logz.io

ADDO
ALL DAY DEVOPS

Sampling in Open Source
Distributed Tracing

ALL DAY DEVOPS

Jonah Kowall - @jkowall
Logz.io - CTO

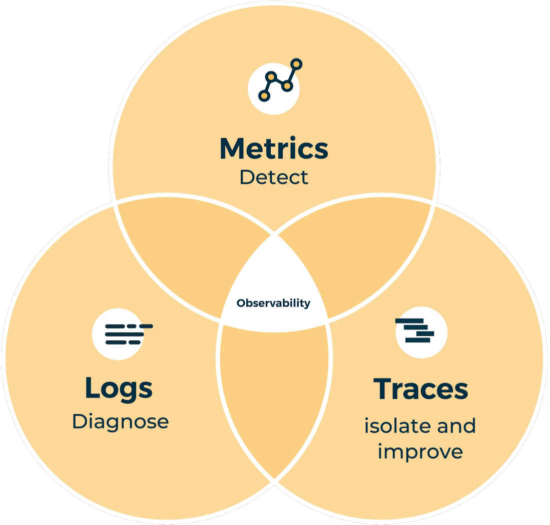
Jonah Kowall
@jkowall

- 15+ years Ops, Network, Security, Performance Engineering for Enterprises and Startups
- Security - CISSP, CISA, PCI
- Head of global monitoring at Thomson Reuters
- Head of IT Ops at MFG.com (Bezos Expeditions)
- Gartner Research VP 4 years
- VP Product Strategy AppDynamics/Cisco 4 years
- Kentik CTO 1 year
- Logz.io CTO 7 months

What is Observability?

Automated and manual instrumentation

*"**Observability** is a measure of how well internal states of a system can be inferred from knowledge of its external outputs."*



Three Options for Observability

1

Build and Run your Own

- Difficult to manage and run at scale
- Dedicated team to build monitoring versus solving business problems
- Not as reliable as expected

2

Purchase a proprietary tool

- Lock in
- Not as interoperable
- Less preferred by developers

3

Logz.io best of breed unified observability platform

Open Source Observability is Popular

ELK	700,000+ (2019)
Grafana	500,000+ (2020)
Prometheus	240,000+ (2020)
JAEGER	
splunk>	19,500 (2020)
DATADOG	11,500 (2020)

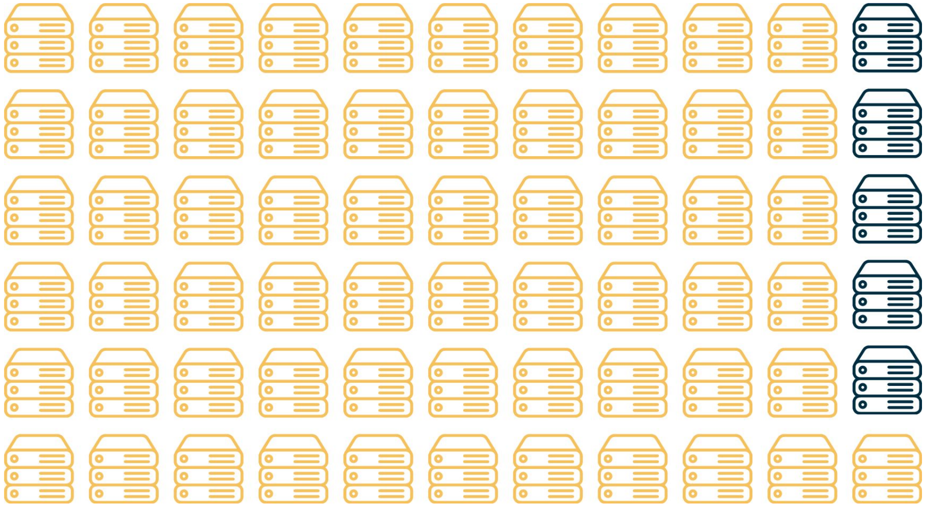
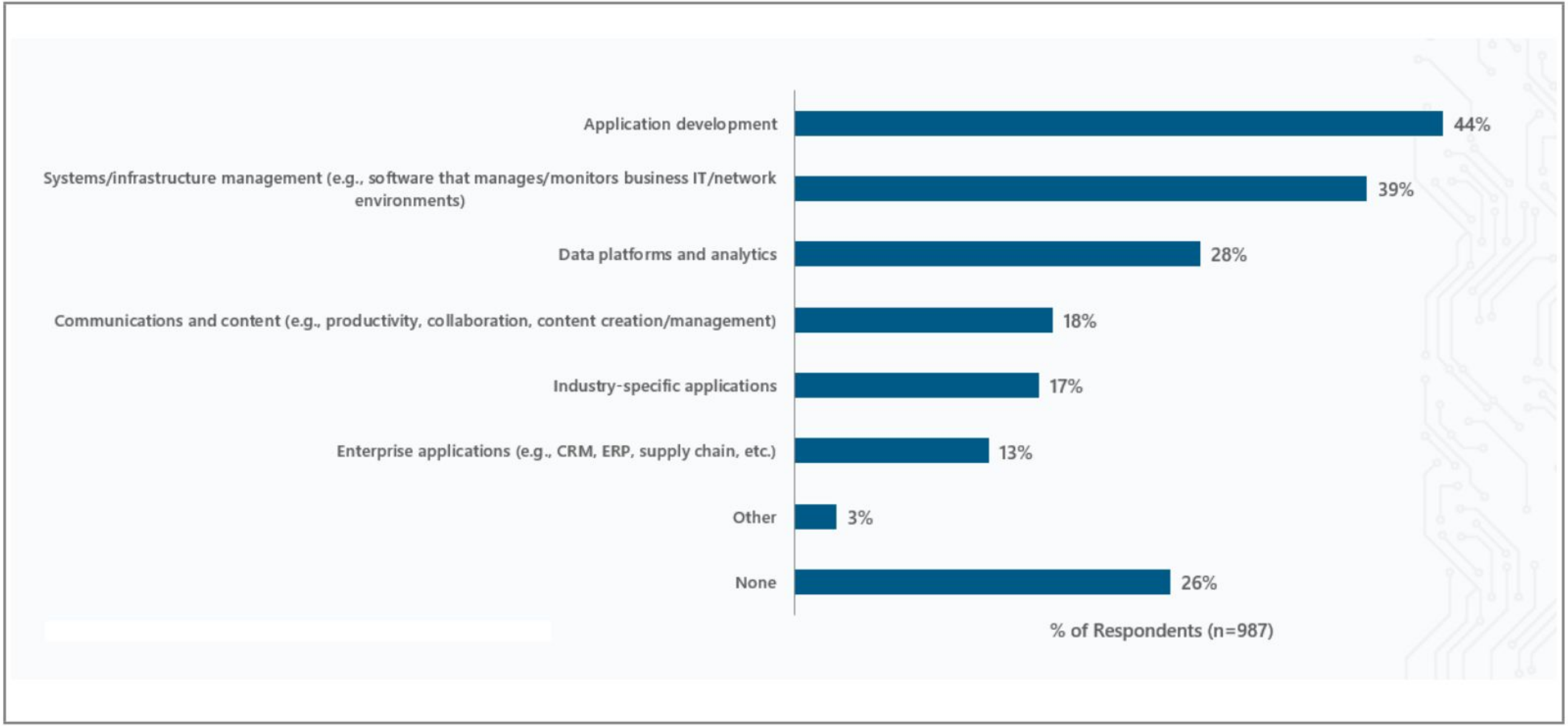


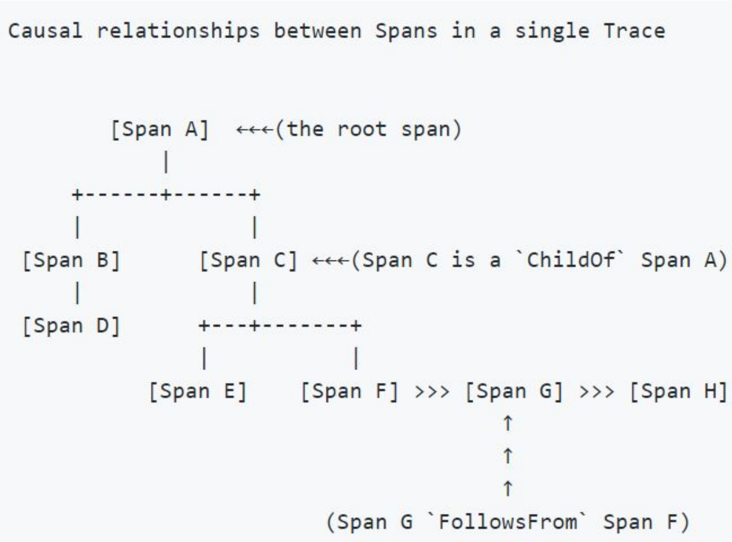
Figure 1: Monitoring is among top use cases for open source software



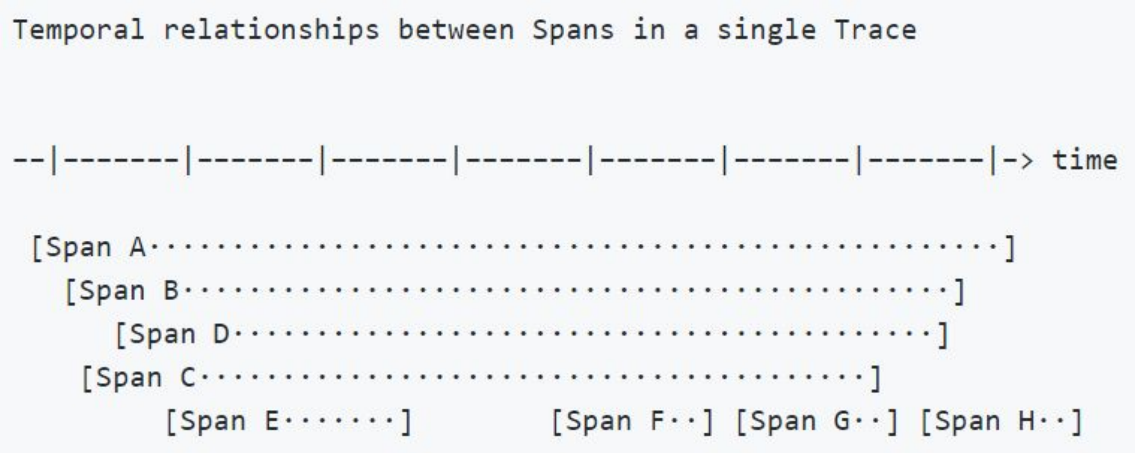
Source: 451 Research, Voice of the Enterprise: Digital Pulse, Vendor Evaluations 2018

Tracing Fundamentals

- **Goal:** monitor, profile, and determine root cause



```
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)
```

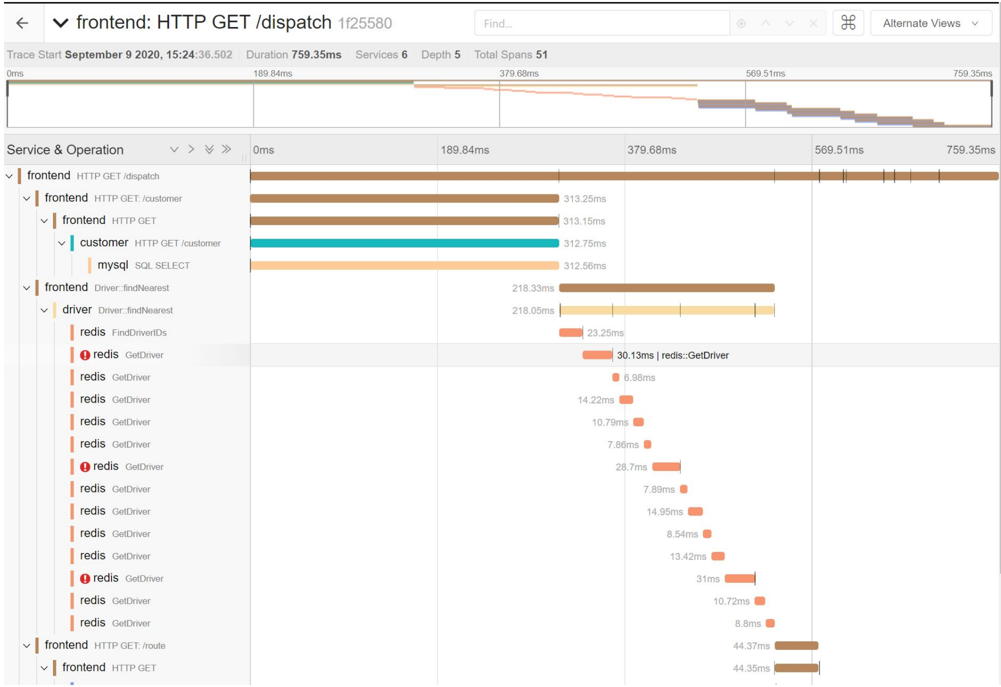


```
Temporal relationships between Spans in a single Trace


--|-------|-------|-------|-------|-------|-------|-------|-> time

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

logz.io

What is tracing

- Follow the end-to-end transaction between sub-transactions (called spans)
    - sub-transaction include dependant transactions, especially in microservices architectures
- Measure errors, latency, and other indicators across each span



frontend: HTTP GET /dispatch 1f25580

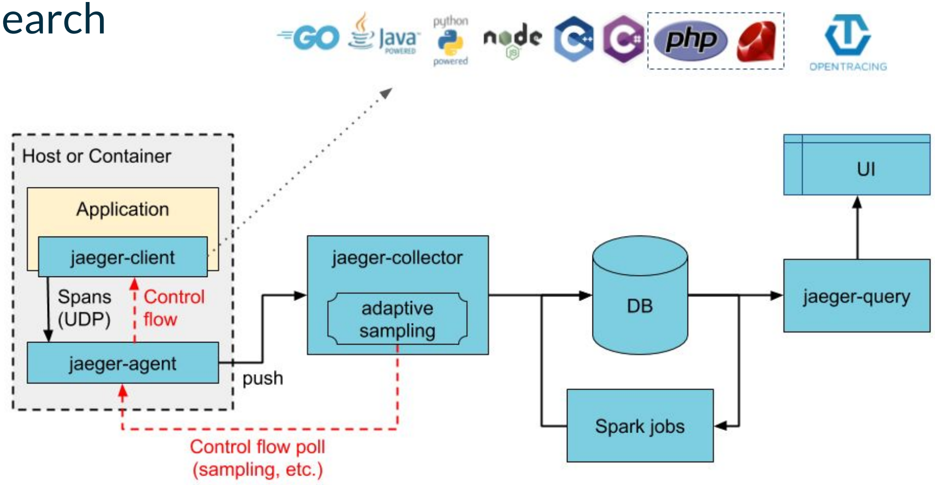
Trace Start **September 9 2020, 15:24**:36.502 | Duration **759.35ms** | Services **6** | Depth **5** | Total Spans **51**

Service & Operation	0ms · 189.84ms · 379.68ms · 569.51ms · 759.35ms
frontend HTTP GET /dispatch	
frontend HTTP GET: /customer	313.25ms
frontend HTTP GET	313.15ms
customer HTTP GET /customer	312.75ms
mysql SQL SELECT	312.56ms
frontend Driver::findNearest	218.33ms
driver Driver::findNearest	218.05ms
redis FindDriverIDs	23.25ms
❗ redis GetDriver	30.13ms | redis::GetDriver
redis GetDriver	6.98ms
redis GetDriver	14.22ms
redis GetDriver	10.79ms
redis GetDriver	7.86ms
❗ redis GetDriver	28.7ms
redis GetDriver	7.89ms
redis GetDriver	14.95ms
redis GetDriver	8.54ms
redis GetDriver	13.42ms
❗ redis GetDriver	31ms
redis GetDriver	10.72ms
redis GetDriver	8.8ms
frontend HTTP GET: /route	44.37ms
frontend HTTP GET	44.35ms

Popular Tracing Open Source Projects

Traces

- Jaeger, Zipkin, Skywalking
- Spark (Dependency Map)
- Kafka
- ElasticSearch



Tracing vs APM

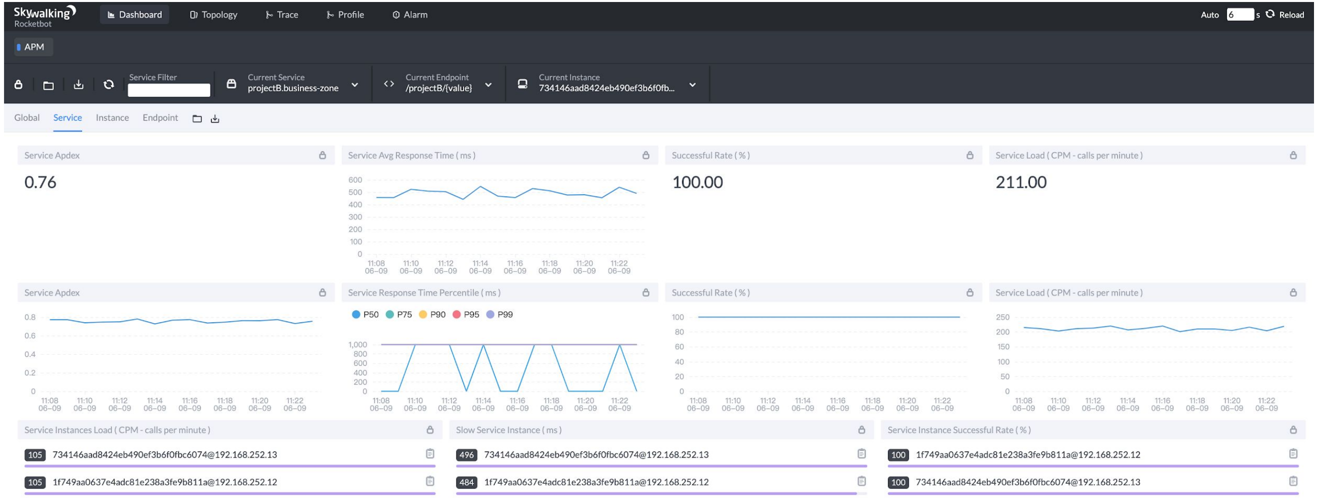
Tracing is a fundamental building block of APM

APM includes the following capabilities:

1. Front-end monitoring
2. **Tracing and diagnostics**
3. Analytics (correlation, anomaly detection, RCA)

Missing from Open Source tracing:

1. Tracking and monitoring service levels (aggregated metrics)
2. Profiling down to the code level if language/technology permits
3. Analytics
4. Front-end monitoring is lacking



Service Filter

Current Service: projectB.business-zone

Current Endpoint: /projectB/{value}

Current Instance: 734146aad8424eb490ef3b6f0fb...

Service Apdex: 0.76

Service Avg Response Time (ms)

Successful Rate (%): 100.00

Service Load (CPM - calls per minute): 211.00

Service Apdex

Service Response Time Percentile (ms)

Successful Rate (%)

Service Load (CPM - calls per minute)

Service Instances Load (CPM - calls per minute)

105 734146aad8424eb490ef3b6f0fbc6074@192.168.252.13

105 1f749aa0637e4adc81e238a3fe9b811a@192.168.252.12

Slow Service Instance (ms)

496 734146aad8424eb490ef3b6f0fbc6074@192.168.252.13

484 1f749aa0637e4adc81e238a3fe9b811a@192.168.252.12

Service Instance Successful Rate (%)

100 1f749aa0637e4adc81e238a3fe9b811a@192.168.252.12

100 734146aad8424eb490ef3b6f0fbc6074@192.168.252.13

How we want to improve tracing solutions

Contributing back to Jaeger project specific features and capabilities

These also go into logz.io service since we have no conflict with on-premises users

1. Implement streaming analytics (Kafka Streams) versus Spark for dependencies
2. Exporting prometheus metrics off streaming for SLA measurement
3. New UI capabilities to provide RCA
4. Linkages directly to Kibana from Jaeger (already in logz.io)

Participation in OpenTelemetry (more on this later…)

Tracing Standards

APIs, Auto-Instrumentation Agents, Instrumented Libraries, Wire protocols

Attempts

ARM - 1996-2007

OpenTracing - 2016-2019

OpenCensus - 2018* - 2019

OpenTelemetry - 2019 - Present

W3C Distributed Tracing Group - 2018 - Present

OpenTelemetry the big bet

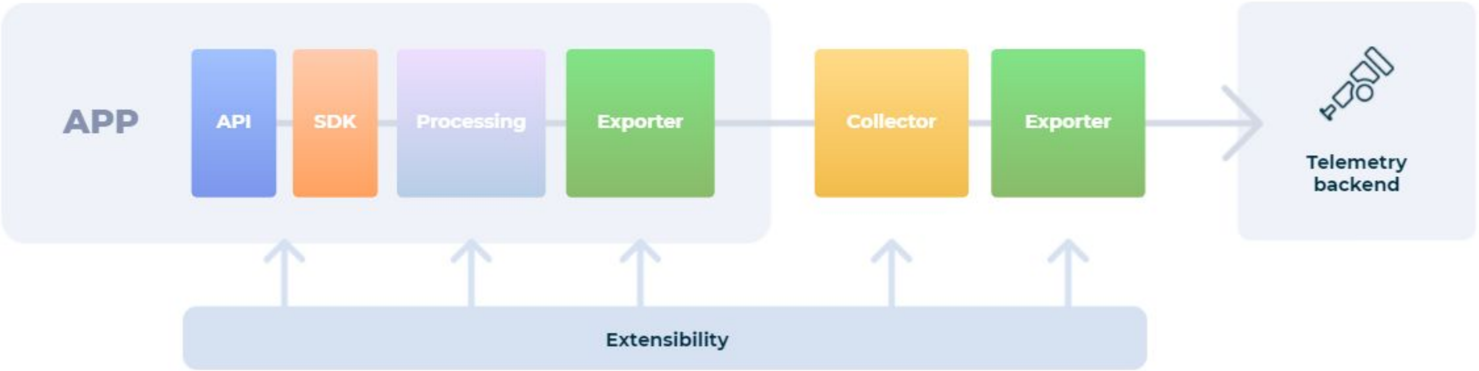
- Open approach to **logging, metrics, and tracing**
- Single **API and SDK** per language with **agents and libraries**
- OpenTelemetry Collector which takes data and exports to 1 or more “backends”

Currently in Beta, GA later this year….. but no Logging

Vendor driven with limited end users

Created by unifying OpenTracing and OpenCensus

What does OpenTelemetry Include

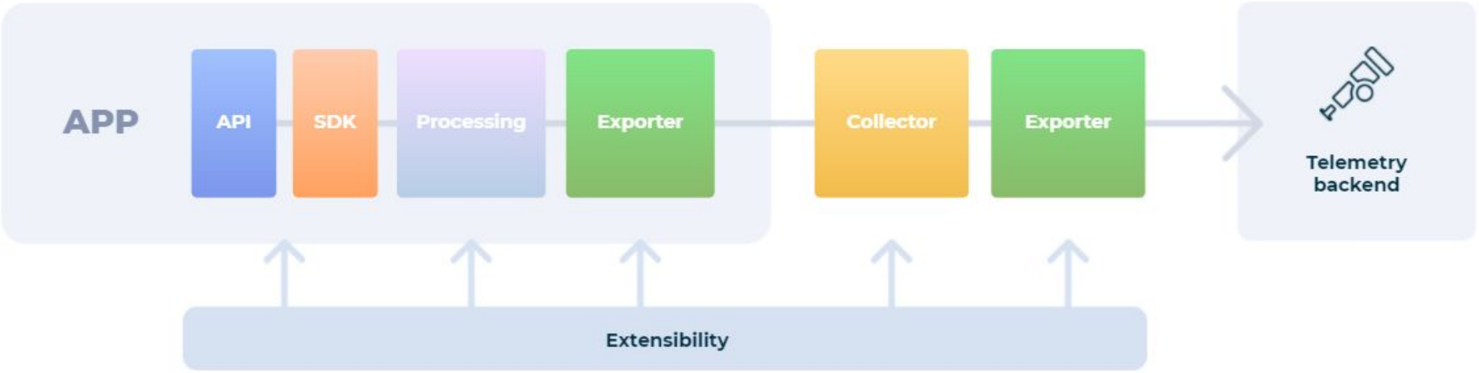


Can auto instrument or manually instrument can be in process or on instance.

Collector can handle sampling and other decisions on data. Often will handle multiple apps or machines. Handles multiple formats of telemetry.

Exporters are specific to the Telemetry Backend.

Architectural Tradeoffs



PROs	CONs
Lots of flexibility in deployment models	Lots of components to manage
Very pluggable to extend analytics throughout the lifecycle of the trace	Commercial tracing tools moved away from collectors to reduce time to value

How the industry is reacting

- Easier to swap out instrumentation and tools for observability
- Agents are not differentiated between solutions
- Move towards consumption + retention based pricing
- Solution’s value comes from...
  - Data ingestion
  - Scale
  - Machine learning/AI
  - Ecosystem of integrations
  - Flexibility

Vendor Reaction to OpenTelemetry

Embracing or Leading

Splunk
Amazon
Microsoft
Datadog
Google
Lightstep
New Relic
Dynatrace
Honeycomb
Logz.io
Sumo Logic

Watching or Supporting *Reluctantly*

Aternity/Riverbed
Instana

Ignoring

Broadcom
AppDynamics/Cisco
Solarwinds
ManageEngine

Any other vendor not listed

Sampling

What do we keep, and when?

How?

- **Simple (head-based) :** percentage
- **Complex (tail-based) :** conditional (error/ slow), user type (paying customers)

Where?

- **Client :** Logic embedded inside application code, but can cause additional overhead and issues. Cannot see full trace, only head-based.
- **Collector :** Can apply post transaction sampling decisions, but often has latency and storage implications. May not have all spans in trace.
- **Observability Platform :** Decide when ingesting, but costs a lot of bandwidth. Scale concerns of backend. Can send unsampled data and make decision after analyzing all data.

Sampling in OpenTelemetry

Working on a pluggable framework, but not happening in 2020

Design: Decided before Span creation context is passed to a `Sampler` which returns a sampling decision.

Built in Samplers:

- AlwaysOn
- AlwaysOff
- TraceIdRatioBased
- ParentBased

Overhead

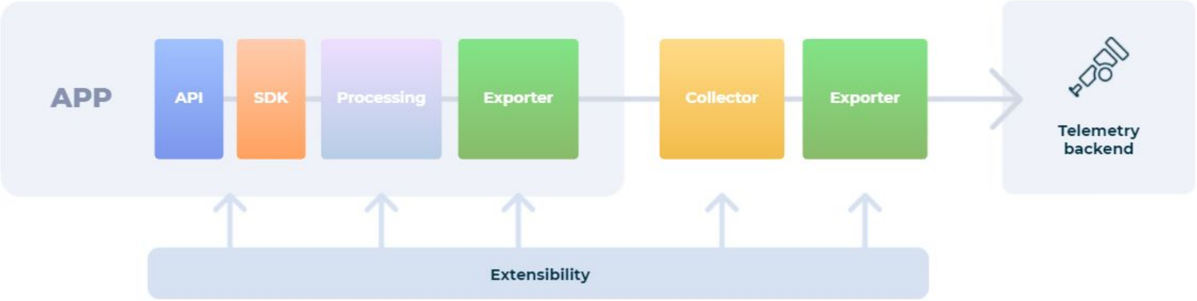
Data collection creates resource constraints, but transactional impact can be direct or indirect (ex: logging, metrics, and especially tracing)

In Band:

Sampling or other intelligence in application creates additional latency.

Out of Band:

Done outside the application minimizes transactional impact. Still uses resources, but not directly impacted



Putting it together

- Tracing is complex with many options and technologies to choose from
  - Projects available which are open-standard based
  - Sampling must be considered for any kind of volume
- Experiment with open source or low cost tracing solutions to gain benefits
  - Free or Freemium options
  - Evaluate your observability strategy to align to standards

- Tracing can be used to solve many operational and business problems think creatively

- **Contribute and participate in the community, everyone is very open in the open source and open standards groups**

logz.io

THANK YOU.