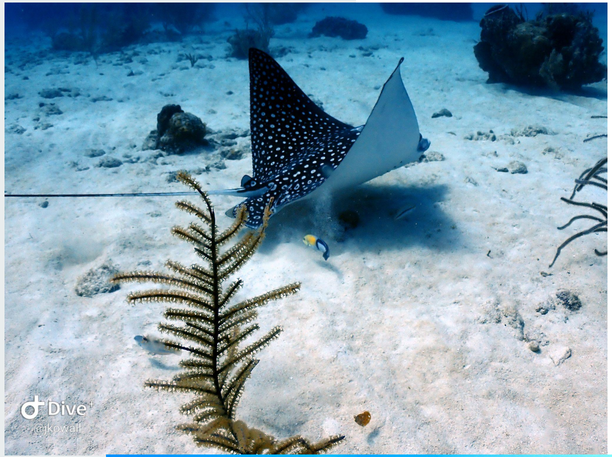
Jaeger present and future

Maintainer Talk

Kubecon EU 2022

Jonah Kowall [@jkowall]
Pavol Loffay [@ploffay]

Jonah Kowall?



logz.io

jkowall

Pavol Loffay?

ploffay

Agenda

1. Intro to Distributed Tracing and Jaeger (Jonah)
2. Jaeger and OpenTelemetry (Pavol)
3. New Monitoring tab and Prometheus support (Jonah)
4. Jaeger Kubernetes Operator deployment (Pavol)
5. New Key Features + Roadmap for Jaeger (Jonah)
6. Q&A from audience in room and online (Jonah + Pavol)

Intro to Distributed Tracing and Jaeger

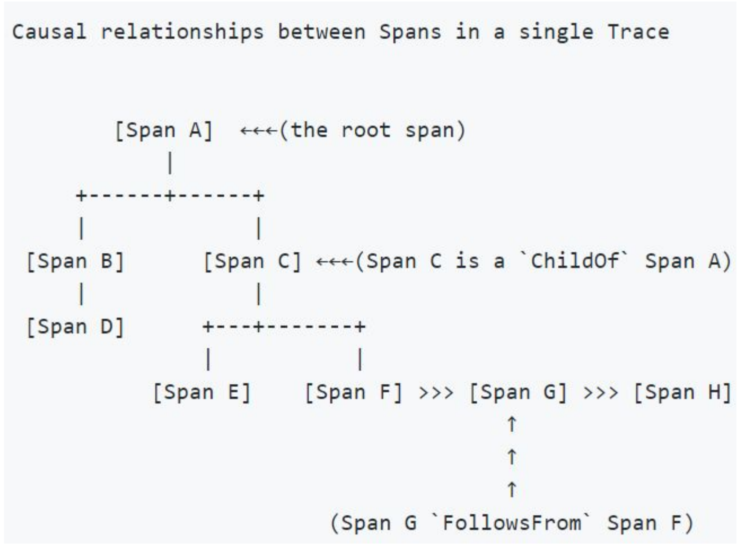
OpenTelemetry Semantics

Trace represents an end-to-end request (and response); made up of single or multiple **Spans**

Span represents work done by a single-service or component with time intervals and associated metadata such as **Tags**

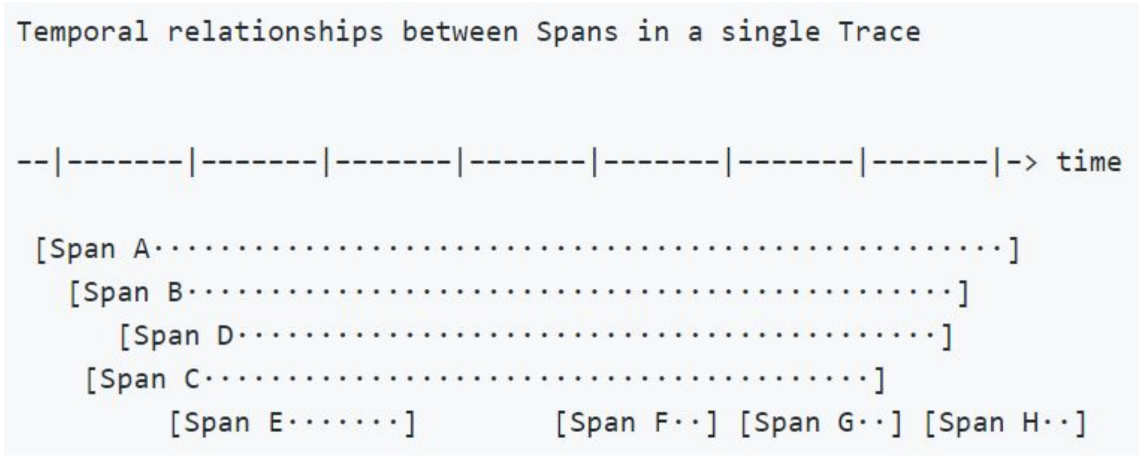
Tags contain metadata to help contextualize a span

Relationships in tracing



```
Causal relationships between Spans in a single Trace


        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
 [Span B]      [Span C] ←←←(Span C is a `ChildOf` Span A)
     |             |
 [Span D]      +---+-------+
               |           |
           [Span E]    [Span F] >>> [Span G] >>> [Span H]
                                       ↑
                                       ↑
                                       ↑
                         (Span G `FollowsFrom` Span F)
```

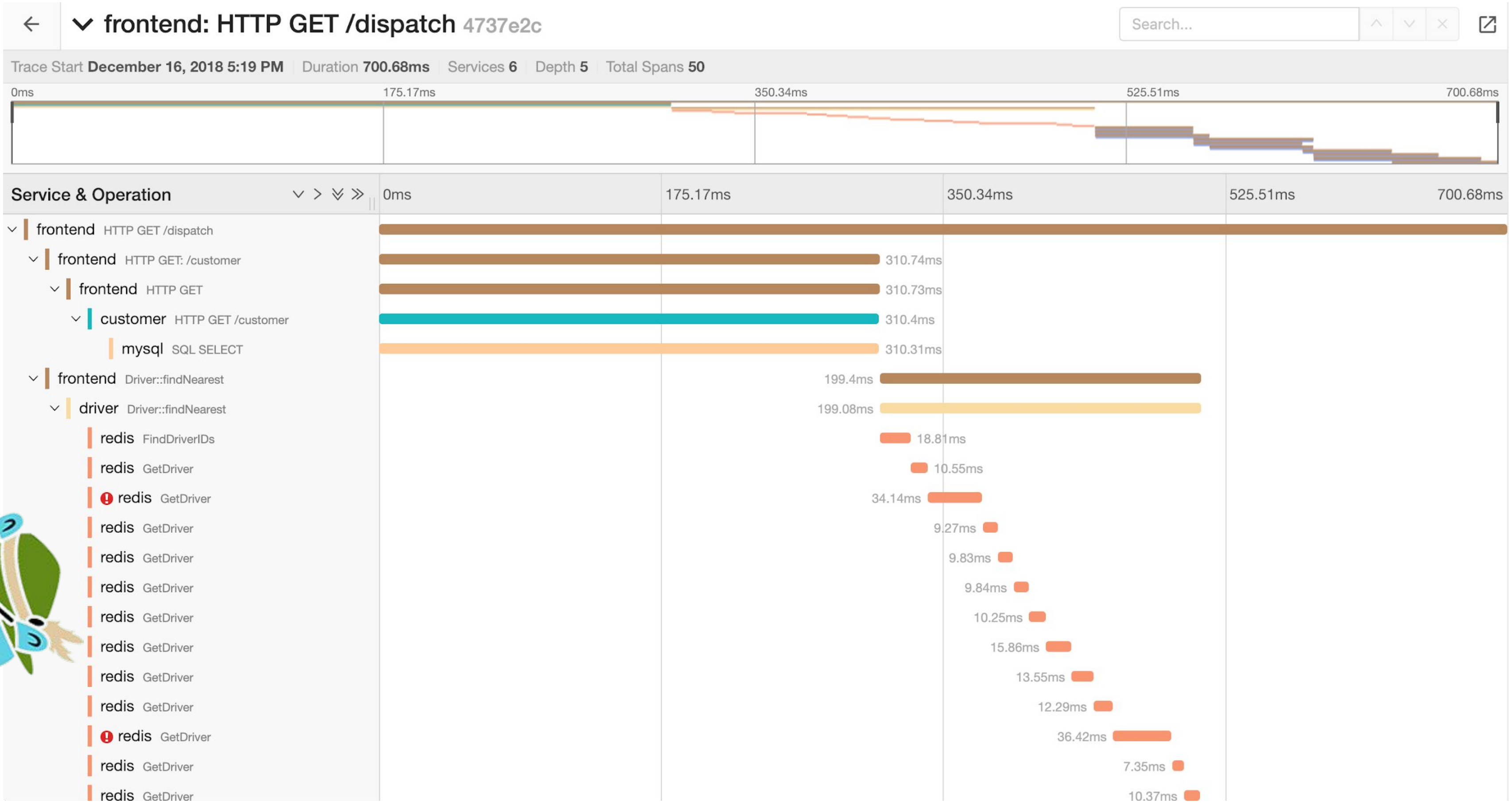


```
Temporal relationships between Spans in a single Trace


––|–––––––|–––––––|–––––––|–––––––|–––––––|–––––––|–––––––|–> time

 [Span A···················································]
   [Span B··············································]
      [Span D··········································]
    [Span C········································]
         [Span E·······]        [Span F··] [Span G··] [Span H··]
```

Measure errors, latency, and other indicators across each span


frontend: HTTP GET /dispatch 4737e2c

Search...

Trace Start **December 16, 2018 5:19 PM** | Duration **700.68ms** | Services **6** | Depth **5** | Total Spans **50**

0ms 175.17ms 350.34ms 525.51ms 700.68ms

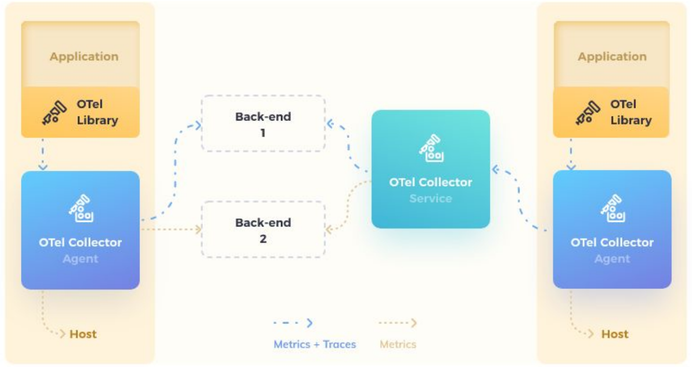
Service & Operation	0ms 175.17ms 350.34ms 525.51ms 700.68ms
frontend HTTP GET /dispatch	
frontend HTTP GET: /customer	310.74ms
frontend HTTP GET	310.73ms
customer HTTP GET /customer	310.4ms
mysql SQL SELECT	310.31ms
frontend Driver::findNearest	199.4ms
driver Driver::findNearest	199.08ms
redis FindDriverIDs	18.81ms
redis GetDriver	10.55ms
❗ redis GetDriver	34.14ms
redis GetDriver	9.27ms
redis GetDriver	9.83ms
redis GetDriver	9.84ms
redis GetDriver	10.25ms
redis GetDriver	15.86ms
redis GetDriver	13.55ms
redis GetDriver	12.29ms
❗ redis GetDriver	36.42ms
redis GetDriver	7.35ms
redis GetDriver	10.37ms

Jaeger and OpenTelemetry

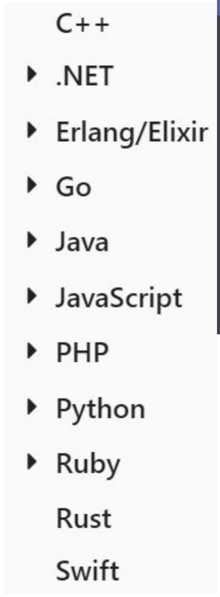
OpenTelemetry and Jaeger

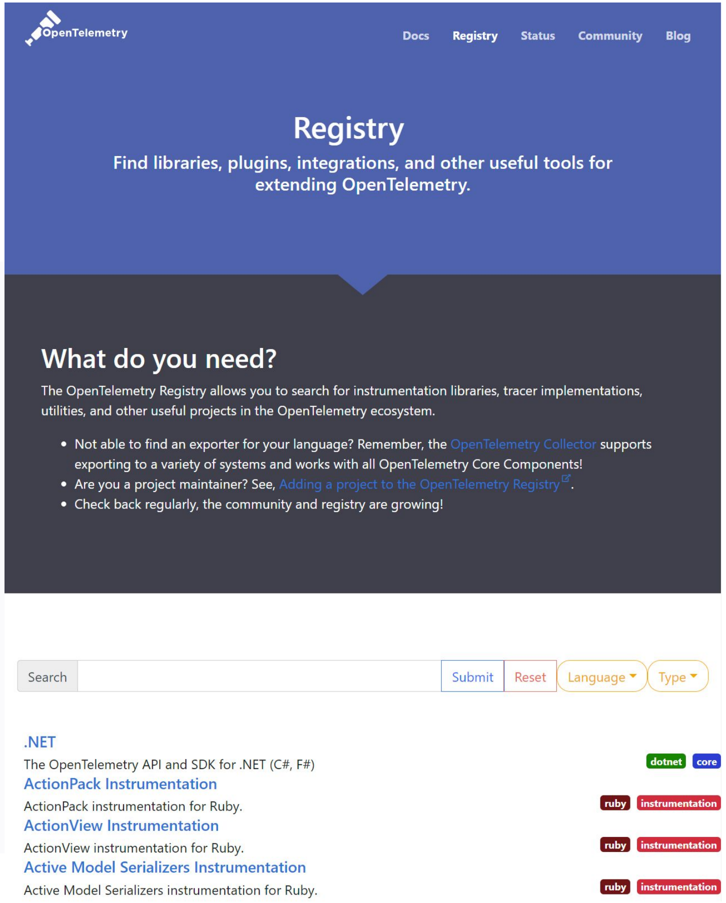
- Jaeger - platform
- OpenTelemetry - data collection

OpenTelemetry Components



REFERENCE ARCHITECTURE





Registry

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

What do you need?

The OpenTelemetry Registry allows you to search for instrumentation libraries, tracer implementations, utilities, and other useful projects in the OpenTelemetry ecosystem.

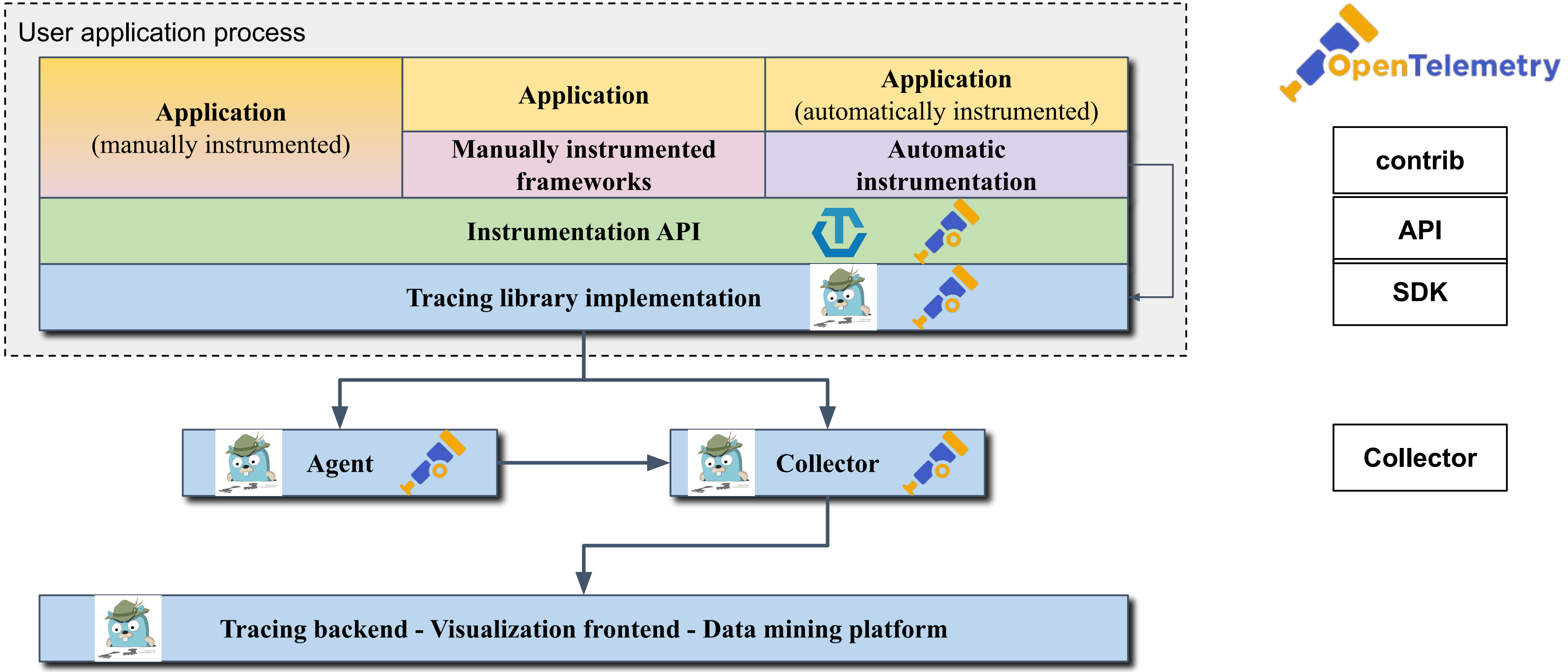
- Not able to find an exporter for your language? Remember, the OpenTelemetry Collector supports exporting to a variety of systems and works with all OpenTelemetry Core Components!
- Are you a project maintainer? See, Adding a project to the OpenTelemetry Registry.
- Check back regularly, the community and registry are growing!

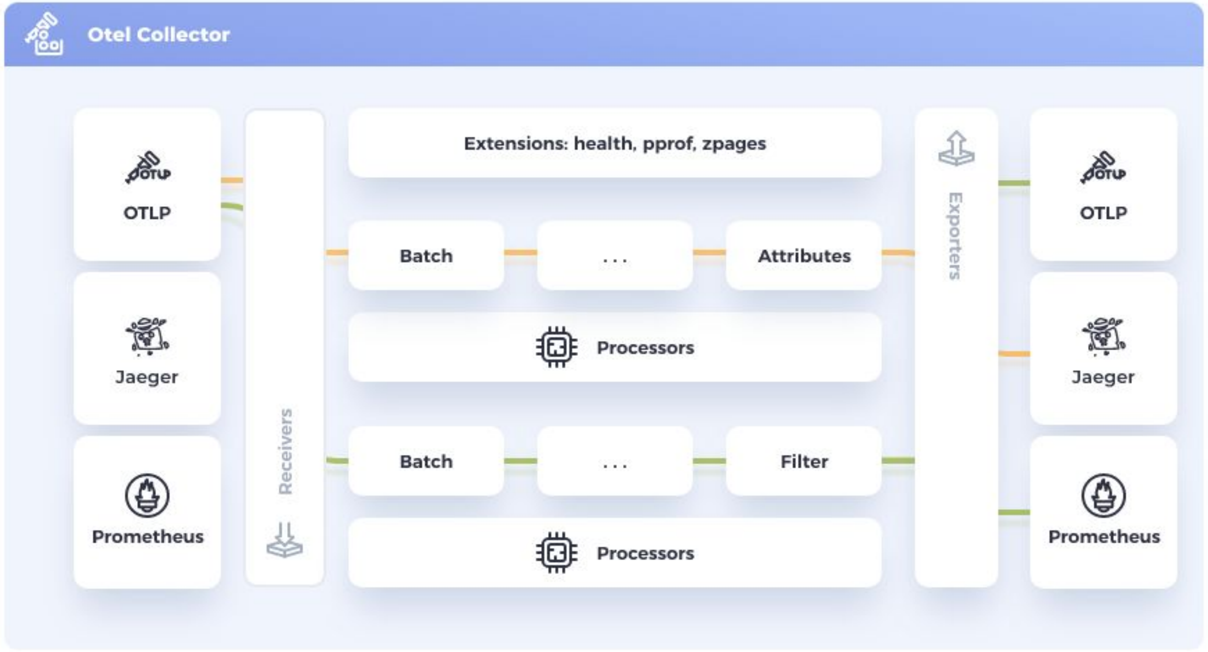
.NET
The OpenTelemetry API and SDK for .NET (C#, F#)

ActionPack Instrumentation
ActionPack instrumentation for Ruby.

ActionView Instrumentation
ActionView instrumentation for Ruby.

Active Model Serializers Instrumentation
Active Model Serializers instrumentation for Ruby.


User application process
Application
(manually instrumented)
Application
Application
(automatically instrumented)
Manually instrumented frameworks
Automatic instrumentation
Instrumentation API
Tracing library implementation
Agent
Collector
Tracing backend - Visualization frontend - Data mining platform
OpenTelemetry
contrib
API
SDK
Collector



OTEL COLLECTOR

Instrumentation

- Jaeger clients are deprecated in favor of OpenTelemetry
- OpenTelemetry SDKs support
    - Jaeger context-propagation header
    - Jaeger remote sampler

OpenTelemetry Collector

- Jaeger receiver
  - proto over gRPC (default endpoint = 0.0.0.0:14250)
  - thrift_binary (default endpoint = 0.0.0.0:6832)
  - thrift_compact (default endpoint = 0.0.0.0:6831)
  - thrift_http (default endpoint = 0.0.0.0:14268)
- Jaeger exporter
  - proto over gRPC
- Jaeger remote sampler extension
  - serves HTTP
- Kafka receiver/exporter
  - Jaeger proto
  - Jaeger JSON

Jaeger V3 Query API

```
// Response object with spans.
message SpansResponseChunk {
  // A list of OpenTelemetry ResourceSpans.
  // In case of JSON format the ids (trace_id, span_id, parent_id)
  //    are encoded in base64 even though OpenTelemetry specification
  //    mandates to use hex encoding [2].
  // Base64 is chosen to keep compatibility with JSONPb codec.
  // [1]:
https://github.com/open-telemetry/opentelemetry-proto/blob/main/opentelemetry/proto/trace/v1/tr
ace.proto
  // [2]:
https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/protocol/
otlp.md#otlphttp

  repeated opentelemetry.proto.trace.v1.ResourceSpans resource_spans = 1;
}
```

New Monitoring tab and Prometheus support

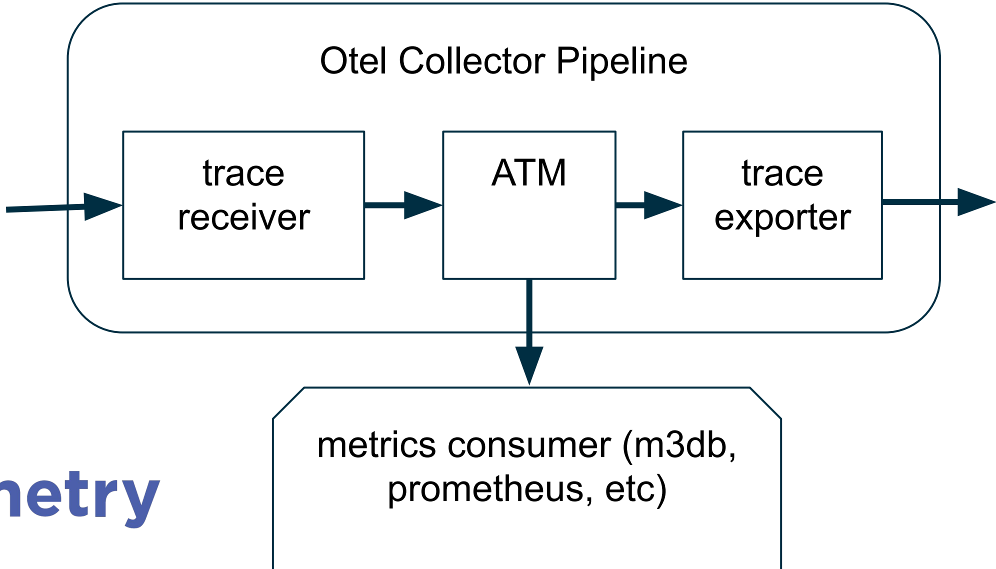
Tracing and Monitoring

- What’s the difference between “distributed tracing” and “Application Performance Monitoring - APM”?
    - Traces / Events
    - Metrics
- Use cases
    - Monitoring
    - Alerting
    - Planning

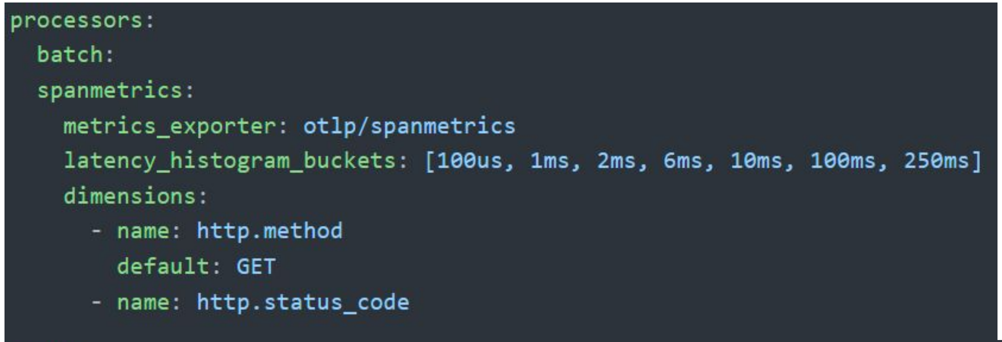
Aggregated Trace Metrics (ATM)

- Prometheus can handle all of these metrics use cases
- We just have to generate the metrics from the traces

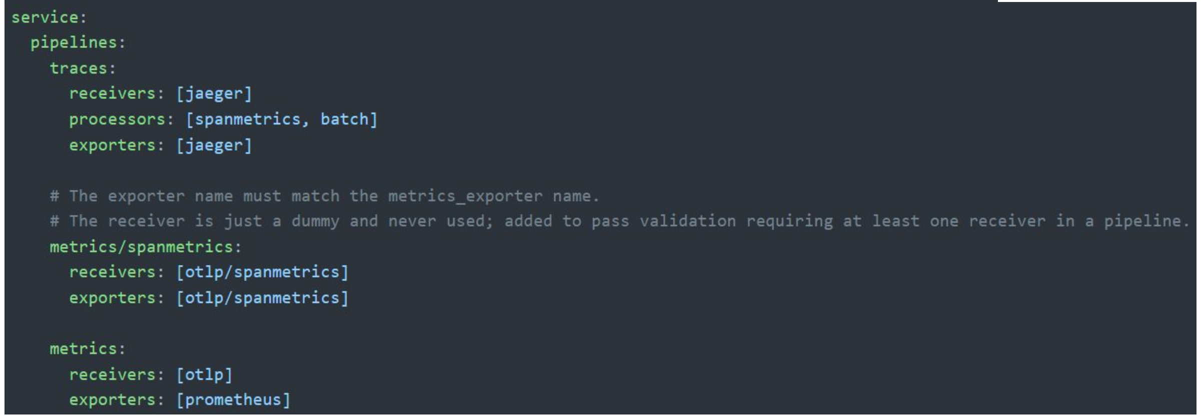
Derive aggregated metrics from traces



SpanMetrics Processor in OpenTelemetry



```yaml
processors:
  batch:
  spanmetrics:
    metrics_exporter: otlp/spanmetrics
    latency_histogram_buckets: [100us, 1ms, 2ms, 6ms, 10ms, 100ms, 250ms]
    dimensions:
      - name: http.method
        default: GET
      - name: http.status_code
```



```yaml
service:
  pipelines:
    traces:
      receivers: [jaeger]
      processors: [spanmetrics, batch]
      exporters: [jaeger]

    # The exporter name must match the metrics_exporter name.
    # The receiver is just a dummy and never used; added to pass validation requiring at least one receiver in a pipeline.
    metrics/spanmetrics:
      receivers: [otlp/spanmetrics]
      exporters: [otlp/spanmetrics]

    metrics:
      receivers: [otlp]
      exporters: [prometheus]
```

Define how many metrics (in this case method type and status code) and the buckets.

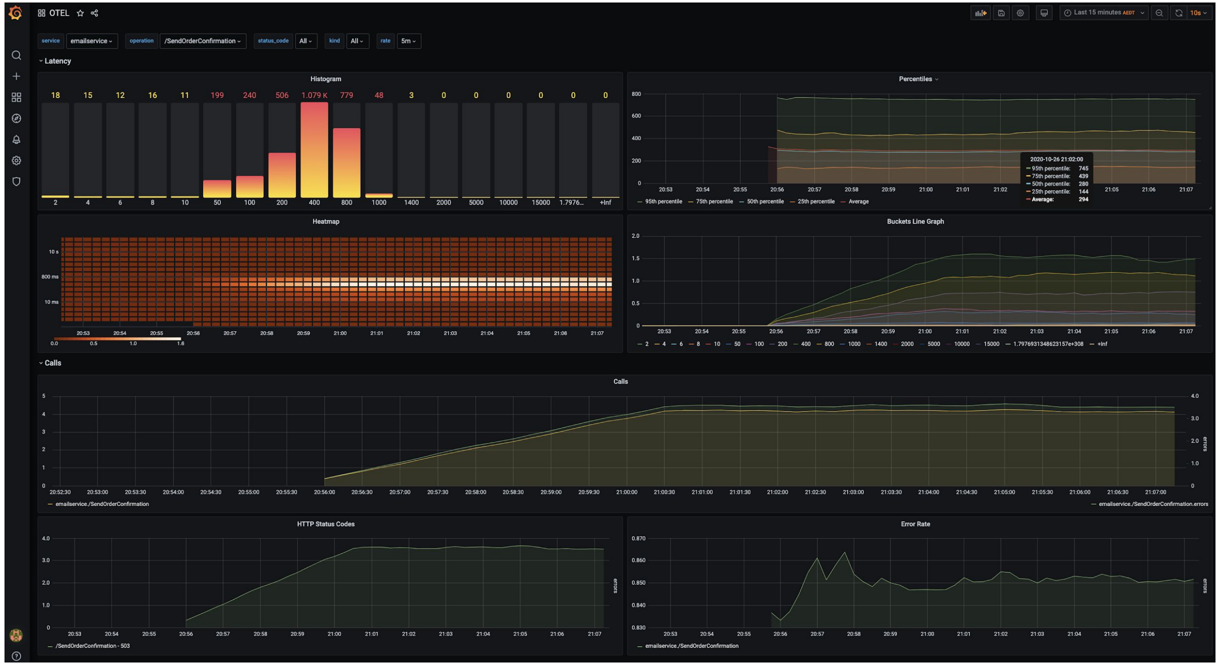
Result: Generate a metric per bucket per status code

[opentelemetry-collector-contrib/processor/spanmetricsprocessor at main · open-telemetry/opentelemetry-collector-contrib (github.com)]

Using Prometheus Metrics

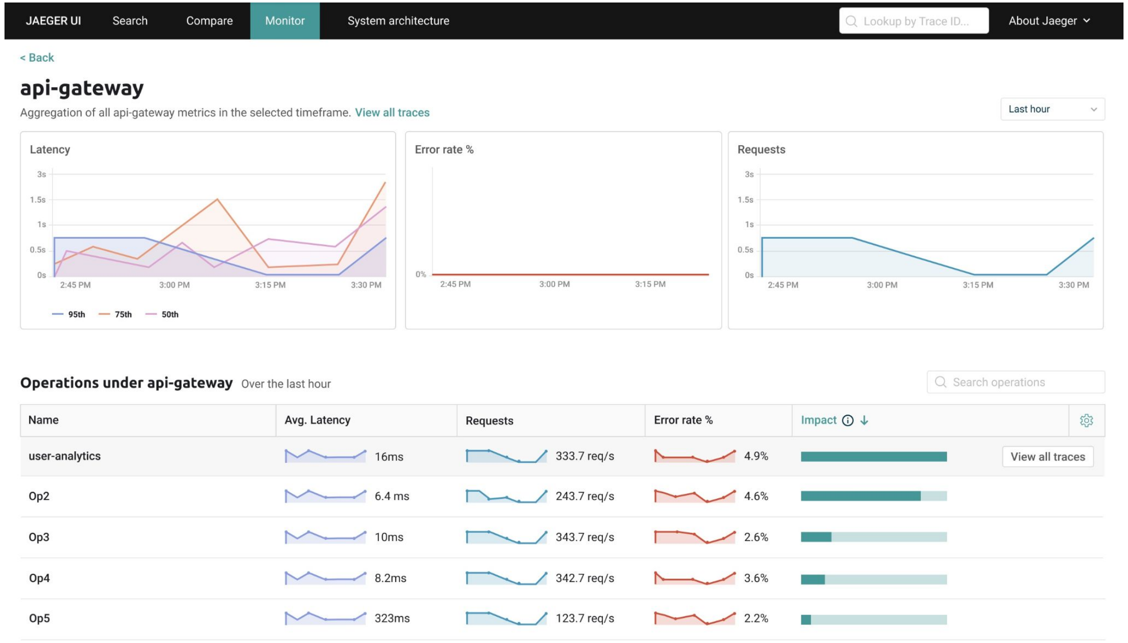
MetricQuery service in Jaeger to query metric backends.

- First support will be for promql compatible backends (ex: Prometheus, Cortex, Thanos, M3DB)
- Community can add other systems as needed



Using Prometheus Metrics inside Jaeger UI

- New “monitor” homepage in Jaeger to provide status and health of transactions



api-gateway

Aggregation of all api-gateway metrics in the selected timeframe. View all traces

Operations under api-gateway Over the last hour

Name	Avg. Latency	Requests	Error rate %	Impact	
user-analytics	16ms	333.7 req/s	4.9%		View all traces
Op2	6.4 ms	243.7 req/s	4.6%		
Op3	10ms	343.7 req/s	2.6%		
Op4	8.2ms	342.7 req/s	3.6%		
Op5	323ms	123.7 req/s	2.2%		

Jaeger Kubernetes Operator

Jaeger Operator

"operates" Jaeger on Kubernetes

- github.com/jaegertracing/jaeger-operator
- jaegertracing.io/docs/latest/operator

Getting Started

- jaegertracing.io/docs/1.34/operator
- Storage schema creation
- Jaeger upgrades
- Auto recognizes available APIs - OpenShift/Kubernetes, ES/Strimzi
- Can generate plain Kubernetes manifest files

Jaeger CRD

- CRD - Custom Resource Definition

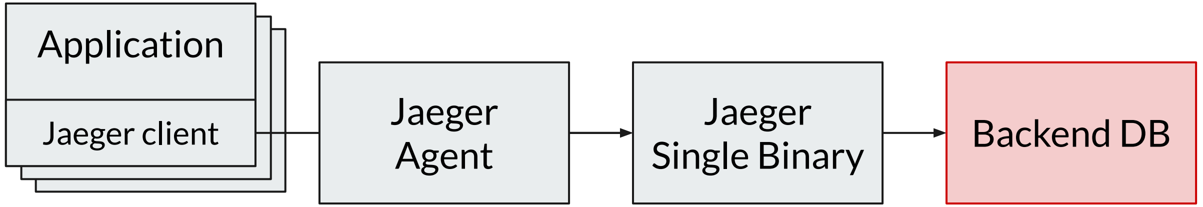
```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  strategy: allInOne | production | streaming
```

- `kubectl get jaegers`

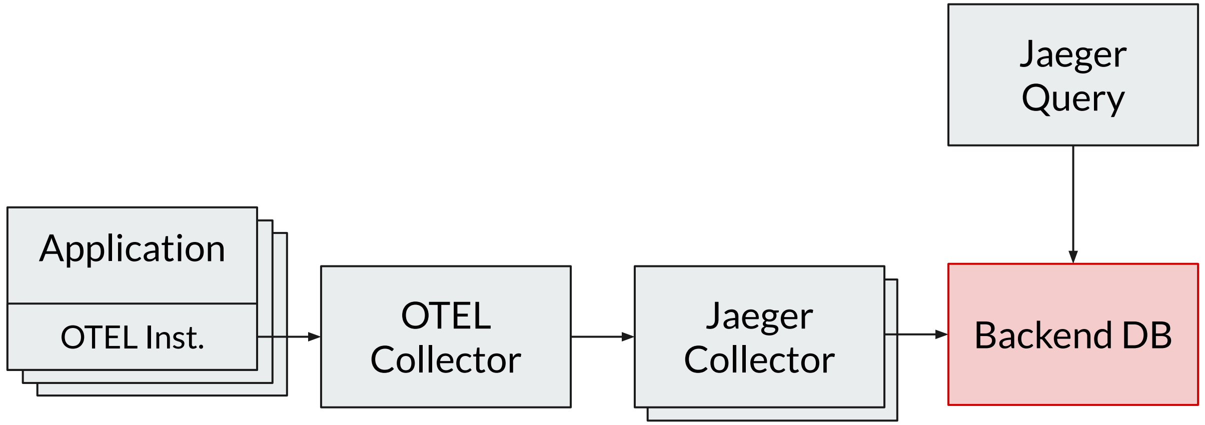
Jaeger Operator

Strategies

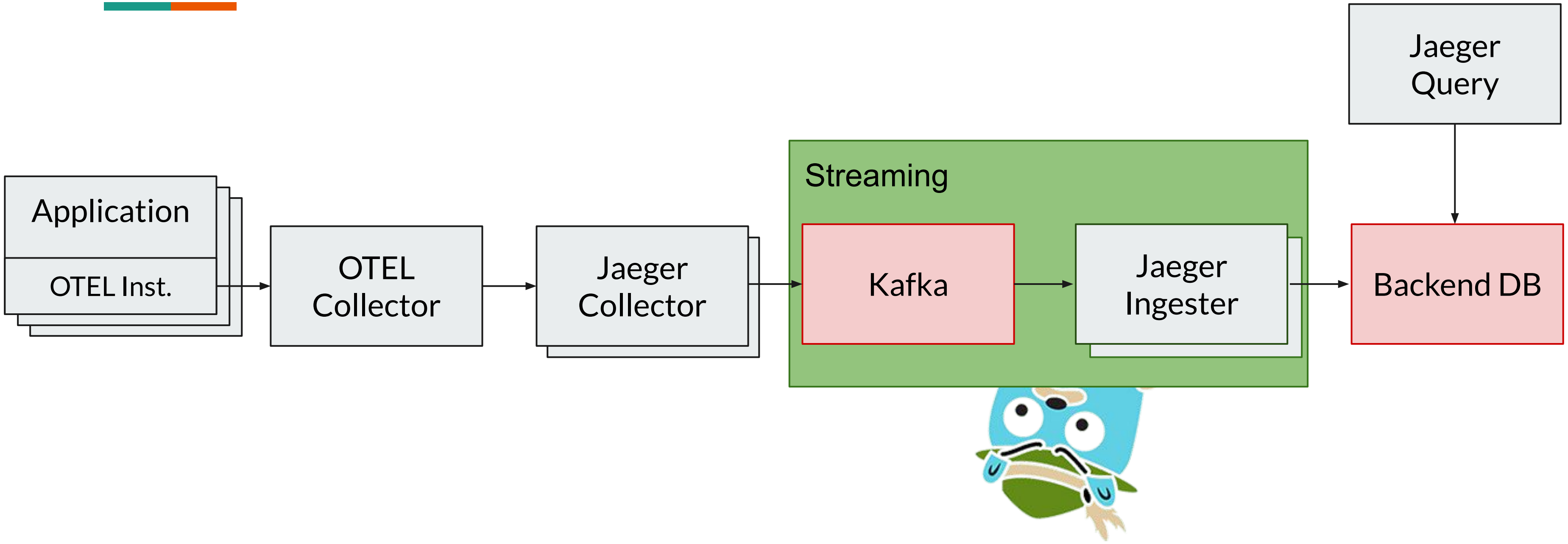
Strategies: all-in-one



Strategies: production



Strategies: streaming



Jaeger Operator

Storage configuration

Storage Configuration (backend)

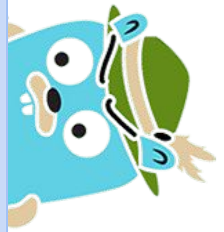
spec.storage

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  storage:
    type: memory|elasticsearch|cassandra|badger|grpc-plugin|kafka
    options:
      es:
        server-urls: http://some-elastic-cluster-somewhere:9200
```

Detailed Configuration

```
$ docker run -e SPAN_STORAGE_TYPE=elasticsearch jaegertracing/jaeger-collector --help
...
 --collector.num-workers int    The number of workers pulling items from the queue
 --collector.queue-size int     The queue size of the collector
 --collector.queue-size-memory uint  (experimental) The max memory size in MiB
...
```

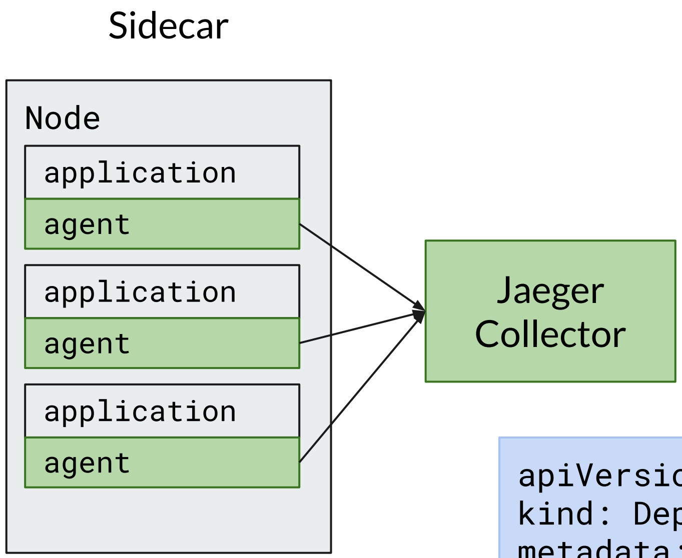
```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  collector:
    options:
      collector:
        queue-size: 100
```



Jaeger Operator

Agent strategies

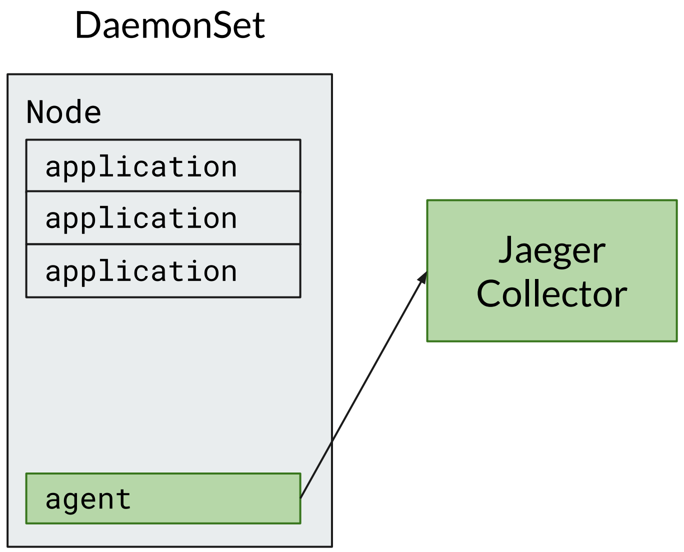
Agent Strategy (Sidecar)



```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-dev
spec:
  agent:
    strategy: Sidecar
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    "sidecar.jaegertracing.io/inject": "true"
```

Agent Strategy (DaemonSet)



```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  agent:
    strategy: DaemonSet
```

Remote Sampling

```yaml
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  sampling:
    options:
      default_strategy:
        type: probabilistic
        param: 0.5
      service_strategies:
      - service: foo
        type: probabilistic
        param: 0.8
        operation_strategies:
        - operation: get
          type: probabilistic
          param: 0.2
```

Autoscaling Collectors and Ingesters

- Collectors and Ingesters
- By default creates an HPA with max of 100

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  collector:
    maxReplicas: 20
```

```
apiVersion: jaegertracing.io/v1
kind: Jaeger
metadata:
  name: jaeger-cluster
spec:
  collector:
    autoScale: false
    replicas: 10
```

Operator Integrations

- Storage
  - Kafka - Strimzi
  - Elasticsearch - OpenShift cluster logging
- Monitoring Jaeger operator
  - Prometheus
  - OpenTelemetry

New Key Features + Roadmap

New Key Features

- Adaptive Sampling - Jaeger backend can be configured to perform fully automated and dynamic control of sampling rates based on predefined targets
- Service Performance Monitoring - We covered this
- All-in on OpenTelemetry - The Jaeger Clients/SDKs have been officially retired in favor of OpenTelemetry

Roadmap

- Updates to dependency graphs
    - Normalize the three types of graphs in Jaeger
    - Overlay service performance metrics on graph
    - Potentially move calculations from Spark/Kafka streams to OpenTelemetry collector
- Move towards OpenTelemetry collector
    - Remove the need for Jaeger collector and normalize on a distribution of the collector for writes to Jaeger data stores
    - Native OTLP support

And more interesting capabilities coming in the future

Q&A from audience in room and online

Resources

jaegertracing.io/docs

monthly community call and Notes
CNCF Slack #jaeger : https://slack.cncf.io

@jaegertracing

medium.com/jaegertracing