

Breaking Free with Open Standards: **OpenTelemetry and Perses for Observability**

Kasper Borg Nissen, Developer Advocate at  dash0

Who?

Developer Advocate at Dash0

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

CNCF Ambassador

Golden Kubestronaut

CNCG Aarhus, KCD Denmark Organizer

Co-founder & Community Lead Cloud Native Nordics



**GOLDEN
Kubestronaut**



Observability course for Platform Engineering

Gain a solid foundation of modern observability in platform engineering, from essential concepts to practical tooling.



Sign up for free

<https://university.platformengineering.org/observability-for-platform-engineering>



Michele Mancioppi

Head of Product @ Dash0



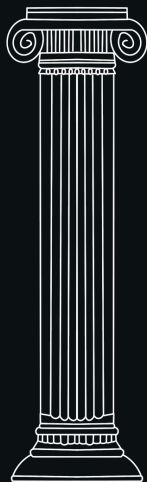
Kasper Borg Nissen

Developer Relations @ Dash0

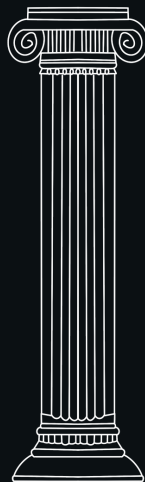
tl;dr

- **OpenTelemetry** is standardizing telemetry collection.
- **Perses** is standardizing dashboarding.
- Applying **Platform Engineering** principles transforms observability into a seamless, scalable, and developer-friendly experience.
- Building on **Open Standards** allows you to freely move between vendors, ensuring they stay on their toes and provide you the best possible experience.

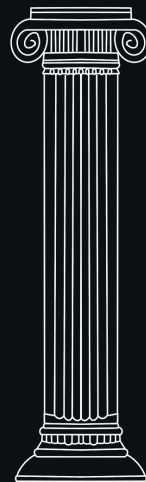
Observability is still fragmented



Metrics



Logs



Traces

Observability is still fragmented



We don't have a *metrics* problem,
or a *tracing* problem.
We have *systems* problems.



Metrics

Logs

Traces



This fragmentation, leads to

This fragmentation, leads to



Complex Query
Languages

This fragmentation, leads to



Complex Query
Languages



Vendor lock-in

This fragmentation, leads to



Complex Query
Languages



Vendor lock-in



Metadata Inconsistency

This fragmentation, leads to



Complex Query
Languages



Vendor lock-in



Metadata Inconsistency



No instrumentation due to
high complexity

This fragmentation, leads to



Complex Query
Languages



Vendor lock-in



Metadata Inconsistency



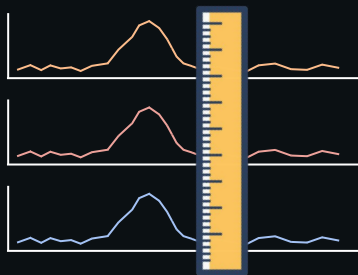
No instrumentation due to
high complexity



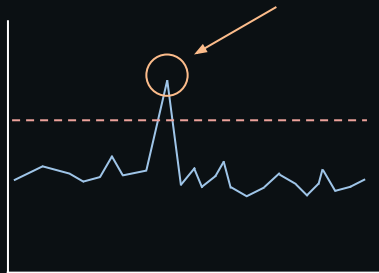
Lack of unified insights

A **shift** is happening.

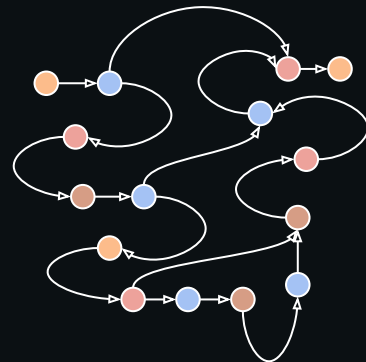
A shift toward correlation



Find related
information

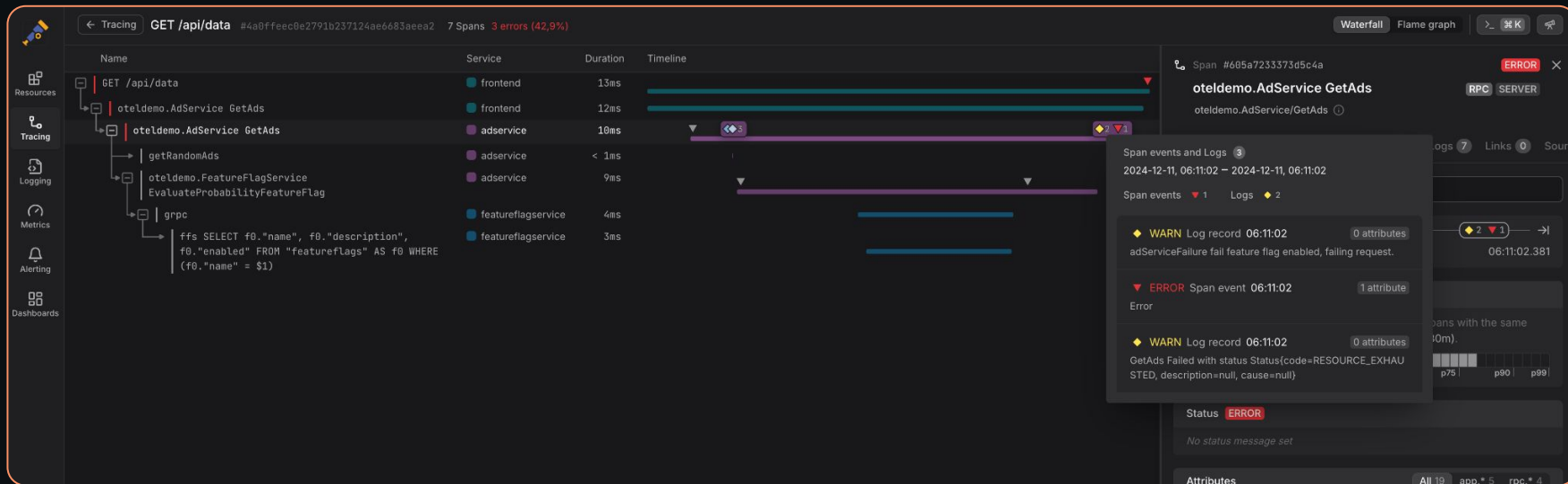


Jump between
signals



Reconstruct chain of
events

A shift toward correlation



A shift toward...



OpenTelemetry

OpenTelemetry (OTel) is an open source project designed to provide standardized tools and APIs for generating, collecting, and exporting telemetry data such as traces, metrics, and logs

The de-facto standard for distributed tracing, also supports metrics and logs (soon profiling)

The main goals of the project are:

- Unified telemetry
- Vendor-neutrality
- Cross-platform



OpenTelemetry in a nutshell



What it is

2nd largest CNCF project by contributor count

A set of various things focused on letting you collect telemetry about systems:

- **Data models**
- **API specifications**
- **Semantic conventions**
- **Library implementations in many languages**
- **Utilities**
- **and much more**

OpenTelemetry in a nutshell

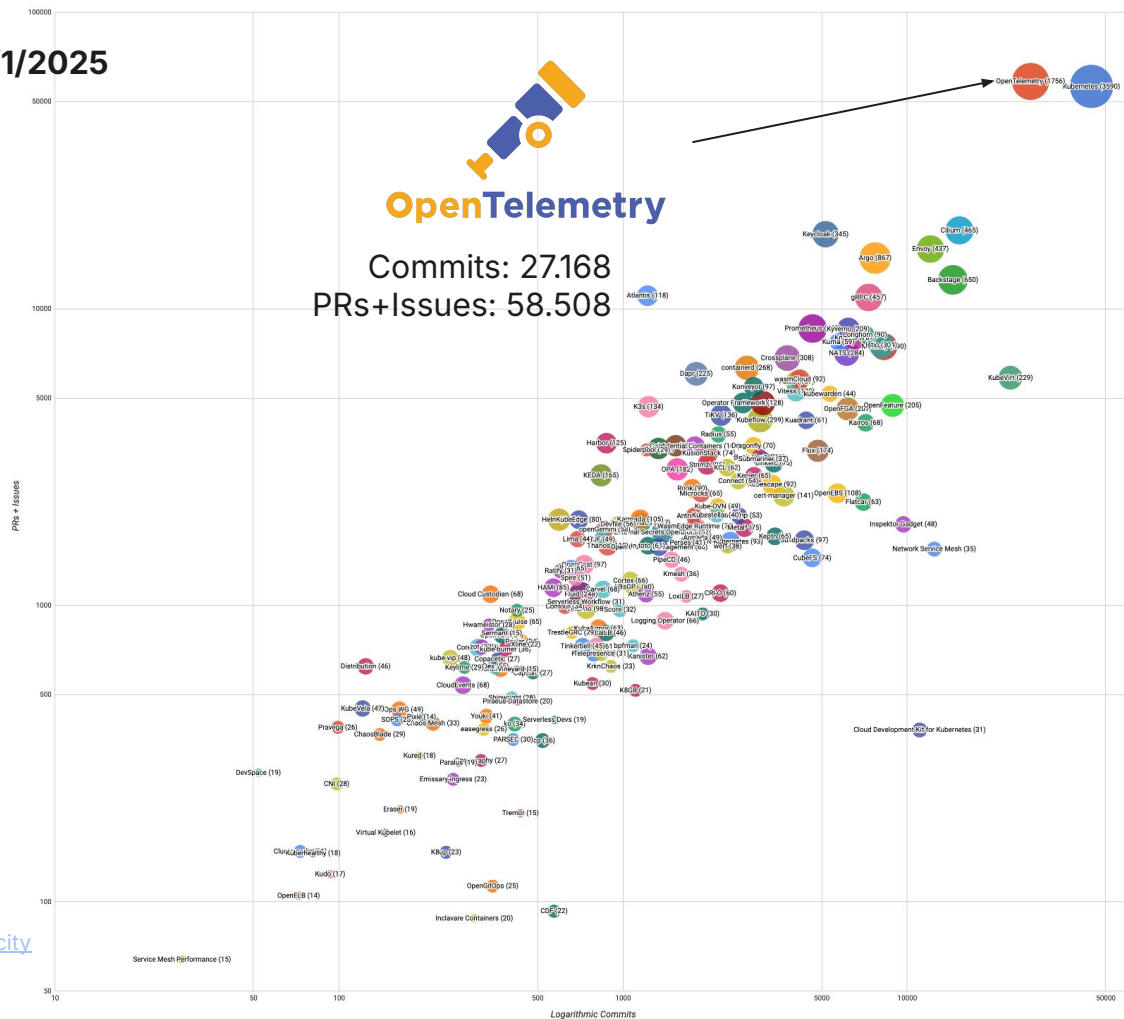


What it is NOT

- **Proprietary**
- **An all-in-one observability tool**
- **A data storage or dashboarding solution**
- **A query language**
- **A Performance Optimizer**
- **Feature complete**



1/1/2024-1/1/2025

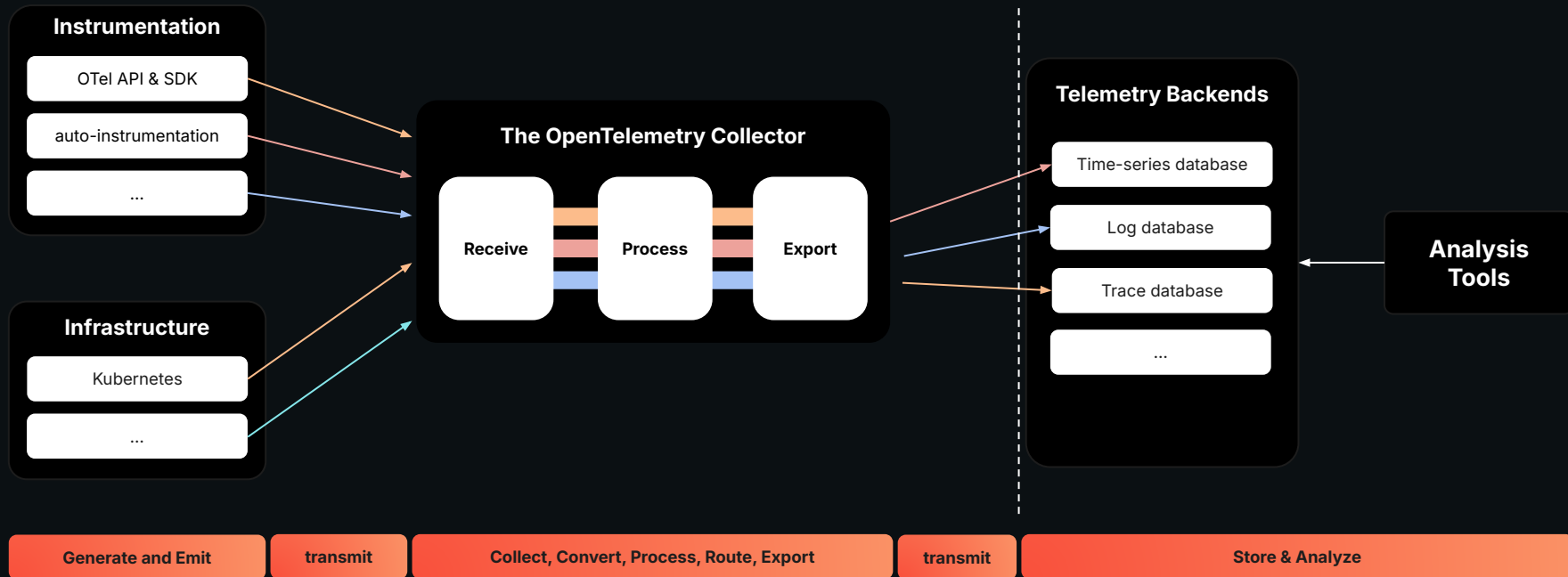


kubernetes

Commits: 44.486

PRs+Issues: 56.299

OpenTelemetry: A 1000 miles view



OpenTelemetry: A 1000 miles view

Collection of Telemetry is
standardized



"The last observability agent you will ever install"

Vendor space



... and many more.

Generate and Emit

transmit

Collect, Convert, Process, Route, Export

transmit

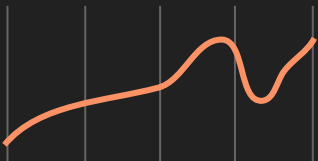
Store & Analyze



Signals

METRICS

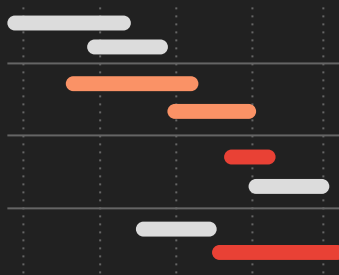
42



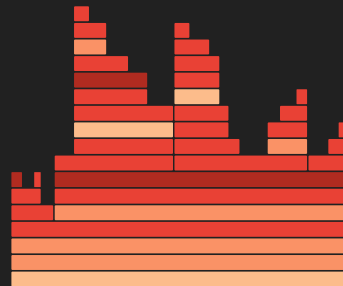
LOGS

■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200
■ [20/JUN/2025] "GET / HTTP/1.1" 200

TRACES



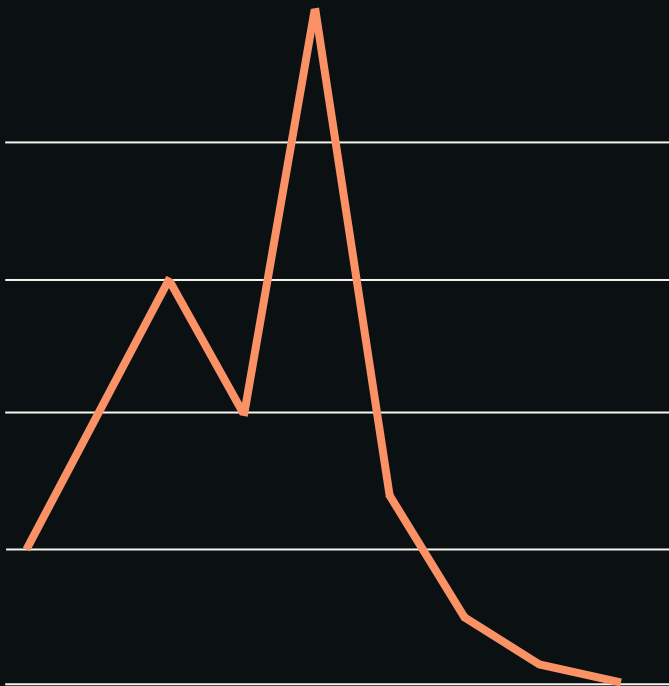
PROFILES



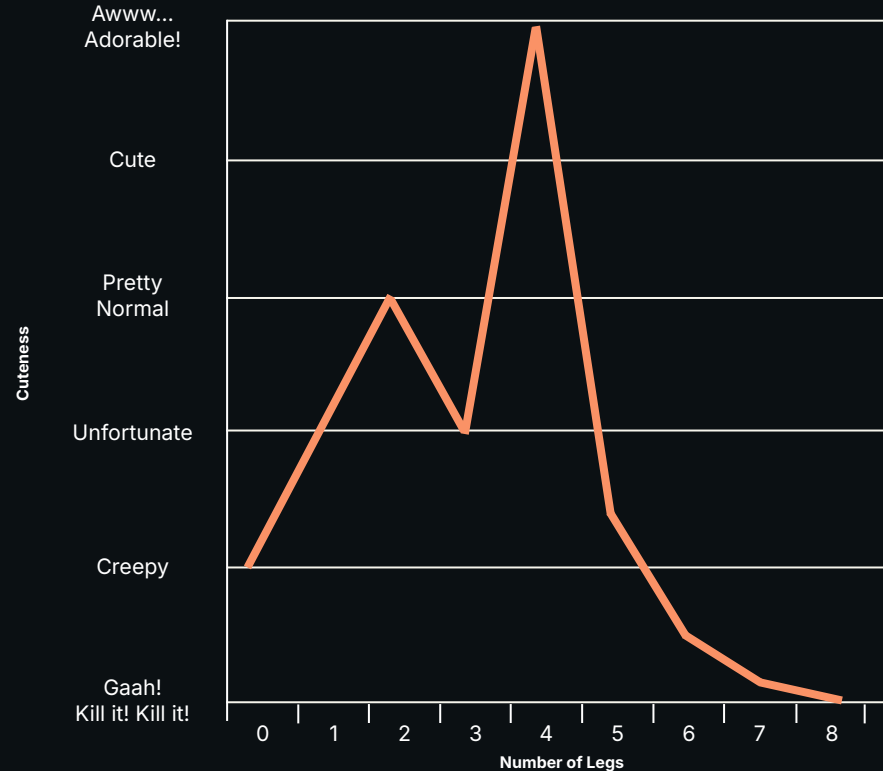
+ Real user monitoring (browser, app)

» Telemetry without context is just data «

What are we looking at?



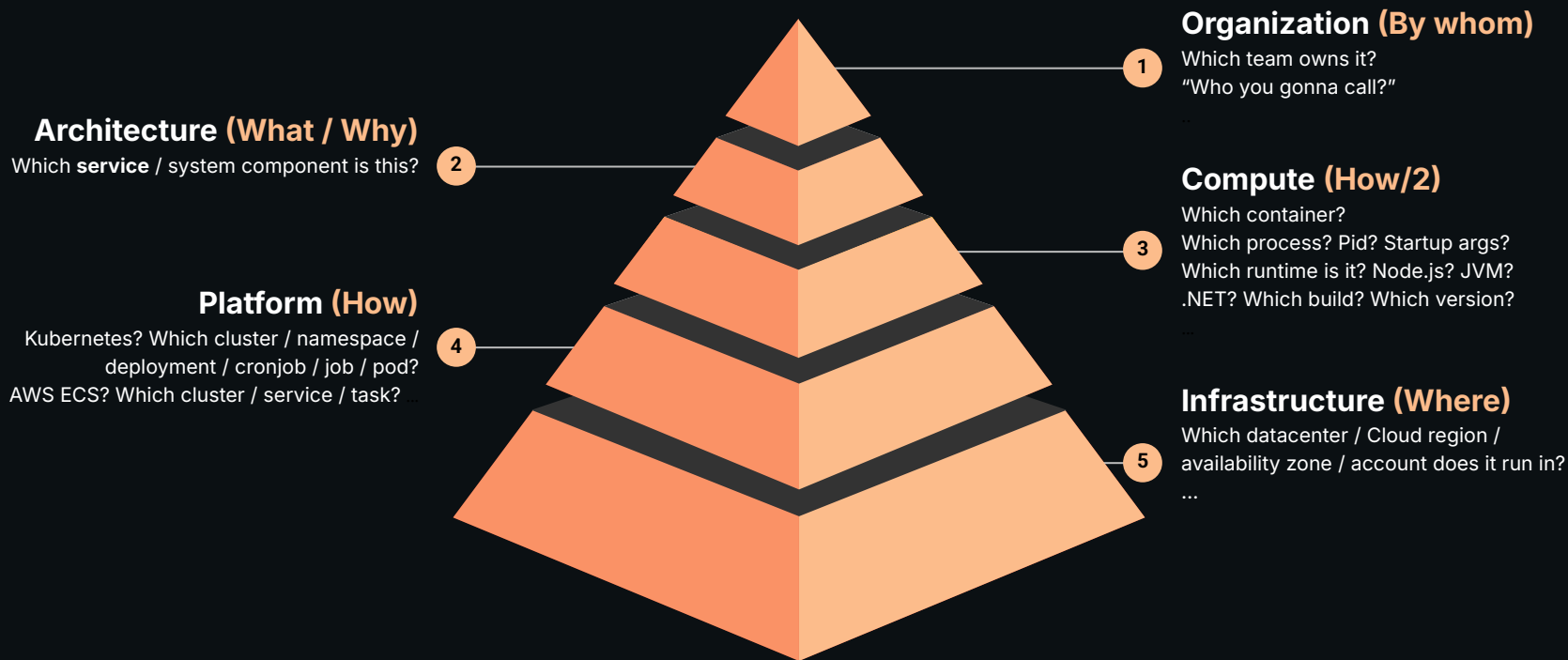
What are we looking at?



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



How we talk about system context



How to set resource attributes?

- Resource detectors & manual “hard-coding”.
- `OTEL_RESOURCE_ATTRIBUTES` env var
- Added to telemetry “in transit” using the OpenTelemetry Collector.

```
import { NodeSDK } from '@opentelemetry/sdk-node';
import { ConsoleSpanExporter } from '@opentelemetry/sdk-trace-node';
import { envDetector, processDetector, Resource } from '@opentelemetry/resources';
import { awsEcsDetector } from '@opentelemetry/resource-detector-aws';

const sdk = new NodeSDK({
  traceExporter: new ConsoleSpanExporter(),
  // Skip metric exporter, auto-instrumentations and more. See
  // https://opentelemetry.io/docs/languages/js/getting-started/nodejs/
  instrumentations: [getNodeAutoInstrumentations()],
  // Specify which resource detectors to use
  resourceDetectors: [envDetector, processDetector, awsEcsDetector],
  // Hard-coded resource
  resources: [new Resource({
    team: 'awesome',
  })],
});

sdk.start();
```

Sample initialization of the OpenTelemetry JS Distro in a Node.js application



OpenTelemetry without context
semantic conventions is just data

Semantic Conventions

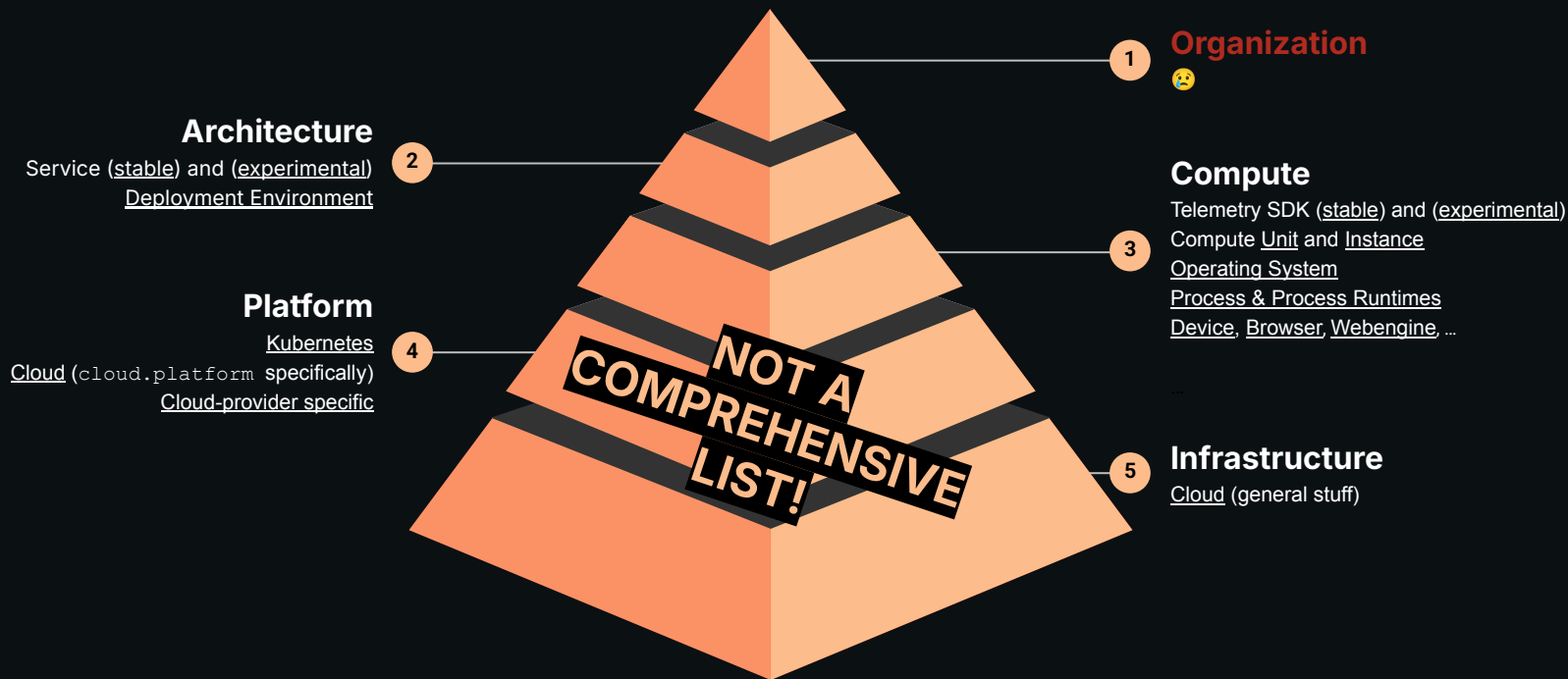
Semantic Conventions define a common set of (semantic) attributes which provide meaning to data when collecting, producing and consuming it.

<https://github.com/open-telemetry/semantic-conventions>

Semantic Conventions by signals:

- **Events**: Semantic Conventions for event data.
- [Logs](#): Semantic Conventions for logs data.
- [Metrics](#): Semantic Conventions for metrics.
- [Resource](#): Semantic Conventions for resources.
- [Trace](#): Semantic Conventions for traces and spans.

OpenTelemetry semantic conventions to context layers



So, why OpenTelemetry?



Instrument once,
use everywhere



Separate telemetry
generation from
analysis



Make software
observable by
default



Improve how we use
telemetry

**That's all great, but how do I make it
easily accessible for my developers?**

The **dual** role of Platform Engineers in Observability



Observe the running infrastructure







(metrics, logs)



Provide Observability as a product for developers

(traces, metrics, logs,
profiling)

What types of Telemetry do I need?

-  End-user devices and IoT
-  Runtimes, applications and services
-  Cloud, FaaS, Container orchestration
-  Operating system
-  Virtualisation
-  Bare metal

Prevalent telemetry types



Infrastructure context

Application context

Platform Engineering for Observability



Self-Service Experience



Explicit and Consistent APIs



Golden Paths



Modularity



Platform as a Product



Core Requirements

Platform Engineering for Observability



Self-Service Experience
Auto-Instrumentation



Explicit and Consistent APIs
Semantic Conventions



Golden Paths
Observability built-in



Modularity
Collector Pipelines



Platform as a Product
Documentation + Support



Core Requirements
Cross-signal correlation

That's all great, but I ask again,
how do I make it **easily accessible for
my developers?**

The answer:
Auto-instrumentation + Operators

=

No-touch Instrumentation

OpenTelemetry Operator



Instrumentation



OpenTelemetryCollector



OpAMPBridge



TargetAllocator



**OpenTelemetry
Operator**

Auto-Instrumentation with the OpenTelemetry Operator



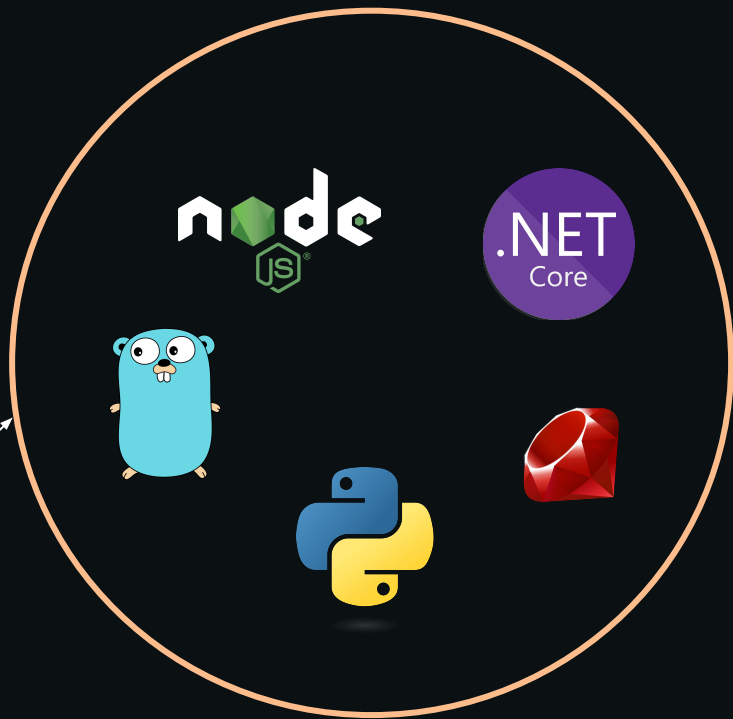
Instrumentation

*Instructs how to inject
auto-instrumentation*

*Injects
instrumentation in
to the pod*



OpenTelemetry
Operator



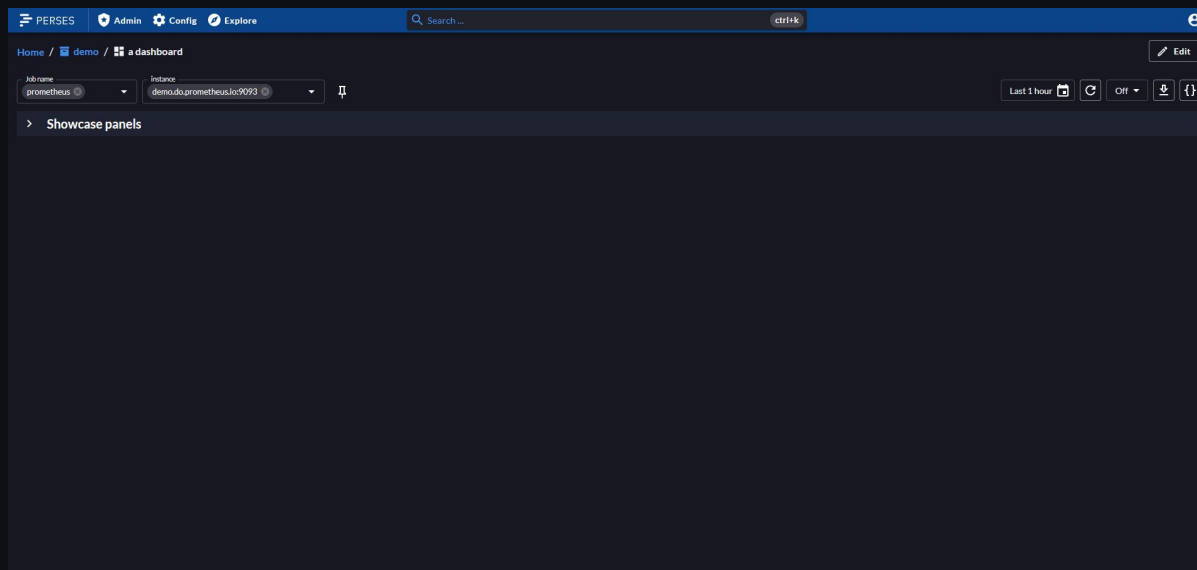
Observability doesn't **stop** at
instrumentation.

Persees



An open specification for
dashboards.

CNCF Sandbox project



Dashboards as Code



Perses



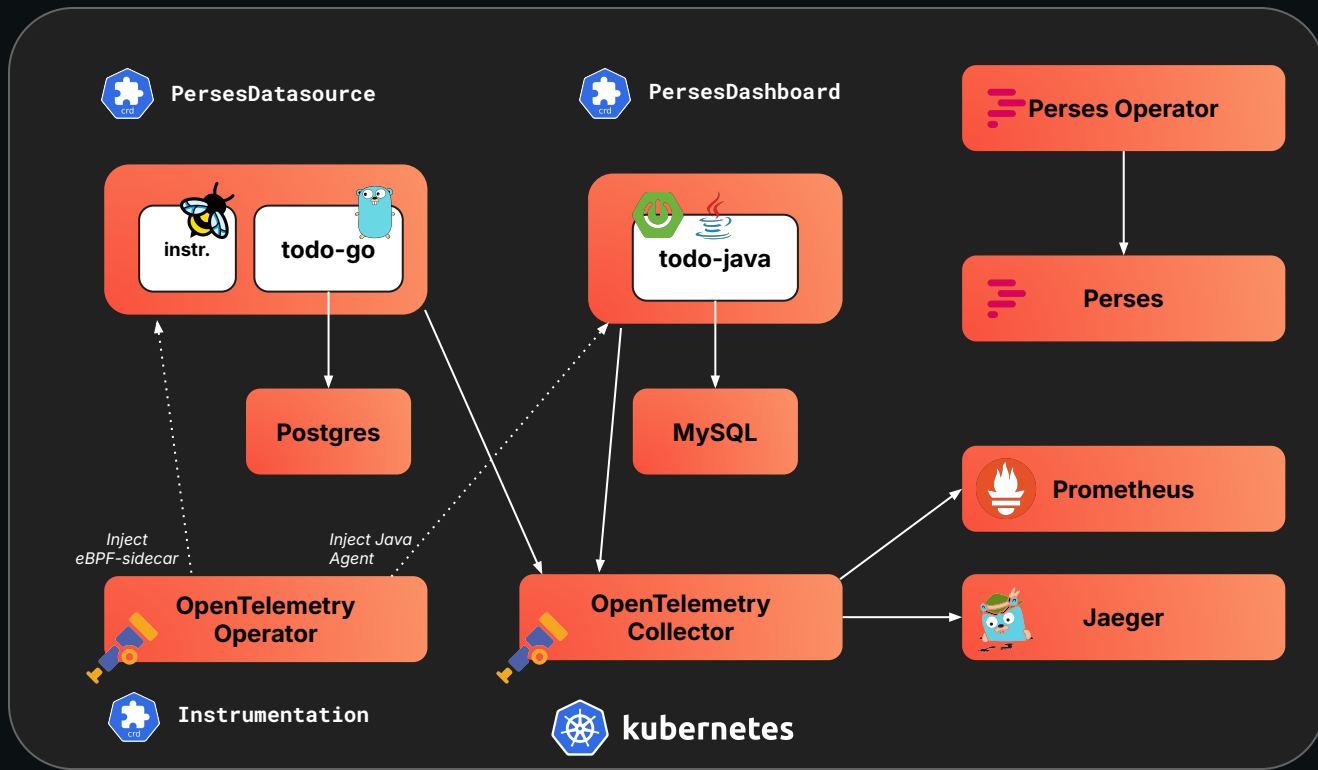
PersesDatasource



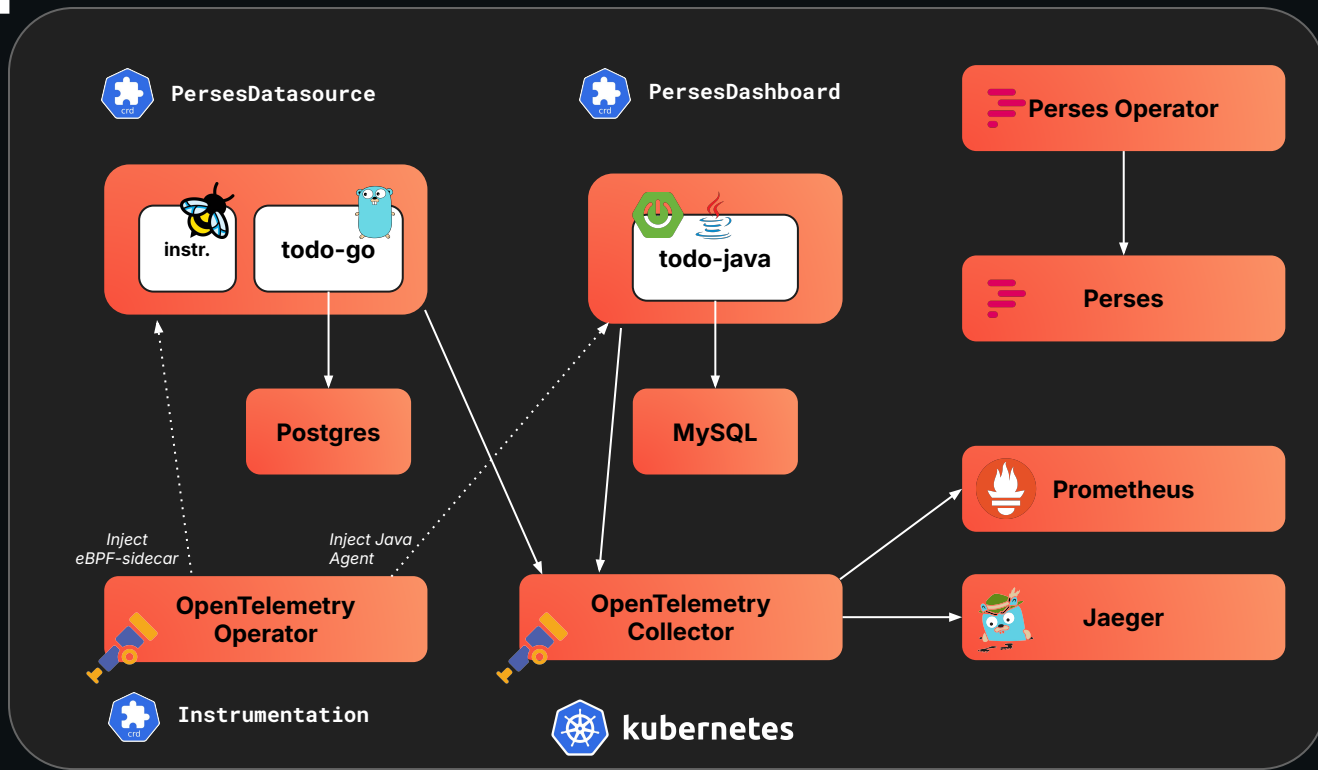
PersesDashboard

perses-operator

Demo



Recap



Observability is evolving - fast.

OpenTelemetry is **standardizing** telemetry collection.

Perses is standardizing dashboarding.

Applying **Platform Engineering** principles
can transform **observability** from an
afterthought into a seamless, scalable, and
developer-friendly experience.

Observability is a **systems problem**
- not a tracing, logging, or metrics problem.

When we connect signals together,
we empower developers to solve problems
faster.

And last but not least,
Building on Open Standards allows you to
freely move between vendors, ensuring they
stay on their toes and provide you the best
possible experience.

Shameless plug: OTelBin

Forever free, OSS

Editing, visualization and validation of OpenTelemetry Collector configurations

With ❤️ by Dash0!

<https://www.otelbin.io/>

The screenshot displays the OTelBin web application interface. The top bar shows the OTelBin logo and a validation status: "Validation: OpenTelemetry Collector Contrib - v0.94.0". A "Share" button is visible in the top right corner.

The main interface is divided into two panels. The left panel shows the configuration editor with a code editor displaying the following configuration:

```
processors > k8sattributes > extract > metadata > k8s.pod.name  
  
10 processors:  
11   k8sattributes:  
12     extract:  
13       metadata:  
14         - k8s.pod.name  
15         - k8s.pod.uid  
16         - k8s.deployment.name  
17         - k8s.namespace.name  
18         - k8s.node.name  
19         - k8s.pod.start_time  
20       transform/add_cluster_name:  
21         log_statements:  
22           - context: resource  
23           statements:  
24             - set(attributes["k8s.cluster.name"], "prod-eu-1")  
25  
26 service:  
27   pipelines:  
28     logs:  
29       receivers:  
30         - otlp  
31       processors:  
32         - k8sattributes  
33         - transform/add_cluster_name  
34       exporters:  
35         - otlphttp  
36     metrics:  
37       receivers:  
38         - otlp  
39       processors:  
40         - k8sattributes  
41         - transform/add_cluster_name  
42       exporters:  
43         - otlphttp  
44     traces:  
45       receivers:  
46         - otlp
```

The right panel shows three visualizations of the configuration, each representing a different data type: Logs, Metrics, and Traces. Each visualization shows a flow diagram with four components: otlp (receiving), k8sattributes (processing), transform (adding cluster name), and otlphttp (exporting). The flow is indicated by arrows connecting the components in sequence.

At the bottom of the interface, there are status indicators for "0 Errors" and "0 Warnings". The footer includes links to "Join CNCF Slack", "#OTelBin on CNCF Slack", "GitHub", and "Legal". The bottom right corner features the "dash0" logo and the text "Crafted with ❤️ by dash0".

Thank you!

Get in touch!

Kasper Borg Nissen, Developer Advocate at  dash0



Demo can be found here!
<https://github.com/dash0hq/container-days-hamburg-2025>