dynatrace

OpenTelemetry in Practice

How to build production-grade OTel pipelines

PRESENTER

Mikko Viitanen, Dynatrace

Principal Product Manager, OpenTelemetry

OpenTelemetry

A system ran for a thousand days without error.
On the thousand-and-first day, it failed.

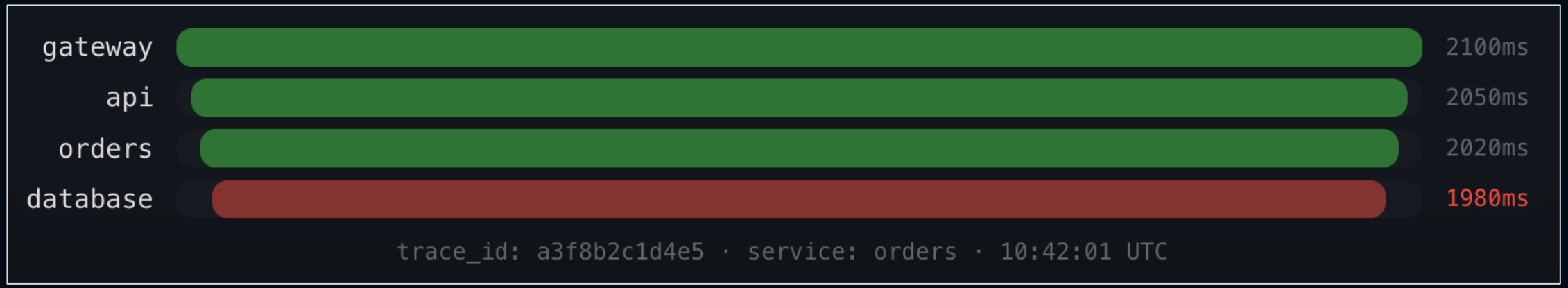
The developer asked, *“When did this begin?”*
The system had never learned to speak.

A system speaks, but not with one voice

Which signal is it, and what does it tell you?

A metric tells something changed. What would you need to understand more?

A different lens on the same problem



Which signal is it, and what does it tell you?

The trace shows WHERE the time was spent. But WHY is the database slow?

The moment of failure, captured

CORRELATED LOG – DATABASE – 10:42:01.980

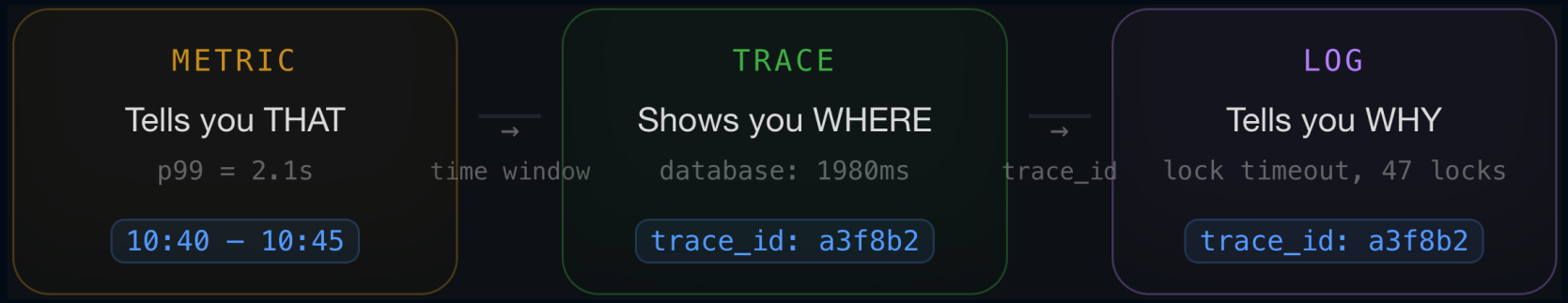
```
10:42:01.980 ERROR  lock timeout waiting for table 'orders'
             concurrent_locks=47 wait_time=1980ms table=orders
```

A log line shows what happened at that specific moment.

Three signals. Three partial truths.

- **Metrics:** *What changed (the symptom)*
- **Traces:** *Where time is spent (the path)*
- **Logs:** *Why it happened (the context)*

Each signal is a different lens. You need all three.



Instrumentation: Teach the system to speak

- **Auto-instrumentation ->** Coverage
- **Manual instrumentation ->** Precision

Most teams need both to get what matters

Instrumentation in the age of coding agents

- Asking your agent to ”add OTel spans" now works. Why?
  - OTel has extensive public docs, examples, blog posts, GitHub discussions
  - Coding agents have absorbed all of it

What used to take half a day is now a 30-second prompt

Instrumentation is no longer the bottleneck. Choosing what to measure is.

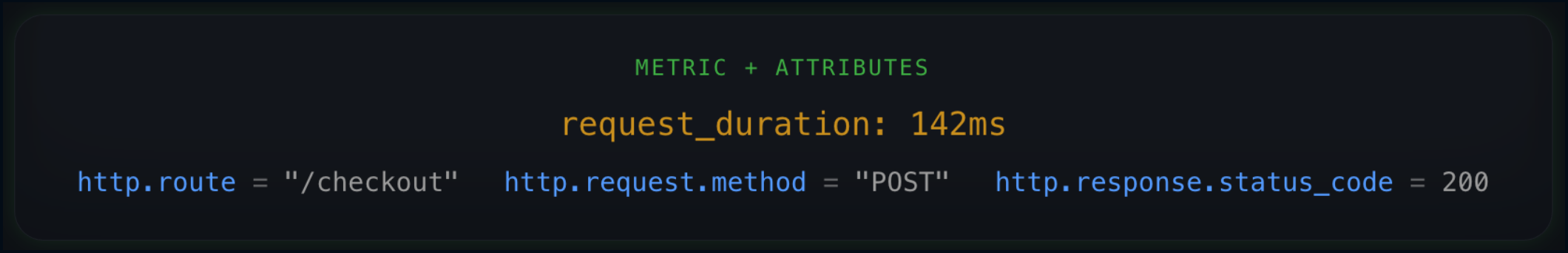
Attributes: the who, what, where, and why

```
METRIC
request_duration: 142ms
```

A developer sees this and asks:

- **Which endpoint?**
- **What method?**
- **Did it succeed?**

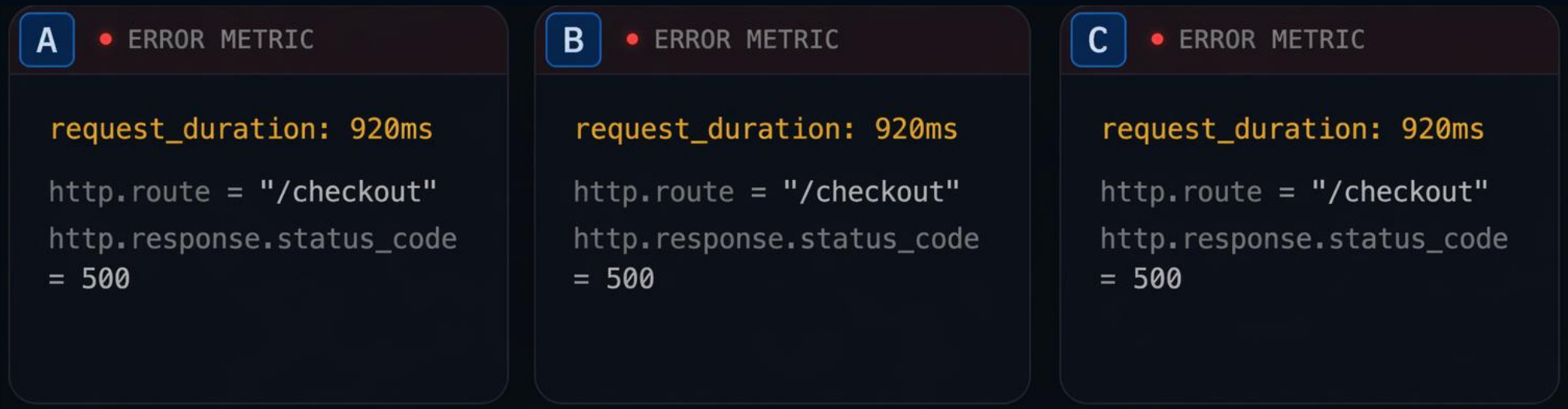
Attributes give telemetry its meaning



Filtering, grouping, alerting, and dashboards - all depend on attributes
Every metric, span, and log can carry them

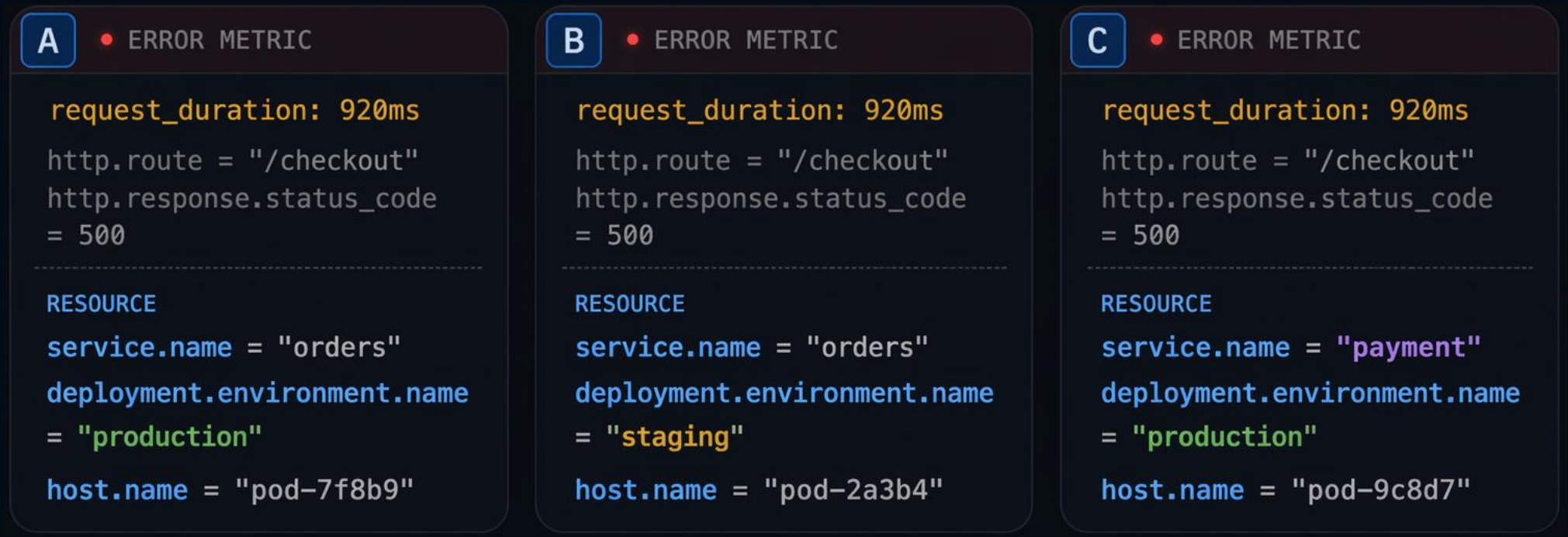
Resource attributes

- Three alerts fire at once
- Three identical errors
- But where did they come from?



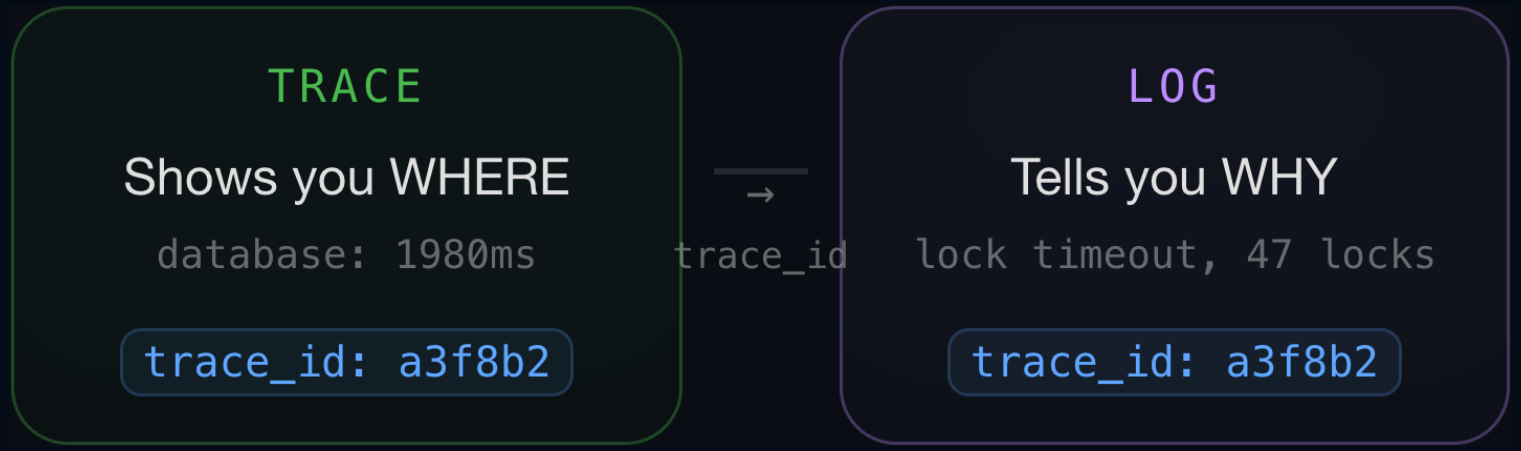
Resource attributes — the identity card

- Resource attributes tell you who reported it



Context propagation – Correlation at scale

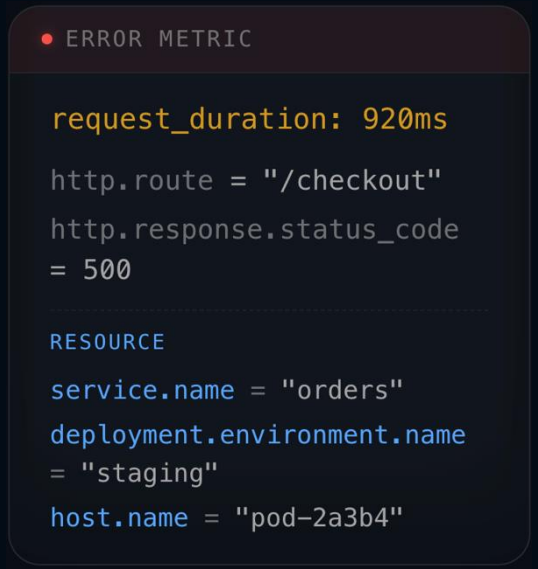
- Consider: hundreds of traces and logs at the same time.
- How can you correlate a log record to a trace?



Trace context (trace_id/span_id) flows through every call
This is what makes traces / logs clickable

How it all connects

- **Attributes:** *What happened*
- **Resource attributes:** *Where it happened*
- **Context propagation:** *Connects everything (across services, across signals)*



The naming problem

- Your HTTP handler returned an error. You want to record the status code as an attribute.
- **What will you name the key?**

```
Team A:  statusCode

Team B:  response_status

Team C:  http_code
```

Inconsistent attributes destroy observability

- Query your backend for: `http_code = 500`

Team	Attribute key	Result
A	statusCode	❌ NO MATCH
B	response_status	❌ NO MATCH
C	http_code	✅ MATCH

Most of your data is invisible

OTel semantic conventions: a shared vocabulary

- HTTP status code is always `http.response.status_code`
- Service identity is always `service.name`
- DB system is always `db.system.name`

When everyone speaks the same language, one query finds everything

OTel resource detectors

- Auto-populate attributes at startup
  `host.*, os.*, process.*, cloud.*, k8s.*, container.*`
- Built into OTel SDKs and Collector

Zero drift: No typos, no forgotten attributes, no team-by-team variation

Consistency and correlation for free

All of this works… only if you can process it at scale.

The Collector

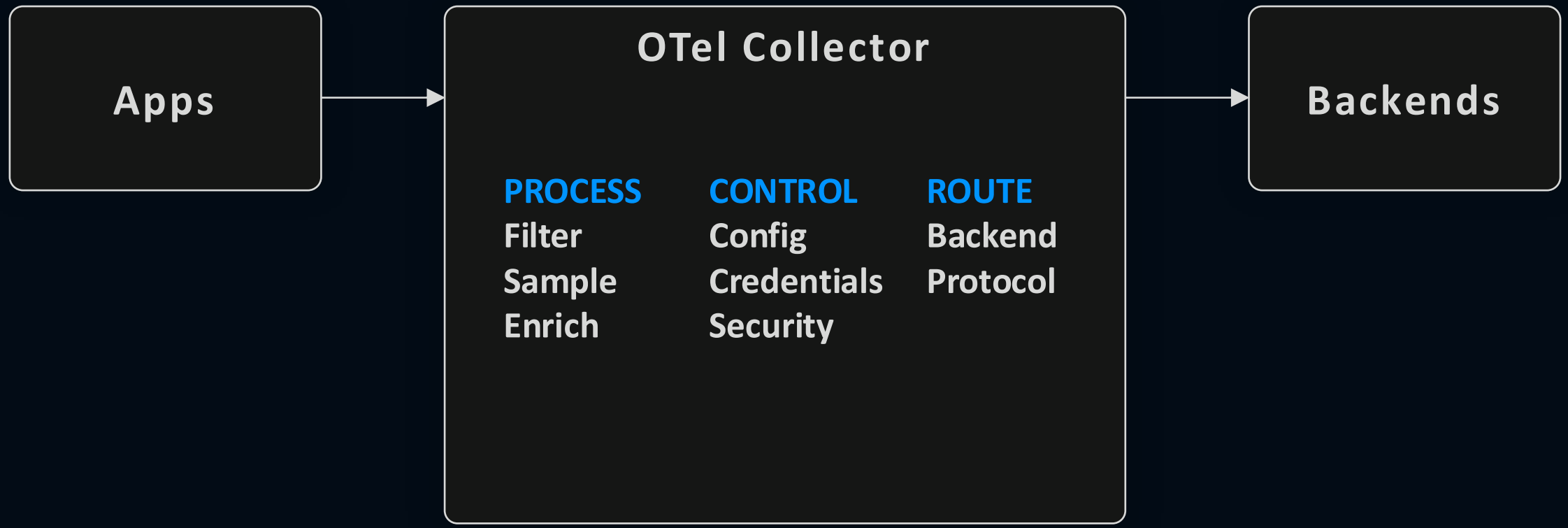
This is where OpenTelemetry becomes production infrastructure

Do you need an OTel Collector?

- **One service, one backend — do you need a Collector?**
- No

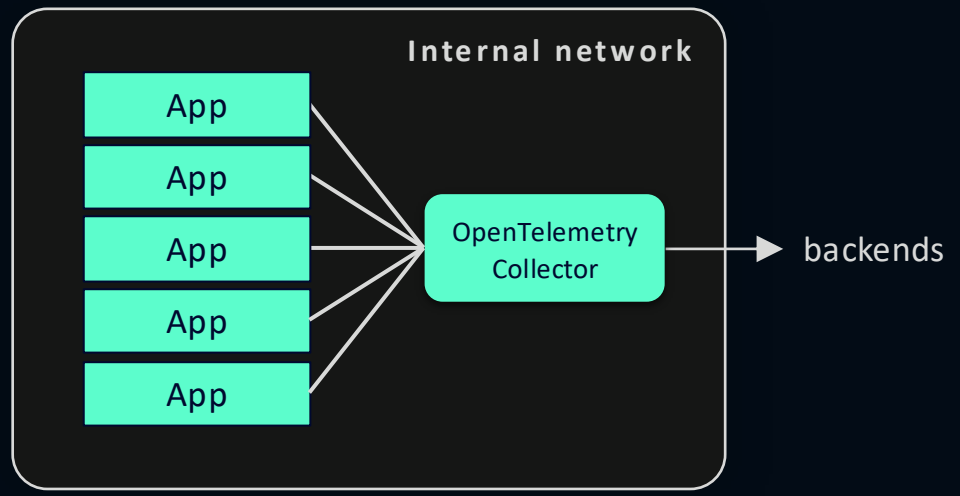
- **Ten services, one backend — do you need a Collector?**
- Yes. But why?

Apps emit. The Collector decides what happens next.



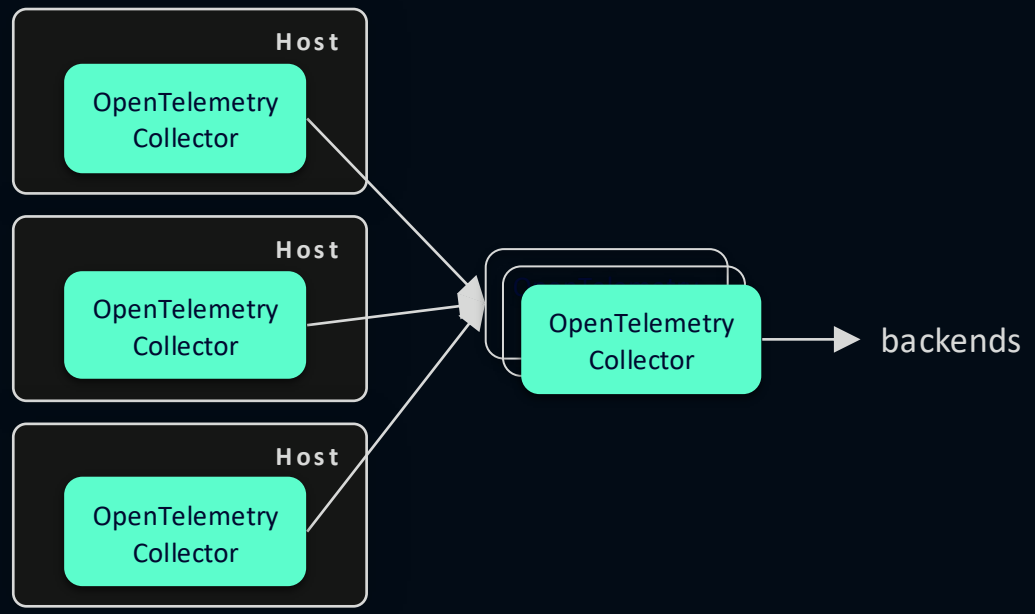
Collector: keep it simple

- Start with one Collector. Don't architect for scale you don't have.
- A single Collector handles a surprising amount of load.



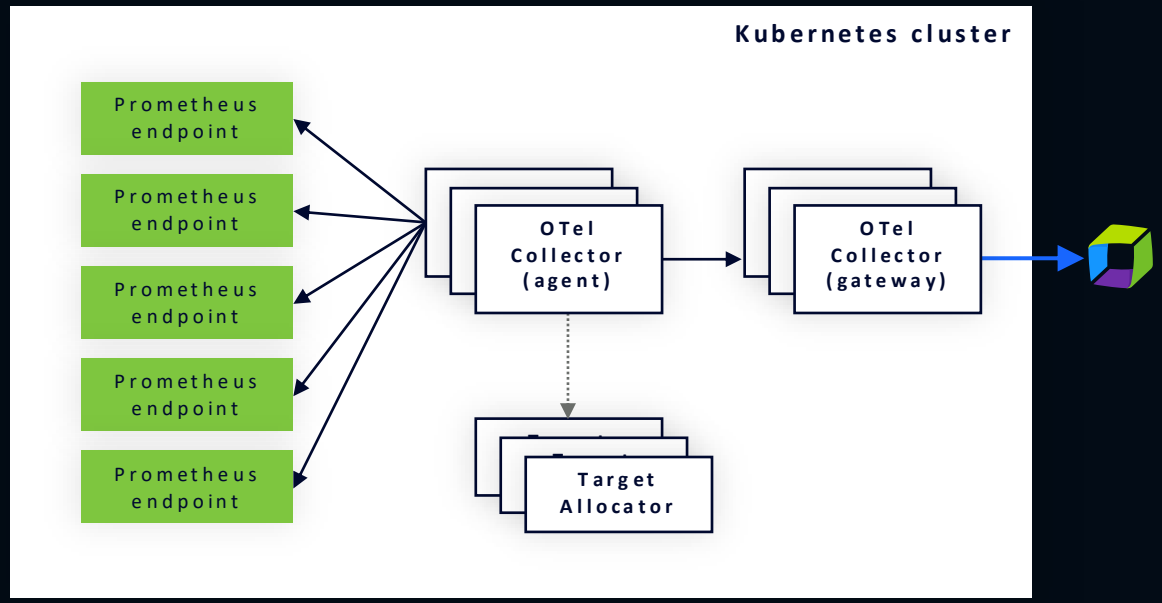
Collector: large-scale deployment

- Two-tier pipeline once you outgrow a single Collector
- Tier 1 (agent) — runs near the workload: DaemonSet or sidecar
- Tier 2 (gateway) — central pool: sampling, routing, fan-out



Scaling Prometheus ingest on Kubernetes

- **Agent Collectors**
  - Scraping Prometheus metrics
- **Gateway Collectors**
  - Centralized processing, secret management and exporting
  - Enrichment with k8s metadata
- **Target Allocator**
  - Discovery of Prometheus endpoints
  - Distribution of Prometheus targets



Decoupling service discovery, metric collection, processing, and exporting
Scaling for 1000+ node clusters

OpenTelemetry has graduated (May 2026)

- Built by hundreds of companies and thousands of contributors.
- It was already the standard. Now it’s official.
- CNCF Graduation = production maturity, stability, and trust

Graduation means you can build on it with confidence

This talk was **inspired by**
OTel Koans, Matt Reider
https://otel.mreider.com/

28 interactive exercises.
No backend, no prior knowledge required.

Questions?

One signal sees the shadow
Another finds the shape
A third reveals the cause