ABOUT ME

Vincent Behar
Senior Engineer

Twitter
https://twitter.com/vbehar

Ubisoft
French video game company

GitHub
https://github.com/vbehar
AGENDA

1. OpenTelemetry
2. Our context and goals
3. Implementation
4. Adoption
5. Benefits
High-quality, ubiquitous, and portable telemetry to enable effective observability

**Open source**

- 2019
- OpenTracing
- OpenCensus
- CNCF - #2 project

**Contributors**

- Amazon, Google, Microsoft, RedHat, …
- Splunk, Datadog, Grafana, Dynatrace, New Relic, Elastic, …
OPENTELEMETRY

What’s in it?

Specifications
- Traces: stable
- Metrics: stable
- Logs: experimental
- Semantic Conventions
- Propagation
- Protocol (OTLP)
- ...

Implementations
- APIs
- SDKs
- Libraries instrumentation
- 11 languages

Collector
- Interoperability
- Written in Go
- OpenCensus Service
- OTel’s killer feature
OPENTELEMETRY COLLECTOR

Vendor-agnostic way to receive, process and export telemetry data

Components

- 50+ receivers
- 40+ exporters
- 20+ processors
- 10+ extensions
- Custom components
- Distributions

[Diagram of the OPENTELEMETRY COLLECTOR showing components like OTLP, Jaeger, Prometheus, and their interactions with Receivers, Batch, Processors, and Exporters.]
OUR CONTEXT

Why we reconsidered our monitoring strategy?

Silos

- **Total isolation** between logs and metrics
- **Heterogenous** agents setup between services

Internal requirements

- Push **logs** to an Elasticsearch-based platform
- Push **metrics** to a VictoriaMetrics-based platform

Timing

- GA approaching...
- Integrating more services
Unified Visualization Platform

- Quickly surface relevant data
- With correlation between logs, metrics and traces
- Jump from graphs (metrics) to traces – with exemplars – to logs, ...

Unified collection and processing platform

- Common set of metadata and naming convention
- Simplify operations
OUR GOALS

Platform Features

Standards and conventions

- **Interoperability** between our applications
- Interoperability with 3rd party components
- Improve **troubleshooting and understanding** of our system

Extensible

- **Custom use-cases**
- Logs, metrics and traces today, **continuous profiling tomorrow**?
OUR GOALS

Integration in Ubisoft Ecosystem

Compliance with internal requirements

• Push our logs to an internal Elasticsearch-based service
• Set pre-defined labels on specific logs for security audit
• Push our metrics to an internal VictoriaMetrics-based service

Alignment with other teams

• Lots of teams / services at Ubisoft
• **Align** on the technology stack
• **Share** knowledge, experience
WHY OPENTELEMETRY

And what are we using?

Semantic conventions
- Spans attributes
- Application’s logs
- Collector pipelines

API/SDK for tracing
- Stable API/SDK available in multiple languages
- Auto-instrumentation
- Adoption by libraries
- OTLP

Collector
- Single agent
- Interoperability
- Routing
- Custom processors
- Custom distribution
- Not just an agent, but an extensible platform
IMPLEMENTATION

Deployment Strategy

- Kubernetes
- **DaemonSet**
- Per-node collection of logs and metrics
- Spans ingestion through a Service
IMPLEMENTATION

Input / Output

Log files

Open & Read

Scrape metrics

Receive spans

OpenTelemetry Collector

Logs

Elasticsearch

Loki

Traces

Tempo

Metrics

VictoriaMetrics

Prometheus

App Pod
IMPLEMENTATION

Logs Pipeline

Log files

filelog receiver

Logs Pipeline

- Move semantic fields from body to attributes
- Set Kubernetes Resource Attributes
- Set Cloud Resource Attributes
- Set Custom Attributes
- Fix 3rd party logs
...

Loki

Loki exporter

Elasticsearch

Elasticsearch exporter
IMPLEMENTATION

Metrics Pipeline

Pod

/metrics

Prometheus
receiver

Metrics Pipeline

Set Kubernetes Resource Attributes

Set Cloud Resource Attributes

... 

Routing (per-namespace)

VictoriaMetrics
Tenant 1

VictoriaMetrics
Tenant 2

Prometheus

Prometheus
Remote Write
Exporter

Prometheus
Remote Write
Exporter

Prometheus
Remote Write
Exporter
IMPLEMENTATION

Traces Pipeline
IMPLEMENTATION

Custom (logs) processors

```go
func (p *logProcessor) ProcessLogs(ctx context.Context, logs pdata.Logs) (pdata.Logs, error) {
    rLogs := logs.ResourceLogs()
    for i := 0; i < rLogs.Len(); i++ {
        rLog := rLogs.At(i)
        ills := rLog.InstrumentationLibraryLogs()
        for j := 0; j < ills.Len(); j++ {
            ls := ills.At(j).Logs()
            for k := 0; k < ls.Len(); k++ {
                record := ls.At(k)
                record.Body()
                record.Attributes()
                rLog.Resource().Attributes()
            }
        }
    }
    return logs, nil
}
```
IMPLEMENTATION

Monitoring the collector

- Per component instance metrics
Exemplars - in Grafana
ADOPTION

Changing people’s mindsets about monitoring

POC & demo

• Start with a single service, end to end
• **Showcase the result**: how correlation can help get a better understanding of the system
• Provide value to the users

Formalize

• **ADR**: Architecture Decision Records
• Explore different solutions
• Highlight benefits and shortcomings
• Write standards and conventions
BENEFITS

Of adopting OpenTelemetry

Reducing cognitive load
• Single stack
• Semantic convention
• Simpler to use and operate

Towards observability
• (almost) no more silos
• Auto generation of metrics from traces
• Easier troubleshooting and understanding of the platform

Owning the pipeline
• No lock-in
• Extensible platform
• Open source
• Active development
Various level of maturity depending on the components

- Logs data model is not stable yet – although in practice it should not change
- Prometheus metrics labels naming convention vs Otel semantic convention
- Prometheus Exemplars are not fully supported
WHAT’S NEXT

Our next steps

Tracing first

• Simplify instrumentation
• Generate metrics and logs from traces at the collector level

Continuous Profiling

• Parca – inspired by Prometheus
• Would be great to collect profiles from the OpenTelemetery Collector
• Backends: Parca, Pyroscope, …
CONCLUSION

1. BREAK THE SILOS
2. UNIFIED PLATFORM
3. EMBRACE THE COLLECTOR
4. ENJOY
THANK YOU!