FLINK'S TABLE API & SQL ECOSYSTEM

TIMO WALTHER, SOFTWARE ENGINEER

FLINK FORWARD BERLIN
SEPTEMBER 3, 2018

dataArtisans

WHAT SQL CAN DO FOR US

- Schema-awareness at all times
  - including null support!
- Smooth integration with catalogs, connectors, formats
- Large set of built-in operations and functions
- Defining pipelines by "copy-pasting strings"

ISSUES THAT SQL CAN'T SOLVE

- Complex time or state operations
  - setting or reschedule timers
  - split and merge pipelines
  - custom state management
- Integration with other Flink features
  - such as CEP* or queryable state*
- More support during pipeline creation
  - available operations, operator/function docs, etc.

*not yet

TABLE API

dataArtisans

AS A UTILITIES LIBRARY (TABLE API)

```
val sensorData: DataStream[(String, Long, Double)] = ???

// convert DataStream into Table
val sensorTable: Table = sensorData.toTable(tableEnv, 'location, 'rowtime, 'tempF)

// define query on Table
val avgTempCTable: Table = sensorTable

  .window(Tumble over 1.day on 'rowtime as 'w)

  .groupBy('location, 'w)

  .select('w.start as 'day, 'location, (('tempF.avg - 32) * 0.556) as 'avgTempC)

  .where('location like "room%")

// go back to DataStream API
val avgTempC: DataStream[Row] = avgTempCTable.toAppendStream[Row]
```


AS A UTILITIES LIBRARY (SQL)

```
val sensorData: DataStream[(String, Long, Double)] = ???

// register DataStream
tableEnv.registerDataStream("sensorData", sensorData, 'location, 'rowtime, 'tempF)

// query registered Table
val avgTempCTable: Table = tableEnv.sqlQuery("""
    SELECT
         TUMBLE_START(TUMBLE(time, INTERVAL '1' DAY) AS day,
         location,
         AVG((tempF - 32) * 0.556) AS avgTempC

    FROM sensorData

    WHERE location LIKE 'room%’

    GROUP BY location, TUMBLE(time, INTERVAL '1' DAY)
  """)

// go back to DataStream API
val avgTempC: DataStream[Row] = avgTempCTable.toAppendStream[Row]
```

 |

AS CONNECTOR & PREPROCESSOR

```
val tEnv = TableEnvironment.getTableEnvironment(env)
tEnv.registerFunction("myParser", new MyParser())


// configure your data source and register as a table
tEnv
  .connect(new Kafka().topic("MyTopic") …)
  .withFormat(new Json().deriveSchema())
  .withSchema(
    new Schema()
      .field("name", Types.STRING)
      .field("prefs", Types.STRING))
  .registerTableSource("customers")

// define your table program
val table: Table = tEnv.sqlQuery("SELECT LOWER(name), myParser(prefs) FROM customers")
val table: Table = tEnv.scan("customers").select('name.lowerCase(), myParser('prefs))

// convert
val ds: DataStream[Customer] = table.toAppendStream[Customer]
```


UNIFIED SOURCES & SINKS

dataArtisans

CONNECT EXTERNAL SYSTEMS

- Declarative and unified abstractions
- Map schemas from external systems to Flink and vice-versa

Java/Scala

```
tableEnvironment
  .connect(...)
  .withFormat(...)
  .withSchema(...)
  .inAppendMode()
  .registerTableSink("MyTable")
```

YAML

```
name: MyTable
type: sink
update-mode: append
connector: ...
format: ...
schema: ...
```

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

CONNECT EXTERNAL SYSTEMS - CONNECTOR

Java/Scala

```
.connect(
  new Kafka()
    .version("0.10")
    .topic("test-input")
    .startFromEarliest()
    .property(
      "zookeeper.connect",
      "localhost:2181")
    .property(
      "bootstrap.servers",
      "localhost:9092")
)
```

YAML

```
connector:
  type: kafka
  version: "0.10"
  topic: test-input
  startup-mode: earliest-offset
  properties:
    - key: zookeeper.connect
      value: localhost:2181
    - key: bootstrap.servers
      value: localhost:9092
```


CONNECT EXTERNAL SYSTEMS - FORMAT

Java/Scala

```
.withFormat(
  new Avro()
    .avroSchema("""
      {
        "namespace": "org.myorganization",
        "type": "record",
        "name": "UserMessage",
        "fields": [
          {"name": "user", "type": "long"},
          {"name": "message", "type": "string"}
        ]
      }""")
)
```

YAML

```
format:
  type: avro
  avro-schema: >
    {
      "namespace": "org.myorganization",
      "type": "record",
      "name": "UserMessage",
      "fields": [
        {"name": "ts", "type": "string"},
        {"name": "user", "type": "long"},
        {"name": "message", "type": "string"}
      ]
    }
```


SUPPORTED CONNECTORS/FORMATS

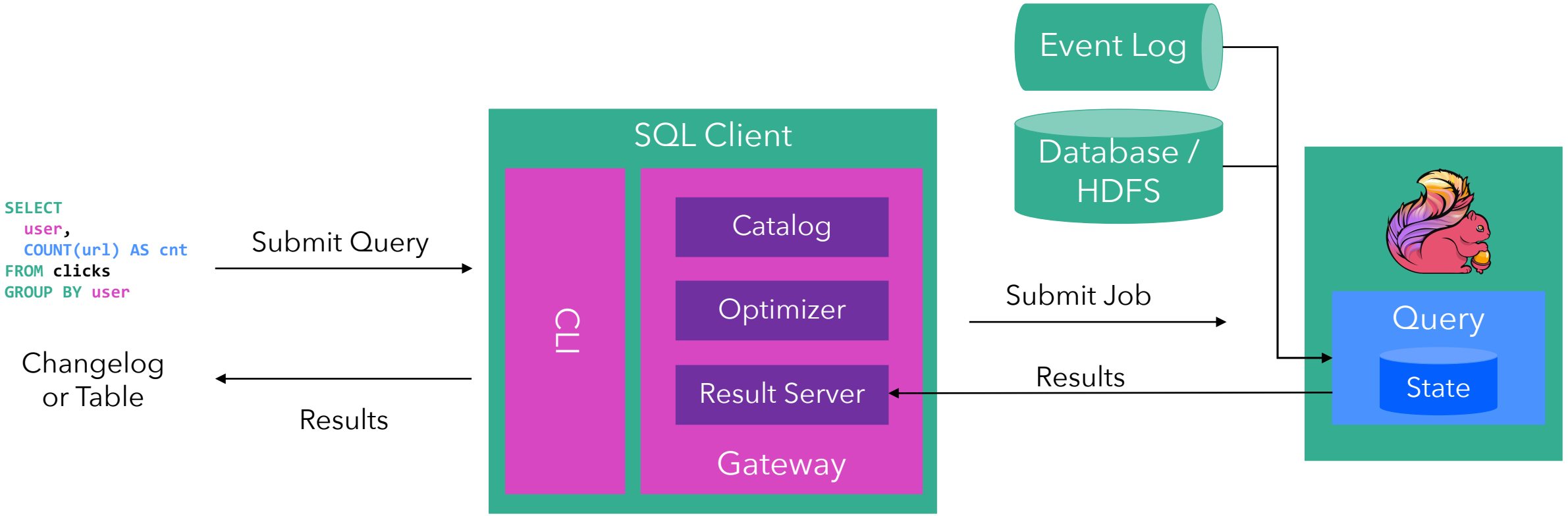
- Connectors:
  - Kafka 0.8, 0.9, 0.11, 1.0*, 1.1*
  - Elasticsearch 6*
  - Filesystem
- Formats:
  - JSON
  - Avro
  - CSV
- API for custom connectors/formats (Java/Scala and SQL Client)

*not yet

FUTURE OF SQL CLIENT

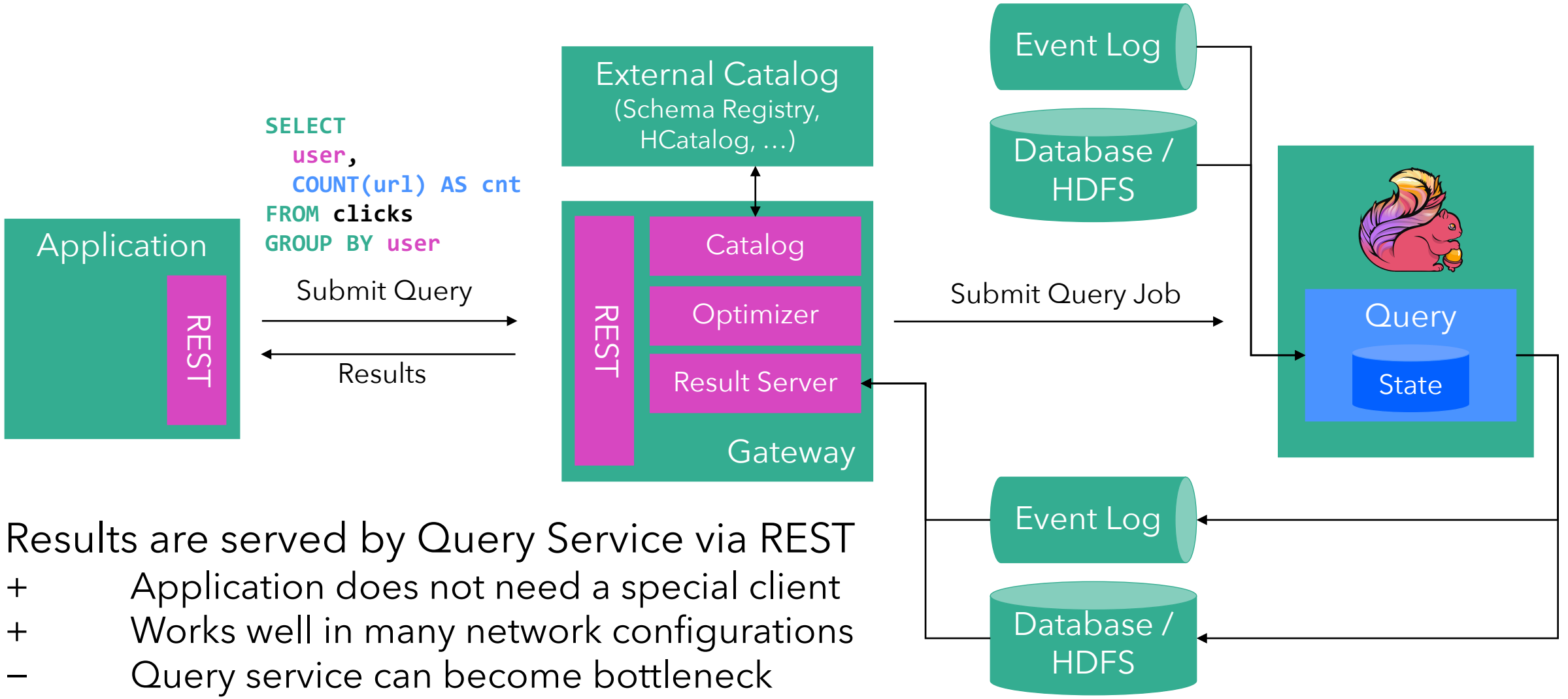
dataArtisans

CURRENT STATUS



 |

FLIP-24 – A SQL QUERY SERVICE



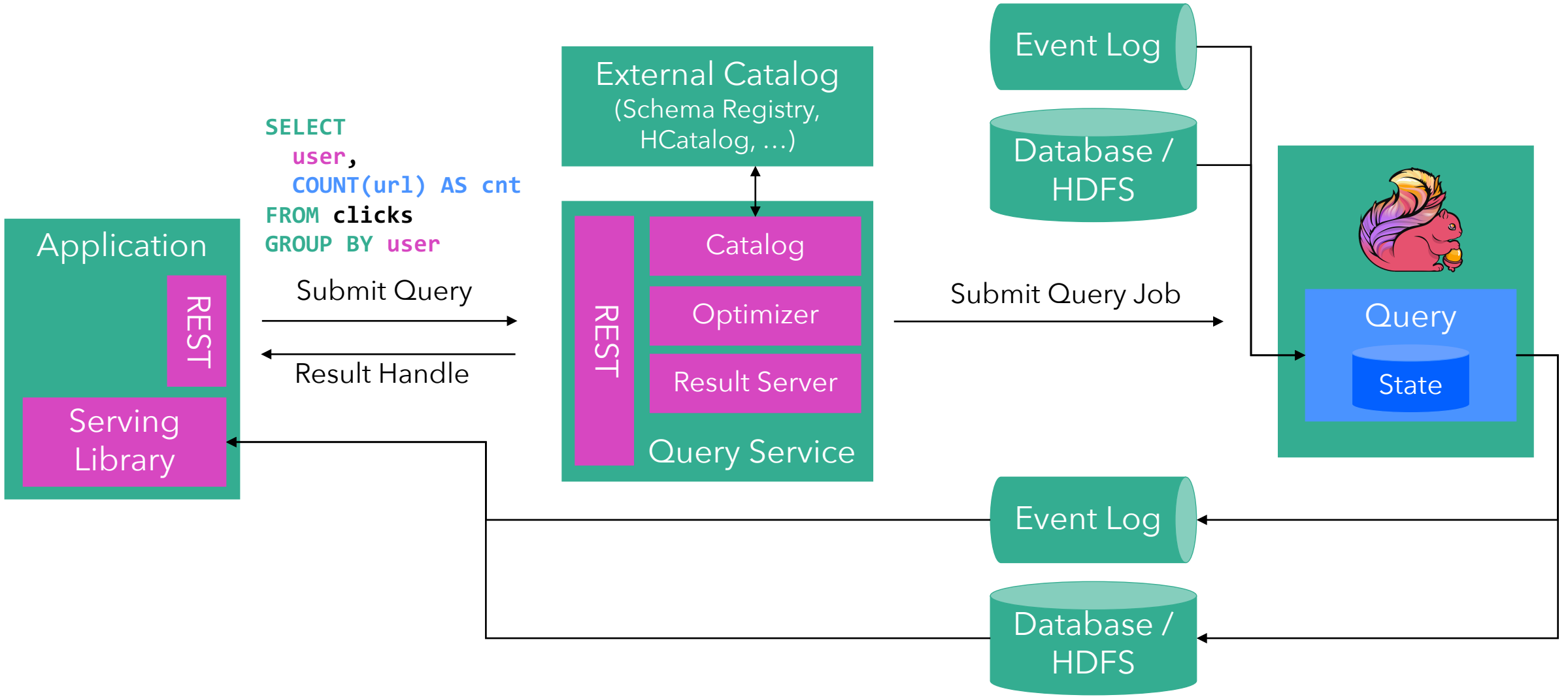
Results are served by Query Service via REST

\+ Application does not need a special client

\+ Works well in many network configurations

– Query service can become bottleneck

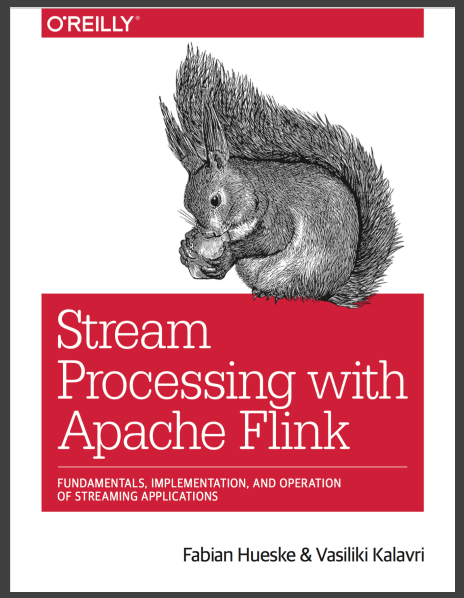
FLIP-24 – A SQL QUERY SERVICE



 |

THANK YOU!

@twalthr
@dataArtisans
@ApacheFlink



Available on O'Reilly Early Release!

WE ARE HIRING

data-artisans.com/careers

dataArtisans