Where's the "Auto" in Auto-Instrumentation?

Josh Lee • Sept 14 2025 • Community Over Code

Josh Lee

Open Source Advocate
@ Altinity

Observability is our ability to understand a system from its outputs alone


FUEL TRANSFER IN FLIGHT PROHIBITED
PROPELLER FEATHERING BUTTONS
GENERATOR LOAD PANEL
HEATER CONTROL PANEL
PRESSURIZATION CONTROL PANEL

Observability is not any one signal…

Metrics

Aggregable

Is there a problem?

Traces

Request-Scoped

Where is the problem?

Logs

Verbose, time-stamped records

What is the problem?

A Typical Request Log

```
2024-07-01 09:35:34 GET /home 200 ...
```

Adding Duration

```
2024-07-01 09:35:34 231ms GET /home 200
```

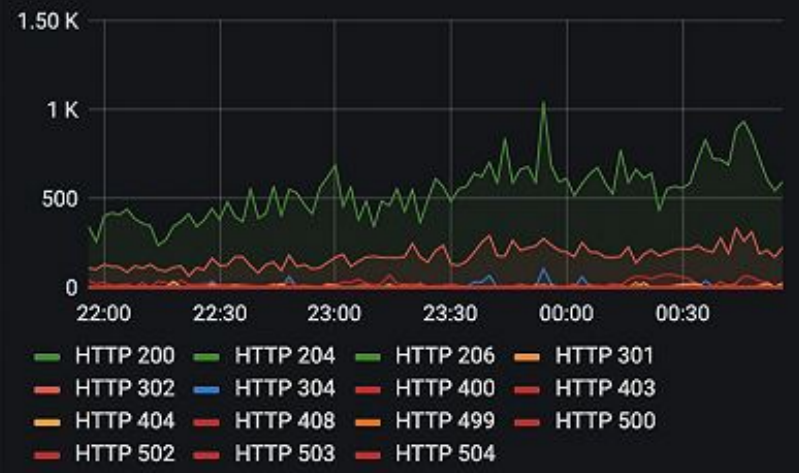
Controller Class: All | Namespace: live | Ingress: All | Ingress Pod: All | Config Reloads

Overview

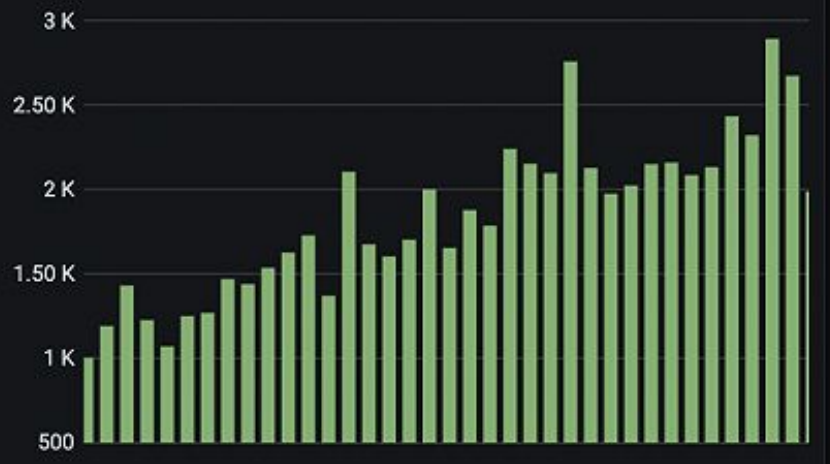
Requests (period)	% Succe...	Conns (...	Reqs (2...	% Success (2m)	HTTP 1/2xx (2m)	HTTP 3xx (2m)	HTTP 4xx (2m)	HTTP 5xx (2m)
67.3 K	97.0%	311	850	99.3%	592	228	30	0

HTTP Requests / Ingress

HTTP Status Codes



Total HTTP Requests

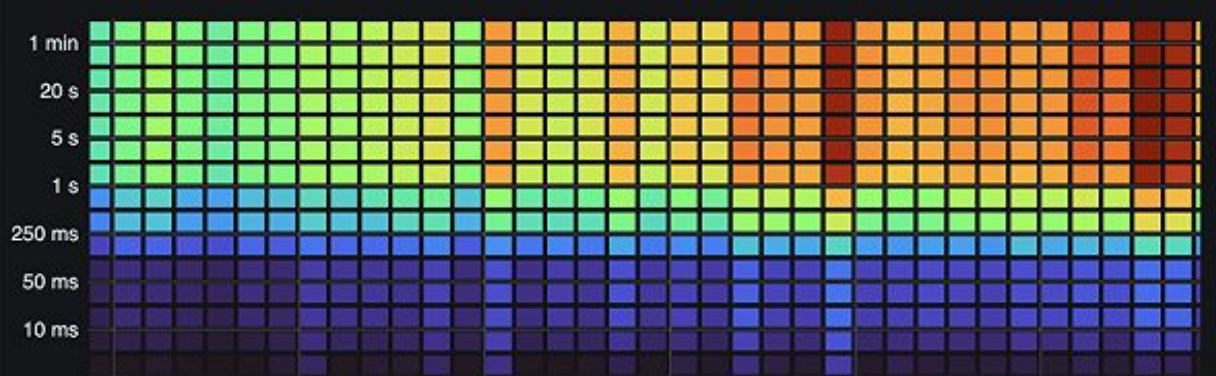


Latency

Latency (Average Percentiles)

	min	max	avg
0.99	4.3 s	42.3 s	6.5 s
0.95	2.4 s	4.5 s	2.8 s
0.9	2.1 s	2.7 s	2.3 s
0.5	357.2 ms	631.6 ms	443.7 ms
0.75	1.0 s	1.8 s	1.5 s
Average	3.2 s	5.1 s	4.1 s

Latency Heatmap



Back to our log…

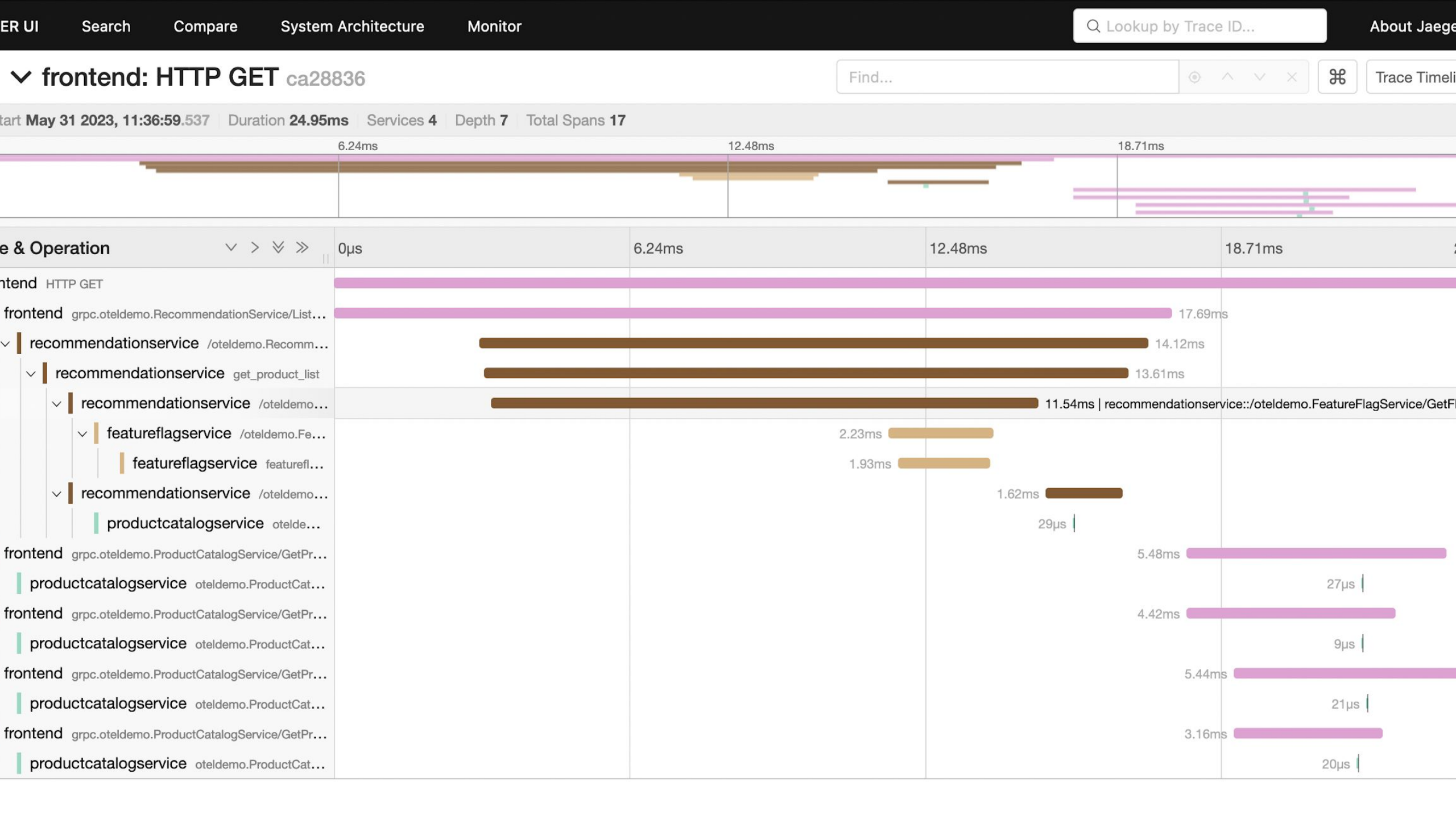
```
2024-07-01 09:35:34 231ms GET /home 200
```

Back to our log…

```
Request:123 2024-07-01 09:35:34 231ms GET /home
200
```

Connecting the trace:

```
Trace:4ea3 Span:123 2024-07-01 09:35:34 231ms GET
/home 200

Trace:4ea3 Span:456 ParentSpan:123 2024-07-01
09:35:34 201ms GET /api/users 201
```


ER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaege
frontend: HTTP GET ca28836
Find...
Trace Timeli
tart May 31 2023, 11:36:59.537
Duration 24.95ms
Services 4
Depth 7
Total Spans 17
6.24ms
12.48ms
18.71ms
e & Operation
0µs
6.24ms
12.48ms
18.71ms
ntend HTTP GET
frontend grpc.oteldemo.RecommendationService/List...
17.69ms
recommendationservice /oteldemo.Recomm...
14.12ms
recommendationservice get_product_list
13.61ms
recommendationservice /oteldemo...
11.54ms | recommendationservice::/oteldemo.FeatureFlagService/GetF
featureflagservice /oteldemo.Fe...
2.23ms
featureflagservice featurefl...
1.93ms
recommendationservice /oteldemo...
1.62ms
productcatalogservice otelde...
29µs
frontend grpc.oteldemo.ProductCatalogService/GetPr...
5.48ms
productcatalogservice oteldemo.ProductCat...
27µs
frontend grpc.oteldemo.ProductCatalogService/GetPr...
4.42ms
productcatalogservice oteldemo.ProductCat...
9µs
frontend grpc.oteldemo.ProductCatalogService/GetPr...
5.44ms
productcatalogservice oteldemo.ProductCat...
21µs
frontend grpc.oteldemo.ProductCatalogService/GetPr...
3.16ms
productcatalogservice oteldemo.ProductCat...
20µs

Observability is not any one signal…

Metrics

Aggregable

Is there a problem?

Traces

Request-Scoped

Where is the problem?

Logs

Verbose, time-stamped records

What is the problem?

Observability is not any one signal…

Metrics

Aggregable

Is there a problem?

Traces

Request-Scoped

Where is the problem?

Logs

Verbose, time-stamped records

What is the problem?

Distributed Tracing is the *"Killer App"*

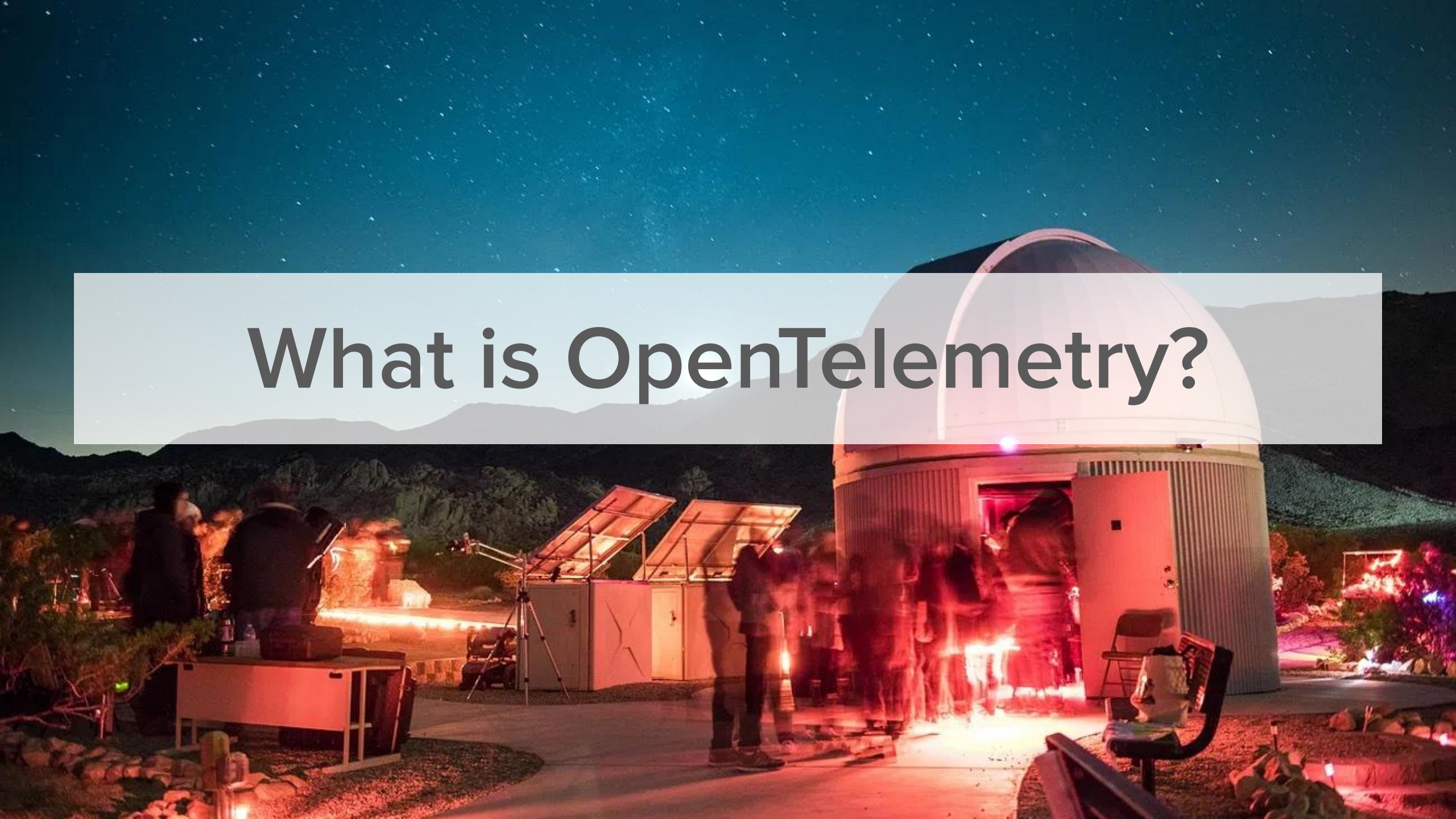
Understand complete request flows

Create a real-time map of system topology and dependencies

Derive metrics from the richness of trace metadata

Enrich logs and metrics with context

16



What is OpenTelemetry?

What is OpenTelemetry?

Specifications

- W3C Trace Context
- Language APIs
- OTLP
- Semantic Conventions

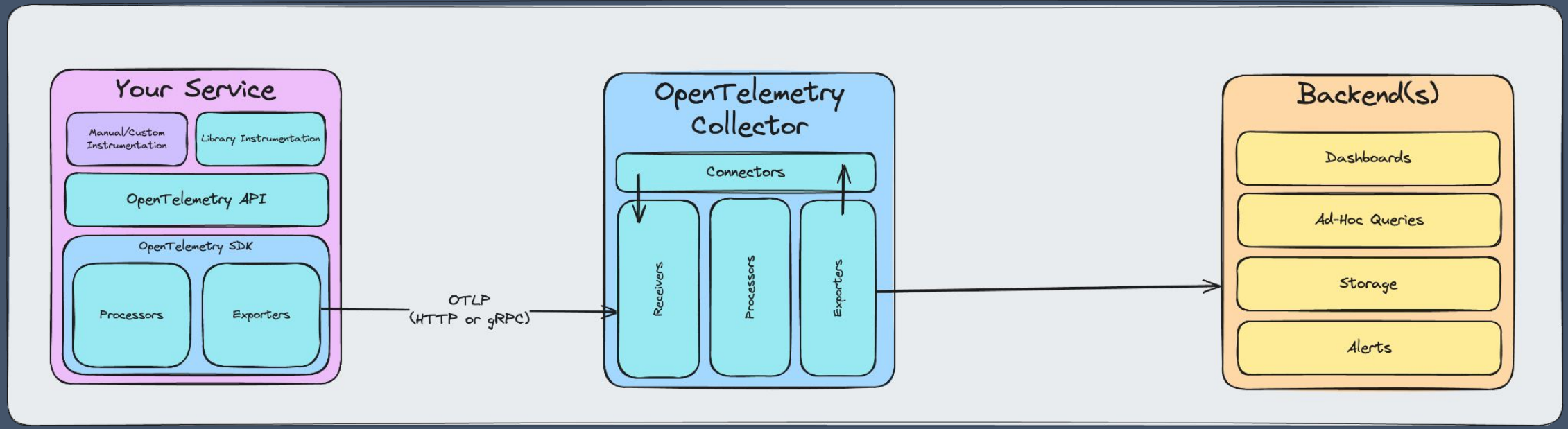
Libraries & Tools

- Language SDKs
- Instrumentation Libraries
- The Collector
- Kubernetes Operator

Community

- 2nd most-active CNCF project
- Events & Meetups
- OpenTelemetry End User WG (+ other WGs/SIGs)

What is OpenTelemetry?



Questions about Auto-Instrumentation

Do I need to modify my application code?

Do I need to use specific libraries?

Will it work without Kubernetes?

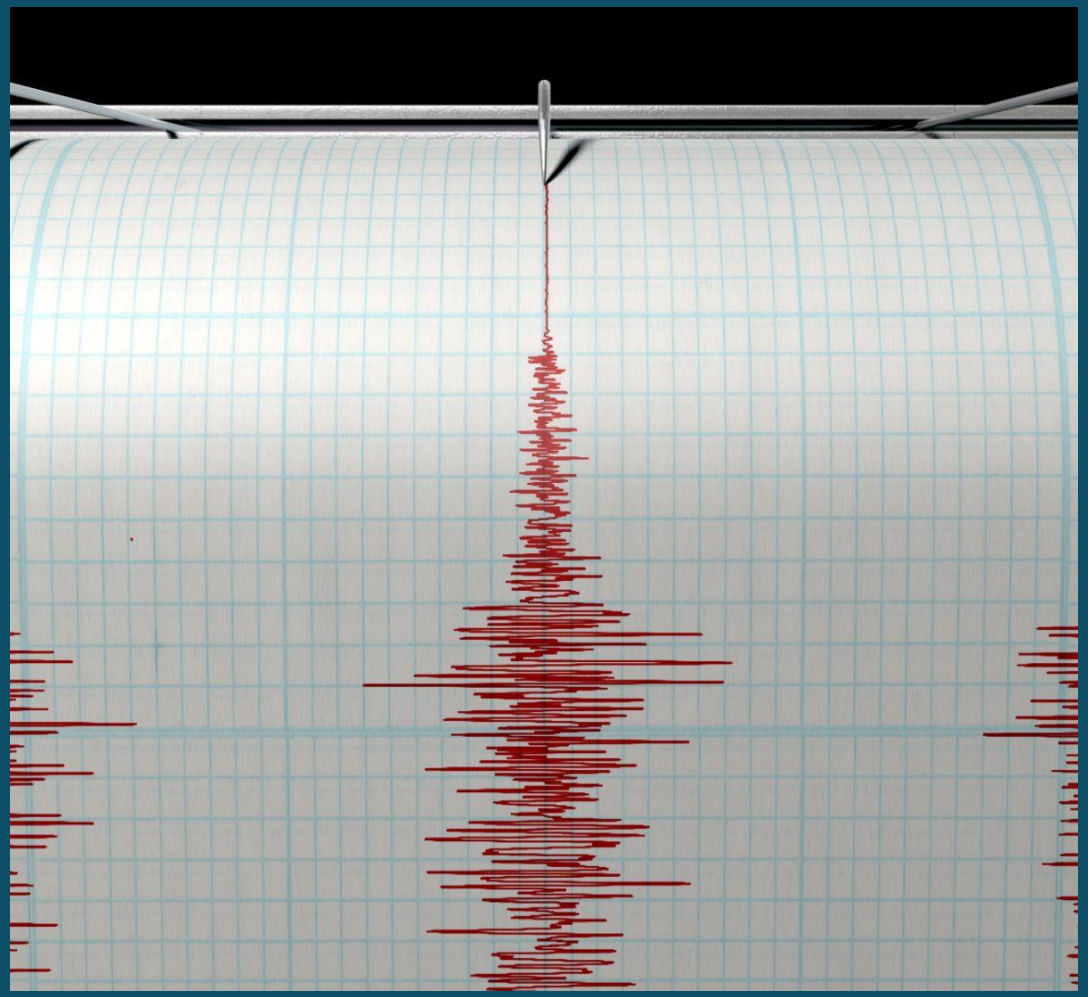
It depends…

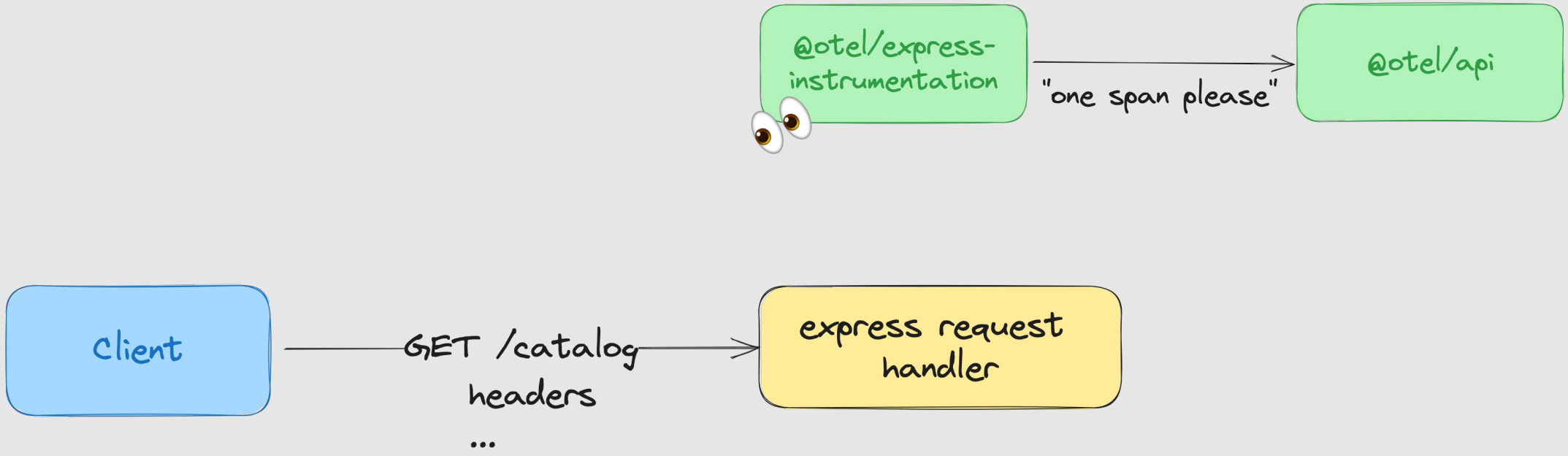
01 — What are we trying to do with instrumentation?
What is “auto-instrumentation”?
Hint: there is more than one right answer

02 — Different kinds of auto-instrumentation
The Kubernetes Operator

03 — When is Auto-Instrumentation not enough?
When is it *awesome?*

Instrumentation is the process of translating *interesting things* into telemetry signals




@otel/express-instrumentation
"one span please"
@otel/api
Client
GET /catalog
headers
...
express request handler

Interesting events

Requests,
Queries,
& Messages

Errors,
Exceptions,
& Events

Function Calls
(with arguments)

Contextualizing metadata

User Context

Who are they and what are they trying to do?

Infrastructure Context

What is the state of the resources?

Organizational Context

Who is responsible?

```
// Copyright The OpenTelemetry Authors
// SPDX-License-Identifier: Apache-2.0

const opentelemetry = require('@opentelemetry/sdk-node');
const {getNodeAutoInstrumentations} = require('@opentelemetry/auto-instrumentations-node');
const {OTLPTraceExporter} = require('@opentelemetry/exporter-trace-otlp-grpc');
const {OTLPMetricExporter} = require('@opentelemetry/exporter-metrics-otlp-grpc');
const {PeriodicExportingMetricReader} = require('@opentelemetry/sdk-metrics');
...

const sdk = new opentelemetry.NodeSDK({
  traceExporter: new OTLPTraceExporter(),
  instrumentations: [
    getNodeAutoInstrumentations(...)
  ],
  metricReader: new PeriodicExportingMetricReader({
    exporter: new OTLPMetricExporter(),
  }),
  resourceDetectors: [
    ...
  ],
});

sdk.start();
```

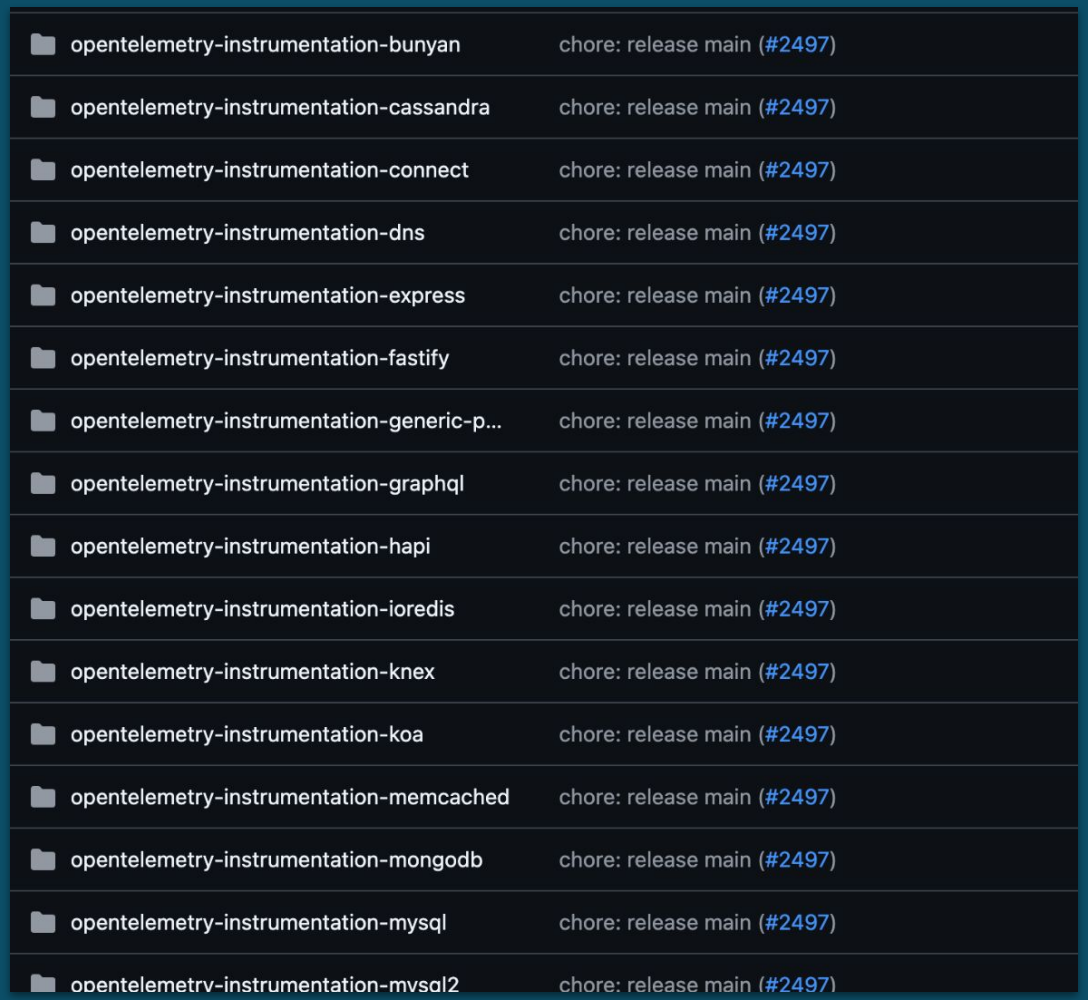
Instrumentation in OpenTelemetry

- OpenTelemetry SDK
- OpenTelemetry API
- Instrumentation Libraries

```
// Copyright The OpenTelemetry Authors
// SPDX-License-Identifier: Apache-2.0

const opentelemetry = require('@opentelemetry/sdk-node');
const {getNodeAutoInstrumentations} = require('@opentelemetry/auto-instrumentations-node');
const {OTLPTraceExporter} = require('@opentelemetry/exporter-trace-otlp-grpc');
const {OTLPMetricExporter} = require('@opentelemetry/exporter-metrics-otlp-grpc');
const {PeriodicExportingMetricReader} = require('@opentelemetry/sdk-metrics');
...

const sdk = new opentelemetry.NodeSDK({
  traceExporter: new OTLPTraceExporter(),
  instrumentations: [
    getNodeAutoInstrumentations(...)
  ],
  metricReader: new PeriodicExportingMetricReader({
    exporter: new OTLPMetricExporter(),
  }),
  resourceDetectors: [
    ...
  ],
});

sdk.start();
```

… Isn’t that Auto-Instrumentation?

Instrumentation Libraries *target specific libraries* and modify them (or observe them) to call the OTel APIs, creating telemetry



What is Auto-Instrumentation?

"Auto-Instrumentation" Meta Packages

Packages that can be configured to automatically include relevant instrumentation libraries based on the presence of other libraries.

No-Code Instrumentation Agents + Extensions

Mechanisms for adding instrumentation to an application package after it has already been compiled or bundled.

Instrumentation-Injection (w/ Kubernetes Operator)

The Kubernetes Operator can automatically inject no-code instrumentation into matching workloads.

Mechanisms

Code-based Library-Instrumentation

Observe other modules in the same process without directly modifying their source, and emit telemetry*

No-Code Instrumentation

Modify or observe another program (binary, bytecode, etc.) and emit telemetry*

Code-based Library-instrumentation
Monkey Patching
Function Wrapping
Middleware Injection
Event Observation

No-Code Instrumentation Mechanisms
Monkey Patching (JS, Python)
Runtime Agent (Java, .NET)
Interpreter Extension (PHP)
eBPF (Go)

	Instrumentation Libraries	No-Code Instrumentation	Mechanism
Java	✅	✅	Java Agent + Bytecode Injection
Python	✅	✅	Python Agent + Monkey patching
JavaScript	✅	✅	Monkey patching
.NET	✅	✅	.NET Profiler + Bytecode Injection
Go	✅	✅*	eBPF
PHP	✅	✅	Interpreter Extension + Autoloading
Erlang / Elixir	✅		
Ruby	✅		

Monkey Patching

```
new InstrumentationNodeModuleDefinition(
    'express',
    ['>=4.0.0 <5'],
    moduleExports => {
      const routerProto = moduleExports.Router as unknown as express.Router;
      // patch express.Router.route
      if (isWrapped(routerProto.route)) {
       this._unwrap(routerProto, 'route');
      }
      this._wrap(routerProto, 'route', this._getRoutePatch());
      ...
      return moduleExports;
    },
    moduleExports => {
      if (moduleExports === undefined) return;
      const routerProto = moduleExports.Router as unknown as express.Router;
      this._unwrap(routerProto, 'route');
      this._unwrap(routerProto, 'use');
      this._unwrap(moduleExports.application, 'use');
    }
  ),
];
```

Function Wrapping

```php
private static function _hook(..., ?string $class, string $function, string $name, int $spanKind = SpanKind::KIND_SERVER): void
{
    hook(
        class: $class,
        function: $function,
        pre: static function ($object, ?array $params, ?string $class, ?string $function, ?string $filename, ?int $lineno) use ($instrumentation, $name, $spanKind) {
            $span = self::builder($instrumentation, $name, $function, $class, $filename, $lineno)
                ->setSpanKind($spanKind)
                ->startSpan();
            Context::storage()->attach($span->storeInContext(Context::getCurrent()));
        },
        post: static function ($object, ?array $params, mixed $return, ?Throwable $exception) {
            self::end($exception);
        }
    );
}
```

Middleware Injection

```python
from fastapi import FastAPI
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import ConsoleSpanExporter, SimpleSpanProcessor

# Setup OpenTelemetry tracing provider
provider = TracerProvider()
provider.add_span_processor(SimpleSpanProcessor(ConsoleSpanExporter()))

# Create FastAPI app
app = FastAPI()

# Inject middleware explicitly
FastAPIInstrumentor.instrument_app(app, tracer_provider=provider)

@app.get("/")
async def read_root():
    return {"Hello": "Middleware injection!"}
```

Event Observation

```elixir
@doc false
def attach_router_start_handler(_opts) do
  :telemetry.attach(
    {__MODULE__, :router_dispatch_start},
    [:phoenix, :router_dispatch, :start],
    &__MODULE__.handle_router_dispatch_start/4,
    %{}
  )
end
```

No-Code Instrumentation Mechanisms

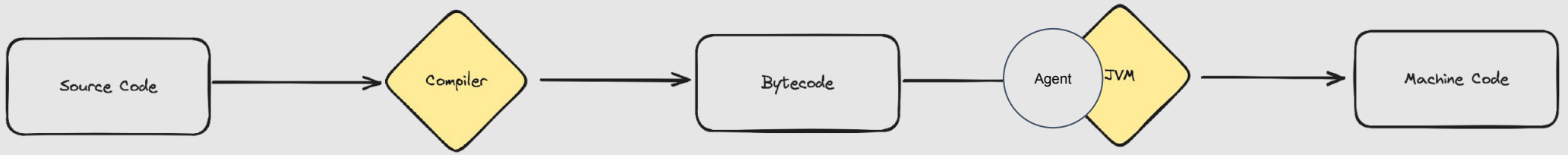
No-Code Instrumentation

```
export OTEL_SERVICE_NAME=my-service
export OTEL_TRACES_SAMPLER=parentbased_traceidratio
export OTEL_TRACES_SAMPLER_ARG=0.5
export OTEL_INSTRUMENTATION_HTTP_CLIENT_CAPTURE_HEADERS=true
```

Bytecode Injection

Runtime agents can inject OpenTelemetry SDKs and library instrumentation at the bytecode level.

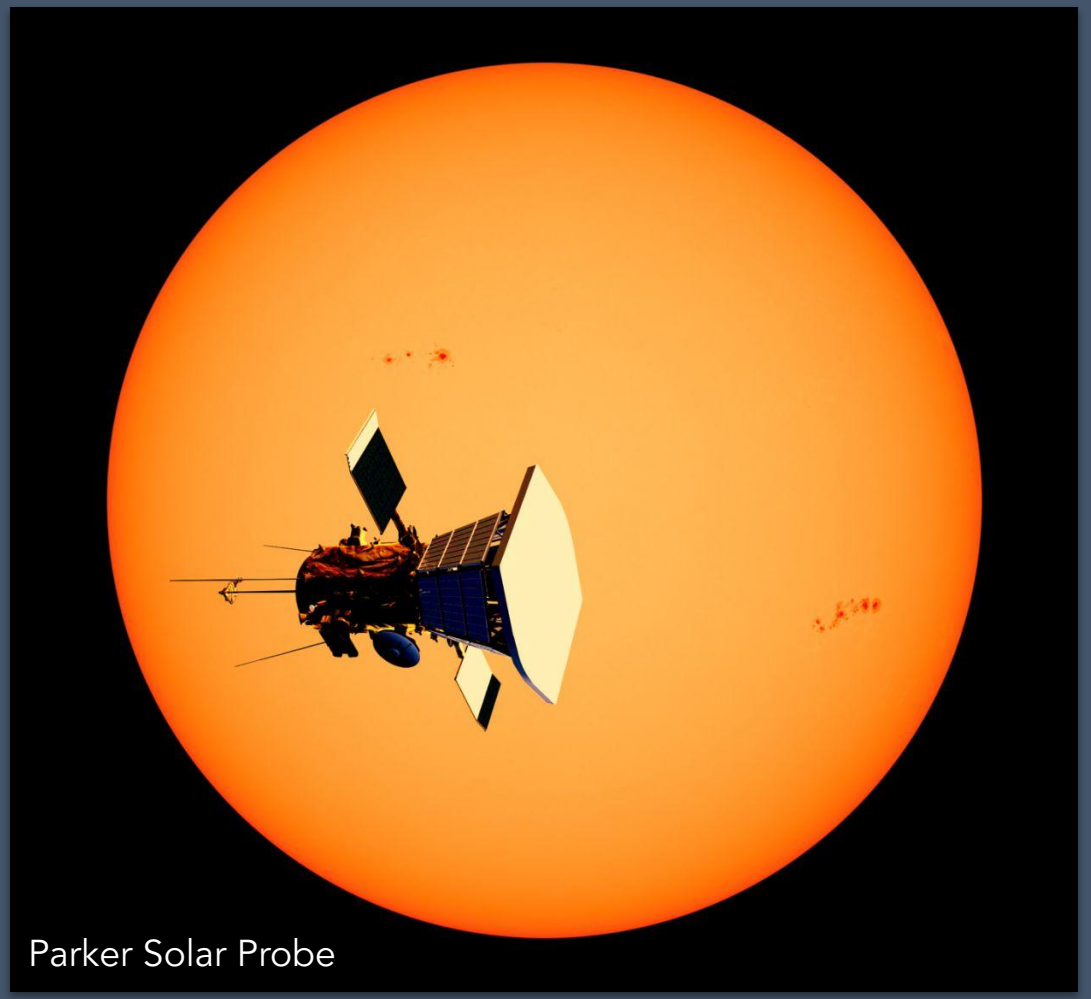
Functions can also be wrapped and/or monkey patched.


Source Code
Compiler
Bytecode
Agent
JVM
Machine Code

eBPF

Compiled binaries can’t be modified with instrumentation.

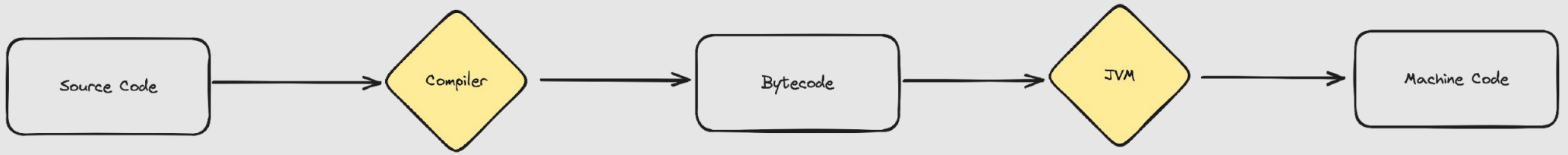
eBPF + uProbes offer visibility.



Parker Solar Probe

eBPF Example

```
//go:build ignore

#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __type(key, __u32);
    __type(value, __u64);
    __uint(max_entries, 1);
} pkt_count SEC(".maps");

// count_packets atomically increases a packet counter on every invocation.
SEC("xdp")
int count_packets() {
    __u32 key    = 0;
    __u64 *count = bpf_map_lookup_elem(&pkt_count, &key);
    if (count) {
        __sync_fetch_and_add(count, 1);
    }

    return XDP_PASS;
}

char __license[] SEC("license") = "Dual MIT/GPL";
```


Source Code
Compiler
Bytecode
JVM
Machine Code

Injecting Instrumentation

Kubernetes Operator

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: demo-instrumentation
spec:
  exporter:
    endpoint: http://demo-collector:4318
  propagators:
    - tracecontext
    - baggage
  sampler:
    type: parentbased_traceidratio
    argument: "1"
EOF
```

What have we learned?

Questions about Auto-Instrumentation

Do I need to modify my application code?

Do I need to use specific libraries?

Will it work without Kubernetes?

When is Auto-instrumentation *not enough*

Essential Custom Spans

"Stack" trace information

Incompatible frameworks or libraries

Especially for monolithic services

Essential Manual Annotation

User Experience & Client Details

Details about the user (who they are or what they are trying to do) from the Application or Client contexts.

Business Metrics & Dimensions

Runtime details about business operations (e.g. regions, departments, revenue, cost)

Team / Business Unit Ownership

Especially if it isn't defined by infrastructure context.
Extra useful in monolith-like services.

Incompatible or Bespoke System Architecture

Heavy use of non-HTTP messaging, IPC, RPC *
Compiled languages (besides Go)

When is Auto-Instrumentation *awesome*

When is Auto-Instrumentation *awesome*

As a starting-point / baseline

Reduce toil with a consistent starting point for customization

Fill gaps in E2E distributed tracing

Legacy services

Skill-gap on central observability team

Quickly understand system topology

Complete tracing can reveal architecture details

Cannot modify original source code

Auto-instrumentation is the only option here

Off-the-shelf OSS components can be deployed as-is

Built-in Instrumentation*

> Auto-Instrumentation

> Instrumentation Libraries

> API Calls Everywhere

(*) More on this in my next session at 2:10PM today…

Thank You!

🦋 @joshleecreates.bsky.social

🐘 @joshleecreates@hachyderm.io

🧳 linkedin.com/in/joshuamlee

💬 altinity.com/slack

Connect with me