

OpenTelemetry Journey

2024.09

Jinwoong Kim

Who am I?

- 김진웅 (Jinwoong Kim)
- Cloud Architect @AWS Professional Services
- Speaker, Translator
- @ddiwoong

Observability Introduction

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

Observability

Logs - 무슨 일이 일어난거야? (Lines of text)

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

Metrics - 어떤 지표가 문제야? (Time-ordered set of data)

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

Tracing - 그 일이 왜 일어났지? (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

- 시스템의 프로세스에서 특정 부분의 지연(latency)을 알려주는 원격 측정(telemetry)방법
- 요청(Requests)이 마이크로서비스 및 서버리스 아키텍처를 통해 전파될때 이동하는 경로를 기록
- 마이크로서비스 환경같은 최신 아키텍처에서 수많은 구성 요소간의 종속성과 관계를 측정하고 지연 병목을 찾아내는 도구이기 때문에 Observability에서 매우 중요

Trace



Proxy

Service A

Service B

Service C

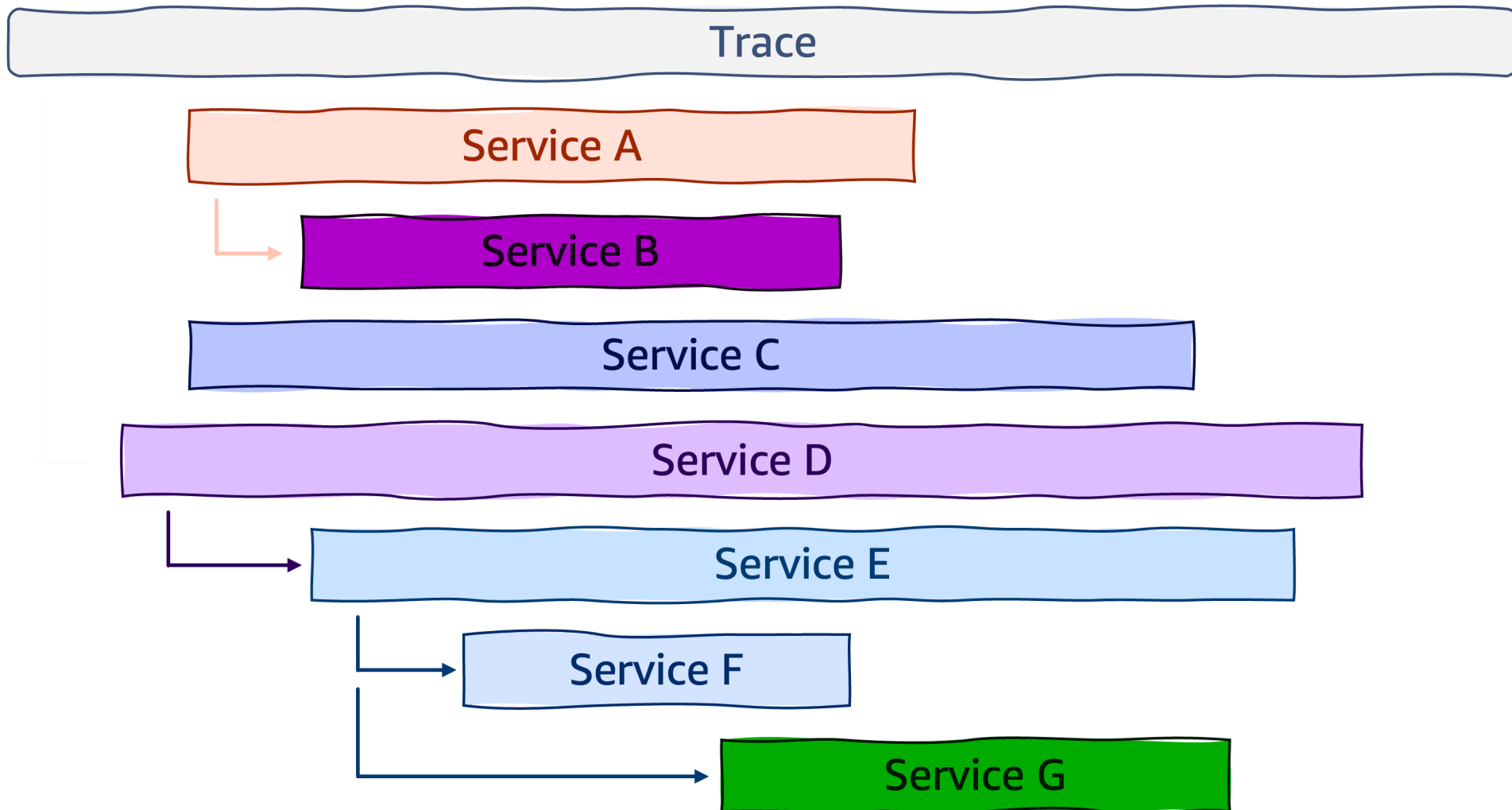
Service D

Service E

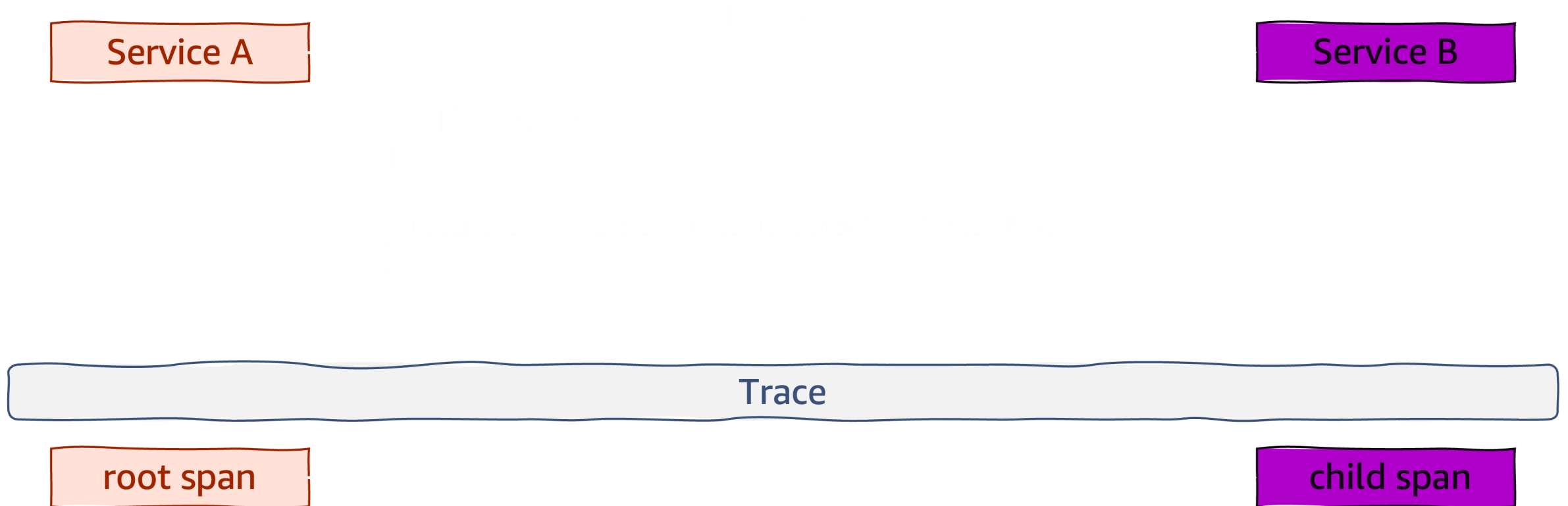
Service F

Service G

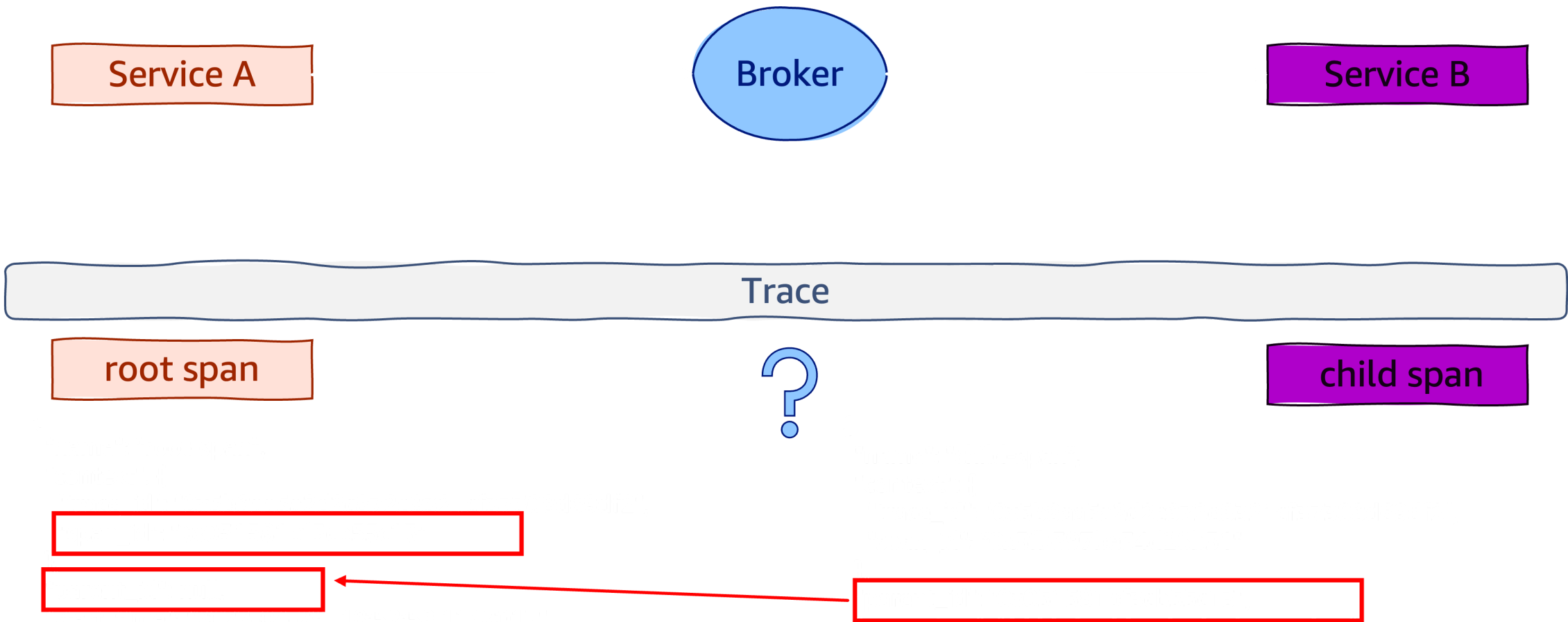
Trace (Span)



Trace - context propagation

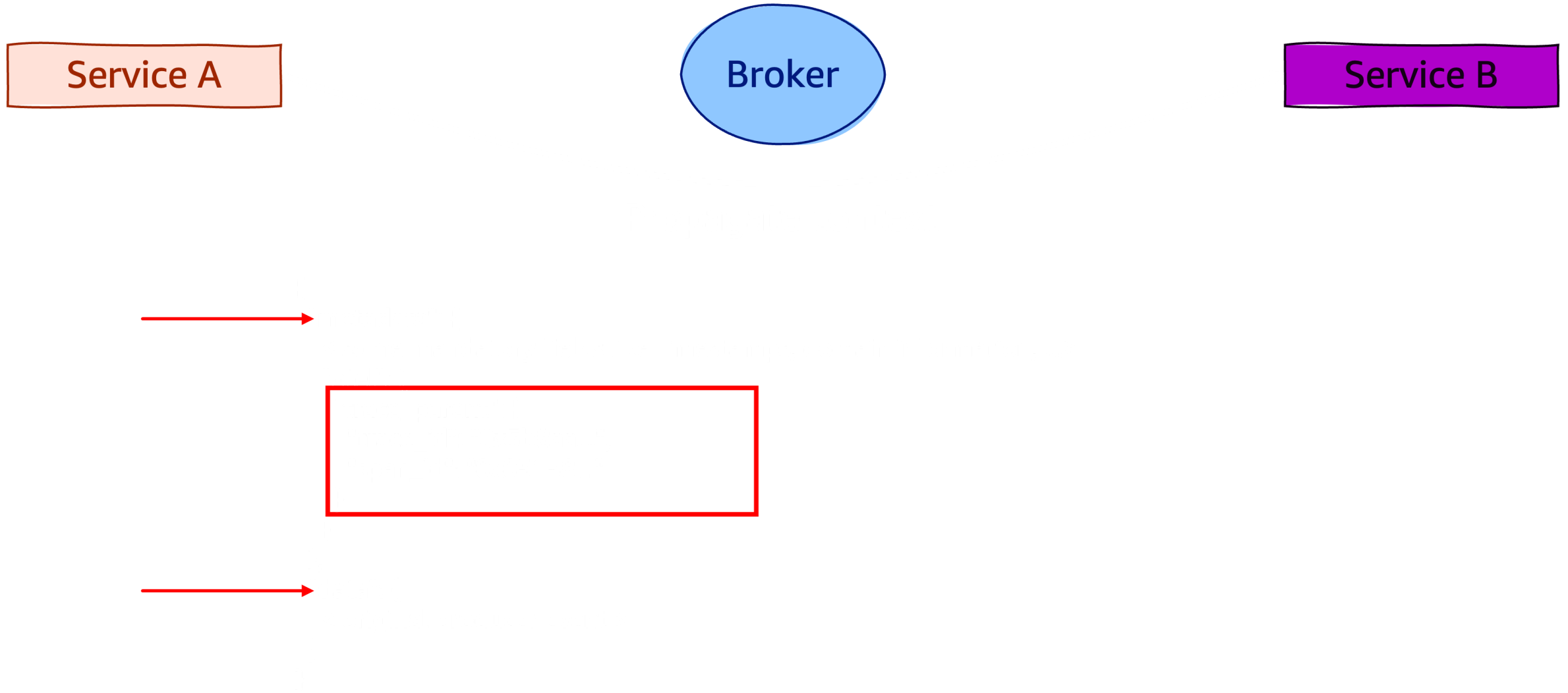


Trace - context propagation

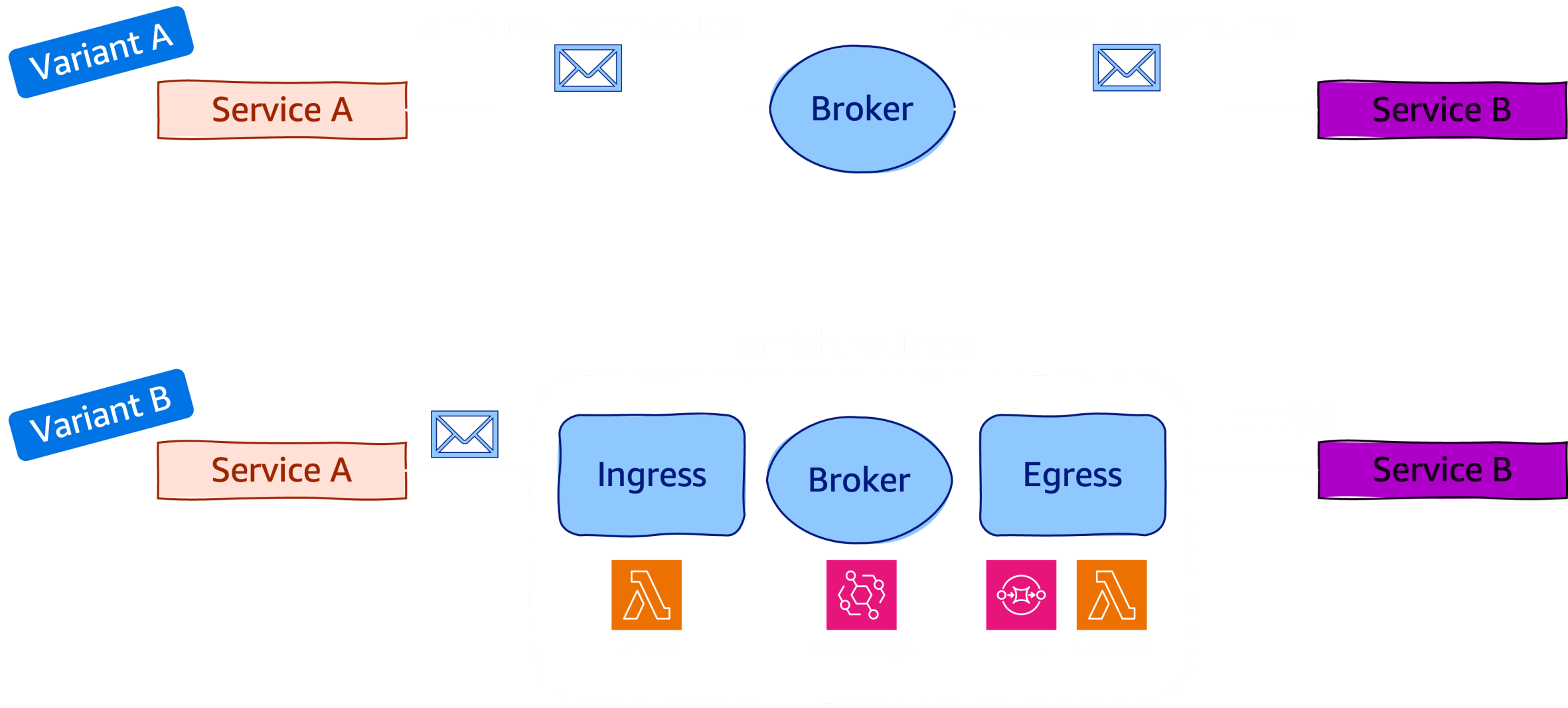


```
graph LR; SA[Service A] -- "Request" --> B((Broker)); B -- "Request" --> SB1[Service B]; B -- "Request" --> SB2[Service B]; B -- "Response" --> SA; SB1 -- "Response" --> B; SB2 -- "Response" --> B;
```

The diagram illustrates a Request-Reply messaging pattern. It features three main components: Service A (orange rounded rectangle), Broker (blue oval), and Service B (purple rounded rectangle). Service A sends a request to the Broker. The Broker then distributes the request to multiple instances of Service B. The responses from Service B are sent back to the Broker, which then returns the aggregated response to Service A. A red box highlights the Broker's role in distributing the request and aggregating the responses.



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

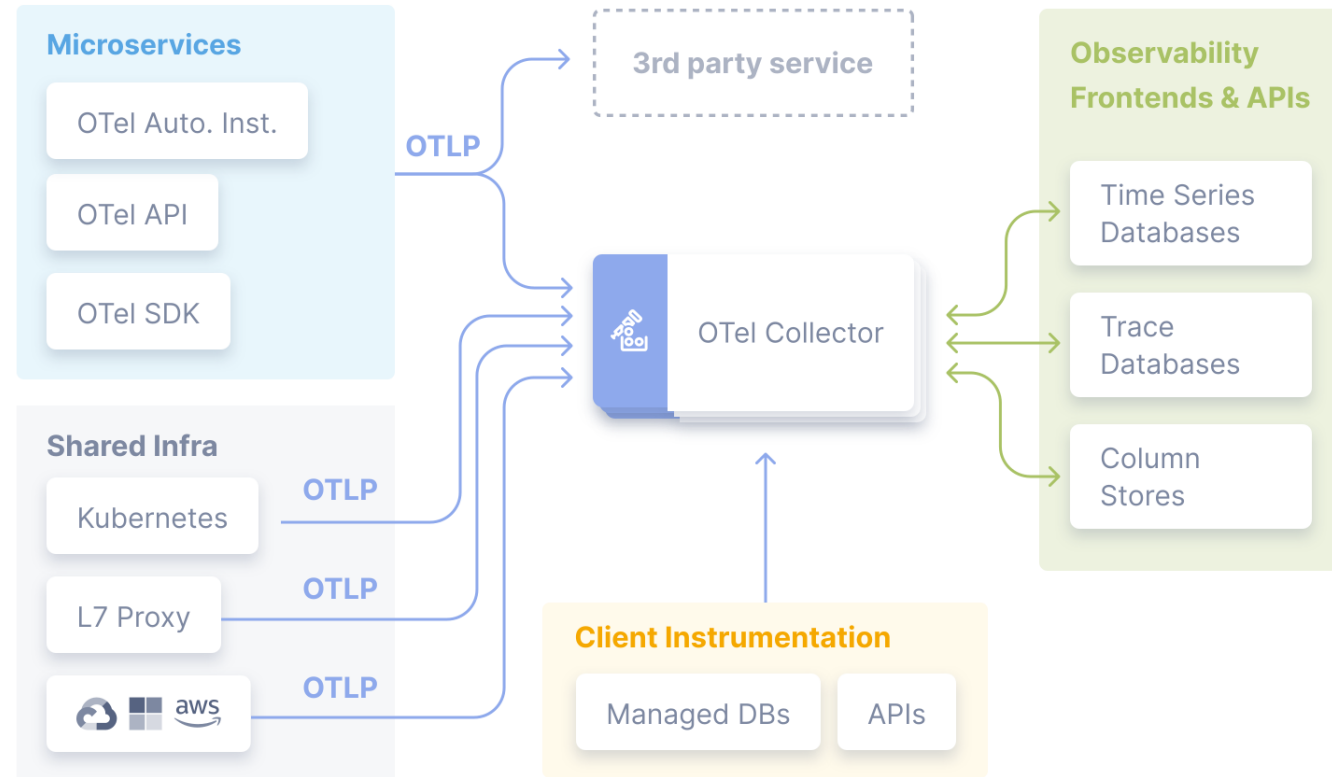


Hello, OpenTelemetry

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](#)

Language	Traces	Metrics	Logs
C++	Stable	Stable	Stable
C#/.NET	Stable	Stable	Stable
Erlang/Elixir	Stable	Development	Development
Go	Stable	Stable	Beta
Java	Stable	Stable	Stable
JavaScript	Stable	Stable	Development
PHP	Stable	Stable	Stable
Python	Stable	Stable	Development
Ruby	Stable	Development	Development
Rust	Beta	Alpha	Alpha
Swift	Stable	Development	Development

Manually Instrumentation (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)

```
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/>

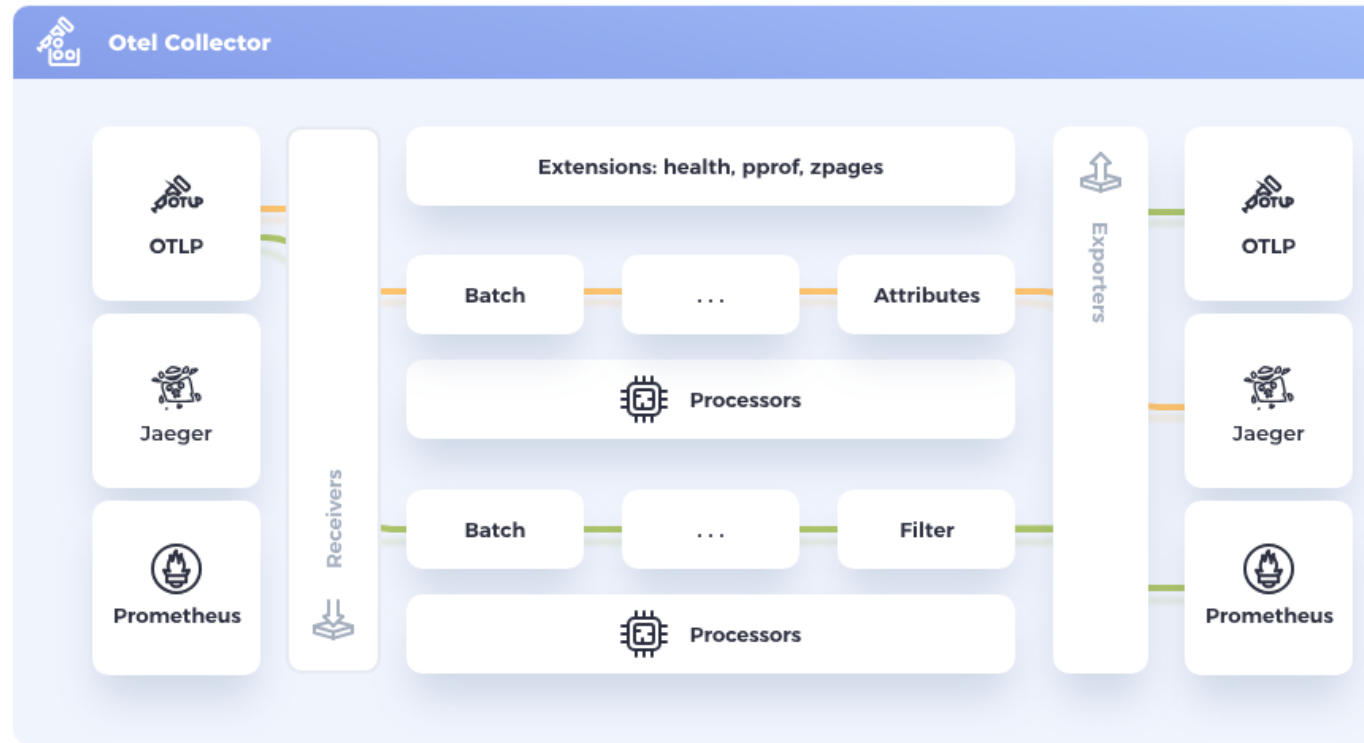
Search **NGINX**

The screenshot displays the OpenTelemetry Registry website. The header includes the OpenTelemetry logo and navigation links: Docs, Ecosystem, Status, Community, Blog, English, and a search bar. The main heading is "Registry" with the subtitle "Find libraries, plugins, integrations, and other useful tools for using and extending OpenTelemetry." Below this, a search bar shows "Search 838 entries" with the filter "nginx" applied. A "Submit" button and a "Reset" button are visible, along with dropdown menus for "Language" and "Type".

The search results are displayed in a list. The first result is "NGINX Instrumentation" by "OpenTelemetry Authors". It is described as "NGINX OpenTelemetry module to add OpenTelemetry distributed tracing support to NGINX." The metadata on the right indicates it is a "C++ Language" "Instrumentation Component" with an "Apache 2.0 License". A "Repository" link is provided at the bottom.

The second result is "NGINX Native OTEL Module" by "NGINX, Inc.". It is marked as "new" and a "first party integration". The description states: "The `ngx_otel_module` dynamic module enables NGINX Open Source or NGINX Plus to send telemetry data to an OTEL collector." The metadata on the right indicates it is a "C++ Language" "Instrumentation Component" with an "Apache-2.0 License". Links for "Website" and "Repository" are provided at the bottom.

OpenTelemetry collector



OTEL COLLECTOR

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.

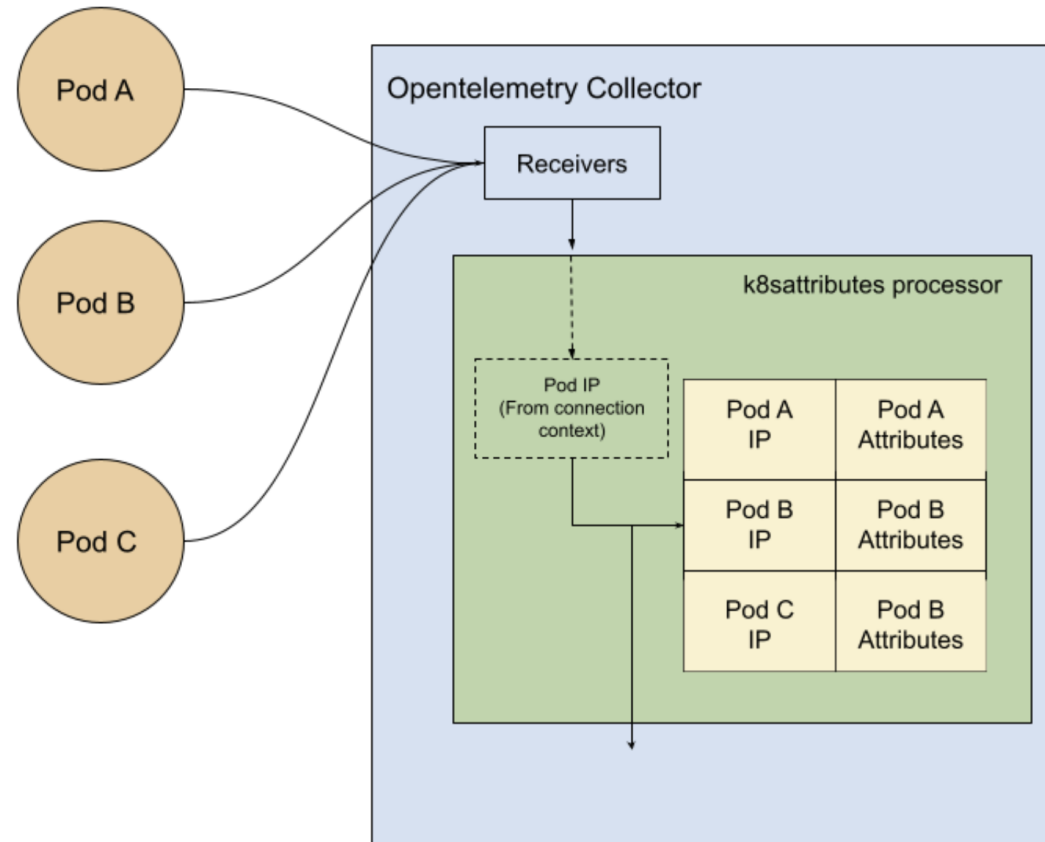
OTLP is a request/response style protocol where `clients` send requests and the `server` replies with corresponding responses.

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

Resource Semantic Conventions

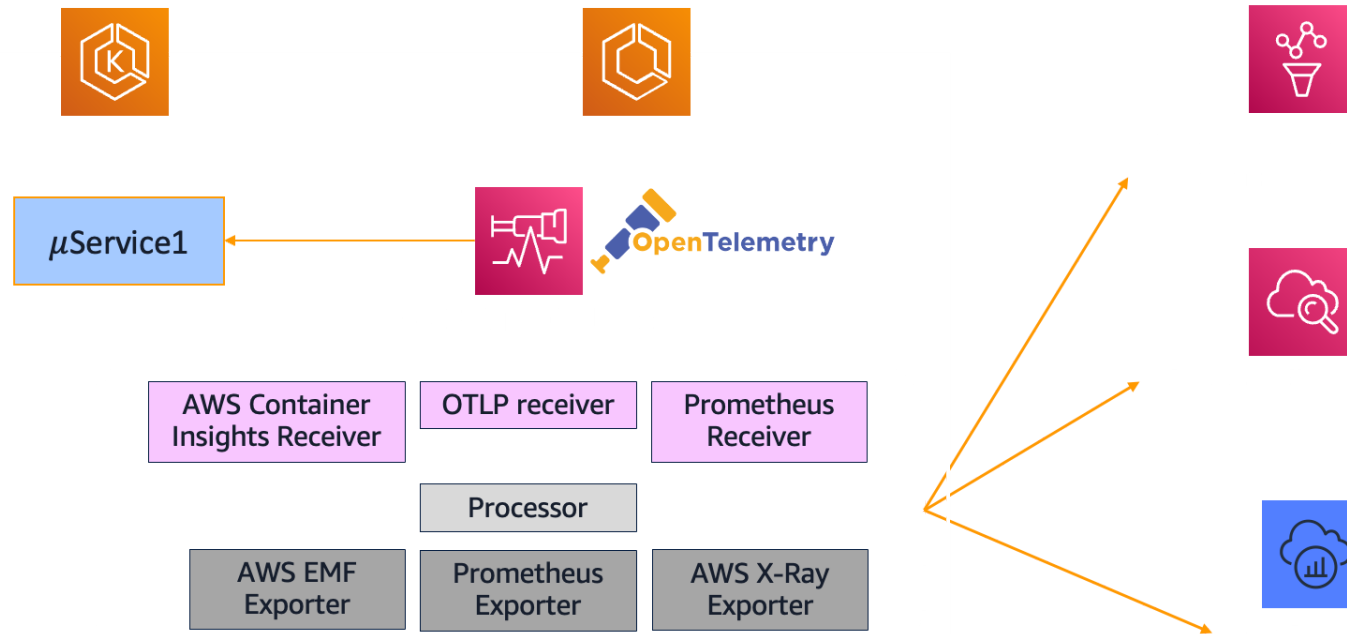
A **Resource** represents the entity producing telemetry as resource attributes.

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

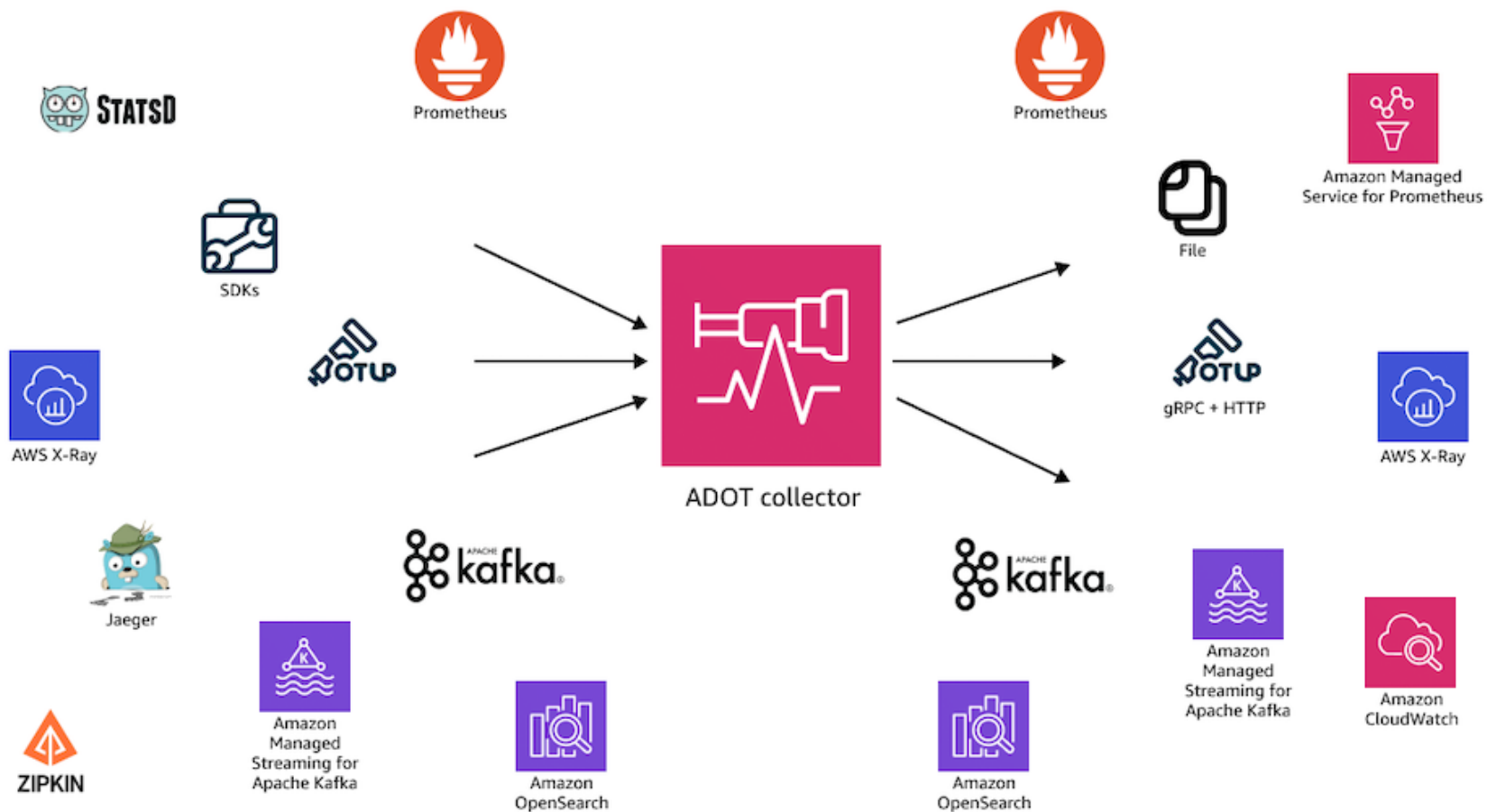


AWS Distro for OpenTelemetry (ADOT)

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



AWS Distro for OpenTelemetry (ADOT)



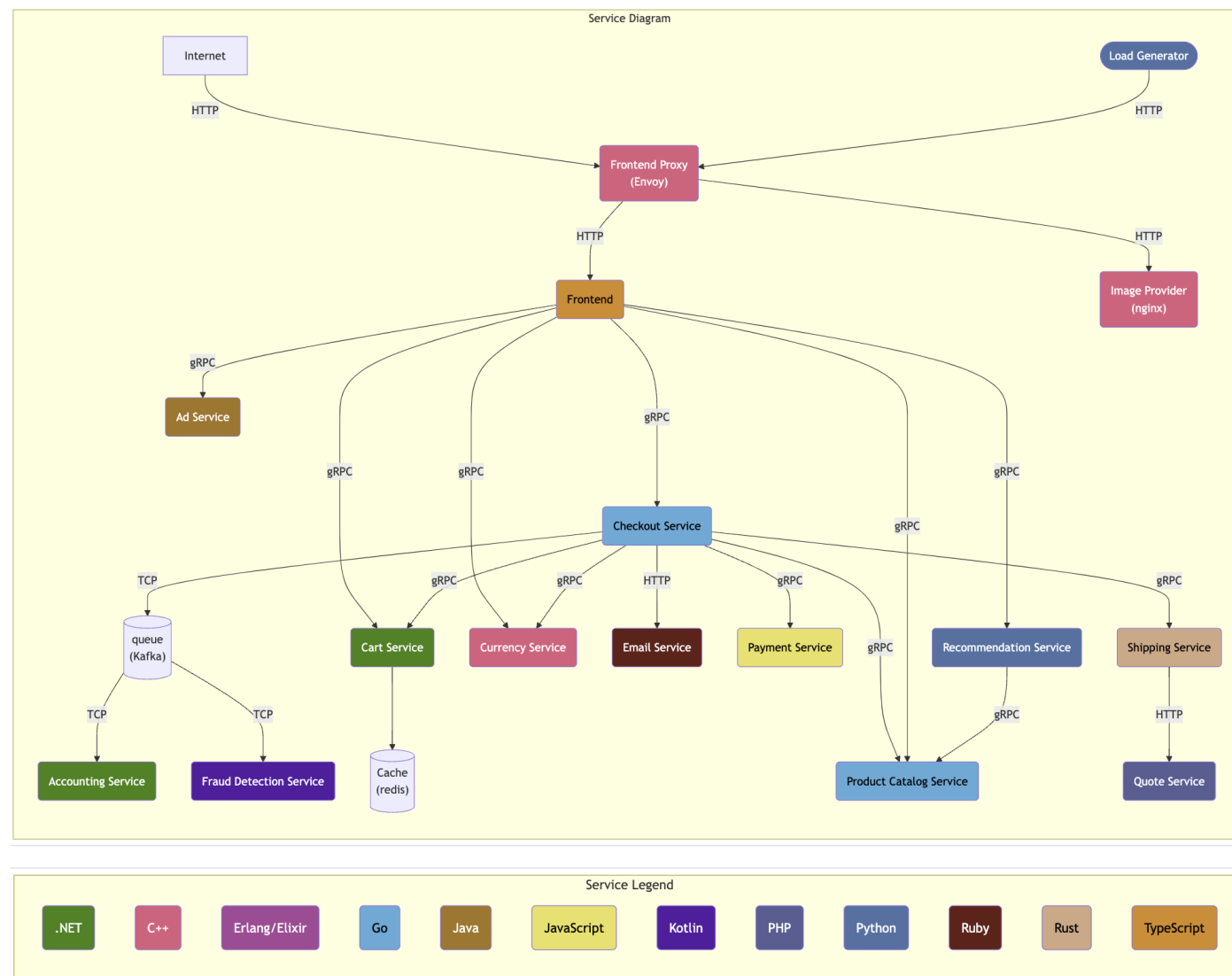
sources

destinations

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

Envoy and Istio

Profiling Agent

LLM Observability

OpenTelemetry Collector Antipatterns

References

- <https://opentelemetry.io/docs/>
- <https://w3c.github.io/trace-context/>
- <https://w3c.github.io/baggage/>
- <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