Observability: buzz-word or real need?

https://github.com/montrealjug/jug-montreal-otel

Maxime DAVID & Olivier GATIMEL
Montreal JUG - Jan 2023

whoarewe >

Maxime David
Serverless @Datadog

AWS Community Builder
Youtuber (rust)

Olivier Gatimel
Lead dev @CARL Berger-Levrault

Agenda

1. Overview
2. Definitions, concepts and use cases
3. Demo
4. Going further
5. Questions!

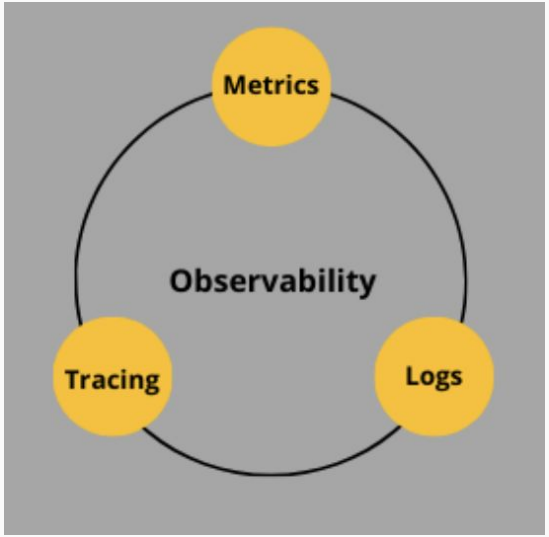
Observability != Monitoring

Observability refers to the ability to understand the internal state of a system by examining its outputs, such as **logs, metrics and traces**. It allows for the diagnosis of issues by providing insight into the system's behavior over time.

Monitoring refers to the continuous collection of data from a system to **check for any abnormal behavior or performance issues**

Without observability -> no monitoring!

Three pillars of observability



Source: https://iamondemand.com/blog/the-3-pillars-of-system-observability-logs-metrics-and-tracing/

- **Logs** provide a record of events that occur within a system
- **Metrics** provide measurable values that can be used to track the performance and health of a system.
- **Traces** provide a detailed record of the steps taken by a request or process as it flows through a distributed system, and can be used for debugging and performance analysis.

Logs - Example

Plain text (apache access log)

```
10.1.2.3 - rehg [20/Jan/2023:19:22:12 -0000] "GET /hiMontrealJug  HTTP/1.1" 200 3423
```

Logs - Example

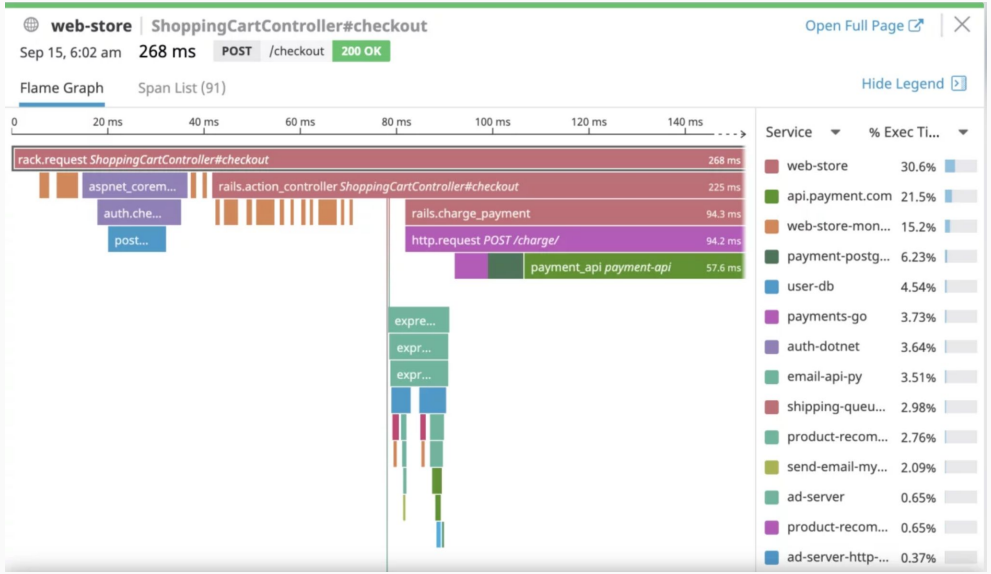
JSON (GCP example)


```
{
  "insertId": "42",
  "jsonPayload": {
    "message": "There was an error in the application",
    "times": "2019-10-12T07:20:50.52Z"
  },
  "httpRequest": {
    "requestMethod": "GET"
  },
  "resource": {
    "type": "k8s_container",
    "labels": {
      "container_name": "hello-app",
      "pod_name": "helloworld-gke-6cfd6f4599-9wff8",
      "project_id": "stackdriver-sandbox-92334288",
      "namespace_name": "default",
      "location": "us-west4",
      "cluster_name": "helloworld-gke"
    }
  },
  "timestamp": "2020-11-07T15:57:35.945508391Z",
  "severity": "ERROR",
  "labels": {
    "user_label_2": "value_2",
    "user_label_1": "value_1"
  },
  "logName": "projects/stackdriver-sandbox-92334288/logs/stdout",
  "operation": {
    "id": "get_data",
    "producer": "github.com/MyProject/MyApplication",
    "first": true
  },
  "sourceLocation": {
    "file": "get_data.py",
    "line": "142",
    "function": "getData"
  },
  "receiveTimestamp": "2020-11-07T15:57:42.411414059Z"
}
```

Metrics - Example

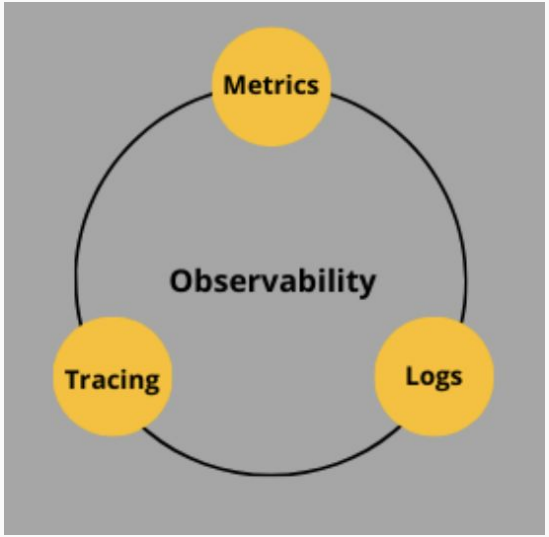
- System metrics
  - system.cpu.idle
  - redis.keys.evicted
- Business metrics
  - my.project.cart.item.added
  - My.project.password.forget.co
- OpenTelemetry metric data models
  - Sum (or counter), Gauge, Histogram
  - Exemplar : associates metric with a trace or a span

Traces - Example



Source: https://docs.datadoghq.com/fr/video-categories/flamegraph/

Three pillars of observability



Source: https://iamondemand.com/blog/the-3-pillars-of-system-observability-logs-metrics-and-tracing/

OpenTelemetry

OpenTelemetry is a collection of tools, APIs, and SDKs.

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/

NOT to store **NOR** visualize data

OpenTelemetry history

- Started in mid 2019 as a sandbox merge project of OpenTracing (traces) and OpenCensus (metrics)
- Supported by Cloud Native Computing Foundation (https://www.cncf.io/)
- Switched to incubating project in mid 2021
- Second most important project in CNCF (after Kubernetes)

Is it ready?

Current status : https://opentelemetry.io/status (moving fast)

- **Tracing**
  - Stable with LTS. Version 1.x out since Feb 2021
- **Metric**
  - API and protocol stable since Nov 2021
  - SDK still under active dev, but Meter is ready in java-sdk
- **Logs**
  - Draft/experimental
  - Current focus is integration with existing framework
  - An API will come later

Java is usually the language to get things implemented first

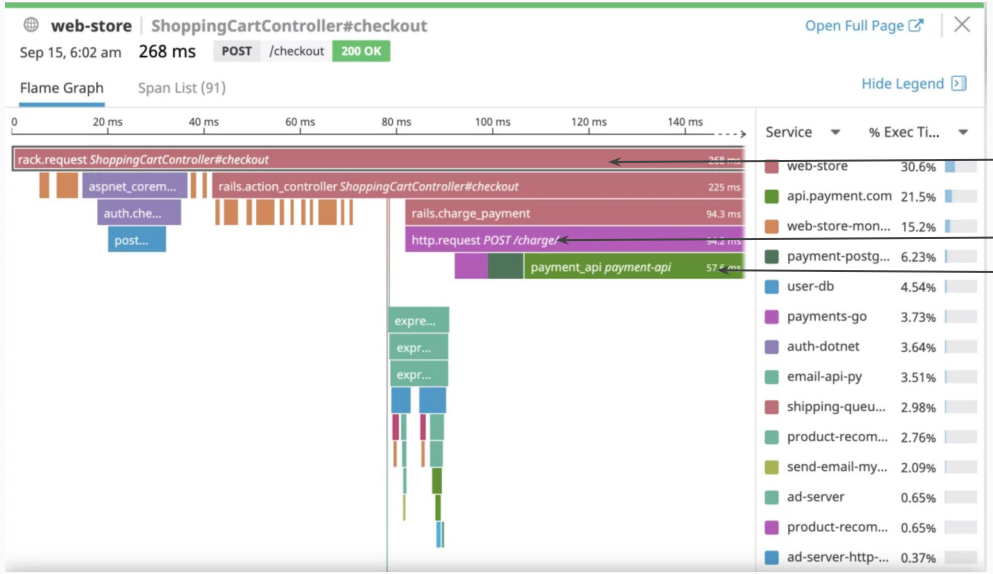
Focus on Tracing -> (more definitions 🙈)

Traces can be viewed as a **<u>directed acyclic graph</u>** of **Spans**

```
        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
[Span B]      [Span C] ←←←(Span C is a `child` of Span A)
     |             |
[Span D]      +---+-------+
              |           |
          [Span E]    [Span F]
```

Source: https://opentelemetry.io/docs/reference/specification/overview

Focus on Tracing



root span

(child) spans

Source: https://docs.datadoghq.com/fr/video-categories/flamegraph/

Spans

Span = operation within a transaction.

Span contains

- Parent's Span identifier (remember the DAG)
- An operation name
- A start and finish timestamp
- Attributes -> key-value pairs.
- A set of zero or more Events, each of which is itself a tuple (timestamp, name, attributes)

```
Span #2
    Trace ID       : 75e9a2a6eb8482613901c261d8cf6428
    Parent ID      : 98e14f3fbcdb9be2
    ID             : 1f19681c573efdf5
    Name           : HTTP POST
    Kind           : Client
    Start time     : 2023-01-23 20:05:40.243723707 +0000 UTC
    End time       : 2023-01-23 20:05:40.245862987 +0000 UTC
    Status code    : Error
Attributes:
     -> http.method: Str(POST)
     -> http.url: Str(http://localhost:8080/person/carolyn)
Events:
SpanEvent #0
     -> Name: exception
     -> Timestamp: 2023-01-23 20:05:40.245746748 +0000 UTC
     -> Attributes::
          -> exception.stacktrace: Str(java.net.ConnectException
```

Spans (going further)

- *Links to zero or more causally-related Spans (via the SpanContext of those related Spans).*
- *SpanContext information required to reference a Span*

Context propagation

By default in java-instrumentation,

W3C Trace Context (https://www.w3.org/TR/trace-context/)
-> published in november 2021

```
traceparent: 00-0af7651916cd43dd8448eb211c80319c-b7ad6b7169203331-01
```

version - traceId - parentId - traceFlags

Alternatives are possible (jaeger, xray, ...)

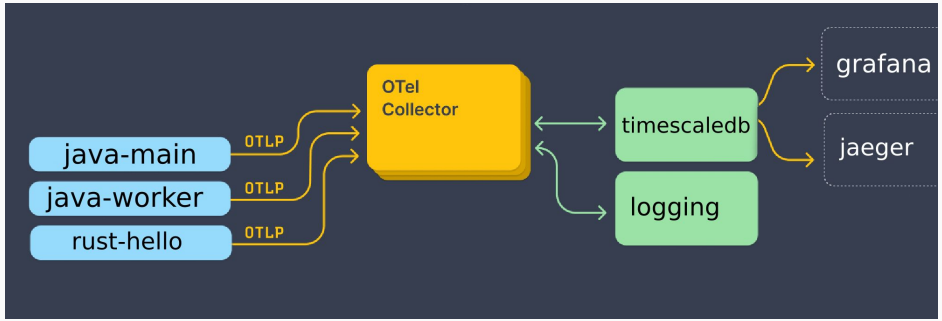
Demo deployment

Receivers:

- java-main: plain Java with sdk
- java-worker: SpringBoot with api + agent
- rust-hello : with sdk

Exporters:

- logging : otel collector stdout exporter
- timescaledb : postgres extension for time series storage
- grafana + jaeger : for visualization



DEMO 🔥

Going further

More than 3 pillars

- Profiling
- Code integration

Baggage

- A baggage is an attribute which is propagated between spans (e.g. article code)
- It is a public information, so caution with its visibility
- W3C Baggage (https://www.w3.org/TR/baggage/)
  -> working draft since september 2022

Visualization

- You likely need a tool to visualize

Links

Migration

- From OpenTracing
  https://github.com/open-telemetry/opentelemetry-java/blob/main/opentracing-shim
- From OpenCensus
  https://github.com/open-telemetry/opentelemetry-java/blob/main/opencensus-shim
- From micrometer with Java agent
  https://github.com/open-telemetry/opentelemetry-java-instrumentation/tree/main/instrumentation/micrometer/micrometer-1.5
- Send OTEL metrics with micrometer
  https://github.com/open-telemetry/opentelemetry-java-contrib/tree/main/micrometer-meter-provider
- Use Micrometer to ship metrics in OTLP format
  https://github.com/micrometer-metrics/micrometer/issues/2864

Links

Java agent alternative ?

Add java-sdk dependency and initialize AutoConfigure
https://github.com/open-telemetry/opentelemetry-java-docs/blob/main/autoconfigure/src/main/java/io/opentelemetry/example/autoconfigure/AutoConfigExample.java

Show trace+span id into logging with MDC

https://github.com/open-telemetry/opentelemetry-java-instrumentation/blob/main/docs/logger-mdc-instrumentation.md

Testing with traces

https://github.com/kubeshop/tracetest

QUESTIONS ?