Introduction to OpenTelemetry

Juraci Paixão Kröhling
Software Engineer

Olly GARDEN

Speaker

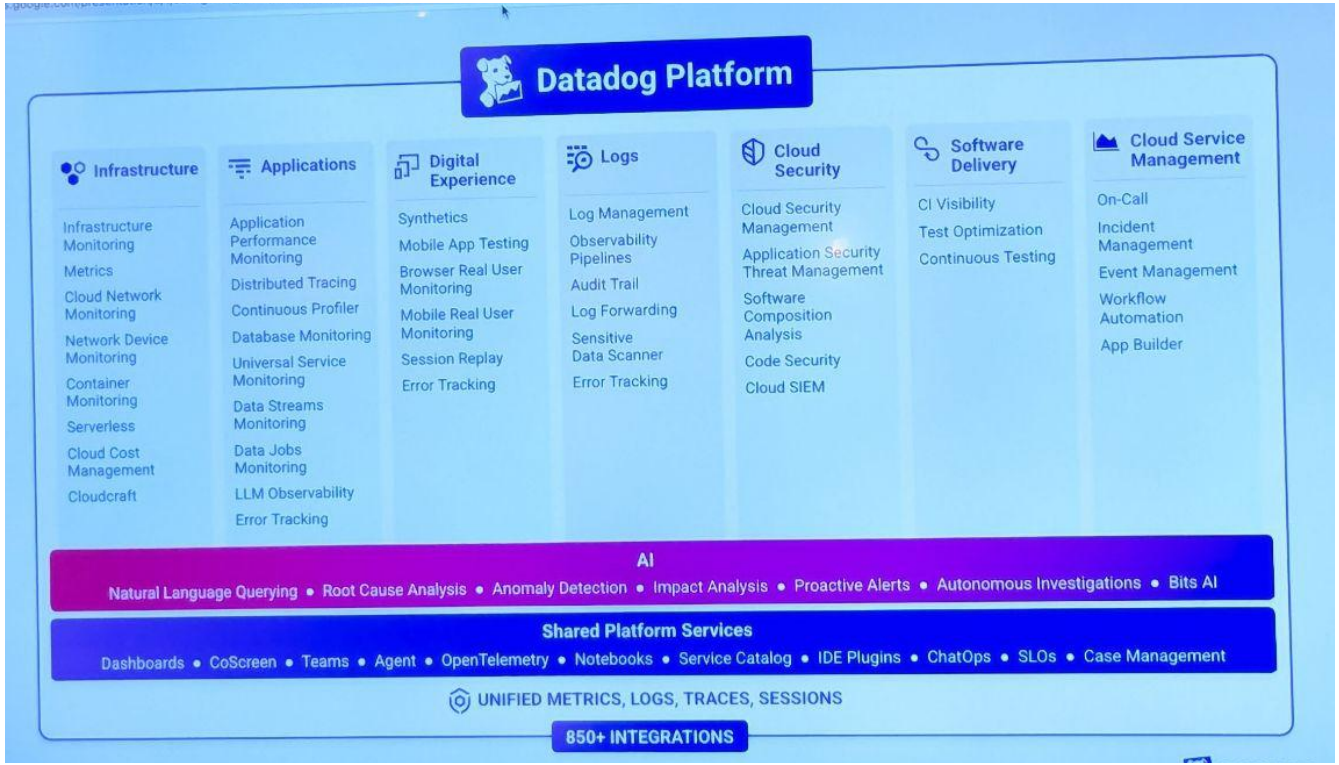
Juraci Paixão Kröhling
Software Engineer

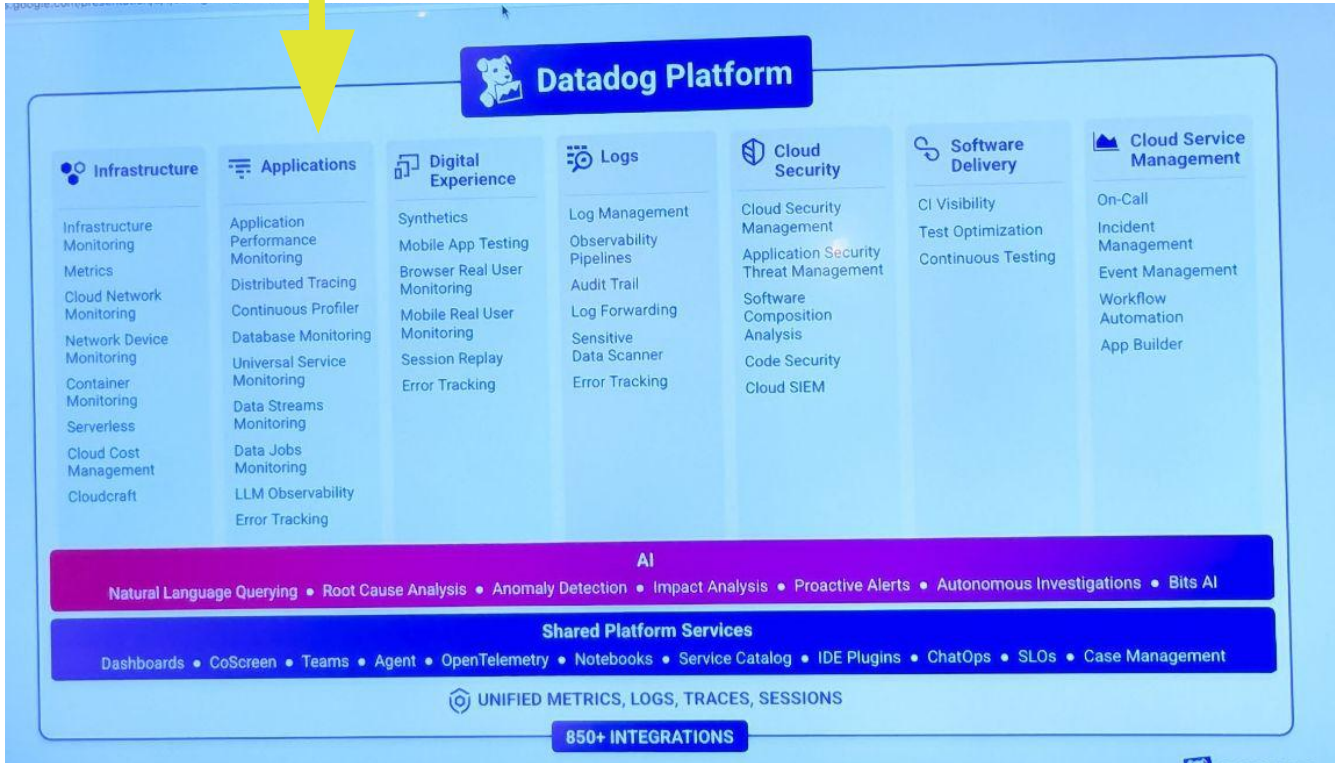
- Software Engineer at OllyGarden
- OTel Collector contributor
- Governance Committee - OpenTelemetry
- Cloud Native Computing Foundation (CNCF) Ambassador
- Organizer OTel Night Berlin

Agenda

- The problem we are solving
- What is OpenTelemetry (OTel)
- Show me the code!
- The ecosystem
- Q&A

The problem


Datadog Platform
Infrastructure
Infrastructure Monitoring
Metrics
Cloud Network Monitoring
Network Device Monitoring
Container Monitoring
Serverless
Cloud Cost Management
Cloudcraft
Applications
Application Performance Monitoring
Distributed Tracing
Continuous Profiler
Database Monitoring
Universal Service Monitoring
Data Streams Monitoring
Data Jobs Monitoring
LLM Observability
Error Tracking
Digital Experience
Synthetics
Mobile App Testing
Browser Real User Monitoring
Mobile Real User Monitoring
Session Replay
Error Tracking
Logs
Log Management
Observability Pipelines
Audit Trail
Log Forwarding
Sensitive Data Scanner
Error Tracking
Cloud Security
Cloud Security Management
Application Security Threat Management
Software Composition Analysis
Code Security
Cloud SIEM
Software Delivery
CI Visibility
Test Optimization
Continuous Testing
Cloud Service Management
On-Call
Incident Management
Event Management
Workflow Automation
App Builder
AI
Natural Language Querying • Root Cause Analysis • Anomaly Detection • Impact Analysis • Proactive Alerts • Autonomous Investigations • Bits AI
Shared Platform Services
Dashboards • CoScreen • Teams • Agent • OpenTelemetry • Notebooks • Service Catalog • IDE Plugins • ChatOps • SLOs • Case Management
UNIFIED METRICS, LOGS, TRACES, SESSIONS
850+ INTEGRATIONS


Datadog Platform
Infrastructure
Infrastructure Monitoring
Metrics
Cloud Network Monitoring
Network Device Monitoring
Container Monitoring
Serverless
Cloud Cost Management
Cloudcraft
Applications
Application Performance Monitoring
Distributed Tracing
Continuous Profiler
Database Monitoring
Universal Service Monitoring
Data Streams Monitoring
Data Jobs Monitoring
LLM Observability
Error Tracking
Digital Experience
Synthetics
Mobile App Testing
Browser Real User Monitoring
Mobile Real User Monitoring
Session Replay
Error Tracking
Logs
Log Management
Observability Pipelines
Audit Trail
Log Forwarding
Sensitive Data Scanner
Error Tracking
Cloud Security
Cloud Security Management
Application Security Threat Management
Software Composition Analysis
Code Security
Cloud SIEM
Software Delivery
CI Visibility
Test Optimization
Continuous Testing
Cloud Service Management
On-Call
Incident Management
Event Management
Workflow Automation
App Builder
AI
Natural Language Querying • Root Cause Analysis • Anomaly Detection • Impact Analysis • Proactive Alerts • Autonomous Investigations • Bits AI
Shared Platform Services
Dashboards • CoScreen • Teams • Agent • OpenTelemetry • Notebooks • Service Catalog • IDE Plugins • ChatOps • SLOs • Case Management
UNIFIED METRICS, LOGS, TRACES, SESSIONS
850+ INTEGRATIONS

Not so distant past

- Proprietary instrumentation API

Not so distant past

- Proprietary instrumentation API
- Proprietary instrumentation agents

Not so distant past

- Proprietary instrumentation API
- Proprietary instrumentation agents
- Proprietary wire protocols

Not so distant past

- Switching vendors was HARD and PAINFUL

How about we... commoditize instrumentation?

- Standardize instrumentation!
- Standardize collection!
- Standardize transmission!

Of course, we ended up with multiple solutions...

What is OpenTelemetry (OTel)

“

OpenTelemetry is a collection of APIs, SDKs, and tools. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software’s performance and behavior.

Source: https://opentelemetry.io/

APIs

- What software engineers use to instrument their software
- Each language has an API
- They all follow the API Specification
- “Contrib” instrumentation
    - we instrumented quite a few things, so you don’t have to!

SDKs

- Defines the behavior of the API
- Each language has an SDK
- They all follow the SDK Specification

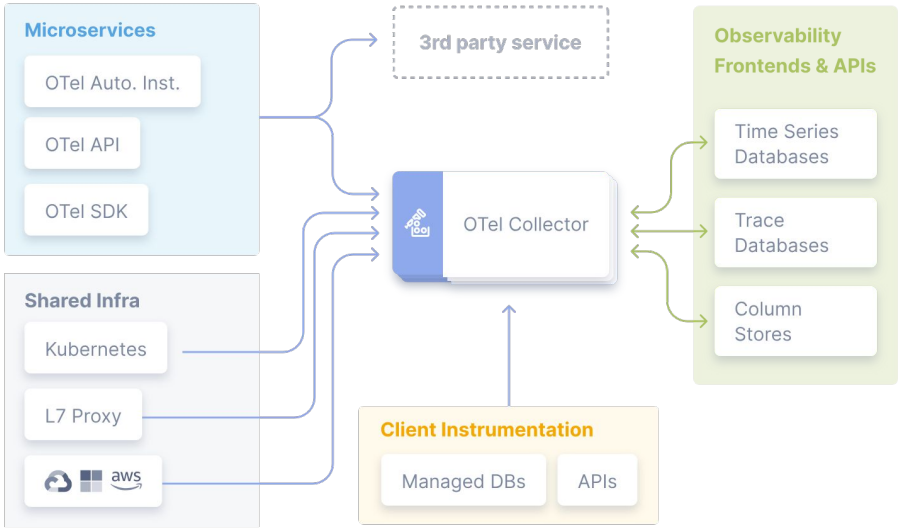
Specifications

- OTLP - OTel Protocol
- OpAMP - OTel Agent Management Protocol
- Semantic conventions
    - how you should instrument things

Tools

- OTel Collector
- OTel Operator
- Instrumentation tools
  - at runtime (Java Agent, eBPF, ...)
  - at compile time (code manipulation)

The big picture



Show me the code!

What we need to do

1. Initialize the SDK
2. Add instrumentation
3. ???
4. Profit!

Initializing the SDK - the config file

```yaml
file_format: "0.3"
disabled: false
resource:
  schema_url: https://opentelemetry.io/schemas/1.26.0
  attributes:
    - name: service.name
      value: "raincatcher"
    - name: service.version
      value: "0.0.1"
propagator:
  composite: [ tracecontext, baggage ]
tracer_provider:
  processors:
    - batch:
        exporter:
          otlp:
            protocol: http/protobuf
            endpoint: http://localhost:4318
```

Initializing the SDK - bootstrap

```
b, err := os.ReadFile(cfgFile)
if err != nil {
    return nil, err
}
// parse the config
conf, err := config.ParseYAML(b)
if err != nil {
    return nil, err
}
sdk, err := config.NewSDK(config.WithContext(ctx), config.WithOpenTelemetryConfiguration(*conf))
if err != nil {
    return nil, err
}
otel.SetTracerProvider(sdk.TracerProvider())
otel.SetMeterProvider(sdk.MeterProvider())
global.SetLoggerProvider(sdk.LoggerProvider())
```

Add instrumentation - New “span”

```
ctx, span := telemetry.Tracer().Start(ctx, "raincatcher.Run")
defer span.End()
```

Add instrumentation - Record errors

```
fileAuth, err := file.New(ctx, file.WithFile(app.cfg.Auth.TokensFile))
if err != nil {
    span.RecordError(err)
    span.SetStatus(codes.Error, err.Error())
    return fmt.Errorf("failed to configure auth: %w", err)
}
```

Add instrumentation - Record events

```
span.AddEvent("starting the NATS ingester")
err = app.ing.Start(ctx)
if err != nil {
    span.RecordError(err)
    span.SetStatus(codes.Error, err.Error())
    return fmt.Errorf("failed to start NATS ingester: %w", err)
}
```

Add instrumentation - instrumentation libraries

```
r.Handle("/", otelhttp.WithRouteTag("/", ingesterHandler))

authHandler := auth.Handler(ctx, fileAuth, r)
app.srv.Handler = otelhttp.NewHandler(authHandler, "server")
```

Add instrumentation - initialize a metric

```
if h.payloadSizeHistogram == nil {
    h.payloadSizeHistogram = internal.Must(telemetry.Meter().Float64Histogram(
        "raincatcher.ingester.handler.otlp.http.payload.size",
        metric.WithDescription("The size of the payload."),
        metric.WithUnit("By"),
    ))
}
```

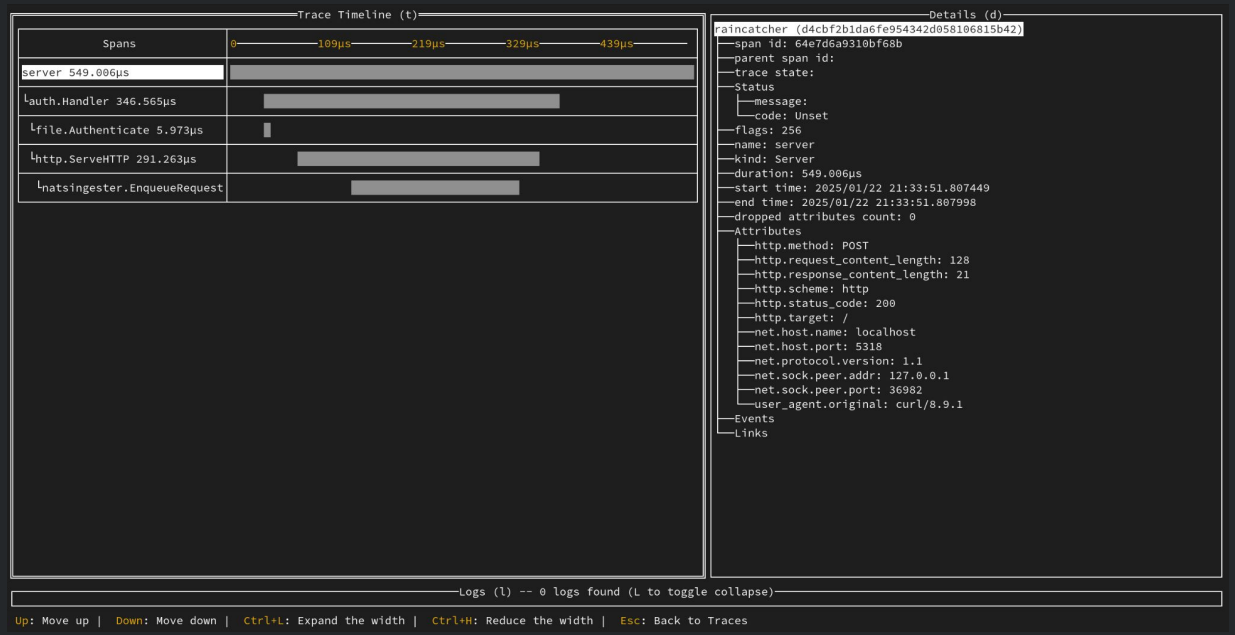
Add instrumentation - use a metric

```
h.payloadSizeHistogram.Record(ctx, float64(len(rm.Body)), metric.WithAttributes(
    telemetry.ContentType(rm.ContentType),
    telemetry.Tenant(rm.Tenant),
    telemetry.Organization(authData.Organization),
))
```

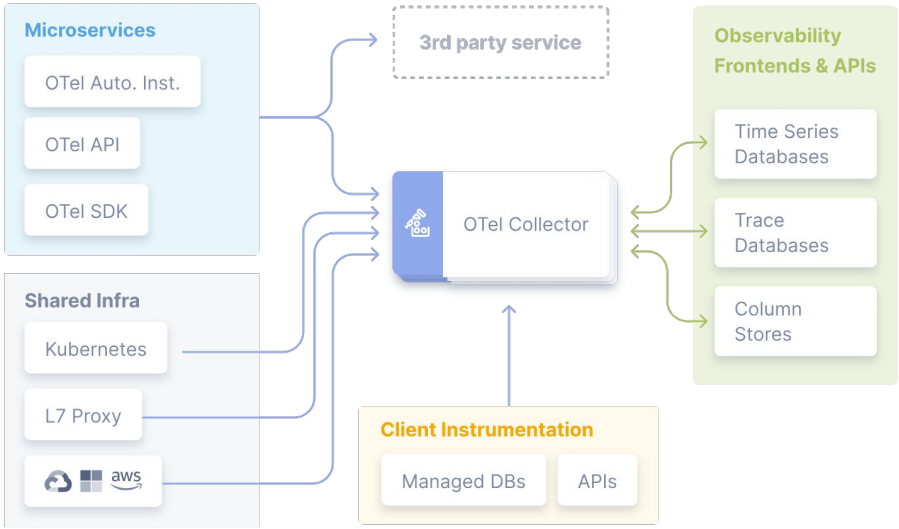
???

- Select your backend

Profit!


Trace Timeline (t)
Spans
0
109µs
219µs
329µs
439µs
server 549.006µs
└auth.Handler 346.565µs
└file.Authenticate 5.973µs
└http.ServeHTTP 291.263µs
└natsingester.EnqueueRequest
Details (d)
raincatcher (d4cbf2b1da6fe954342d058106815b42)
├─span id: 64e7d6a9310bf68b
├─parent span id:
├─trace state:
├─Status
│ ├─message:
│ └─code: Unset
├─flags: 256
├─name: server
├─kind: Server
├─duration: 549.006µs
├─start time: 2025/01/22 21:33:51.807449
├─end time: 2025/01/22 21:33:51.807998
├─dropped attributes count: 0
├─Attributes
│ ├─http.method: POST
│ ├─http.request_content_length: 128
│ ├─http.response_content_length: 21
│ ├─http.scheme: http
│ ├─http.status_code: 200
│ ├─http.target: /
│ ├─net.host.name: localhost
│ ├─net.host.port: 5318
│ ├─net.protocol.version: 1.1
│ ├─net.sock.peer.addr: 127.0.0.1
│ ├─net.sock.peer.port: 36982
│ └─user_agent.original: curl/8.9.1
├─Events
└─Links
Logs (l) -- 0 logs found (L to toggle collapse)
Up: Move up | Down: Move down | Ctrl+L: Expand the width | Ctrl+H: Reduce the width | Esc: Back to Traces

Where's the Collector?



“

The OpenTelemetry Collector offers a vendor-agnostic implementation of how to receive, process and export telemetry data.

Source: https://opentelemetry.io/docs/collector/

OTel Collector

The ecosystem

Ecosystem

We are organized in Special Interest Groups (SIGs)

- Spec SIGs (API, SDK, Semantic Conventions, OpAMP, GenAI, Mainframes, ...)
- Implementation SIGs (language SIGs, Collector, Operator, ...)
- Cross-cutting (Security, User Group, Contributor Experience, ...)

More info: https://github.com/open-telemetry/community/

Q&A

Obrigado!

Let’s connect:

linkedin.com/in/jpkroehling

Olly GARDEN