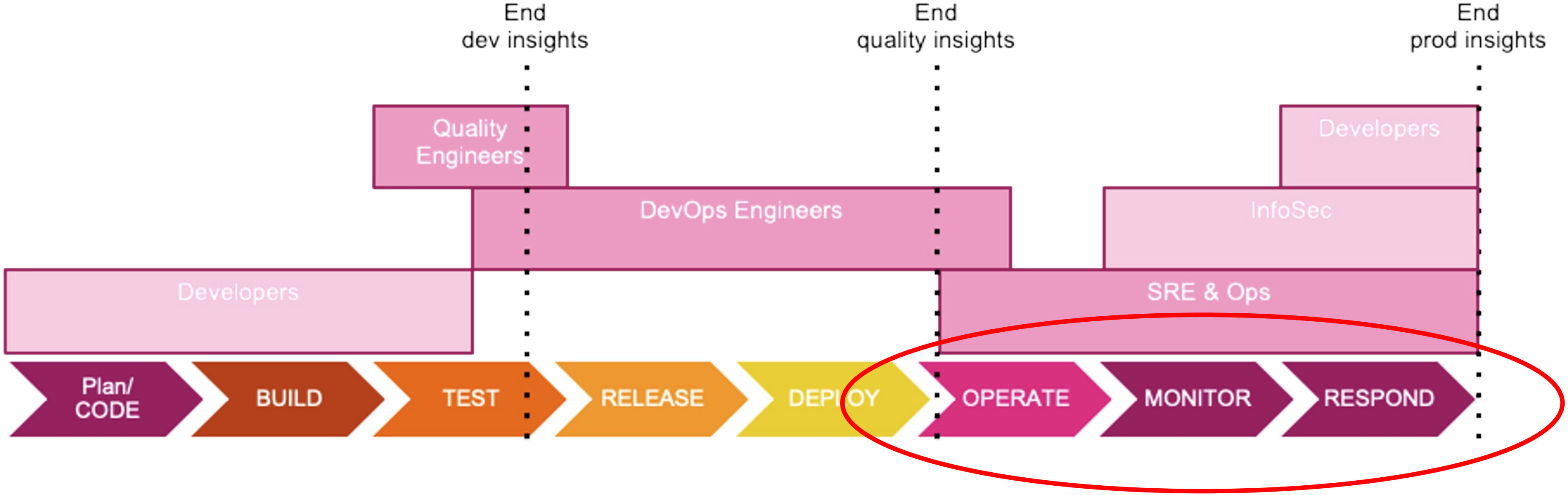
Getting started with Observability & OpenTelemetry

Mar 20, 2022

Who and When in Agile Ops



Area of focus today!

Agenda

- Observability?
- OpenTelemetry?
- Demo in Action!

Introduction

wwongpairoj@splunk.com

wwongpai

donler

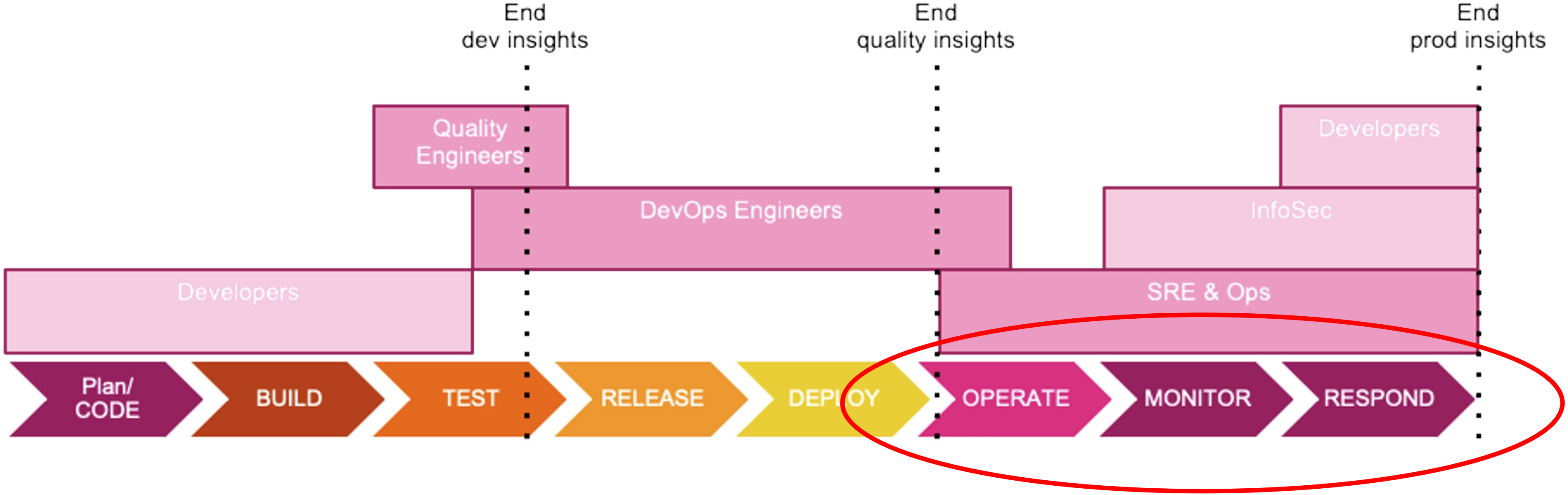
Warach Wongpairoj

Senior Sales Engineer, Splunk

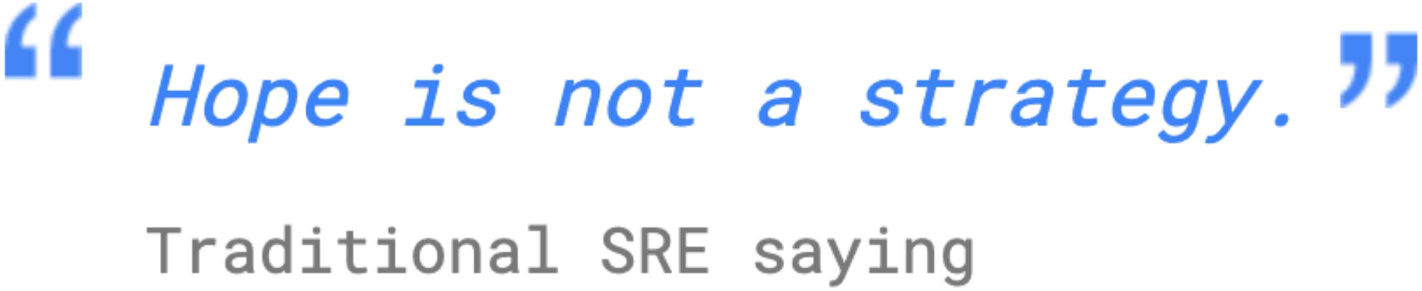
Previously:

- Senior Sales Engineer, AppDynamics
- Sales Engineer, Oracle Cloud (Cloud Manageability & Cloud Security)

Who and When in Agile Ops



Area of focus today!



> “*Hope is not a strategy.*”
>
> Traditional SRE saying

It is a truth universally acknowledged that systems do not run themselves. How, then, should a system—particularly a complex computing system that operates at a large scale—be run?

Reference - https://sre.google/sre-book/introduction/

Goal & Responsibility

To addresses how a complex computing system operating at a large scale should run to achieve scalable and highly reliable software systems.

Responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their service(s).

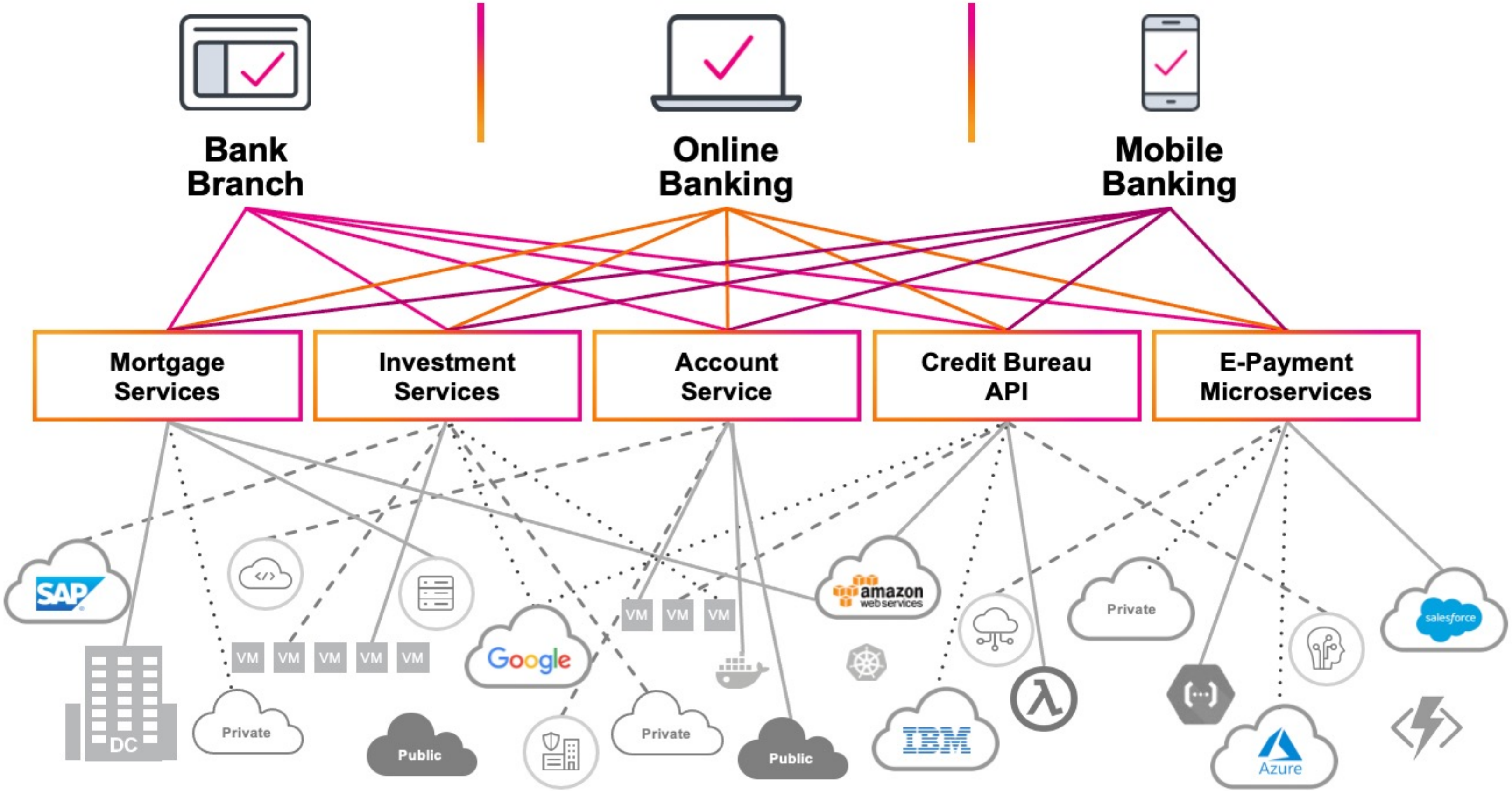
Goal & Responsibility ……in reality

To know who to wake up and how to find root cause

Which customer or action is impacted?

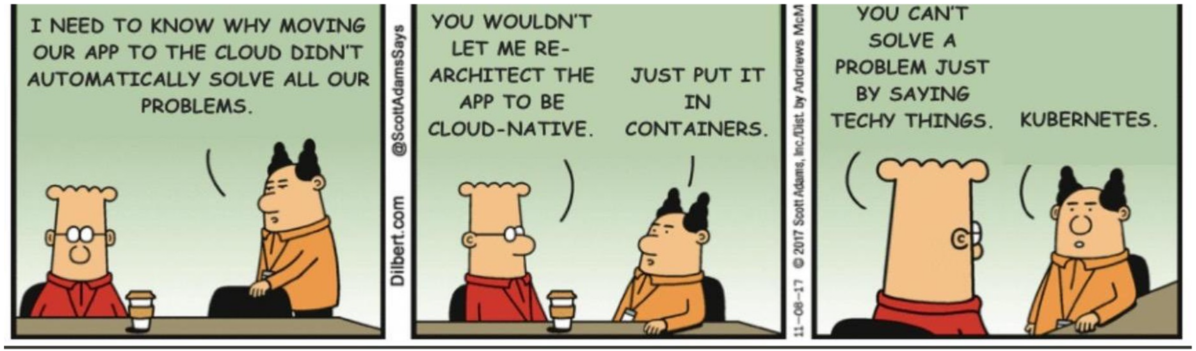
MTTD, MTTI, MTTR, Error Budget
and so on……

Challenges


Bank Branch
Online Banking
Mobile Banking
Mortgage Services
Investment Services
Account Service
Credit Bureau API
E-Payment Microservices
SAP
DC
Private
VM
VM
VM
VM
VM
Public
Google
VM
VM
VM
Private
Public
amazon web services
IBM
Private
salesforce
Azure

To Summarize Challenges --> Why Obeservability is needed

- Microservices create complex interactions
- To mange system at scale is difficult
- Failures don’t exactly repeat
- Debugging multi-tenancy is painful
- Traditional monitoring approach can’t save us


I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.
YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE.
JUST PUT IT IN CONTAINERS.
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS.
KUBERNETES.

Dilbert.com @ScottAdamsSays

11-08-17 © 2017 Scott Adams, Inc./Dist. by Andrews McM

What is Observability?

- We need to answer questions about our systems.

What characteristics did the queries that timed out at 500ms share in common? Service versions? Browser plugins?

- Instrumentation produces data.
- Querying data answers our questions.

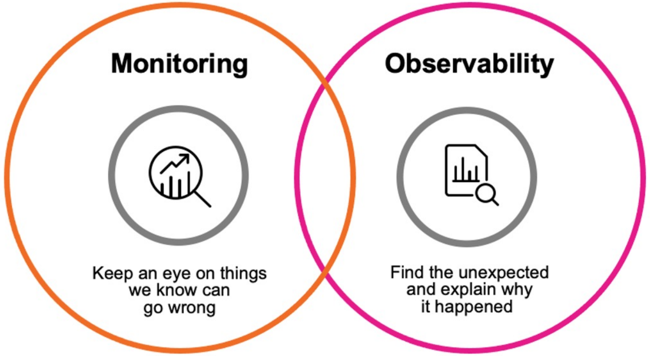
What difference btw Monitoring & Observability?

Is it just a fancy word for "Monitoring"?

Looking for expected problems, e.g.:

- Overloaded CPU
- High memory utilization
- Disk space
- High response latency
- High error rate
- Service availability

Alerting on issues that have occurred.


Monitoring
Keep an eye on things we know can go wrong
Observability
Find the unexpected and explain why it happened

Looking for unknown unknowns, e.g.:

- Why are the alerts firing?
- What is in common between problem areas?
- Find the "needle in the haystack"
- Troubleshooting solution for complex systems

Diagnosing why issues have occurred.

But how do I implement these?

- You need data (3 pillars of observability data)
- You need an instrumentation framework!
- and a place to send the data!
- and a way to visualize the data!

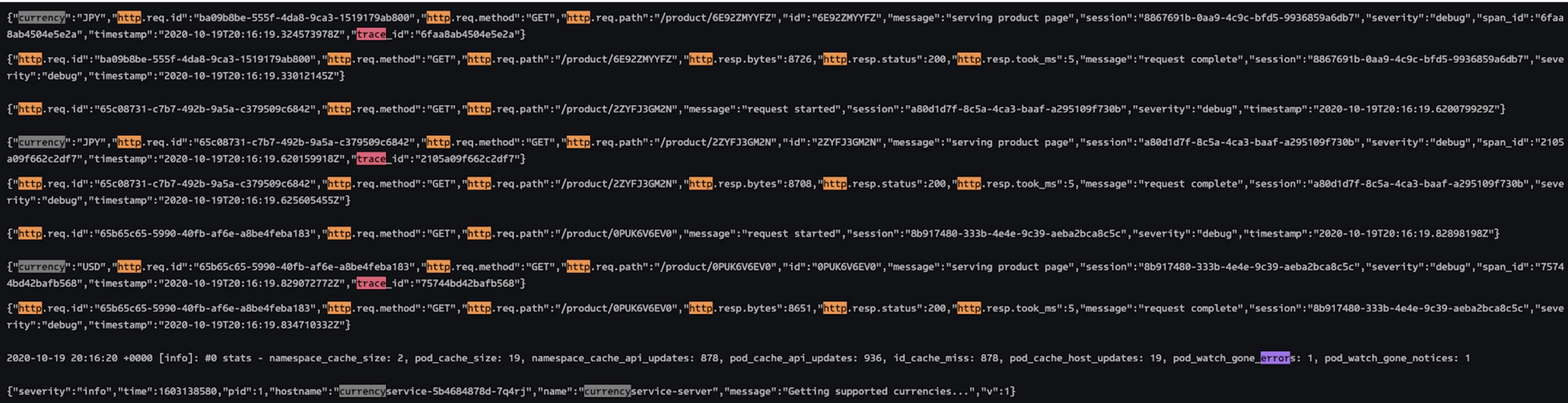
LOGS

Detailed debugging information emitted by processes

- A time stamped text record
- Can be structured (recommended) or unstructured
- Enhanced with metadata

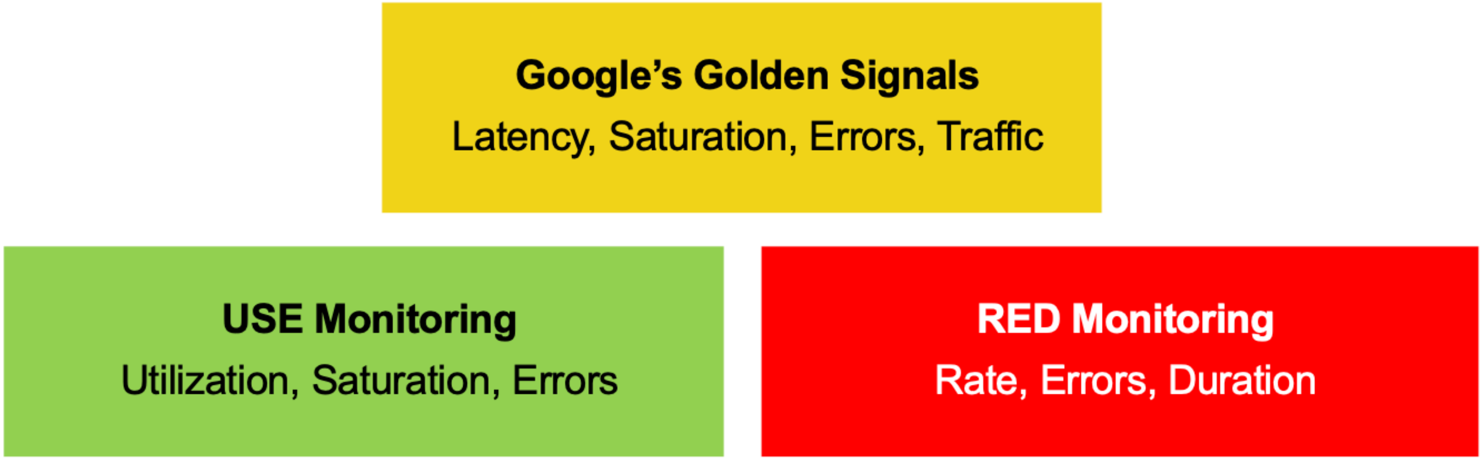
Data format: LOGS

OSS: Graylog, Elastics, Fluentd, Fluentbit



```
{"currency":"JPY","http.req.id":"ba09b8be-555f-4da8-9ca3-1519179ab800","http.req.method":"GET","http.req.path":"/product/6E92ZMYYFZ","id":"6E92ZMYYFZ","message":"serving product page","session":"8867691b-0aa9-4c9c-bfd5-9936859a6db7","severity":"debug","span_id":"6faa
8ab4504e5e2a","timestamp":"2020-10-19T20:16:19.324573978Z","trace_id":"6faa8ab4504e5e2a"}

{"http.req.id":"ba09b8be-555f-4da8-9ca3-1519179ab800","http.req.method":"GET","http.req.path":"/product/6E92ZMYYFZ","http.resp.bytes":8726,"http.resp.status":200,"http.resp.took_ms":5,"message":"request complete","session":"8867691b-0aa9-4c9c-bfd5-9936859a6db7","seve
rity":"debug","timestamp":"2020-10-19T20:16:19.33012145Z"}

{"http.req.id":"65c08731-c7b7-492b-9a5a-c379509c6842","http.req.method":"GET","http.req.path":"/product/2ZYFJ3GM2N","message":"request started","session":"a80d1d7f-8c5a-4ca3-baaf-a295109f730b","severity":"debug","timestamp":"2020-10-19T20:16:19.620079929Z"}

{"currency":"JPY","http.req.id":"65c08731-c7b7-492b-9a5a-c379509c6842","http.req.method":"GET","http.req.path":"/product/2ZYFJ3GM2N","id":"2ZYFJ3GM2N","message":"serving product page","session":"a80d1d7f-8c5a-4ca3-baaf-a295109f730b","severity":"debug","span_id":"2105
a09f662c2df7","timestamp":"2020-10-19T20:16:19.620159918Z","trace_id":"2105a09f662c2df7"}

{"http.req.id":"65c08731-c7b7-492b-9a5a-c379509c6842","http.req.method":"GET","http.req.path":"/product/2ZYFJ3GM2N","http.resp.bytes":8708,"http.resp.status":200,"http.resp.took_ms":5,"message":"request complete","session":"a80d1d7f-8c5a-4ca3-baaf-a295109f730b","seve
rity":"debug","timestamp":"2020-10-19T20:16:19.625605455Z"}

{"http.req.id":"65b65c65-5990-40fb-af6e-a8be4feba183","http.req.method":"GET","http.req.path":"/product/0PUK6V6EV0","message":"request started","session":"8b917480-333b-4e4e-9c39-aeba2bca8c5c","severity":"debug","timestamp":"2020-10-19T20:16:19.82898198Z"}

{"currency":"USD","http.req.id":"65b65c65-5990-40fb-af6e-a8be4feba183","http.req.method":"GET","http.req.path":"/product/0PUK6V6EV0","id":"0PUK6V6EV0","message":"serving product page","session":"8b917480-333b-4e4e-9c39-aeba2bca8c5c","severity":"debug","span_id":"7574
4bd42bafb568","timestamp":"2020-10-19T20:16:19.829072772Z","trace_id":"75744bd42bafb568"}

{"http.req.id":"65b65c65-5990-40fb-af6e-a8be4feba183","http.req.method":"GET","http.req.path":"/product/0PUK6V6EV0","http.resp.bytes":8651,"http.resp.status":200,"http.resp.took_ms":5,"message":"request complete","session":"8b917480-333b-4e4e-9c39-aeba2bca8c5c","seve
rity":"debug","timestamp":"2020-10-19T20:16:19.834710332Z"}

2020-10-19 20:16:20 +0000 [info]: #0 stats - namespace_cache_size: 2, pod_cache_size: 19, namespace_cache_api_updates: 878, pod_cache_api_updates: 936, id_cache_miss: 878, pod_cache_host_updates: 19, pod_watch_gone_errors: 1, pod_watch_gone_notices: 1

{"severity":"info","time":1603138580,"pid":1,"hostname":"currencyservice-5b4684878d-7q4rj","name":"currencyservice-server","message":"Getting supported currencies...","v":1}
```

The Golden Signals/Telemetry



Google's Golden Signals

Latency, Saturation, Errors, Traffic

USE Monitoring

Utilization, Saturation, Errors

RED Monitoring

Rate, Errors, Duration

USE

Utilization, Saturation, Error

For every resource, check:

- Utilization

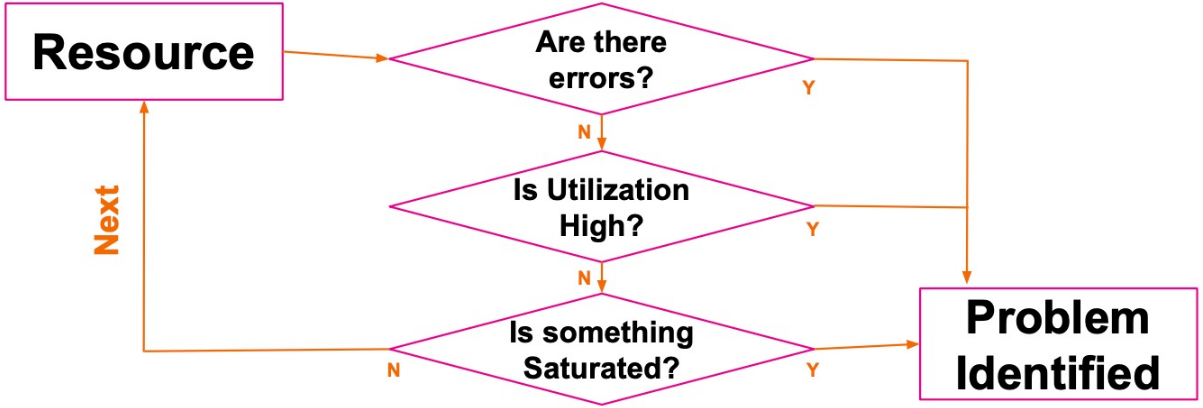
  How busy is the resource or amount in use
- Saturation

  How much extra work is not being process due to lack of resource
- Errors

  All errors

USE

Utilization, Saturation, Error



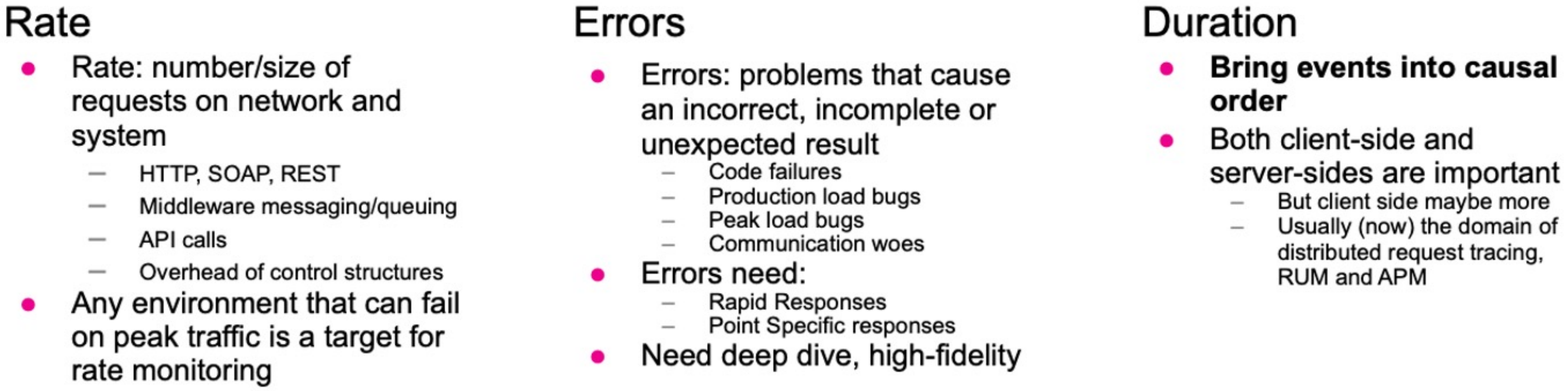
Data format: METRICS (Time-Series Metrics)

OSS: Prometheus, Zabbix, Nagios, Cacti, Collectd etc.

So How About RED?

Rate, Error, Duration

- Designed for request-driven systems, microservices



Rate

- Rate: number/size of requests on network and system
  - HTTP, SOAP, REST
  - Middleware messaging/queuing
  - API calls
  - Overhead of control structures
- Any environment that can fail on peak traffic is a target for rate monitoring

Errors

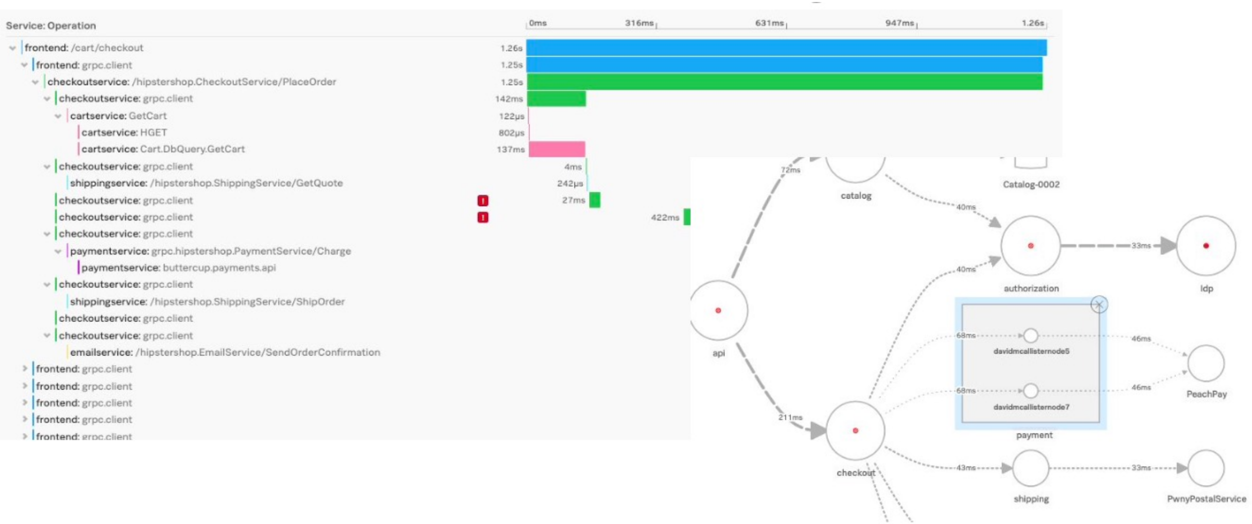
- Errors: problems that cause an incorrect, incomplete or unexpected result
  - Code failures
  - Production load bugs
  - Peak load bugs
  - Communication woes
- Errors need:
  - Rapid Responses
  - Point Specific responses
- Need deep dive, high-fidelity

Duration

- **Bring events into causal order**
- Both client-side and server-sides are important
  - But client side maybe more
  - Usually (now) the domain of distributed request tracing, RUM and APM

RED

Rate, Error, Duration



Service: Operation		
frontend: /cart/checkout		1.26s
frontend: grpc.client		1.25s
checkoutservice: /hipstershop.CheckoutService/PlaceOrder		1.25s
checkoutservice: grpc.client		142ms
cartservice: GetCart		122μs
cartservice: HGET		802μs
cartservice: Cart.DbQuery.GetCart		137ms
checkoutservice: grpc.client		4ms
shippingservice: /hipstershop.ShippingService/GetQuote		242μs
checkoutservice: grpc.client	!	27ms
checkoutservice: grpc.client	!	422ms
checkoutservice: grpc.client		
paymentservice: grpc.hipstershop.PaymentService/Charge		
paymentservice: buttercup.payments.api		
checkoutservice: grpc.client		
shippingservice: /hipstershop.ShippingService/ShipOrder		
checkoutservice: grpc.client		
checkoutservice: grpc.client		
emailservice: /hipstershop.EmailService/SendOrderConfirmation		
frontend: grpc.client		
frontend: grpc.client		
frontend: grpc.client		
frontend: grpc.client		
frontend: grpc.client		

Data format: TRACES

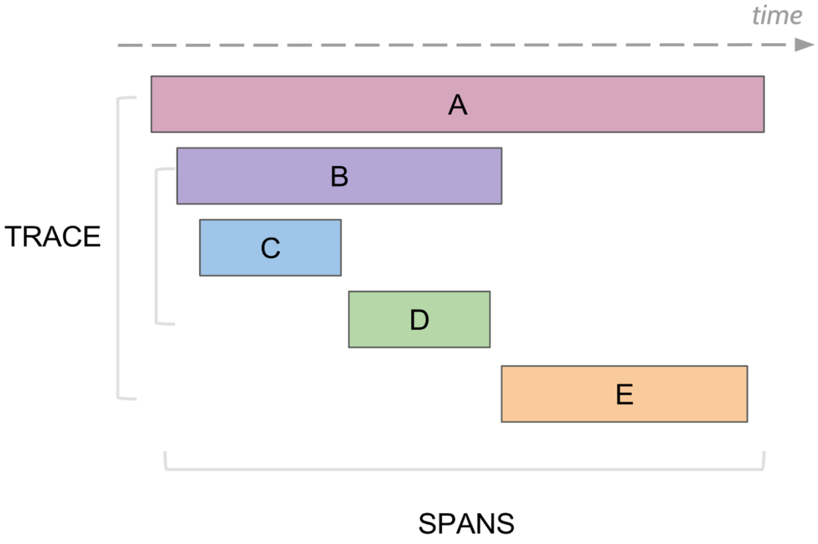
OSS: OpenTracing, Jaeger, Zipkin etc.

RED == Distributed Tracing

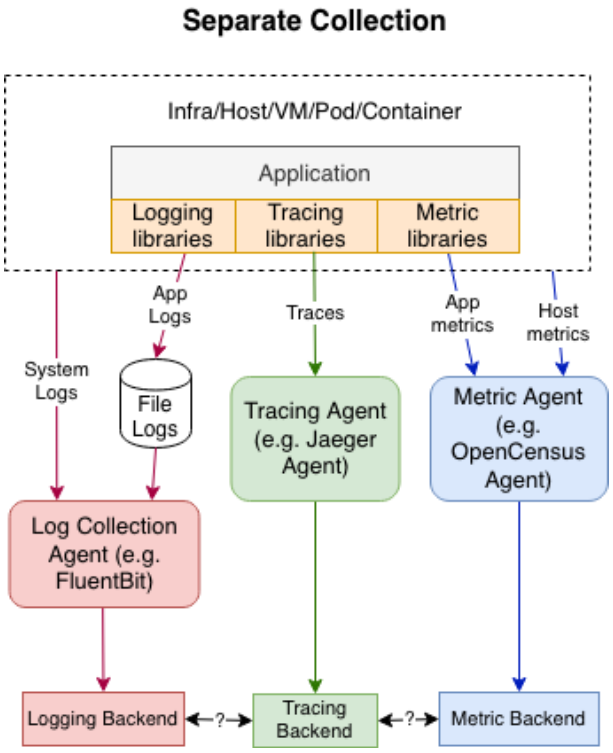
Traces

Distributed tracing traverses process, network and security boundaries

- Tracing is a way to record all the operations involved in the handling of a request or transaction through the entire application stack and backend infrastructure
- Spans record individual operations or RPCs, in particular their name, how long they took and where they took place
- Distributed Context contains tracing identifiers, tags, and options that propagated from parent to child spans



Finally, Log, Metric (USE), Trace (RED)..... Done?



Challenges

- Lack of standardization
- Some are vendor-lockin
- Lack of data portability
- The burden on the user to maintain the instrumentation
- Difficult to fix issue with 3 format of data are isolated
- No correlation & Causation

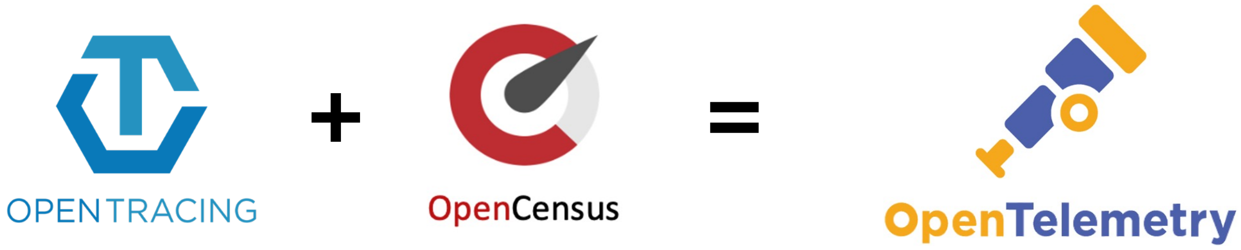
- You need an instrumentation framework!
- Consolidate data (from different format and correlate all logs, metrics and traces)
- Open-standard semantic conventions
- Context-propagation
- Vendor-agnostic

OpenTelemetry

What’s OpenTelemetry

The Cloud Native Computing Foundation (CNCF) OpenTelemetry project provides a single set of open source APIs, libraries, and agents to collect and correlate distributed traces, metrics and logs.

With OpenTelemetry you can instrument your application in a vendor-agnostic way, and then analyze the telemetry data in your backend tool of choice, whether Splunk ,Prometheus, Jaeger or others. Instrument once, and use it anywhere.



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!

End-users

Mailchimp (PHP)

Postmates (Erlang)

Shopify (Ruby)

Other

Jaeger > OtelCol

Fluent-bit <3 log SIG

Envoy roadmap

OpenMetrics roadmap

Spring roadmap

Cloud Native Telemetry

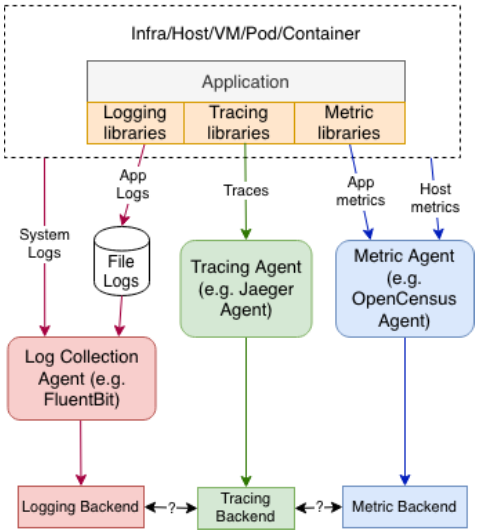
Telemetry “verticals”

Telemetry “layers”

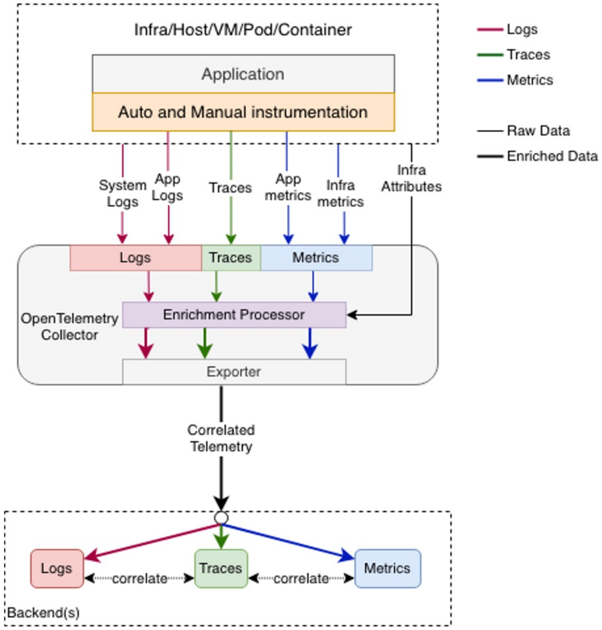
	Tracing	Metrics	Logs, etc
Instrumentation APIs	foreach(language)		
Canonical implementations	foreach(language)		
Data infrastructure	collectors, sidecars, etc		
Interop formats	w3c trace-context, wire formats for trace data, metrics, logs, etc		

From Separate to Consolidate

Separate Collection



OpenTelemetry Collection



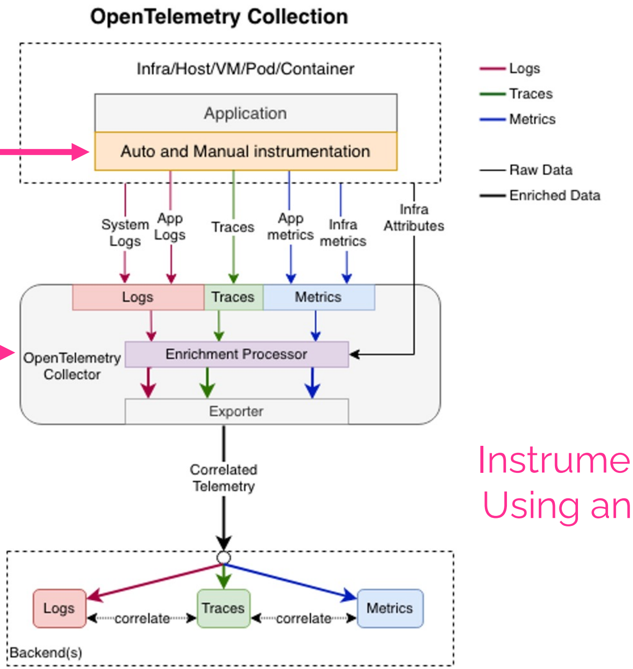
OpenTelemetry is NOT

- It’s not an observability back-end like Jaeger or Prometheus.
- Instead it supports exporting data to a variety of open-source and commercial back-ends.

OpenTelemetry Instrumentation Library

OpenTelemetry Collector

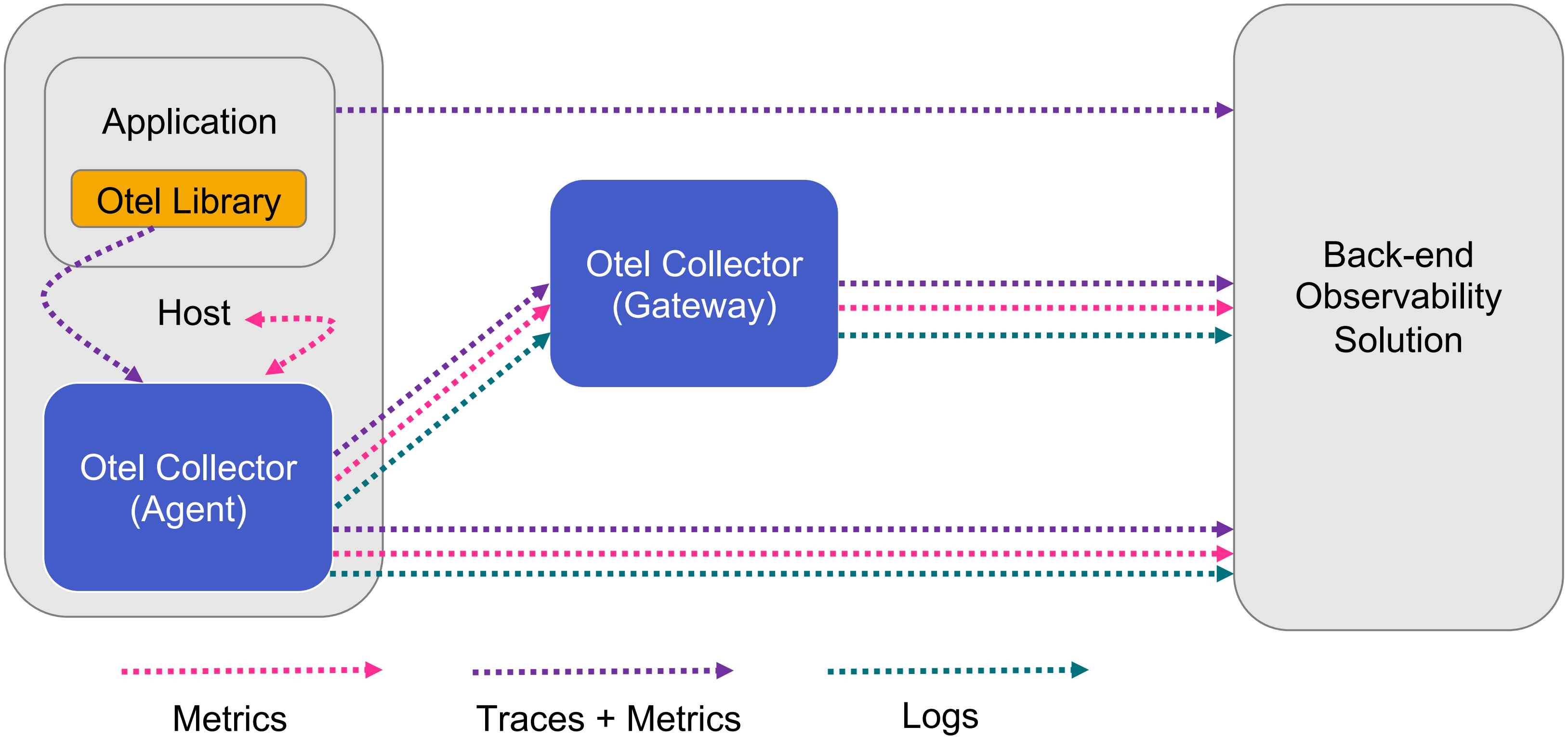
Back-end of choice



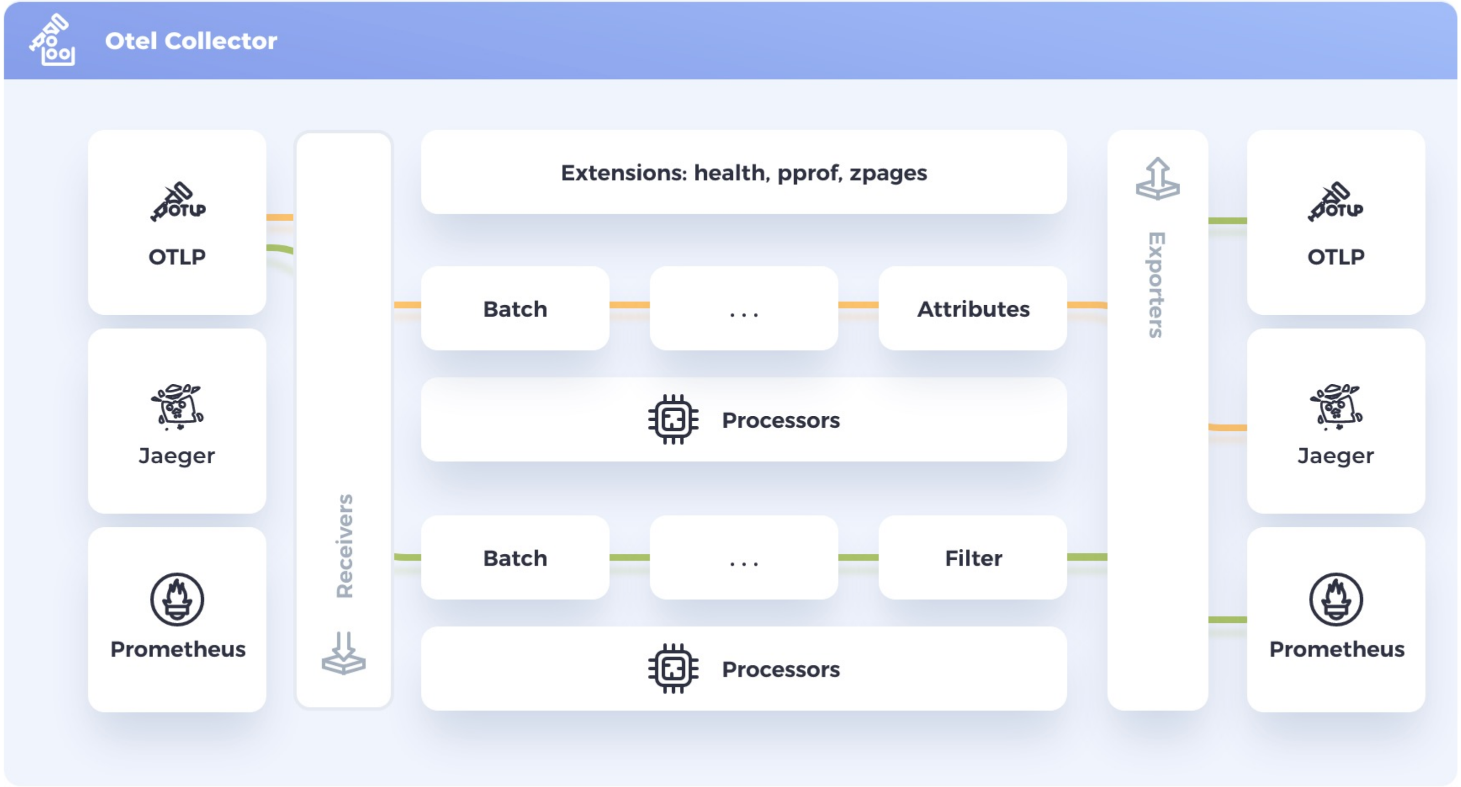
Instrument Once, Using anywhere!

Before DEMO

Reference Architecture: OpenTelemetry



Reference Architecture: OpenTelemetry



Example of 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

Example of Contrib (Community) Components

Traces

- Processors
  - Kubernetes
- Exporters:
  - Alibaba
  - AWS X-ray
  - Azure Monitor
  - Elastic
  - Honeycomb
  - Kinesis
  - Lightstep
  - New Relic
  - Splunk
  - Stackdriver

Metrics

- Receivers
  - Carbon
  - Kubernetes
  - Redis
  - Splunk
  - Wavefront
- Processors
  - Metrics Transform
- Exporters
  - Carbon
  - Splunk
  - 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());
```

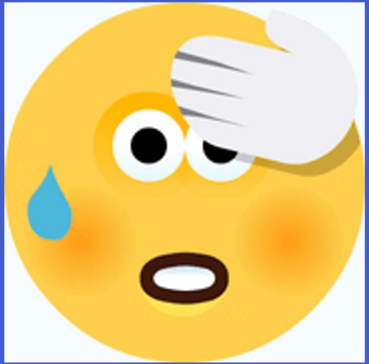
Getting Started: Traces (Manual)

```
// 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());
```

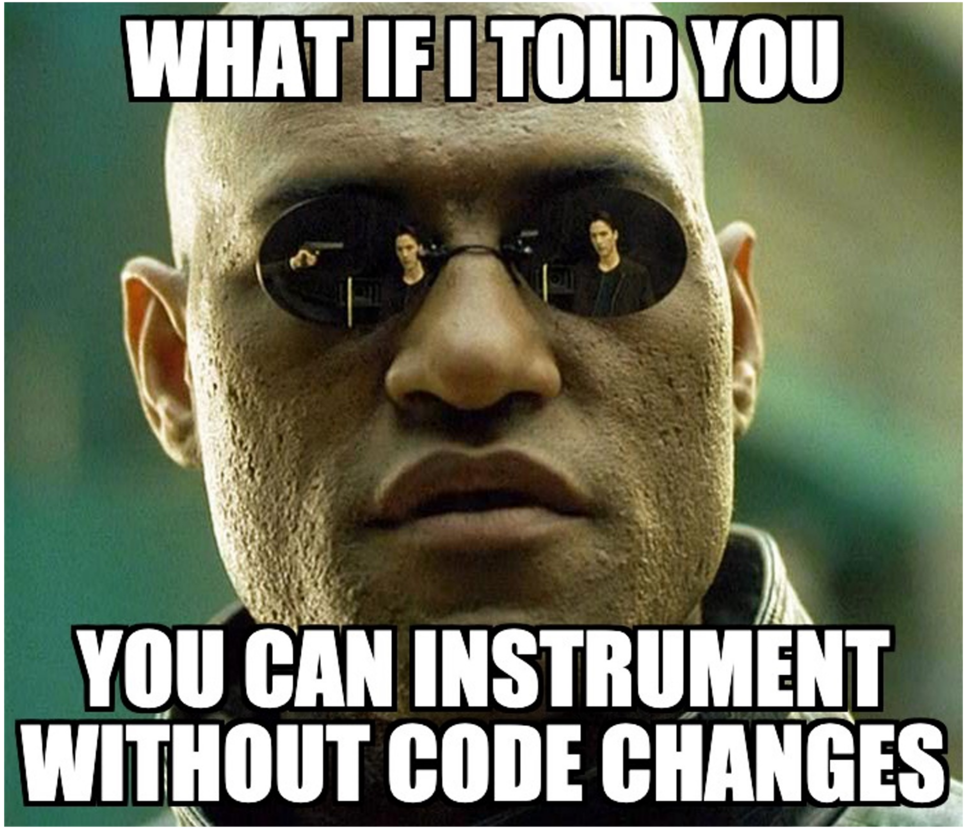
Getting Started: Traces (Manual)

```
// 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+

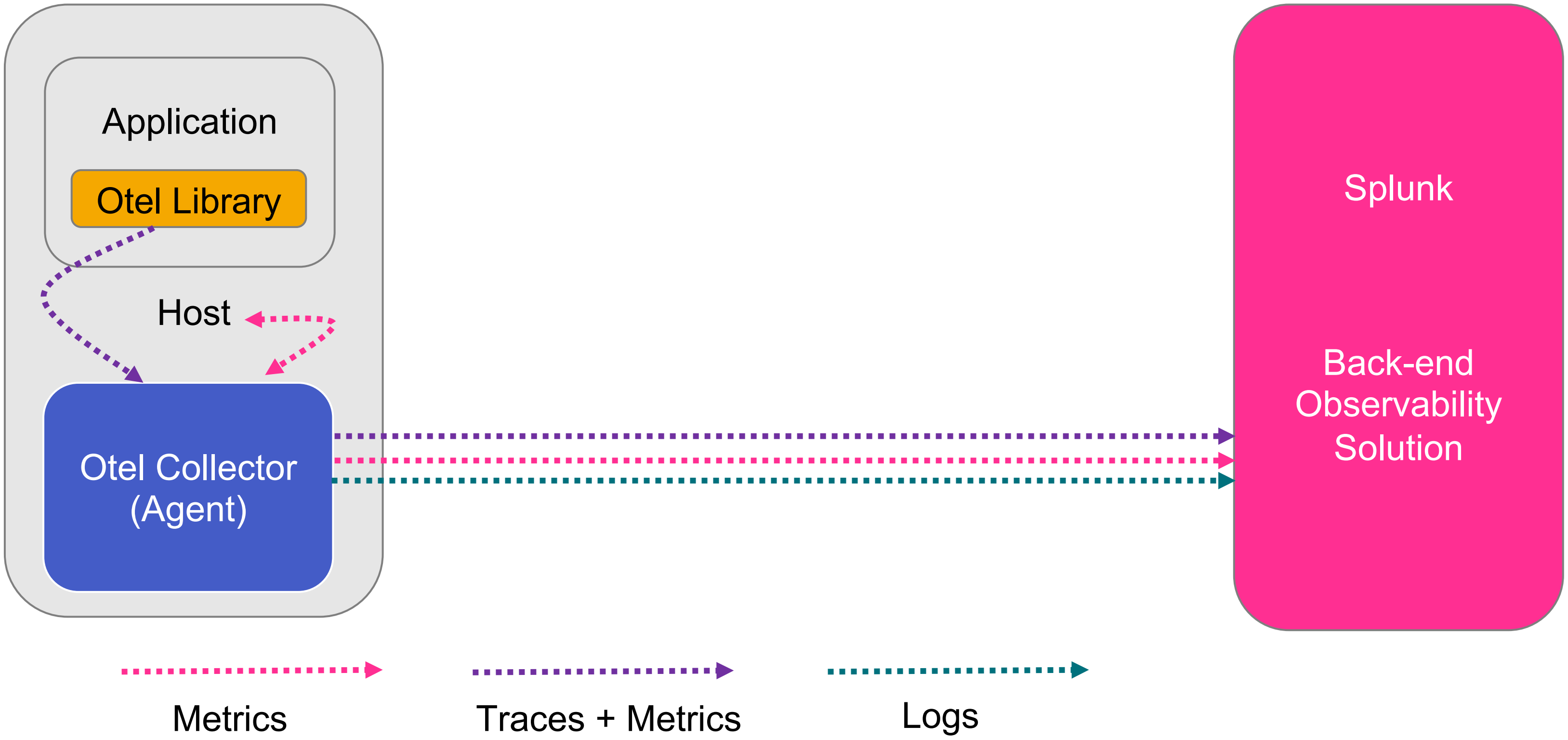
Spymemcached 2.12+

Twilio 6.6+

OpenTelemetry Java auto-instrumentation library support

DEMO

Demo Architecture: OpenTelemetry



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!

Additional resources:

- https://opentelemetry.io
- https://github.com/open-telemetry/community
- https://www.cncf.io/webinars/how-opentelemetry-is-eating-the-world/