OpenTelemetry Journey

2024.09

Jinwoong Kim

Who am I?

- Jinwoong Kim
- Cloud Architect @AWS Professional Services
- Speaker, Translator
- @ddiiwoong

Observability Introduction

A *system* is `observable` if you can determine the *behavior* of the system based on its *outputs*.

Observability

Logs - Lines of text

```
hikari-pool-1 - Connection is not available, request timed out after 30000ms
```

Metrics - Time-orderd set of data

```
hikaricp_connections_timeout_total{pool="HikariPool-1",} 10.0
```

Tracing - Correlation analysis with Context

```
2022-05-28 18:09:04.165  INFO [service-b,757d0493f099b94b,4e8d66a6aa1c1ed6] 9989 ---
[nio-8686-exec-3] c.example.msaerrorresponse.BServiceApi   : =======b-service======
```

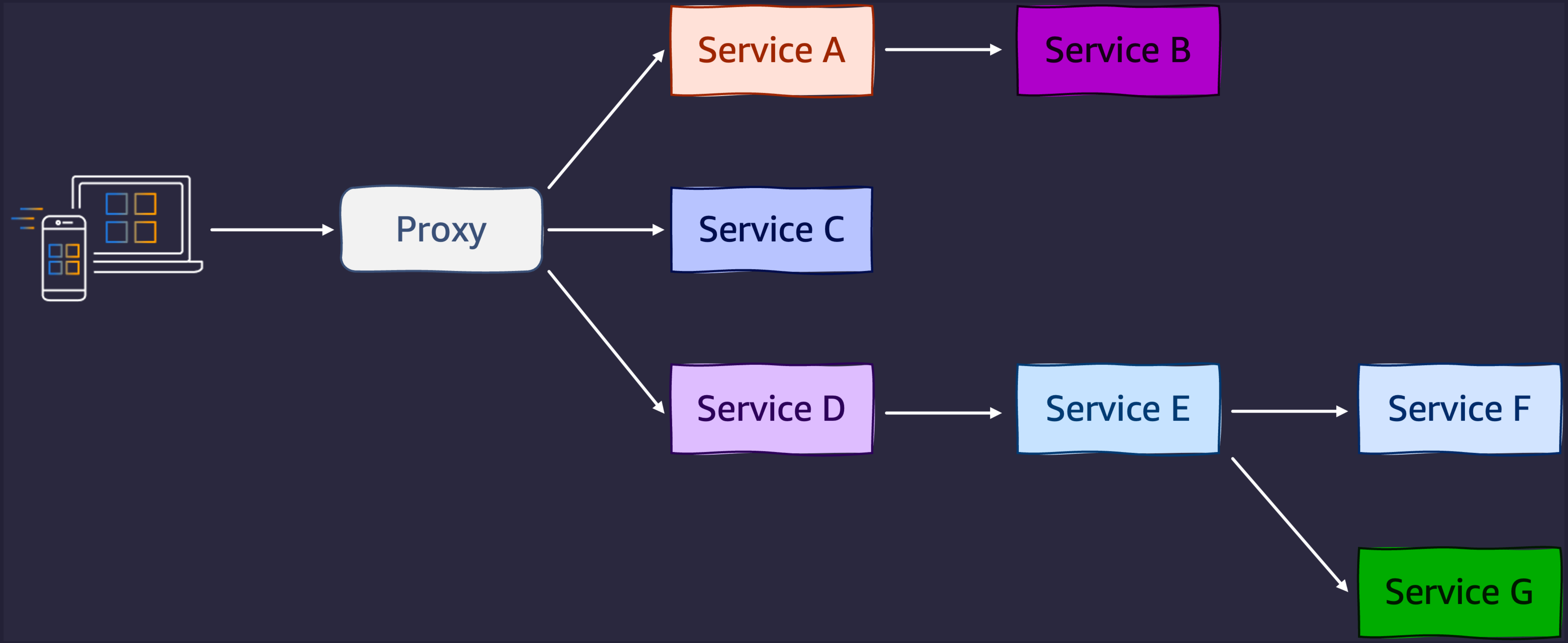
Distributed Trace

A `telemetry` method that indicates `latency` in specific parts of a system's process:

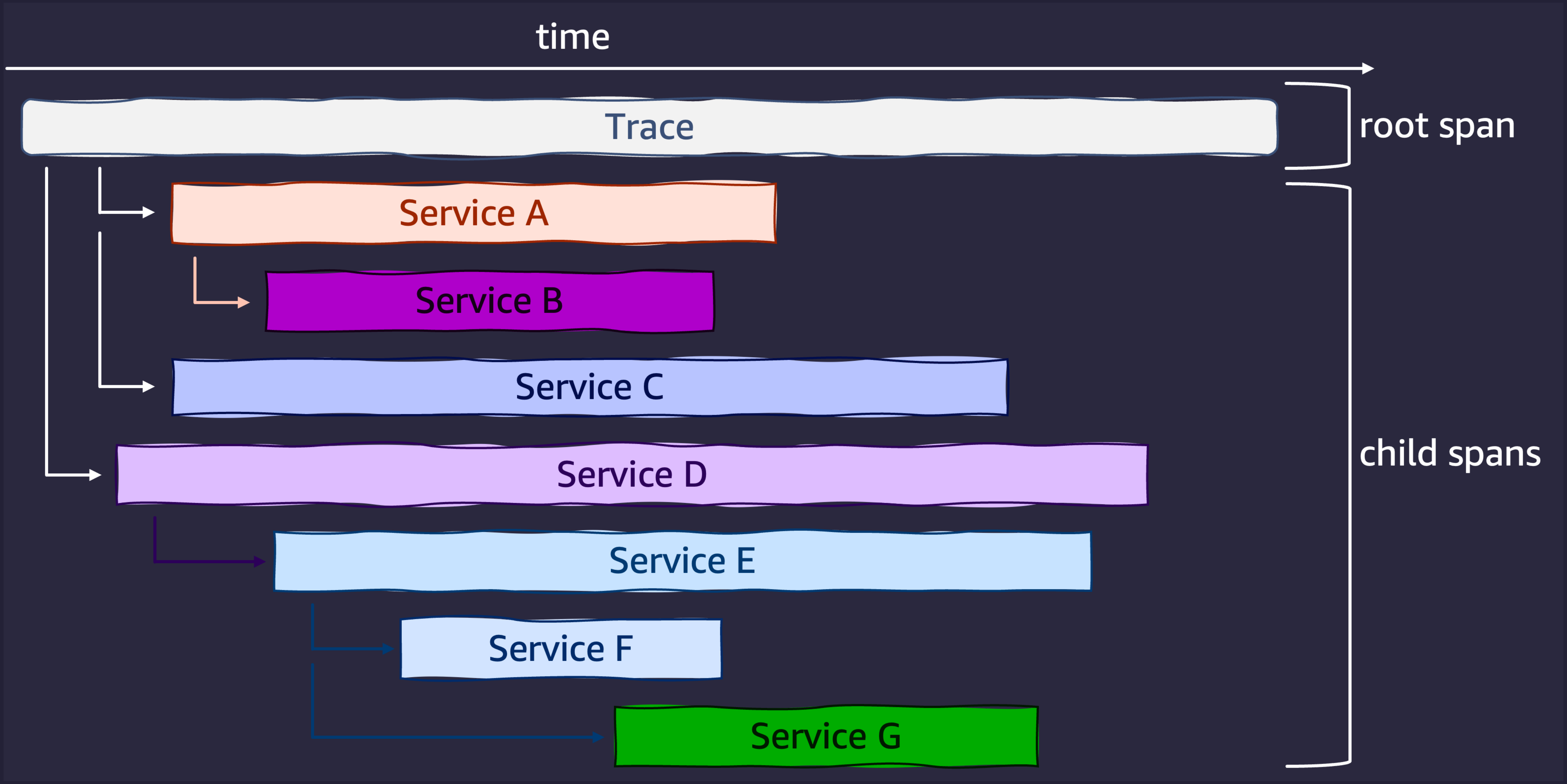
`Records` the path that requests take as they propagate through microservices and serverless architectures.

It is crucial for `observability` because it measures `dependencies` and `relationships` among numerous components in modern architectures like microservices environments, identifying `latency bottlenecks`.

Trace

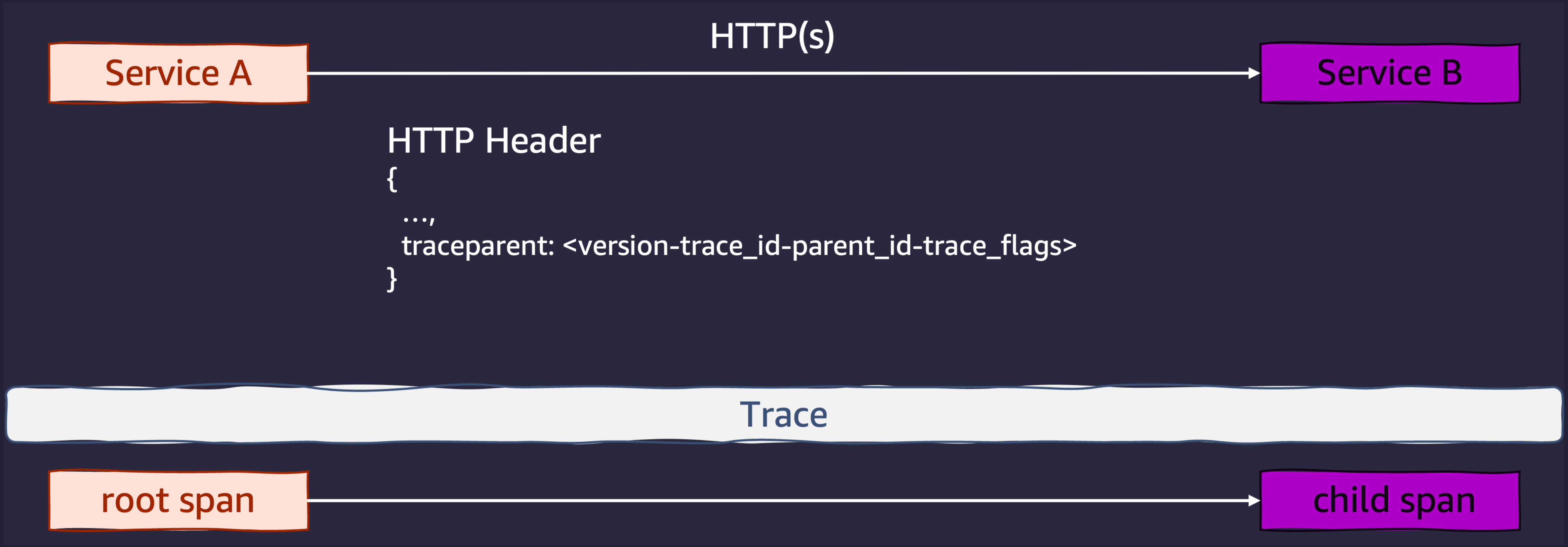

Proxy
Service A
Service B
Service C
Service D
Service E
Service F
Service G

Trace (Span)


time
Trace
root span
Service A
Service B
Service C
Service D
Service E
Service F
Service G
child spans

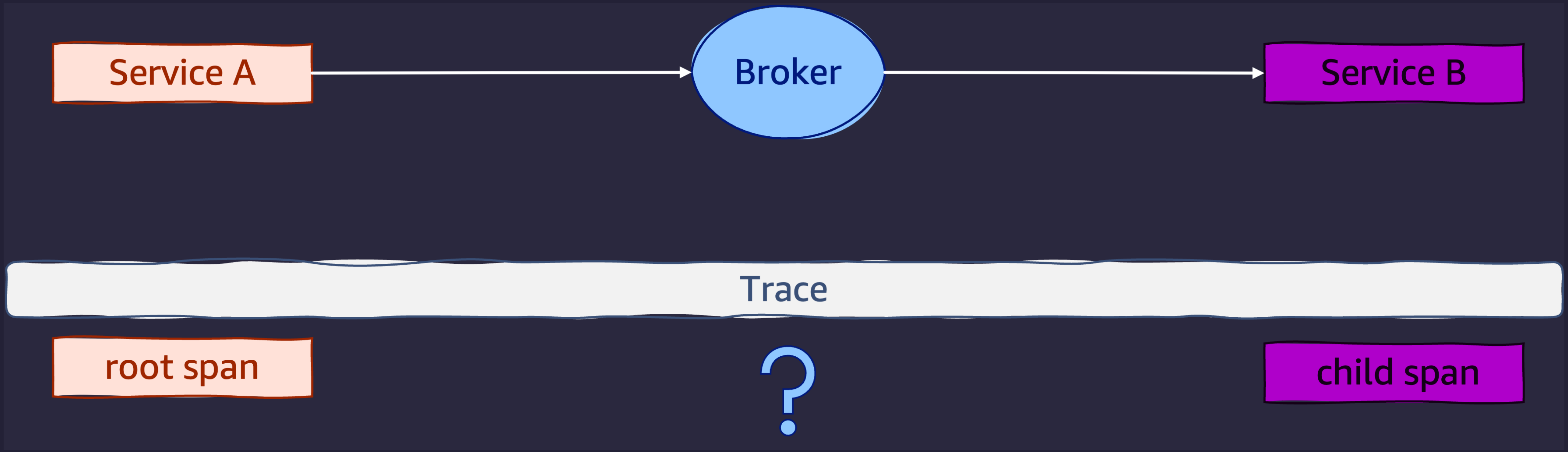
Trace - context propagation

W3C Trace Context :https://w3c.github.io/trace-context/

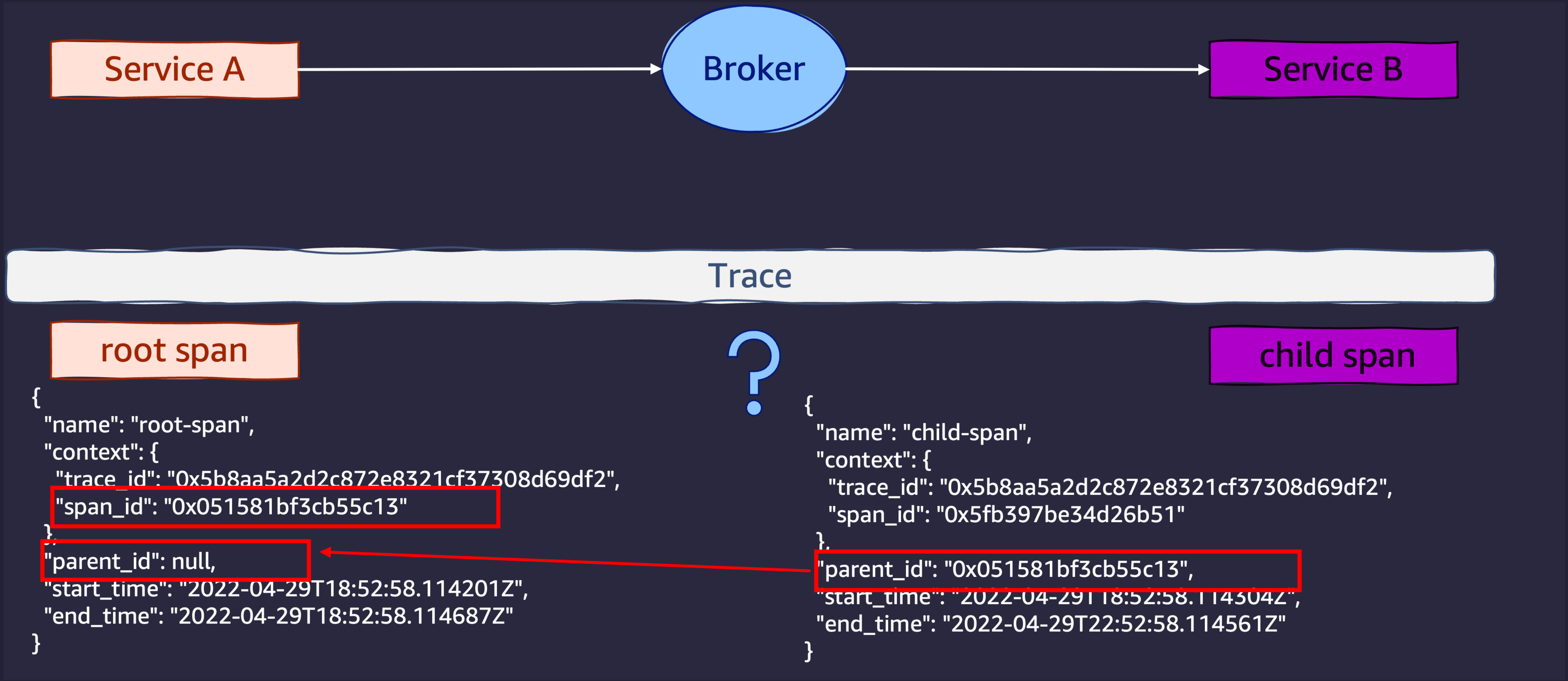


HTTP Header
```
{
  ...,
  traceparent: <version-trace_id-parent_id-trace_flags>
}
```

Trace - context propagation



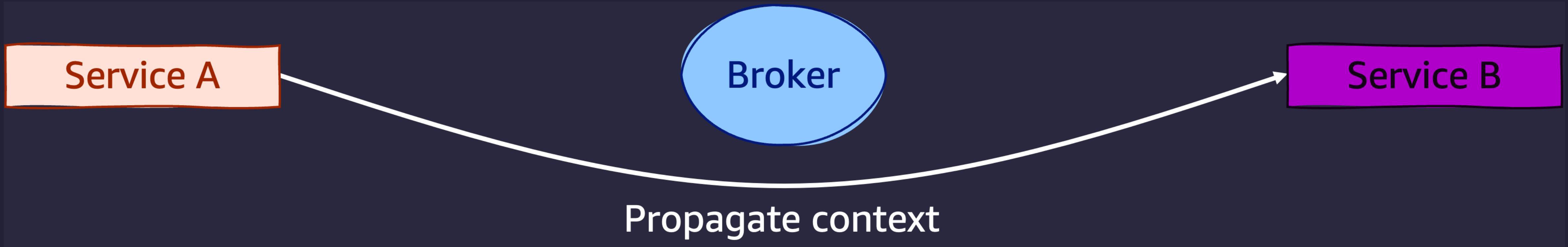
Trace - context propagation



```
{
  "name": "root-span",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x051581bf3cb55c13"
  },
  "parent_id": null,
  "start_time": "2022-04-29T18:52:58.114201Z",
  "end_time": "2022-04-29T18:52:58.114687Z"
}
```

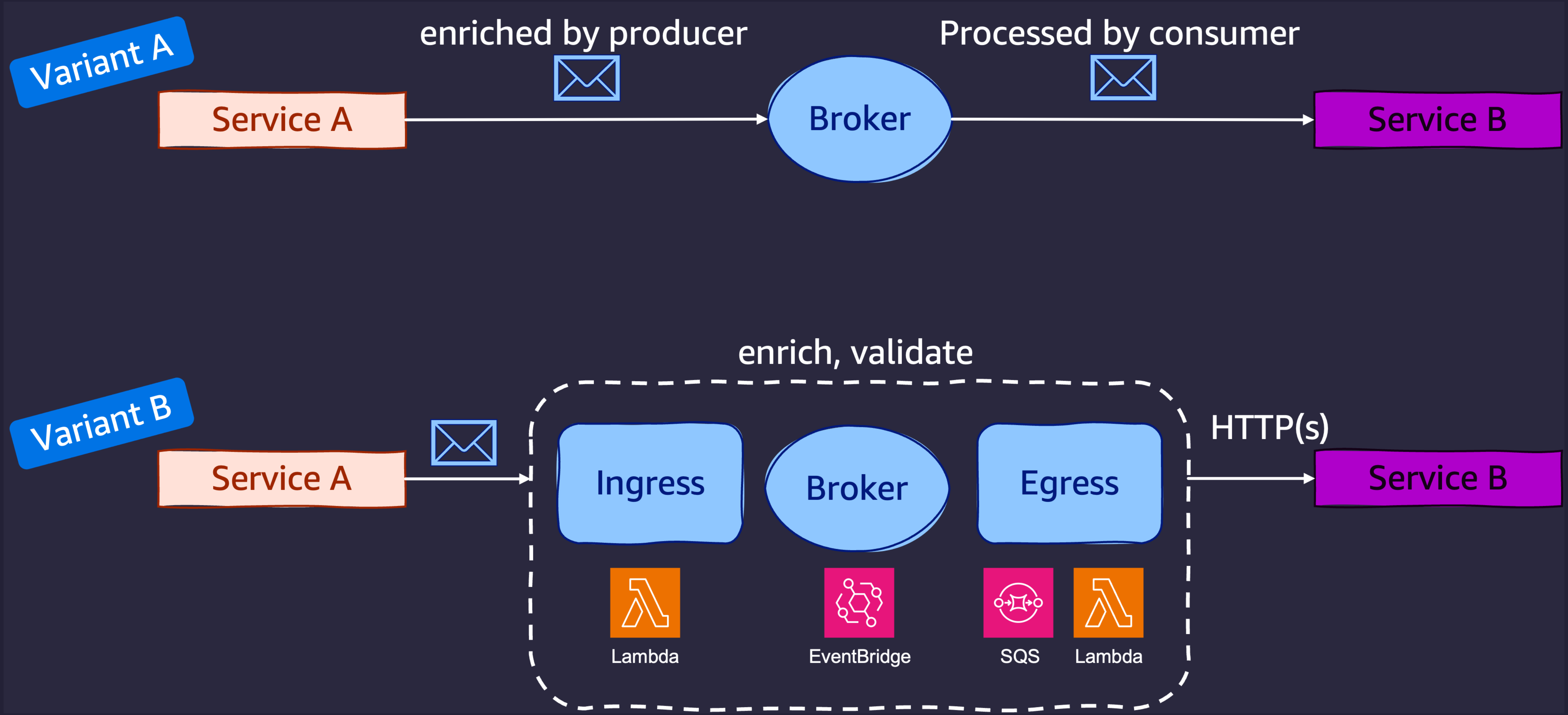
```
{
  "name": "child-span",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x5fb397be34d26b51"
  },
  "parent_id": "0x051581bf3cb55c13",
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z"
}
```

Trace - context propagation



```
{
 "metadata": {
   < some mandatory fields like timestamps, domain information, ...>
   "otel": {
     "trace_parent": {
       "trace_id": "0x5b8aa...",
       "span_id": "0x05158..."
     }
   }
 },
 "data": {
   < original producer event >
 }
}
```

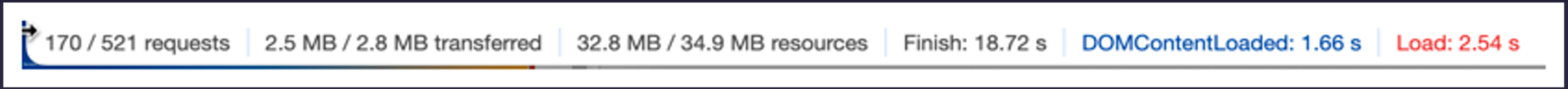
Trace - context propagation



Span

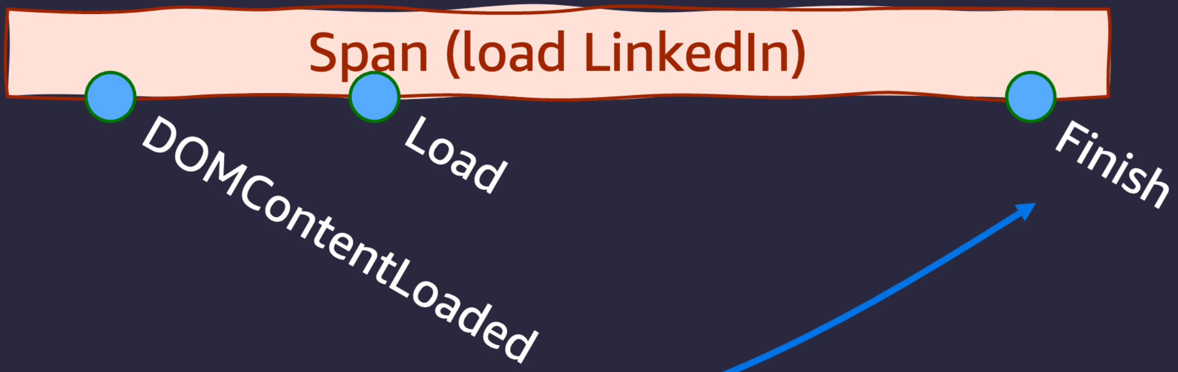
```
{
  "name": "/v1/sys/health",
  "context": {
    "trace_id": "7bba9f33312b3dbb8b2c2c62bb7abe2d",
    "span_id": "086e83747d0e381e"
  },
  "parent_id": "",
  "start_time": "2021-10-22 16:04:01.209458162 +0000 UTC",
  "end_time": "2021-10-22 16:04:01.209514132 +0000 UTC",
  "status_code": "STATUS_CODE_OK",
  "status_message": "",
  "attributes": {
    "http.scheme": "http",
    "http.host": "10.177.2.152:26040",
  },
  "events": [
    {
      "name": "",
      "message": "OK",
      "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
    }
  ]
}
```

Span



170 / 521 requests | 2.5 MB / 2.8 MB transferred | 32.8 MB / 34.9 MB resources | Finish: 18.72 s | DOMContentLoaded: 1.66 s | Load: 2.54 s

```
{
  "attributes": {
    "response_size": "2.5",
    "content_size": "2.8",
  },
  "events": [
    {
      "name": "Finish",
      "message": "OK",
      "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
    }
  ]
}
```

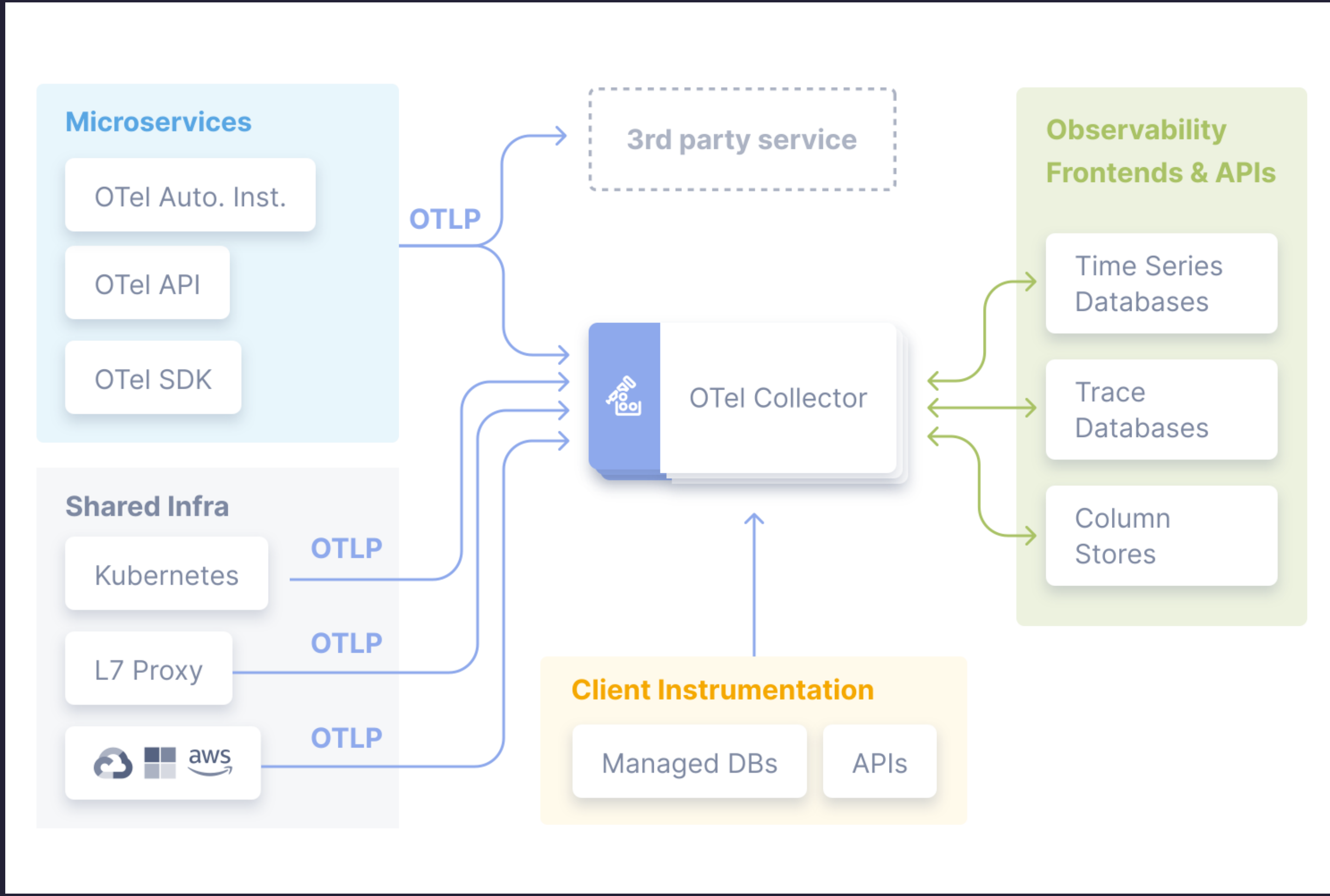


https://opentelemetry.io/docs/specs/otel/trace/exceptions/

Hello, OpenTelemetry

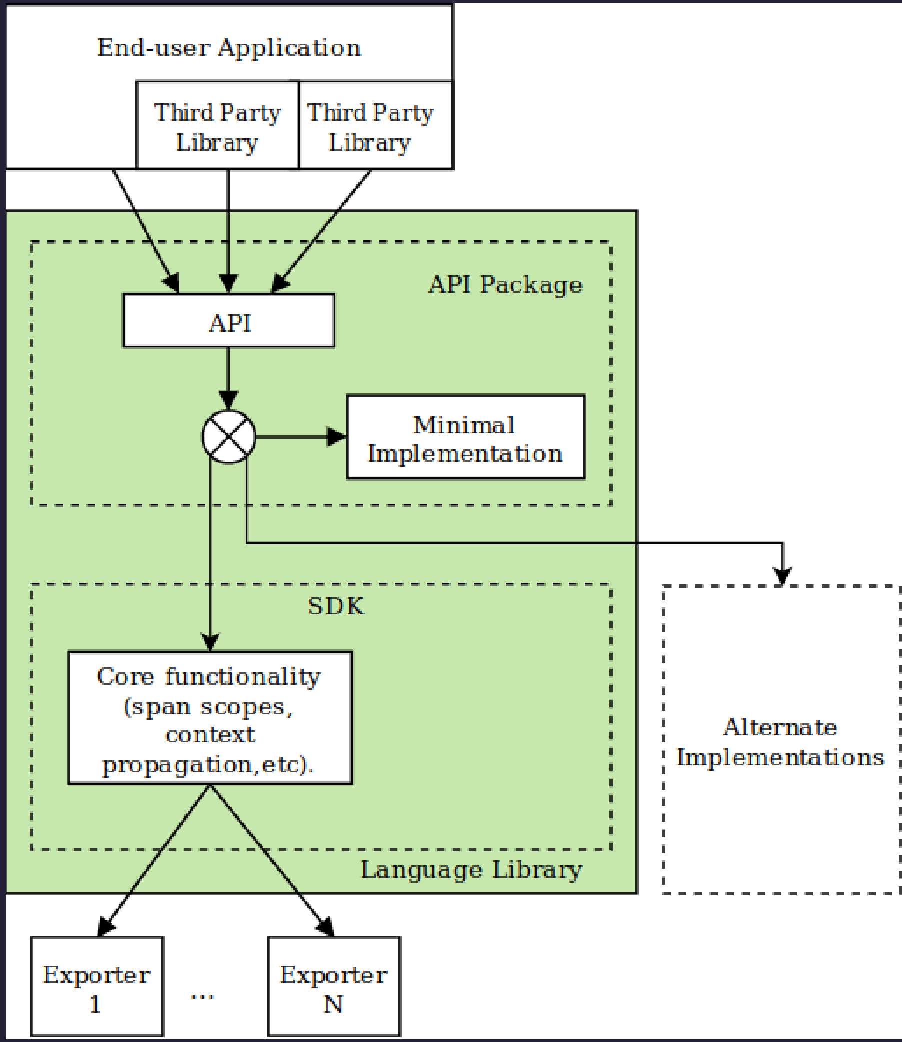
Open source project hosted on CNCF
Specifications, Implementations for instrumentation and transmissions of telemetry data (metrics, logs, traces)

1. Cross-language specifications
2. OpenTelemetry Collector (agent)
3. SDKs for each language
4. Auto Instrumentation



OpenTelemetry Instrumentation

1. **Code-based solutions via official APIs and SDKs for most languages**
   - `API` defines data types and how to generate telemetry data.
   - `SDK` defines a language-specific implementation of the API, plus configuration, data processing and exporting.
2. **Zero-code solutions**
   - Go, .NET, PHP, Python, Java, JavaScript



Manually Instrumentation (Python)

```python
@app.route("/server_request")
def server_request():
    with tracer.start_as_current_span(
        "server_request",
        context=extract(request.headers),
        kind=trace.SpanKind.SERVER,
        attributes=collect_request_attributes(request.environ),
    ):
        print(request.args.get("param"))
        return "served"
```

Programmatically-instrumented server (Python)

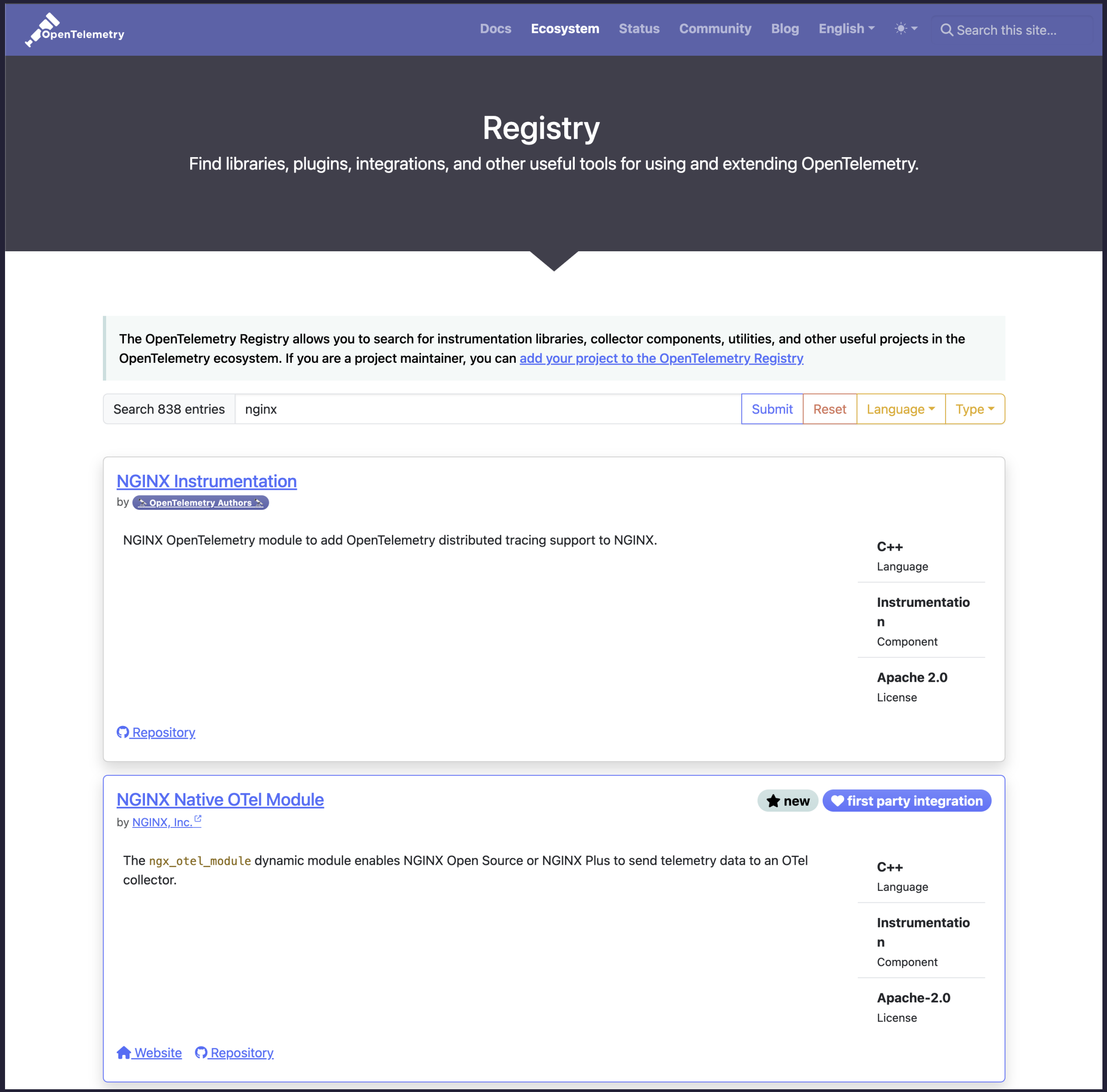
```python
instrumentor = FlaskInstrumentor()

app = Flask(__name__)

instrumentor.instrument_app(app)
# instrumentor.instrument_app(app, excluded_urls="/server_request")
@app.route("/server_request")
def server_request():
    print(request.args.get("param"))
    return "served"
```

OpenTelemetry Registry

OpenTelemetry instrumentation libraries

https://opentelemetry.io/ecosystem/registry/

Search `NGINX`



Registry

Find libraries, plugins, integrations, and other useful tools for using and extending OpenTelemetry.

The OpenTelemetry Registry allows you to search for instrumentation libraries, collector components, utilities, and other useful projects in the OpenTelemetry ecosystem. If you are a project maintainer, you can add your project to the OpenTelemetry Registry

Search 838 entries: nginx — Submit | Reset | Language | Type

NGINX Instrumentation
by OpenTelemetry Authors

NGINX OpenTelemetry module to add OpenTelemetry distributed tracing support to NGINX.

C++ — Language
Instrumentation — Component
Apache 2.0 — License

Repository

NGINX Native OTel Module — new — first party integration
by NGINX, Inc.

The `ngx_otel_module` dynamic module enables NGINX Open Source or NGINX Plus to send telemetry data to an OTel collector.

C++ — Language
Instrumentation — Component
Apache-2.0 — License

Website | Repository

OpenTelemetry Protocol (OTLP)

https://github.com/open-telemetry/opentelemetry-proto/blob/main/docs/specification.md

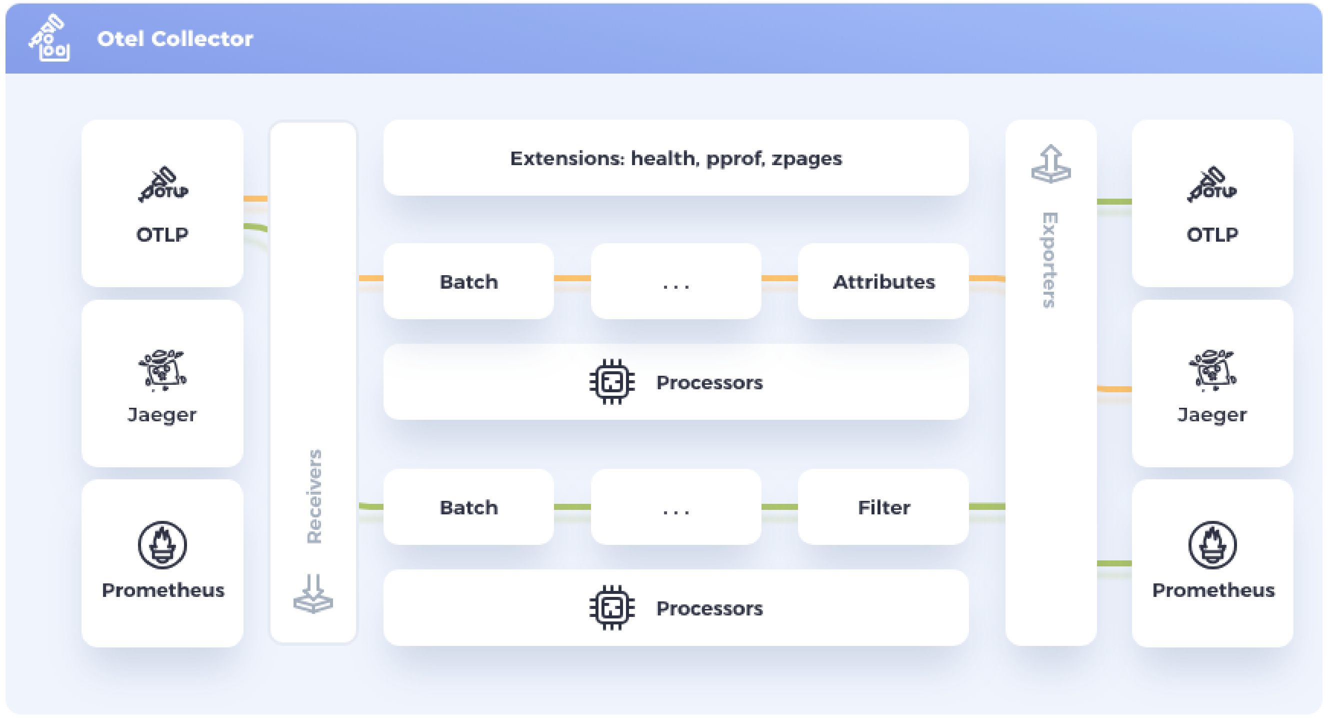
OTLP is implemented over `gRPC` and `HTTP` transports and specifies the Protocol Buffers schema used for payloads.

OTLP is a request/response style protocol where `clients` send requests and the `server` replies with corresponding responses.

All server components must support the following transport compression options: `none`, `gzip`

OpenTelemetry Collector

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



OTEL COLLECTOR

OpenTelemetry Collector Receiver

```yaml
receivers:
  jaeger:
    protocols:
      grpc:
        endpoint: 0.0.0.0:14250
      thrift_compact:
      thrift_http:
  kafka:
    protocol_version: 2.0.0
  prometheus:
    config:
      scrape_configs:
        - job_name: otel-collector
          scrape_interval: 5s
          static_configs:
            - targets: [localhost:8888]
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
```

OpenTelemetry Collector Processors

```yaml
processors:
  attributes:
    actions:
      - key: environment
        value: production
        action: insert
      - key: db.statement
        action: delete
      - key: email
        action: hash
  probabilistic_sampler:
    hash_seed: 22
    sampling_percentage: 15
  memory_limiter:
    check_interval: 5s
    limit_mib: 4000
    spike_limit_mib: 500
  filter:
    metrics:
      include:
        match_type: regexp
        metric_names:
        - prefix/.*
        - prefix_.*
```

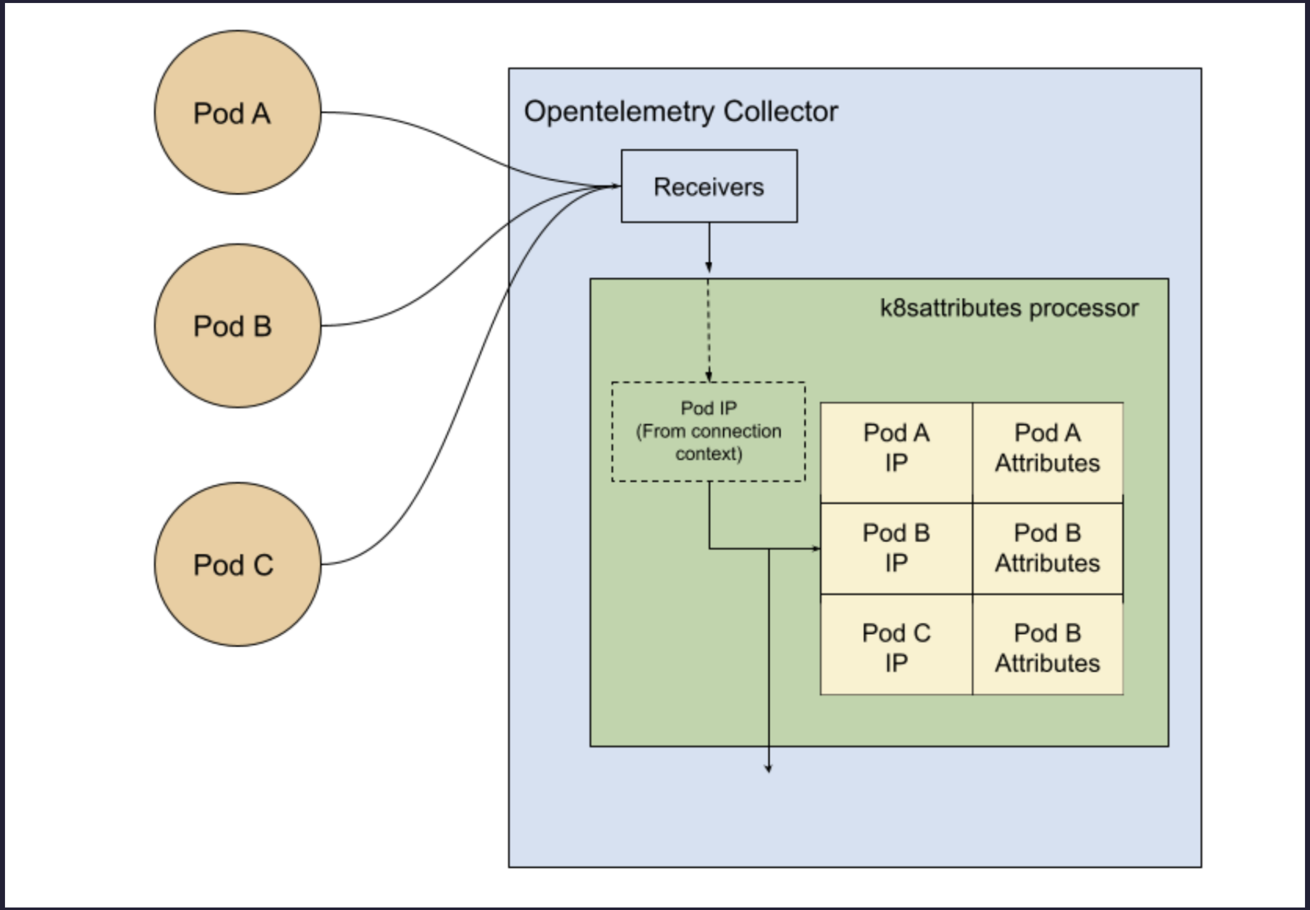
OpenTelemetry Collector Exporter

```yaml
exporters:
  file:
    path: ./filename.json
  jaeger:
    endpoint: http://jaeger-all-in-one:14250
    insecure: true
  kafka:
    protocol_version: 2.0.0
  otlphttp:
    endpoint: https://otlp.example.com:4318
  prometheus:
    endpoint: prometheus:8889
    namespace: default
  prometheusremotewrite:
    endpoint: "http://some.url:9411/api/prom/push"
  zipkin:
    endpoint: "http://localhost:9411/api/v2/spans"
```

Resource Semantic Conventions

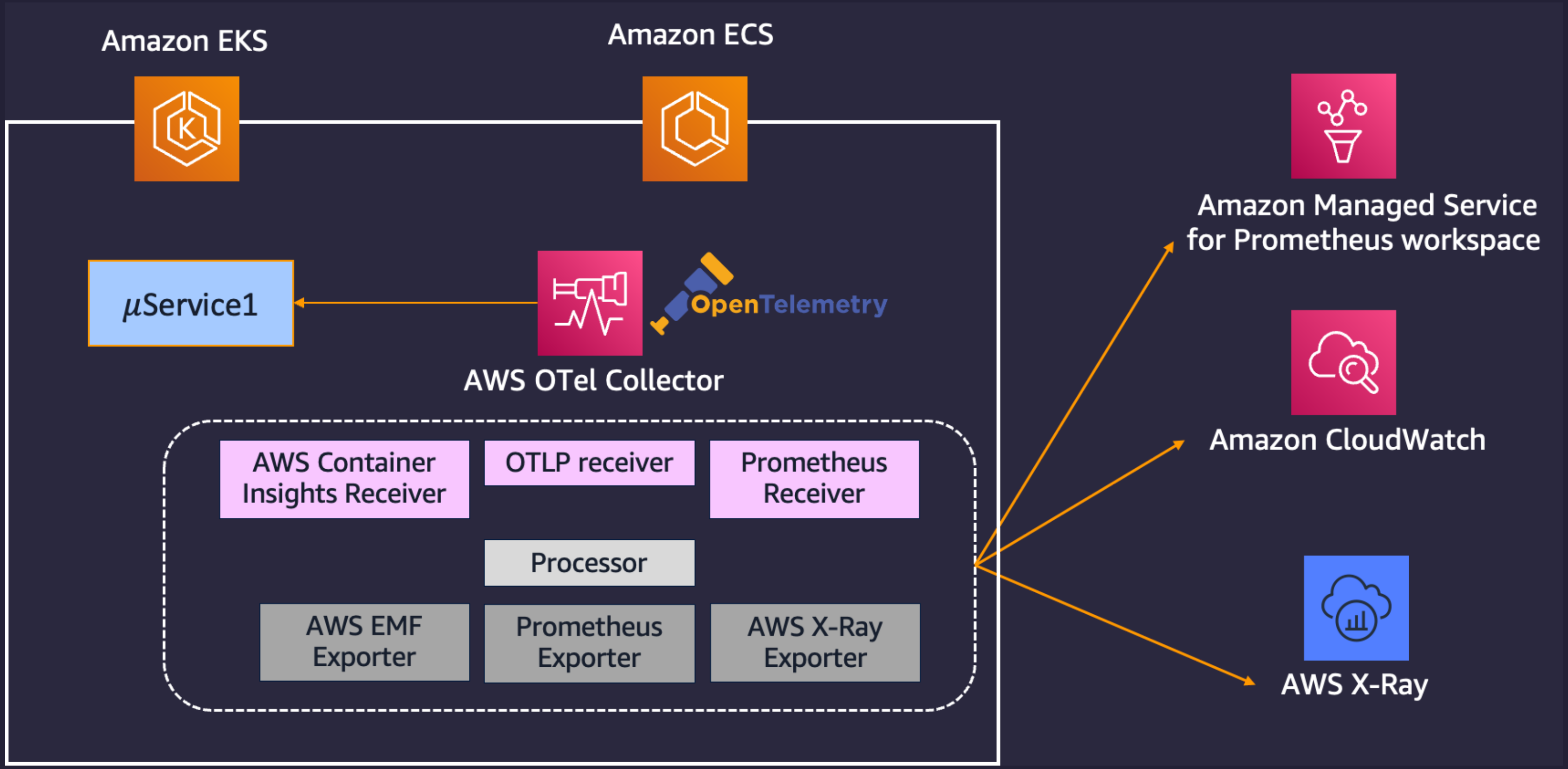
A `Resource` represents the entity producing telemetry as resource attributes.

https://opentelemetry.io/docs/specs/semconv/

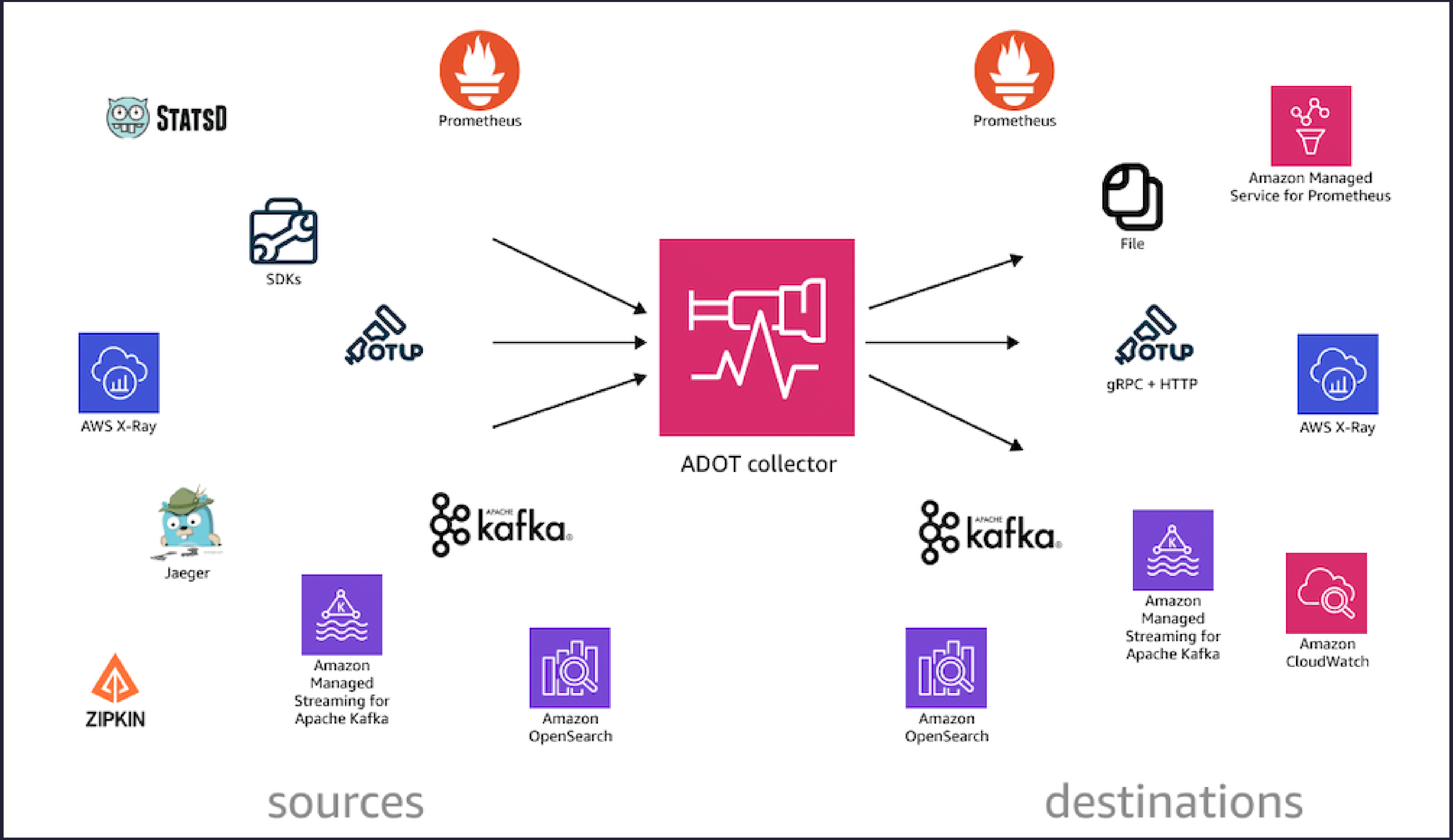


AWS Distro for OpenTelemetry (ADOT)

- Secure, production ready, and supported by AWS OpenTelemetry distribution
- Extend functionality for ease of use on AWS



AWS Distro for OpenTelemetry (ADOT)



OpenTelemetry Ecosystem

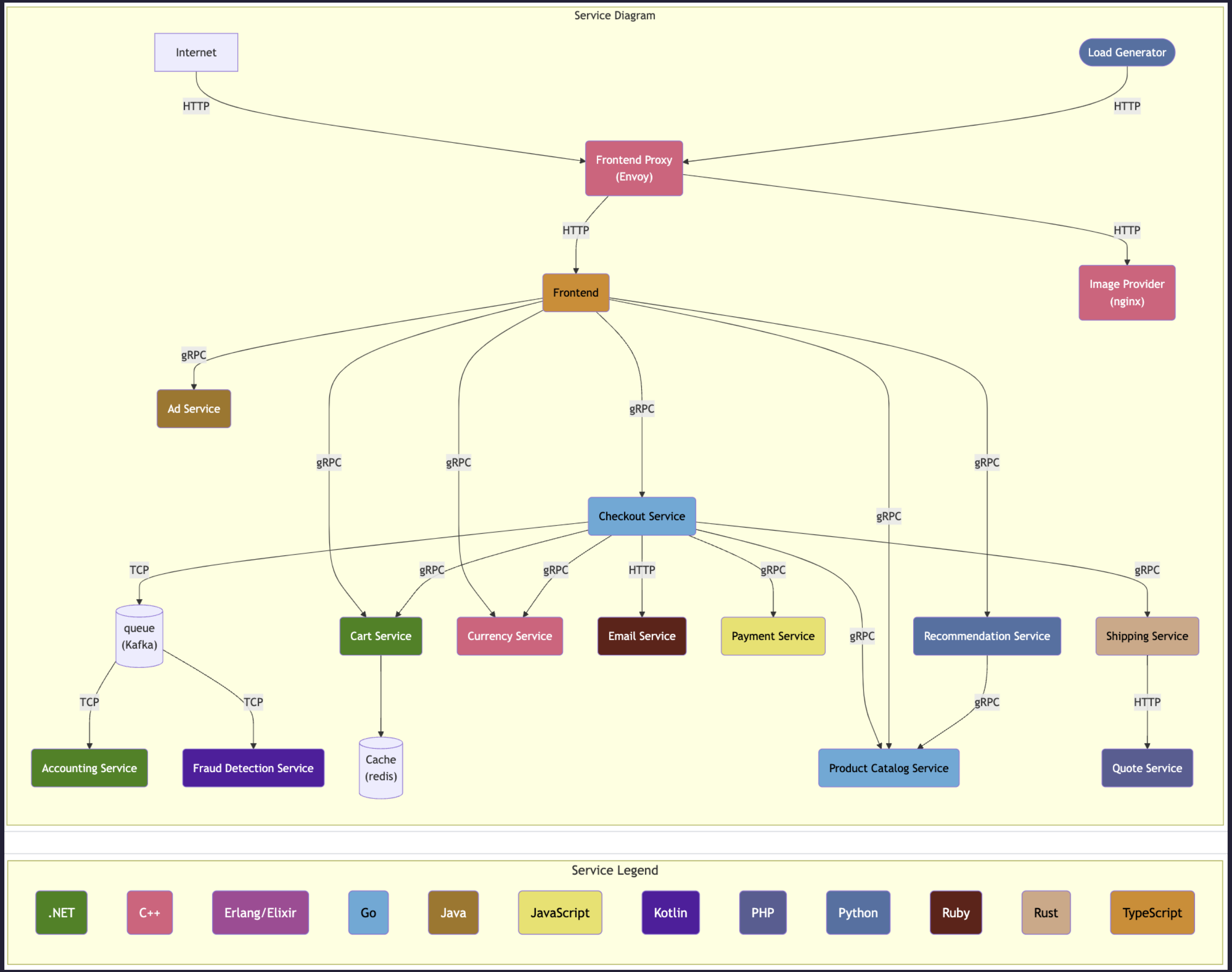
https://opentelemetry.io/ecosystem/

- OpenTelemetry Demo
- Registry
- Adopters
  Organizations that use OpenTelemetry
- Distributions
  List of open source OpenTelemetry distributions maintained by third parties.
- Integrations
  Libraries, services, and apps with first-party support for OpenTelemetry.
- Vendors
  Vendors who natively support OpenTelemetry

OpenTelemetry Demo

OpenTelemetry Demo is composed of microservices written in different programming languages that talk to each other over gRPC and HTTP; and a load generator which uses Locust to fake user traffic.

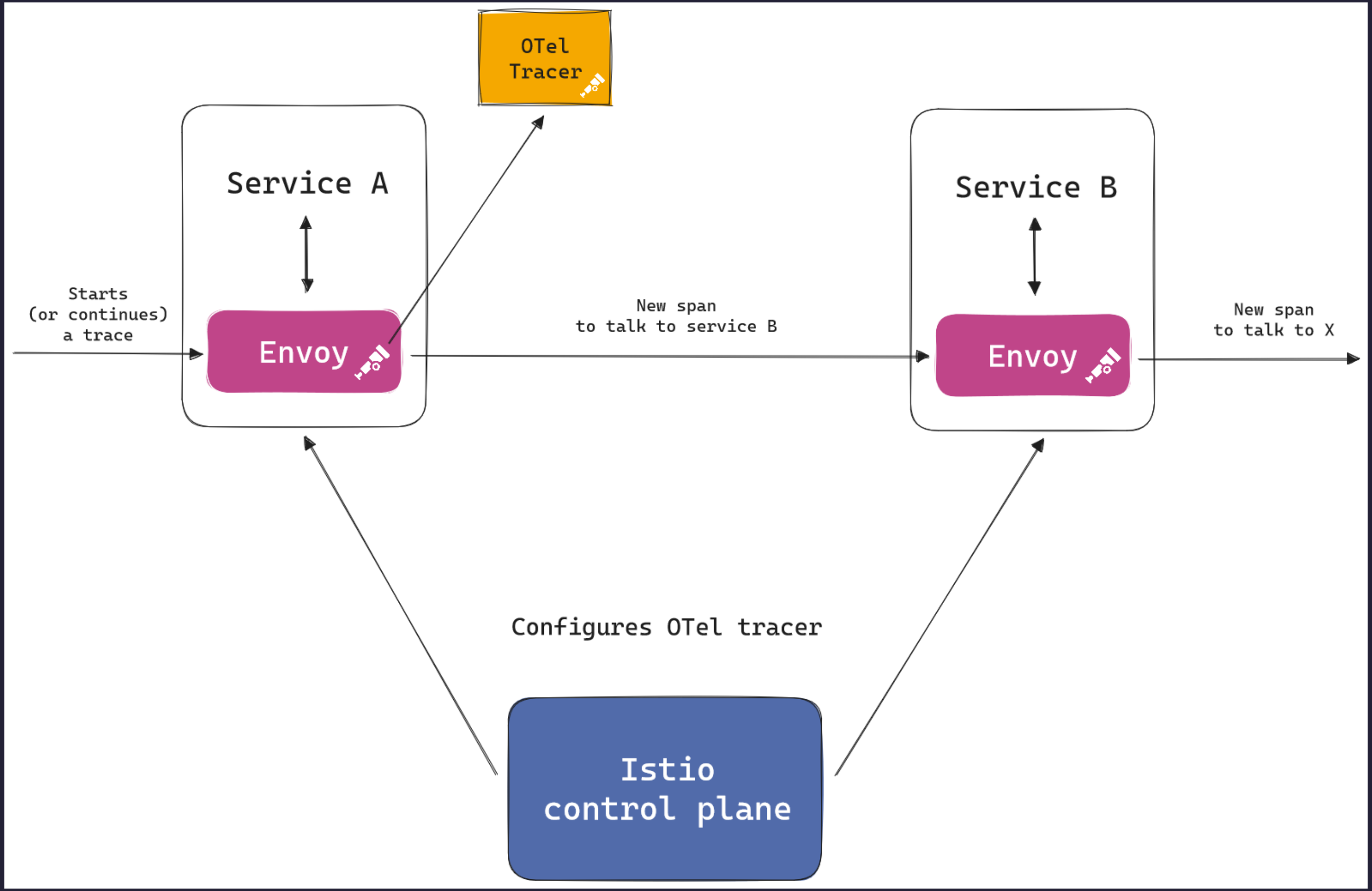
- Web store
- Grafana
- Load Generator
- Jaeger UI



New Otel Feature

Observability in Envoy and Istio

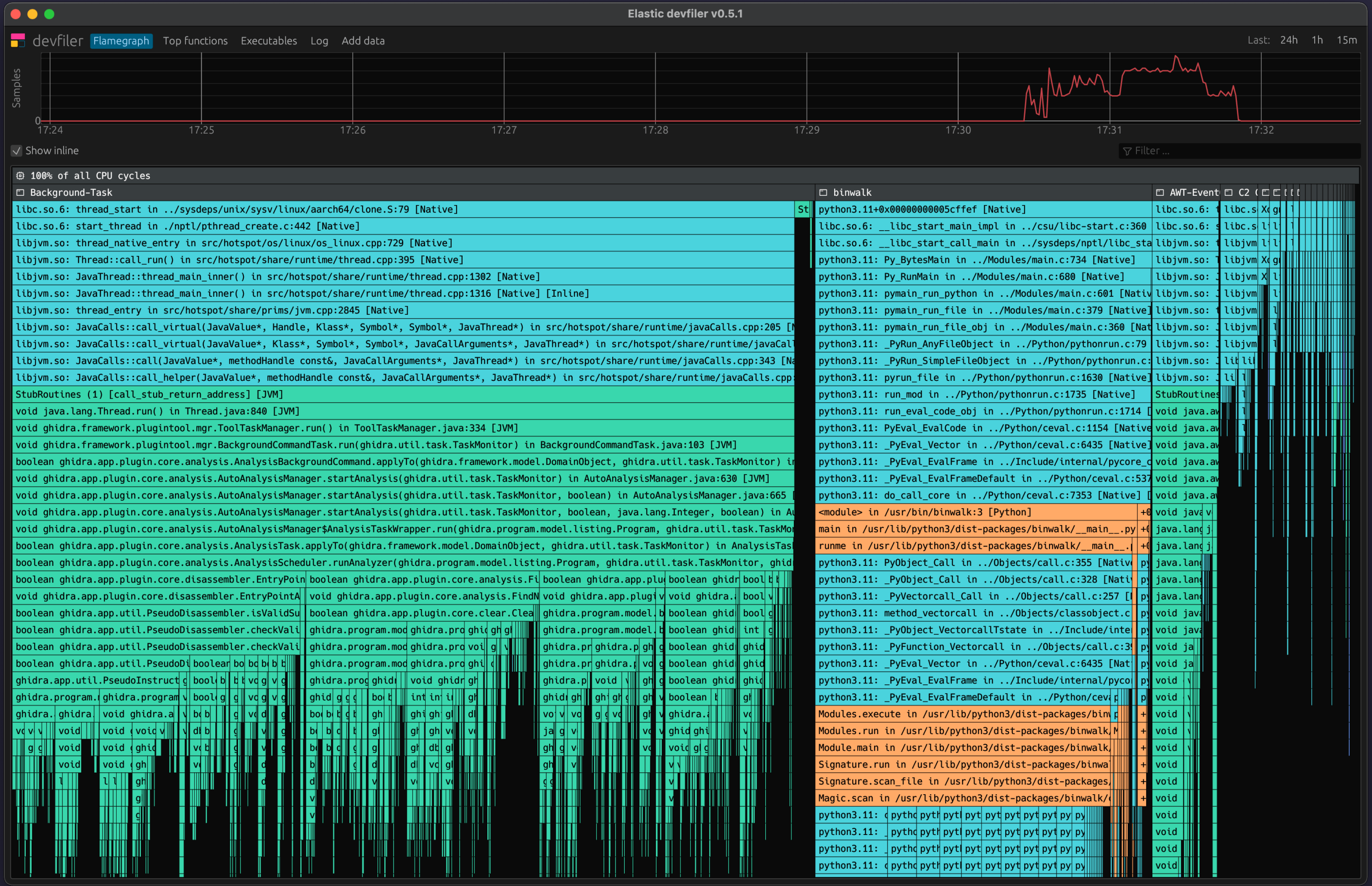
https://opentelemetry.io/blog/2024/new-otel-features-envoy-istio/



New Otel Feature

Continuous Profiling Agent

https://github.com/open-telemetry/opentelemetry-ebpf-profiler



```
100% of all CPU cycles
Background-Task
libc.so.6: thread_start in ../sysdeps/unix/sysv/linux/aarch64/clone.S:79 [Native]
libc.so.6: start_thread in ./nptl/pthread_create.c:442 [Native]
libjvm.so: thread_native_entry in src/hotspot/os/linux/os_linux.cpp:729 [Native]
libjvm.so: Thread::call_run() in src/hotspot/share/runtime/thread.cpp:395 [Native]
libjvm.so: JavaThread::thread_main_inner() in src/hotspot/share/runtime/thread.cpp:1302 [Native]
libjvm.so: JavaThread::thread_main_inner() in src/hotspot/share/runtime/thread.cpp:1316 [Native] [Inline]
libjvm.so: thread_entry in src/hotspot/share/prims/jvm.cpp:2845 [Native]
libjvm.so: JavaCalls::call_virtual(JavaValue*, Handle, Klass*, Symbol*, Symbol*, JavaThread*) in src/hotspot/share/runtime/javaCalls.cpp:205 [N
libjvm.so: JavaCalls::call_virtual(JavaValue*, Klass*, Symbol*, Symbol*, JavaCallArguments*, JavaThread*) in src/hotspot/share/runtime/javaCal
libjvm.so: JavaCalls::call(JavaValue*, methodHandle const&, JavaCallArguments*, JavaThread*) in src/hotspot/share/runtime/javaCalls.cpp:343 [Na
libjvm.so: JavaCalls::call_helper(JavaValue*, methodHandle const&, JavaCallArguments*, JavaThread*) in src/hotspot/share/runtime/javaCalls.cpp
StubRoutines (1) [call_stub_return_address] [JVM]
void java.lang.Thread.run() in Thread.java:840 [JVM]
void ghidra.framework.plugintool.mgr.ToolTaskManager.run() in ToolTaskManager.java:334 [JVM]
void ghidra.framework.plugintool.mgr.BackgroundCommandTask.run(ghidra.util.task.TaskMonitor) in BackgroundCommandTask.java:103 [JVM]
boolean ghidra.app.plugin.core.analysis.AnalysisBackgroundCommand.applyTo(ghidra.framework.model.DomainObject, ghidra.util.task.TaskMonitor) in
void ghidra.app.plugin.core.analysis.AutoAnalysisManager.startAnalysis(ghidra.util.task.TaskMonitor) in AutoAnalysisManager.java:630 [JVM]
void ghidra.app.plugin.core.analysis.AutoAnalysisManager.startAnalysis(ghidra.util.task.TaskMonitor, boolean) in AutoAnalysisManager.java:665
void ghidra.app.plugin.core.analysis.AutoAnalysisManager.startAnalysis(ghidra.util.task.TaskMonitor, boolean, java.lang.Integer, boolean) in Au
void ghidra.app.plugin.core.analysis.AutoAnalysisManager$AnalysisTaskWrapper.run(ghidra.program.model.listing.Program, ghidra.util.task.TaskMon
boolean ghidra.app.plugin.core.analysis.AnalysisTask.applyTo(ghidra.framework.model.DomainObject, ghidra.util.task.TaskMonitor) in AnalysisTas
boolean ghidra.app.plugin.core.analysis.AnalysisScheduler.runAnalyzer(ghidra.program.model.listing.Program, ghidra.util.task.TaskMonitor, ghid
boolean ghidra.app.plugin.core.disassembler.EntryPoin | boolean ghidra.app.plugin.core.analysis.Fi | boolean ghidra.app.plu | boolean ghidr | bool
void ghidra.app.plugin.core.disassembler.EntryPointA | void ghidra.app.plugin.core.analysis.FindN | void ghidra.app.plugi | void ghidra. | bool
boolean ghidra.app.util.PseudoDisassembler.isValidSu | boolean ghidra.app.plugin.core.clear.Clea | ghidra.program.model. | boolean ghid | bool
boolean ghidra.app.util.PseudoDisassembler.checkVali | ghidra.program.mod | ghidra.program.model. | boolean ghid
boolean ghidra.app.util.PseudoDisassembler.checkVali | ghidra.program.mod | ghidra.pr ghidra.p | boolean ghid
boolean ghidra.app.util.PseudoDi | ghidra.program.mod | ghidra.pr ghidra. | boolean ghid
ghidra.app.util.PseudoInstruct | ghidra.prog | ghidra.p | boolean ghid
ghidra.program. ghidra.program | ghid | void ghidr | boolean
```

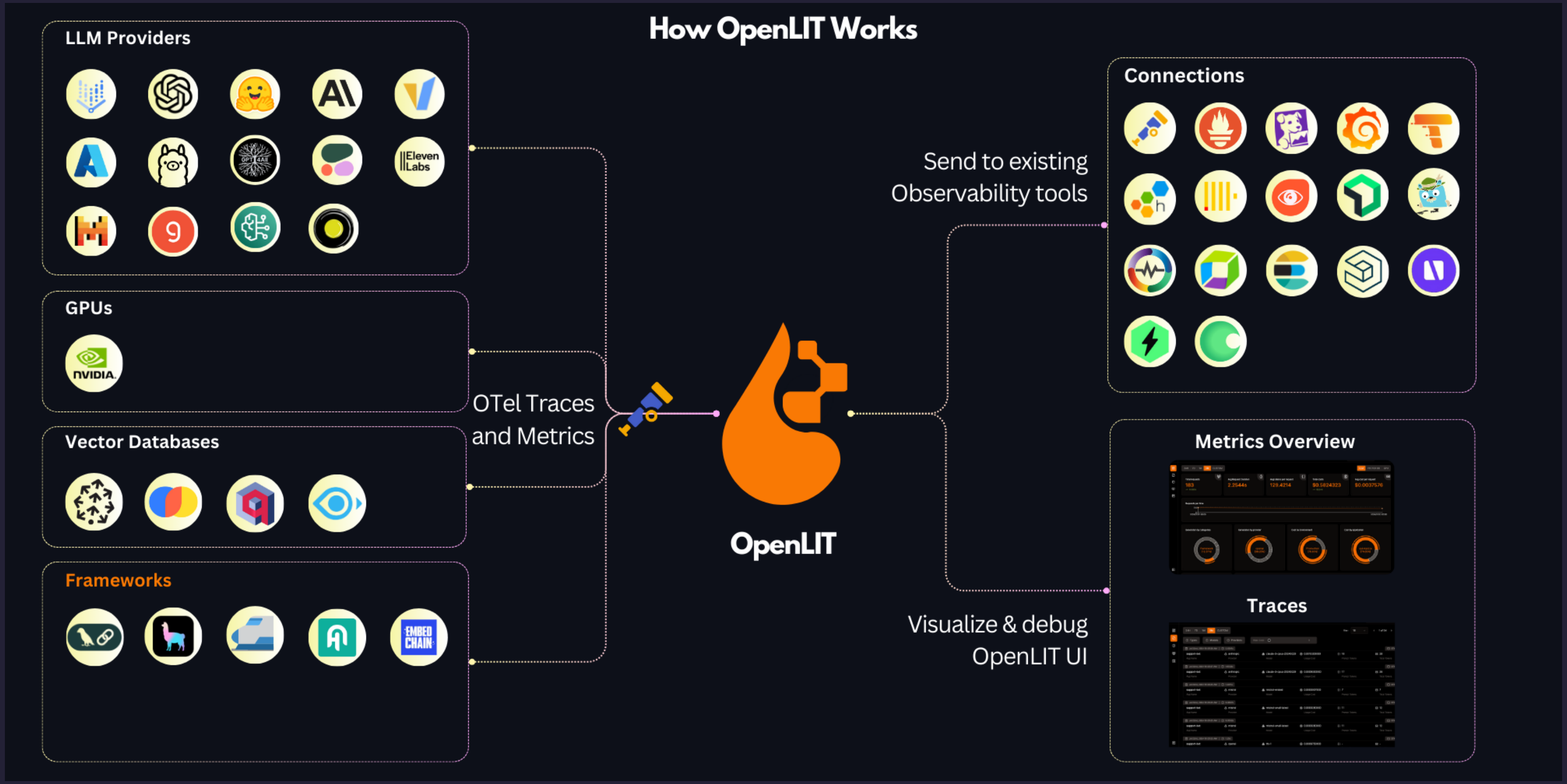
```
binwalk
python3.11+0x00000000005cffef [Native]
libc.so.6: __libc_start_main_impl in ../csu/libc-start.c:360
libc.so.6: __libc_start_call_main in ../sysdeps/nptl/libc_sta
python3.11: Py_BytesMain in ../Modules/main.c:734 [Native]
python3.11: Py_RunMain in ../Modules/main.c:680 [Native]
python3.11: pymain_run_python in ../Modules/main.c:601 [Nativ
python3.11: pymain_run_file in ../Modules/main.c:379 [Native]
python3.11: pymain_run_file_obj in ../Modules/main.c:360 [Nat
python3.11: _PyRun_AnyFileObject in ../Python/pythonrun.c:79
python3.11: _PyRun_SimpleFileObject in ../Python/pythonrun.c:
python3.11: pyrun_file in ../Python/pythonrun.c:1630 [Native]
python3.11: run_mod in ../Python/pythonrun.c:1735 [Native]
python3.11: run_eval_code_obj in ../Python/pythonrun.c:1714 [
python3.11: PyEval_EvalCode in ../Python/ceval.c:1154 [Native
python3.11: _PyEval_Vector in ../Python/ceval.c:6435 [Native]
python3.11: _PyEval_EvalFrame in ../Include/internal/pycore_c
python3.11: _PyEval_EvalFrameDefault in ../Python/ceval.c:537
python3.11: do_call_core in ../Python/ceval.c:7353 [Native]
<module> in /usr/bin/binwalk:3 [Python]
main in /usr/lib/python3/dist-packages/binwalk/__main__.py
runme in /usr/lib/python3/dist-packages/binwalk/__main__.
python3.11: PyObject_Call in ../Objects/call.c:355 [Native
python3.11: _PyObject_Call in ../Objects/call.c:328 [Nati
python3.11: _PyVectorcall_Call in ../Objects/call.c:257 [
python3.11: method_vectorcall in ../Objects/classobject.c
python3.11: _PyObject_VectorcallTstate in ../Include/inte
python3.11: _PyFunction_Vectorcall in ../Objects/call.c:3
python3.11: _PyEval_Vector in ../Python/ceval.c:6435 [Nat
python3.11: _PyEval_EvalFrame in ../Include/internal/pyco
python3.11: _PyEval_EvalFrameDefault in ../Python/ceva
Modules.execute in /usr/lib/python3/dist-packages/binw
Modules.run in /usr/lib/python3/dist-packages/binwalk/
Module.main in /usr/lib/python3/dist-packages/binwalk/
Signature.run in /usr/lib/python3/dist-packages/binwa
Signature.scan_file in /usr/lib/python3/dist-packages/
Magic.scan in /usr/lib/python3/dist-packages/binwalk/
python3.11:
python3.11:
python3.11:
python3.11:
```

```
AWT-Event
libc.so.6:
libc.so.6:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
libjvm.so:
StubRoutines
void java.aw
void java.aw
void java.aw
void java.aw
void java.aw
void java.aw
void java
java.lang
java.lang
java.lang
java.lang
java.lang
void java
void java
void ja
void ja
void
void
void
void
void
void
void
void
void
void
void
```

New Otel Feature

LLM Observability

https://github.com/openlit/openlit



References

- https://opentelemetry.io/docs/
- https://w3c.github.io/trace-context/
- https://github.com/open-telemetry/opentelemetry-specification
- https://opentelemetry.io/docs/specs/semconv/
- https://opentelemetry.io/docs/specs/otel/protocol/
- https://opentelemetry.io/docs/concepts/sampling/
- https://opentelemetry.io/docs/demo/
- https://opentelemetry.io/blog/2024/

Thank You

@ddiiwoong
jinwoong@amazon.com