



How I stopped worrying and learned to love
microservices in Golang

Before We Begin

- This workshop is designed to be done locally on your computer
- This workshop requires Docker and Docker Compose
- It can work on Linux/Mac or Windows but is optimized for Linux/Mac
- Please clone <https://github.com/vvydier/otel-golang-meetup>
- Run (this may take a while!):

```
cd otel-golang-meetup/otel-docker && \  
  docker-compose up
```

```
cd otel-golang-meetup && \ ls app app-with-otel
```



Vinod Vydier



Observability Specialist at Splunk
OpenTelemetry Contributor and Approver
ex-Observe, ex-New Relic

Agenda

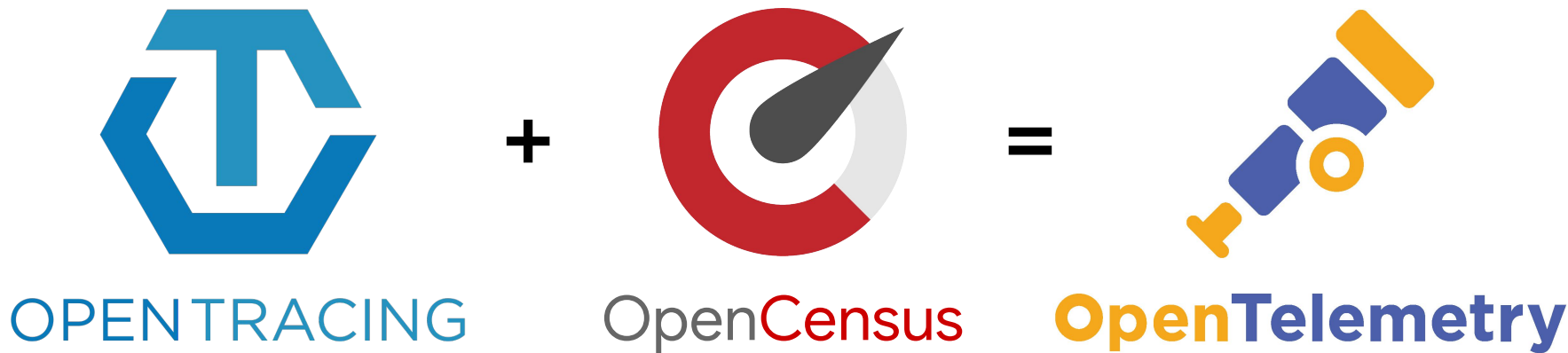
- What is Observability?
- How does OpenTelemetry(OTel) relate to Observability?
- What concepts do I need for OTel
 - Introduction to Instrumentation and Tracing
 - Metrics and Logs
- Introduction to OTel Collector
- How do I record telemetry and where can I send my data?
- Golang Lab
 - 101 (App)
 - 102 (Otel-collector)
 - 103 (App with OTel)
- Review
- AMA

Level Setting

- Are you responsible for writing software?
 - Are you responsible for operating software?
-
- Have you used APM (Application Performance Management) before?
 - Have you used distributed tracing before?
 - Have you used logging before?
 - Have you used metrics before?
 - Have you used OpenTelemetry or written instrumentation before?

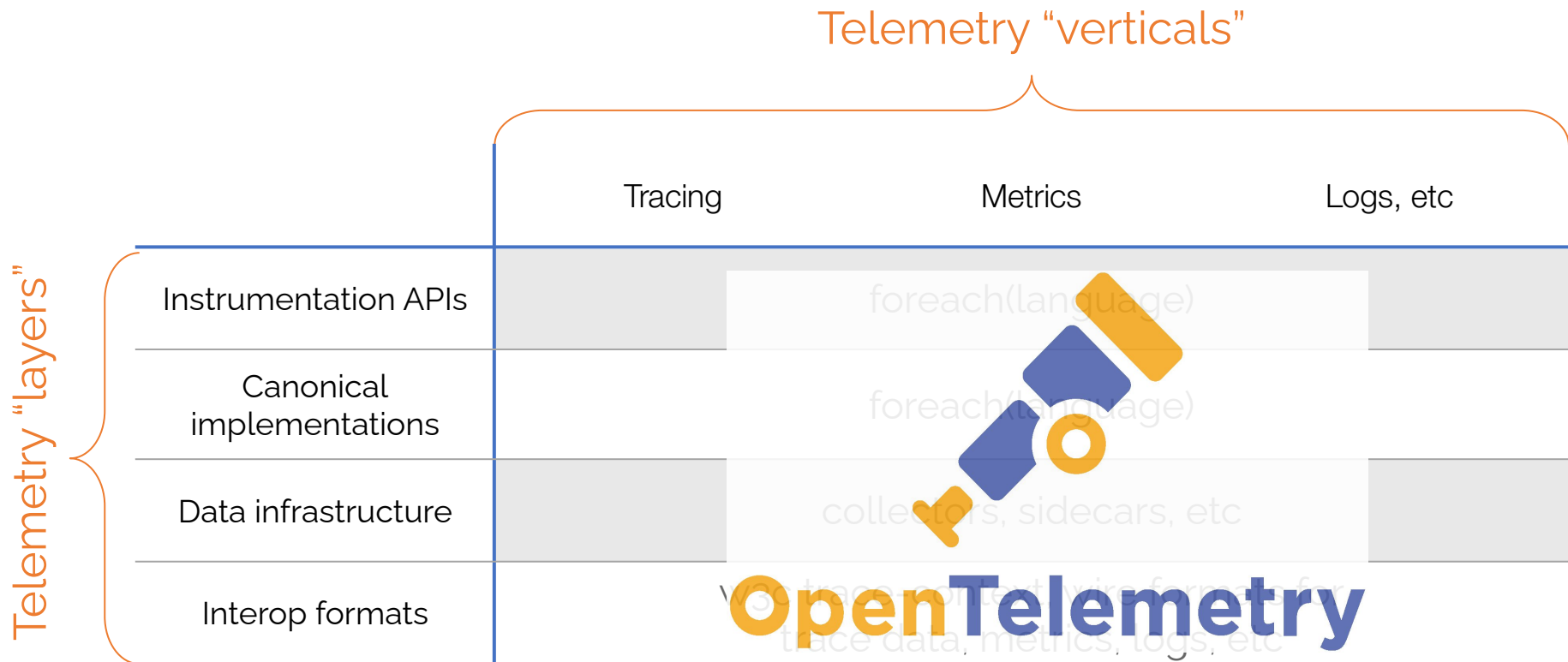
Introduction

What is Observability and OpenTelemetry?



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

Cloud Native Telemetry



OpenTelemetry Components

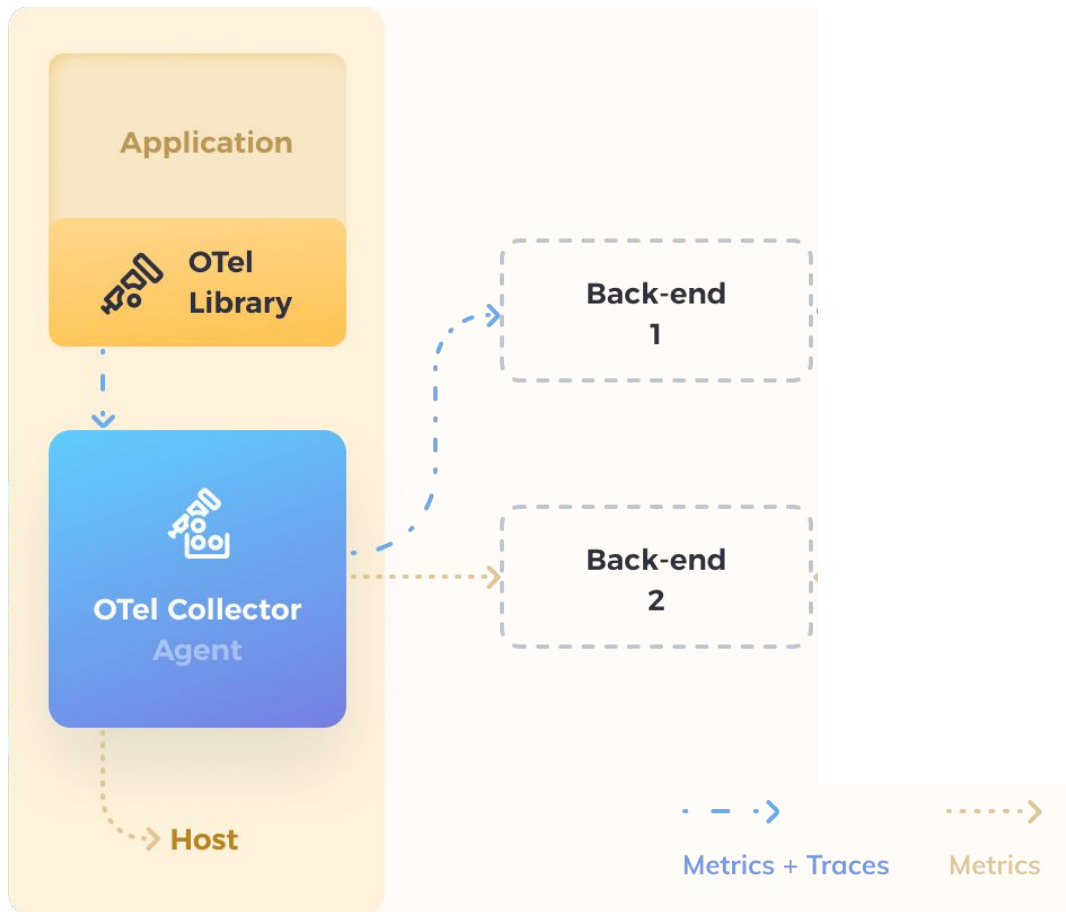


**INSTRUMENTATION
LIBRARIES**
Single implementation
per language

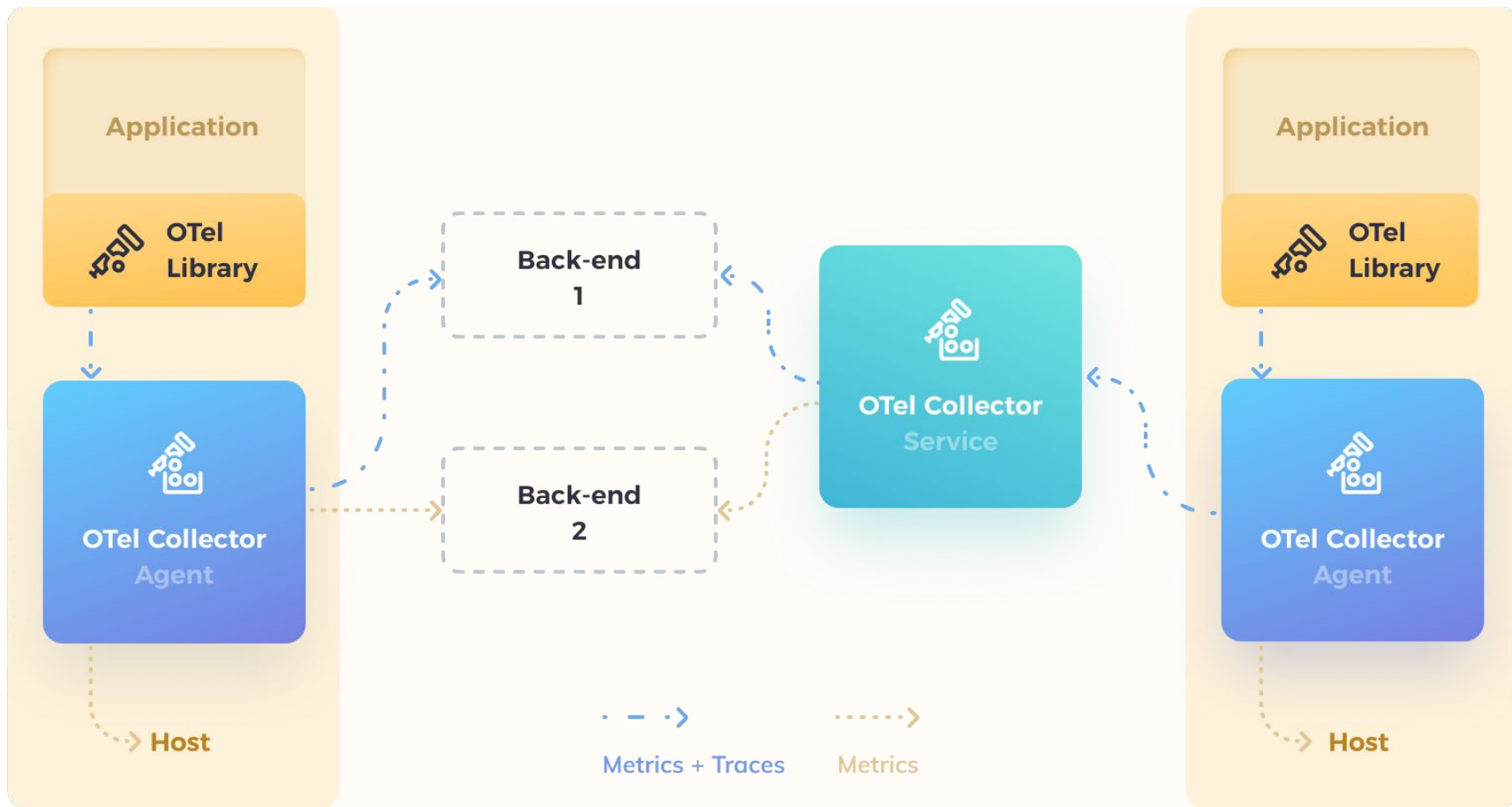
COLLECTOR
Single binary to receive,
process, and export data



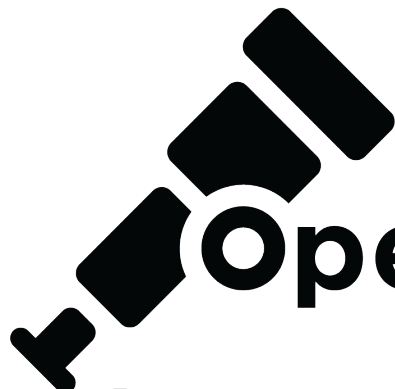
SPECIFICATION
API, SDK, Data



Reference architecture



Reference architecture



OpenTelemetry

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

(per CNCF DevStats)

Everyone is Contributing and Adopting

Cloud Providers



AWS | Azure | GCP

Vendors



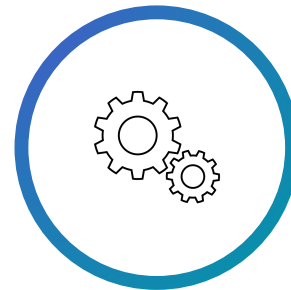
Every major vendor!
Splunk, AWS,
MSFT, DD, NR, DT

End-users



Mailchimp (PHP)
Postmates (Erlang)
Shopify (Ruby)

Other



Jaeger > OtelCol
Fluent-bit <3 log SIG
Envoy roadmap
OpenMetrics roadmap
Spring roadmap

<https://github.com/open-telemetry/community/blob/master/ADOPTERS.md>
<https://medium.com/jaegertracing/jaeger-embraces-opentelemetry-collector-90a545cbcb24>
<https://aws-otel.github.io/docs/partners/splunk>

Instrumentation



Instrumentation Libraries

Application



php



RUM

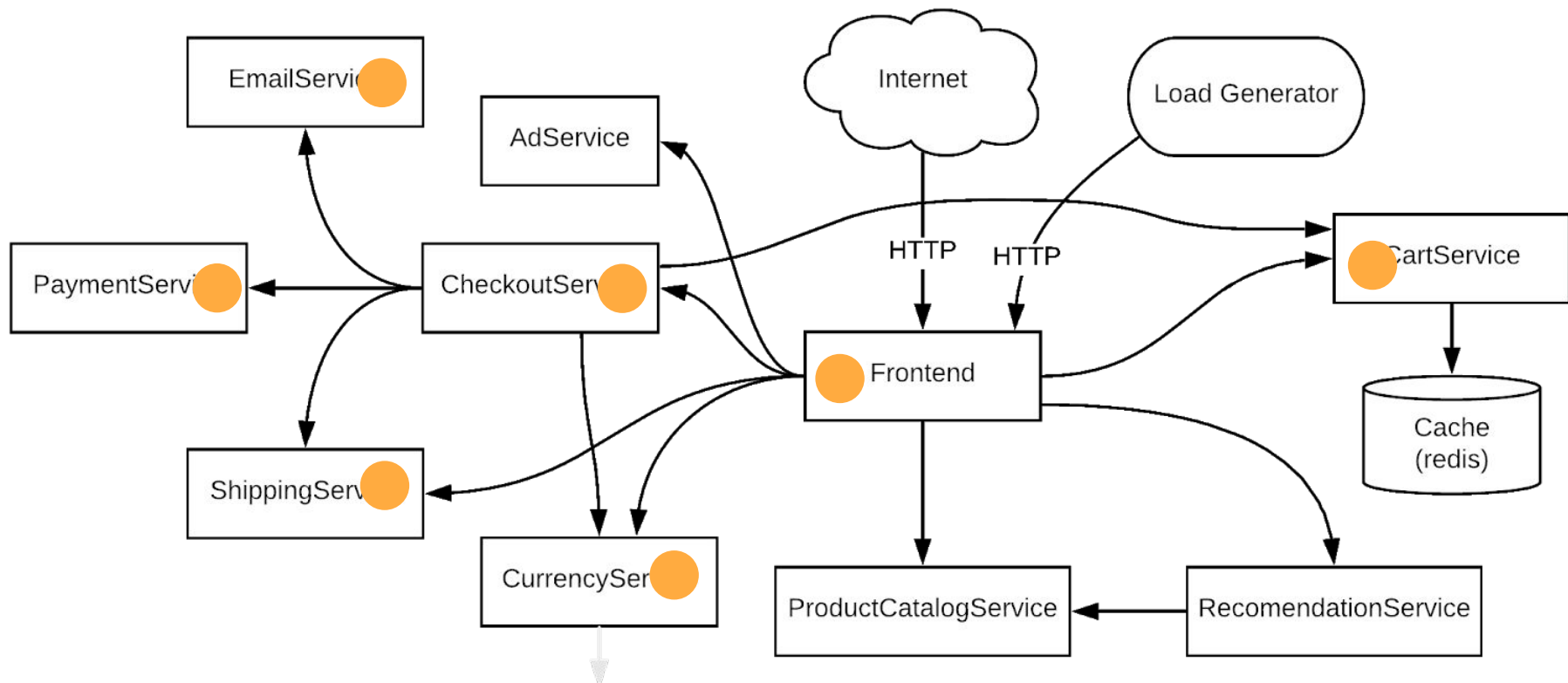


Serverless



- Single instrumentation library per language
- All support manual instrumentation; some automatic
- Priority 1) traces 2) metrics 3) logs

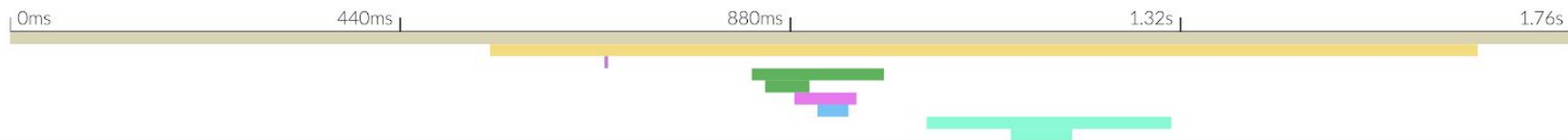
What is distributed tracing?



Distributed trace request visualization

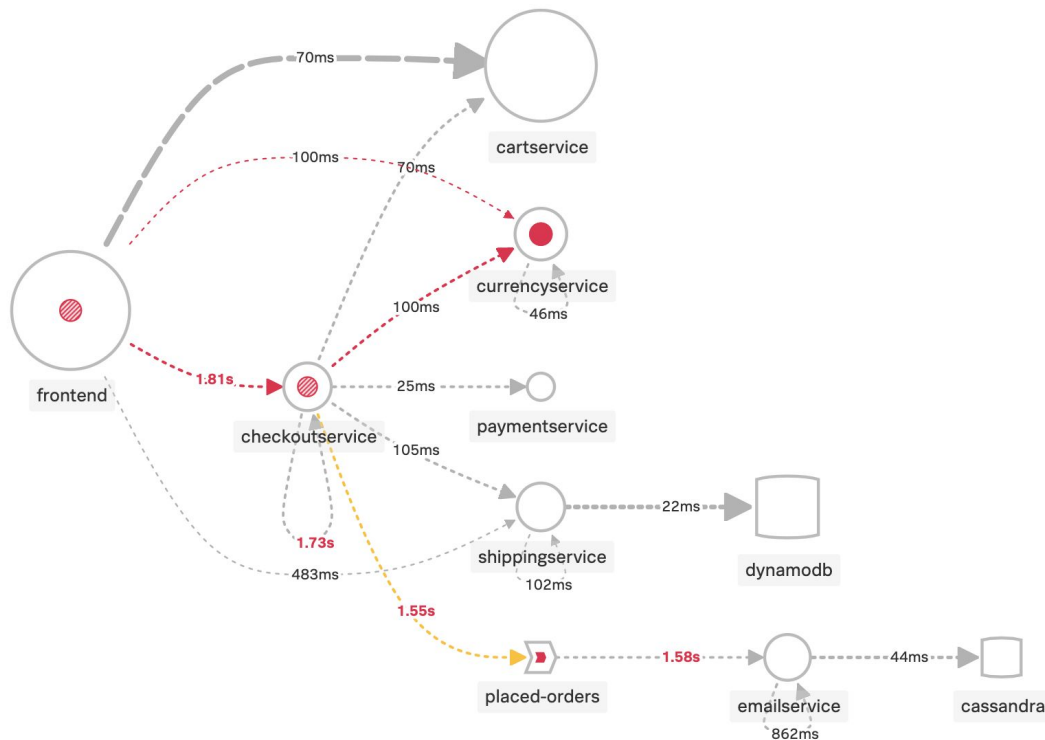
frontend >
/checkout

2 minutes ago
Today at 12:23:03 PM



SERVICE	OPERATION	0ms	440ms	880ms	1.32s	1.76s
▼ frontend	/checkout	1.76s				
▼ checkoutservice	/PlaceOrder		1.11s			
cartservice	/GetCart		4ms			
▼ currencyservice	/GetConversion			149ms		
currencyservice	/Money			50ms		
shippingservice	/Address			70ms		
paymentservice	/CreditCardInfo			35ms		
▼ emailservice	/SendOrderConfirmation			276ms		
emailservice	/OrderResult			69ms		

Distributed trace service map visualization



Lab 101, 102, 103

Review

What did we accomplish?

- Instrumented an application with tracing
- Experienced exporters and configuration

What are the basic steps (language specific)?

- Download/Install dependencies
- Configure environment variables
- Update runtime parameters

Tracing Basics

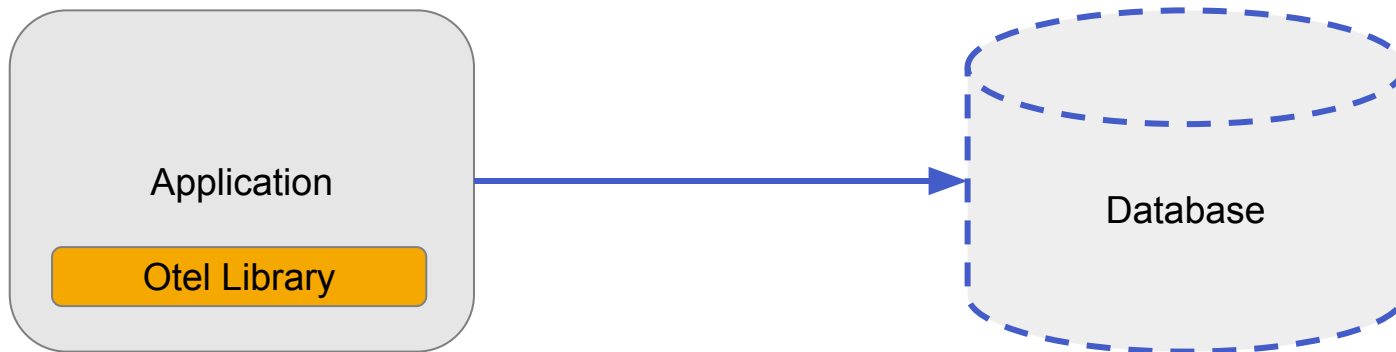
- **Context:** W3C trace-context, B3, etc.
- **Tracer:** get context
- **Spans:** “call” in a trace
 - **Kind:** client/server, producer/consumer, internal
 - **Attributes:** key/value pairs; tags; metadata
 - **Events:** named strings
 - **Links:** useful for batch operations
- **Sampler:** always,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



Resource SDK + Semantic Conventions

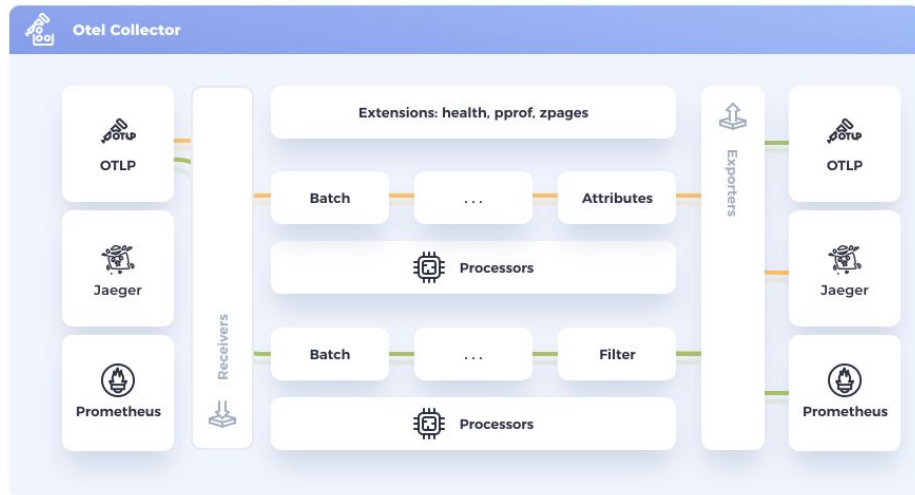
A Resource is an immutable representation of the entity producing telemetry. For example, a process producing telemetry that is running in a container on Kubernetes has a Pod name, it is in a namespace and possibly is part of a Deployment. All three of these attributes can be included in the Resource.

- **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

How is the OTEL Collector architected?

<https://opentelemetry.io/docs/collector/>



- Components
 - **Receivers:** how you get data in (can be push or pull-based)
 - **Processors:** what you do to the data (e.g. batching, metadata, etc.)
 - **Exporters:** how you get data out (can be push or pull-based)
 - **Extensions:** things you do in the collector typically outside processing data (e.g. health check)
- Configuration is done in YAML and consists of two steps:
 - Define component configuration
 - Enable the component

How is the OTel Collector configured?

<https://opentelemetry.io/docs/collector/configuration/>

```
receivers:
  host:
    scrapers: [cpu, disk, memory]
  zipkin:
processors:
  batch:
exporters:
  jaeger:
    endpoint: jaeger-all-in-one:14250
  prometheus:
    endpoint: prometheus:9091
service:
  pipelines:
    metrics:
      receivers: [host]
      processors: [batch]
      exporters: [prometheus]
    traces:
      receivers: [zipkin]
      processors: [batch]
      exporters: [jaeger]
```

How to configure components. Many components come with default configuration baked in.

How to enable components. Note the order of processors matters!

Review(cont.)

What did we accomplish?

- Configuring the collector
- CRUD metadata via collector

What are the basic steps (language specific)?

- Configure component (receiver, processor, exporter)
- Enable component (pipeline)

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>)

Next Steps

- Join the conversation: <https://cloud-native.slack.com>
- Join a SIG:
<https://github.com/open-telemetry/community#special-interest-groups>
- Submit a PR (consider [good-first-issue](#) and [help-wanted](#) labels)

Links

- Specification
 - <https://github.com/open-telemetry/opentelemetry-specification>
- OpenTelemetry Collector
 - <https://opentelemetry.io/docs/collector/about/>
 - <https://opentelemetry.io/docs/collector/configuration/>
- Golang client library
 - <https://github.com/open-telemetry/opentelemetry-go>
 - <https://opentelemetry.io/ecosystem/registry/?language=go&component=instrumentation>
 - <https://github.com/open-telemetry/opentelemetry-go-contrib>
 - <https://github.com/open-telemetry/opentelemetry-go-instrumentation>
- Java client library
 - <https://github.com/open-telemetry/opentelemetry-java/blob/master/QUICKSTART.md>
 - <https://github.com/open-telemetry/opentelemetry-java-instrumentation>
- Other
 - <https://opentelemetry.io/docs/workshop/resources/>
 - <https://devstats.cncf.io/>
 -

Thank You!

AMA