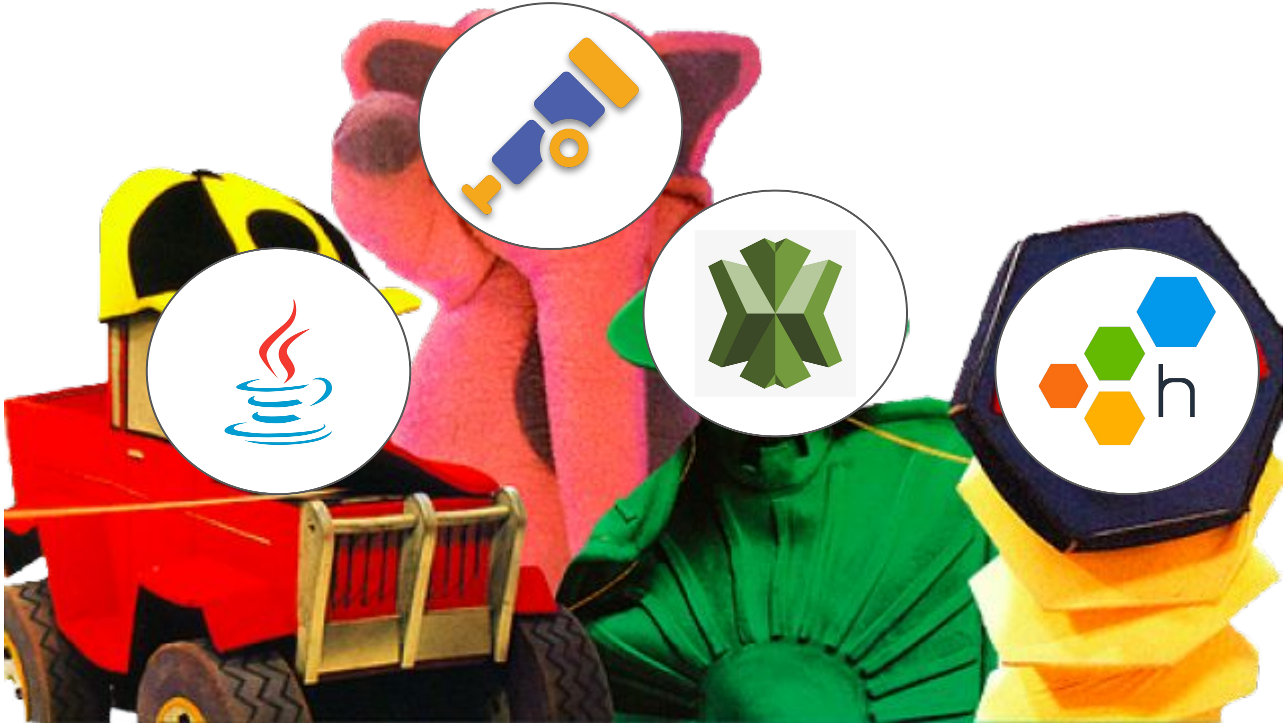
OpenTelemetry & Friends



Who am I?

Name: Gerard Gigliotti

I'm a full stack engineer at **Ippon Australia**.

What is Telemetry?

TRACES

A trace is a request documented through one or more components, linked together within a common ID.

Example

Microservice A talks to Microservice B over a REST endpoint; Microservice A provides trace_id data within a header in the call.

METRICS

A metric is a measurement about a service, captured as the service is running.

Example

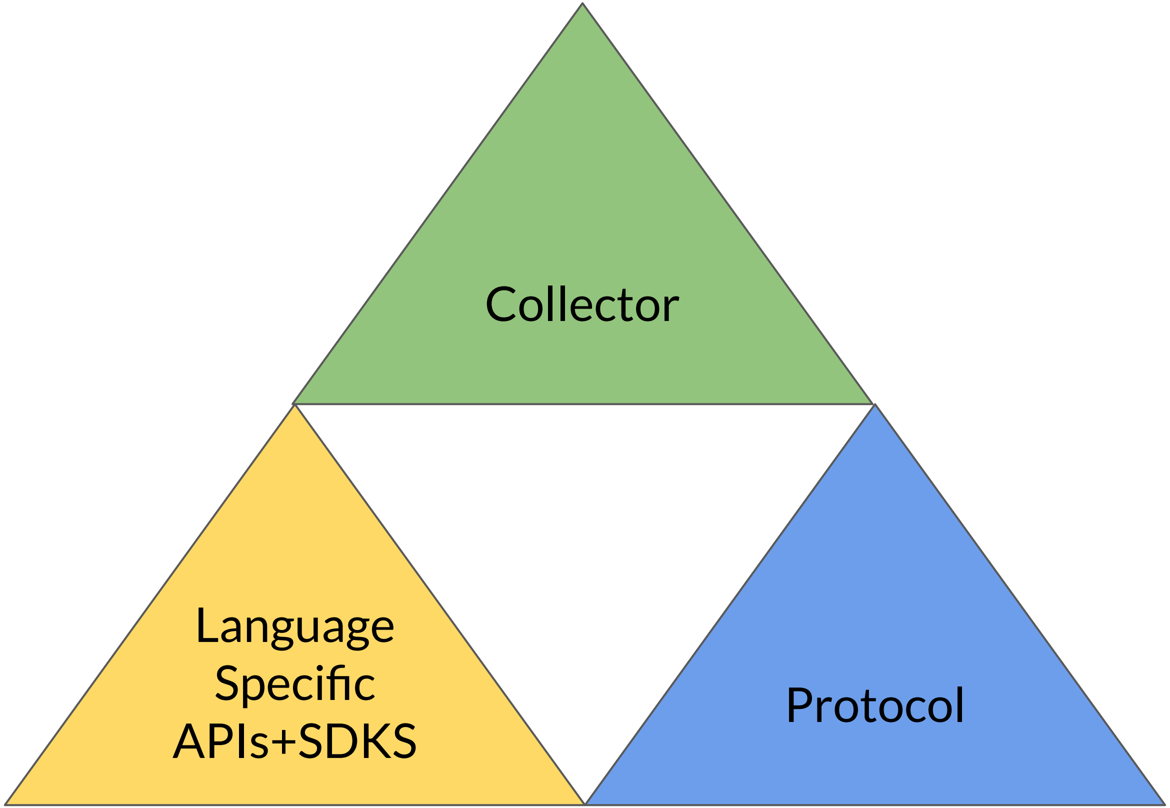
Microservice B records the number of requests made to its endpoint, using a counter.

LOGS

A lo estamp
r tr
(rec
unstru ata.

Mi s
o eques

What is OpenTelemetry?

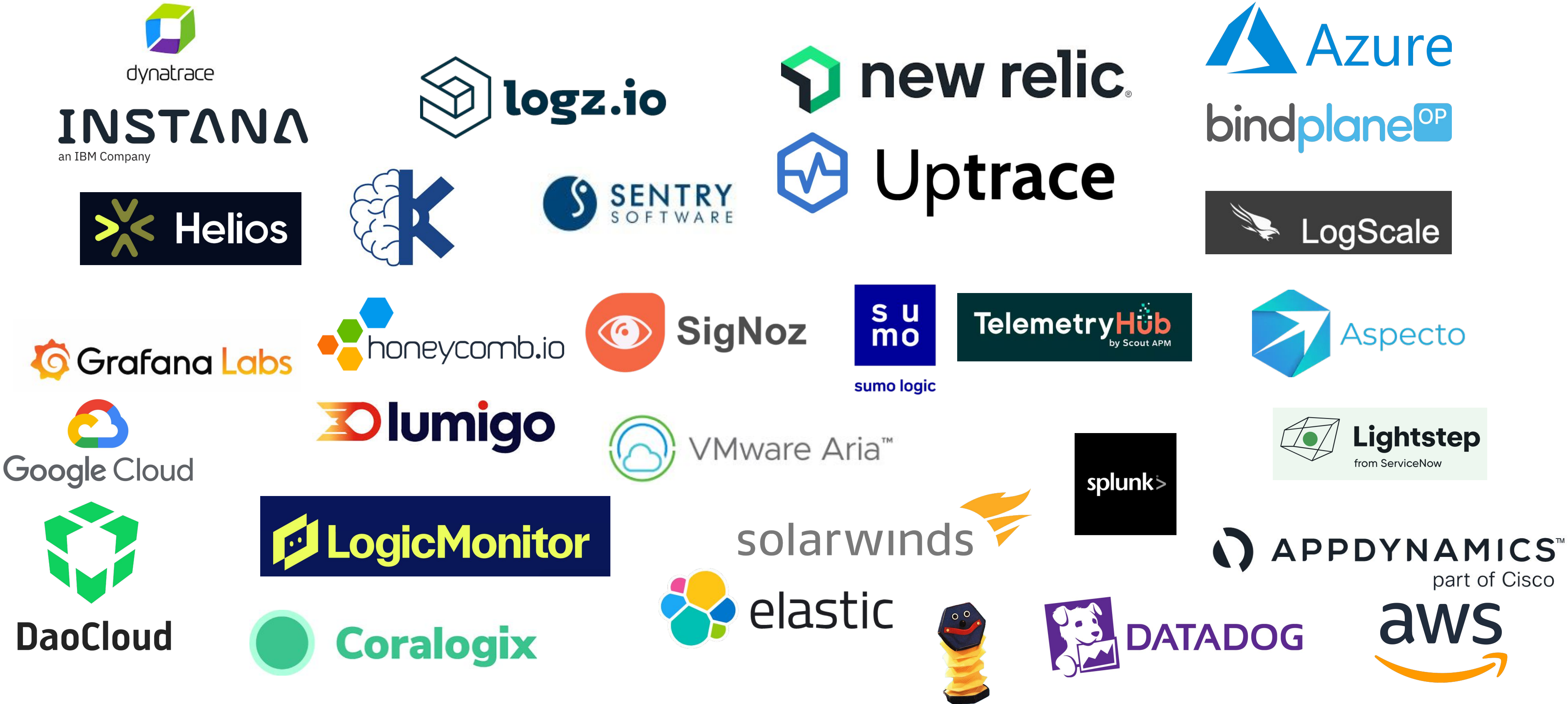


Collector

Language Specific APIs+SDKS

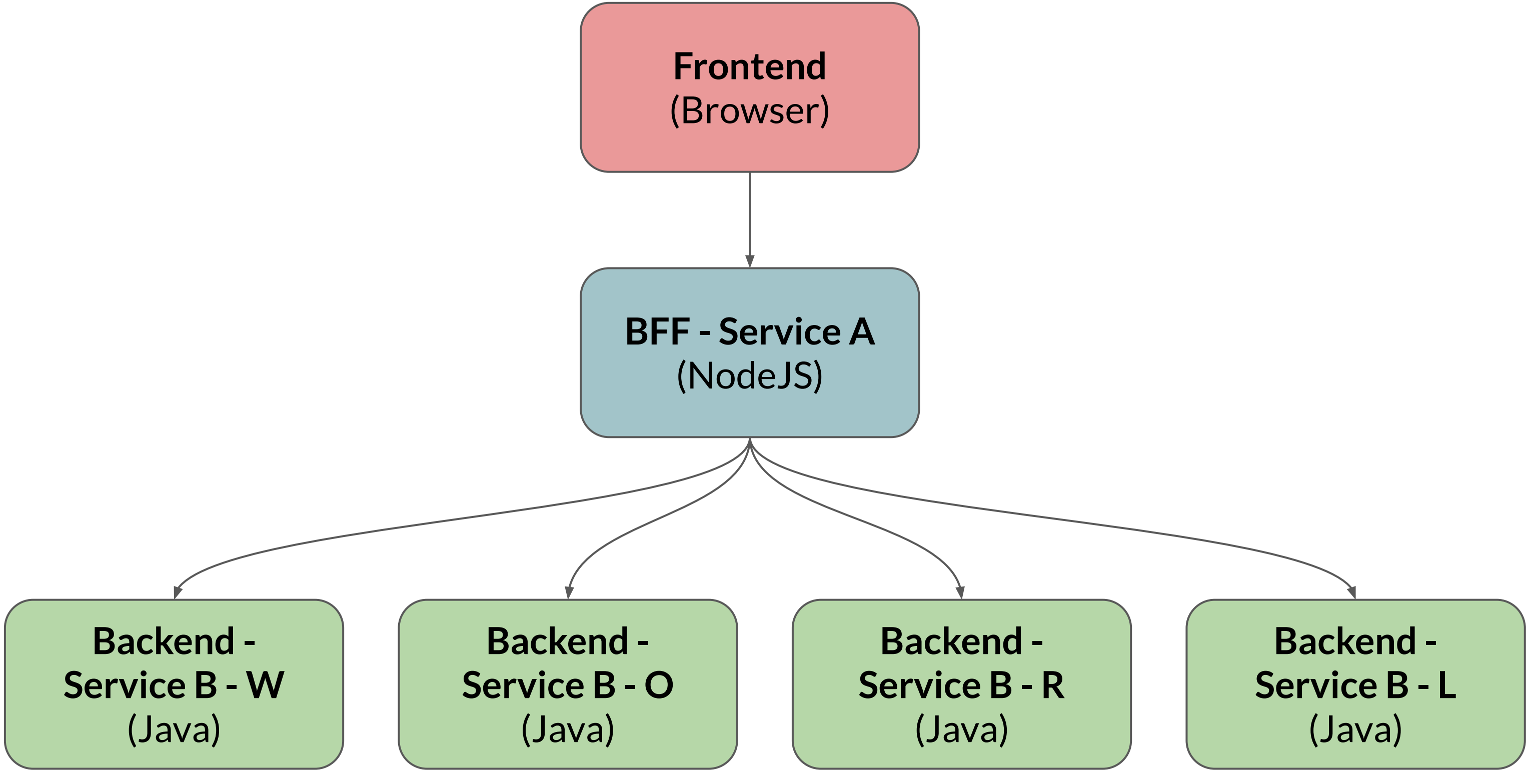
Protocol

Who are its friends?



Integration Extraveganza

Sample “Hello World” Stack



Frontend
(Browser)

BFF - Service A
(NodeJS)

Backend - Service B - W
(Java)

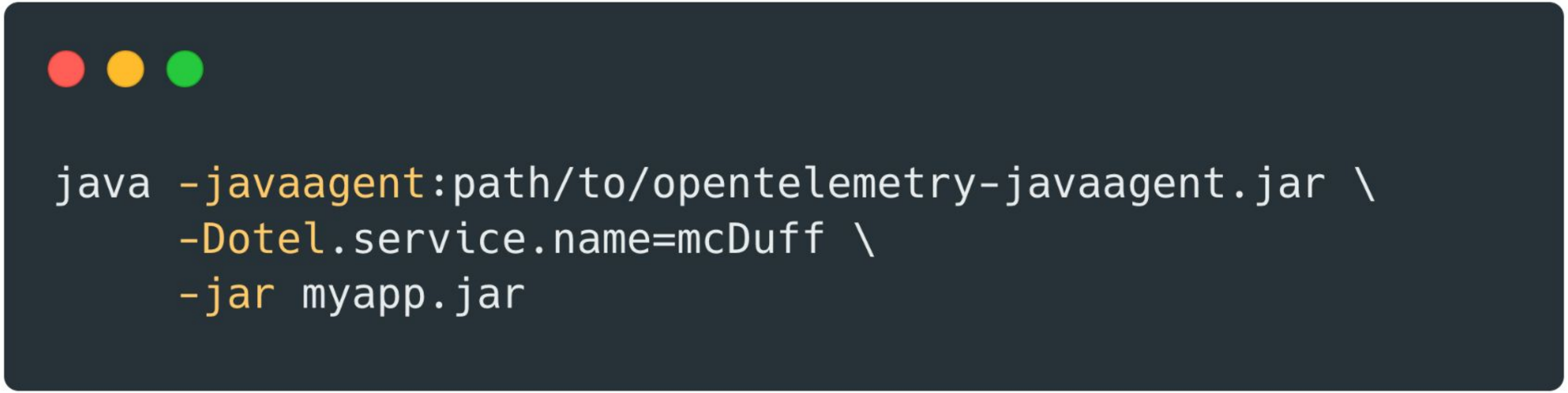
Backend - Service B - O
(Java)

Backend - Service B - R
(Java)

Backend - Service B - L
(Java)

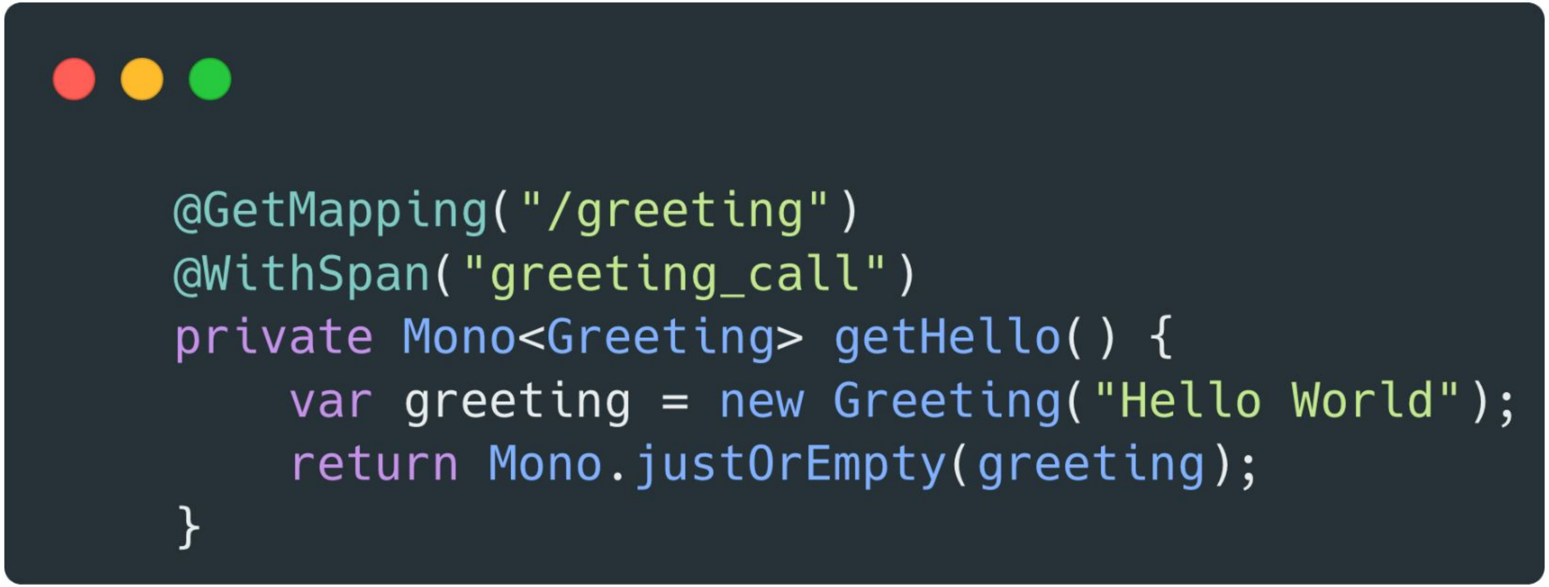
Java Agent (~~for the Lazy~~ Practical)

Injectable Bytecode Agent, which you supply as an argument at startup.



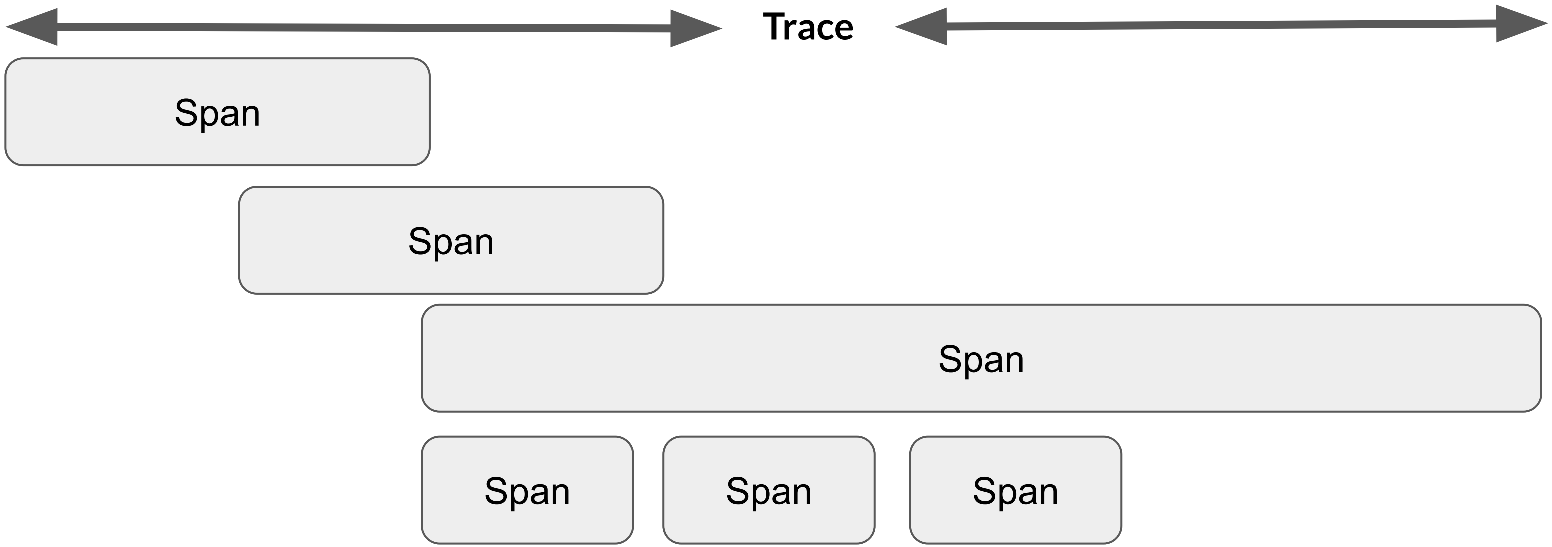
```
java -javaagent:path/to/opentelemetry-javaagent.jar \
     -Dotel.service.name=mcDuff \
     -jar myapp.jar
```

Java API - Spans With Annotations

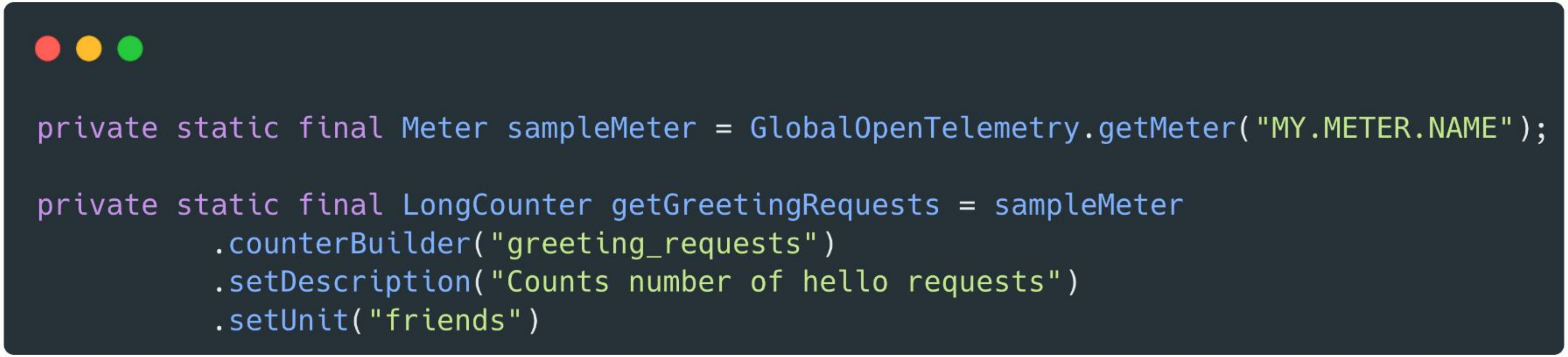


```
@GetMapping("/greeting")
@WithSpan("greeting_call")
private Mono<Greeting> getHello() {
    var greeting = new Greeting("Hello World");
    return Mono.justOrEmpty(greeting);
}
```

What’s a Span?

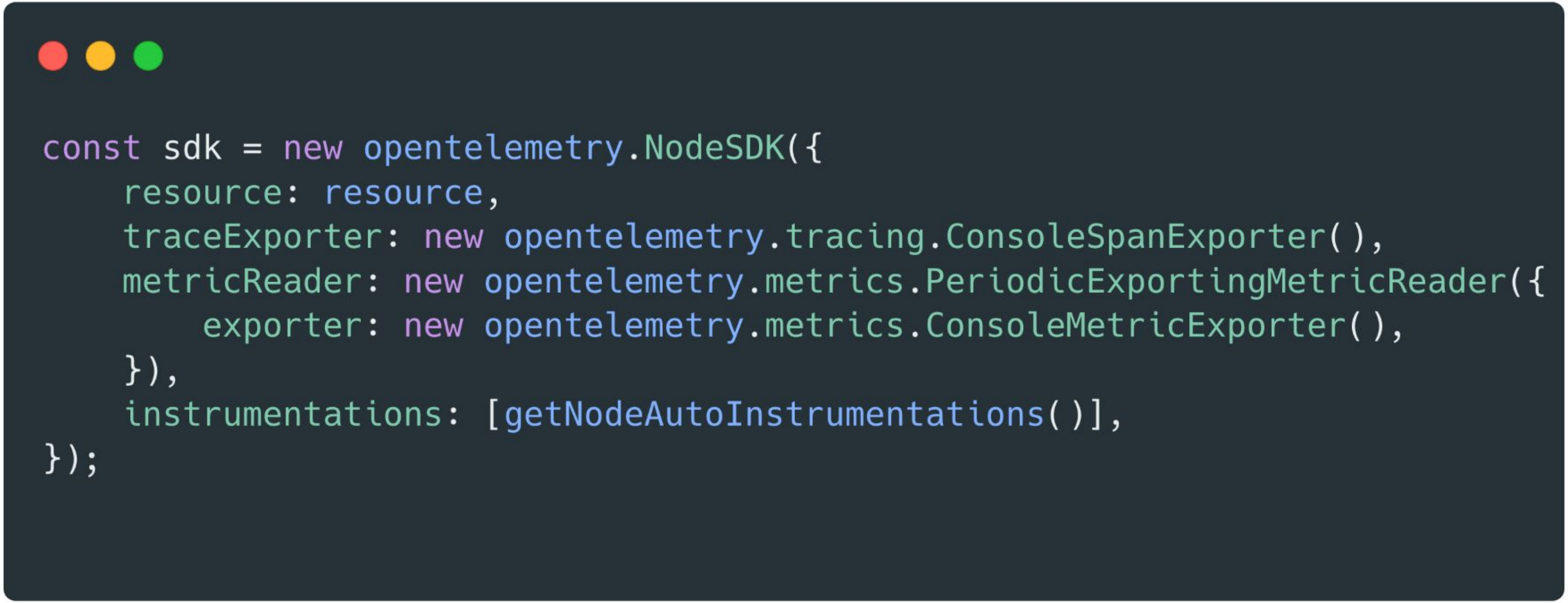


Java API - Metrics



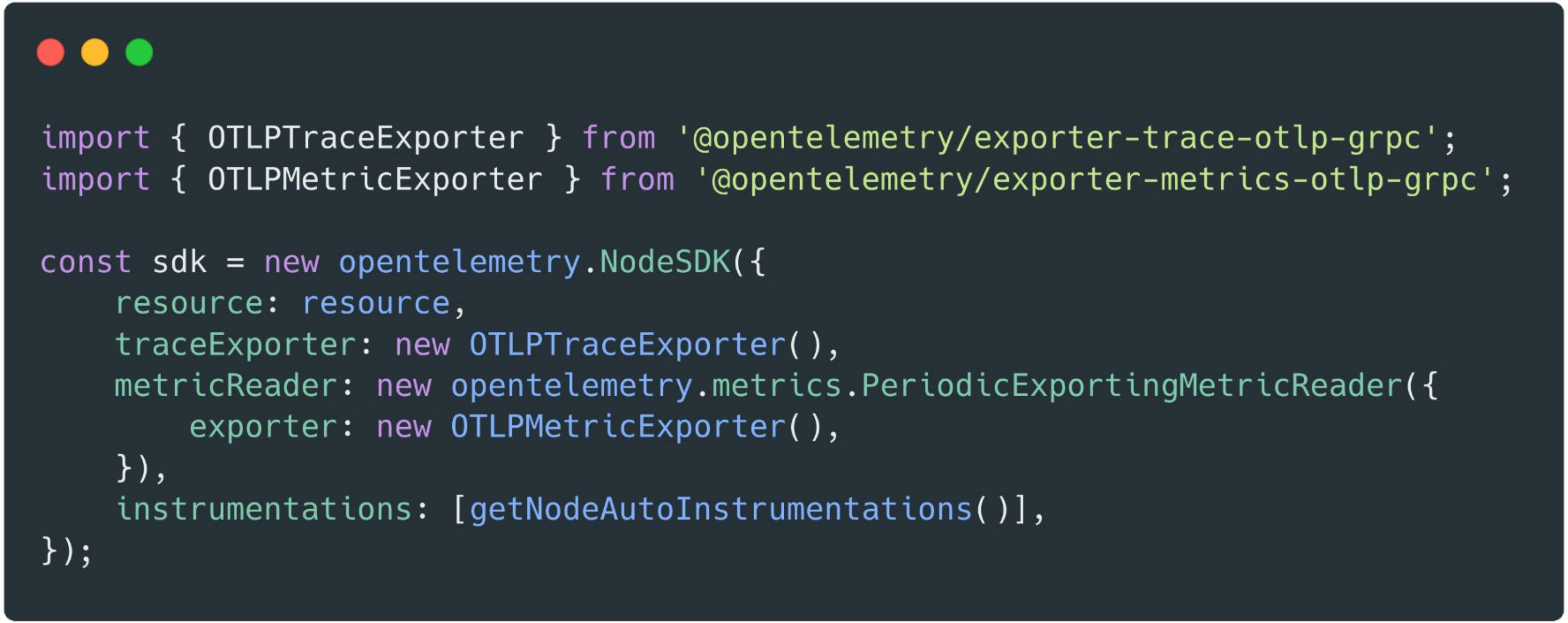
```java
private static final Meter sampleMeter = GlobalOpenTelemetry.getMeter("MY.METER.NAME");

private static final LongCounter getGreetingRequests = sampleMeter
          .counterBuilder("greeting_requests")
          .setDescription("Counts number of hello requests")
          .setUnit("friends")
```

NodeJS - Trace & Console Exporting



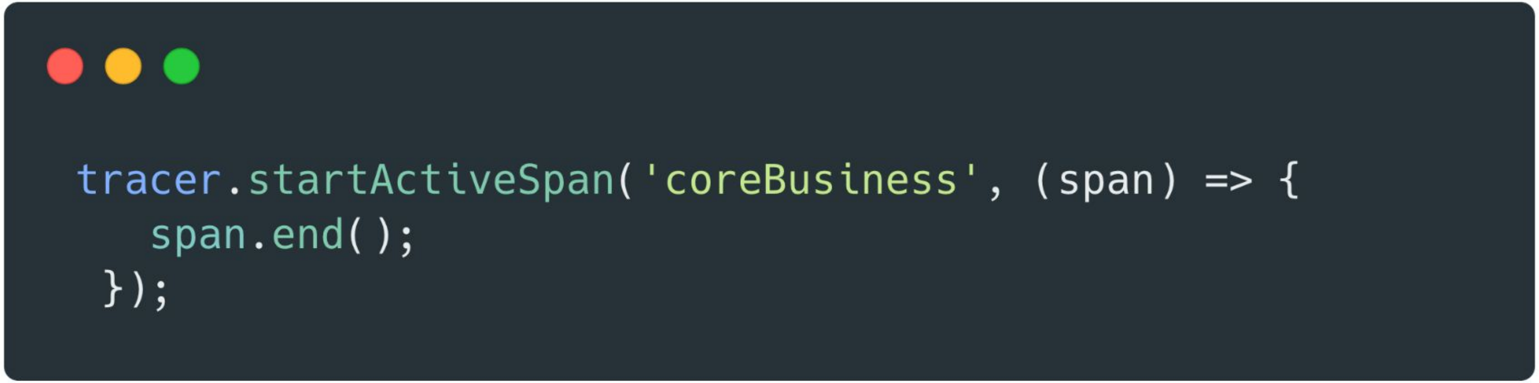
```
const sdk = new opentelemetry.NodeSDK({
    resource: resource,
    traceExporter: new opentelemetry.tracing.ConsoleSpanExporter(),
    metricReader: new opentelemetry.metrics.PeriodicExportingMetricReader({
        exporter: new opentelemetry.metrics.ConsoleMetricExporter(),
    }),
    instrumentations: [getNodeAutoInstrumentations()],
});
```

NodeJS - Exporting via GRPC



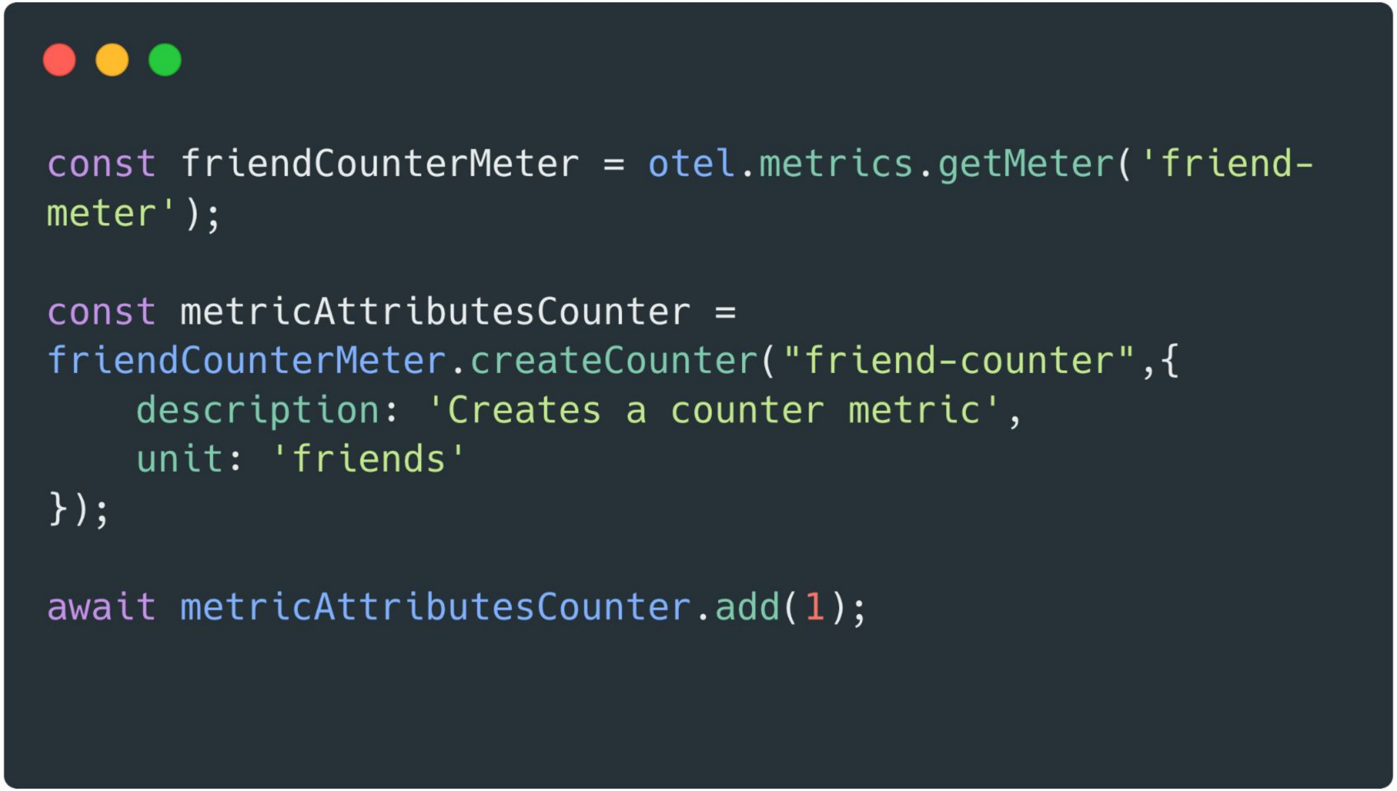
```
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-grpc';
import { OTLPMetricExporter } from '@opentelemetry/exporter-metrics-otlp-grpc';

const sdk = new opentelemetry.NodeSDK({
    resource: resource,
    traceExporter: new OTLPTraceExporter(),
    metricReader: new opentelemetry.metrics.PeriodicExportingMetricReader({
        exporter: new OTLPMetricExporter(),
    }),
    instrumentations: [getNodeAutoInstrumentations()],
});
```

NodeJS - Spans



```
tracer.startActiveSpan('coreBusiness', (span) => {
   span.end();
 });
```

NodeJS - Metrics

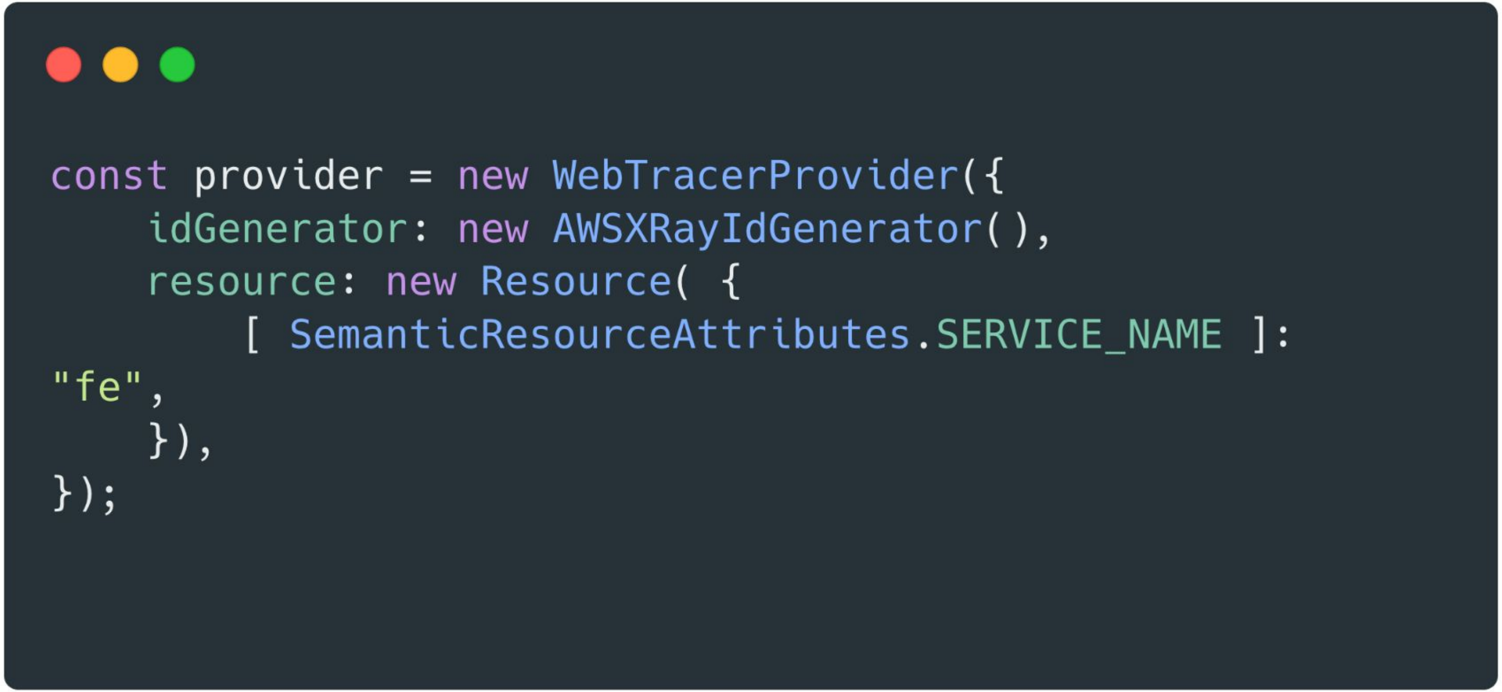


```
const friendCounterMeter = otel.metrics.getMeter('friend-
meter');

const metricAttributesCounter =
friendCounterMeter.createCounter("friend-counter",{
    description: 'Creates a counter metric',
    unit: 'friends'
});

await metricAttributesCounter.add(1);
```

JavaScript Frontend - Caveats Caveats Caveats

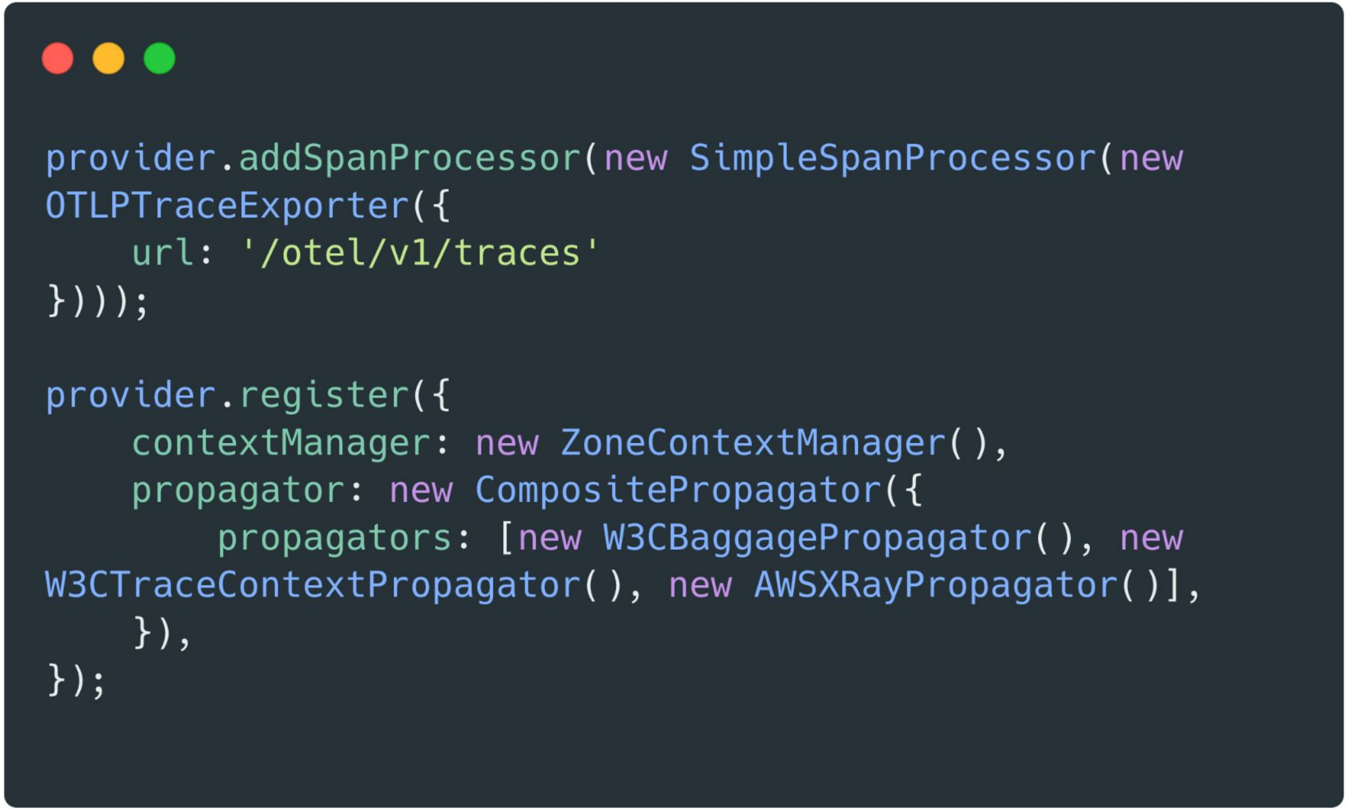
- There is support for running OpenTelemetry via the Frontend.
- However, you need to allow the frontend access to a collector, and they recommend you run it behind a proxy for additional protection.
- Only Otel-over-HTTP is supported, no GRPC.

Frontend



```
const provider = new WebTracerProvider({
    idGenerator: new AWSXRayIdGenerator(),
    resource: new Resource( {
        [ SemanticResourceAttributes.SERVICE_NAME ]:
"fe",
    }),
});
```

Frontend



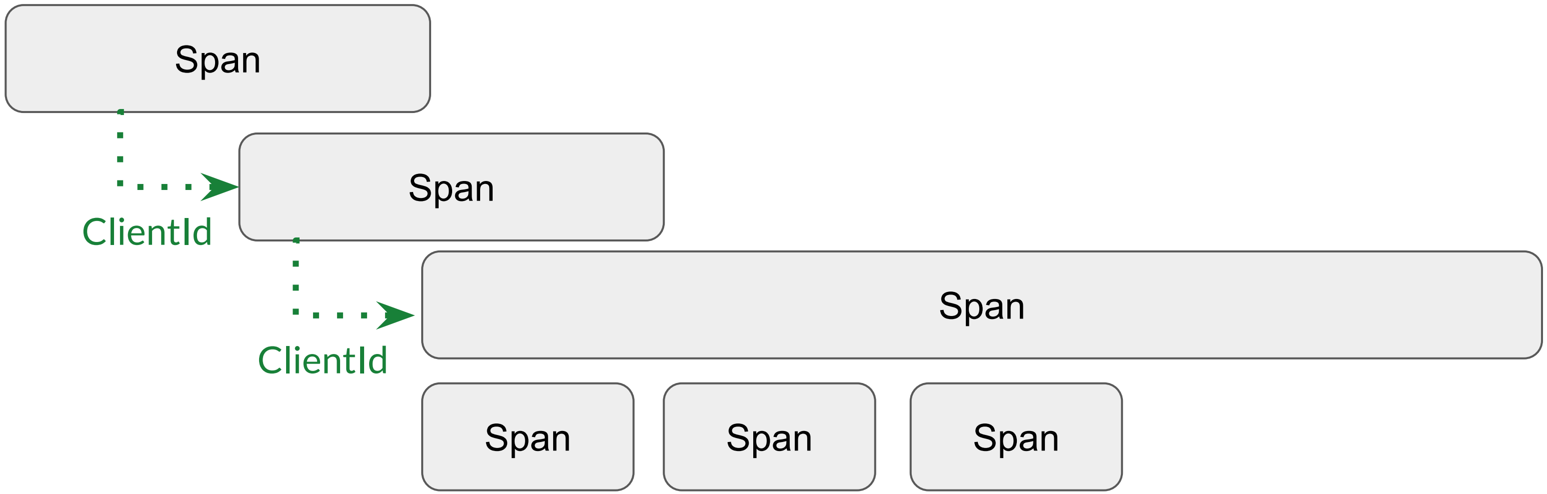
```
provider.addSpanProcessor(new SimpleSpanProcessor(new
OTLPTraceExporter({
    url: '/otel/v1/traces'
})));

provider.register({
    contextManager: new ZoneContextManager(),
    propagator: new CompositePropagator({
        propagators: [new W3CBaggagePropagator(), new
W3CTraceContextPropagator(), new AWSXRayPropagator()],
    }),
});
```

Propagation

How do the traces connect together? Generally via headers (in the case of HTTP). Supports:

- W3C TraceContext (recommended)
- W3C Baggage (recommended)
- B3
- Jaeger
- OT Trace

Emotional Baggage

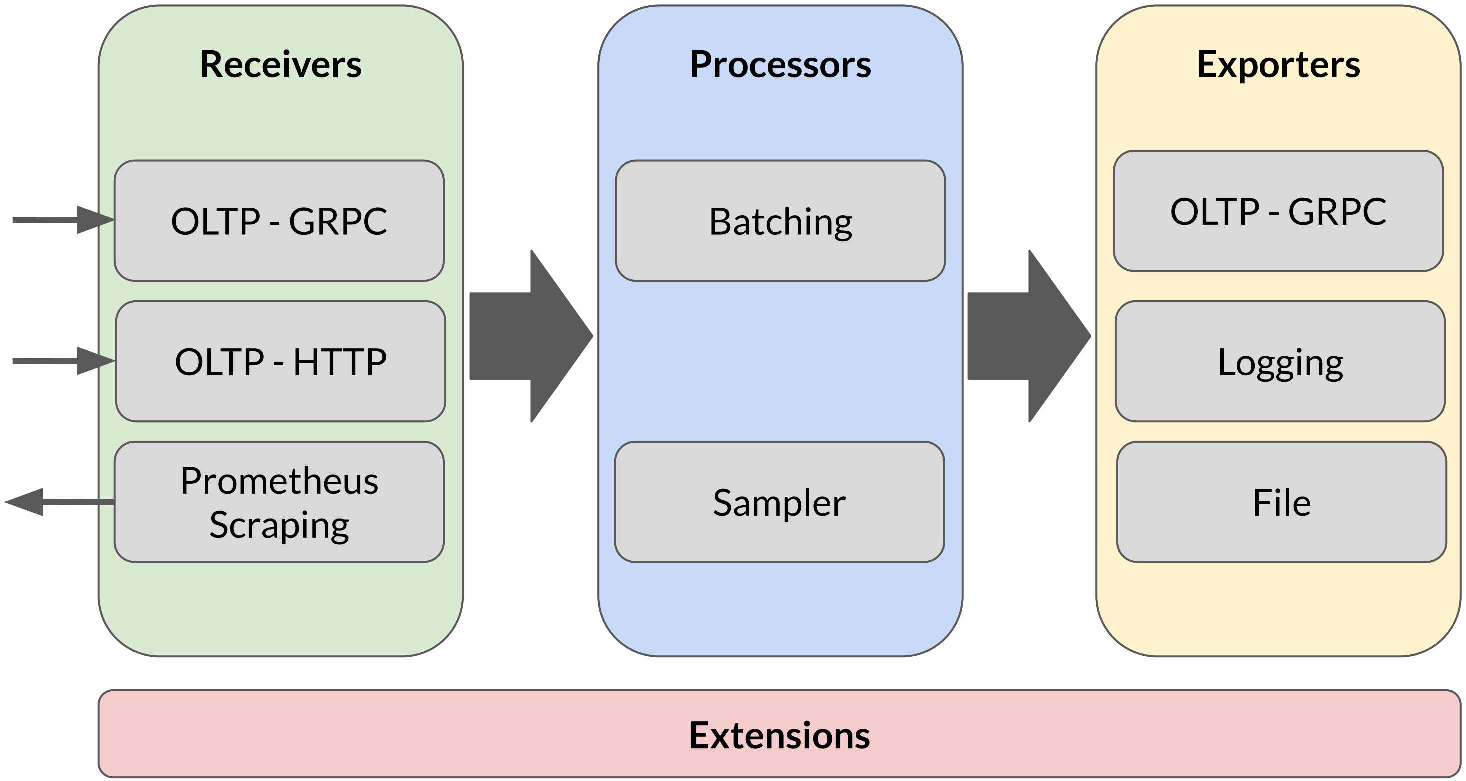


Collector

Collector

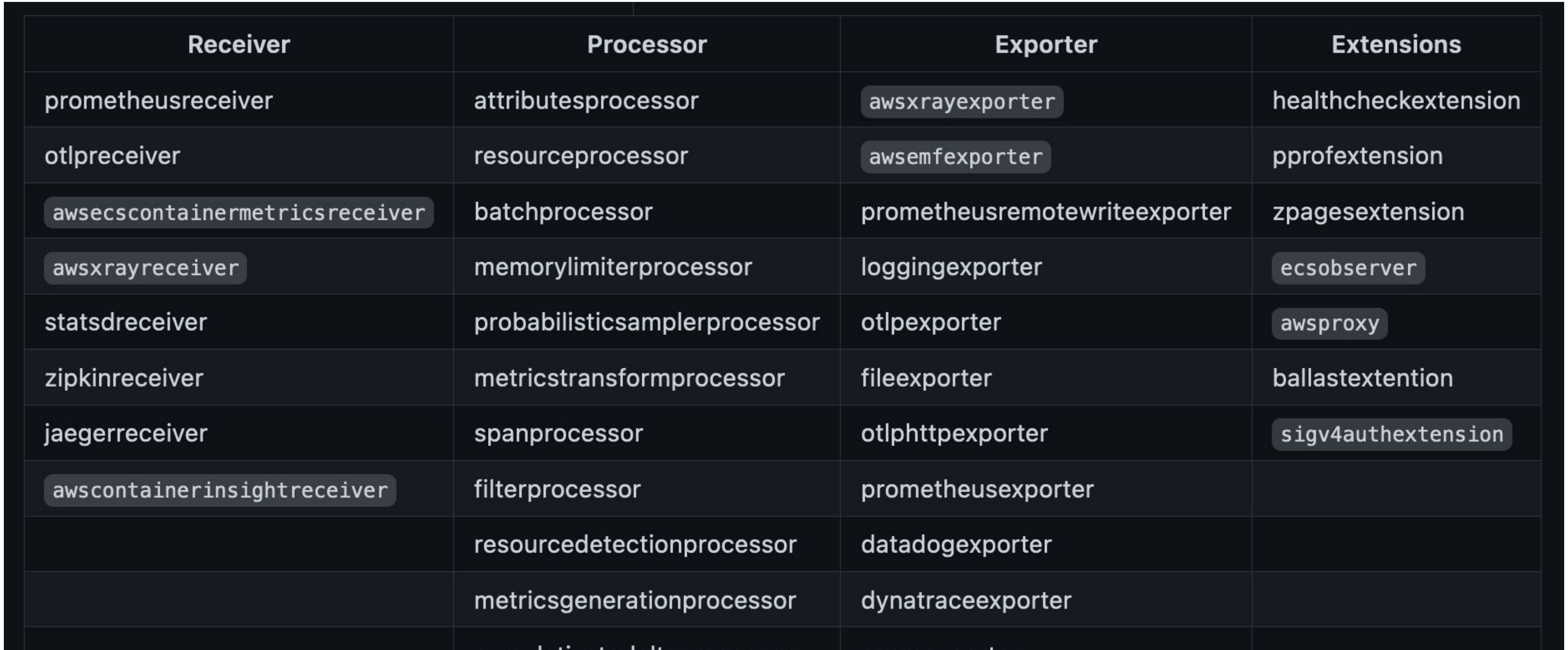
- Small, standalone Go-Application.
- Generally used as a container side-car in Kubernetes/ECS.
- Configured via YAML

Collector


Receivers
OLTP - GRPC
OLTP - HTTP
Prometheus Scraping
Processors
Batching
Sampler
Exporters
OLTP - GRPC
Logging
File
Extensions

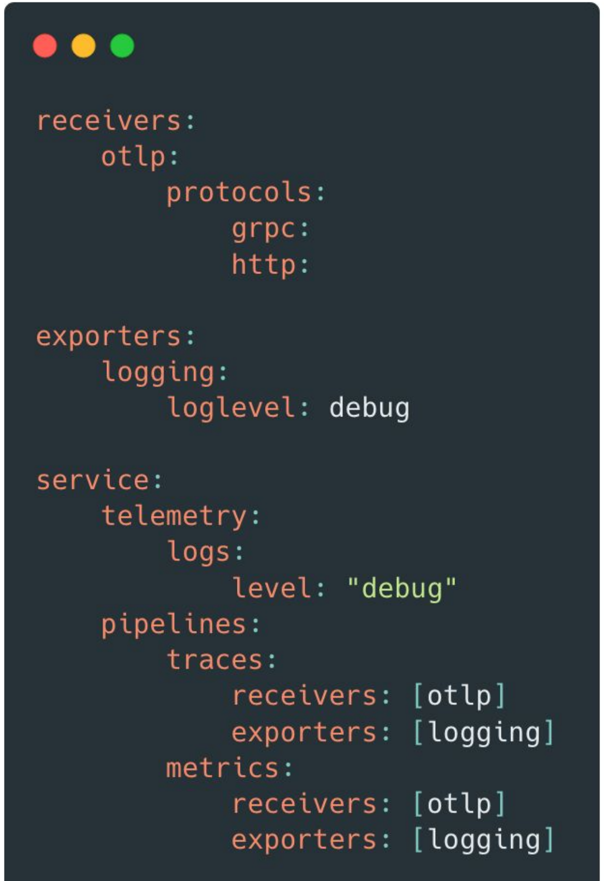
Collector Distributions

- The Standard/Pure OpenTelemetry Distribution
- Vendor Specific Distributions



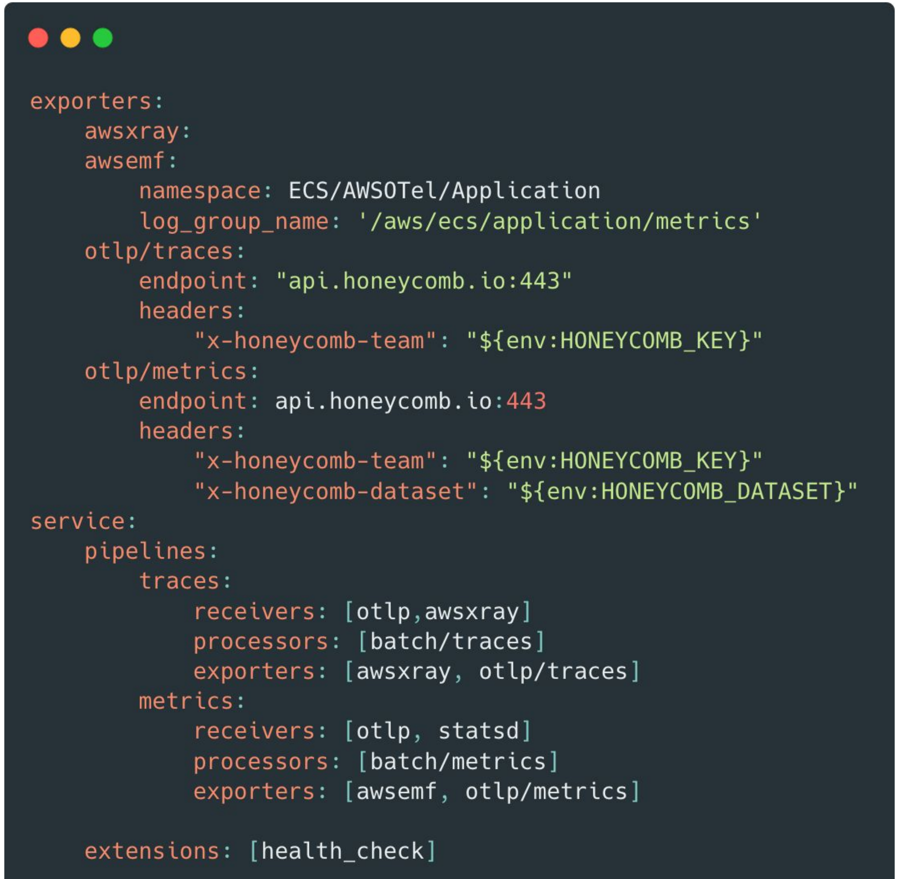
Receiver	Processor	Exporter	Extensions
prometheusreceiver	attributesprocessor	`awsxrayexporter`	healthcheckextension
otlpreceiver	resourceprocessor	`awsemfexporter`	pprofextension
`awsecscontainermetricsreceiver`	batchprocessor	prometheusremotewriteexporter	zpagesextension
`awsxrayreceiver`	memorylimiterprocessor	loggingexporter	`ecsobserver`
statsdreceiver	probabilisticsamplerprocessor	otlpexporter	`awsproxy`
zipkinreceiver	metricstransformprocessor	fileexporter	ballastextention
jaegerreceiver	spanprocessor	otlphttpexporter	`sigv4authextension`
`awscontainerinsightreceiver`	filterprocessor	prometheusexporter	
	resourcedetectionprocessor	datadogexporter	
	metricsgenerationprocessor	dynatraceexporter	

Simple Collector Config



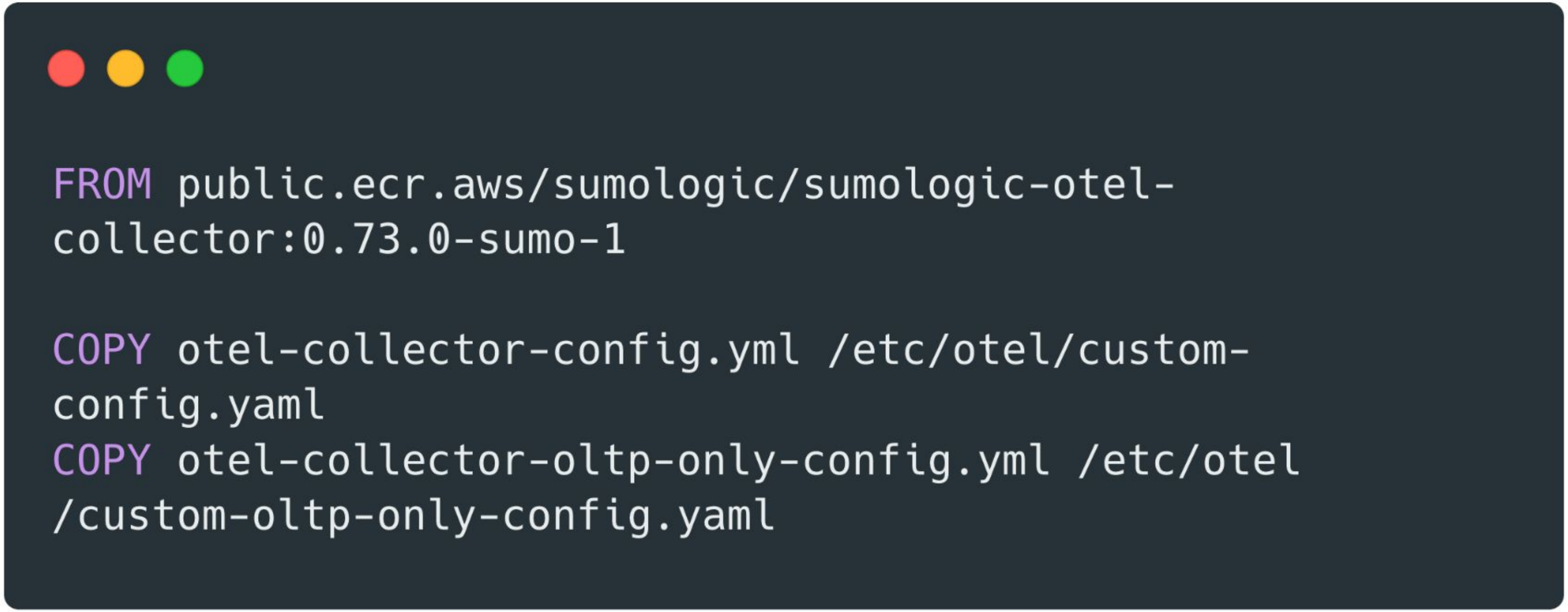
```yaml
receivers:
    otlp:
        protocols:
            grpc:
            http:

exporters:
    logging:
        loglevel: debug

service:
    telemetry:
        logs:
            level: "debug"
    pipelines:
        traces:
            receivers: [otlp]
            exporters: [logging]
        metrics:
            receivers: [otlp]
            exporters: [logging]
```

AWS XRay Collector



```
exporters:
    awsxray:
    awsemf:
        namespace: ECS/AWSOTel/Application
        log_group_name: '/aws/ecs/application/metrics'
    otlp/traces:
        endpoint: "api.honeycomb.io:443"
        headers:
            "x-honeycomb-team": "${env:HONEYCOMB_KEY}"
    otlp/metrics:
        endpoint: api.honeycomb.io:443
        headers:
            "x-honeycomb-team": "${env:HONEYCOMB_KEY}"
            "x-honeycomb-dataset": "${env:HONEYCOMB_DATASET}"
service:
    pipelines:
        traces:
            receivers: [otlp,awsxray]
            processors: [batch/traces]
            exporters: [awsxray, otlp/traces]
        metrics:
            receivers: [otlp, statsd]
            processors: [batch/metrics]
            exporters: [awsemf, otlp/metrics]

    extensions: [health_check]
```

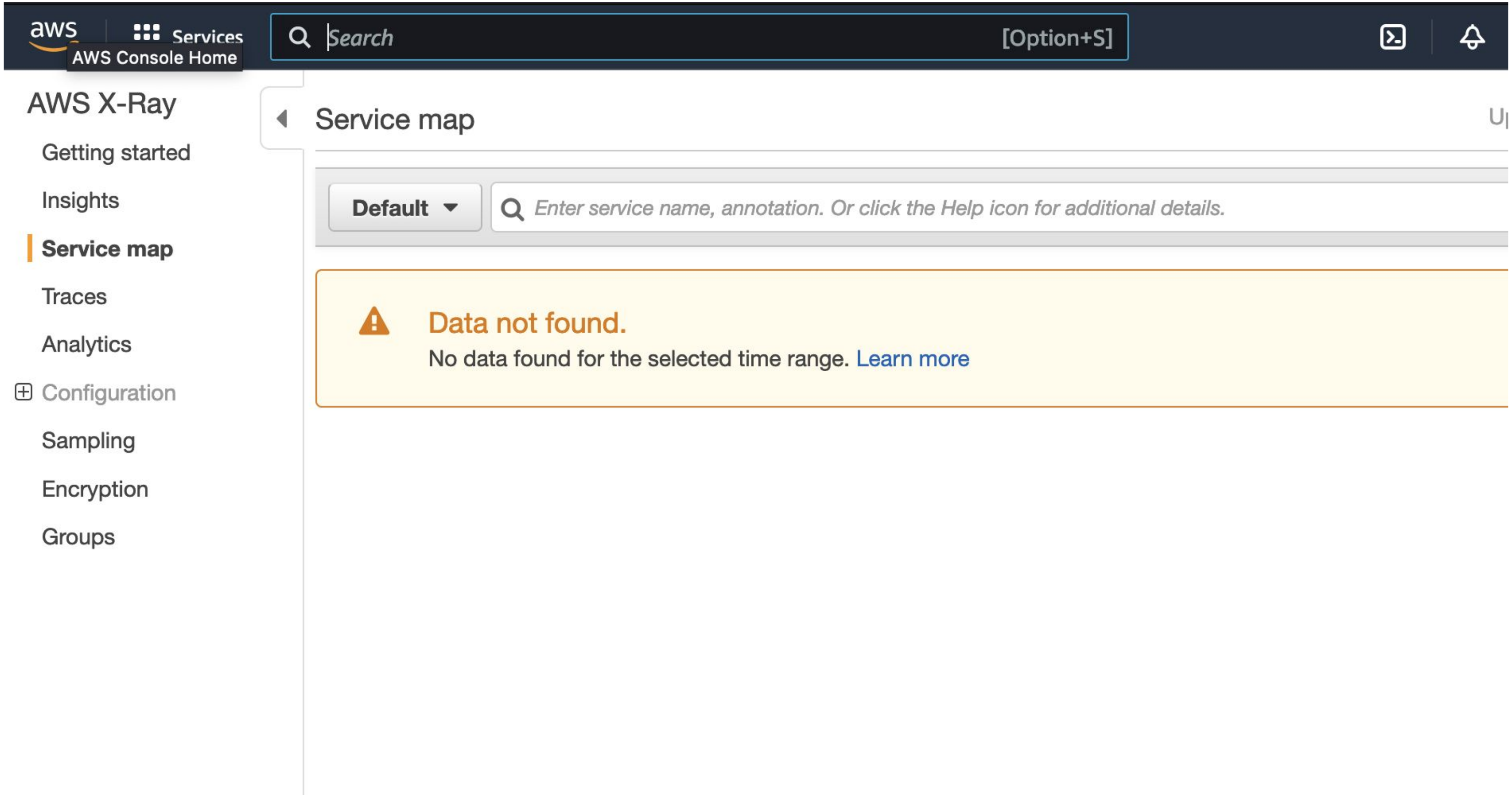
Docker Sidecar



```
FROM public.ecr.aws/sumologic/sumologic-otel-
collector:0.73.0-sumo-1

COPY otel-collector-config.yml /etc/otel/custom-
config.yaml
COPY otel-collector-oltp-only-config.yml /etc/otel
/custom-oltp-only-config.yaml
```

Provider Examples

AWS XRay my Heart

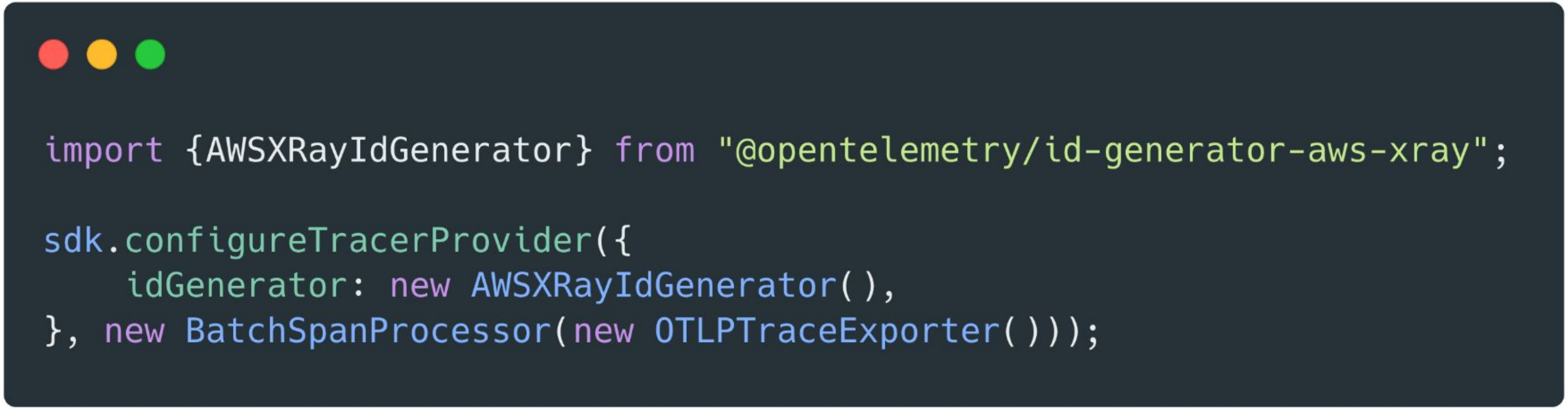


Data not found.

No data found for the selected time range. Learn more

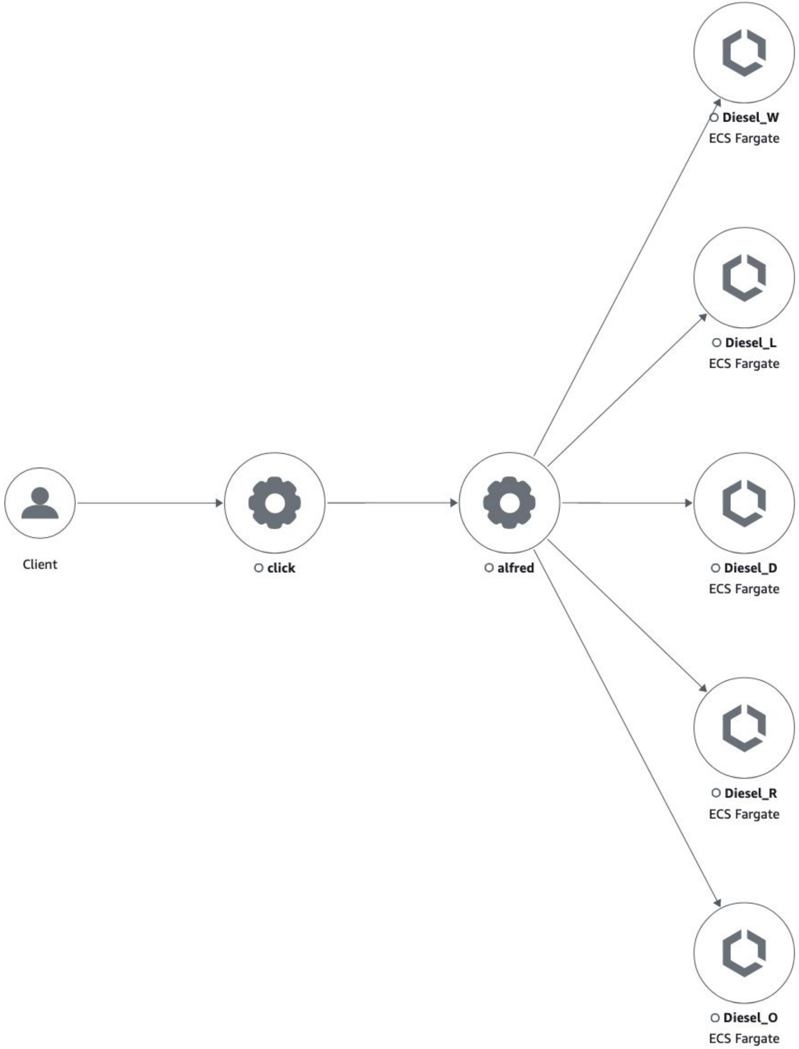
AWS XRay Issues

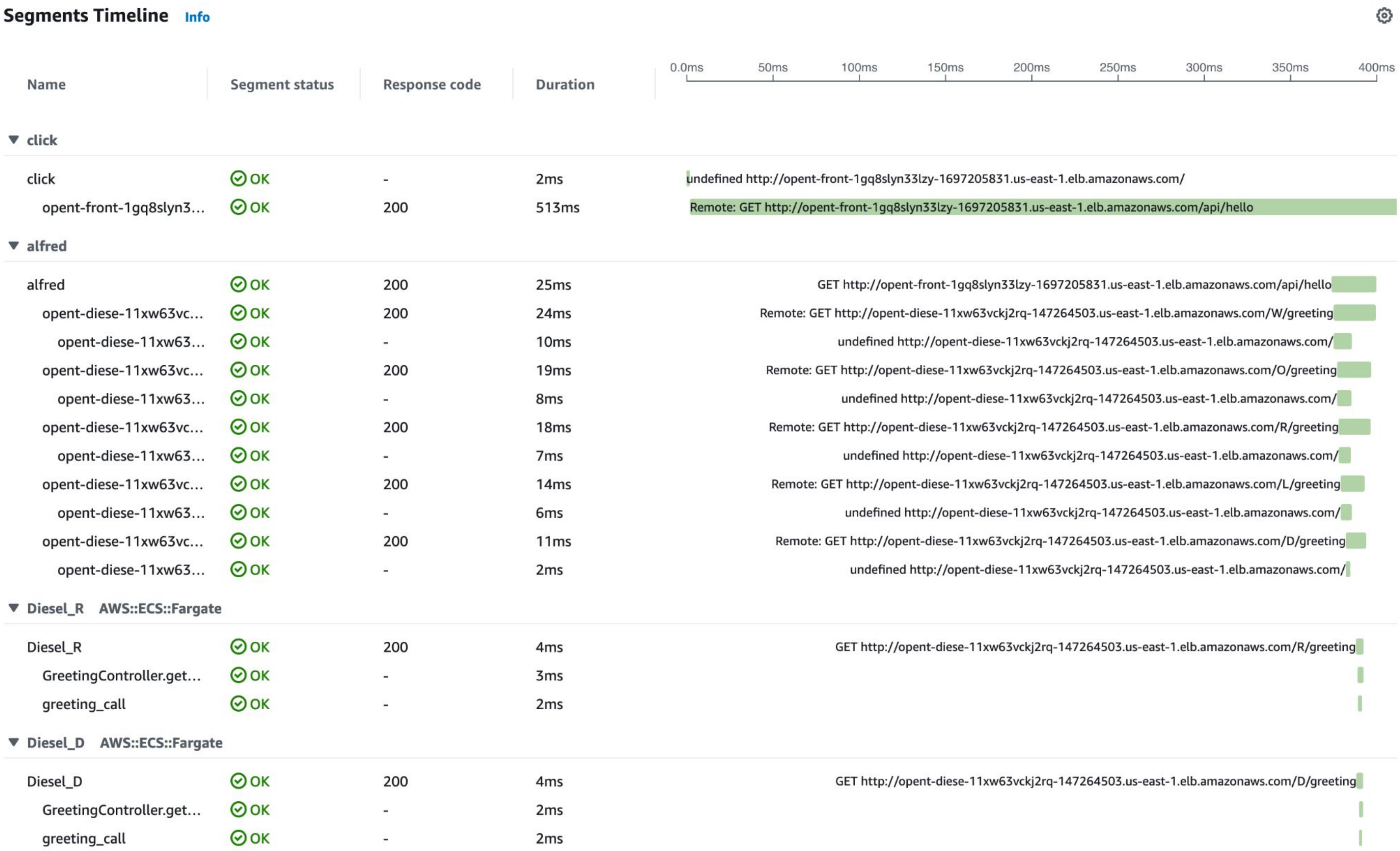
- ID Generation.
- Propagation



```
import {AWSXRayIdGenerator} from "@opentelemetry/id-generator-aws-xray";

sdk.configureTracerProvider({
    idGenerator: new AWSXRayIdGenerator(),
}, new BatchSpanProcessor(new OTLPTraceExporter()));
```

AWS XRay

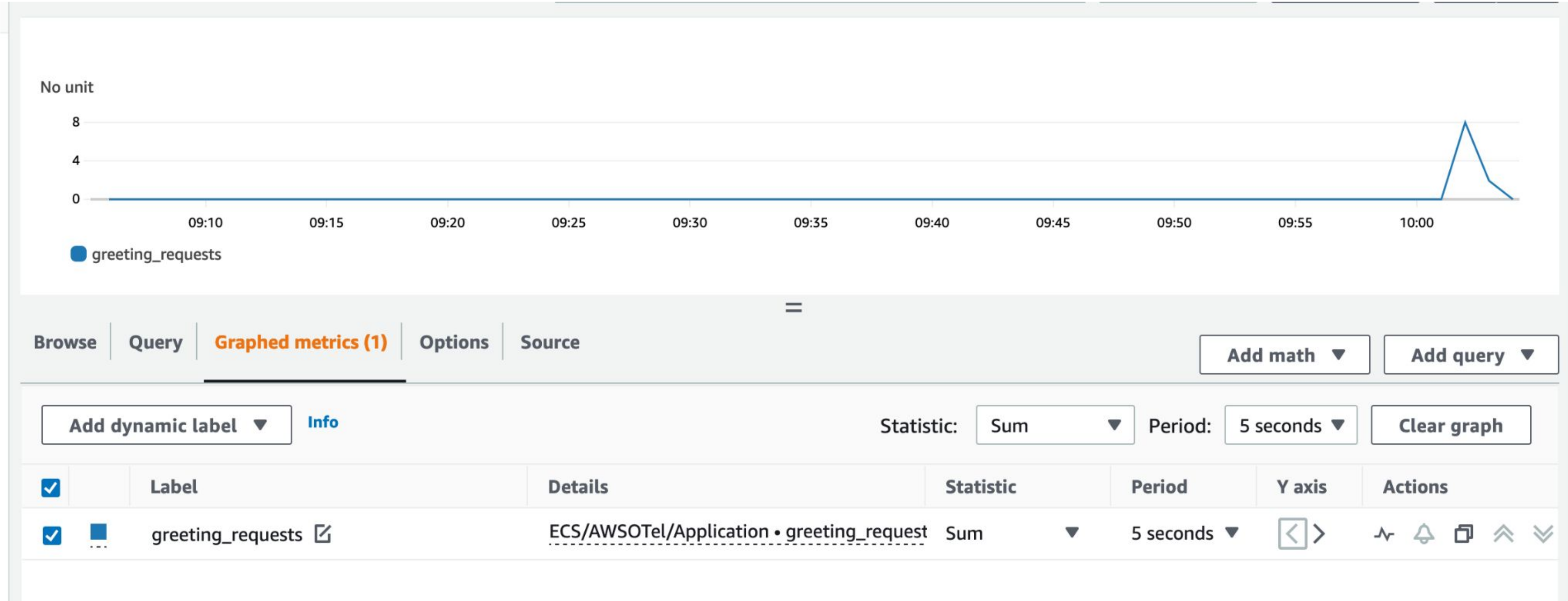

Diesel_W
ECS Fargate
Diesel_L
ECS Fargate
Client
click
alfred
Diesel_D
ECS Fargate
Diesel_R
ECS Fargate
Diesel_O
ECS Fargate



Segments Timeline

Name	Segment status	Response code	Duration	0.0ms 50ms 100ms 150ms 200ms 250ms 300ms 350ms 400ms
▼ click				
click	✓ OK	-	2ms	undefined http://opent-front-1gq8slyn33lzy-1697205831.us-east-1.elb.amazonaws.com/
opent-front-1gq8slyn3...	✓ OK	200	513ms	Remote: GET http://opent-front-1gq8slyn33lzy-1697205831.us-east-1.elb.amazonaws.com/api/hello
▼ alfred				
alfred	✓ OK	200	25ms	GET http://opent-front-1gq8slyn33lzy-1697205831.us-east-1.elb.amazonaws.com/api/hello
opent-diese-11xw63vc...	✓ OK	200	24ms	Remote: GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/W/greeting
opent-diese-11xw63...	✓ OK	-	10ms	undefined http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/
opent-diese-11xw63vc...	✓ OK	200	19ms	Remote: GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/O/greeting
opent-diese-11xw63...	✓ OK	-	8ms	undefined http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/
opent-diese-11xw63vc...	✓ OK	200	18ms	Remote: GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/R/greeting
opent-diese-11xw63...	✓ OK	-	7ms	undefined http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/
opent-diese-11xw63vc...	✓ OK	200	14ms	Remote: GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/L/greeting
opent-diese-11xw63...	✓ OK	-	6ms	undefined http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/
opent-diese-11xw63vc...	✓ OK	200	11ms	Remote: GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/D/greeting
opent-diese-11xw63...	✓ OK	-	2ms	undefined http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/
▼ Diesel_R AWS::ECS::Fargate				
Diesel_R	✓ OK	200	4ms	GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/R/greeting
GreetingController.get...	✓ OK	-	3ms	
greeting_call	✓ OK	-	2ms	
▼ Diesel_D AWS::ECS::Fargate				
Diesel_D	✓ OK	200	4ms	GET http://opent-diese-11xw63vckj2rq-147264503.us-east-1.elb.amazonaws.com/D/greeting
GreetingController.get...	✓ OK	-	2ms	
greeting_call	✓ OK	-	2ms	

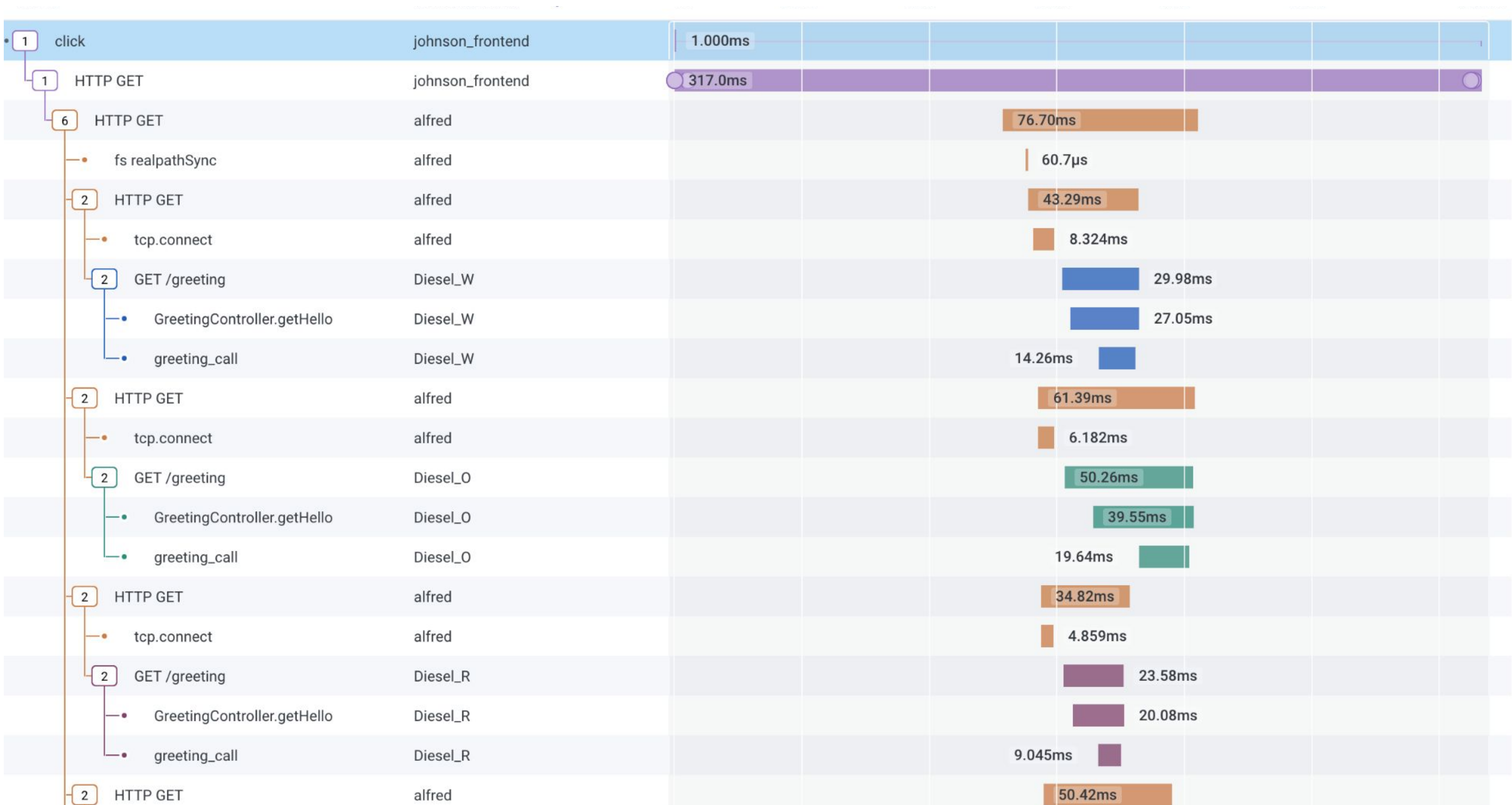
CloudWatch Metrics



Statistic: Sum ▼ Period: 5 seconds ▼

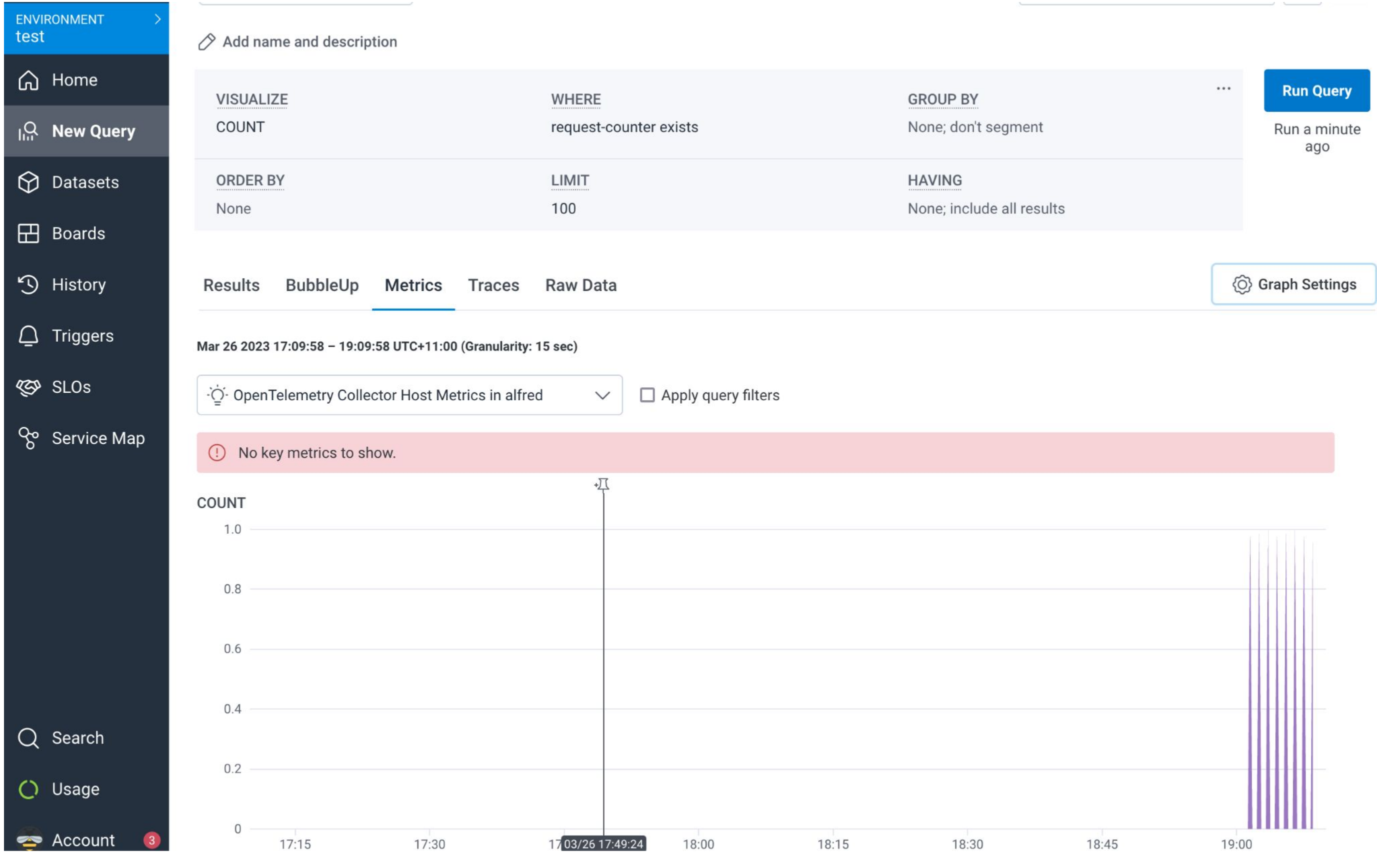
☑		Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	greeting_requests	ECS/AWSOTel/Application • greeting_request	Sum ▼	5 seconds ▼	< >	

Honeycomb.io



1	click	johnson_frontend	1.000ms
1	HTTP GET	johnson_frontend	317.0ms
6	HTTP GET	alfred	76.70ms
	fs realpathSync	alfred	60.7µs
2	HTTP GET	alfred	43.29ms
	tcp.connect	alfred	8.324ms
2	GET /greeting	Diesel_W	29.98ms
	GreetingController.getHello	Diesel_W	27.05ms
	greeting_call	Diesel_W	14.26ms
2	HTTP GET	alfred	61.39ms
	tcp.connect	alfred	6.182ms
2	GET /greeting	Diesel_O	50.26ms
	GreetingController.getHello	Diesel_O	39.55ms
	greeting_call	Diesel_O	19.64ms
2	HTTP GET	alfred	34.82ms
	tcp.connect	alfred	4.859ms
2	GET /greeting	Diesel_R	23.58ms
	GreetingController.getHello	Diesel_R	20.08ms
	greeting_call	Diesel_R	9.045ms
2	HTTP GET	alfred	50.42ms

Honeycomb.io Metrics



VISUALIZE
COUNT

WHERE
request-counter exists

GROUP BY
None; don't segment

Run a minute ago

ORDER BY
None

LIMIT
100

HAVING
None; include all results

Mar 26 2023 17:09:58 – 19:09:58 UTC+11:00 (Granularity: 15 sec)

OpenTelemetry Collector Host Metrics in alfred

□ Apply query filters

No key metrics to show.

Lessons Learnt

Implementation Lessons

- Spec (for tracing+metrics) is stable. SDK is still all over the place, but getting better.
- Java specific, a strategy for managing your Java-agent.
- Add in Otel As Soon as Possible, but only if you don’t have another provider integrated.

Operational Lessons

- Health Checks endpoints on apps are bothersome for low volume apps.
- Understand what you’re sending your provider.

Questions?

Thank You

Feedback? Questions? Drop me an email at **ggigliotti@ippon.tech** , or come and see me a the Ippon Booth.