Introduction to SQL on Apache Flink®

Flink SQL Training

https://github.com/ververica/sql-training

Motivation

Flink's Powerful Abstractions

Layered abstractions to
navigate simple to complex use cases

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

High-level
Analytics API

SQL / Table API (dynamic tables)

Stream- & Batch
Data Processing

DataStream API (streams, windows)

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

Stateful Event-
Driven Applications

Process Function (events, state, time)

```
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)
}
```

The DataStream API is great…

- Very expressive stream processing API
  - Transform, aggregate, and join events
  - Java and Scala
- Control how events are processed with respect to time
  - Timestamps, Watermarks, Windows, Timers, Triggers, Allowed Lateness, …
- Maintain and update application state
  - Keyed state, operator state, state backends, checkpointing, …

… but it‘s not made for everyone.

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

Why not SQL (or another relational API)?

- Relational APIs are declarative
  - User says what is needed, system decides how to compute it
- Queries can be effectively optimized
  - Less imperative black-box code
  - Well-researched field
- Queries are efficiently executed
  - Let Flink deal with state and time
- "Everybody" knows and uses SQL

Goals

- Easy, declarative, and concise relational API
- Expressive enough for a wide range of use cases
- Unified syntax and semantics for batch & streaming data

Table API & SQL

Apache Flink's Relational APIs

ANSI SQL

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

LINQ-style Table API

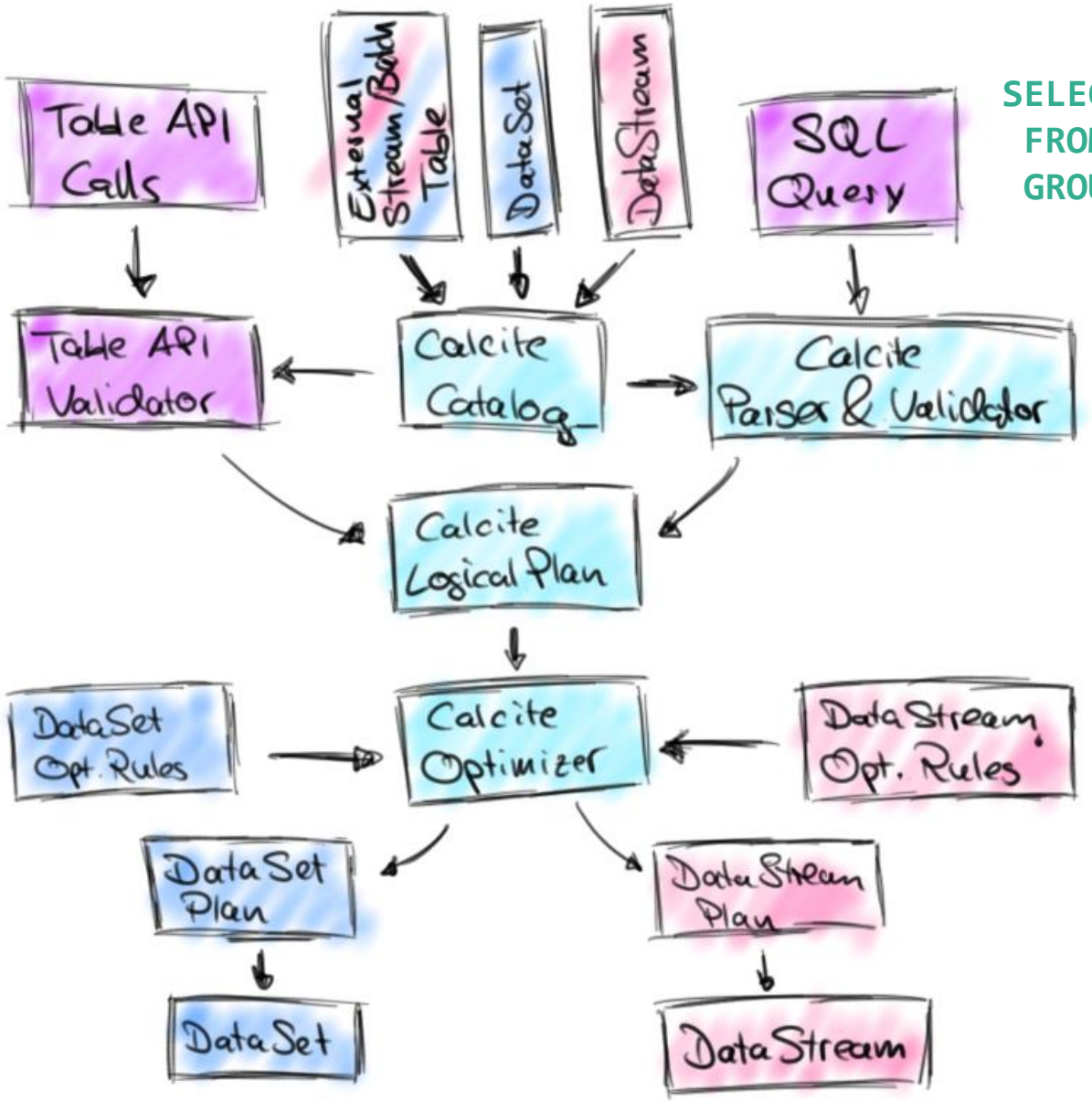
```
tableEnvironment
  .scan("clicks")
  .groupBy('user)
  .select('user, 'url.count as 'cnt)
```

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

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




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

Input data is **bounded** (batch)

Input data is **unbounded** (streaming)

What if “Clicks” is a File?

Clicks

Input data is read at once

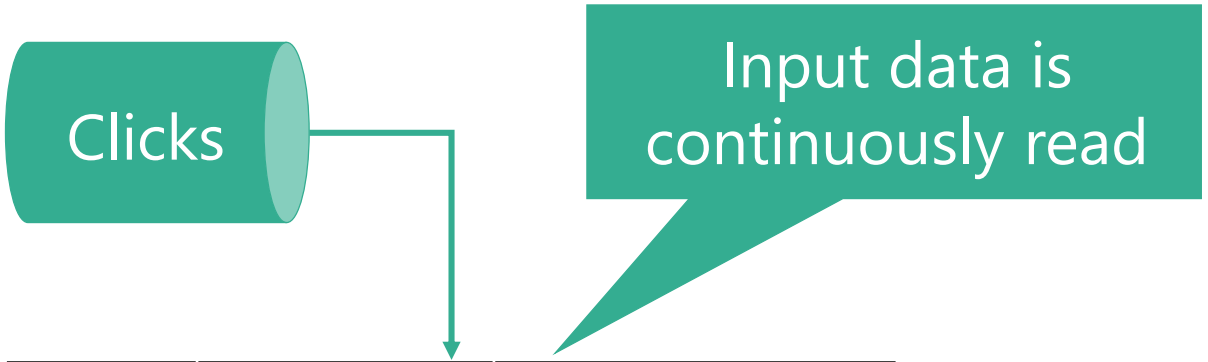
user	cTime	url
Mary	12:00:00	https://...
Bob	12:00:00	https://...
Mary	12:00:02	https://...
Liz	12:00:03	https://...

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

Result is produced at once

user	cnt
Mary	2
Bob	1
Liz	1

What if “Clicks” is a Stream?



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

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

Result is continuously updated

user	cnt
Mary	2
Bob	1
Liz	1

The result is the same!

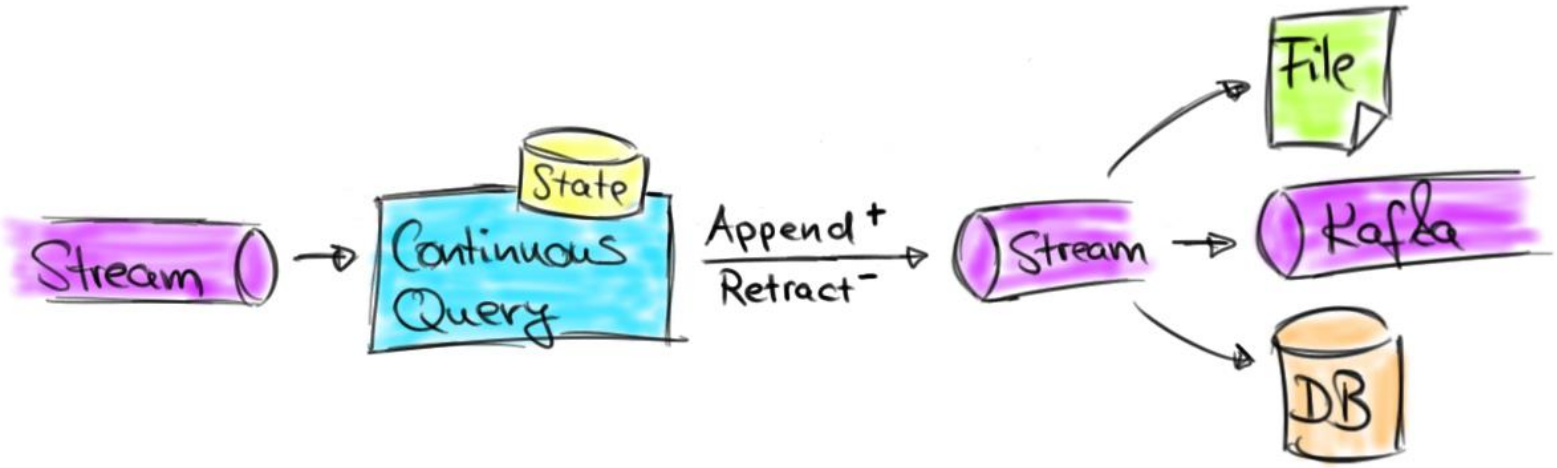
Use Cases

Flink SQL is Used in Production

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

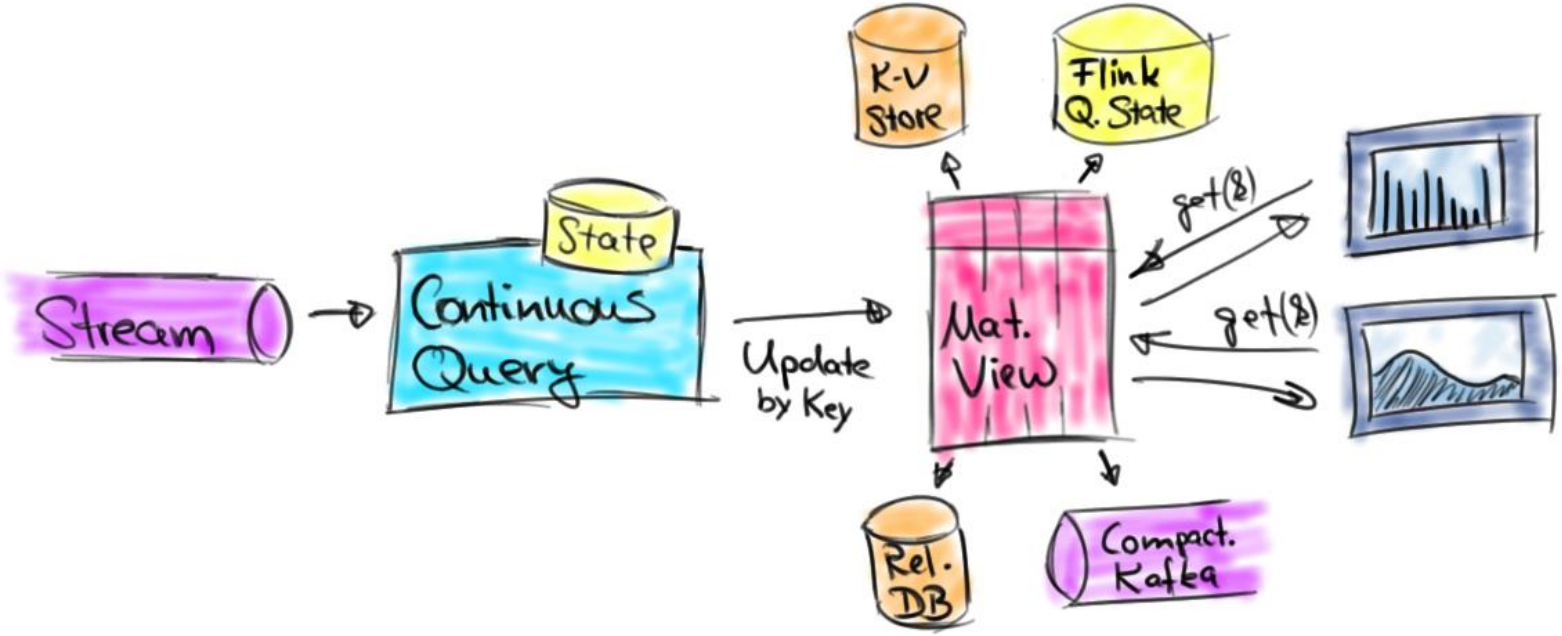
- Large set of built-in functions (150+)
  - `LIKE, EXTRACT, TIMESTAMPADD, FROM_BASE64, MD5, STDDEV_POP, AVG, …`

- 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 1.7

STREAMING & BATCH

- SELECT FROM WHERE
- GROUP BY [HAVING]
  - Non-windowed
  - TUMBLE, HOP, SESSION windows
- JOIN
  - Time-Windowed INNER + OUTER JOIN
  - Non-windowed INNER + OUTER JOIN
- User-Defined Functions
  - Scalar
  - Aggregation
  - Table-valued

STREAMING ONLY

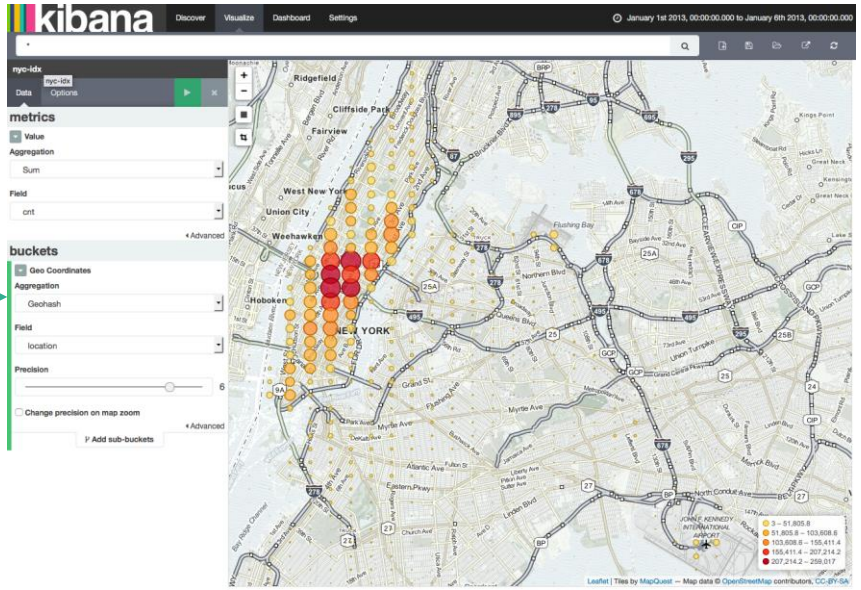
- OVER / WINDOW
  - UNBOUNDED / BOUNDED PRECEDING
- INNER JOIN with time-versioned table
- MATCH_RECOGNIZE
  - Pattern Matching/CEP (SQL:2016)

BATCH ONLY

- UNION / INTERSECT / EXCEPT
- ORDER BY

Building a Dashboard

```
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)
```

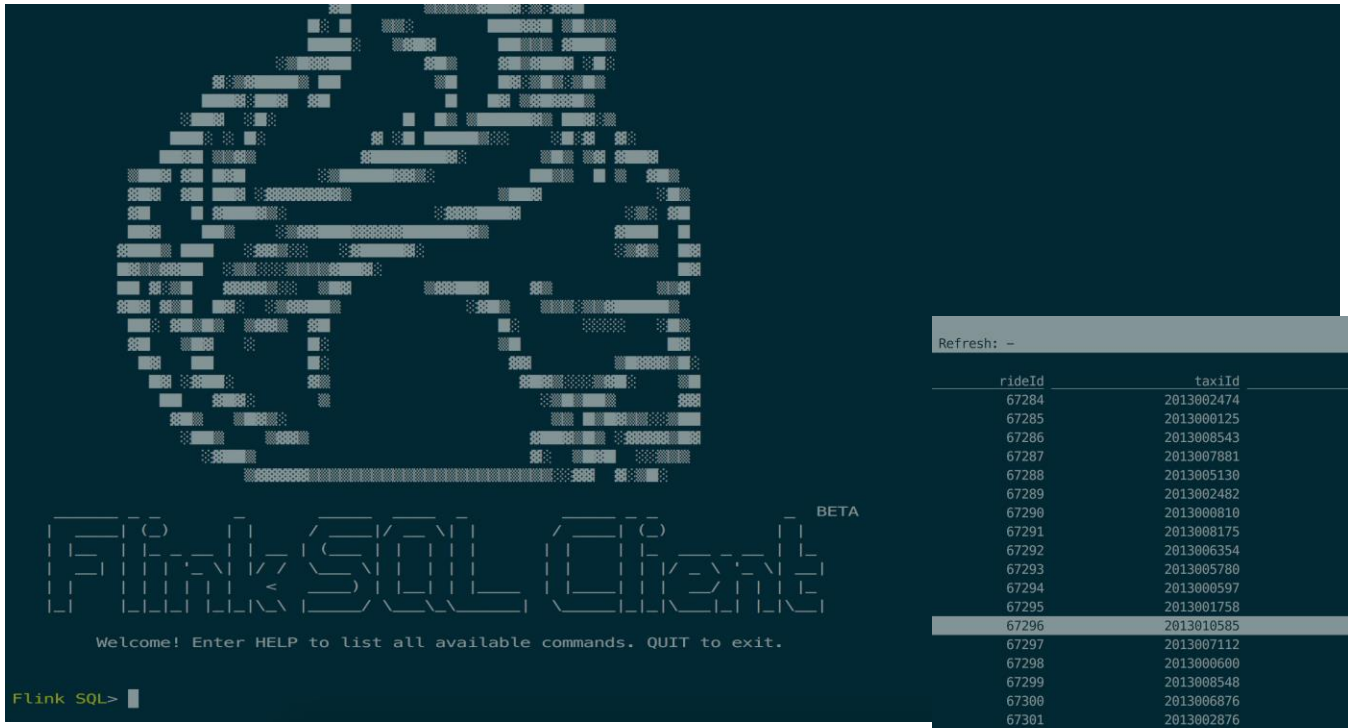


Elastic Search

Kafka

SQL Client

Introduction to SQL Client



SQL Query Result (Table)

Refresh: - Page: Last of 5647 Updated: 09:27:49.910

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	2013008543	2013008539	true	-73.988106	40.741108	2013-01-01 02:13:00.0
67287	2013007881	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	2013002482	2013002479	true	-74.00155	40.72888	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	-73.99083	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	-74.00884	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	2013008548	2013008544	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	2013002876	2013002873	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	2013006093	2013006089	true	-74.0027	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

Q Quit R Refresh + Inc Refresh - Dec Refresh G Goto Page L Last Page N Next Page P Prev Page O Open Row

Introduction to SQL Client

- Flink without a single line of code
  - Only SQL and YAML
  - Add connectors and formats by downloading SQL JAR files

- Use cases
  - Query prototyping
  - Ad-hoc stream analytics & inspection
  - Detached query submission

SQL Client Environment Files

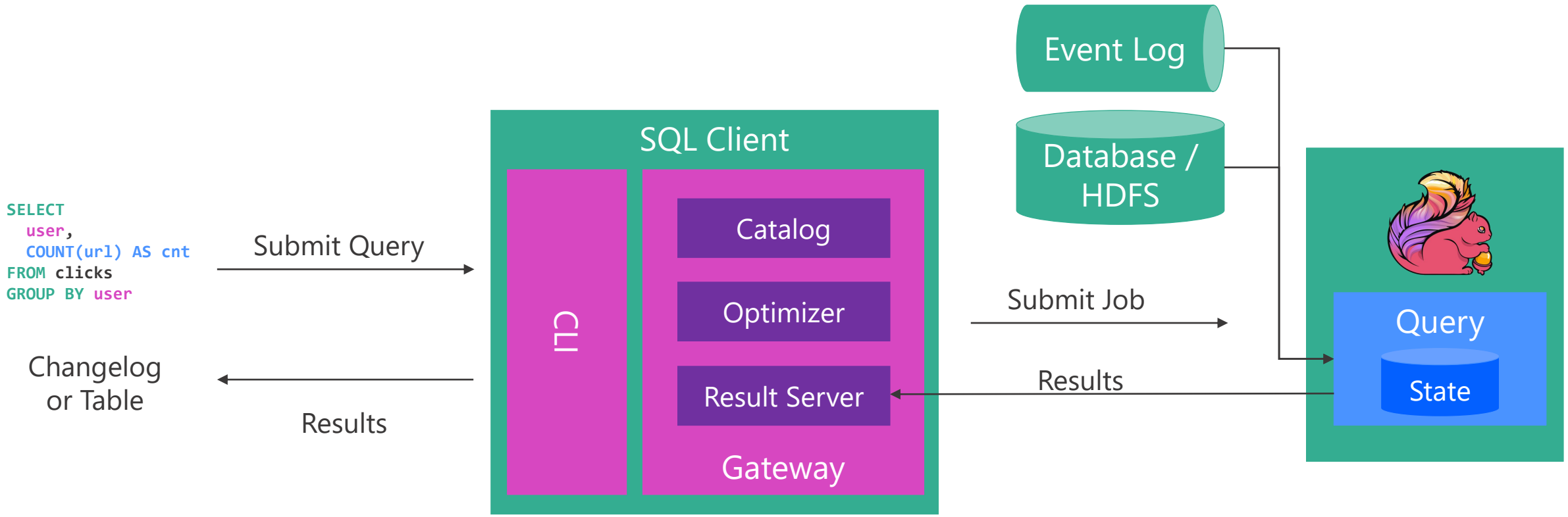
- Non-programmatic way of configuring Flink jobs
- Per-session and/or global configuration in YAML
- Environment file defines
  - Table schema and connection details to external systems
  - Views
  - User-defined functions
  - Execution properties (e.g. result mode, execution mode)
  - Deployment properties

SQL Client Environment File Example

```yaml
# Define table sources and sinks here.
tables:
  - name: MyTableSource
    type: source
    update-mode: append
    connector:
      type: filesystem
      path: "/path/to/something.csv"
    format:
      type: csv
      fields:
        - name: MyField1
          type: INT
        - name: MyField2
          type: VARCHAR
      line-delimiter: "\n"
      comment-prefix: "#"
    schema:
      - name: MyField1
        type: INT
      - name: MyField2
        type: VARCHAR

# Define table views here.
views:
  - name: MyCustomView
    query: "SELECT MyField2 FROM MyTableSource"

# Define user-defined functions here.
functions:
  - name: myUDF
    from: class
    class: foo.bar.AggregateUDF

# Execution properties allow for changing the behavior of a table program.
execution:
  type: streaming                    # required: execution mode either 'batch' or 'streaming'
  result-mode: table                 # required: either 'table' or 'changelog'
  parallelism: 1                     # optional: Flink's parallelism (1 by default)
```

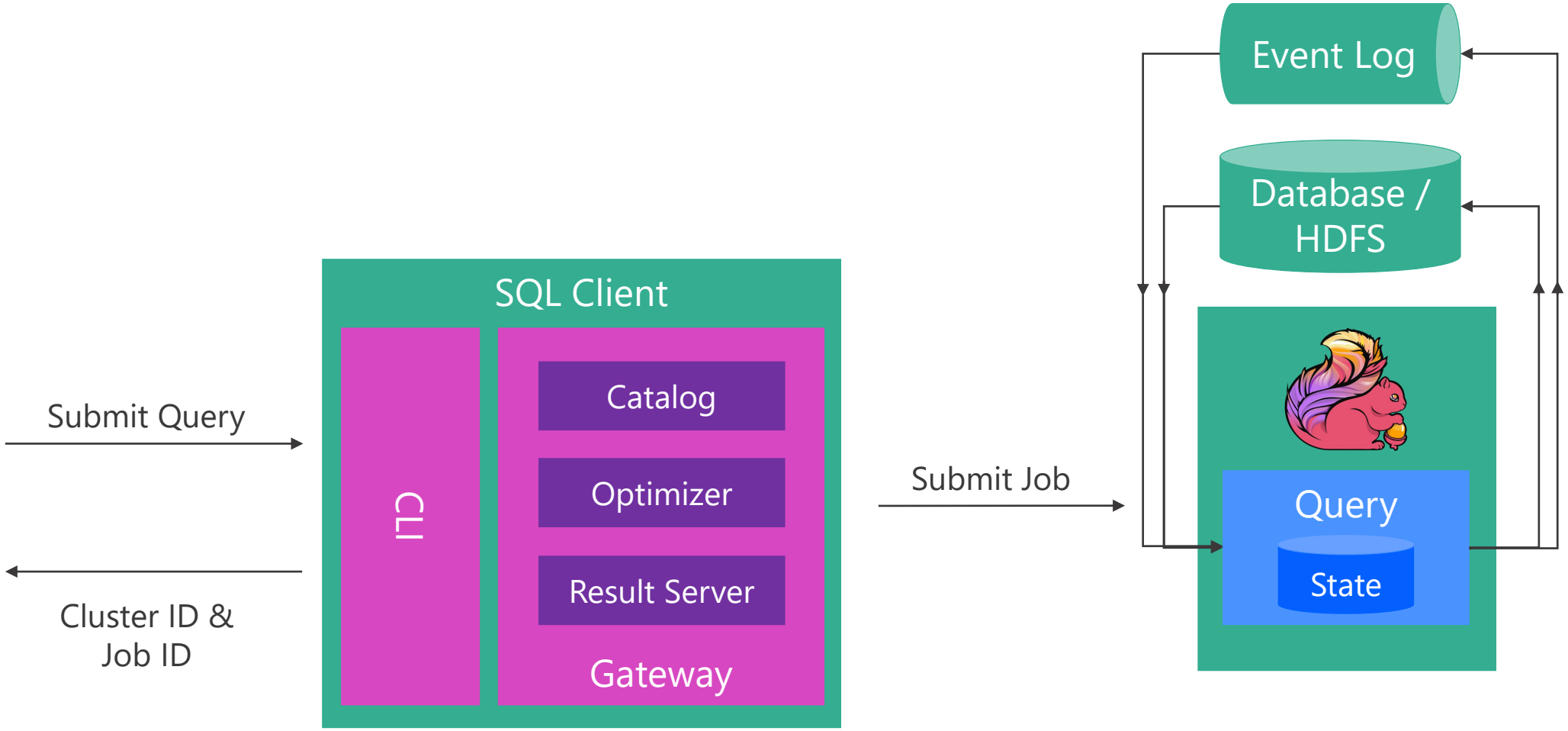
See also:
https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html

Interactive Query Submission via SQL Client



Detached Query Submission via SQL Client

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



Hands On Exercises

Prepared Docker Image & Exercises

Please visit the SQL training wiki to start:

https://github.com/ververica/sql-training/wiki

We are here to help!

ververica

www.ververica.com

@VervericaData