Collibra Telemetry Backbone

OpenTelemetry and Apache Beam
Alex Van Boxel
Principal System Architect
Collibra

Apache Beam Committer (but you have to forgive me, it’s been a while…)

Google Developer Expert
Built to **connect** to the data ecosystem
Telemetry

What is it
Metrics
Logs
OpenTelemetry

An observability framework for cloud-native software.

OpenTelemetry is a collection of tools, APIs, and SDKs. You use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis in order to understand your software's performance and behavior.
Backbone Goals

Exploring brave new data points
Observability is not a luxury

it should be a **core feature** of a SaaS solution
Vendor Independence

Removing lock-in at the collection side

We should always have the possibility of easily switching backend vendors. Without rolling out vendor dependent agents. OpenTelemetry collector promises vendor independent collection.
Owning our own telemetry data

Only when the protocol is open, can you own the data

OpenTelemetry has an open protocol (defined in Protobuf) and well defined semantic conventions. Only through this openness can you start building on top of the data.
Serving data back to our customers

If you own your data, only then can you serve it back

Taking control and understanding the data you can aggregate and think about serving part of the data back.
Building the backbone

blocks everywhere
OpenTelemetry Collector

Oh, that’s also a pipeline?!
What is the OpenTelemetry Collector

https://github.com/open-telemetry/opentelemetry-collector-contrib
The OpenTelemetry Collector as a receiver
The OpenTelemetry Collector as a exporter
The OpenTelemetry Collector as a processor
The OpenTelemetry Collector as backbone ingress
Telemetry Stream

Versatility little thing
The OpenTelemetry Collector as messaging producer consumer
Beam Pipelines

Power of streams
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>End time</th>
<th>Elapsed time</th>
<th>Start time</th>
<th>Status</th>
<th>SDK version</th>
</tr>
</thead>
<tbody>
<tr>
<td>otlp-egress-logs</td>
<td>Streaming</td>
<td></td>
<td>21 days 40 min</td>
<td>May 10, 2022, 11:43:11 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>features-collection</td>
<td>Streaming</td>
<td></td>
<td>27 days 16 hr</td>
<td>May 3, 2022, 8:18:11 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>otlp-egress-traces</td>
<td>Streaming</td>
<td></td>
<td>27 days 16 hr</td>
<td>May 3, 2022, 7:59:06 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>sla-collection</td>
<td>Streaming</td>
<td></td>
<td>27 days 17 hr</td>
<td>May 3, 2022, 7:01:35 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>otlp-backup</td>
<td>Streaming</td>
<td></td>
<td>32 days 4 hr</td>
<td>Apr 29, 2022, 8:14:43 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-openapi-to-bigquery</td>
<td>Streaming</td>
<td></td>
<td>32 days 4 hr</td>
<td>Apr 29, 2022, 8:13:25 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>classifier</td>
<td>Streaming</td>
<td></td>
<td>32 days 4 hr</td>
<td>Apr 29, 2022, 8:06:39 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>otlp-egress-metrics-to-elastic</td>
<td>Streaming</td>
<td></td>
<td>32 days 6 hr</td>
<td>Apr 29, 2022, 6:17:45 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-openapi-to-bigtable</td>
<td>Streaming</td>
<td></td>
<td>32 days 6 hr</td>
<td>Apr 29, 2022, 6:13:25 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>customer-logs</td>
<td>Streaming</td>
<td></td>
<td>32 days 6 hr</td>
<td>Apr 29, 2022, 6:09:53 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-openapi</td>
<td>Streaming</td>
<td></td>
<td>32 days 16 hr</td>
<td>Apr 28, 2022, 8:12:24 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-state</td>
<td>Streaming</td>
<td></td>
<td>32 days 16 hr</td>
<td>Apr 28, 2022, 7:42:04 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-javaapi-to-bigquery</td>
<td>Streaming</td>
<td></td>
<td>35 days 21 hr</td>
<td>Apr 25, 2022, 2:39:31 PM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>calculated-javaapi-to-bigtable</td>
<td>Streaming</td>
<td></td>
<td>39 days 53 min</td>
<td>Apr 22, 2022, 11:30:04 AM</td>
<td>Running</td>
<td>2.38.0</td>
</tr>
<tr>
<td>environment-state</td>
<td>Streaming</td>
<td></td>
<td>53 days 17 hr</td>
<td>Apr 7, 2022, 6:36:25 PM</td>
<td>Running</td>
<td>2.35.0</td>
</tr>
<tr>
<td>beats-backup</td>
<td>Streaming</td>
<td></td>
<td>166 days 23 hr</td>
<td>Dec 15, 2021, 11:25:33 AM</td>
<td>Running</td>
<td>2.34.0</td>
</tr>
</tbody>
</table>
Apache Beam as attribute enricher

- A resource can be uniquely identified, and should have enough attributes at collection time to make it useful for observability systems.
- Adding extra attributes could be interesting for analytical systems, example:
  - tenant id
  - environment type
Apache Beam as attribute enrichter

- Adding extra attributes can be easier in post, then deploying them on thousands of machines
- A special case in the same class: trace sampling… we same at 100% for analytical purposes. We don’t want to get billed for all our spans
Apache Beam as attribute enricher, why not in the collector?

- Attributes in the infrastructure could be managed by different teams (collection time)
- Collector also has a pipelines, this could be and easier one, but doing it in the Beam pipeline has the advantage of running on historical data in batch
Apache Beam as backup

- If you want to run on historical data, you need to start backing up your stream.
- We started backing up before the OpenTelemetry spec had a file format available, so we use CloudEvent spec.
- Window per 15 minutes and use the standard AvroIO from Beam (CloudEvent has a Avro spec, we pack the proto in an Avro container).
Apache Beam as backup
Apache Beam as backup, why not in the collector?

- Same reason as enrichment, the build up of reusable component
- As the CloudEvent spec allowed to mix types (metrics, traces and logs) we did this, but changed to different files per type
Apache Beam as a analysis pipeline

Most analytical use-cases that come up are centered around usage. In our case API usage, but slowly other type of usage. We use both:

- traces
- logs (structured)

Try to avoid teams creating metrics to track usage, they lose information through aggregation.
Apache Beam as a analysis pipeline, use-case API Usage
Apache Beam as a analysis pipeline, use-case API

Usage

Based on traces, each span that is relevant a SQL row is extracted.

```java
public static Schema SCHEMA = Schema.builder()
    .addStringField("trace_id")
    .addNullableField("trace_start", FieldType.DATETIME)
    .addNullableField("trace_duration", FieldType.INT64)
    .addNullableField("trace_name", FieldType.STRING)
    .addNullableField("service_name", FieldType.STRING)
    .addNullableField("service_version", FieldType.STRING)
    .addNullableField("salesforce_id", FieldType.STRING)
    .addNullableField("environment_name", FieldType.STRING)
    .addNullableField("host_name", FieldType.STRING)
    .addNullableField("span_name", FieldType.STRING)
    .addNullableField("function_name", FieldType.STRING)
    .addNullableField("span_duration", FieldType.INT64)
    .addNullableField("http_user_agent", FieldType.STRING)
    .addNullableField("has_ui_rendering", FieldType.BOOLEAN)
    .build();
```
Apache Beam as a analysis pipeline, use-case API Usage
Apache Beam as a **analysis pipeline**, calculated **-metrics**

Metrics can be created from traces and logs, into the Beam pipeline. It’s like feature extraction, something that Apache Beam is very good at.

Three use-cases of **calculated-metrics**:

- calculated-openapi
- calculated-javaapi
- calculated-state

All end up on a **dedicated** Pubsub topic
Apache Beam as a analysis pipeline, calculated -metrics (openapi)

Proxy logs (ApacheD, NGNX, Envoy), have detail enough to reverse engineer the operationId from the OpenAPI spec.
Apache Beam as a analysis pipeline, calculated -metrics (openapi)

- The proxy logs are OTLP logs
- Convert them to spans, because logs don’t have a semantic convention yet, so we use the Semantic conventions for HTTP spans
- Then we create Semantic Conventions for HTTP Metrics out of the spans
  - duration
  - request size
  - response size
Apache Beam as a classifier

All those calculated-metrics are put to good use, not only do they go the observability tools, they are used to create feature vectors.

- Different metrics are grouped together in **different window sizes** (1m, 5m and 15m)
- The vector is used to create **CloudEvents** (this could be an alert)
Serving observability data back to the product

The same OTLP types are also easy to store into Bigtable. OTLP structs are easy to store as is, and easy to work with for real time aggregations.
Conclusion

and learnings
Learning process

- Protobuf (OTLP) in
- Out:
  - Protobuf (OTLP)
  - BigQuery/Bigtable/Elastic
  - CloudEvent
- Developed reusable model for all pipelines (internal Protobuf replaces Row based)
What would we do different?

- As the engineerings in the operations we would now start investigating the Go SDK (two years ago it was too early)
- Some parts would be a better fit for the opentelemetry-collector (pipeline), switching to the Go SDK maybe makes it easier to share code.
Thank you

Questions?