



CONTROL
T H E O R Y

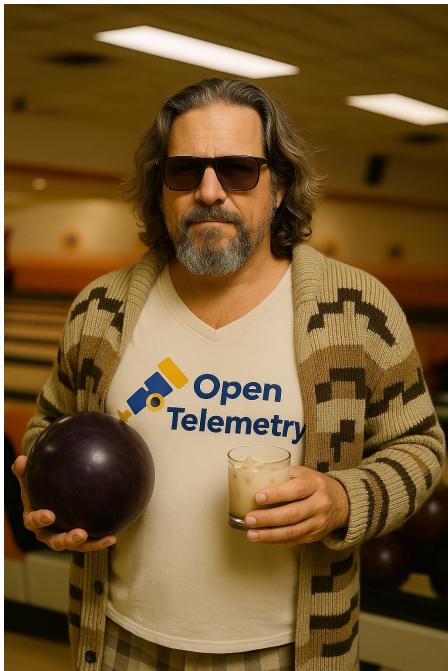
Taming Metric Cardinality

Eric Anderson, CTO, Co-founder (The Dude)
Jon Reeve, CPO, Co-founder (Walter)



Taming Metric Cardinality: The Dude Abides by OTTL

Eric



<https://bsky.app/profile/destari.bsky.social>

Jon



<https://bsky.app/profile/jreeve.bsky.social>



We're not heroes. Just here to stop cardinality explosions. That's our opinion, man.



Cardinality – What Is It, and Why Does It Hate You?

- Low cardinality: 10 time series
- High cardinality: **100,000+** time series

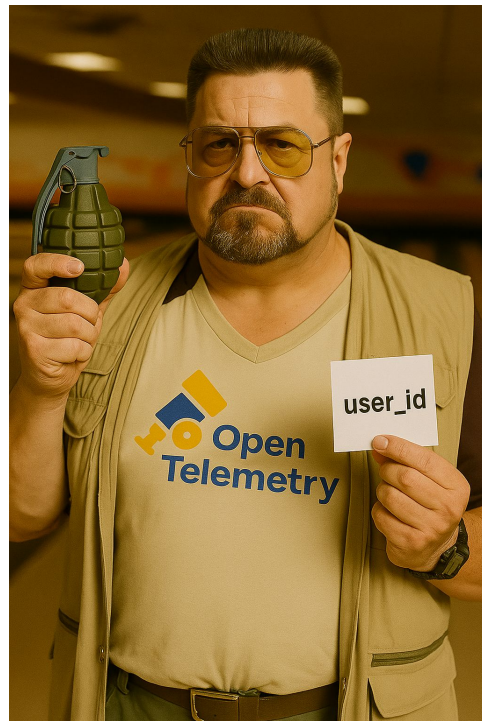
1000 customers, each customer has 100 users, and they have 10 projects:

`/api/{cust_id}/{project_id}/{user_id}/...`

$1,000 * 10 * 100 = \mathbf{1,000,000}$ time series!

More time series = more cost (& noise)

Every new label value is a new time series. That innocent *user_id* (*session_id*, *request_id*)? It's a cost bomb...





Cardinality – Examples

User id is a unique identifier:

```
/api/users/123456/profile
```

Order number is unique:

```
/api/orders/abcde-12345/details
```

Parameters can be a killer too:

```
/api/search?q=foo&page=2&sort=asc
```



```
http_request_duration_seconds{user_id="123456"}
```

```
container_cpu_usage_seconds_total{container_id="docker://2ae8f091b5d..."}
```

```
http_requests_total{path="/user/123/profile"}
```

The Problem – A League of Extraordinary Tags



This rug tied the dashboard together...
until 50k time series walked in.





Solution - OpenTelemetry Collector – Our Rug of Sanity



Doesn't care about your backend. Just wants peace... and fewer series.



What is “OTTL”?

- OTTL or “OpenTelemetry Transformation Language”
- Domain Specific Language in OTel collector to transform telemetry in flight
 - Filter
 - Enrich
 - Mask/Redact
 - Modify (conditionally)





The Dude's Five Steps

1. Standardize & measure
2. Transform
3. Filter
4. Aggregate
5. Test & Deploy





Standardize & Measure

- OTel Semantic Conventions
 - Common naming conventions
 - e.g. `system.cpu.utilization` vs
(`host.cpu.utilization` AND `node.cpu.utilization`
AND `mysnowflake.cpu.utilization`)
- Reference Tagging/label Namespace
 - e.g. your consistent labels/tags
- Measure Adherence



I'll tell you what I'm blathering about! I've got information, man!



Transform Processor

- Ideally update instrumentation at source
- When you can't → *transform* processor FTW!

```
replace_all_matches(attributes, "/user/*/list/*", "/user/{userId}/list/{listId}")  
  
delete_key(attributes, attributes["http.request.header.authorization"])  
  
set(attributes["environment"], "production") where attributes["service.name"] ==  
"my_service"
```

This isn't Nam. There are rules. OTTL rules.



Filter Processor – The Nihilist Slayer

- Drop user-level tags
- Strip vanity metrics
- Block noisy pods



We don't negotiate. We filter.



Filter Processor – The Nihilist Slayer

```
# Drop metric by name
metrics:
  metric:
    - 'name == "my.gauge"'

# Drop by tag key
metrics:
  metric:
    - 'HasAttKeyOnDatapoint("user_id")'

# Drop by key and value
metrics:
  metric:
    - 'HasAttrOnDatapoint("user_id", "12345")'

# Drop by datapoint only
metrics:
  datapoint:
    - attributes["user_id"] == "12345"
```



Aggregation

- We don't always need every metric
- Max, Min, Avg or Percentile etc..
- e.g. P95 Service Latency
- Interesting connectors
 - Logs to Metrics
 - Spans to Metrics



This aggregation will not stand, man.



Aggregation

```
metric_statements:  
  - context: metric  
    statements:  
      - copy_metric(name="my.second.histogram") where name == "my.histogram"  
      - aggregate_on_attributes("sum", []) where name == "my.second.histogram"
```

Transform Processor Example



Aggregation - Count processor

Use the *count processor* to turn a stream of high cardinality telemetry into a counter.

```
processors:  
  # Count processor to aggregate metrics  
  count/aggregate:  
    metrics:  
      api.request.count:  
        # Choose attributes to group by - fewer attributes = lower cardinality  
        attributes:  
          - key: project_id  
        aggregation_type: sum  
        aggregation_temporality: cumulative
```

Count Processor Example



Test & Deploy

- Pre-Collector
 - otl.run (transform & filter processor)
 - Avoids "OTTL rage" (be careful with AI)
- Your Collector
 - Synthetic Telemetry
 - Otelgen; Telemetrygen
 - Your Telemetry
 - debug exporter
 - file exporter, otlpjsonfilereceiver
 - AWS S3 Exporter/Receiver
 - Tap Processor
 - Validation
 - OTelBin
- Deployment
 - Helm; OTel Operator; Control Planes (OpAMP)



I can get you a toe by 3 o'clock this afternoon... with nail polish.



Synthetic Telemetry

```
# telemetrygen
docker run --rm
ghcr.io/open-telemetry/opentelemetry-collector-contrib/telemetrygen:latest \
  traces \
  --otlp-endpoint=host.docker.internal:4317 \
  --otlp-insecure

#otelgen
docker run --rm ghcr.io/krzko/otelgen \
  --otel-exporter-otlp-endpoint host.docker.internal:4317 \
  --rate 50 \
  --service-name test-app \
  --insecure \
  t m -t 150
```



Synthetic Telemetry - Cardinality Spammer

```
# telemetrygen
# creates up to 5 million unique time series!

telemetrygen metrics \
  --otlp-endpoint=localhost:4317 \
  --otlp-insecure \
  --duration=60s \
  --rate=200 \
  --attributes user_id={1..500},project_id={1..100},order_id={1..500} \
  --metrics-count=1 \
  --metric-names=api.request.count \
  --value-distribution=normal
```



OTelBin - Config validator

The screenshot displays the OTelBin Config validator interface. On the left, a configuration file is shown with the following content:

```
1 connectors:
2   count/exporter_pipelines:
3     datapoints:
4       controltheory_pipeline_exporter
5       attributes:
6         - key: pipeline
7         - key: exporter
8       description: The number of
9       logs:
10        controltheory_pipeline_exporter
11        attributes:
12          - key: pipeline
13          - key: exporter
14        description: The number of
15        spans:
16        controltheory_pipeline_exporter
17        attributes:
18          - key: pipeline
19          - key: exporter
20        description: The number of
21        count/exporters:
22        controltheory_exporters_metric
23        attributes:
24          - key: exporter
25        description: The number of
26        logs:
27        controltheory_exporters_log_cm
28        attributes:
29          - key: exporter
30        description: The number of
31        spans:
32        controltheory_exporters_span_cc
33        attributes:
34          - key: exporter
35        description: The number of
36        count/exporters_log_severity:
37        logs:
38        controltheory_exporters_log_det
39        attributes:
40          - key: exporter
41          - key: pipeline
42          - severity_number >= 5
43          - attributes['severity']
44          - attributes['level']
45        controltheory_exporters_log_err
46        attributes:
47          - key: exporter
48          - key: pipeline
49          - severity_number >= 17
50          - attributes['severity']
51          - attributes['level']
52        controltheory_exporters_log_fat
53        attributes:
54          - key: exporter
55          - key: pipeline
56          - severity_number >= 21
57          - attributes['severity']
58          - attributes['level']
59        controltheory_exporters_log_inf
60        attributes:
61          - key: exporter
62          - key: pipeline
63          - severity_number >= 9
64          - attributes['severity']
65          - attributes['level']
66        controltheory_exporters_log_not
67        attributes:
68          - key: exporter
69          - key: pipeline
70          - severity_number >= 9
71          - attributes['severity']
72          - attributes['level']
73        controltheory_exporters_log_not
```

On the right, a complex dependency graph is visualized, showing the relationships between various components and their dependencies. The graph is composed of numerous nodes and edges, representing the intricate dependencies within the configuration.



Demo - OTTL Playground

The screenshot shows the OTTL Playground interface with the following sections:

- Configuration** (YAML):

```
1 metrics:
2   datapoint:
3     - metric.name == "my.histogram" and count == 2
```
- OTLP payload** (JSON):

```
1 {
2   "resourceMetrics": [
3     {
4       "resource": {
5         "attributes": [
6           {
7             "key": "service.name",
8             "value": {
9               "stringValue": "my.service"
10            }
11          },
12          {
13            "key": "timestamp",
14            "value": {
15              "stringValue": "2018-12-01T16:17:18Z"
```
- Result** (JSON):

```
{
  resourceMetrics: [
    0: {
      scopeMetrics: [
        0: {
          metrics: [
            2: {
              "name": "my.histogram",
              "unit": "1",
              "description": "I am a Histogram",
              "histogram": {
                "aggregationTemporality": 1,
                "dataPoints": [
                  {
                    "startTimeUnixNano": "1544712660300000000",
                    "timeUnixNano": "1544712660300000000",
                    "count": "2",
                    "sum": 2,
                    "bucketCounts": [
                      "1",
                      "1"
                    ],
                    "explicitBounds": [
                      1
                    ]
                  }
                ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

At the bottom, there are links for [GitHub](#) and [Privacy](#), and a note "Crafted by elastic".



Links That Tie the Room Together!

Resource	Link
OTel Collector	https://opentelemetry.io/docs/collector/
Transform Processor	https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/transformprocessor
Filter Processor	https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/filterprocessor
Telemetrygen	https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/cmd/telemetrygen
Otelgen	https://github.com/krzko/otelgen
Semantic Conventions	https://opentelemetry.io/docs/concepts/semantic-conventions/
OTelBin	https://www.otelbin.io/
OTTL	https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/pkg/otl
OTTL Playground	https://otl.run/
ControlTheory Resources	https://www.controltheory.com/resources/



TL;DR

- Cardinality blows up bills & noise
- Audit telemetry and set a common destination
- Fix instrumentation, & where you can't:
 - Transform, Filter, Aggregate
- Test, deploy, repeat.



Talk &
Resources

Come see us after at the lanes!



Questions?

www.controltheory.com



Eric Anderson

CTO
ControlTheory



<https://www.linkedin.com/in/eanderson/>



@destari



@destari



Jon Reeve

CPO
ControlTheory



<https://www.linkedin.com/in/jonathan-m-reeve-phd-8406a71/>



@j0nr33v3



@j_m_reeve