KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Filter, Transform, Route

Mastering OTTL in the OpenTelemetry Collector



KubeCon CloudNativeCon
North America 2026

#KubeCon #CloudNativeCon



Evan Bradley



Edmo Vamerlatti Costa



Jamie Danielson



Tyler Helmuth



1

Intro to OTTL

Tyler Helmuth

Why OTTL exists, the data model, and our scenario

2

OTTL basics and debugging

Edmo Vamerlatti Costa

The basics, then debugging broken statements

3

Advanced OTTL with a live Collector

Jamie Danielson

Filtering, transforming, and routing

4

Extending the transform processor

Evan Bradley

Building a custom Collector

5

Audience questions

All of us

Ask us anything about OTTL

Demo code and every scenario link

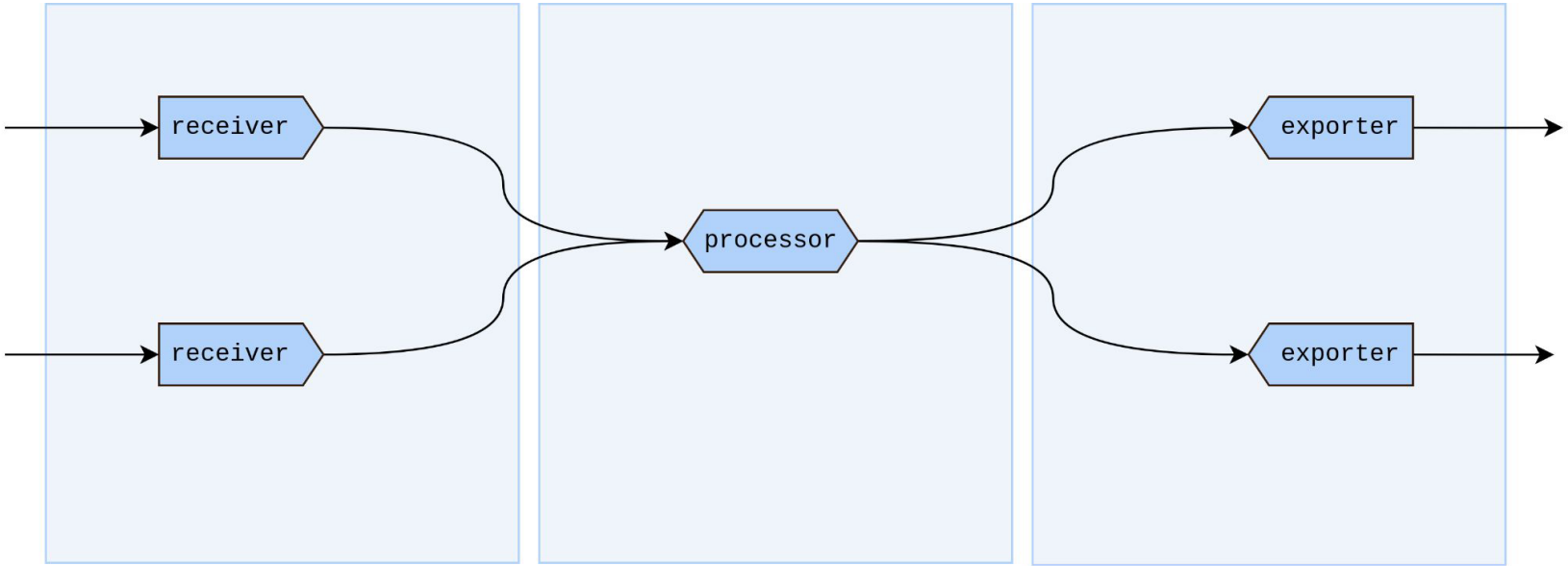
github.com/jamiedanielson/ottl-demo



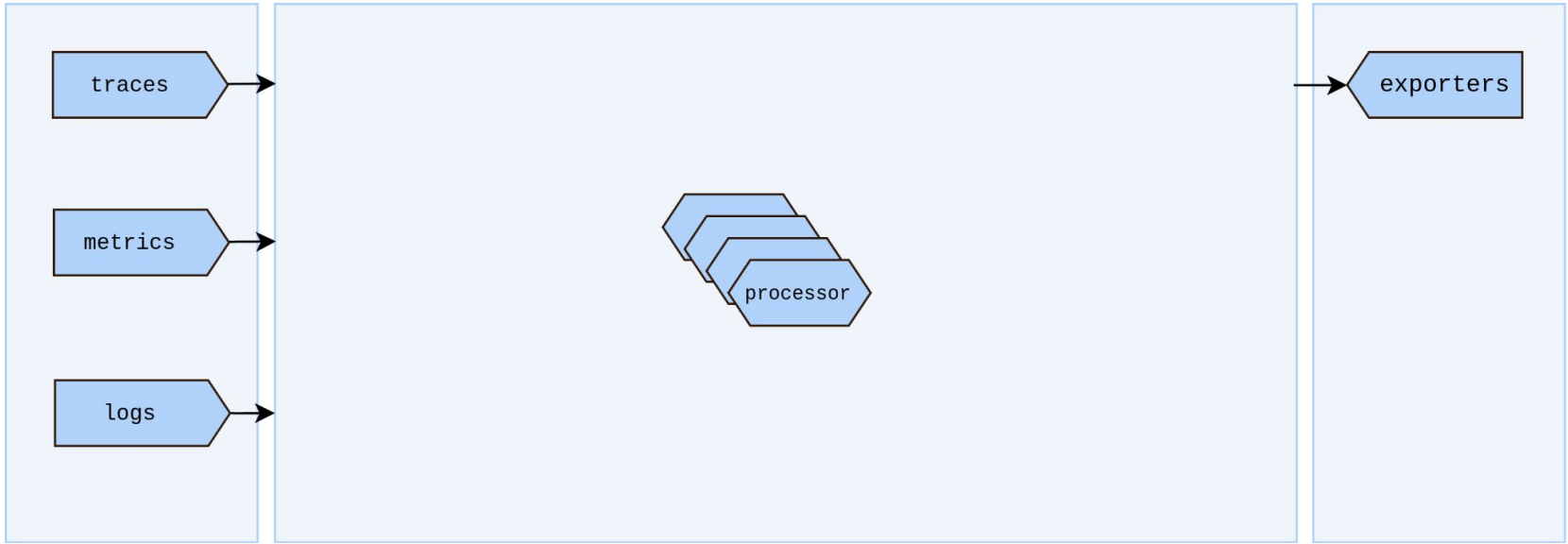
Why use the OTel Collector

01	Processing at the edge	• Scalability • Cost reduction • Control and data governance
02	Deploy anywhere	• A sidecar in a pod • Your hosts or near infrastructure • An ingress point to your systems
03	Versatility	• Speed • Low overhead • Customizability and extensibility

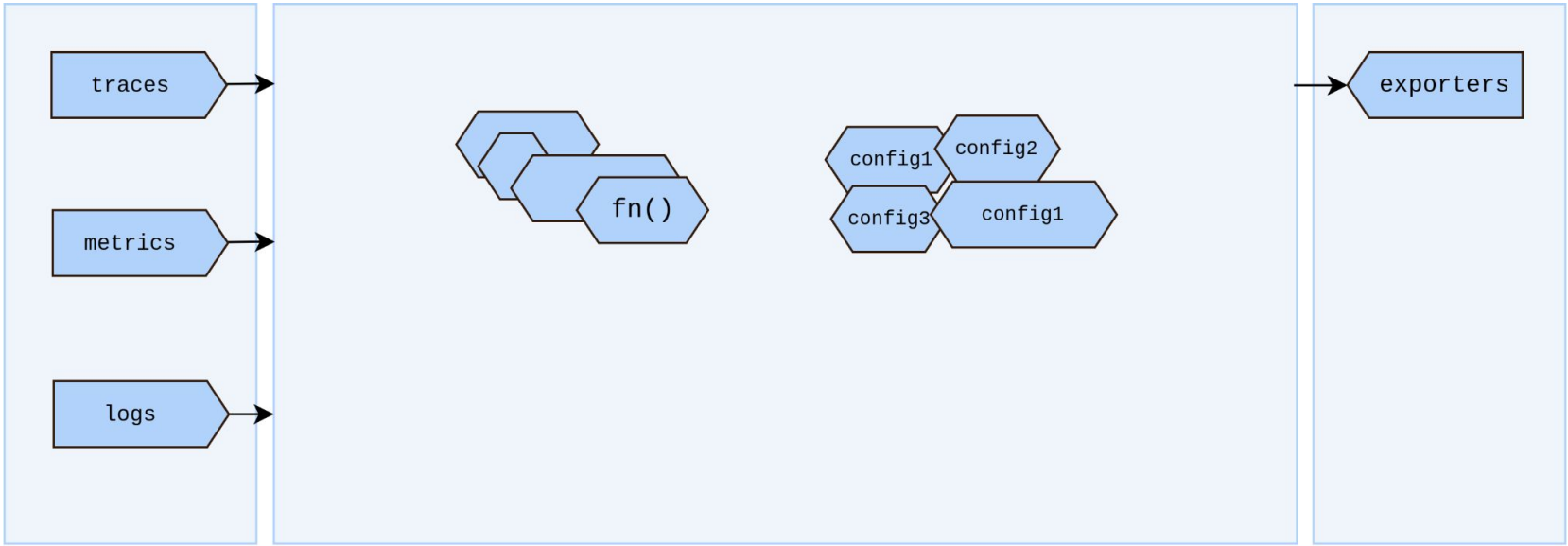
OTel Collector Components



Processing in the OTel Collector



Processing in the OTel Collector

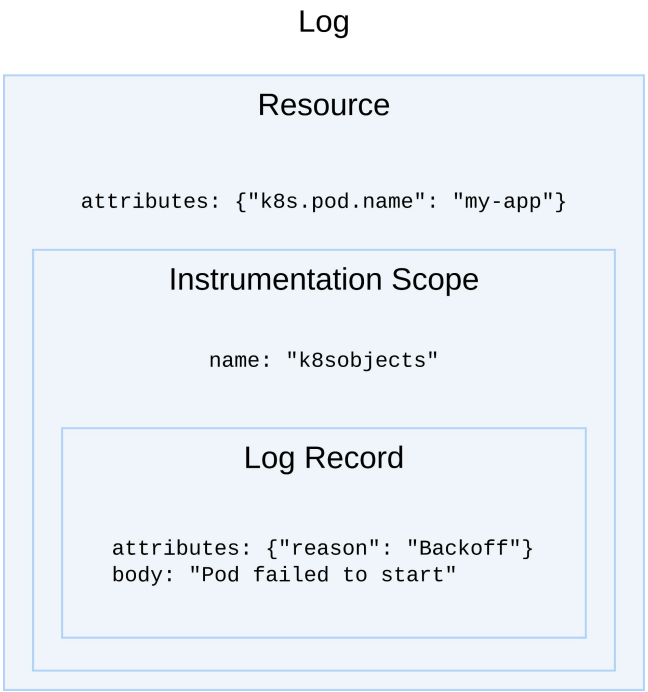


Processors lack alignment in purpose, functionality, and configuration.

Introducing OTTL

OTTL is a Domain Specific Language (DSL) tailor-made for interacting with telemetry in the OTel Collector.

- Provides access to all OTLP fields.
- Enables expressing complex transformations using a simple syntax.



OTTL 1.0 is here!

Stable for traces, metrics, and logs starting with Collector Contrib v1.3.1

Available in the v0.164.0 Collector distributions: otelcol, otelcol-contrib, otelcol-k8s, and otelcol-ebpf-profiler



Language Features

Paths

Access all OTLP fields

Functions

Edit telemetry or convert values

Conditions

Where clauses to gate statements

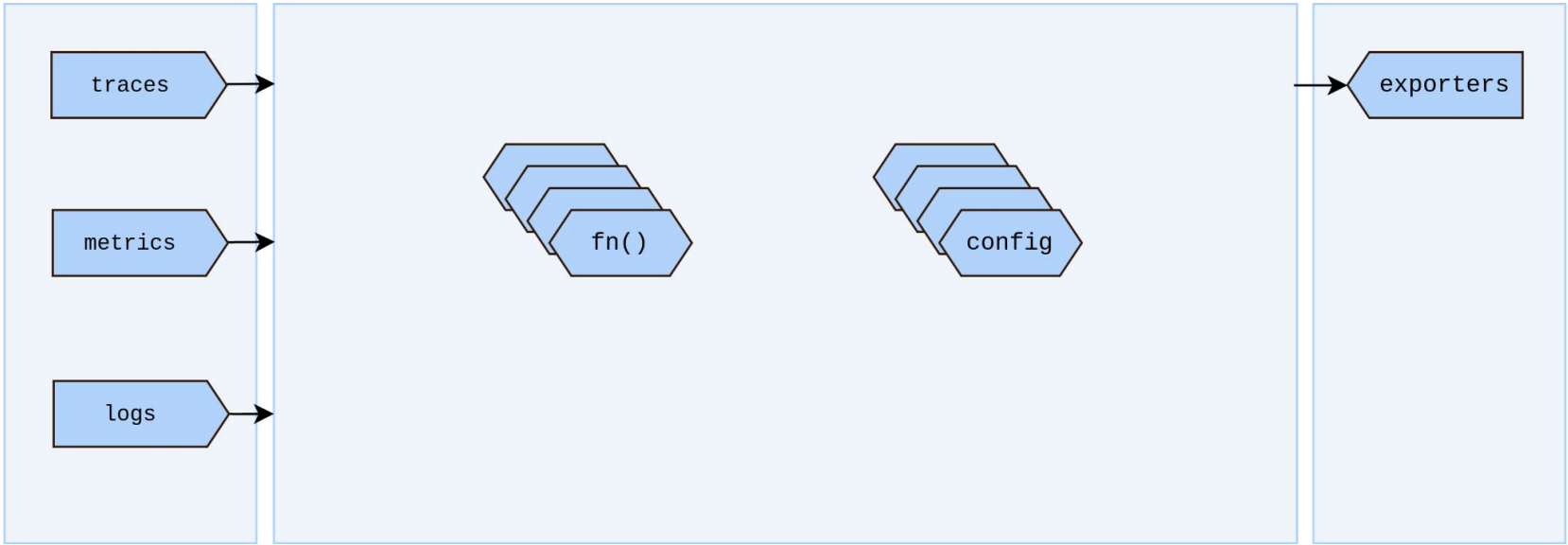
Comparators

Boolean logic and arithmetic

Lambdas

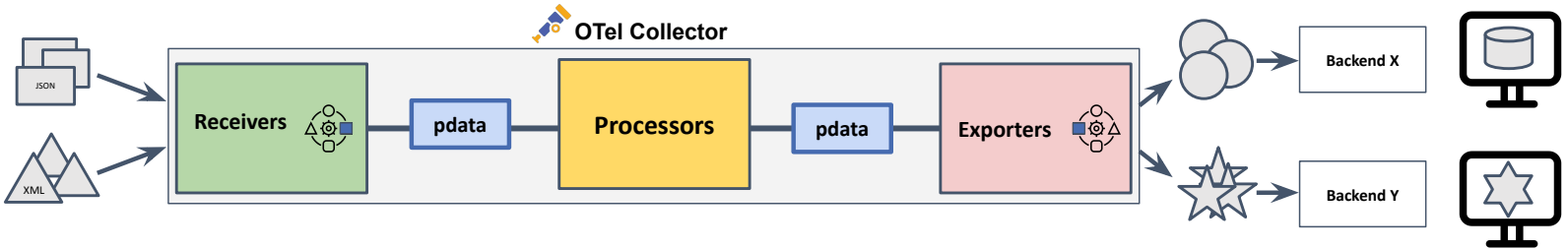
Inline dynamic functions

Processing in the OTel Collector



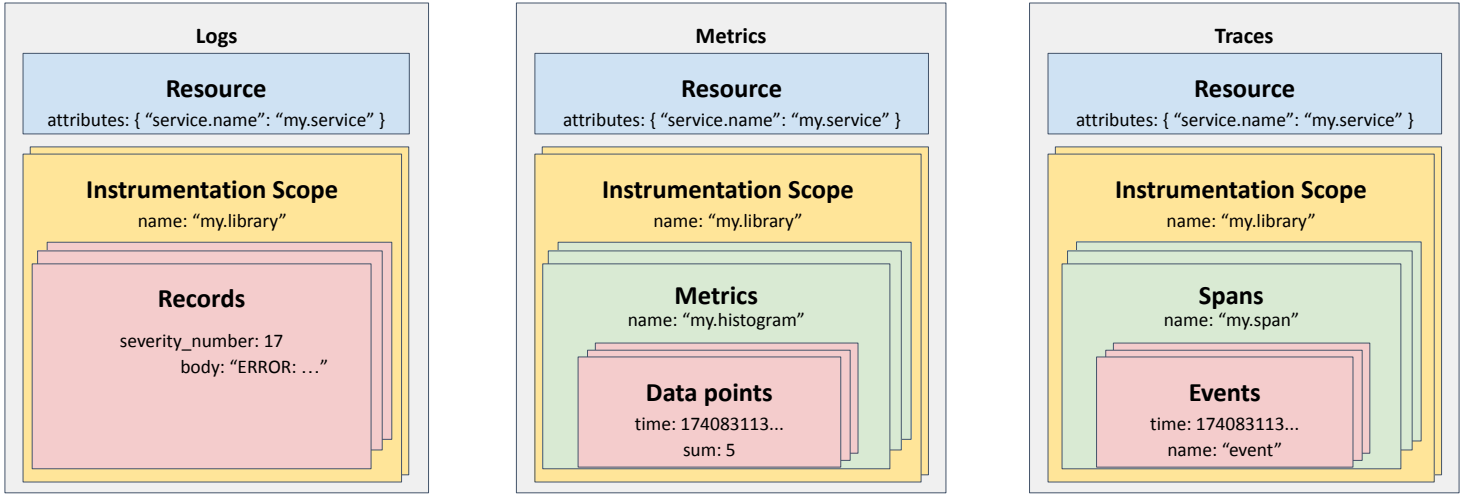
Pipeline Data (a.k.a. pdata)

- In-memory representation of the telemetry data model.
- The working format inside the OTel Collector pipeline.
- OTTL works on pdata, not raw payloads or backend data.



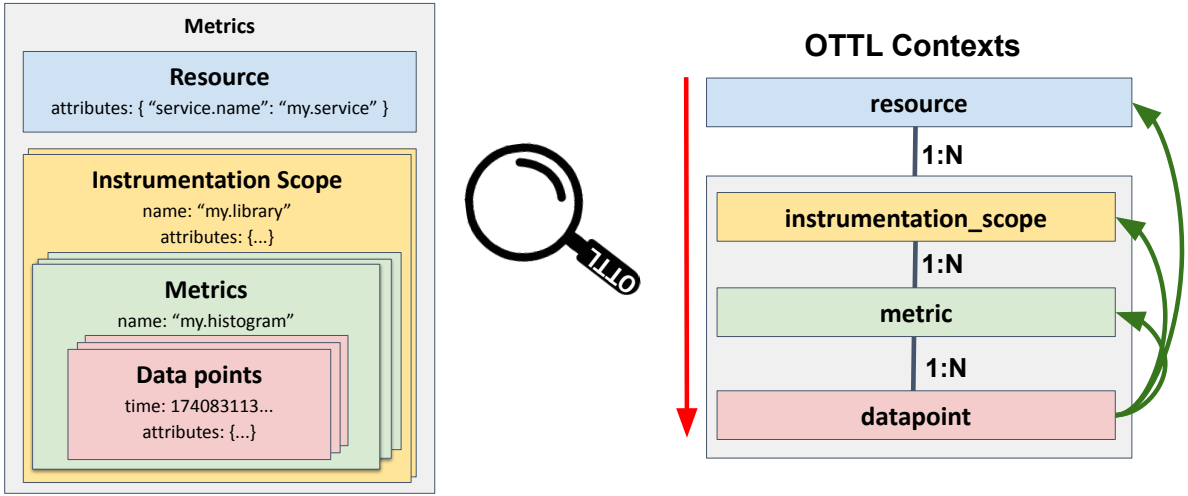
Telemetry Data Model

- Consistent structure for representing observability data.
- Describes what data should exist and what it means.



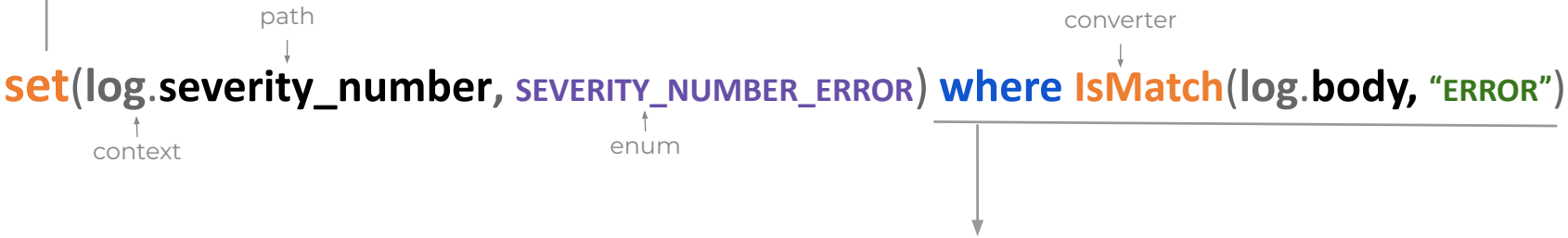
OTTL Contexts

- Contexts define which part of pdata OTTL operates on
- Bridge between exposed data paths and the underlying pdata
- Manipulate what's inside the Collector, not raw receiver payloads



OTTL Statement

Function that transforms the telemetry data by mutating or writing values into specified paths.



Optional condition that determines whether the statement function is executed.

Not sure which OTTL path to use? Start here →

Explore the documentation to find the right context, statements, and path references for your telemetry pipeline.

bit.ly/ottl-contexts

How to Troubleshoot OTTL?

01



Debug Exporter

Inspect full telemetry structure, spans, and attributes directly in collector output.

02



Debug Logs

Track statement execution step-by-step and verify condition evaluation rules.

03



OTTL Playground

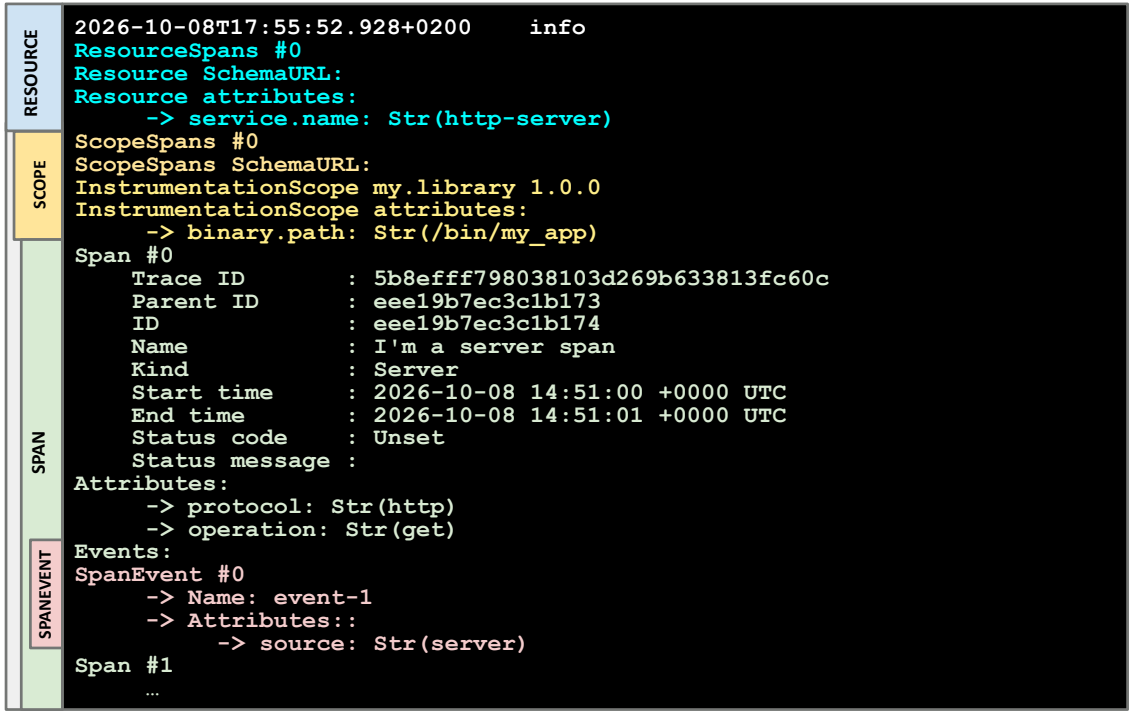
Interactively test and validate OTTL statements in an isolated web environment.

Debug Exporter

```yaml
# Collector config

exporters:
 debug:
   verbosity: detailed

service:
 pipelines:
   traces:
     receivers:  [...]
     processors: [...]
     exporters:  [debug]
```



Debug Logs

- Shows how OTTL executes statements
- Records before and after data for each statement
- Confirms whether the condition matched
- Useful for debugging tricky issues

```
# Collector config
service:
 telemetry:
   logs:
     level: debug

     # Optionally send internal logs to an
     # OTLP  backend
     processors:
       -batch:
          exporter:
            otlp:
              endpoint:<endpoint>
```

Inside the Logs

```
transform:
    log_statements:
      - set(log.severity_number,  17) where IsMatch(log.body, "ERROR")
      - set(log.severity_text, "INFO") where log.severity_number == 9
```

```
2026-10-29T10:57:42.460+0100 DEBUG ottl@v0.138.0/parser.go:400  initial TransformContext before executing
{...}

2026-10-29T10:57:42.460+0100 DEBUG ottl@v0.138.0/parser.go:37   TransformContext after statement execution
{...}

2026-10-29T10:57:42.460+0100 DEBUG ottl@v0.138.0/parser.go:37   TransformContext after statement execution
{...}
```

Making Sense of Debug Logs

```
DEBUG initial TransformContext before executing
StatementSequence
{
 "TransformContext":{
   body: "ERROR:internal failure"
   severity_number: 9
   severity_text: ""
   ...
 }
}
```

Making Sense of Debug Logs

```
DEBUG TransformContext after statement execution
{
 "statement": set(log.severity_number, 17)
              where IsMatch(log.body, "ERROR")

 "condition matched": true

 "TransformContext": {
   body: "ERROR:internal failure"
   severity_number: 17 9
   severity_text: ""
   ...
 }
}
```

Making Sense of Debug Logs

```
DEBUG TransformContext after statement execution
{
 "statement": set(log.severity_text,"INFO")
              where log.severity_number == 9

 "condition matched": false

 "TransformContext": {
   body: "ERROR:internal failure"
   severity_number: 17
   severity_text: ""
   ...
 }
}
```

```
transform:
  log_statements:
    - set(log.severity_number, 17) where IsMatch(log.body, "ERROR")  # A
    - set(log.severity_text, "INFO") where log.severity_number == 9   # B
```

```
DEBUG TransformContext after
statement execution (last B)

"TransformContext": {
  body: "ERROR:internal failure"
  severity_number: 17
  severity_text: ""
  ...
}
```

```
DEBUG TransformContext after
statement execution (last A)

"TransformContext": {
  body: "ERROR:internal failure"
  severity_number: 17 9
  severity_text: "INFO"
  ...
}
```

OTTL Playground (ottl.run)

A powerful and user-friendly tool for experimenting with OTTL.

- No need to run a Collector
- Visual diffs and **debug logs** included
- WebAssembly-powered for near-live accuracy



KubeCon CloudNativeCon North America 2026

Advanced OTTL with a live Collector

Filter. Transform. Route.

The telescope shop's telemetry is noisy and leaking. Let's fix it in the Collector, without touching the apps.

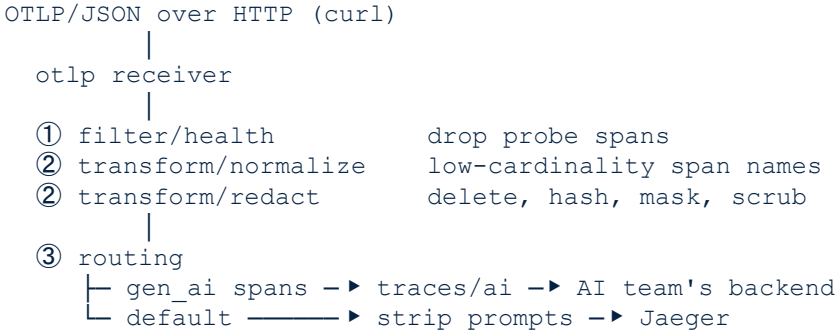


Right tool for the job

Job	Component	Config key	Stability
Drop what you don't need	Filter processor	`trace_conditions`	alpha
Reshape and protect	Transform processor	`trace_statements`	beta
Send data elsewhere	Routing connector	`table`	alpha

OTTL is stable (1.0). The components that host it each have their own stability level, so check each one.

What we'll build




```
processors:
  transform/redact:
    trace_statements:
      - 'delete_key(span.attributes, "user.email")'
```

Five things that bit us

1. **Anchor your regexes, values and keys.** `"health"` also matched `/blog/healthy-stargazing`, and a key pattern of `.*token.*` also deletes `gen_ai.usage.input_tokens`.
2. **The filter works one span at a time.** Dropping a parent orphans its children: that readiness probe's `SELECT 1` survived on its own.
3. **No path, no inference.** `set_semconv_span_name(...)` has no path, so give its group a `context:`. `validate` catches it before you deploy.
4. **One group, one context.** Mixing `span` and `spanevent` paths in one list runs the whole list once per span *event*. Spans without events are silently skipped.
5. **Routing on a span condition moves spans, not traces.** With the default `move`, Jaeger got an incomplete trace. `action: copy` sends the spans to both pipelines.

In Collector config, `${...}` is expanded before OTTL sees it, so write a literal `$` as `$$` (`${1}` fails to load).

What's new and worth watching

- **Lambdas** (alpha, `--feature-gates=ottl.functions.enableLambda`), with experimental converters `All`, `Any`, `Filter`, `Find`, `MapEach`, `MapKeys`, `Reduce` and `When`:

  ```
  set(span.attributes, MapEach(span.attributes,
      (k, v) => When(() => IsMatch(k, "(?i)email"), "<redacted>", v)))
  ```

  This masks the value of every attribute whose key looks like an email field, and keeps the key.

- **`set_semconv_span_name("1.43.0")`**: semconv-based span names, as used in the demo (semconv 1.37.0–1.43.0).

- **Context inference and the flat config style**, plus routing's `action: copy` and the `otelcol.client.metadata` / `otelcol.grpc.metadata` paths that replace the deprecated `request` context.

- **`shared_cache`**: share the OTTL cache across statement groups in one transform processor.

- **Profiles context**: in development.

Try this pipeline yourself

<short link to the repo: TBD>



`./demo stage 00-baseline` → `./demo send` → edit `config/live.yaml` → `./demo apply`

KubeCon CloudNativeCon North America 2026

Extending the transform processor

Building a custom Collector

When the built-in functions aren't enough, write your own and register it in your own Collector distribution.



KubeCon CloudNativeCon North America 2026

Your turn: write some OTTL

Build it. Break it. Fix it.

Pre-built telescope shop scenarios in ottl.run. Nothing to install, just a browser.



How the scenarios work

1. **Open a scenario link.** It loads a telescope shop payload and a starter config.
2. **Read the comments.** The top of the config has the situation, your goal, a stretch goal, and functions to reach for.
3. **Write your statements.** Replace `[ ]` with your statements, click Run, and compare the result with the input.
4. **Stuck? Use the debugger.** Set a breakpoint next to a statement to see the data before and after it runs.
5. **Check the solution.** Every scenario has a solution link. Try it yourself first.



github.com/jamiedanielson/ottl-demo

Rename legacy attributes



https://is.gd/ottl_scenario_1

Solution: Rename legacy attributes



https://is.gd/ottl_scenario_1_solution

Drop health-check spans



https://is.gd/ottl_scenario_2

All scenarios: github.com/JamieDanielson/ottl-demo

Solution: Drop health-check spans



https://is.gd/ottl_scenario_2_solution

All scenarios: github.com/JamieDanielson/ottl-demo

Redact PII and secrets



https://is.gd/ottl_scenario_3

Solution: Redact PII and secrets



https://is.gd/ottl_scenario_3_solution

Fix high-cardinality span names



https://is.gd/ottl_scenario_4

Solution: Fix high-cardinality span names



https://is.gd/ottl_scenario_4_solution

Parse messy logs



https://is.gd/ottl_scenario_5

All scenarios: github.com/JamieDanielson/ottl-demo

Solution: Parse messy logs



https://is.gd/ottl_scenario_5_solution

Debugging clinic: it runs, it's wrong. Why?

1. **Read the goal.** The comments say what the config should do and what you'd expect to see.
2. **Compare the result.** Each config looks right and runs, but the output isn't what the goal describes.
3. **Find and fix the bug.** Each scenario has one. Fix it and run again.
4. **Read the errors too.** Two of the configs don't even load. The error message is the clue.
5. **Use your tools.** The result diff, breakpoints, and switching `error_mode` to `propagate`.

!= on a missing attribute



https://is.gd/ottl_scenario_6a

Solution: != on a missing attribute



https://is.gd/ottl_scenario_6a_solution

Int vs. string



https://is.gd/ottl_scenario_6b

All scenarios: github.com/JamieDanielson/ottl-demo

Solution: Int vs. string



https://is.gd/ottl_scenario_6b_solution

Unanchored regex



https://is.gd/ottl_scenario_6c

Solution: Unanchored regex



https://is.gd/ottl_scenario_6c_solution

Escaping



https://is.gd/ottl_scenario_6d

All scenarios: github.com/JamieDanielson/ottl-demo

Solution: Escaping



https://is.gd/ottl_scenario_6d_solution

Statement order



https://is.gd/ottl_scenario_6e

Solution: Statement order



https://is.gd/ottl_scenario_6e_solution

Metric vs. datapoint context



https://is.gd/ottl_scenario_6f

Solution: Metric vs. datapoint context



https://is.gd/ottl_scenario_6f_solution

error_mode: ignore



https://is.gd/ottl_scenario_6g

Solution: error_mode: ignore



https://is.gd/ottl_scenario_6g_solution

Mask emails with lambdas



https://is.gd/ottl_scenario_7

Solution: Mask emails with lambdas



https://is.gd/ottl_scenario_7_solution

Metric identity



https://is.gd/ottl_scenario_8

Solution: Metric identity



https://is.gd/ottl_scenario_8_solution

KubeCon CloudNativeCon
North America 2026

Thank you!

Keep practicing at ottl.run

Every scenario link keeps working after today. Share them with your team.



KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Questions?

Ask us anything about OTTL

Filter, Transform, Route: Mastering OTTL in the OpenTelemetry Collector

