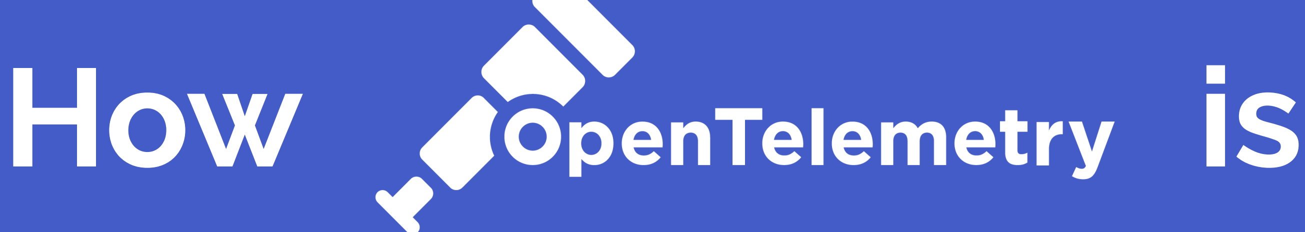



# Eating the O11y World

May 8, 2020

# Agenda

- Introduction
- Architecture
- Specifications
- Collector
- Client Libraries
- Demo

# Introduction

@smflanders

flands

https://sflanders.net

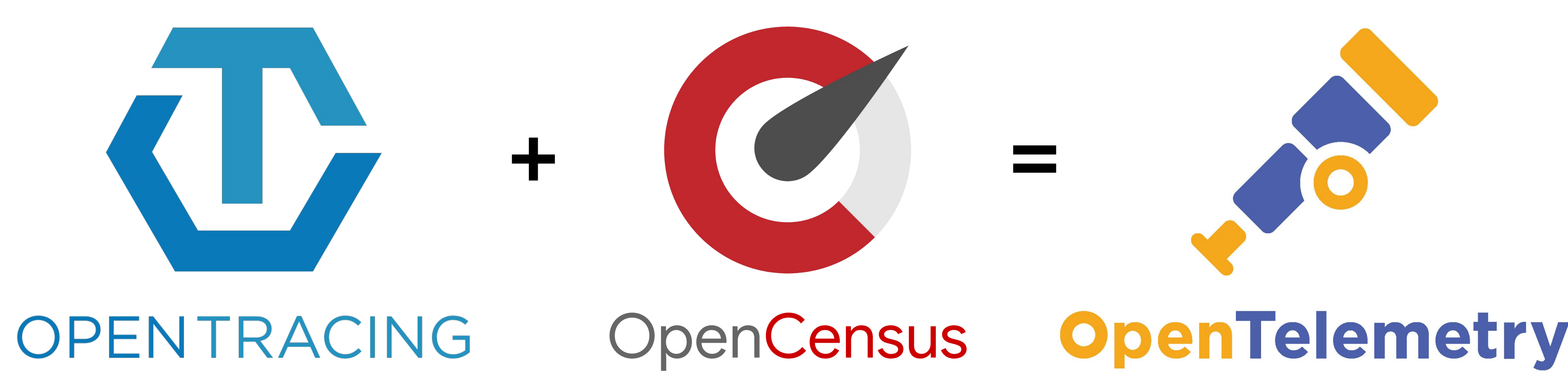
# Steve Flanders

Director of Engineering, Splunk
OpenTelemetry Collector Approver
CNCF SIG-Observability Chair Nominee

Previously:

- Head of Product, Omnition
- Global Engineering Manager for Logs, VMware

# What is OpenTelemetry?




OpenTelemetry: **the next major version** of *both* OpenTracing and OpenCensus

# Cloud Native Telemetry

Telemetry “verticals”

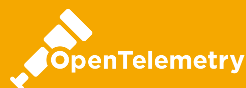
Telemetry “layers”

| | Tracing | Metrics | Logs, etc |
| --- | --- | --- | --- |
| Instrumentation APIs | | foreach(language) | |
| Canonical implementations | | foreach(language) | |
| Data infrastructure | | collectors, sidecars, etc | |
| Interop formats | | W3C trace-context, wire formats for trace data, metrics, logs, etc | |

OpenTelemetry

# Project Stats

- **CNCF DevStats**
  - **General:** 104 members (245+ active contributors) from 45+ companies and 40+ countries
  - **Contributors:** 660+ unique contributors and 60K+ contributions
- **Community Stats**
  - **Cloud Providers:** Azure and GCP
  - **Vendors:** Datadog, Dynatrace, Honeycomb, Lightstep, New Relic, Splunk, Stackdriver
  - **Users (and contributors)**: Mailchimp, Postmates, Shopify, Zillow
- **CNCF Project Collaboration**
  - **Fluentbit:** Potential log agent for OpenTelemetry
  - **Jaeger:** Plan to leverage client libraries and collector (collector already announced)



# OpenTelemetry

## is the second most active project in CNCF today!

(per CNCF DevStats)

# Architecture

# Components

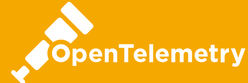
1. **Specifications**
    a. API
    b. SDK
    c. Data
2. **Collector**
    a. Vendor-agnostic way to receive, process, and export data
    b. Default way to collect instrumented apps
    c. Can be deployed as an agent or service
3. **Client Libraries**
    a. Vendor-agnostic app instrumentation
    b. Support for traces and metrics
    c. Automatic trace instrumentation
4. **Incubating: Logging**

**Status = Beta for Traces + Metrics:**

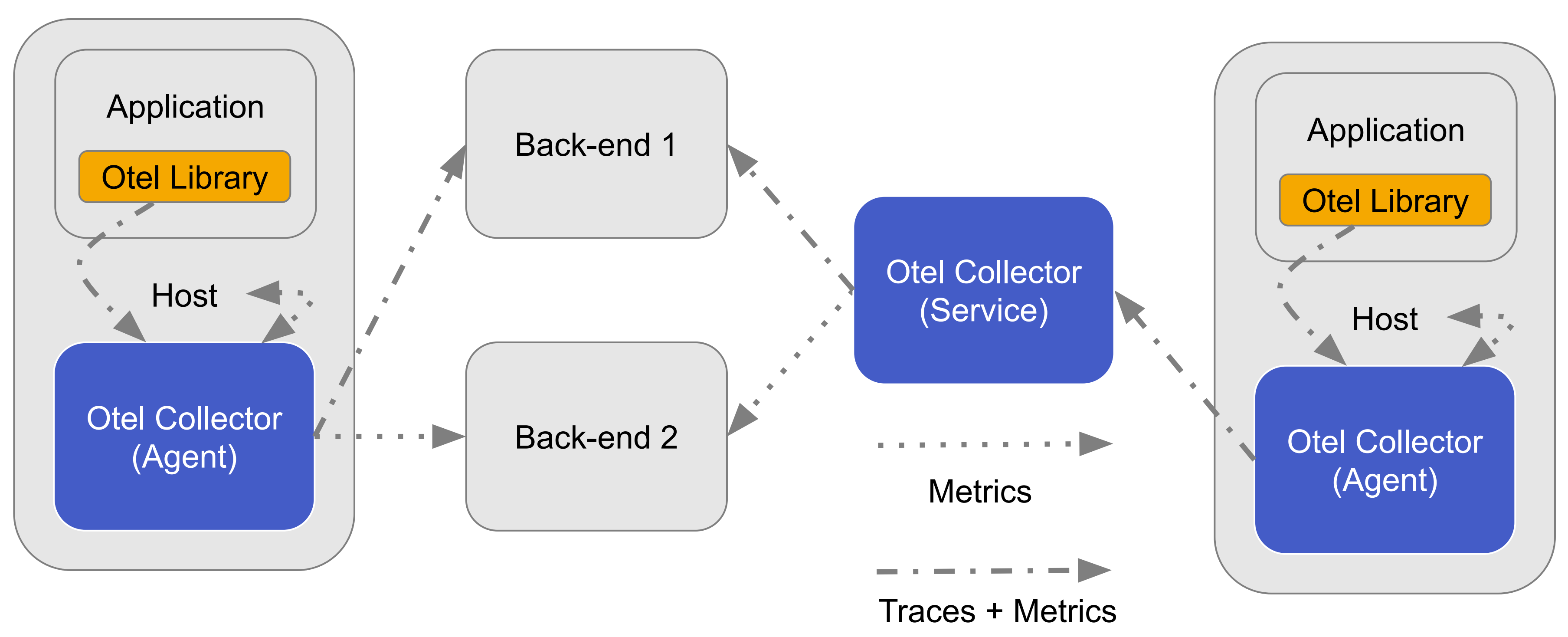
- Collector
- Erlang
- Go
- Java (including auto instrumentation)
- Javascript (including web)
- Python (auto instrumentation planned)

**Coming soon:**

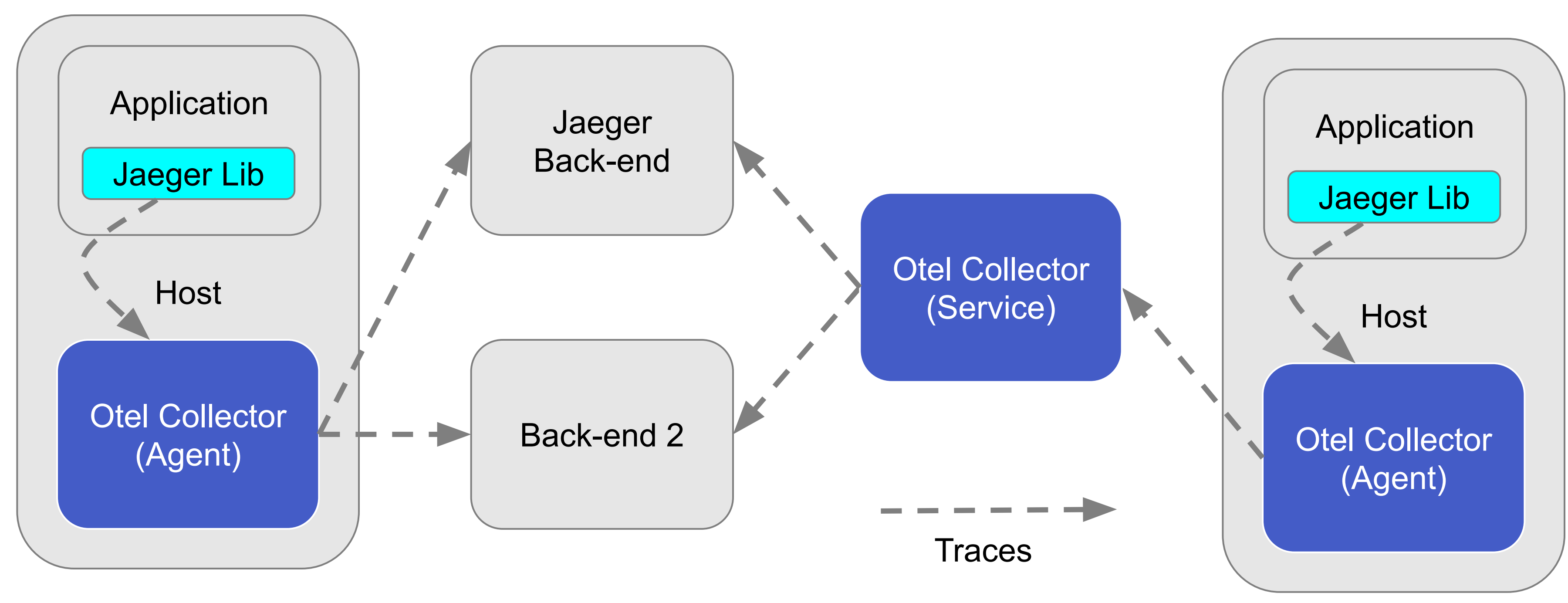
- .NET (auto instrumentation planned)
- Ruby (auto instrumentation planned)



# Reference Architecture: OpenTelemetry



# Reference Architecture: Jaeger

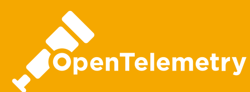



https://medium.com/jaegertracing/jaeger-embraces-opentelemetry-collector-90a545cbc24

# Specifications

# Tracing Basics

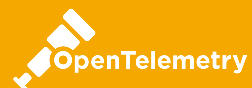
- **Context:** W3C trace-context, B3, etc.
- **Tracer:** get context
- **Spans:** “call” in a trace
  - **Kind:** client/server, producer/consumer, internal
  - **Attributes:** key/value pairs; tags; metadata
  - **Events:** named strings
  - **Links:** useful for batch operations
- **Sampler:** always, probabilistic, etc.
- **Span processor:** simple, batch, etc.
- **Exporter:** OTLP, Jaeger, Prometheus, Zipkin, etc.



# Tracing Semantic Conventions

In OpenTelemetry, spans can be created freely and it's up to the implementor to annotate them with attributes specific to the represented operation. Some span operations represent calls that use well-known protocols like HTTP or database calls. It is important to unify attribution.

- **HTTP:** http.method, http.status_code
- **Database:** db.type, db.instance, db.statement
- **Messaging:** messaging.system, messaging.destination
- **FaaS:** faas.trigger



# Semantic Conventions: Example

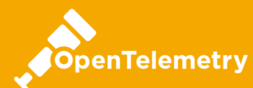
# Metric Basics

- **Context:** span and correlation
- **Meter:** used to record a measurement
- **Raw Measurement**
  - **Measure:** name, description, unit of values
  - **Measurement:** single value of a measure
- **Metric:** a measurement
  - **Kind:** counter, measure, observer
  - **Label:** key/value pair; tag; metadata
- **Aggregation**
- **Time**

# Resource SDK + Semantic Conventions

A Resource is an immutable representation of the entity producing telemetry. For example, a process producing telemetry that is running in a container on Kubernetes has a Pod name, it is in a namespace and possibly is part of a Deployment. All three of these attributes can be included in the Resource.

- **Environment:** Attributes defining a running environment (e.g. cloud)
- **Compute instance:** Attributes defining a computing instance (e.g. host)
- **Deployment service:** Attributes defining a deployment service (e.g. k8s).
- **Compute unit:** Attributes defining a compute unit (e.g. container, process)



# Collector

# Objectives

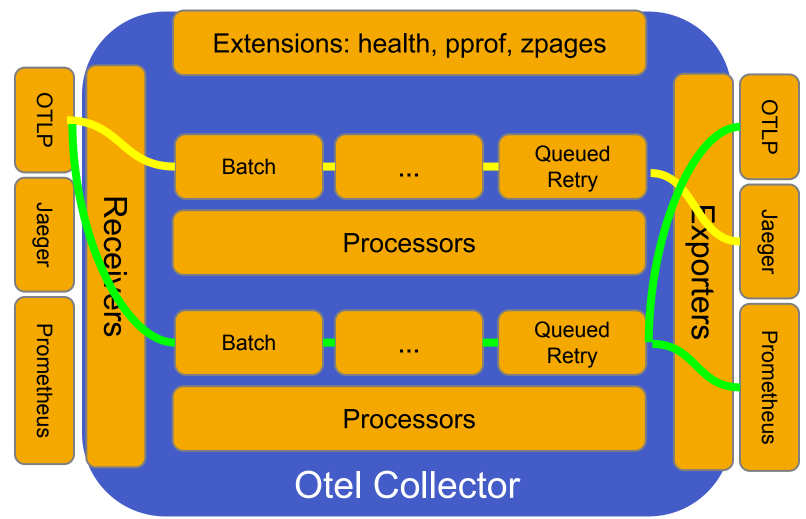
The OpenTelemetry Collector offers a vendor-agnostic implementation on how to receive, process, and export telemetry data in a seamless way.

- **Usable:** Reasonable default configuration, supports popular protocols, runs and collects out of the box.
- **Performant:** Highly performant under varying loads and configurations.
- **Observable:** An exemplar of an observable service.
- **Extensible:** Customizable without touching the core code.
- **Unified:** Single codebase, deployable as an agent or collector with support for traces, metrics, and logs (future).

# But why?

- Offload responsibility from the application
  - Compression
  - Encryption
  - Retry
  - Tagging / Redaction
  - Vendor-specific exporting
- Time-to-value
  - Language-agnostic; makes changes easier
  - Set it and forget it; instrumentation that is ready for the Collector
  - Vendor-agnostic and easily extensible

# Architecture



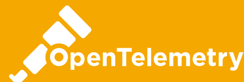
# Core (Maintainers) Components

**Traces**

- Receivers/Exporters
  - OTLP
  - Jaeger
  - Zipkin
- Processors
  - Attributes
  - Batch
  - Queued Retry
  - Resource
  - Sampling
  - Span

**Metrics**

- Receivers
  - OTLP
  - Host (CPU, Disk, Memory, Network)
  - Prometheus
- Processors
  - *Coming soon…*
- Exporters
  - OTLP
  - Prometheus



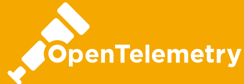
# Contrib (Community) Components

**Traces**

- Receivers
  - SignalFx
- Processors
  - Kubernetes
- Exporters:
  - AWS X-ray
  - Azure Monitor
  - Honeycomb
  - Kinesis
  - Lightstep
  - SignalFx
  - Stackdriver

**Metrics**

- Receivers
  - Carbon
  - Kubernetes
  - Redis
  - Wavefront
- Exporters
  - Carbon
  - SignalFx
  - Stackdriver



# Client Libraries: Java

# Getting Started

### Traces

1. Instantiate a tracer
2. Create spans
3. Enhance spans
4. Configure SDK

### Metrics

1. Instantiate a meter
2. Create metrics
3. Enhance metrics
4. Configure observer

# Getting Started: Traces (Manual)

```
# Instantiate tracer
Tracer tracer =
      OpenTelemetry.getTracer("instrumentation-library-name","semver:1.0.0");

# Create span
Span span = tracer.spanBuilder("my span").startSpan();
try (Scope scope = tracer.withSpan(span)) {
      // your use case
      # Enhance span
      span.setAttribute("version", "1.2");
} catch (Throwable t) {
    Status status = Status.UNKNOWN.withDescription("Change it to your error message");
    span.setStatus(status);
} finally {
    span.end(); // closing the scope does not end the span, this has to be done manually
}
```

# Getting Started: Traces (Manual)

```
# Instantiate tracer
Tracer tracer =
      OpenTelemetry.getTracer("instrumentation-library-name","semver:1.0.0");

# Create span
Span span = tracer.spanBuilder("my span").startSpan();
try (Scope scope = tracer.withSpan(span)) {
      // your use case
      # Enhance span
      span.setAttribute("version", "1.2");
} catch (Throwable t) {
    Status status = Status.UNKNOWN.withDescription("Change it to your error message");
    span.setStatus(status);
} finally {
    span.end(); // closing the scope does not end the span, this has to be done manually
}
```

# Getting Started: Traces (Manual)

```
# Instantiate tracer
Tracer tracer =
      OpenTelemetry.getTracer("instrumentation-library-name","semver:1.0.0");

# Create span
Span span = tracer.spanBuilder("my span").startSpan();
try (Scope scope = tracer.withSpan(span)) {
      // your use case
      # Enhance span
      span.setAttribute("version", "1.2");
} catch (Throwable t) {
    Status status = Status.UNKNOWN.withDescription("Change it to your error message");
    span.setStatus(status);
} finally {
    span.end(); // closing the scope does not end the span, this has to be done manually
}
```

# Getting Started: Traces (Manual)

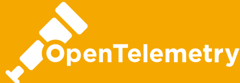
```
# Instantiate tracer
Tracer tracer =
      OpenTelemetry.getTracer("instrumentation-library-name","semver:1.0.0");

# Create span
Span span = tracer.spanBuilder("my span").startSpan();
try (Scope scope = tracer.withSpan(span)) {
      // your use case
      # Enhance span
      span.setAttribute("version", "1.2");
} catch (Throwable t) {
    Status status = Status.UNKNOWN.withDescription("Change it to your error message");
    span.setStatus(status);
} finally {
    span.end(); // closing the scope does not end the span, this has to be done manually
}
```

# Getting Started: Traces (Manual)

```
// Get the tracer
TracerSdkProvider tracerProvider = OpenTelemetrySdk.getTracerProvider();

// Configure the sampler to use
tracerProvider.updateActiveTraceConfig(
    TraceConfig alwaysOn = TraceConfig.getDefault().toBuilder().setSampler(
        Samplers.alwaysOn()
    ).build();
);

// Set to export the traces to via Jaeger
ManagedChannel jaegerChannel =
    ManagedChannelBuilder.forAddress([ip:String], [port:int]).usePlaintext().build();
JaegerGrpcSpanExporter jaegerExporter = JaegerGrpcSpanExporter.newBuilder()
    .setServiceName("example").setChannel(jaegerChannel).setDeadline(30000)
    .build();
tracerProvider.addSpanProcessor(
    BatchSpansProcessor.newBuilder(
        jaegerExporter
      ).build());
```

# Getting Started: Traces (Manual)

```
// Get the tracer
TracerSdkProvider tracerProvider = OpenTelemetrySdk.getTracerProvider();

// Configure the sampler to use
tracerProvider.updateActiveTraceConfig(
    TraceConfig alwaysOn = TraceConfig.getDefault().toBuilder().setSampler(
        Samplers.alwaysOn()
    ).build();
);

// Set to export the traces to via Jaeger
ManagedChannel jaegerChannel =
    ManagedChannelBuilder.forAddress([ip:String], [port:int]).usePlaintext().build();
JaegerGrpcSpanExporter jaegerExporter = JaegerGrpcSpanExporter.newBuilder()
    .setServiceName("example").setChannel(jaegerChannel).setDeadline(30000)
    .build();
tracerProvider.addSpanProcessor(
    BatchSpansProcessor.newBuilder(
        jaegerExporter
      ).build());
```



# Getting Started: Traces (Manual)

```
// Get the tracer
TracerSdkProvider tracerProvider = OpenTelemetrySdk.getTracerProvider();

// Configure the sampler to use
tracerProvider.updateActiveTraceConfig(
    TraceConfig alwaysOn = TraceConfig.getDefault().toBuilder().setSampler(
        Samplers.alwaysOn()
    ).build();
);

// Set to export the traces to via Jaeger
ManagedChannel jaegerChannel =
    ManagedChannelBuilder.forAddress([ip:String], [port:int]).usePlaintext().build();
JaegerGrpcSpanExporter jaegerExporter = JaegerGrpcSpanExporter.newBuilder()
    .setServiceName("example").setChannel(jaegerChannel).setDeadline(30000)
    .build();
tracerProvider.addSpanProcessor(
    BatchSpansProcessor.newBuilder(
        jaegerExporter
      ).build());
```

# Getting Started: Traces (Manual)

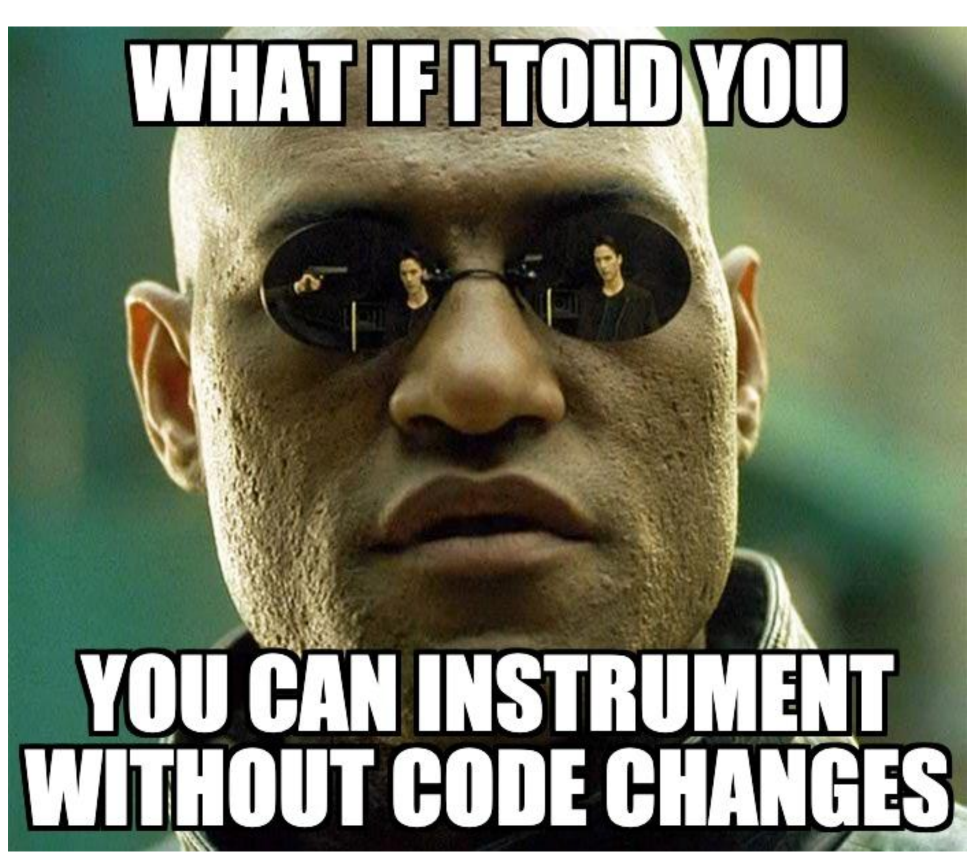
```
// Get the tracer
TracerSdkProvider tracerProvider = OpenTelemetrySdk.getTracerProvider();

// Configure the sampler to use
tracerProvider.updateActiveTraceConfig(
    TraceConfig alwaysOn = TraceConfig.getDefault().toBuilder().setSampler(
        Samplers.alwaysOn()
    ).build();
);

// Set to export the traces to via Jaeger
ManagedChannel jaegerChannel =
    ManagedChannelBuilder.forAddress([ip:String], [port:int]).usePlaintext().build();
JaegerGrpcSpanExporter jaegerExporter = JaegerGrpcSpanExporter.newBuilder()
    .setServiceName("example").setChannel(jaegerChannel).setDeadline(30000)
    .build();
tracerProvider.addSpanProcessor(
    BatchSpansProcessor.newBuilder(
        jaegerExporter
      ).build());
```

# Getting Started: Traces (Manual)

```
// Get the tracer
TracerSdkProvider tracerProvider = OpenTelemetrySdk.getTracerProvider();

// Configure the sampler to use
tracerProvider.updateActiveTraceConfig(
    TraceConfig alwaysOn = TraceConfig.getDefault().toBuilder().setSampler(
        Samplers.alwaysOn()
    ).build();
);

// Set to export the traces to via Jaeger
ManagedChannel jaegerChannel =
    ManagedChannelBuilder.forAddress([ip:String], [port:int]).usePlaintext().build();
JaegerGrpcSpanExporter jaegerExporter = JaegerGrpcSpanExporter.newBuilder()
    .setServiceName("example").setChannel(jaegerChannel).setDeadline(30000)
    .build();
tracerProvider.addSpanProcessor(
    BatchSpansProcessor.newBuilder(
        jaegerExporter
      ).build());
```

# Still with me?

# There must be an easier way…

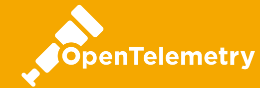

WHAT IF I TOLD YOU
YOU CAN INSTRUMENT
WITHOUT CODE CHANGES

# Getting Started: Traces (Automatic)

```
java -javaagent:path/to/opentelemetry-auto-<version>.jar \
     -Dota.exporter.jar=path/to/opentelemetry-auto-exporters-otlp-<version>.jar \
     -Dota.exporter.otlp.endpoint=localhost:55680 \
     -Dota.exporter.otlp.service.name=shopping \
     -jar myapp.jar
```

- Instruments known libraries with no code (only runtime) changes
- Adheres to semantic conventions
- Configurable via environment and/or runtime variables
- Can co-exist with manual instrumentation

**WARNING:** Do not use two different auto-instrumentation solutions on the same service.

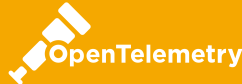


- Akka HTTP 10.0+
- Apache HttpAsyncClient 4.0+
- Apache HttpClient 2.0+
- AWS SDK 1.11.x and 2.2.0+
- Cassandra Driver 3.0+ (not 4.x yet)
- Couchbase Client 2.0+ (not 3.x yet)
- Dropwizard Views 0.7+
- Elasticsearch API 2.0+ (not 7.x yet)
- Elasticsearch REST Client 5.0+
- Finatra 2.9+
- Geode Client 1.4+
- Google HTTP Client 1.19+
- Grizzly 2.0+
- gRPC 1.5+
- Hibernate 3.3+
- HttpURLConnection Java 7+
- Hystrix 1.4+
- Java.util.logging Java 7+
- JAX-RS 0.5+
- JAX-RS Client 2.0+
- JDBC Java 7+
- Jedis 1.4+
- Jetty 8.0+
- JMS 1.1+
- JSP 2.3+
- Kafka 0.11+
- Lettuce 4.0+
- Log4j 1.1+
- Logback 1.0+
- MongoDB Drivers 3.3+
- Netty 3.8+
- OkHttp 3.0+
- Play 2.3+ (not 2.8.x yet)
- Play WS 1.0+
- RabbitMQ Client 2.7+
- Ratpack 1.5+
- Reactor 3.1+
- Rediscala 1.8+
- RMI Java 7+
- RxJava 1.0+
- Servlet 2.3+
- Spark Web Framework 2.3+
- Spring Data 1.8+
- Spring Scheduling 3.1+
- Spring Servlet MVC 3.1+
- Spring Webflux 5.0+
- Spymemcached 2.12+
- Twilio 6.6+

# OpenTelemetry Java auto-instrumentation library support

# Getting Started: Metrics

```java
// Instantiate a meter
Meter meter = OpenTelemetry.getMeter("instrumentation-library-name","semver:1.0.0");

// Create a metric
LongCounter counter = meter
        .longCounterBuilder("processed_jobs")
        .setDescription("Processed jobs")
        .setUnit("1")
        .build();


// Configure observer
observer.setCallback(
        new LongObserver.Callback<LongObserver.ResultLongObserver>() {
          @Override
          public void update(ResultLongObserver result) {
            // long getCpuUsage()
            result.observe(getCpuUsage(), "Key", "SomeWork");
          }
        });
```



# Demo!

# Other Project Aspects

- Governance Board
  - Code of conduct
  - Technical steering committee
- OpenTelemetry Enhancement Proposals (OTEPs)
  - OTLP protocol and support for HTTP
  - Log SIG
- Core versus Contrib
- Website (https://opentelemetry.io)

# Roadmap

- Rest of client libraries to beta ASAP
- Move to GA later this year for traces and metrics

- Tracing auto instrumentation for all languages
- Add initial log support (goal of beta later this year)

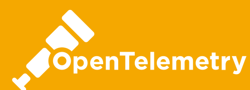
- Improve documentation
- Increase adoption; get case studies
- Make getting started really easy

# Next Steps

- Join the conversation: https://gitter.im/open-telemetry/community
- Join a SIG: https://github.com/open-telemetry/community#special-interest-groups
- Submit a PR (consider good-first-issue and help-wanted labels)
  - I will be submitting a PR for this template!

# Links

- Specification
  - https://github.com/open-telemetry/opentelemetry-specification
- OpenTelemetry Collector
  - https://opentelemetry.io/docs/collector/about/
  - https://opentelemetry.io/docs/collector/configuration/
- Java client library
  - https://github.com/open-telemetry/opentelemetry-java/blob/master/QUICKSTART.md
  - https://github.com/open-telemetry/opentelemetry-auto-instr-java
- Other
  - https://opentelemetry.io/docs/workshop/resources/
  - https://devstats.cncf.io/
  - https://medium.com/jaegertracing/jaeger-embraces-opentelemetry-collector-90a545cbc24
  - https://github.com/spring-petclinic/spring-petclinic-microservices



# Thank You!