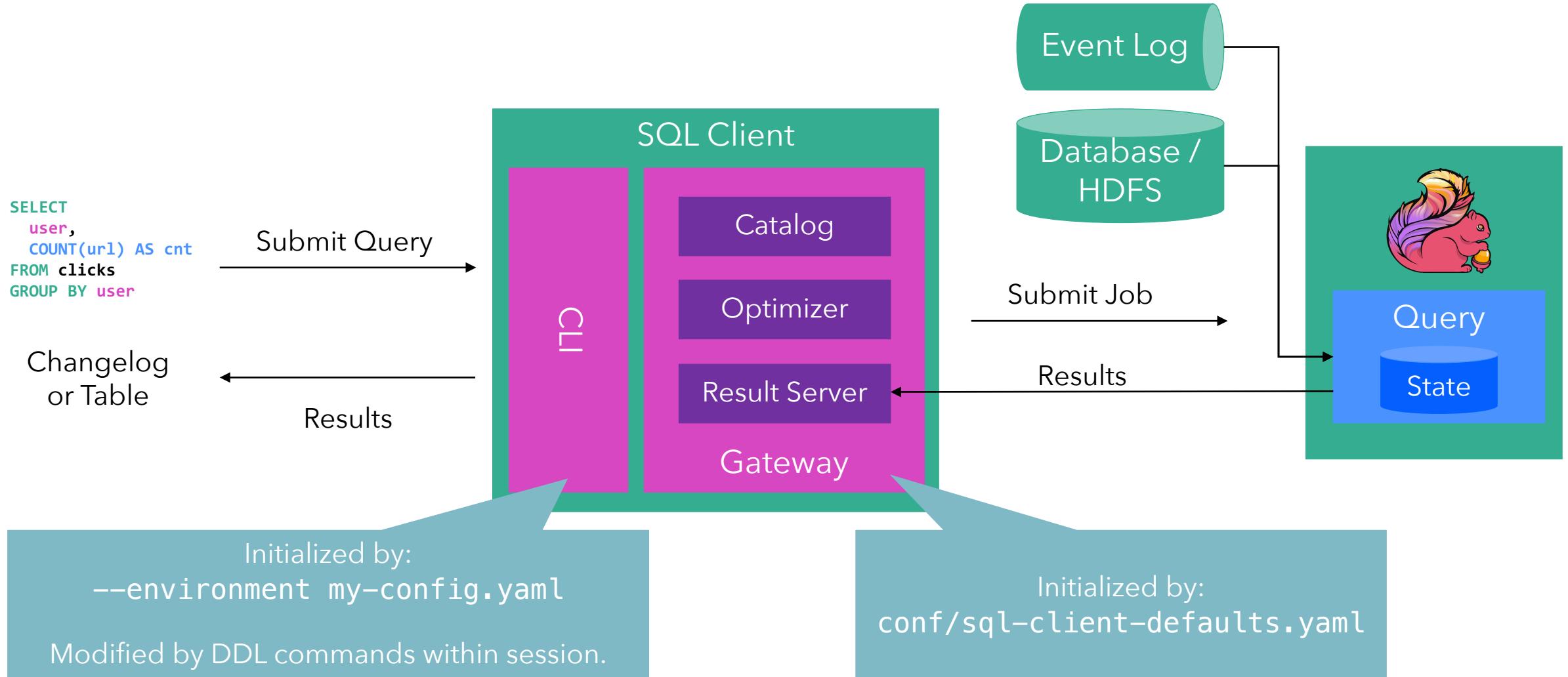
```
1 # Define table sources and sinks here.
2 tables:
3   ...- name: MyTableSource
4   ...- type: source
5   ...- update-mode: append
6   ...- connector:
7   ...- type: filesystem
8   ...- path: "/path/to/something.csv"
9   ...- format:
10  ...- type: csv
11  ...- fields:
12  ...- - name: MyField1
13  ...- - type: INT
14  ...- - name: MyField2
15  ...- - type: VARCHAR
16  ...- line-delimiter: "\n"
17  ...- comment-prefix: "#"
18  ...- schema:
19  ...- - name: MyField1
20  ...- - type: INT
21  ...- - name: MyField2
22  ...- - type: VARCHAR
23
24 # Define table views here.
25 views:
26   ...- name: MyCustomView
27   ...- query: "SELECT MyField2 FROM MyTableSource"
28
29 # Define user-defined functions here.
30 functions:
31   ...- name: myUDF
32   ...- from: class
33   ...- class: foo.bar.AggregateUDF
34
35 # Execution properties allow for changing the behavior of a table program.
36 execution:
37   ...- type: streaming .....# required: execution mode either 'batch' or 'streaming'
38   ...- result-mode: table .....# required: either 'table' or 'changelog'
39   ...- parallelism: 1 .....# optional: Flink's parallelism (1 by default)
```

See also:

<https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html>



# PLAY AROUND WITH FLINK SQL



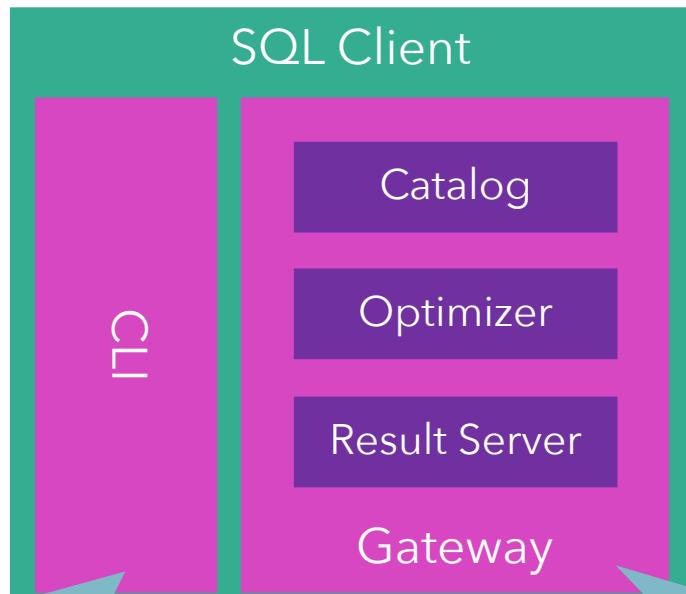
# SUBMIT DETACHED QUERIES

```
INSERT INTO dashboard  
SELECT  
  user,  
  COUNT(url) AS cnt  
FROM clicks  
GROUP BY user
```

Cluster ID &  
Job ID

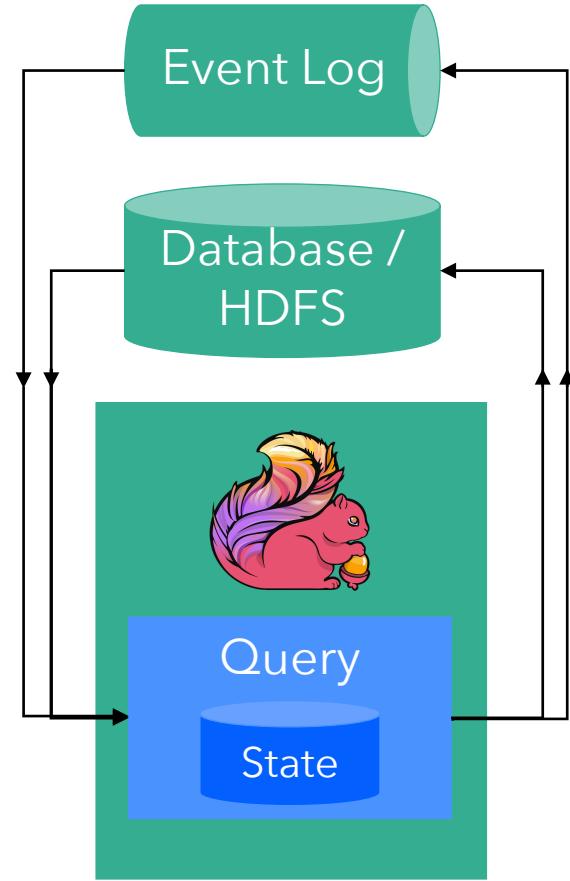
Submit Query

Target Information



Submit Job

Initialized by:  
conf/sql-client-defaults.yaml



---

# HANDS-ON

# PREPARED DOCKER IMAGE & EXERCISES

Please visit the SQL training wiki to start:

<https://github.com/dataArtisans/sql-training/wiki>

We are here to help!



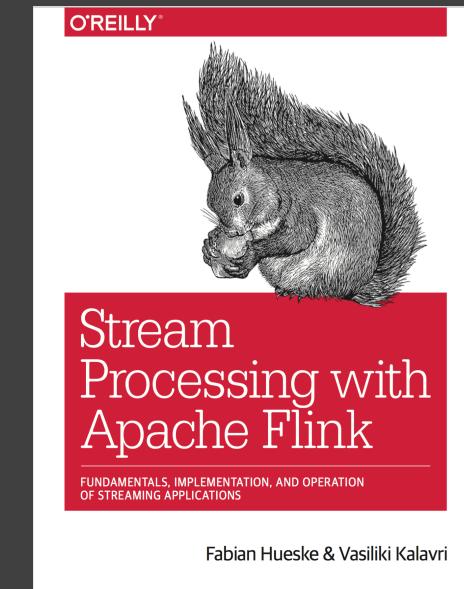
---

# THANK YOU!

@twalthr

@dataArtisans

@ApacheFlink



Available on O'Reilly Early Release!

WE ARE HIRING  
[data-artisans.com/careers](http://data-artisans.com/careers)

**dataArtisans**