



Beyond “Supports OpenTelemetry”: A maturity model for Cloud Native Observability

Kasper Borg Nissen, Director of Developer Relations at Dash0



kaspernissen.xyz



[/in/kaspernissen](https://in.linkedin.com/in/kaspernissen)



[kaspernissen](https://github.com/kaspernissen)



Who?

Director of Developer Relations at Dash0

KubeCon+CloudNativeCon EU/NA 24/25 Co-Chair (former)

Author of OpenTelemetry for Dummies

CNCF, AAIF, MergeForward Ambassador

Golden Kubestronaut

CNCG Aarhus, KCD Denmark Organizer

Co-founder & Community Lead Cloud Native Nordics



Scattered signals. Growing haystacks.

Every signal its own tab. Every tab its own query language or filter. And one engineer trying to correlate it all - in their head.

THREE BROWSER TABS OF OBSERVABILITY

req_id=9f3a...
trace_id=1c4b...

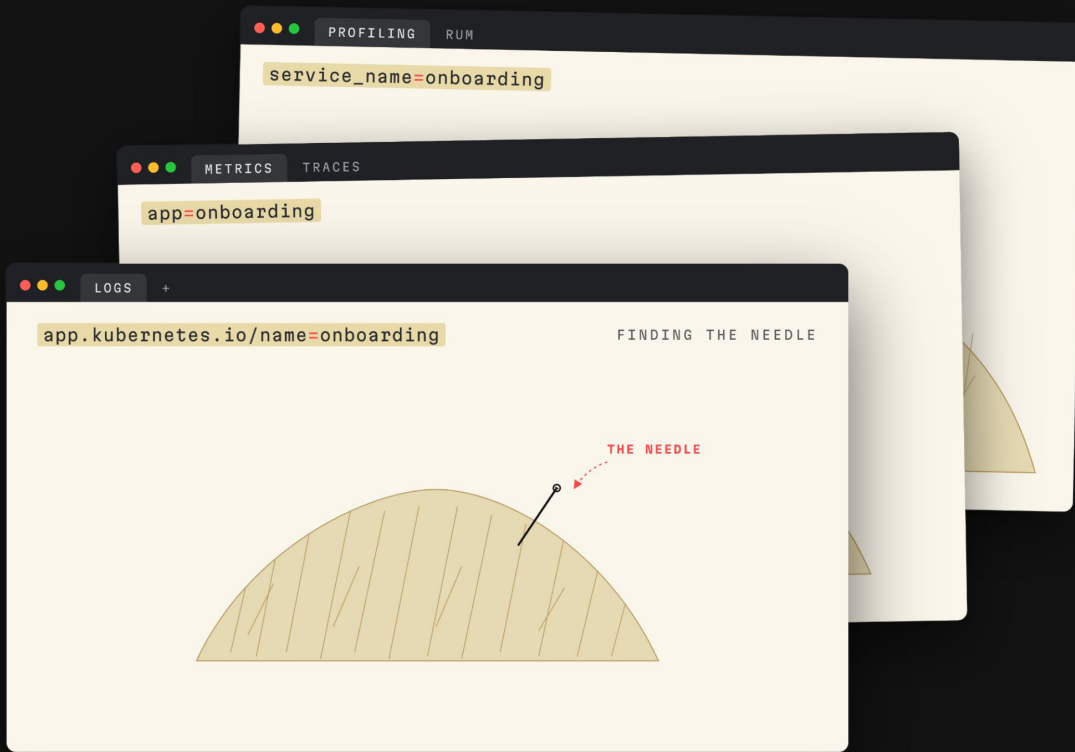
which tab was it again?

“_(ツ)_/”

which label was it?



COGNITIVE OVERLOAD



MORE TABS

BIGGER HAYSTACK

Stamped on every project page.

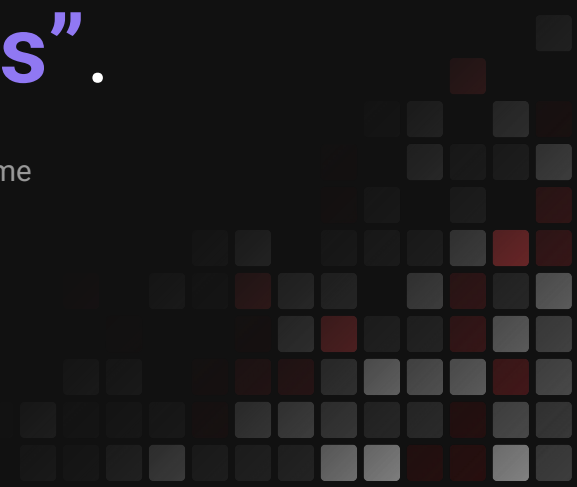
Same two words on READMEs and release notes. The ink is everywhere — what's underneath varies wildly.



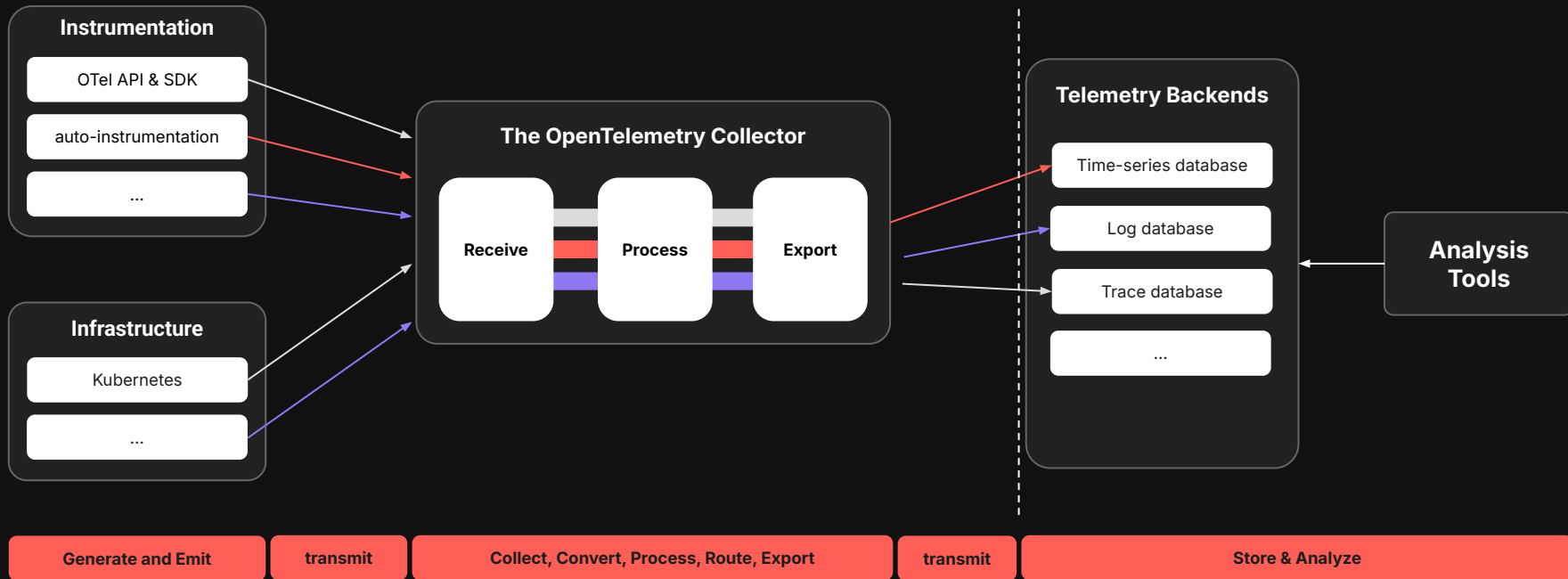


“Supports OpenTelemetry”
has become the new
“Supports Kubernetes”.

A phrase that means everything - and nothing - at the same time



OpenTelemetry: a 1000-mile view.



Three signals. One context.

The building blocks behind the diagram

Produce & Move

SDKs

Auto & manual instrumentation - in apps and on the request path.

OTLP

The wire protocol. One pipe, three signals - spoken by SDK, Collector, and backends.

Collector

Shared plumbing: filter, batch, sample, route - outside your apps.



Make it meaningful

Semantic Conventions

Agreed names & shapes - so data mean the same thing across tools.

Resource attributes

The identity on every signal - who emitted this.

Context

How trace context & baggage flow through sync and async hops.





Telemetry without context is just data



What are we looking at?

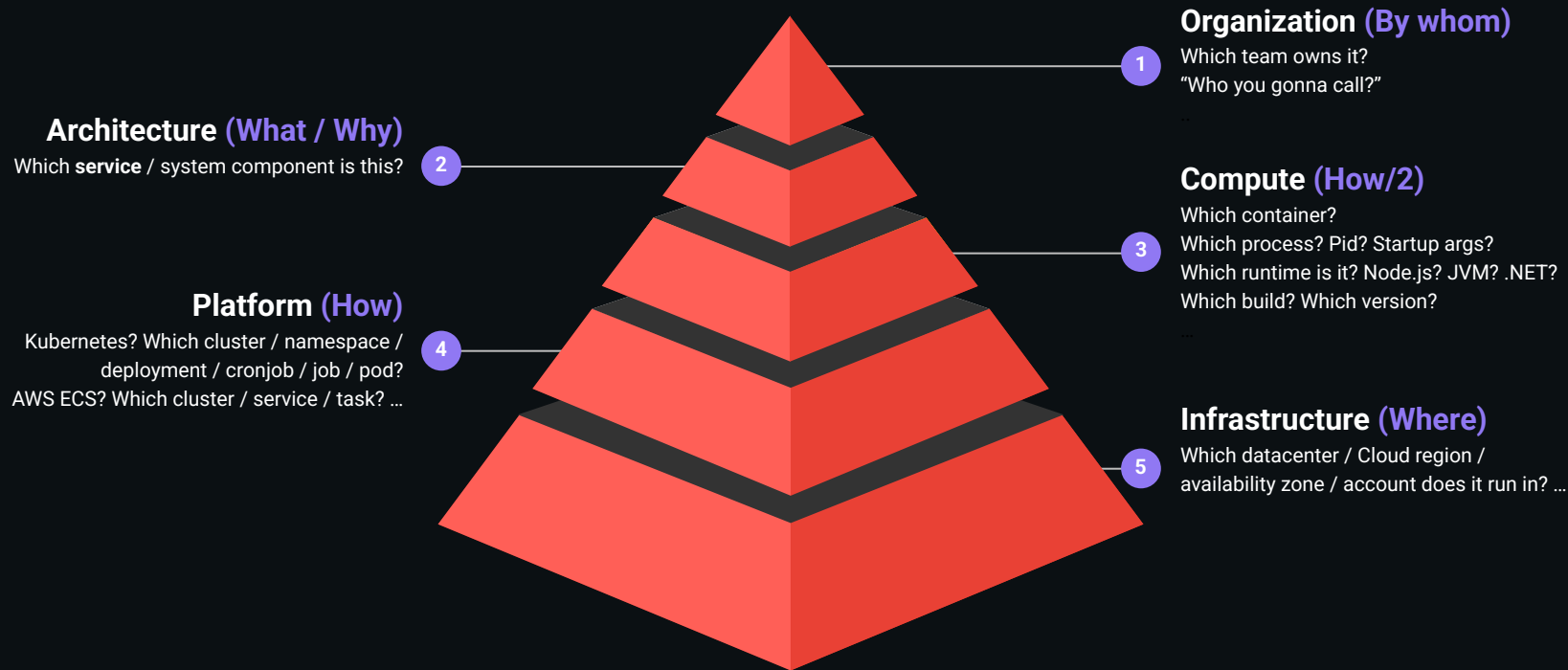


What are we looking at?

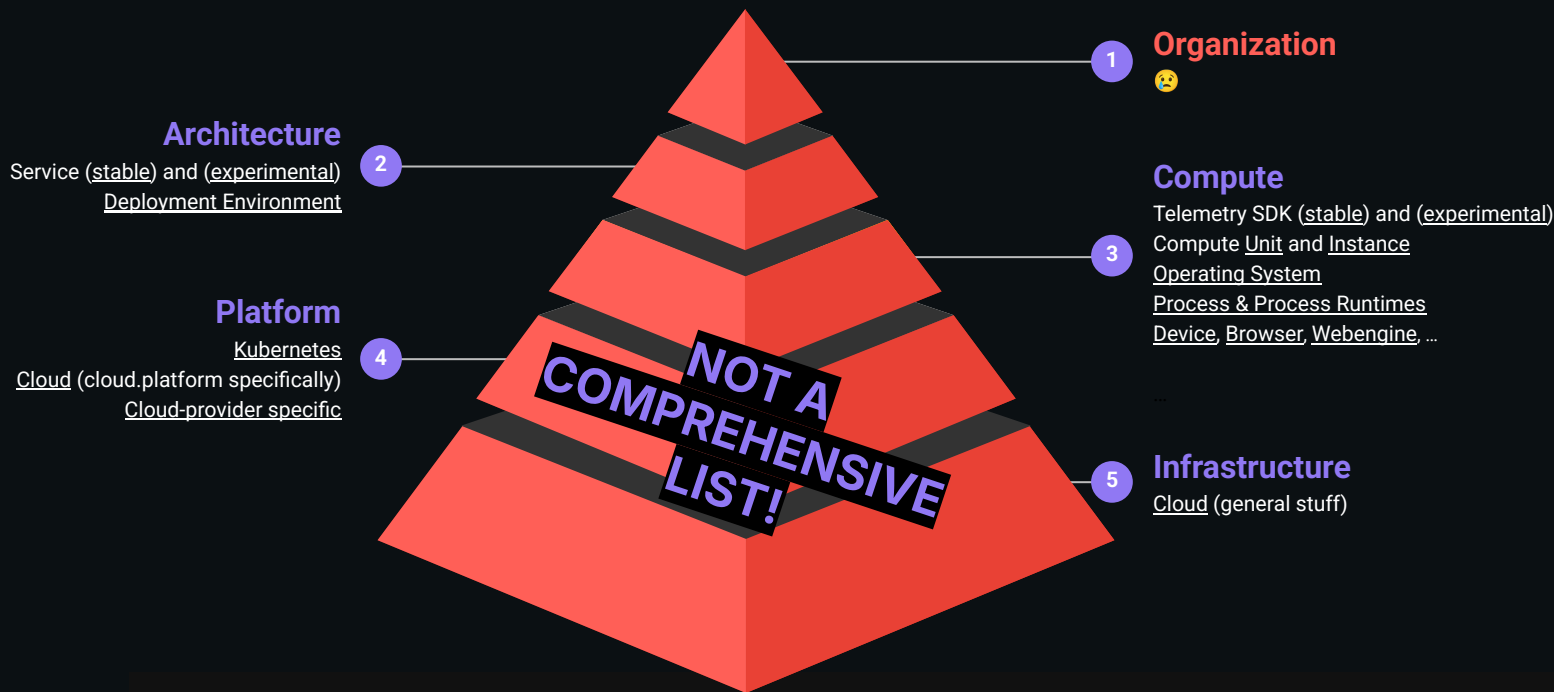


Reddit /r/funny, "Cuteness Vs Number of legs" (circa 2010)

How we talk about system context



OpenTelemetry semantic conventions to context layers



Resource is the **identity layer** - and it belongs at the **source**.

Every span, metric, and log is emitted by something. Resource describes who – the hinge on which correlation, routing, and ownership all turn.

Resource, concretely

A stable bundle of attributes attached once, at process start - then carried on every signal the process emits.

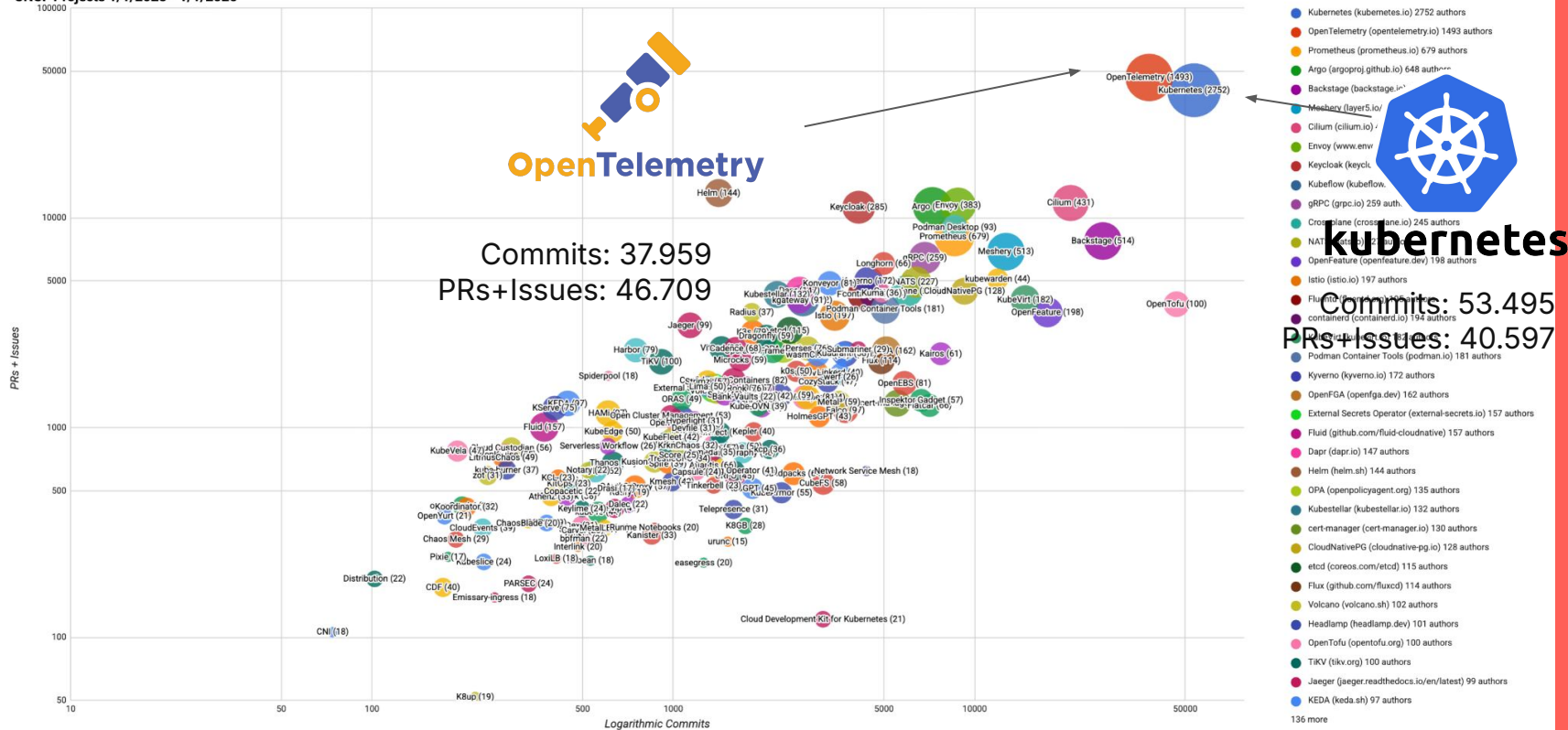
```
# Emitted with every span, metric, log
resource:
  service.name:      api-gateway
  service.namespace: payments
  service.version:   2.14.0
  deployment.environment: production
  k8s.cluster.name:  eu-west-1-prod
  k8s.namespace.name: payments
  k8s.pod.name:      api-gateway-7b9f-x2k
```

Source > Pipeline

- ▶ At the source, the component knows what it is - deployment metadata, config, versioning.
- ▶ In the pipeline, the Collector is guessing - joining on pod IPs, racing pod lifecycle, correlating by convention.
- ▶ Drift is silent. Reconstructed downstream, signals can quietly disagree about who - correlation breaks.

1/1/2025-1/1/2026

CNCF Projects 1/1/2025 - 1/1/2026



49%

of respondents using
OpenTelemetry in
production.

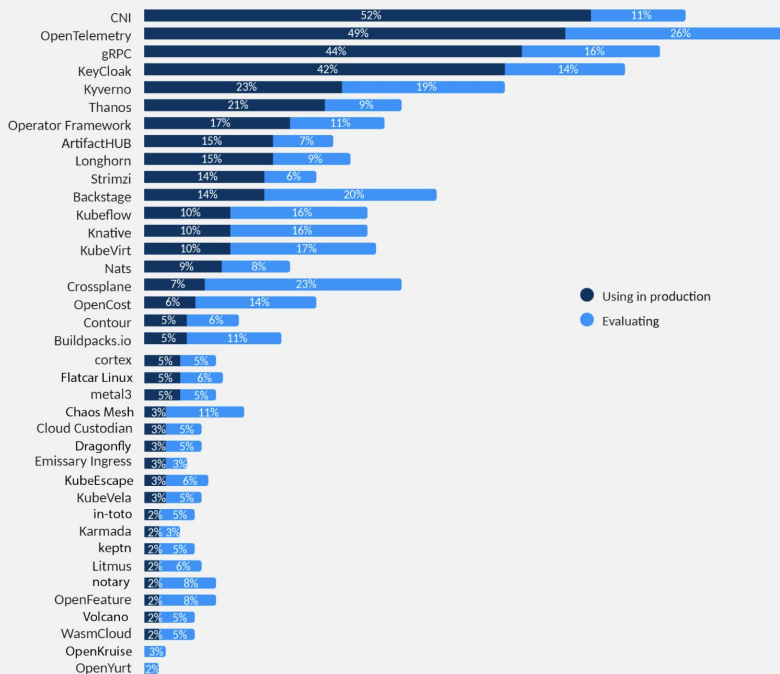
26%

of respondents evaluating
OpenTelemetry.

FIGURE 20

CNCF INCUBATING PROJECTS

Which of these incubating CNCF projects is your organization using in production or evaluating? (select one)



2025 CNCF Annual Survey, Q33, Sample size = 395-502, DKNS excluded

Foundational infrastructure **raises expectations** on the ecosystem.

A natural consequence...

01

Plug into existing pipelines

INTEGRATION

Connect to an OpenTelemetry Collector without custom adapters, sidecars, or per-project flags.

02

Speak a shared language

SEMANTICS

Consistent attributes across traces, metrics, and logs - so dashboards and alerts transfer.

03

Stable, predictable, documented

IDENTITY & STABILITY

Resource identity you can trust across environments. Telemetry treated as a long-lived contract.

Users don't just want OTLP ingest - they want a component that behaves like a first-class citizen of their observability platform.

Ecosystem support is **scattered**.

In practice...

WHAT MOST PROJECTS SHIP

Supports OTLP

→ a wire protocol

- Can export to a Collector
- Metric/log/trace bytes on the wire

≠

WHAT PLATFORM TEAMS NEED

Supports OpenTelemetry

→ a platform capability

- Semantic correctness & resource identity
- Trace modeling & propagation
- Multi-signal correlation
- Stable, documented configuration surface.

“Supports OpenTelemetry” is often interpreted as “can export OTLP”. That framing hides major differences in how well projects actually integrate.

The Collector becomes a compensation layer.

Every project that ships half-baked resource identity adds another processor to someone else's pipeline.

01

k8sattributes

reach back into the API server to discover what pods this even is.

02

resource / transform

rewrite `service.name`, synthesise `deployment.environment.name`.

03

attribute processor

paper over wrong SemConv names from individual projects.

04

filter / redact

strip PII the component leaked into attribute values.

05

.. & a per-project branch for every new onboarding

Pipeline grows linearly with the estate. Unowned. Untested.

THE SHIFT

Identity missing *at the source* becomes **config sprawl** in the platform

Correlation isn't emitted. You **manufacture** it.

What "supports OpenTelemetry" looked like in practice

• AT THE SOURCE - CONTOUR.YAML

```
# turn tracing on, wire the Collector by hand
tracing:
  serviceName: contour-gateway
  extensionService: { name: otel-collector }

# then hand-inject the trace id into a log string
accesslog-format: envoy
accesslog-format-string: |
  { ..., "traceparent": "%REQ(traceparent)%", ... }
```

• IN THE COLLECTOR - OTEL-COLLECTOR (OTTL)

```
# recover correlation downstream, by hand
transform/trace-extract:
  log_statements:
    - set(attributes["b"], ParseJSON(body))
      where IsMatch(body, ".*traceparent.*")
    - set(trace_id.string, Substring(b["traceparent"], 3, 32))
    - set(span_id.string, Substring(b["traceparent"], 36, 16))

k8sattributes: # ...and backfill identity too
  extract: [ k8s.namespace.name, k8s.pod.name, k8s.node.name ]
```



Where the **bill is paid**

When projects ship weak OpenTelemetry support, platform teams absorb the cost.

01

Collector YAML sprawl

Per-component transform rules, schema mappings, ad-hoc enrichment.

02

Silos & stitching

Logs, tracing, and metrics correlate only after pipeline massaging.

03

Fragile dashboards

A refactor upstream silently breaks alerts and runbooks downstream.

04

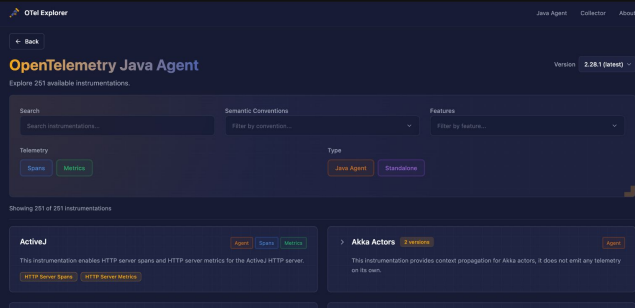
MTTR flatlines

More tooling. Same debugging. Same outcomes.

Platform components are **long-lived dependencies**. Weak OpenTelemetry support limits the value platforms can extract - from reliable debugging to automation and AI-assisted analysis.



We have **discovery**. We have **runtime quality**.



DISCOVERY

Ecosystem Explorer

A registry. Tells you *what exists* - binary inclusion, no maturity dimension.

explorer.opentelemetry.io

Instrumentation Score

A standardized, vendor-neutral metric for assessing OpenTelemetry instrumentation quality.

From 0 to 100. Objective. Actionable. Community-driven.

Calculated using weighted scoring based on rule criticality against OpenTelemetry semantic conventions and community best practices.

0 Poor — 100 Excellent

RUNTIME QUALITY

Instrumentation Score

Rule-based checks on live OTLP. Tells you *how clean* the emitted data is.

instrumentation-score.com

JANUARY 2026 – DRAFT PROPOSAL

OpenTelemetry Support Maturity Model.

A shared vocabulary for evaluating how intentionally a project supports OpenTelemetry - not whether it can speak the protocol.



Four **levels** - from instrumented to optimized.



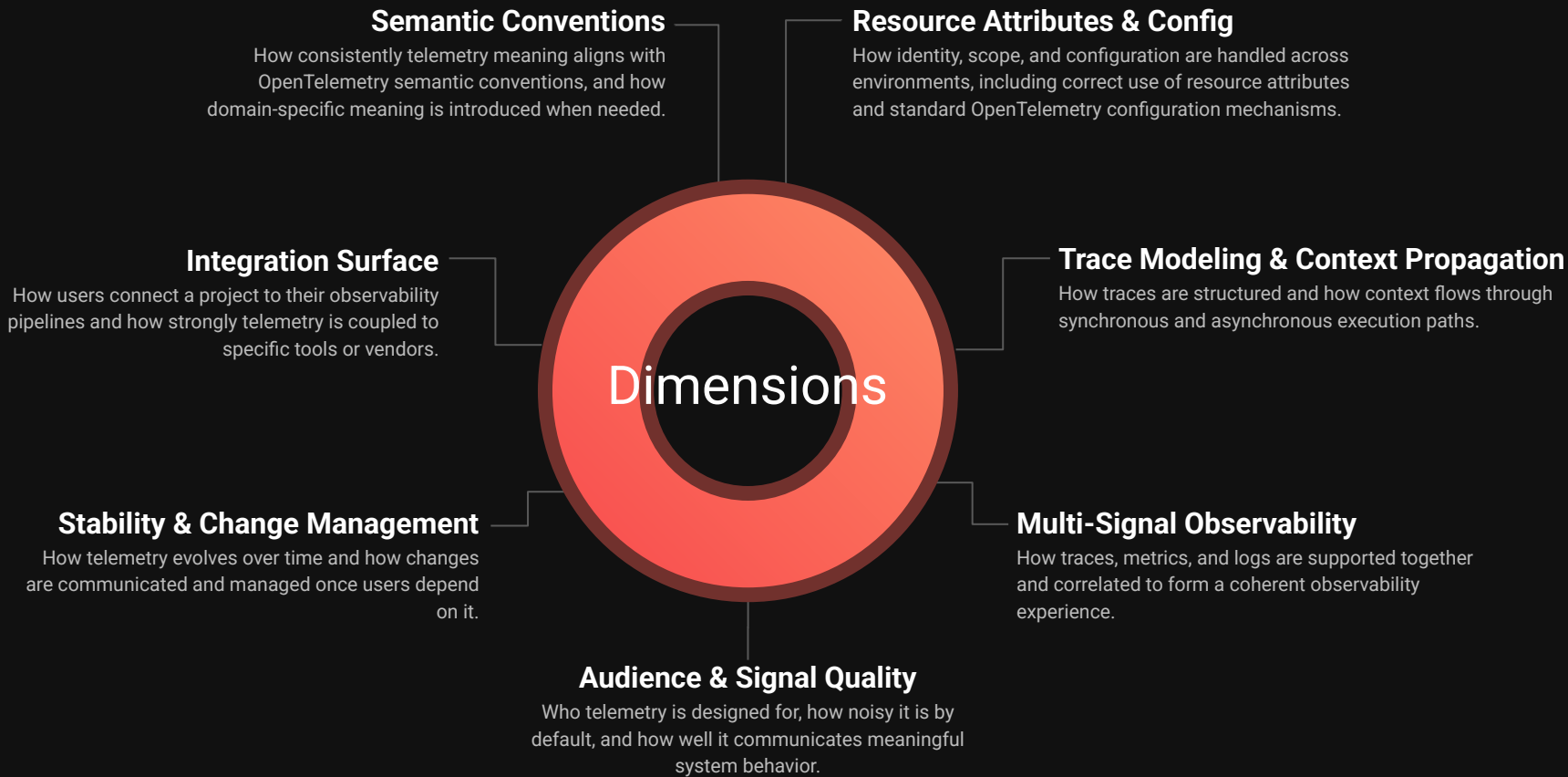
Telemetry exists - mostly to serve internal debugging. OpenTelemetry is not yet a design concern.

OTel supported explicitly - often alongside legacy exporters. Works for common cases; legacy assumptions still shape design.

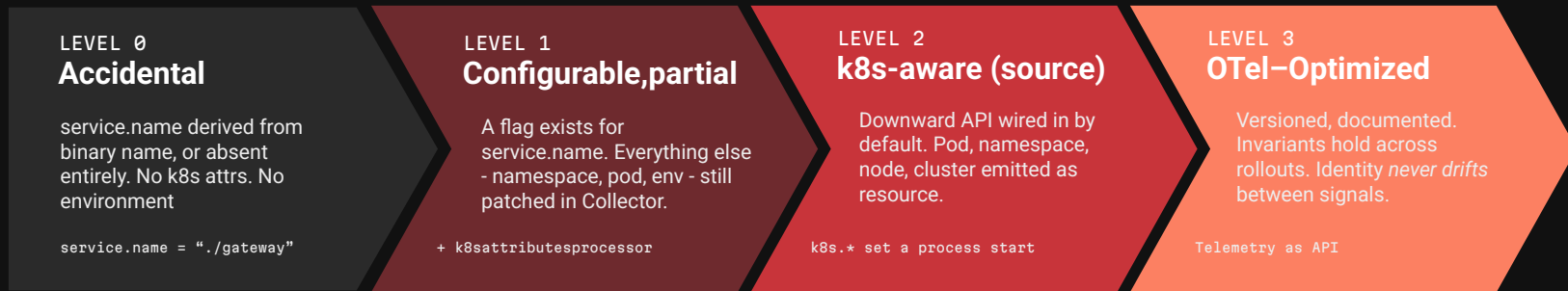
OpenTelemetry is the integration surface. Telemetry is designed intentionally for correlation and UX.

Telemetry as a long-lived product surface. Continuously refined for scale, cost, quality and stability.





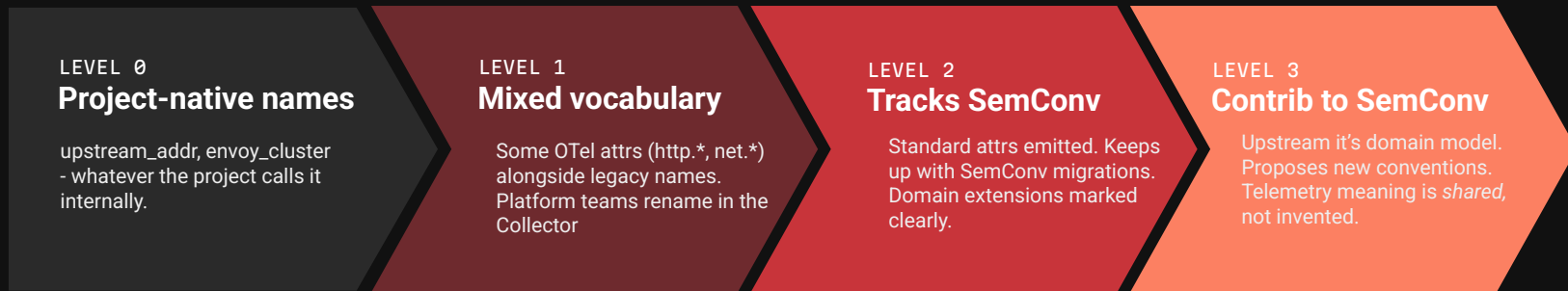
Resources - where identity is set decides who pays



TELLS FROM THE OUTSIDE

- Does it emit k8s.* attrs *without* the Collector k8sattributesprocessor?
- Is service.name first-class config - or reverse-engineered from a flag?

Semantic Conventions - the **vocabulary** that makes data transferable.

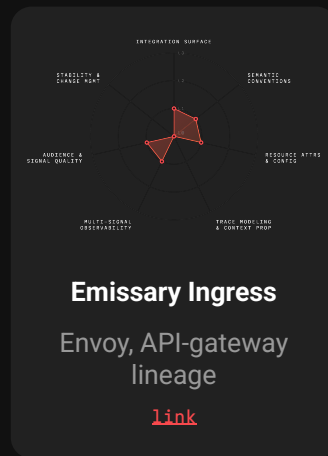
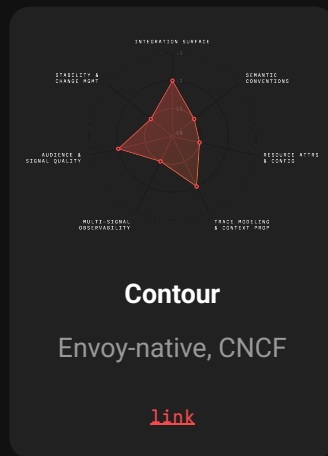
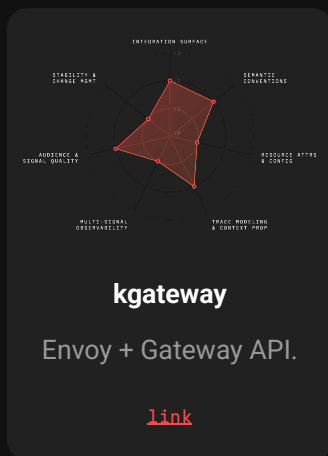
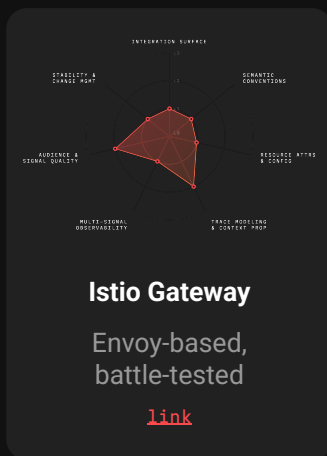
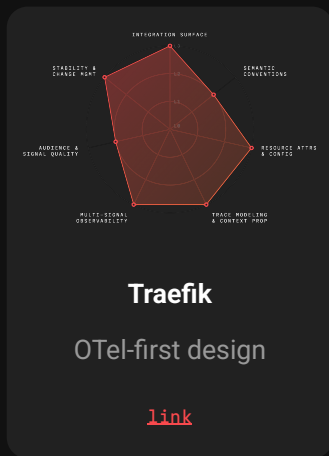


TELLS FROM THE OUTSIDE

- Spans use http.route, url.path, server.address - or project-local names?
- Do docs reference semconv by version?
- Did the http → client/server SemConv migration land?

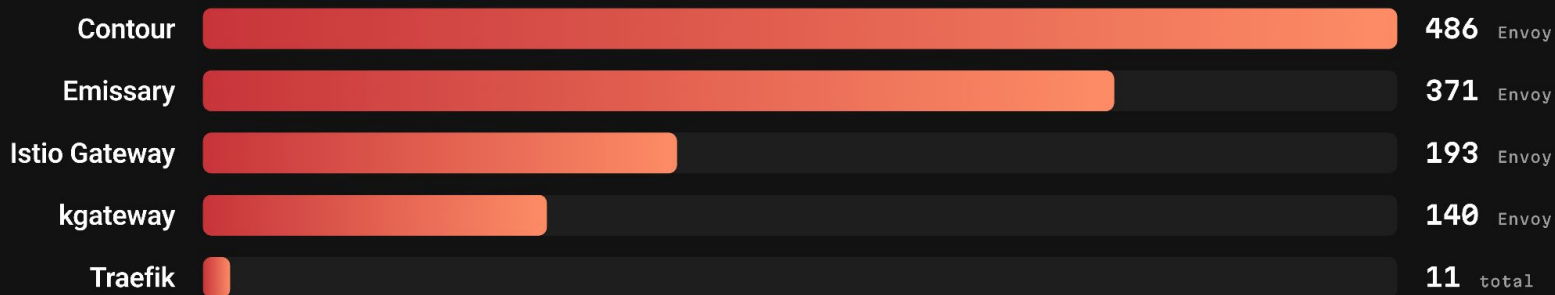
I tested the model against 5 ingress controllers

Ingress is the edge – every request passes through it. If any component should be OpenTelemetry-native



More telemetry ≠ better telemetry.

Default metrics exposed by each ingress controller – same job, the edge of your cluster.



A **44x spread** for the same job - no two projects agree on what “enough” looks like. More isn't better, **fewer isn't better either**. What counts is whether they're the right signals, follow semantic conventions, and let you answer it. **Count is not quality**.

Same box ticked. Very **different shapes**.

01 Only one is OTel-native end-to-end

Traefik. Everyone else is OpenTelemetry-aligned at best - legacy assumptions still shape the design.

02 Metrics still reflect Prometheus gravity

Traces go out via OTLP; metrics still come from a /metrics scrape. The Collector bridges the two. Hybrid by default.

03 Resource identity leaks into your pipeline

Most controllers expect you to enrich service.name, workload, and environment downstream in the Collector.

04 Correlation is not free

Trace-to-log correlation emerges in the pipeline, not at the source. You assemble it; the component doesn't hand it to you.

05 Stability is rarely discussed

Telemetry changes show up in refactors, not release notes. Dashboards break quietly.

06 "Supports OpenTelemetry" hides all of it

Every project ticks the same box. The shape tells a completely different story.



The model didn't survive review.

[community#3435](#) is now “OpenTelemetry Support Self-Assessment and Maintainer Guidance, co-led with Graziano Casto.

WHERE IT STARTED

A comparative maturity score — levels 0–3 on every dimension, one number-shaped verdict per project.



WHERE IT'S HEADING

Opt-in tooling maintainers run on their own telemetry, plus guides per project type. No score. Results belong to whoever ran it.



This only works in the open.

Next steps & how to help.

01

Find it a home

Land the model in an OpenTelemetry SIG so it has owners and a process — not a doc that rots.

02

Build the self-assessment tooling

Deterministic, objective, run locally by the maintainer, no score in the output. Reuse existing projects, like the semantic-conventions-conformance.

03

Write the maintainer guides

Libraries, then services and infrastructure - databases, brokers, proxies, gateways, controllers.

04

We need more help!

Tooling contributors, guide authors, and projects willing to run the prototype against their own telemetry and tell us if the report is useful.



OpenTelemetry is the foundation for **much faster** incident resolution

Lower MTTR requires more than data - it requires telemetry that means the same thing across every component on the request path.

WITHOUT SHARED SEMANTIC CONVENTIONS

Stitching by hand in the war room

Engineers navigate multiple tools, align timestamps, and guess which attribute means what. MTTR stagnates despite "advanced" tooling.

WITH SHARED SEMANTIC CONVENTIONS

Symptom → cause without friction

Start from a metric, drill into a trace, land on correlated logs. Automation & AI-assisted analysis become viable because the data has meaning.



Platform Engineers are the **leverage point**.

You pick the components that every team inherits - the ingress, the service mesh, the message bus, the CI runner. Your selection is the contract your organization runs on, and it quietly decides how much observability tax every product team pays for the next five years.

CONSUME

Select with intent

Use the model in technology radars and golden-path decisions. Prefer components whose shape matches your signal priorities.

PROVIDE

Observability as a contract

Guarantee baseline telemetry on the paved road. Not opt-in heroics - a provision every service gets for free.

ADVOCATE

Push upstream

File the issue. Ask the maintainer. "Which dimensions are you working on?" is a better question than "Does it support OTel?".



AI doesn't replace platform thinking.

It increases the need for it.



GARBAGE IN

Inconsistent,
fragmented,
uncorrelated telemetry



AI/LLM

Doesn't invent clarity.
Doesn't reason. Just
summarizes.

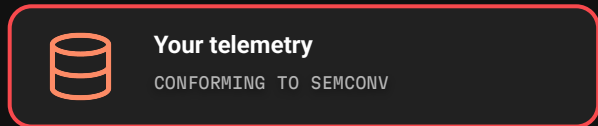
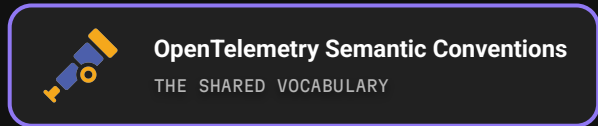


GARBAGE OUT

Automated confusion.
Faster. At scale.

Structure it to the semantic conventions.

Conventions are what turn telemetry into something an LLM understands.



YOUR DATA

Conform to them, and the magic is real.



AI/LLM

Now it has structure to
reason over.



UNICORNS AND RAINBOWS

Correlation. Root cause.
Answers you can trust.



Without conventions, correlation fails.
Without correlation, **AI guesses**.
With structure and context, **AI reasons**.



One vocabulary.

Shared vocabulary for GenAI workloads



```
# span.attributes
gen_ai.operation.name      = "chat"
gen_ai.system              = "openai"
gen_ai.request.model       = "gpt-4o"
gen_ai.request.temperature = 0.2
gen_ai.request.max_tokens  = 2048
gen_ai.usage.input_tokens  = 1283
gen_ai.usage.output_tokens = 412
gen_ai.response.finish_reasons = ["stop"]

# tool call
gen_ai.tool.name           = "kubernetes.list_pods"
gen_ai.tool.call.id        = "call_84ax2"

# resource (process)
service.name               = "agent-sre"
deployment.environment.name = "production"
```

STABILIZED CONCEPTS

- ▶ Operation name & system
- ▶ Model + parameters
- ▶ Token usage
- ▶ Finish reasons
- ▶ Tool calls

One vocabulary? **Not yet.** (picture from a couple of months ago)

Five conventions for the same span



OTel GenAI SemConv

OPTIMIZES FOR VENDOR
NEUTRALITY

`gen_ai.request.model`



OpenInference

OPTIMIZES FOR EVALUATION
WORKFLOWS

`llm.model_name`



OpenLLMetry

OPTIMIZES FOR DEVELOPER
ERGONOMICS

`gen_ai.request.model`
Still some traceloop
leftovers



LangSmith

OPTIMIZES FOR THE
LANGCHAIN ECOSYSTEM

Framework-native
taxonomy



Langfuse

OPTIMIZES FOR IT'S OWN
PLATFORM SCHEMA

`langfuse.observation.m
odel.name`

These are **legitimate differences**. Not naming preferences. They are not going away soon.

One vocabulary? **Not yet.** Now

Five conventions for the same span



OTel GenAI SemConv

OPTIMIZES FOR VENDOR
NEUTRALITY

`gen_ai.request.model`



OpenInference

DONATED TO
OPENTELEMETRY

`llm.model_name`



OpenLLMetry

DONATED TO
OPENTELEMETRY –
REJECTED

aligns with the
`gen_ai.*`



LangSmith

OPTIMIZES FOR THE
LANGCHAIN ECOSYSTEM

Framework-native
taxonomy



Langfuse

OPTIMIZES FOR IT'S OWN
PLATFORM SCHEMA

`langfuse.observation.m
odel.name`

These are **legitimate differences**. Not naming preferences. They are not going away soon.

The platform answer.

Normalize at the edge.

OTel Collector processor

genainormalizer



Rewrites OpenInference and OpenLLMetry spans into GenAI semconv — in the pipeline, not in the apps.

OpenTelemetry

Spring AI

Arconia



All five conventions behind one config property. Switch flavors without touching application code.

By Thomas Vitale

Conformance for OpenTelemetry?

WE'VE DONE THIS BEFORE

CERTIFIED KUBERNETES

Objective conformance tests

Run the suite (Sonobuoy, originally Heptio), pass, get the mark. Machine-checkable. "Certified" means something.

PROMETHEUS CONFORMANCE

"Prometheus-compatible"

A program that verifies PromQL & remote-write compliance — so the claim is testable, not marketing.

TWO COMPLEMENTARY LAYERS

THE TRAJECTORY

Self Assessment — descriptive guidance

How intentionally support evolves. A shape, not a verdict.

sits on top of

THE FLOOR

Conformance — pass / fail baseline

Does it emit (and ingest) OpenTelemetry correctly? Testable. The thing the TC actually asked for.



semantic-conventions-conformance

“Does an instrumentation actually emit what the semantic conventions say it should?” Same answer, every library, every language.

01

Exercise

Run a small program against the library. A mock LLM server keeps it deterministic.

02

Collect

Capture what it actually emitted, through Weaver live-check

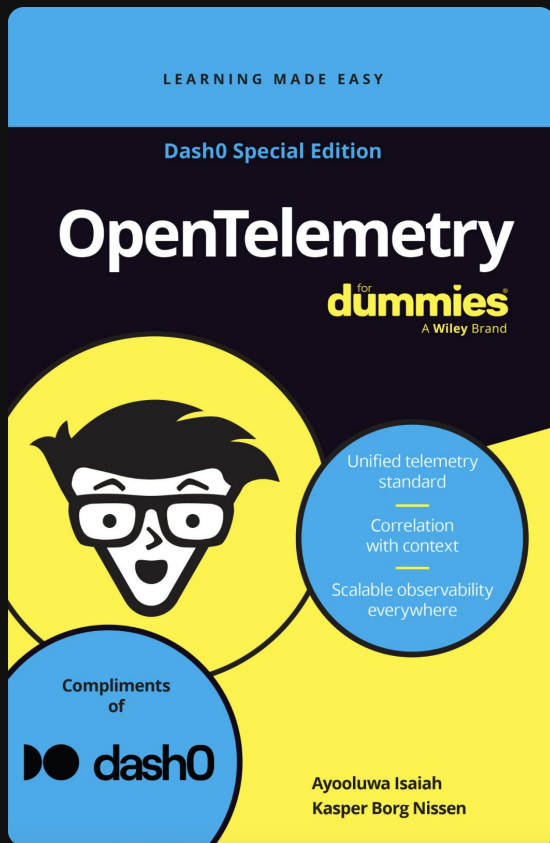
03

Check

Compare against expectations. Pass or fail

```
$ otel-conformance path/to/directory --report-only
```





WANT TO GO DEEPER?

Free copy of **OpenTelemetry for Dummies**.

The Dash0 Special Edition - by Ayooluwa Isaiah and Kasper Nissen. Unified telemetry, correlation with context, and scalable observability, in plain language.



DOWNLOAD:

dash0.com/lp/opentelemetry-for-dummies



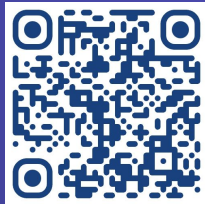
Follow us on
LinkedIn
Merge Forward (CNCF)

Join Merge Forward!

Building a stronger open source future together!

#merge-forward on Slack!

Learn more



community.cncf.io/merge-forward



Thank you!

Kasper Borg Nissen, Director of Developer Relations at Dash0



kaspernissen.xyz



[/in/kaspernissen](https://www.linkedin.com/company/kaspernissen)



[kaspernissen](https://github.com/kaspernissen)





Get in touch!



kaspernissen.xyz



[/in/kaspernissen](https://www.linkedin.com/company/kaspernissen)



[kaspernissen](https://github.com/kaspernissen)

