FROM KAFKA TO BigQuery

A Guide For Delivering Billions of Daily Events

Ofir Sharony
Why are we here?

As data engineers, our goal is building **GREAT** data pipelines.

Delivering data to analysis **reliably & efficiently** with the ability to **scale**.

This session is based [this post](https://medium.com/myheritage-engineering) from MyHeritage tech blog: [https://medium.com/myheritage-engineering](https://medium.com/myheritage-engineering)
Considerations for building data pipelines

- End2End latency
- Data structure
- Data partitioning
- Reprocessing old data
Case Study
MyHeritage data pipeline

Discover, preserve & share your FAMILY HISTORY

Reveal the secrets from your DNA
User behavior and A/B testing
Website Performance & Health
High level architecture

- Database
- Requests from multiple sources
- Change-data-capture events
- User Activities
- Apache Kafka
- Google BigQuery

MyHeritage
Apache Kafka in a nutshell

- Distributed streaming platform
- Highly scalable
- Fault tolerant
- Topics and partitions
Google BigQuery in a nutshell

- Cloud data-warehouse
- Pay as you go
- Data sources
- Streaming support
- Daily partitioning feature. Partitioning options:
  - By processing time - when the event was observed
  - By event time - when the event actually occurred
Our Data Delivery Journey
Take 1 - Batch loading to Google Cloud Storage
Batch loading to Google Cloud Storage

- Two steps for loading
  - Batch loading to GCS with **Secor**
  - Loading from GCS to BigQuery

```
bq load dataset.table$20171201 gs://bucket/topic/dt=2017-12-01/*
```
Tradeoffs - Batch loading with Secor

**Pros**
- Partitioning options
  - By ingestion date (processing time)
  - By record timestamp (event time)
- Concurrency and scaling
- No extra cost for streaming

**Cons**
- Data formats
- End2End latency
Take 2 - Streaming data via BigQuery API
Streaming data via BigQuery API

- No preliminary step
- Comes with a cost and quota limitation
Kafka Streams streaming framework

- Streaming applications on top of Kafka
- Simple Java application
- Auto-replicated application state
- Supports both processing time and event time semantics
Basic Kafka Streams application

```java
public static void main(String[] args) {

    Properties config = new Properties();
    config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "host:port");
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, "AppId");

    KStreamBuilder builder = new KStreamBuilder();
    builder.stream(Serdes.String(), Serdes.String(), "my_topic")
        .filter((key, value) -> value.equals("Please, take me to BigQuery"))
        .foreach((key, value) -> deliverDataToBigQuery(key, value));

    KafkaStreams streams = new KafkaStreams(builder, config);
    streams.start();
}
```
Tradeoffs - Streaming with BigQuery API

Pros

• Low latency
• Complete control in your hands

Cons

• You have to design a robust service
• Handle streaming errors, record batching, etc
Streaming with Kafka Connect

- Deliver data in and out of Kafka
- Supports many connectors out of the box
- Simple connector creation - we used WePay’s BigQuery connector
- You get redundancy and scaling for free
Tradeoffs - Streaming with Kafka Connect

**Pros**
- Plug and play solution
- Leverages Kafka for scaling and auto-failover
- Auto creation of BigQuery tables

**Cons**
- Partition by processing time
- Process data only from Kafka
- Topic to table mapping
Streaming data with Apache Beam

- Unified model for batch and streaming
- Portable platform
- Managed Cloud Dataflow runner
String getDestination(ValueInSingleWindow<PageView> element) {
    String table = element.getValue().isBot() ? "bot_pageviews" : "user_pageviews";

    Long eventTimestamp = element.getValue().getEventTimestamp();
    String partition = new SimpleDateFormat("yyyyMMdd")
                         .format(new Date(eventTimestamp));

    return String.format("%s:%s.%s%$s", project, dataset, table, partition);
}
Tradeoffs - Streaming with Beam

Pros:
- Unified model for multiple runners
- Dataflow runner advantages
- Can process data from different sources
- Dynamic destinations

Cons:
- Extra cost of running managed workers
- Not a part of Kafka ecosystem
- New kid in the block
<table>
<thead>
<tr>
<th></th>
<th>Loading in BATCH</th>
<th>Loading in STREAMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery time</td>
<td>Delayed</td>
<td>Almost instant, near real-time</td>
</tr>
<tr>
<td>Cost</td>
<td>Free (Pay only for storage)</td>
<td>Currently $0.05/GB</td>
</tr>
<tr>
<td>Loading steps</td>
<td>GCS -&gt; BigQuery</td>
<td>Directly to BigQuery</td>
</tr>
<tr>
<td>Data partitioning</td>
<td>By event or processing time</td>
<td>API - by event or processing time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connector - by processing time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beam - by event or processing time</td>
</tr>
<tr>
<td>Data reprocessing</td>
<td>Supported</td>
<td>Depends on your streaming framework</td>
</tr>
</tbody>
</table>
Q & A

ofir.sharony@myheritage.com