Structured Streaming for Columnar Data Warehouses

Jack Gudenkauf
Senior Architect
HPE Big Data Professional Services

Our solution is a

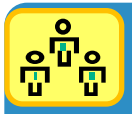
Parallel Streaming Transformation Loader Application

Agenda

- Benefits
- Features
- Architecture
- A self-service *"ETL"*
  - Sources, Transformations, Sinks

PSTL Benefits

Analysts and Data Scientists spend up to 80% of their time cleaning and reshaping data. With PSTL, they will spend 20%



Analysts

- Self-serve "ETL" using Spark SQL

Users

- ad-hoc query & reporting of near real-time data

Engineers

- An extensible, scalable, unified data pipeline

Features

Of our Autonomous Spark Solution

Highly Performant and Highly Scalable

- **Distributed systems**
- **Scale out clusters**

Operational Robustness

- **Observability through metrics and dashboards**
- **Strong durability guarantees**
- **Exactly-once End-to-end**

Self-Serve

- **No developer code needed for**
  - **Streaming data ingestion**
  - **Processing semi-structured data formats**
  - **Filtering data**
  - **Data Transformations or loading**

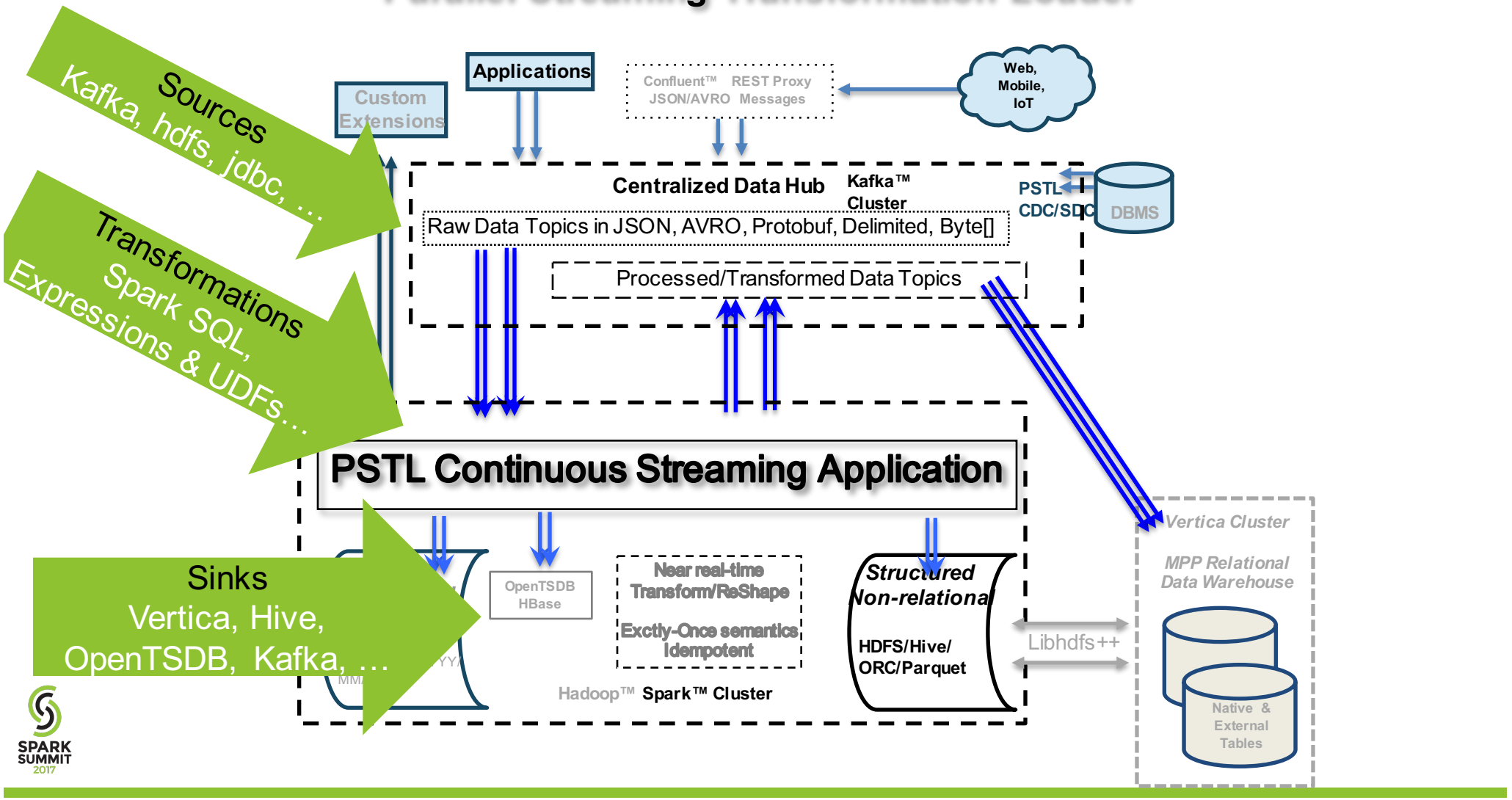
✓ PSTL **removes DevOps complexity** of; high availability, scalability, deployment, autonomous job management, monitoring, and alerting

✓ PSTL provides **all the necessary glue** for complex streaming solutions that would otherwise take many person months of architecture/design, development, testing, and operational infrastructure

✓ PSTL is **a no-code needed** Spark Structured Streaming App with support for Change Data Capture / Slowly Changing Dimensions

✓ PSTL is a ***highly customizable*** Application framework that enables Developers to create common code to dynamically add functionality in SQL over streaming data sources

Parallel Streaming Transformation Loader



SPARK SUMMIT 2017

```
job {
 structured-streaming {
  sources {
   logs {
    format = kafka
    options {
     kafka.bootstrap.servers = "kafka:9092"
     subscribe = kafkatopic
    } } }
  transforms = """
   CREATE TEMPORARY VIEW transform01 AS
   SELECT topic, partition, offset, timestamp,
     split(cast(value as string), '\\|') values
   FROM logs;
   CREATE TEMPORARY VIEW transform02 AS
   SELECT vhash(values[0], values[1]) uniqueKey,
     values[0] col0, values[1] col1
   FROM transform01; """
  sinks {
   store_vertica {
    format = vertica
    dependsOn = transform02
    options {
     vertica.url = "jdbc:vertica://vertica:5433/db"
     vertica.table = "public.logs"
     vertica.property.user = dbadmin
     vertica.property.password = changeMe!
     kafka.topic = vertica-logs
     kafka.property.bootstrap.servers = "kafka:9092"
    } } } } }
```

Each job is a simple config file of Sources, Transforms & Sinks

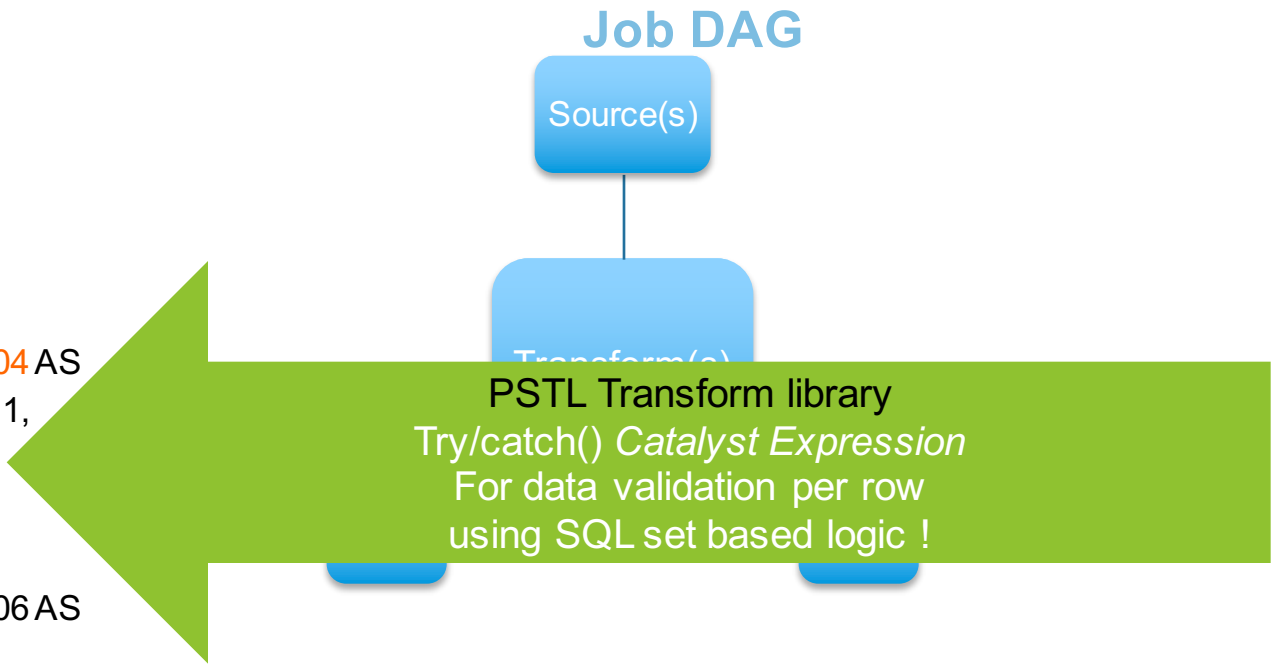
We also support static sources (e.g., hdfs) refreshed on time intervals

Additional Transforms (*Catalyst Expressions*)
fprotobuf(), favro(), fconfluent(), fJSON()
tavro(), try(), vhash(vertica hash), hive udf(), …

Additional sinks
Kafka, S3, hdfs (ORC, Parquet), OpenTSDB/Hbase, console, …

bin/pstl-jobs --initial-contacts 127.0.0.1:2552 --start --job-id job.conf

```
job {
  structured-streaming {
    sources {
      logs {
        format = kafka
        … }   }   }
    transforms = """
  …
CREATE TEMPORARY VIEW transform04 AS
  SELECT  topic, partition, offset, col0, col1,
   try(ThrowOnOdd(offset)) try_result
  FROM transform03
…
CREATE TEMPORARY VIEW transform06 AS
  SELECT
  topic, partition, offset, col0, col1
  , uniqueKey
  , try_result.value, try_result.isSuccess
  , try_result.isFailure, try_result.failure.message
  , try_result.failure.stackTrace
  FROM transform04
  WHERE try_result.isFailure = true
 """
    sink1, sink2, …}   } } } }
```



```
job {
  structured-streaming {
    sources {
      logs { format = kafka … }    }    }

    transforms = """
     CREATE TEMPORARY VIEW transform01 AS
 SELECT
 topic, partition, offset, kafkaMsgCreatedAt
, structCol.*, posexplode(anArray)
 FROM (
   SELECT
   topic, partition, offset, timestamp as kafkaMsgCreatedAt,

favro('{"type":"record","name":"testAllAvroData","namespace":"com.mynamespace","fields":[{"name":"c1_nul
l","type":["null","string"]},…
,{"name":"anArray","type":{"type":"array","items":{"type":"record","name":"anArray","fields":[{"name":"a_name
","type":"string"…'
, value) as structCol

    FROM logs)

"""
  sinks  \  \  \  \  \
```

PSTL catalyst expression
favro(deserialize kafka value Array[Byte])

Thank You

Stop by the Vertica Booth #403

Jack Gudenkauf
Senior Architect
HPE Big Data Professional Services

https://twitter.com/_JG
https://github.com/jackghm

Backup


Grafana - PSTL
JackG
pstl.kafka.query.duration{request=KafkaConsumer.partitionsFor}
pstl.kafka.query.duration{request=KafkaConsumer.seekToEnd}
Kafka Write Duration
14 s
12 s
10 s
8 s
6 s
4 s
2 s
13:14
13:15
13:16
13:17
13:18
pstl.kafka.write.duration
Metrics Write Duration
10.0 s
9.0 s
8.0 s
7.0 s
6.0 s
5.0 s
4.0 s
3.0 s
2.0 s
pstl.metric.write.duration
Batch Duration
1.0 min
50 s
40 s
30 s
20 s
10 s
pstl.batch.duration
HDFS Write Duration
20 s
15 s
10 s
5 s
0 ms
13:13:30
13:14:00
13:14:30
13:15:00
13:15:30
13:16:00
13:16:30
13:17:00
13:17:30
13:18:00
pstl.avro.write.duration
pstl.orc.write.duration
+ ADD ROW

Spark DataFrame to/from Avro

```
// Not needed for no-code PSTL jobs, but
//     a use case: Using PSTL expression library in a Spark Kafka Producer App

// serialize the DataFrame to an Avro encoded byte array
FunctionRegistryOps(spark).register(AvroSerializeFunction.registration)
val df_Avro = spark.sql("select tavro(*) as value from(select * from df)")

// debug
df_Avro.show(5, truncate = false)

df_Avro.createOrReplaceTempView("avrobytes")

FunctionRegistryOps(spark).register(AvroDeserializeFunction.registration)
val df1 = spark.sql(s"select favro('$avroSchema', value) as structCol from
avrobytes")
```