Collibra

Collibra Telemetry Backbone

OpenTelemetry and Apache Beam

Alex Van Boxel

Principal System Architect

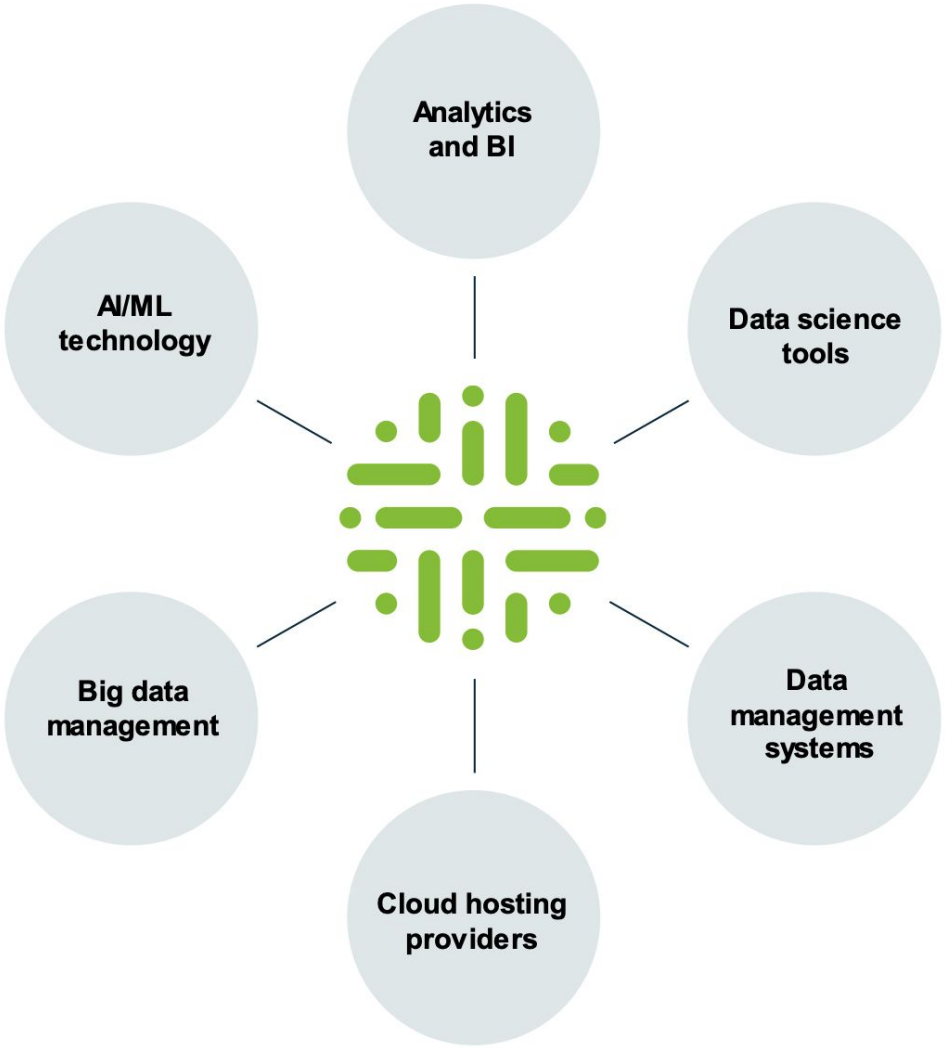
Collibra

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

Photo by Mitchel Boot on Unsplash

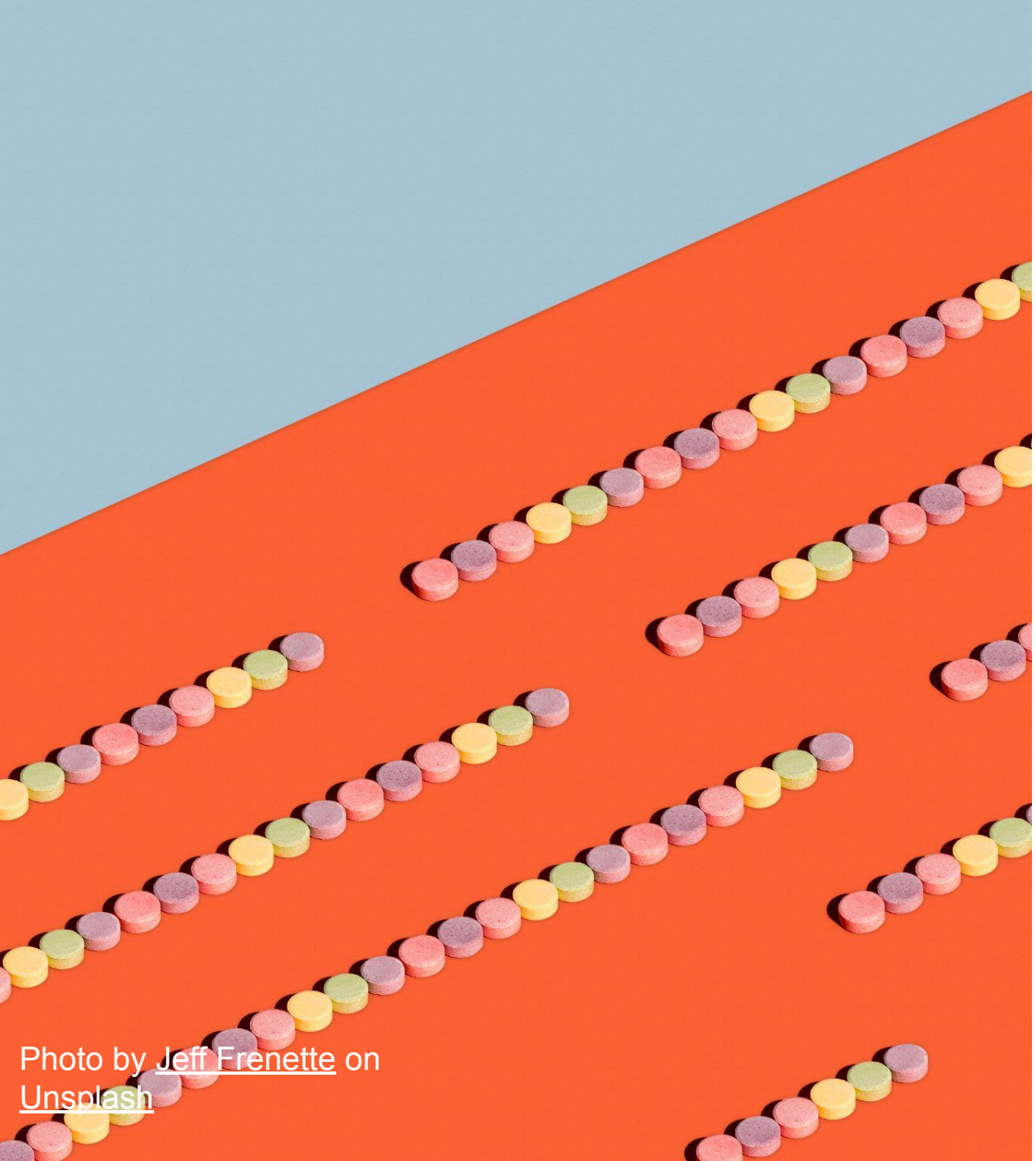
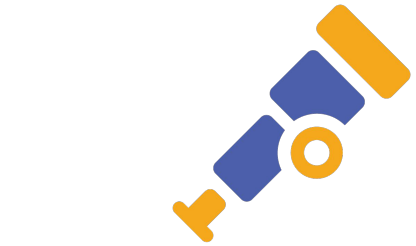


Photo by Jeff Frenette on Unsplash

Logs

Traces

OpenTelemetry



OpenTelemetry

An observability framework for cloud-native software.

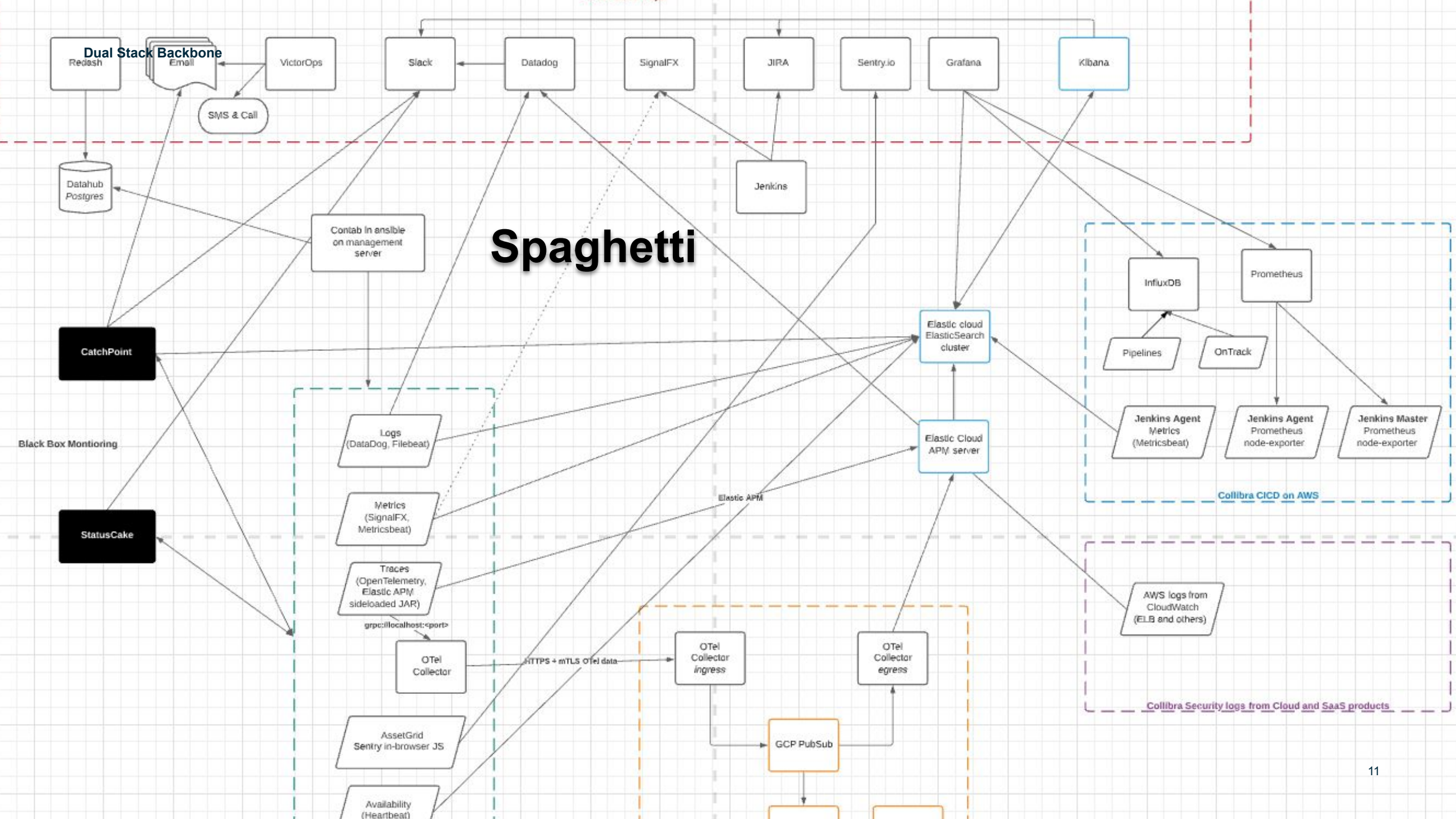
OpenTelemetry is a collection of tools, APIs, and SDKs. You use it to instrument, generate, collect, and export telemetry data (metrics, logs, and **traces**) for analysis in order to understand your software's performance and behavior.

Backbone Goals

Exploring brave new data points

Observability is not a luxury

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


Dual Stack Backbone
Redash
Email
VictorOps
SMS & Call
Slack
Datadog
SignalFX
JIRA
Sentry.io
Grafana
Kibana
Datahub
Postgres
Jenkins
Contab in ansible
on management
server
Spaghetti
CatchPoint
Black Box Montioring
StatusCake
Logs
(DataDog, Filebeat)
Metrics
(SignalFX,
Metricsbeat)
Traces
(OpenTelemetry,
Elastic APM
sideloaded JAR)
grpc://localhost:<port>
OTel
Collector
AssetGrid
Sentry in-browser JS
Availability
(Heartbeat)
HTTPS + mTLS OTel data
Elastic cloud
ElasticSearch
cluster
Elastic Cloud
APM server
Elastic APM
InfluxDB
Prometheus
Pipelines
OnTrack
Jenkins Agent
Metrics
(Metricsbeat)
Jenkins Agent
Prometheus
node-exporter
Jenkins Master
Prometheus
node-exporter
Collibra CICD on AWS
AWS logs from
CloudWatch
(ELB and others)
Collibra Security logs from Cloud and SaaS products
OTel
Collector
ingress
OTel
Collector
egress
GCP PubSub

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.

Collibra

Building the backbone

blocks everywhere

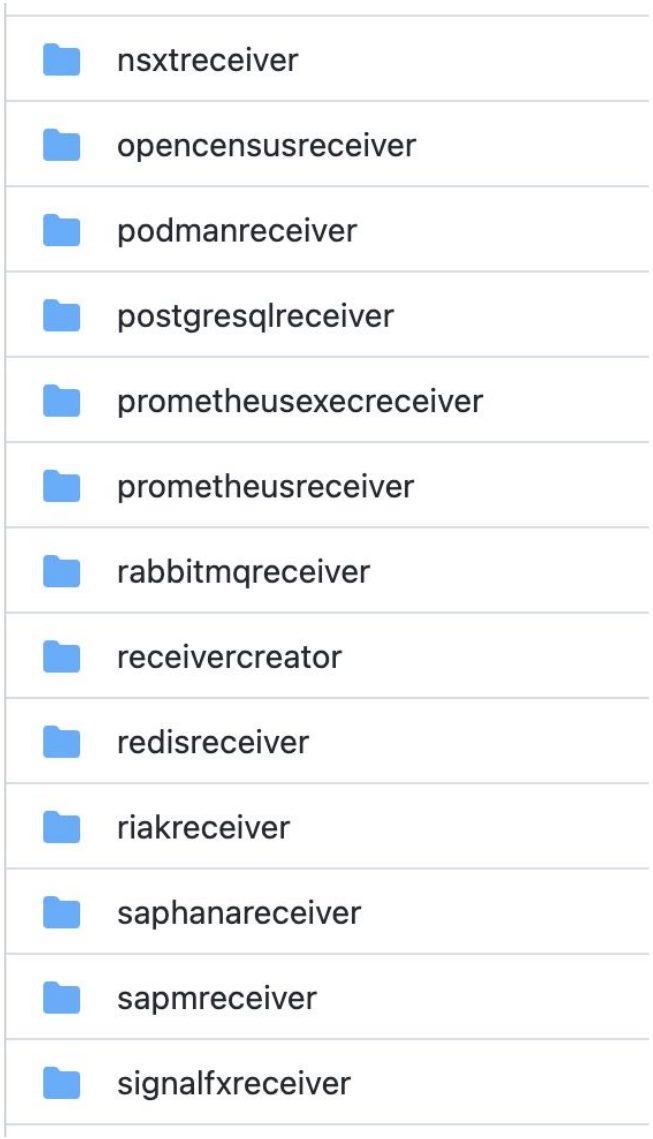
OpenTelemetry Collector

Oh, that’s also a pipeline?!

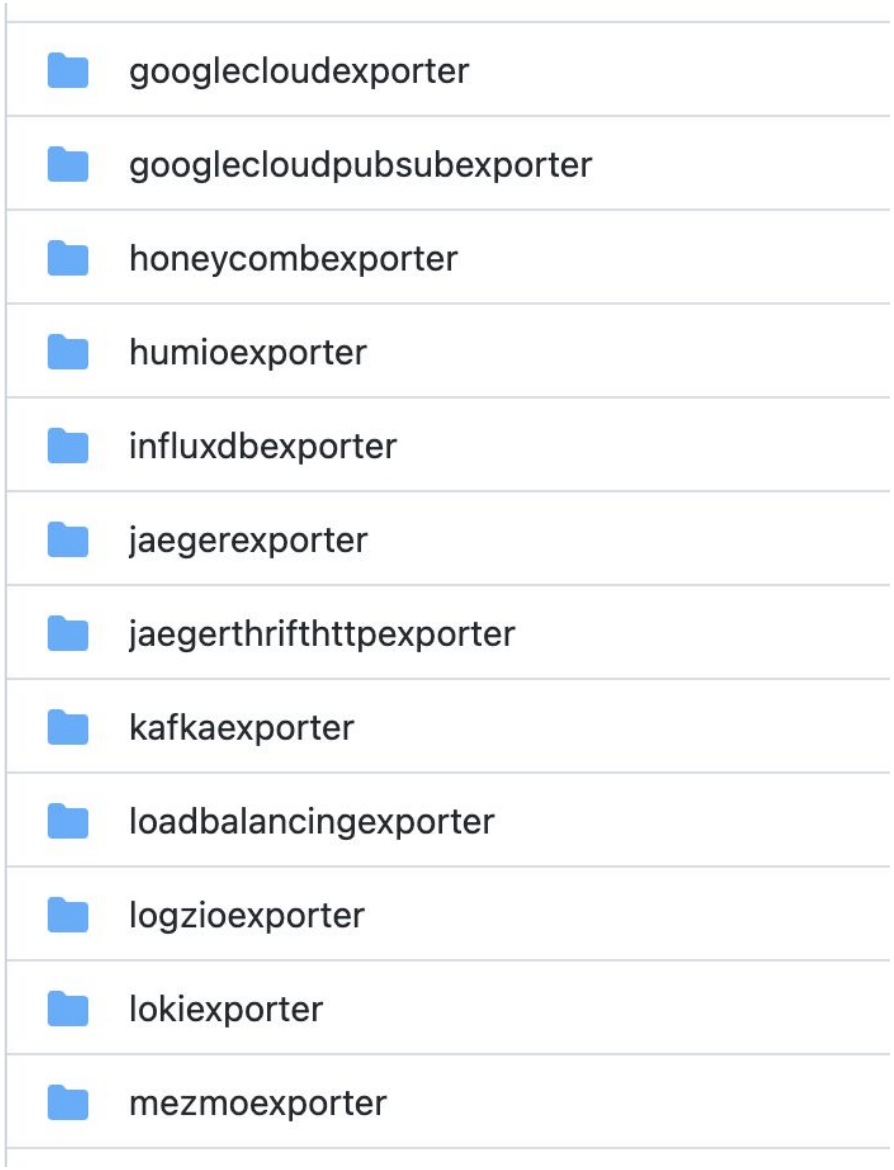
What is the OpenTelemetry Collector

https://github.com/open-telemetry/opentelemetry-collector-contrib

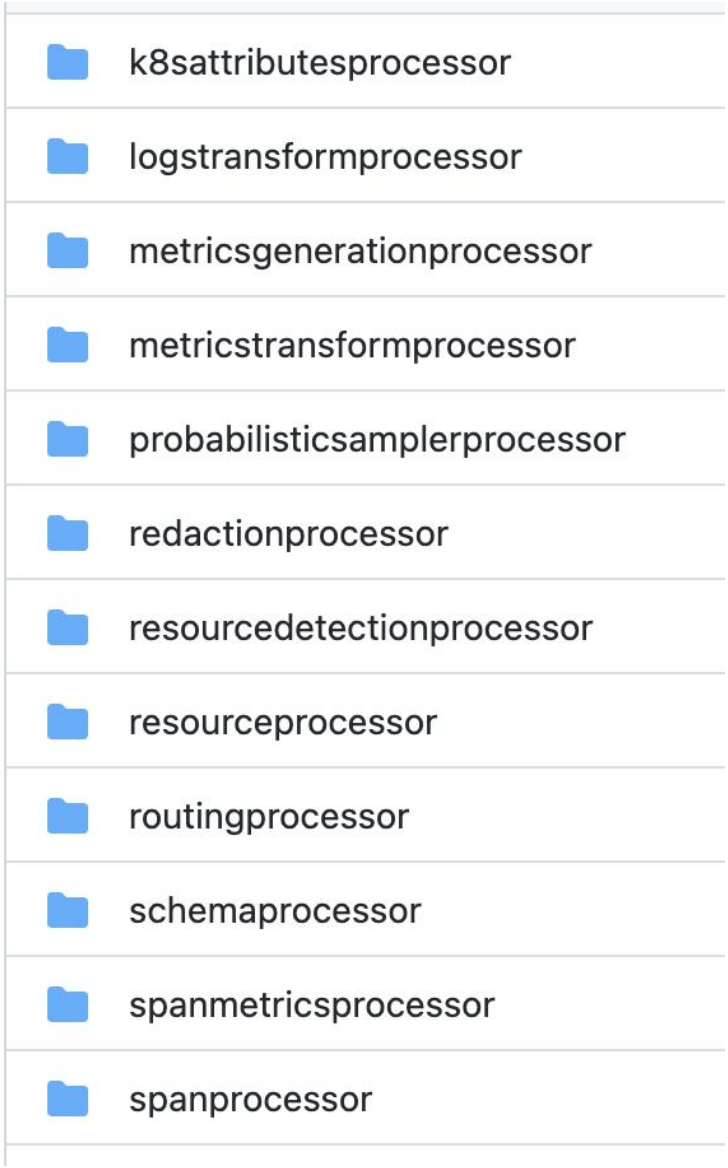
The OpenTelemetry Collector as a receiver

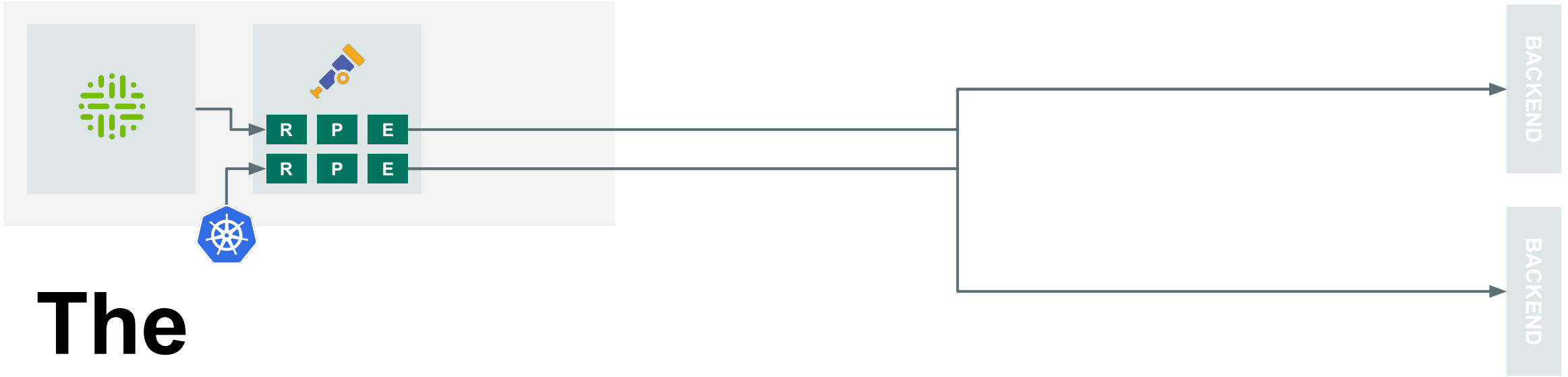


The OpenTelemetry Collector as a exporter



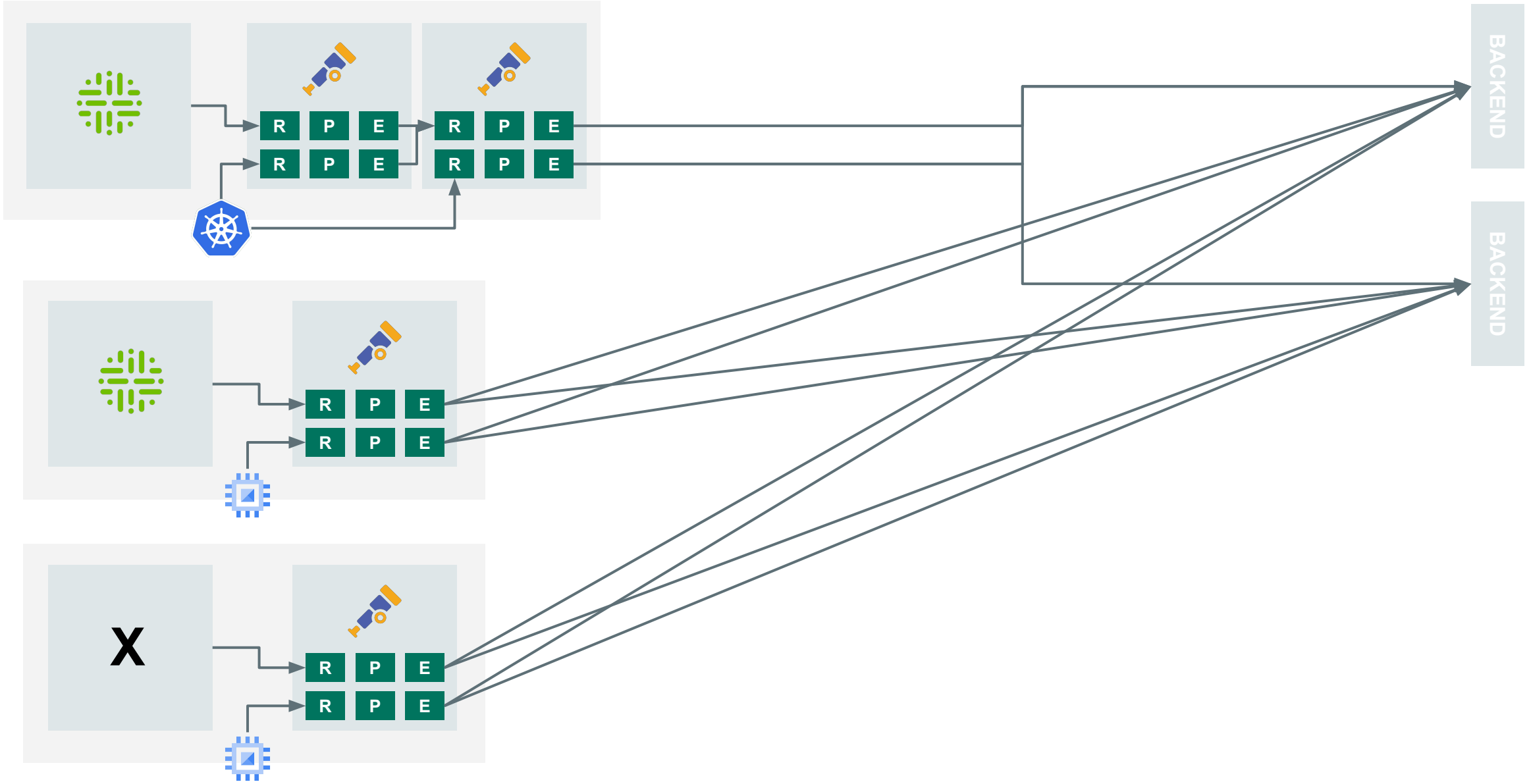
The OpenTelemetry Collector as a processor

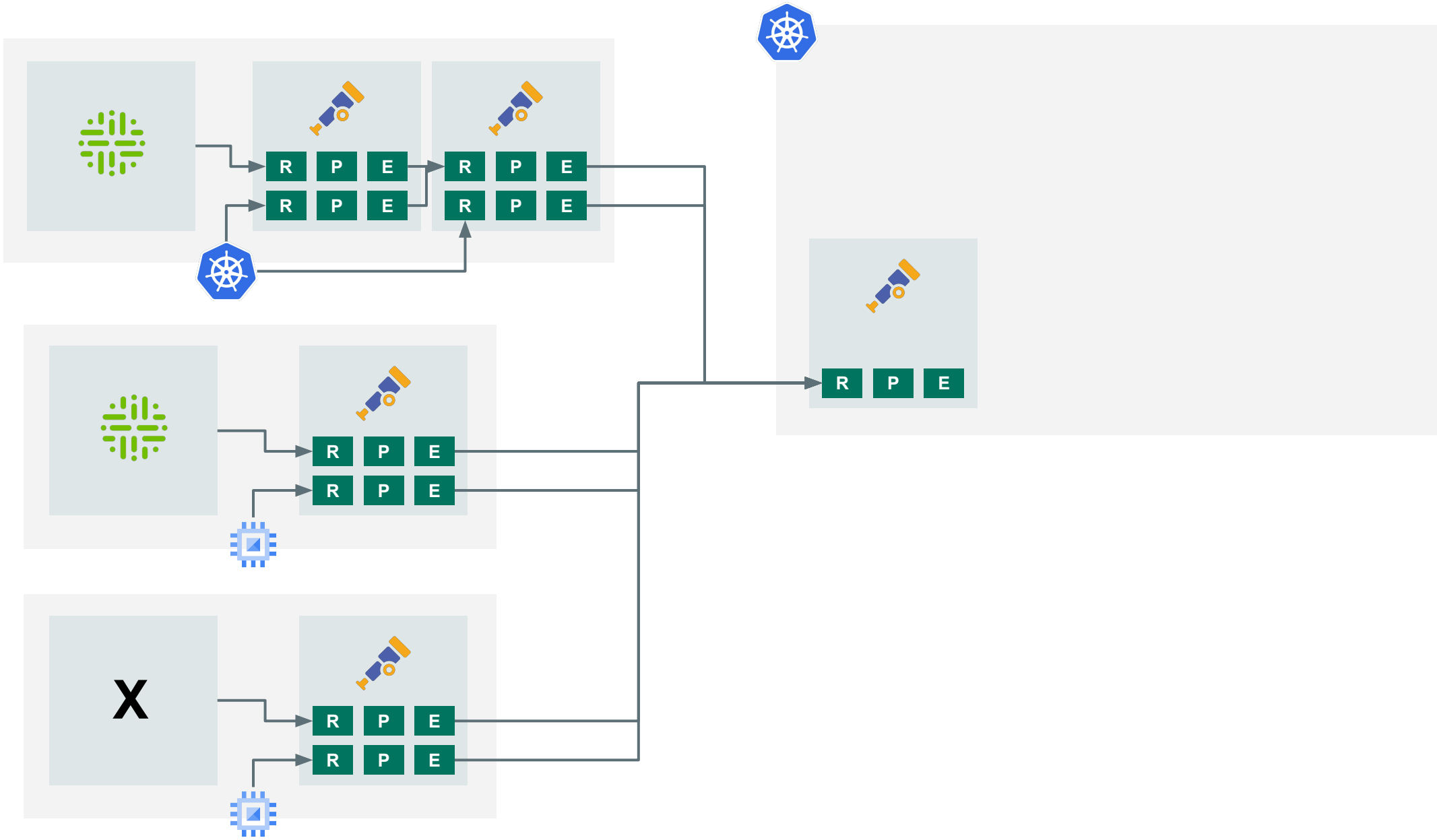




The OpenTelemetry Collector as backbone ingress

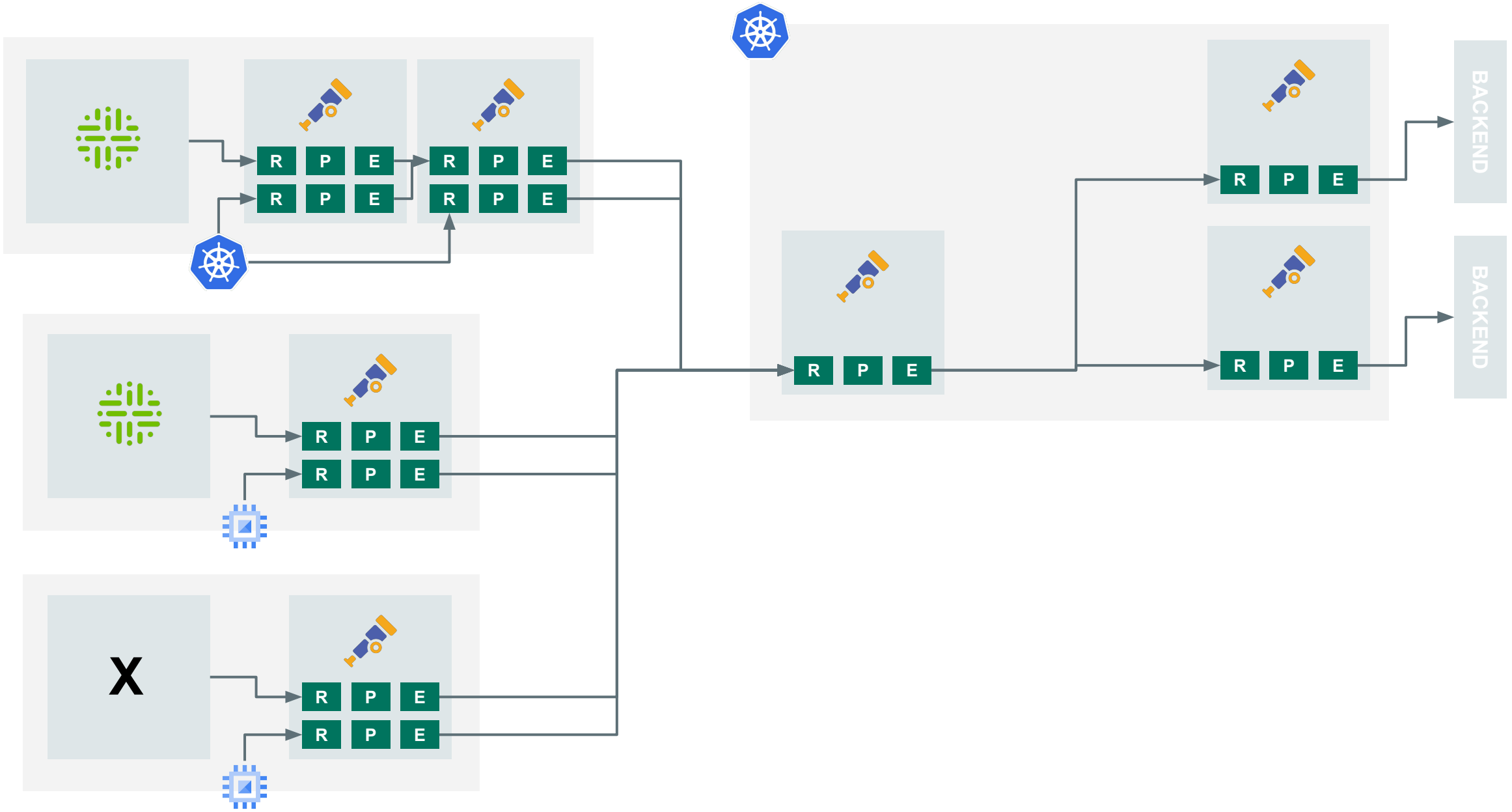

R
P
E
R
P
E
R
P
E
R
P
E
BACKEND
BACKEND


R
P
E
R
P
E
R
P
E
R
P
E
R
P
E
R
P
E
R
P
E
R
P
E
X
BACKEND
BACKEND


R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
X

Telemetry Stream

Versatility little thing


R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
X
BACKEND
BACKEND

The OpenTelemetry Collector as messaging producer consumer

README.md

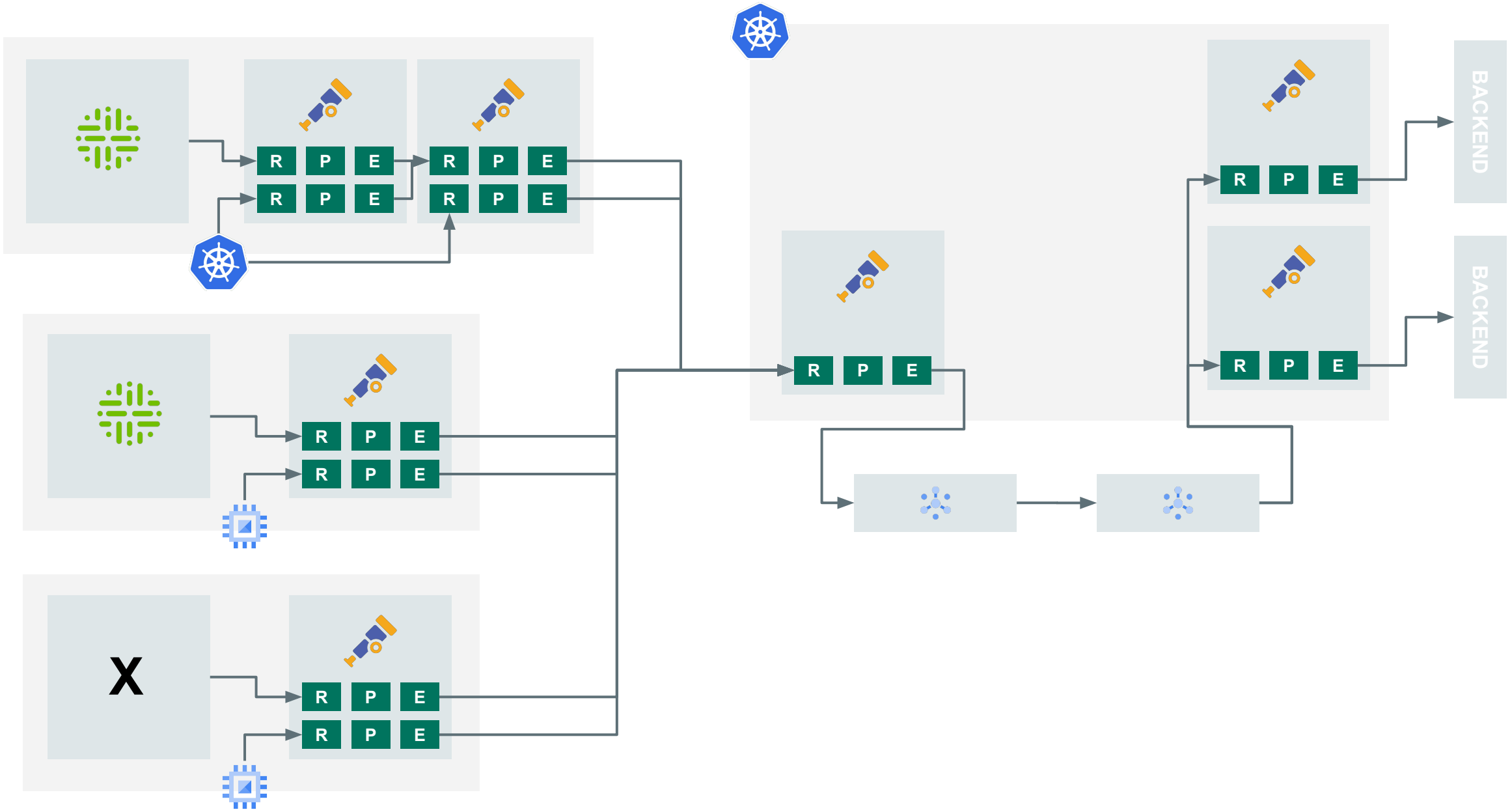
Google Cloud Pubsub Exporter

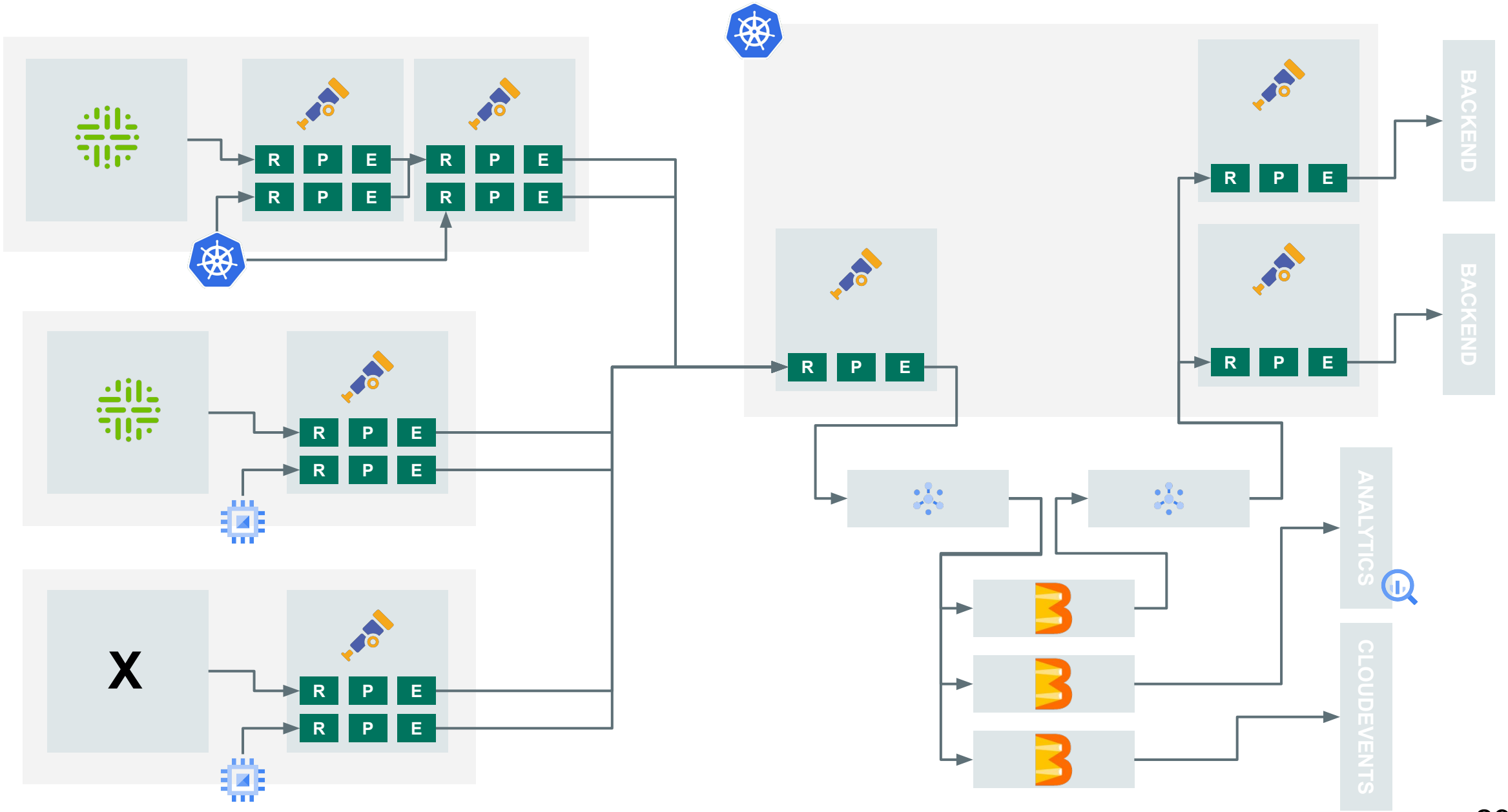
> ⚠️ This is a community-provided module. It has been developed and extensively tested at Collibra, but it is not officially supported by GCP.

This exporter sends OTLP messages to a Google Cloud Pubsub topic.

The following configuration options are supported:

- `project` (Optional): The Google Cloud Project of the topics.
- `topic` (Required): The topic name to receive OTLP data over. The topic name should be a fully qualified resource name (eg: `projects/otel-project/topics/otlp`).
- `compression` (Optional): Set the payload compression, only `gzip` is supported. Default is no compression.
- `watermark` Behaviour of how the `ce-time` attribute is set (see watermark section for more info)
  - `behavior` (Optional): `current` sets the `ce-time`


R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
X
BACKEND
BACKEND

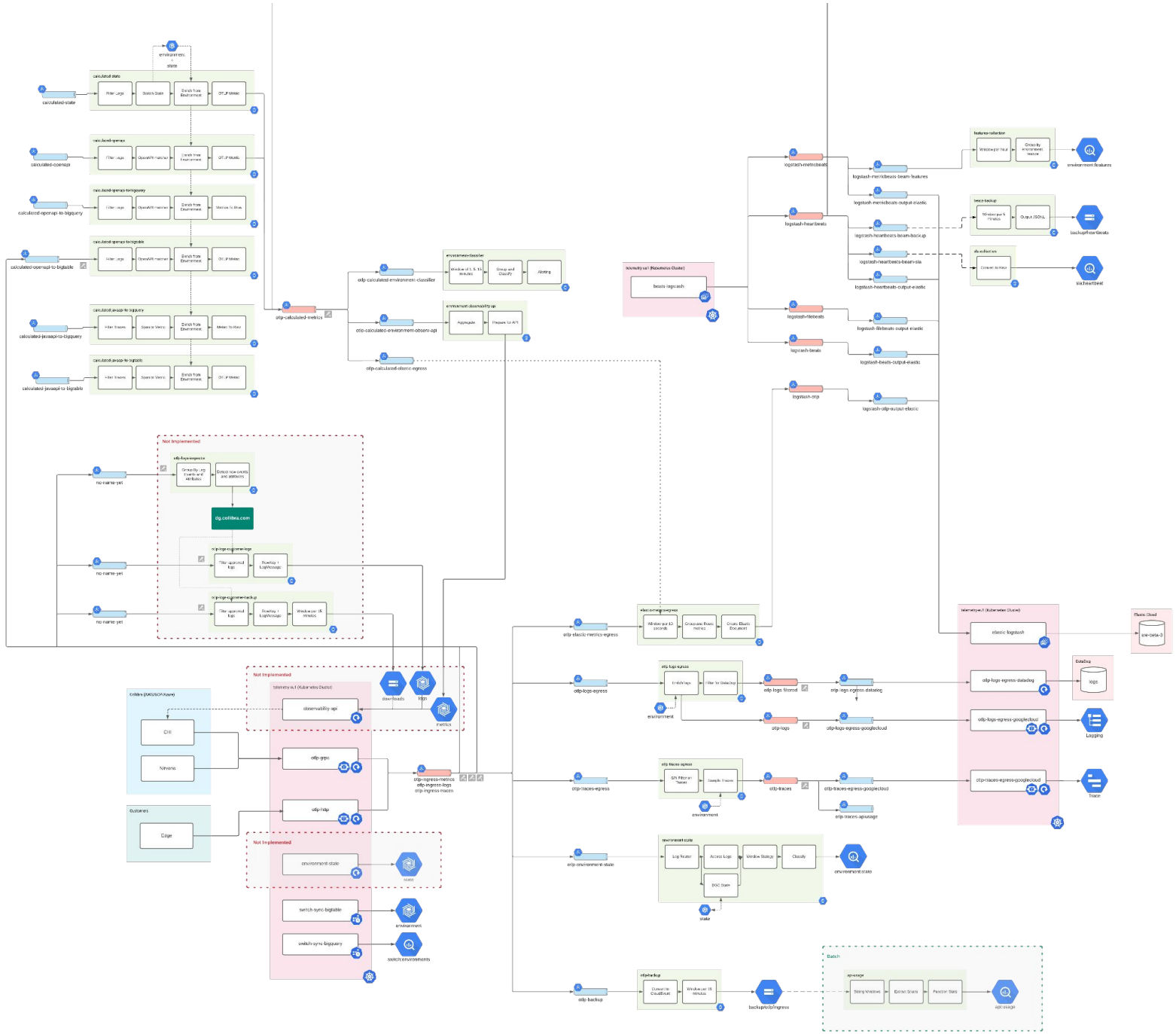

R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
R P E
X
BACKEND
BACKEND
ANALYTICS
CLOUDEVENTS

Collibra

Beam Pipelines

Power of streams

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



Apache Beam as attribute enrichter

- 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

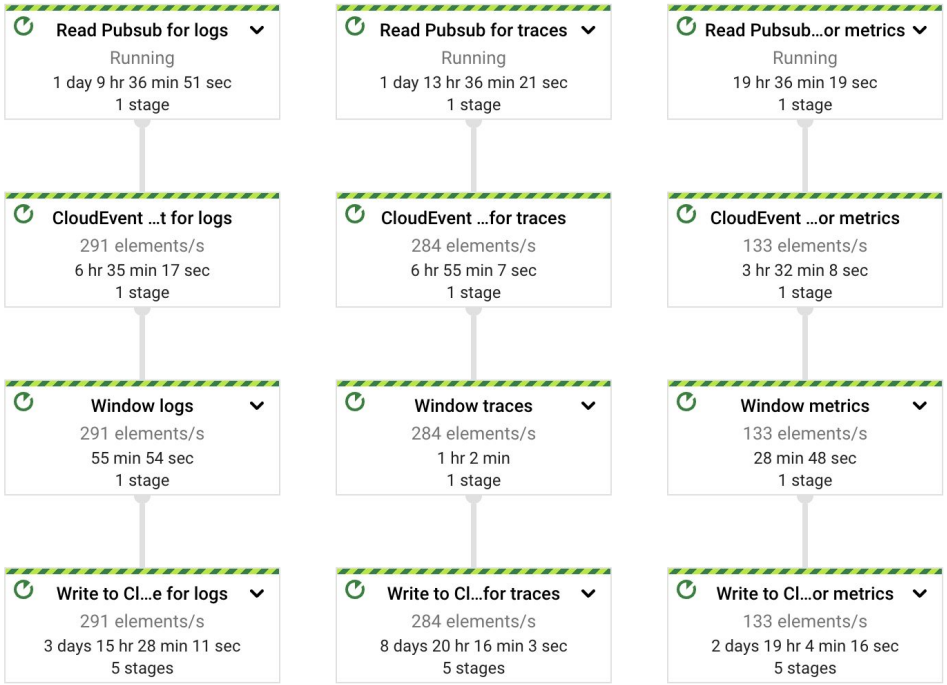
Apache Beam as attribute enrichter, why not in the collector?

- 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** availabe, 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 r**eusable component**
- As the CloudEvent spec allowed to mix types (metrics, traces and logs) we did this, but changed to different files **per type**

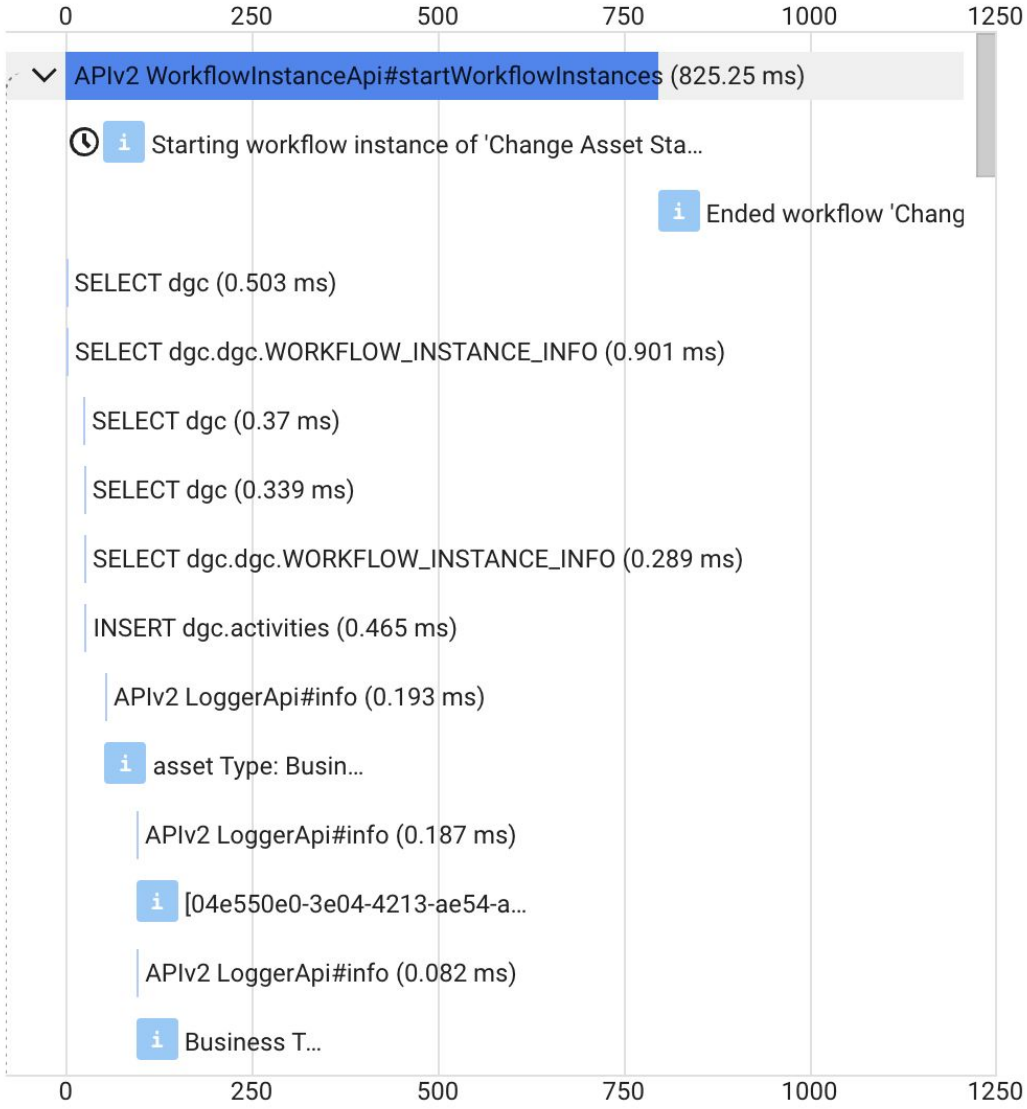
Apache Beam as a analysis pipeline

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



Collibra

Apache Beam as a analysis pipeline, use-case API Usage

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

```
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

```
SELECT CAST(trace_start AS DATE)         AS trace_date,
       func.function_name,
       trace.trace_name,
       trace.has_ui_rendering,
       USER_AGENT_NAME(trace.user_agent) AS user_agent_name,
       trace.environment_name,
       trace.service_version,
       SUM(trace.trace_span_count)       AS span_count
FROM (
         SELECT trace_id,
                function_name
         FROM PCOLLECTION
         WHERE function_name IS NOT NULL
         GROUP BY trace_id, function_name
     ) AS func
         LEFT JOIN
     (SELECT trace_id,
             MAX(trace_start)      AS trace_start,
             MAX(trace_name)       AS trace_name,
             MAX(service_name)     AS service_name,
             MAX(service_version)  AS service_version,
             MAX(environment_name) AS environment_name,
             MAX(http_user_agent)  AS user_agent,
             MAX(has_ui_rendering) AS has_ui_rendering,
             COUNT(span_name)      AS trace_span_count
      FROM PCOLLECTION
      GROUP BY trace_id) AS trace
     ON trace.trace_id = func.trace_id
WHERE trace.trace_name IS NOT NULL
  AND service_name = 'dgc-core'
GROUP BY CAST(trace_start AS DATE),
         func.function_name,
         trace.trace_name,
         trace.has_ui_rendering,
         USER_AGENT_NAME(trace.user_agent),
         trace.environment_name,
         trace.service_version
```

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.

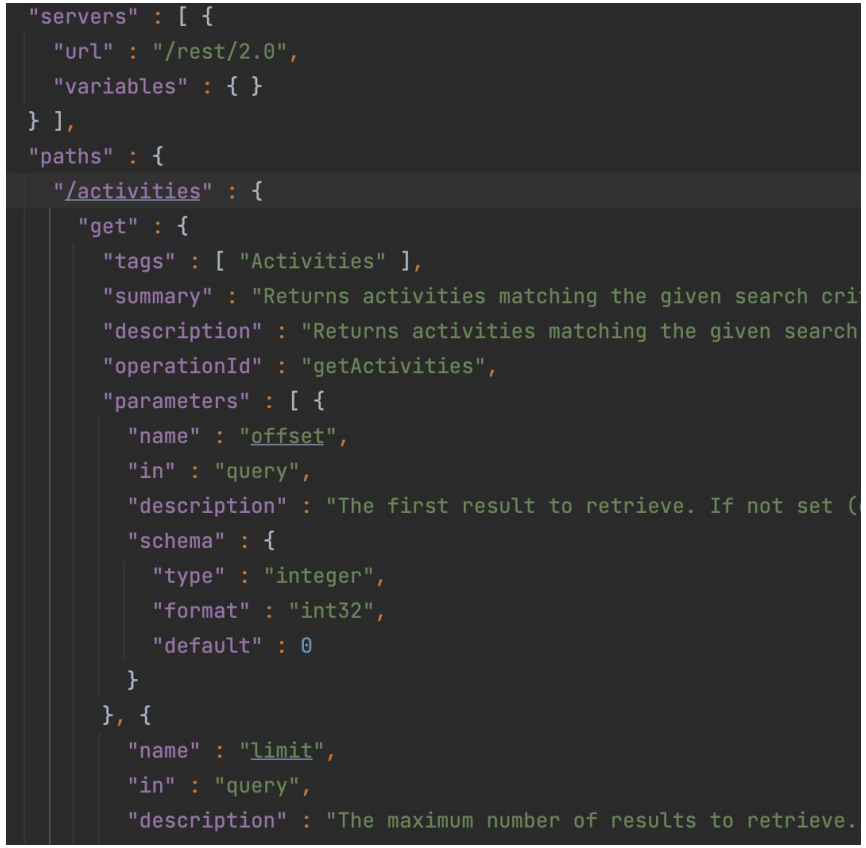
Three use-cases of **calculated-metrics**:

- 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.



```
"servers" : [ {
  "url" : "/rest/2.0",
  "variables" : { }
} ],
"paths" : {
  "/activities" : {
    "get" : {
      "tags" : [ "Activities" ],
      "summary" : "Returns activities matching the given search cri
      "description" : "Returns activities matching the given search
      "operationId" : "getActivities",
      "parameters" : [ {
        "name" : "offset",
        "in" : "query",
        "description" : "The first result to retrieve. If not set (
        "schema" : {
          "type" : "integer",
          "format" : "int32",
          "default" : 0
        }
      }, {
        "name" : "limit",
        "in" : "query",
        "description" : "The maximum number of results to retrieve.
```

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

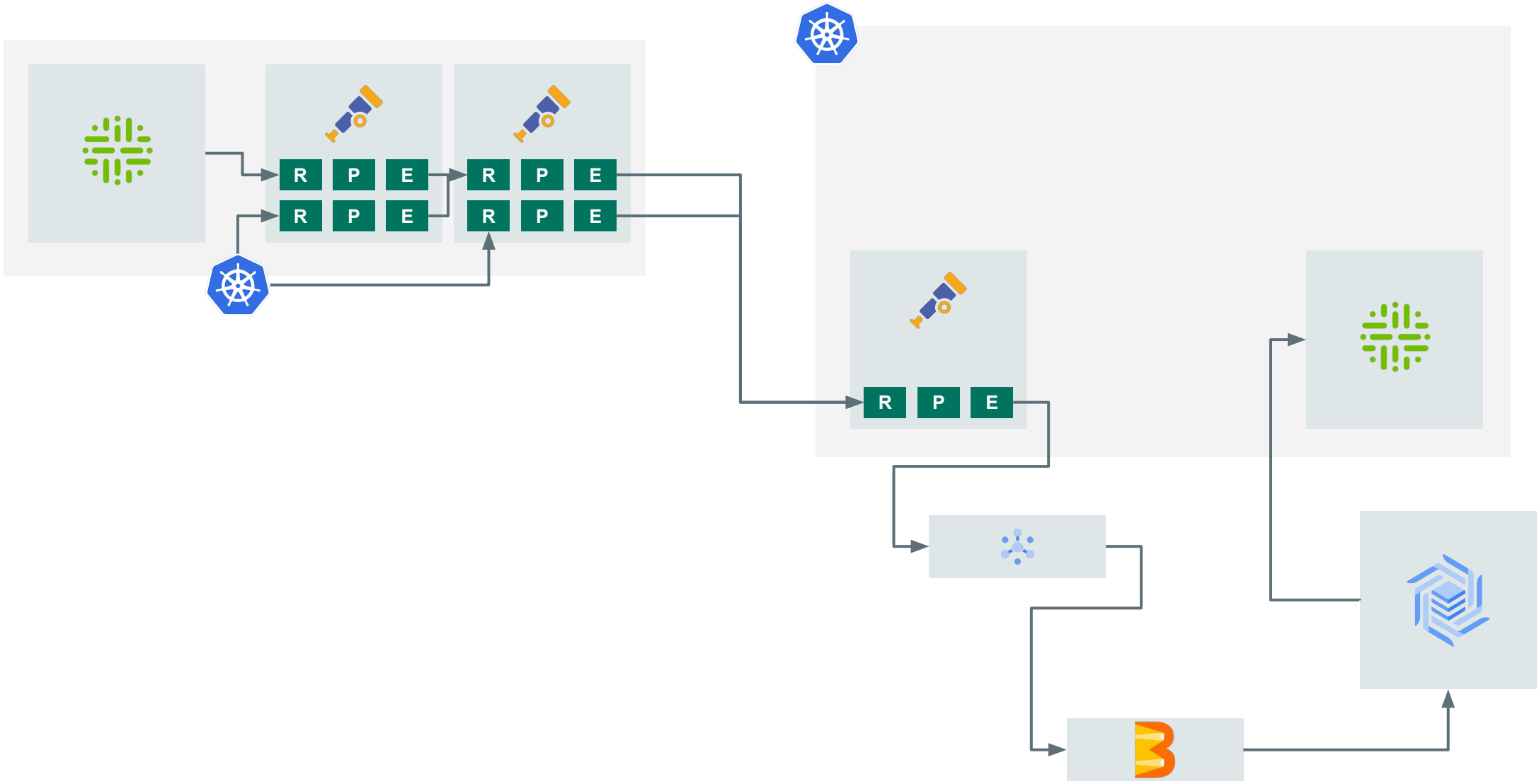
All those calculated-metrics are put to good use, not only do they go the the observability tools, they are used to create feature vectors.

- 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.

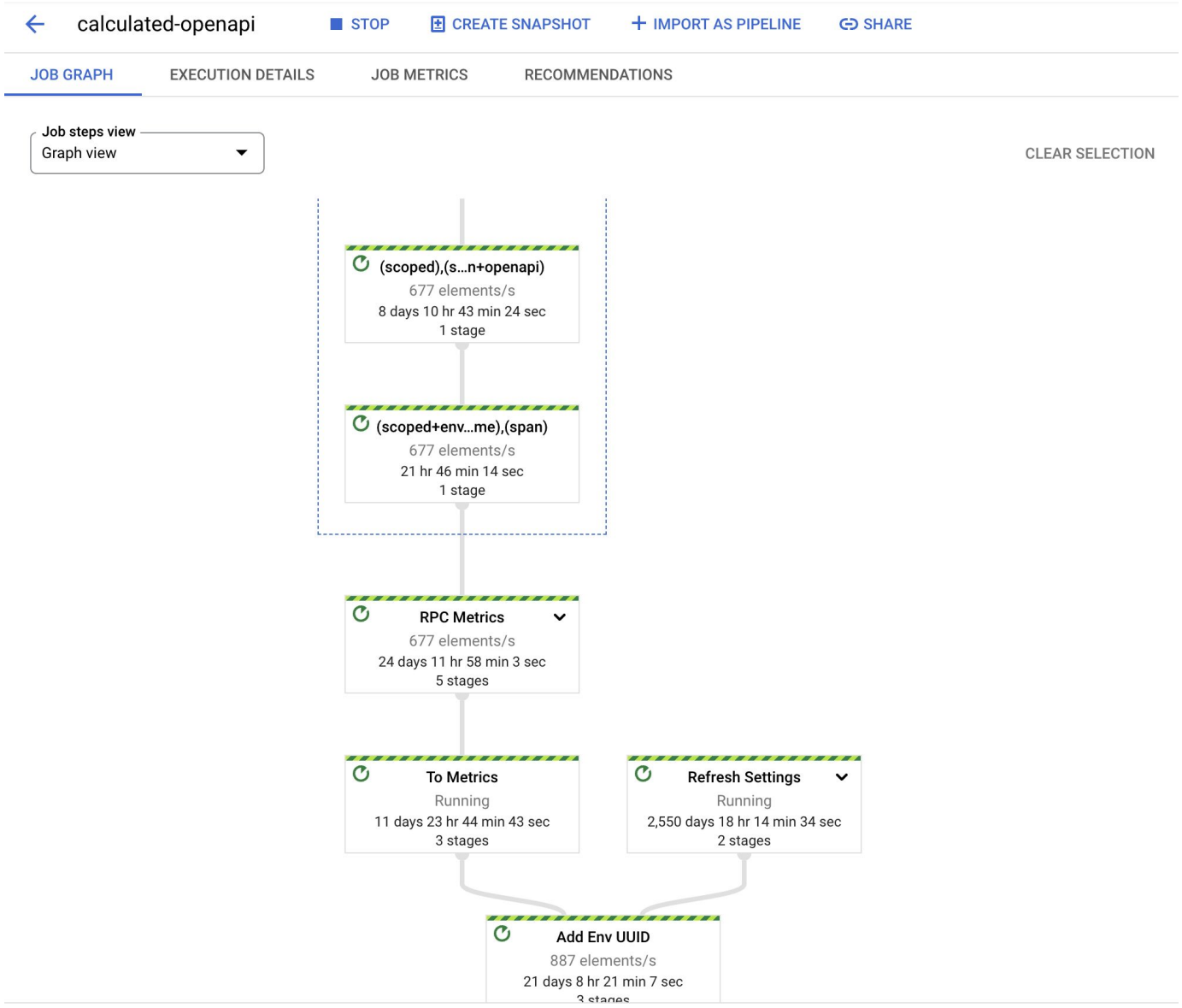
OTLP structs are easy to store as is, and easy to work with for real time aggregations.


R
P
E
R
P
E
R
P
E
R
P
E
R
P
E

Collibra

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)

beam

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?