Observability - Cost Vs Value

OpenTelemetry signals optimization

Traditional Operations - Old School

Operations, as a key entity within businesses, has typically functioned independently, often gaining prominence after deployment and billed according to resource usage

- **Operations are generally seen as separate revenue model.**

OpenTelemetry

Cloud Era - Pay as you use model

- **SaaSification** has transformed the way applications are delivered, shifting from on-premise to the cloud and as a service model.
- With a "**Pay As You Use**" charging system, dev, devOps, and operations are integrated, making Operations equally accountable for the company's bottom line.
- Optimization and applying **FinOps** to reduce operations costs are now **shared responsibilities.**

- **Everyone takes ownership for their cloud usage**

Designed by Freepik

OpenTelemetry

FinOps - Basic principles

- Use the “**Pay As You Use**” model to your advantage
- Know the business value
- Accessibility of Data
- Optimize
- Collaborate, collaborate & collaborate

Designed by Freepik

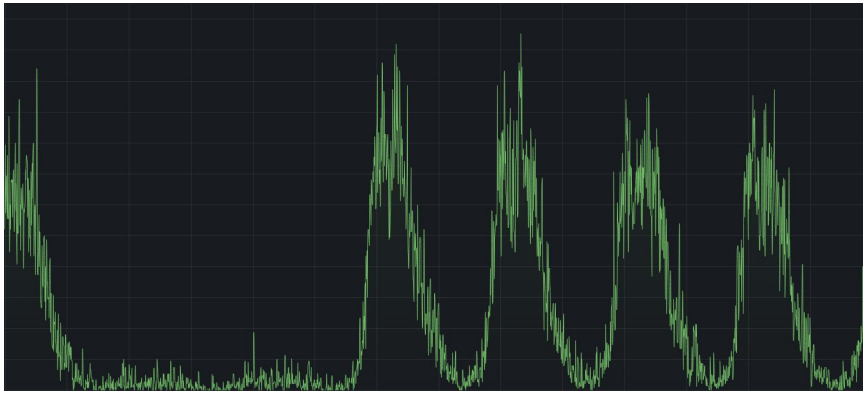
OpenTelemetry

Observability with OpenTelemetry and Grafana

Traces - Distributed trace gives you a hierarchy for calls within an API. Individual call within the trace is called **span**.

Metrics - Metric is a time series representation of data points related to an attribute. Example average response time of an API over last 2 days

Logs - Logs are the streams of output from the application.

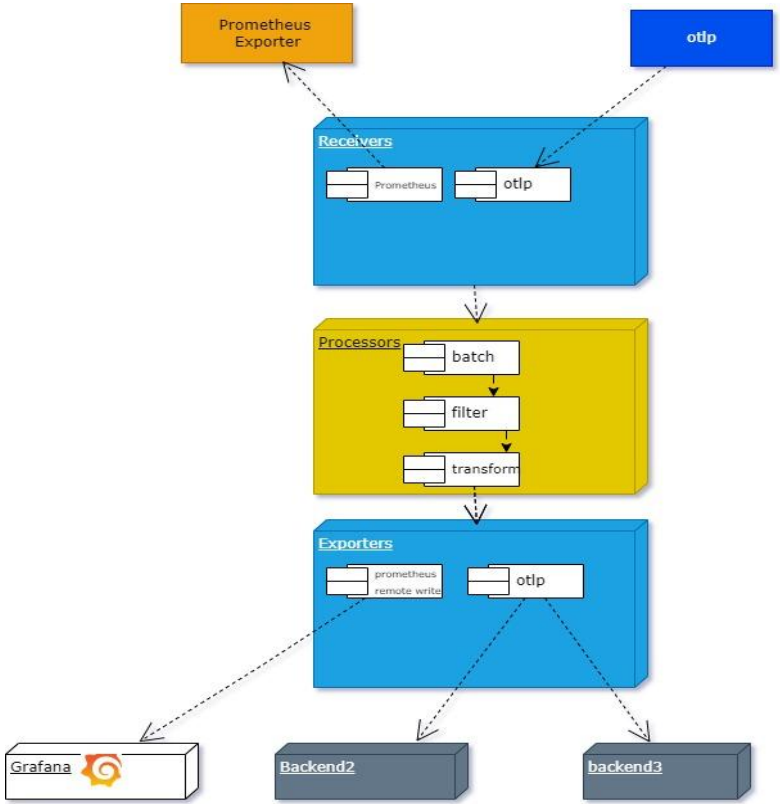


OTEL Collector workflow

Receivers : definition of agents/process which will receive/pull the signal data.

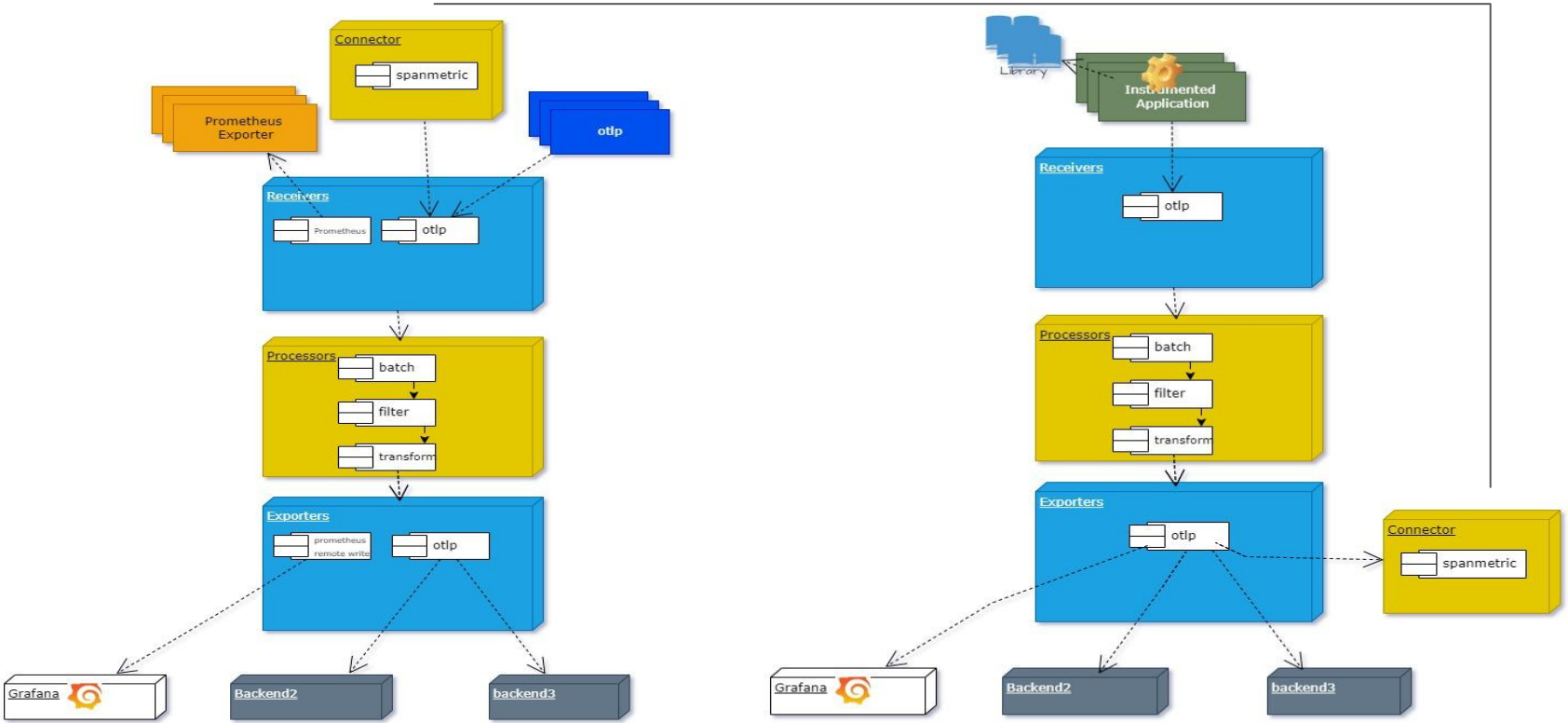
Processors : Define the configuration to enrich, filter or transform signal data.

Exporters : definition of process to push the signal data to a storage medium.Backend storage can accept OTLP schema or exporter can convert it to their native schema.



OpenTelemetry

Trace and Metric workflows



Observability - Signals and their use model

Traces

- Based on the **size**
- Base on the **Memory size** monitored
- Based on the **count**

Metrics

- Based on the **number of series**
- Based on the **number of attributes** being monitored
- Based on **size**

Logs

- Based on **size**

OpenTelemetry

Observability - Know the business value

Traces

- Need for **sampling**
- Understanding of **Key APIs**
- Understanding of various services and components
- Call Patterns

Metrics

- Need for retention.
- Histogram **Granularity**
- Aggregation
- Samples per minute
- Required **labels**

Logs

- Structure and **log pattern**
- Log Levels
- **Filtering** needs

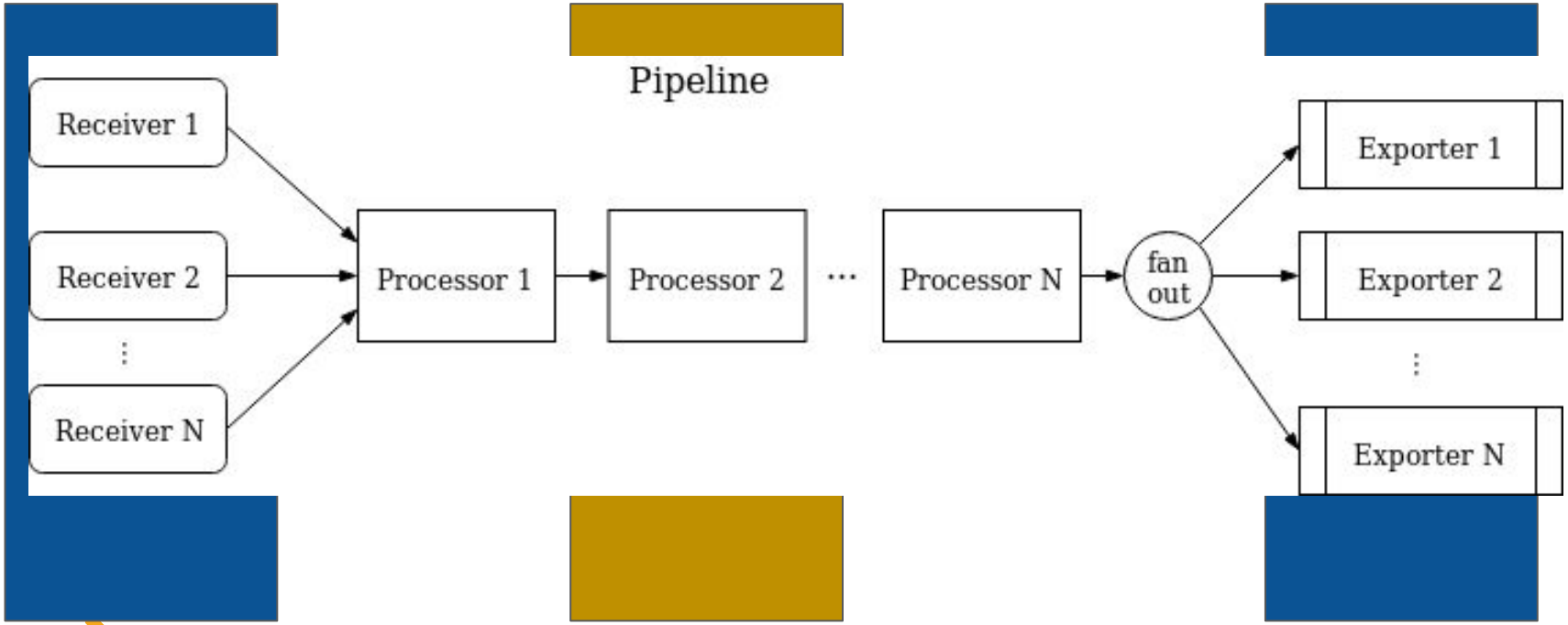
OpenTelemetry

Observability - Accessibility of usage metrics

- Based on the billing criteria, usage data need to be accessible in **real time**.
- Allow you to **drill down** the usage on various parameters.
- Alerts can be setup for any **anomaly** or usage beyond a limit

OpenTelemetry

Cost Optimization in open Telemetry



OpenTelemetry

Tips - Sampling at receiver level - (Trace)

Sampling at the source level allow you to process only selected traces based on specific conditions.

- **Head Sampling** - Sample a percentage of traces.
- **Tail Sampling** - Sample based on the condition and criteria within the trace

```
export OTEL_TRACES_SAMPLER="traceidratio"
export OTEL_TRACES_SAMPLER_ARG="0.1"
```

```
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.samplers.Sampler;

public class Example {

 public static void main(String[] args) {

   // Configure the tracer provider with the desired sampler
   SdkTracerProvider tracerProvider = SdkTracerProvider.builder()
     .setSampler(Sampler.alwaysOn()) // Set to always sample traces     // or
     .setSampler(Sampler.alwaysOff()) // Set to never sample traces     // or
     .setSampler(Sampler.traceIdRatioBased(0.5)) // Set to sample a fraction of tr
     .build();
 }
}
```

OpenTelemetry

Tips - Enabling and Disabling libraries

Many a time traces get **duplicated** as there are multiple layers of instrumentation available. In such cases one layer can be disabled.

- Enabled by default, review the traces and disable one at a time.

OTEL_INSTRUMENTATION_[NAME]_ENABLED=false

OTEL_INSTRUMENTATION_COMMON_DEFAULT_ENABLED = false

OpenTelemetry

Tips - Filter Processor (trace, metric, logs)

Filter processor filter the incoming signal based on the ottl query(**open telemetry transformation language** aka ottl) and discard the selected spans before sending to exporters.

```
processors:
  filter/ottl:
    error_mode: ignore
    traces:
      span:
        - 'attributes["container.name"] == "app_container_1"'
        - 'name == "app_3"'
      spanevent:
        - 'attributes["grpc"] == true'
        - 'IsMatch(name, ".*grpc.*")'
    metrics:
      metric:
        - 'name == "my.metric" and resource.attributes["my_label"] == "abc
      datapoint:
        - 'metric.type == METRIC_DATA_TYPE_SUMMARY'
        - 'resource.attributes["service.name"] == "my_service_name"'
    logs:
      log_record:
        - 'IsMatch(body, ".*password.*")'
```

OpenTelemetry

Tips - Transform Processor (trace, metric, logs)

Transform processor allow you to apply various functions based on the query filtering through ottl. It allows applying the functions like to **rename, replace, keep, set, truncate** etc.

```
transform:
  error_mode: ignore
  trace_statements:
    - context: resource
      statements:
        - keep_keys(attributes, ["service.name", "service.namespace",
"cloud.region", "process.command_line"])
        - replace_pattern(attributes["process.command_line"],
"password\\=[^\\s]*(\\s?)", "password=***")
        - limit(attributes, 100, [])
        - truncate_all(attributes, 4096)
    - context: span
      statements:
        - set(status.code, 1) where attributes["http.path"] == "/health"
        - set(name, attributes["http.route"])
        - replace_match(attributes["http.target"], "/user/*/list/*",
"/user/{userId}/list/{listId}")
        - limit(attributes, 100, [])
        - truncate_all(attributes, 4096)
```

OpenTelemetry

Tips - Transform Processor - Continue

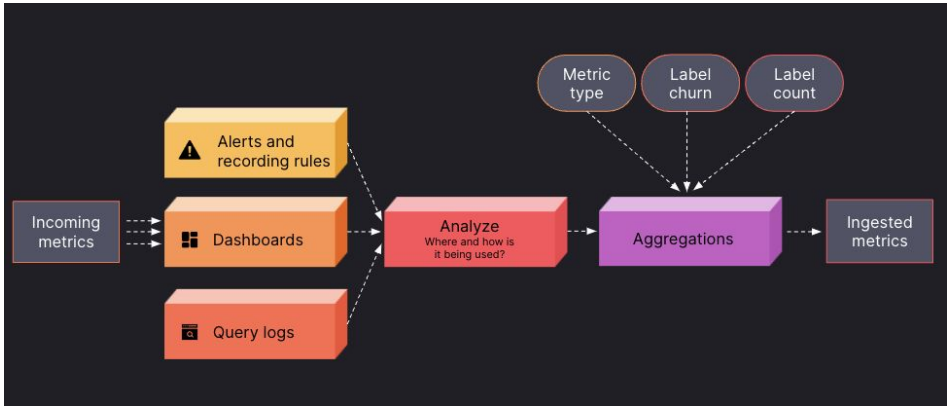
```
transform:
  error_mode: ignore
  metric_statements:
    - context: resource
      statements:
      - keep_keys(attributes, ["host.name"])
      - truncate_all(attributes, 4096)
    - context: metric
      statements:
        - set(description, "Sum") where type == "Sum"
        - convert_sum_to_gauge() where name == "system.processes.count"
        - convert_gauge_to_sum("cumulative", false) where name ==
"prometheus_metric"
    - context: datapoint
      statements:
        - limit(attributes, 100, ["host.name"])
        - truncate_all(attributes, 4096)
```

OpenTelemetry

Tips - Aggregation at exporter/backend level

Adaptive Metrics (Grafana)

- Analyse all the use of any metric (dashboard, alerts, queries) and suggest metrics which are **unused** for filtering.
- Suggest aggregation to **reduce data points**



OpenTelemetry

Designed by Freepik

OpenTelemetry

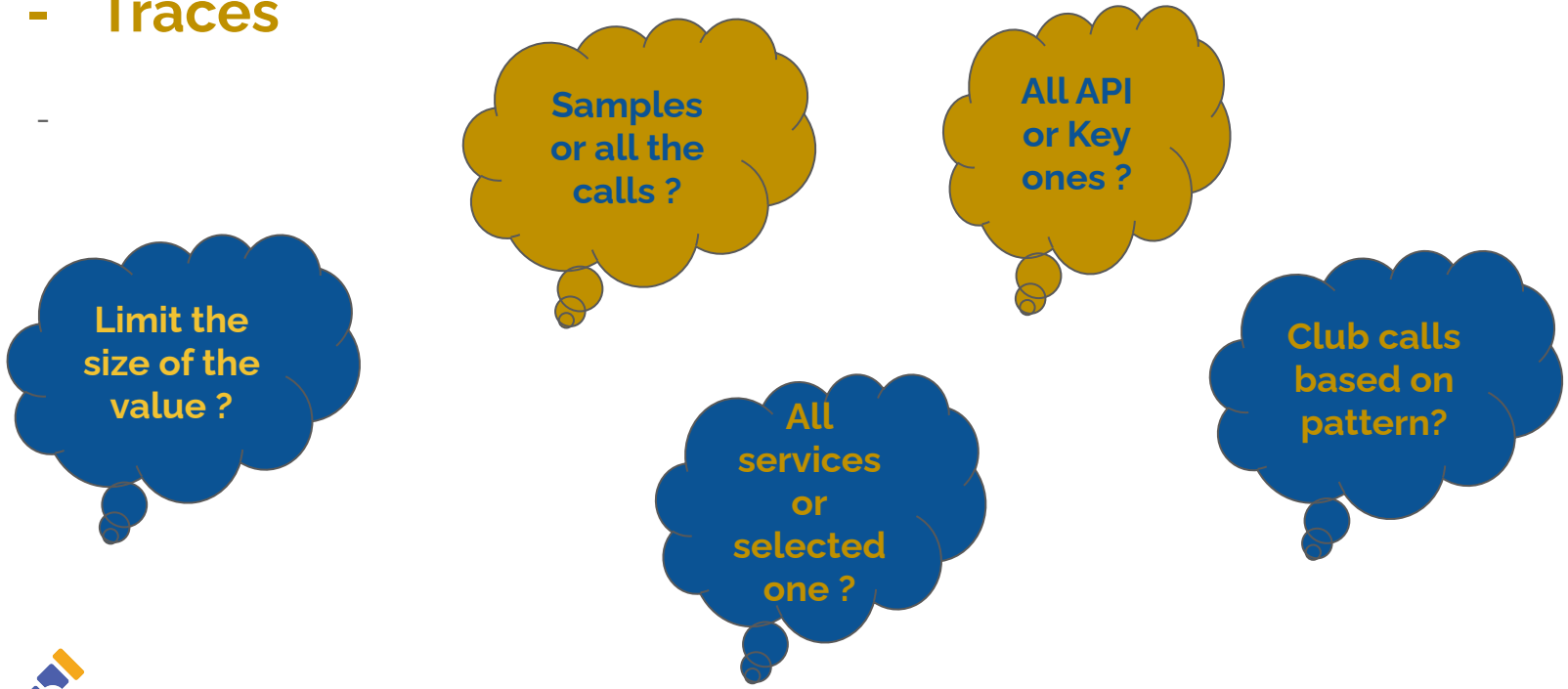
Tips and Tricks - Other Ways

- Drop the Labels
- Reduce data point
- Expire Serieses

Observability - Know the business value

- Traces

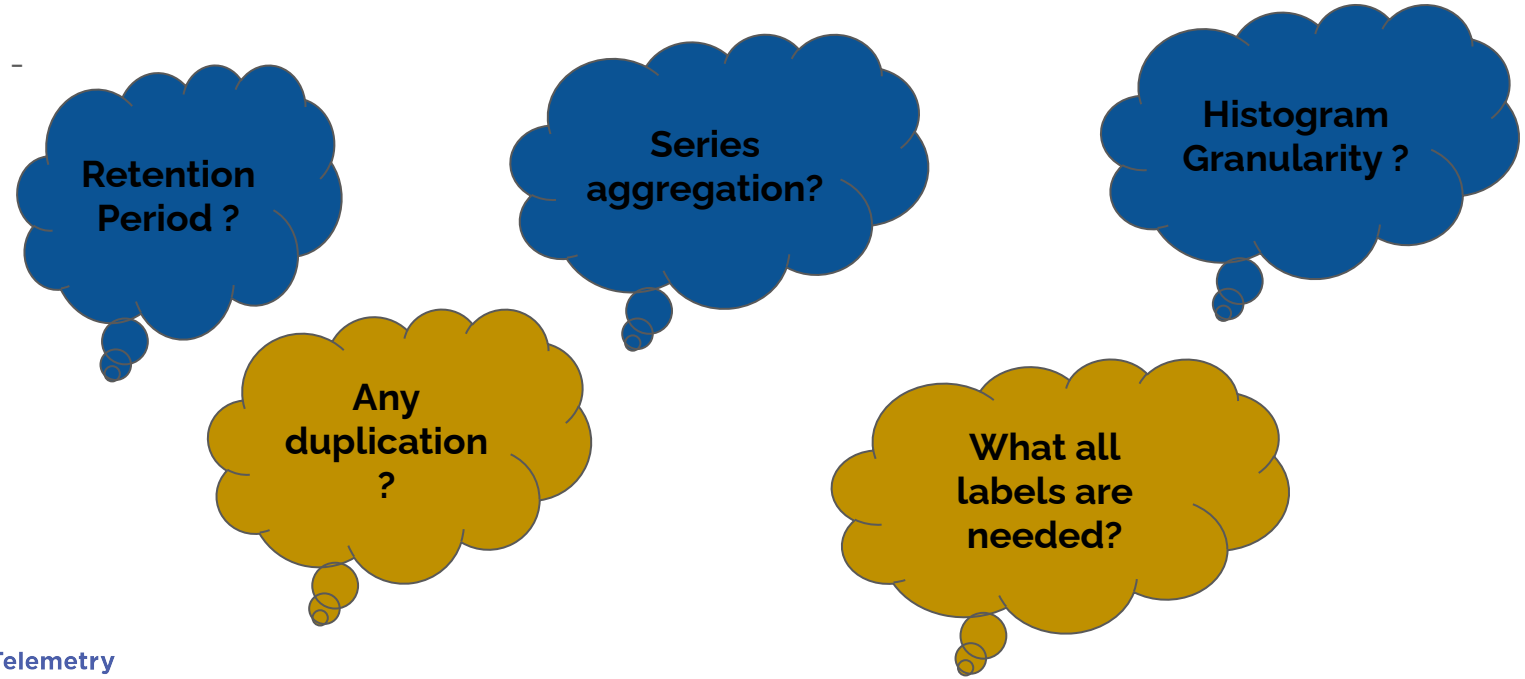
-



OpenTelemetry

Observability - Know the business value

- Metrics



OpenTelemetry

Observability - Know the business value

- Logs
  -

OpenTelemetry

Quiz Question

Q1: Everyone knows K8s is the biggest CNCF project but Which project of CNCF has the 2nd highest velocity ?
A: OpenTelemetry

Q2: What are the 3 major signals of Observability ?
A : Traces, Logs and Metrics

Q3 : What are the major section/clauses of Otel Collector ?
A: Receiver, processor & exporter

Q4 List 3 use model for Observability tools ?
A, Size, Number of series, RAM size

Q5: Name processors which help in finOps for Observability ?
A: Filter, transform

OpenTelemetry