Distributed Tracing Using Open Telemetry

Fahri Yardımcı | Site Reliability Engineer, Atlassian | @FahriYardimci

Agenda

- Monitoring
- Tracing and Concepts
- Distributed Tracing
- OpenTelemetry

Monitoring

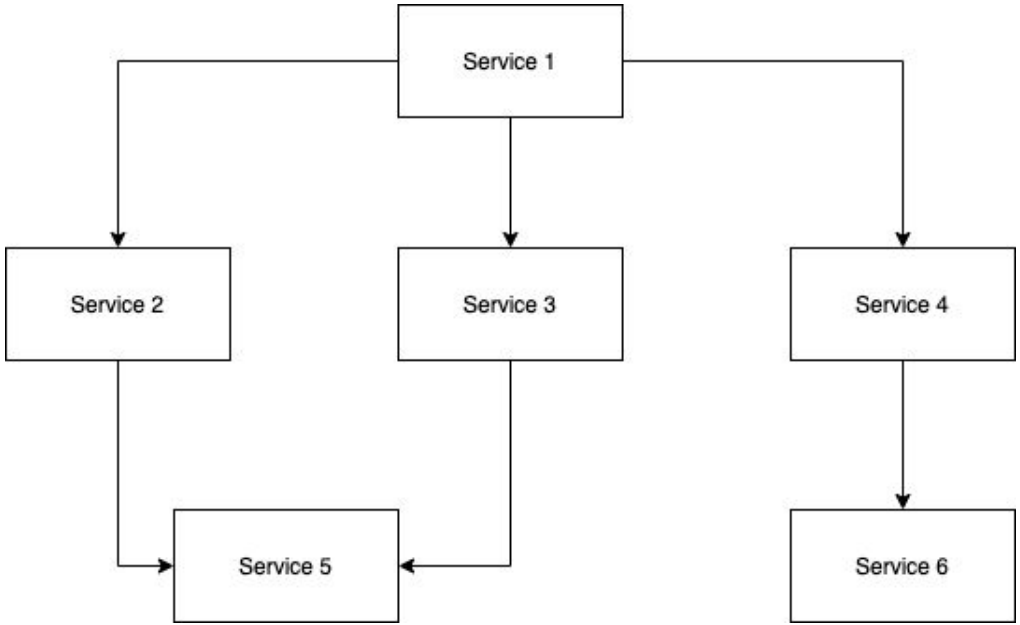
Who doesn’t love metrics ?

Monitor our systems

Ensuring the application flows are work as expected

Configure our capacity plannings

Ensuring our servers are in good condition



Service Dependency Graph - Increasing microservices

Can we identify the problem by just doing blackbox monitoring?

Identifying the problem

Layer

Identifying which layer is causing the problem? Is it a middleware or database service?

Dependent parties

Maybe the problem is cloud provider ?

Latencies

If latencies increases and if we have the durations per services we can query them

Log search

Searching relevant exceptions or errors in logging tools.

Tracing

Trace is collection of spans.

So what is spans?

Tracing

Span

Sampling

Context

Inject/Extract

What is Span?

The “span” is the primary building block of a distributed trace, representing an individual unit of work done in a distributed system.[1]

Basically, span is a work which done by our functions.

1-https://opentracing.io/docs/overview/spans/

Tracing

Span

Sampling

Context

Inject/Extract

Spans have their own identifier and names. Using the identifier we can understand the parent-child relationship between spans.

Our clients will add a timestamp, durations, and other fields into the span.

Tracing

Span

Sampling

Context

Inject/Extract

```
span := opentracing.GlobalTracer().StartSpan( operationName: "hello-world")
defer span.Finish()
```

ROOT	TRACE ID	START TIME		DURATION ↑
TALKDEMO (hello-world)	724f8bda9295a064	07/20 22:32:06:531	(2 minutes ago)	14µs

TALKDEMO (1)

TALKDEMO

hello-world

Span ID: 724f8bda9295a064 Parent ID: None

Annotations

SHOW ALL ANNOTATIONS

Tags

Local Address

127.0.0.1:8080 (talkdemo)

Tracing

Span

Sampling

Context

Inject/Extract

```
tracer := global.Tracer( name: "TALKDEMO")
spanContext, span := tracer.Start(context.Background(),  spanName: "hello")
span.SetStatus(codes.OK, "success")
span.End()
```


```
{
    "SpanContext": {
        "TraceID": "0be39eade3117ccc7f5faff0ffa99fd1",
        "SpanID": "3b98a009981fb833",
        "TraceFlags": 1
    },
    "ParentSpanID": "0000000000000000",
    "SpanKind": 1,
    "Name": "hello",
    "StartTime": "2020-07-20T22:42:45.525801+03:00",
    "EndTime": "2020-07-20T22:42:45.525809081+03:00",
    "Attributes": null,
    "MessageEvents": null,
    "Links": null,
    "StatusCode": 0,
    "StatusMessage": "success",
    "HasRemoteParent": false,
    "DroppedAttributeCount": 0,
    "DroppedMessageEventCount": 0,
    "DroppedLinkCount": 0,
    "ChildSpanCount": 0,
    "Resource": null,
    "InstrumentationLibrary": {
        "Name": "TALKDEMO",
        "Version": ""
    }
}
```

Span Fields

We can add additional informations to our spans.

- Tags
- Logs
- Attributes
- Events
- Status

Tracing

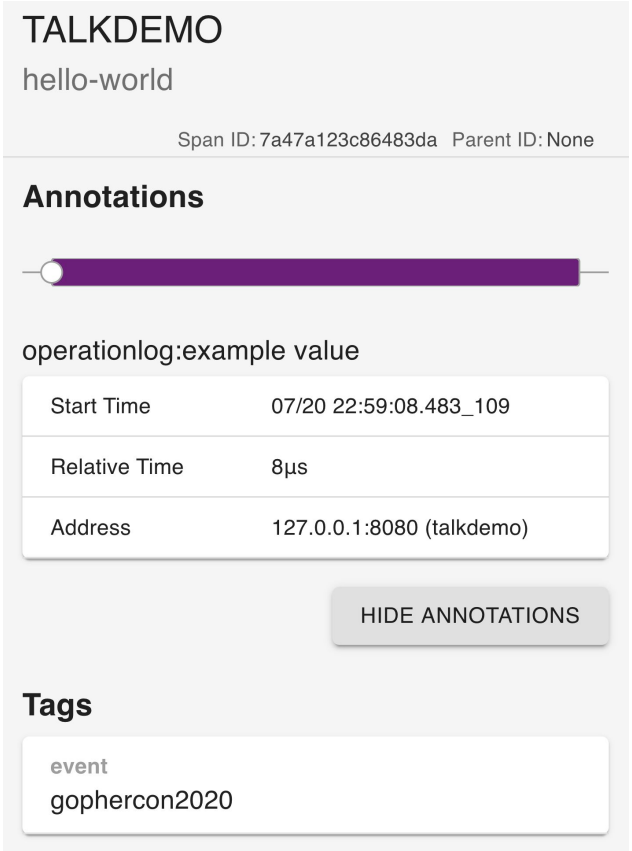
Span

Sampling

Context

Inject/Extract

Tags, Logs, Attributes,Events



Tracing

Span

Sampling

Context

Inject/Extract

Tags, Logs, Attributes,Events

```
"Attributes": [
    {
        "Key": "event",
        "Value": {
            "Type": "STRING",
            "Value": "gophercon2020"
        }
    }
],
"MessageEvents": [
    {
        "Name": "myEvent",
        "Attributes": [
            {
                "Key": "name",
                "Value": {
                    "Type": "STRING",
                    "Value": "event name"
                }
            }
        ],
        "Time": "2020-07-20T23:03:40.857683+03:00"
    }
],
```

span.SetAttribute(**"event"**, **"gophercon2020"**)
span.AddEvent(spanContext, **"myEvent"**, kv.String(**"name"**, **"event name"**))

Tracing

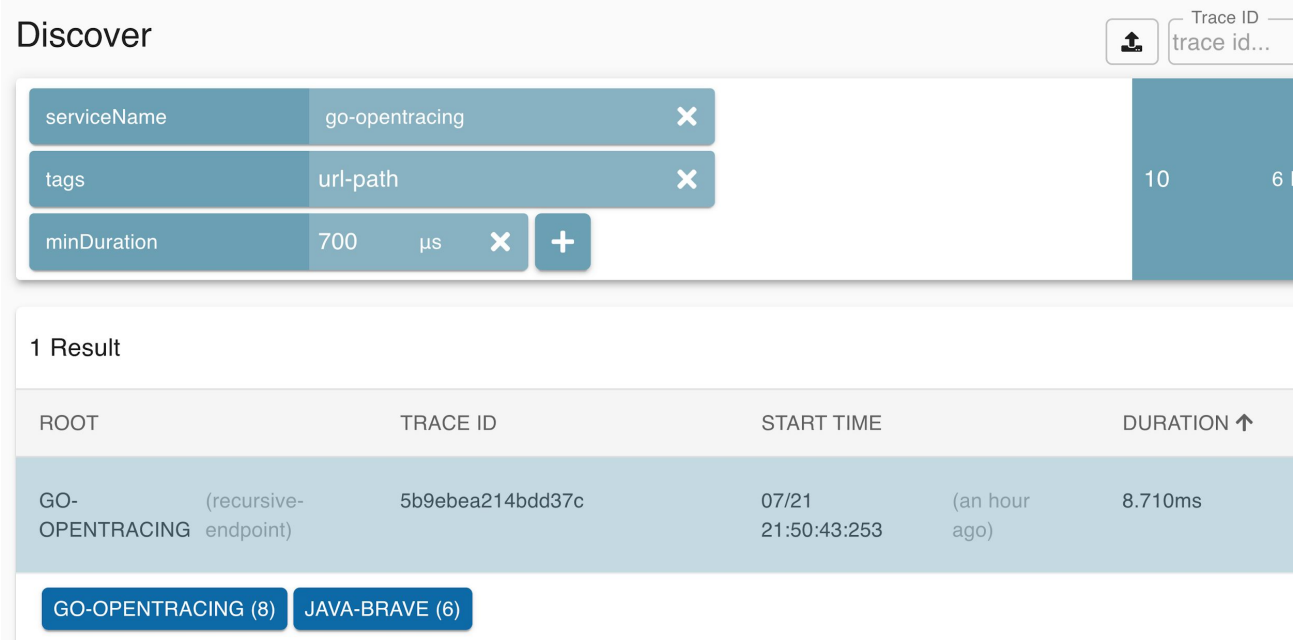
Span

Sampling

Context

Inject/Extract

Tags, Logs, Attributes,Events



Sampling

We can define our sampling options for :

- Sample all requests
- Probabilistic
- Constant Rate (n trace per seconds)
- NoOp

Tracing

Span

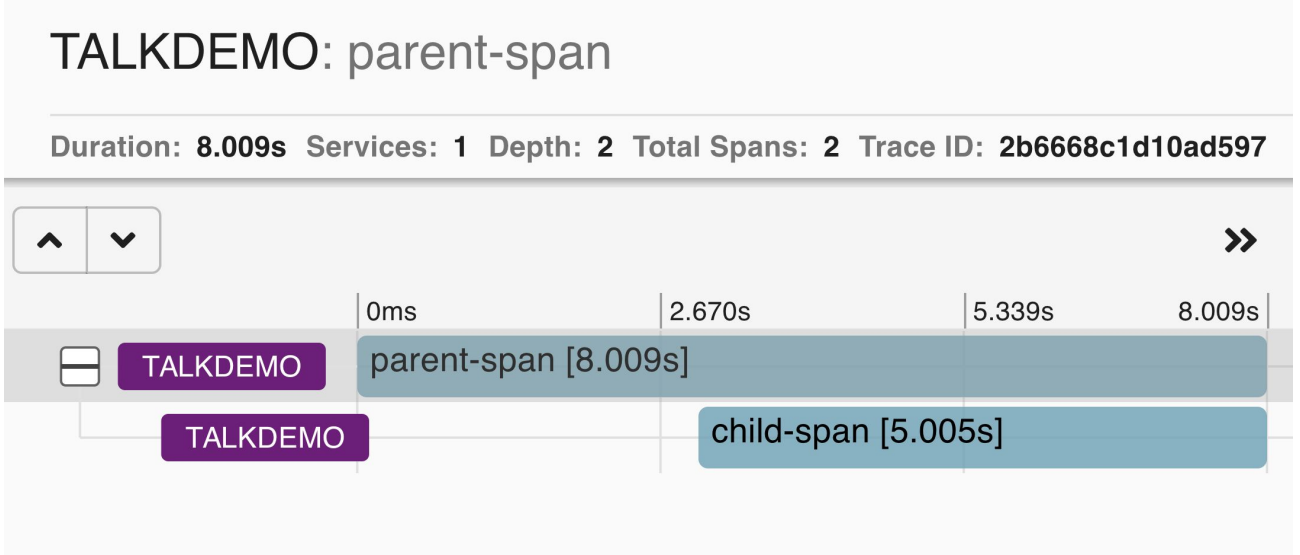
Sampling

Context

Inject/Extract

Span Contexts

Span Contexts helps us move spans across work units. It can be between same class or packages or different services



Inject / Extract

We can use inject extract methods to add headers to our spans.

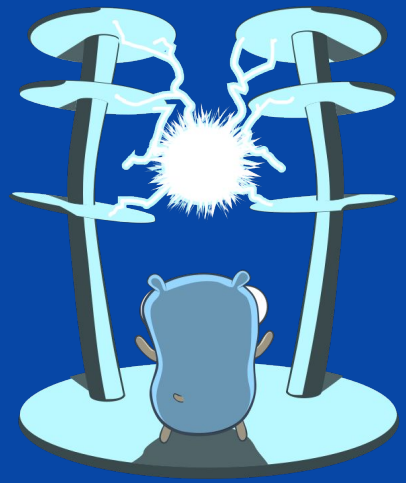
This way we can send our root and child span identifier to the next service.

That service does not need to use the same SDK or the same language at all !!

How to collect traces and link the spans between multiple services ?

Distributed Tracing

Too many services



Distributed Tracing

Multiple Services

Makes it easier to identify the problem across multiple services

Different Languages

These services can be written in different languages.

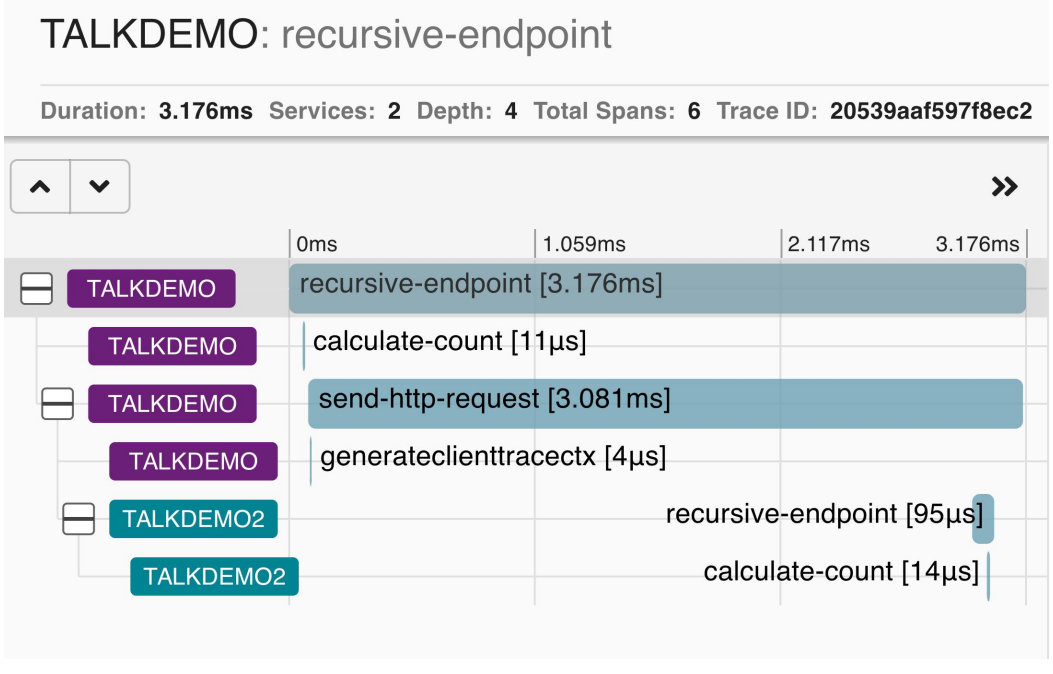
Inject / Extract

In order to move spans across services, we need to use the same headers, etc.

Distributed Tracing

Multiple Services

Makes it easier to identify the problem across multiple services



Distributed Tracing

Multiple Services

Makes it easier to identify the problem across multiple services

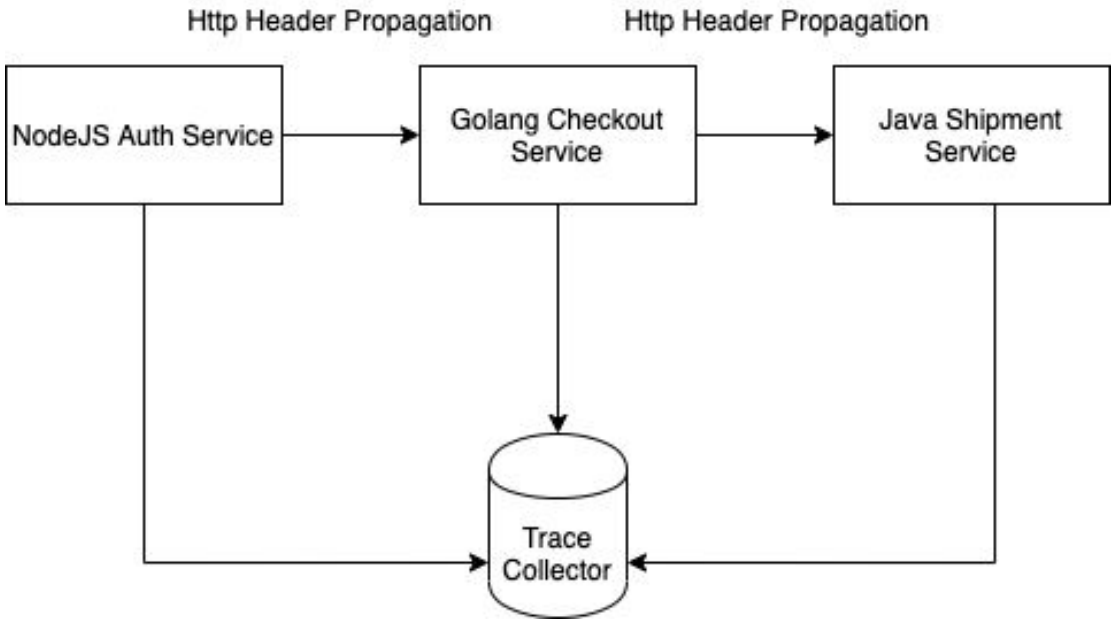
Different Languages

These services can be written in different languages.

Inject/Extract

In order to move spans across services, we need to use the same headers, etc.

Distributed Tracing



https://github.com/GoogleCloudPlatform/microservices-demo - GCP Microservices Demo

Distributed Tracing

Multiple Services

Makes it easier to identify the problem across multiple services

Different Languages

These services can be written in different languages.

Inject/Extract

In order to move spans across services, we need to use the same headers, etc.

Inject with Headers

OpenTracing with HttpHeadersCarrier

```
req = req.WithContext(generateClientTraceCtx(spanCtx, span))
carrier := opentracing.HTTPHeadersCarrier(req.Header)
err = span.Tracer().Inject(span.Context(), opentracing.HTTPHeaders, carrier)
if err != nil {
    return err
}
resp, err := http.DefaultClient.Do(req)
```

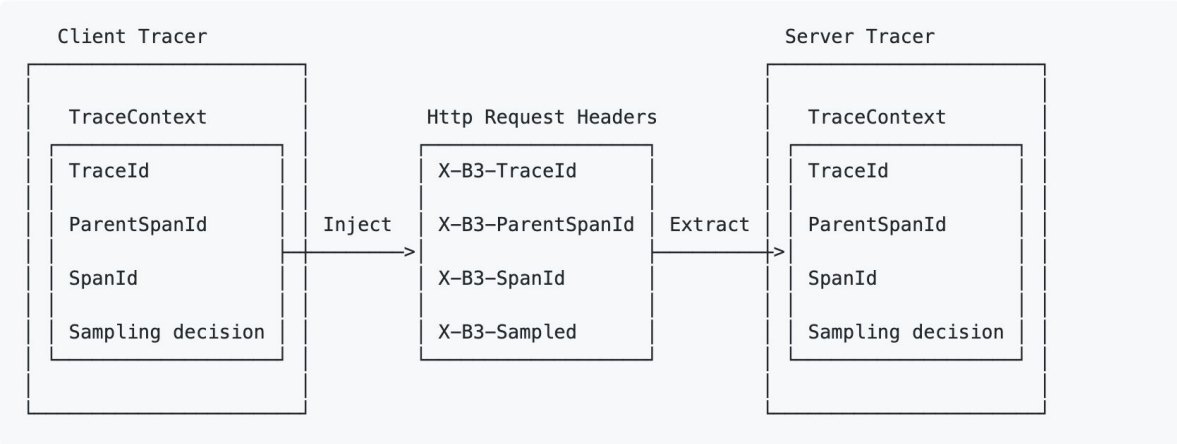
Extract from Headers

```
func (c *ctxUtil) myAwesomeEndpoint(w http.ResponseWriter, r *http.Request) {
    carrier := opentracing.HTTPHeadersCarrier(r.Header)
    sc, err := opentracing.GlobalTracer().Extract(opentracing.HTTPHeaders, carrier)
    if err != nil {
        processErr(err)
    }
    span, ctx := opentracing.StartSpanFromContext(c.Context, operationName: "my-awesome-endpoint", opentracing.ChildOf(sc))
    defer span.Finish()
    count, err := calculateCount(ctx, mux.Vars(r)["count"])
    span.SetTag( key: "url-path", r.URL.Path)
```

Different Trace Propagations

B3 Propagation

x-b3* headers will used to send span identifer



https://github.com/openzipkin/b3-propagation

Different Trace Propagations

W3C Trace Context

- Why
  - Shared Identifiers
  - Multi vendors
- Adopters
  - Elasticsearch
  - NewRelic
  - OpenTelemetry
  - Jaeger

    ...

https://www.w3.org/TR/trace-context/

Before

Hard to identify bottlenecks, quick insight about production when incidents happen

After

Easily find out the problem, we can stop incidents with correct alarms using the traces

Open Telemetry

OpenTelemetry

CNCF

Open Telemetry is a CNCF sandbox project.

OpenTracing + OpenCensus

Merging Open Tracing and Open Census projects.

Currently Beta

Expecting to GA in 2020

OpenTelemetry

- Metrics
- Tracing
- Logging - soon -

Let’s look OpenTelemetry a little bit closer

Open Telemetry

SDK's

Collector

SDK's - Libraries

- Go
- Java
- NodeJs
- Python
- ...

Open Telemetry

SDK's

Collector

Registry

We can use OpenTelemetry Registry to search libraries, utilities.

https://opentelemetry.io/registry/

Open Telemetry

SDK's

Collector

Metrics

OpenTelemetry has a metric instrumentation.

- Counter
- Measure
- Observer

Open Telemetry

SDK’s

Collector

Collector

- Vendor-Agnostic
- Extensible
- One collector for different tools

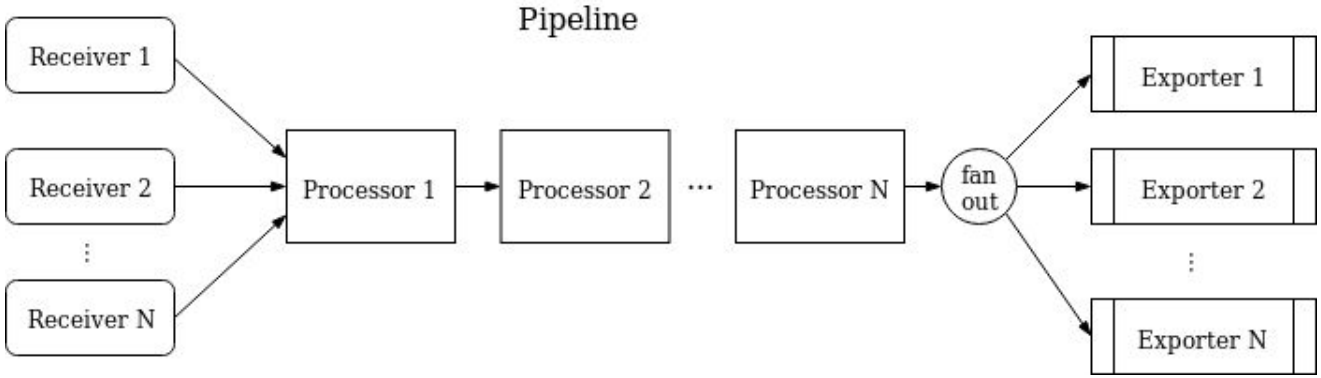
Open Telemetry

SDK's

Collector

Collector

- Pipeline
  Defines how to receive, process and export the data.
  https://github.com/open-telemetry/opentelemetry-collector/blob/master/docs/design.md



Open Telemetry

SDK's

Collector

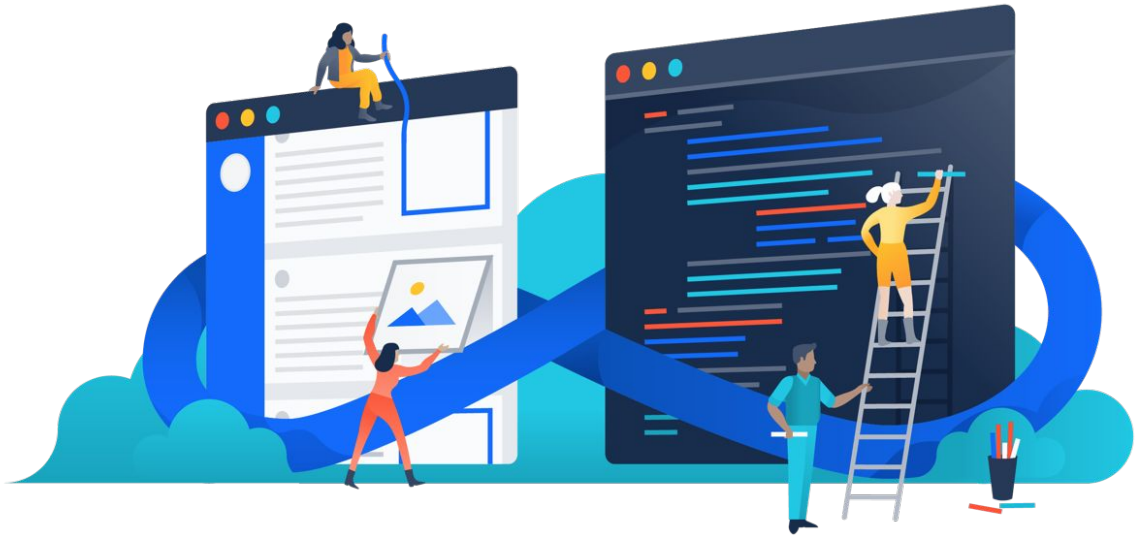
Collector

- Configurations
- https://github.com/open-telemetry/opentelemetry-collector/blob/master/docs/

```
receivers:
  opencensus:
    endpoint: "0.0.0.0:55678"

service:
  pipelines:
    traces:  # a pipeline of “traces” type
      receivers: [opencensus]
      processors: [tags, tail_sampling, batch, queued_retry]
      exporters: [jaeger]
    traces/2:  # another pipeline of “traces” type
      receivers: [opencensus]
      processors: [batch]
      exporters: [opencensus]
```

Let’s sum up



Thank you!

Fahri YARDIMCI | Site Reliability Engineer, Atlassian | @FahriYardimci