



Tracing Kafka with OpenTelemetry





new relic®

Tracing Kafka with OpenTelemetry





Tracing Kafka with OpenTelemetry



Safe Harbor

This presentation and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. ("New Relic") to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic's express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as "believes," "anticipates," "expects" or words of similar import.

Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic's current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings New Relic makes with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic's Investor Relations website at ir.newrelic.com or the SEC's website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this presentation or otherwise, with respect to the information provided.

whoami



Harry Kimpel

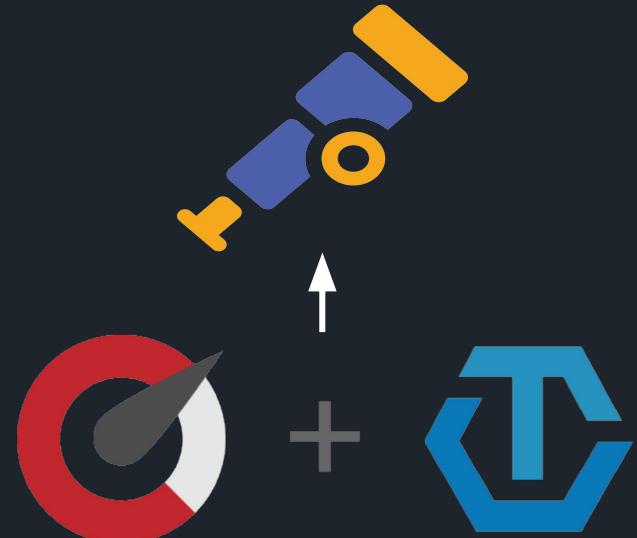
Principal Developer Relations Engineer

New Relic

@harrykimpel

The Rise of OpenTelemetry

- **OpenTelemetry** is a CNCF project
- Formed through a merger of the **OpenTracing** and **OpenCensus** projects in 2019
- Vendor-agnostic - set of APIs, libraries, integrations, and a collector service for telemetry
- **Standardizes** how you collect telemetry data from your applications and services
- Send it to an observability platform of your choice



The Rise of OpenTelemetry



Vendor Neutral

Provides flexibility to change backend



Interoperable

End-to-end visibility with standard instrumentation



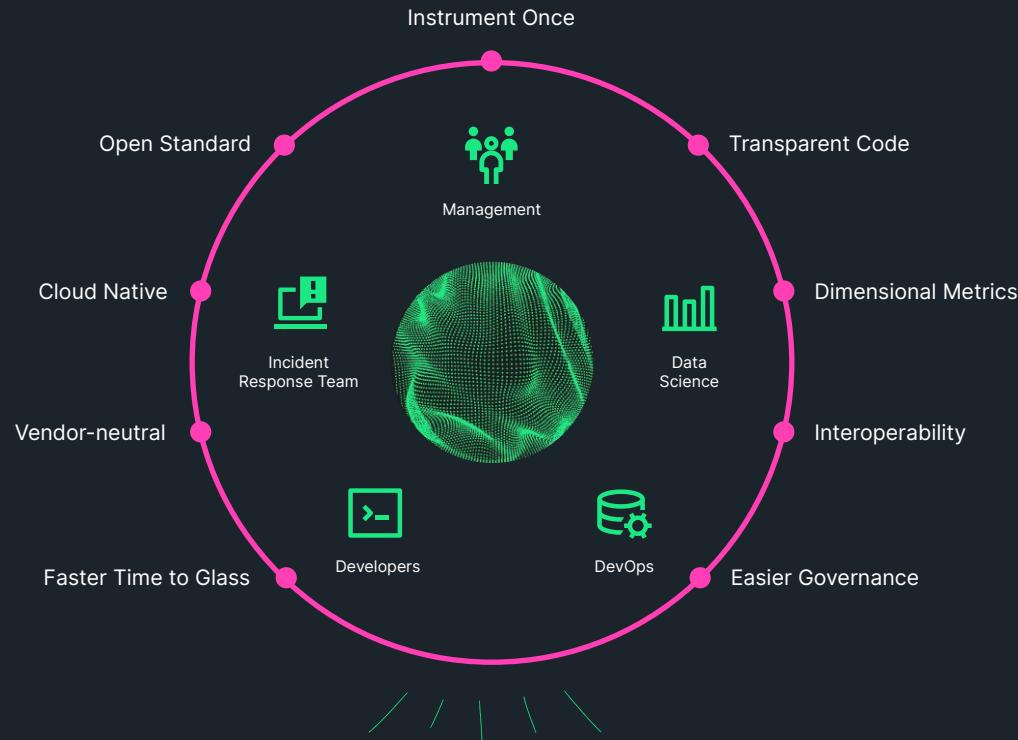
Configurable

Pick and choose from the pieces what is needed

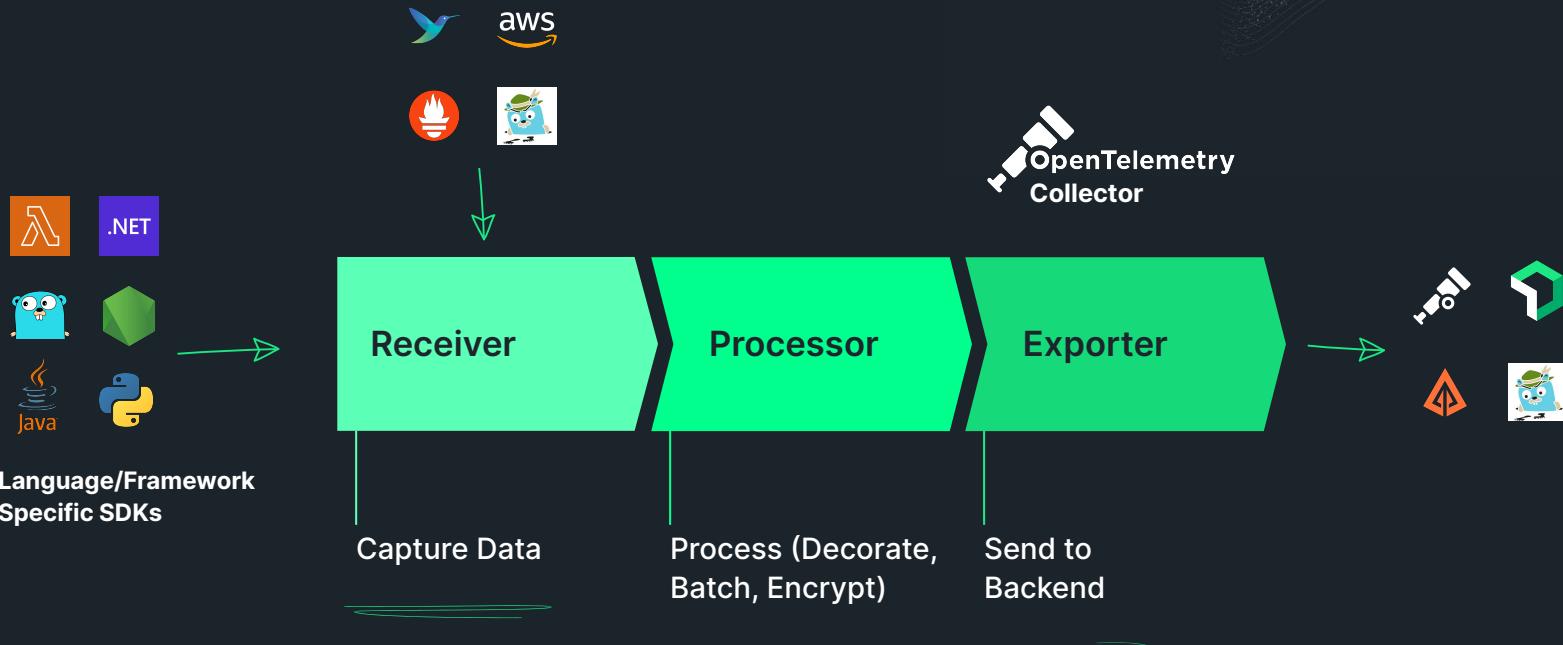
By 2025, **70%** of new cloud-native application monitoring will use open-source instrumentation

⁽¹⁾ Source: Gartner Magic Quadrant 2021 for Application Performance Monitoring - [link](#)

Why OpenTelemetry?



Snapshot of OpenTelemetry



The Rise of OpenTelemetry



Let's all just
go to **OpenTelemetry!**

BUT WAIT ...



Design Considerations & Guidelines

Customize Everything
(aka OTEL)

Ingredients

Custom Made

Flavours

Java, .NET and others mature



End Product
(aka NR Agents)

Ready to Eat

Packaged

Scalable

Most Languages

Design Considerations & Guidelines



OpenTelemetry

**Community
Supported**

**Doesn't have
SLA/Dedicated
Engineering team**



new relic®

Built by NR

**SLAs/Support
provided**

Status & Releases



OpenTelemetry

Screenshot from
<https://opentelemetry.io/docs/instrumentation/>
taken
February 26th 2024

Status and Releases

The current status of the major functional components for OpenTelemetry is as follows:

Language	Traces	Metrics	Logs
C++	Stable	Stable	Stable
C#/.NET	Stable	Stable	Stable
Erlang/Elixir	Stable	Experimental	Experimental
Go	Stable	Stable	In development
Java	Stable	Stable	Stable
JavaScript	Stable	Stable	Development
PHP	Stable	Stable	Stable
Python	Stable	Stable	Experimental
Ruby	Stable	In development	In development
Rust	Beta	Alpha	Alpha
Swift	Stable	Experimental	In development

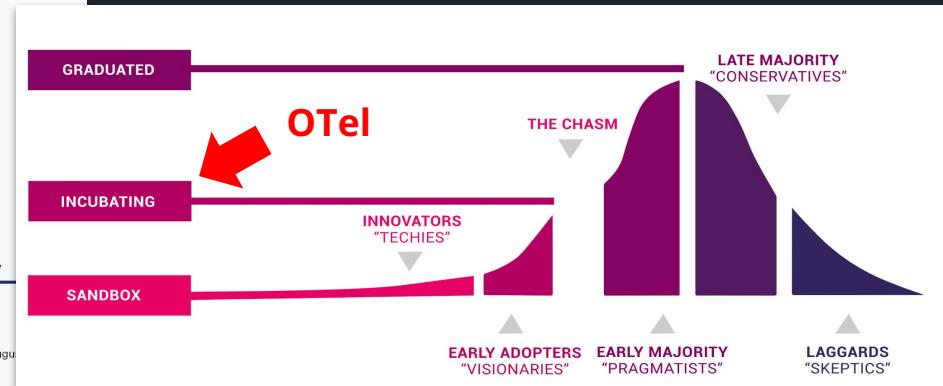
Design Considerations & Guidelines

Hype Cycle for Emerging Tech, 2022



Top 6 Questions To Consider

- Do you understand the difference between Monitoring vs Observability?
- Do you understand the basics of Metrics, Logs & Traces?
- Is tracing widely deployed to provide value?
- Do you have experience in manual instrumentation?
- Is your app friendly to manual instrumentation?
- Is your environment or practice tolerant to change?



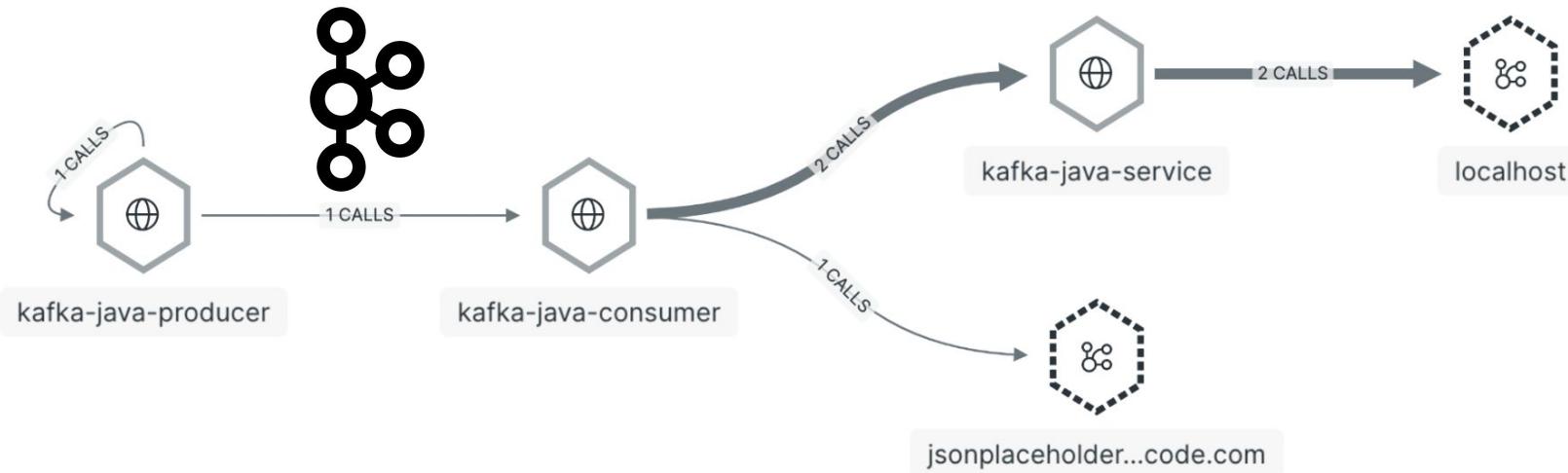
Automatic instrumentation with Java

```
otel-auto-instr > demoConsumer > $ run.sh
1  export JAVA_TOOL_OPTIONS="-javaagent:/Users/hkimpel/projects/kafka/java-local-test/otel-auto-instr/opentelemetry-javaagent.jar"
2  export OTEL_TRACES_EXPORTER=otlp
3  export OTEL_METRICS_EXPORTER=otlp
4  export OTEL_LOGS_EXPORTER=otlp
5  #·US·region
6  export OTEL_EXPORTER_OTLP_ENDPOINT='https://otlp.nr-data.net'
7  #·EU·region
8  #export OTEL_EXPORTER_OTLP_ENDPOINT='https://otlp.eu01.nr-data.net'
9  export OTEL_EXPORTER_OTLP_HEADERS="api-key=NEW_RELIC_LICENSE_KEY"
10 export OTEL_SERVICE_NAME="kafka-java-consumer"
11 export OTEL_SERVICE_VERSION="0.1.0"
12
13 ./mvnw spring-boot:run
```

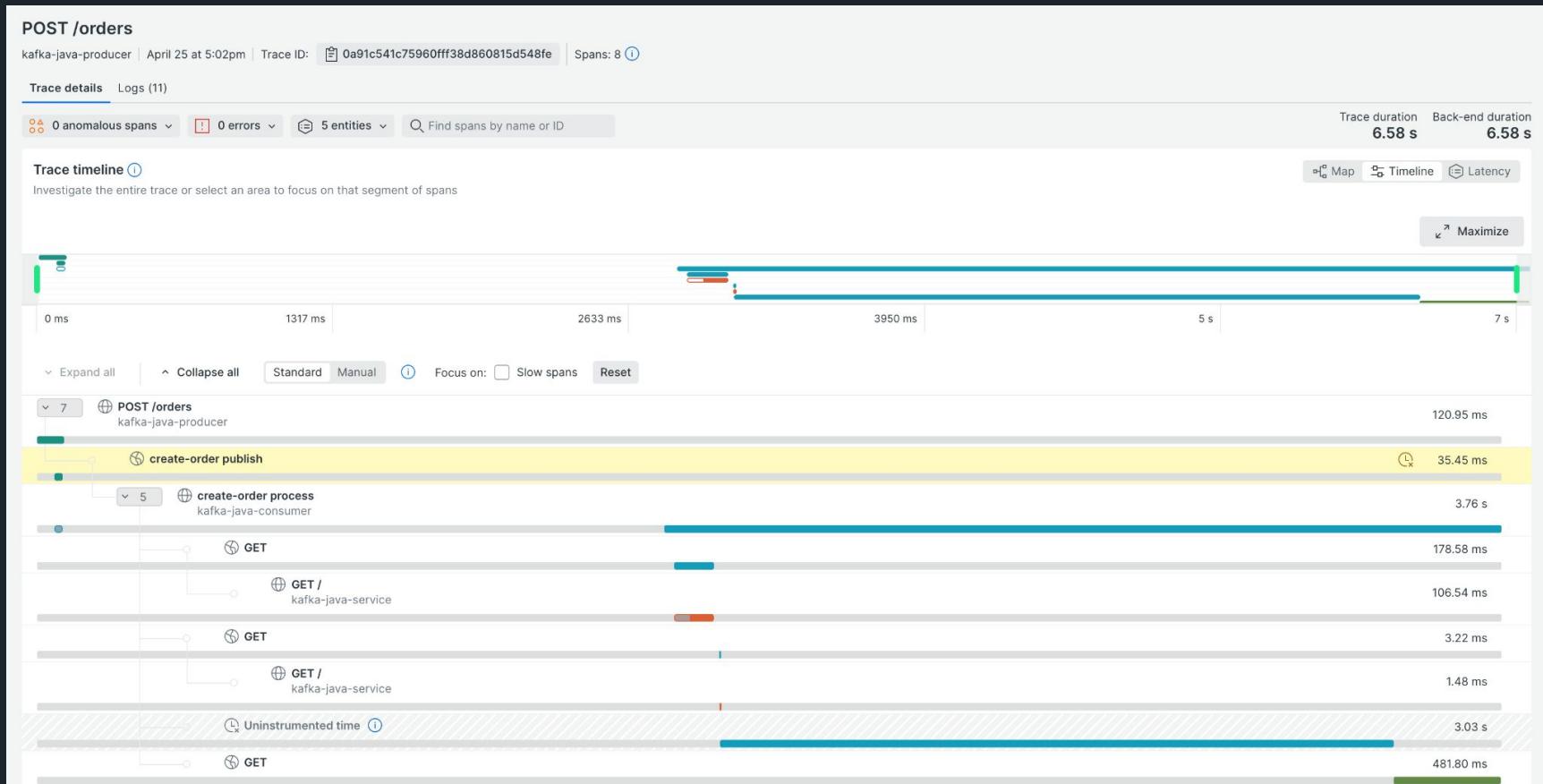
Automatic instrumentation with Java

Entity map i

There are 5 sampled entities in this trace



Automatic instrumentation with Java



Manual instrumentation with Java

≡ application.properties ×

otel-manual-instr > demoProducer > src > main > resources > ≡ application.properties

```
1  spring.kafka.order.bootstrap-servers:·localhost:9092
2  spring.kafka.order.topic.create-order:·create-order
3  #·US·region
4  otel.exporter.otlp.endpoint:·https://otlp.nr-data.net
5  #·EU·region
6  #otel.exporter.otlp.endpoint:·https://otlp.eu01.nr-data.net
7  otel.exporter.otlp.headers.api-key:·NEW_RELIC_LICENSE_KEY
8  otel.jmx.target.system:·tomcat,kafka-broker
9  otel.instrumentation.common.default-enabled:·true
10 otel.instrumentation.kafka.enabled:·true
11 otel.instrumentation.tomcat.enabled:·true
12 otel.javaagent.debug:·true
13 |
```

Manual instrumentation with Java

```
Pieces: Comment | Pieces: Explain
40  ↪ @Bean
41  ↪ public OpenTelemetry openTelemetry() {
42  ↪     Resource resource = Resource.getDefault().toBuilder()
43  ↪         .put(ResourceAttributes.SERVICE_NAME, "value:kafka-java-producer")
44  ↪         .put(ResourceAttributes.SERVICE_VERSION, "value:0.1.0")
45  ↪         .put(key: "otel.jmx.target.system", value:"tomcat,kafka-broker")
46  ↪         .put(key: "otel.instrumentation.kafka.metric-reporter.enabled", value:true).build();
47
48  ↪     SdkTracerProvider sdkTracerProvider = SdkTracerProvider.builder()
49  ↪         .addSpanProcessor(BatchSpanProcessor.builder(OtlpGrpcSpanExporter.builder()
50  ↪             .setEndpoint(
51  ↪                 otlpEndpoint)
52  ↪             .addHeader(key: "api-key",
53  ↪                 otlpHeadersApiKey)
54  ↪             .build()).build())
55  ↪             .setResource(resource)
56  ↪         .build();
57
58  ↪     SdkMeterProvider sdkMeterProvider = SdkMeterProvider.builder()
59  ↪         .registerMetricReader(PeriodicMetricReader.builder(OtlpGrpcMetricExporter.builder()
60  ↪             .setEndpoint(
61  ↪                 otlpEndpoint)
62  ↪             .addHeader(key: "api-key",
63  ↪                 otlpHeadersApiKey)
64  ↪             .build()).build())
65  ↪             .setResource(resource)
66  ↪         .build();
67
68  ↪     SdkLoggerProvider sdkLoggerProvider = SdkLoggerProvider.builder()
69  ↪         .addLogRecordProcessor(
70  ↪             BatchLogRecordProcessor.builder(OtlpGrpcLogRecordExporter.builder()
71  ↪                 .setEndpoint(
72  ↪                     otlpEndpoint)
73  ↪                     .addHeader(key: "api-key",
74  ↪                         otlpHeadersApiKey)
75  ↪                     .build()).build())
76  ↪             .setResource(resource)
77  ↪         .build();
78
79  ↪     OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
80  ↪         .setTracerProvider(sdkTracerProvider)
81  ↪         .setMeterProvider(sdkMeterProvider)
82  ↪         .setLoggerProvider(sdkLoggerProvider)
83  ↪         .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
84  ↪         .buildAndRegisterGlobal();
85
86  ↪     return openTelemetry;
87
88 }
```

Manual instrumentation with Java - Kafka producer

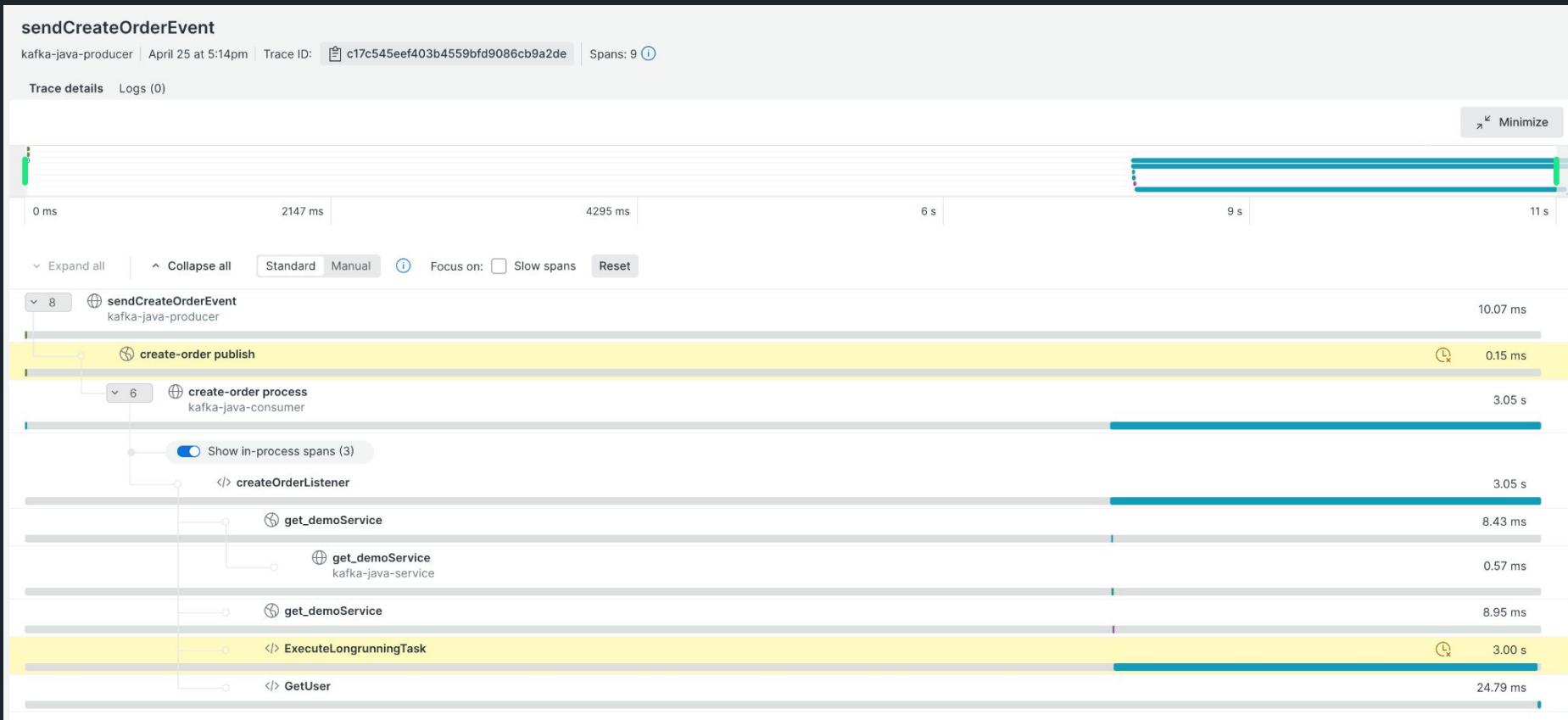
```
Pieces: Comment | Pieces: Explain
22  ...
23  @Bean
24  public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
25      Map<String, Object> config = new HashMap<>();
26      config.put(
27          org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
28          TracingProducerInterceptor.class.getName());
29      ...
30      config.put(org.apache.kafka.clients.producer.ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
31      config.put(org.apache.kafka.clients.producer.ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, value: JsonSerializer.class);
32      config.put(org.apache.kafka.clients.producer.ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
33          value: JsonSerializer.class);
34      return new DefaultKafkaProducerFactory(config);
35  }
```

Manual instrumentation with Java

- Kafka consumer

```
Pieces: Comment | Pieces: Explain
35     ...@Bean("orderConsumerFactoryNotificationService")
36     public ConsumerFactory<String, Order> createOrderConsumerFactory() {
37         Map<String, Object> props = new HashMap<>();
38         props.put(
39             ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
40             TracingConsumerInterceptor.class.getName());
41         props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
42         props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
43         props.put(ConsumerConfig.CLIENT_ID_CONFIG, UUID.randomUUID().toString());
44         props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, value: StringSerializer.class);
45         props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, value: JsonSerializer.class);
46         props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, value: false);
47
48         return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
49             new JsonDeserializer<>(targetType: Order.class));
50     }
```

Manual instrumentation with Java



Java supported libraries

... see [GitHub repo](#) for details

- [Disabled instrumentations](#)

Libraries / Frameworks

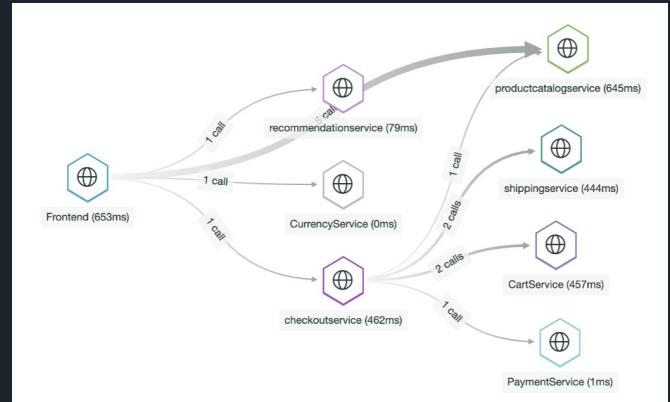
These are the supported libraries and frameworks:

Library/Framework	Auto-instrumented versions	Standalone Library Instrumentation [1]	Semantic Conventions
Akka Actors	2.3+	N/A	Context propagation
Akka HTTP	10.0+	N/A	HTTP Client Spans , HTTP Client Metrics , HTTP Server Spans , HTTP Server Metrics , Provides <code>http.route</code> [2]
Alibaba Druid	1.0+	opentelemetry-alibaba-druid-1.0	Database Pool Metrics
Apache Axis2	1.6+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
Apache Camel	2.20+ (not including 3.x yet)	N/A	Dependent on components in use
Apache CXF JAX-RS	3.2+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
Apache CXF JAX-WS	3.0+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
Apache DBCP	2.0+	opentelemetry-apache-dbc-2.0	Database Pool Metrics
Apache Dubbo	2.7+	opentelemetry-apache-dubbo-2.7	RPC Client Spans , RPC Server Spans
Apache HttpAsyncClient	4.1+	N/A	HTTP Client Spans , HTTP Client Metrics
Apache HttpClient	2.0+	opentelemetry-apache-httpclient-4.3 , opentelemetry-apache-httpclient-5.2	HTTP Client Spans , HTTP Client Metrics
Apache Kafka Producer/Consumer API	0.11+	opentelemetry-kafka-clients-2.6	Messaging Spans
Apache Kafka Streams API	0.11+	N/A	Messaging Spans
Apache MyFaces	1.2+ (not including 3.x yet)	N/A	Provides <code>http.route</code> [2], Controller Spans [3]

Summary

- Exciting times for open source observability!
- Be **mindful about the maturity status**, and plan ahead on the adoption of OpenTelemetry.
- The collector is a very useful tool that also comes with challenges, especially scalability.
- Instrumentation can include traces, logs, and/or metrics to improve observations.
- New Relic is actively investing in OpenTelemetry, and helping engineers do their best work based on **data, not opinions**.

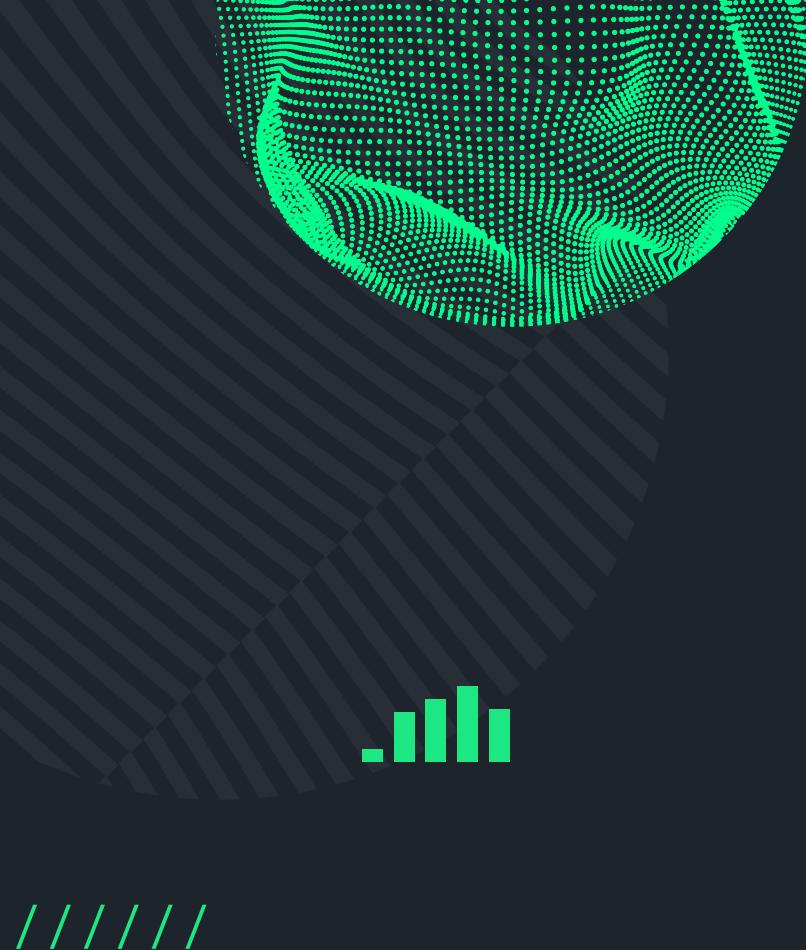
The Stack



Reading Materials for OpenTelemetry

Reading List

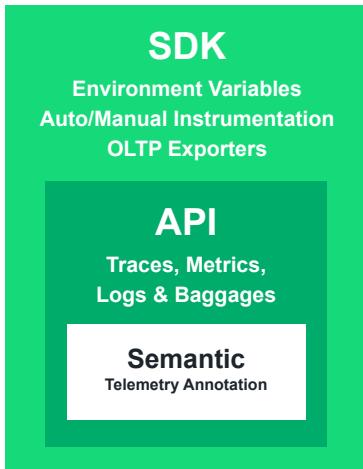
- [OpenTelemetry project mission](#)
- [Observability Primer](#)
- [Java - docs](#)
- [Go - docs](#)
- [Collector - docs](#)
- [OpenTelemetry Github repo](#)
- [Intro to OpenTelemetry x New Relic](#)
- [View your data - Distributed Tracing](#)
- [OpenTelemetry x New Relic Best Practices](#)
- [OpenTelemetry Reference Architecture](#)



Appendix

some more information

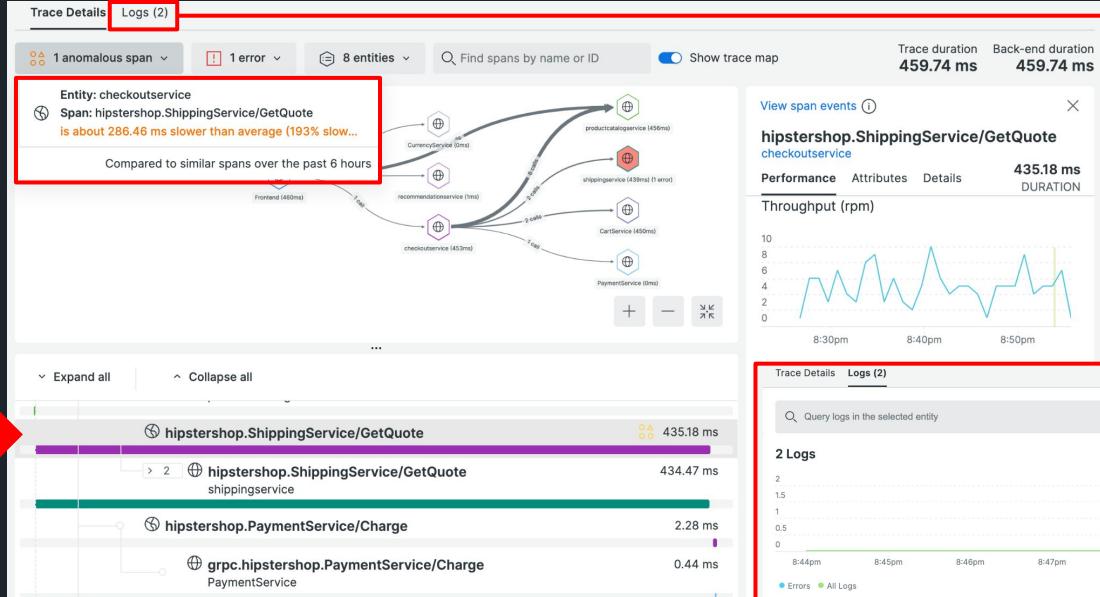
OpenTelemetry - Instrumentation



Core Concepts on Instrumentation

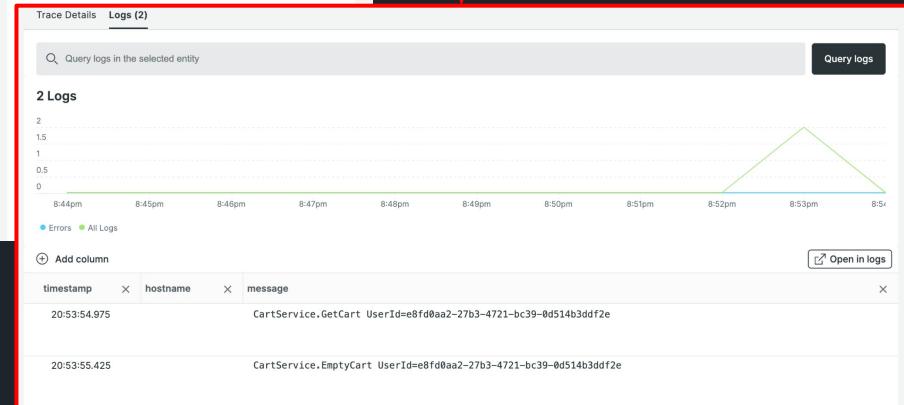
- **Semantic Conventions** - annotate telemetry with attributes specific to the represented operation, such as HTTP calls.
- **API** - standard way to collect instrumentation data.
- **SDK** - language-specific implementation of the API.
 - SDKs incorporate automatic instrumentation for common libraries and frameworks for your application.
- **OpenTelemetry Protocol** (OTLP) - used to send data to your backend Observability platform of choice.
- **Specification** ("spec") - provides blueprints for all of the above to bring standardization across all languages.

Instrumentation in Action



Anomalous Span Detection in
Distributed Tracing for OpenTelemetry

Logs in Context
(correlate to the right trace)



Instrumentation in Action

The screenshot displays the New Relic APM interface, illustrating how instrumentation provides deep insights into application stacks.

Top Left: A trace map for the `hipstershop.ShippingService/GetQuote` service. It shows a tree of spans with a total duration of 434.72 ms. One span, `CreateQuoteFromCount`, has a duration of 434.58 ms. Another, `CreateQuoteFromFloat`, has a duration of 334.18 ms. A red box highlights this section.

Top Right: A detailed view of a span event for `hipstershop.ShippingService/GetQuote`. It shows a timeline from 500 μs to 500 ms. The `shippingService` duration is 434.72 ms. A red box highlights this section.

Middle Left: A trace map for the `hipstershop.ShippingService/ShipOrder` service. It shows a single span with a duration of 0.11 ms. A red box highlights this section.

Middle Right: A detailed view of a span event for `shipOrder`. It shows a timeline from 150 ns to 8:20pm. The `shipOrder` duration is 0.07 ms. A red box highlights this section.

Bottom Center: A callout text: "Pinpoint errors, by improving observation (via span attributes)"

Text Labels:

- Get deeper into the app stack, by building spans.**
- Pinpoint errors, by improving observation (via span attributes)**

OpenTelemetry in Action

```
[main] GET /product/OLJCESPC7Z          44  11(25.00%) |   75   1  1605  17 |
[main] 0.20  0.00
[main] POST /setCurrency               40  6(15.00%)  |   98   2  597  49 |
[main] 0.10  0.00
[main] -----
[main] Aggregated                      565 117(20.71%) |   55   1  1605  18 |
[main] 2.90  0.40
[main]
[server] {"message":"[GetQuote] received request","severity":"info","timestamp": "2022-06-26T09:56:08.259921876Z"}
[server] {"message":"[GetQuote] completed request","severity":"info","timestamp": "2022-06-26T09:56:08.259972475Z"}
[main] Name
eq/a failures/s
[main] -----
[main] GET /
[main] 0.00  0.00
[main] GET /cart
[main] 0.10  0.00
[main] POST /cart
[main] 0.70  0.30
[main] GET /cart/checkout
[main] 0.20  0.00
[main] GET /product/OPUR6V6EVO
[main] 0.20  0.00
[main] GET /product/IYKWWN1N4O
[main] 0.50  0.00
[main] GET /product/2ZYFJ3GM2N
[main] 0.10  0.00
[main] GET /product/66VCHSJNUP
[main] 0.10  0.00
[main] GET /product/6E92ZMYYFZ
[main] 0.20  0.00
[main] GET /product/9SIQTBTOJO
[main] 0.20  0.00
[main] GET /product/L9ECAVTKIM
[main] 0.00  0.00
[main] GET /product/L54PSXNUJM
[main] 0.10  0.00
[main] GET /product/OLJCESPC7Z
[main] 0.20  0.00
[main] POST /setCurrency
[main] 0.20  0.00
[main] -----
[main] Aggregated                      569 119(20.91%) |   55   1  1605  18 |
[main] 2.80  0.30
[main]
```

Deploy the app, and generate load
(successful deployment)



```
sh-5.1# ls
kubectl minikube-linux-amd64 otel-workshop skafold
sh-5.1# cd otel-workshop/
sh-5.1# ls
cloudbuild.yaml  docs      kubernetes-manifests  README.md  skafold.yaml
CODE_OF_CONDUCT.md  hack      LICENSE             release.json  src
CODEOWNERS         images    otel-kubernetes-manifests  renovate.json  values-newrelic.yaml
CONTRIBUTING.md   istio-manifests  pb                 SECURITY.md
sh-5.1# kubectl get nodes
NAME           STATUS   ROLES      AGE   VERSION
minikube       Ready    control-plane   69s  v1.24.1
sh-5.1# kubectl get pods --all-namespaces
NAMESPACE     NAME           READY   STATUS    RESTARTS   AGE
kube-system   coredns-5dd4b75ch6-fwqkh   1/1    Running   0          69s
kube-system   etcd-minikube   1/1    Running   0          62s
kube-system   kube-apiserver-minikube   1/1    Running   0          62s
kube-system   kube-controller-manager-minikube   1/1    Running   0          62s
kube-system   kube-proxy-4k6t6        1/1    Running   0          59s
kube-system   kube-scheduler-minikube   1/1    Running   0          81s
kube-system   storage-provisioner   1/1    Running   0          80s
sh-5.1# [REDACTED]
```

Validate K8s is running, and follow the instructions

Important - deploy might take up to 10 mins

Need the env var? Go [HERE](#).



OpenTelemetry - Prerequisites

This is the first step for the FOK Stack workshop.
Please validate all the required modules are running,
before moving on to the next step.

Deployment behind the scenes:

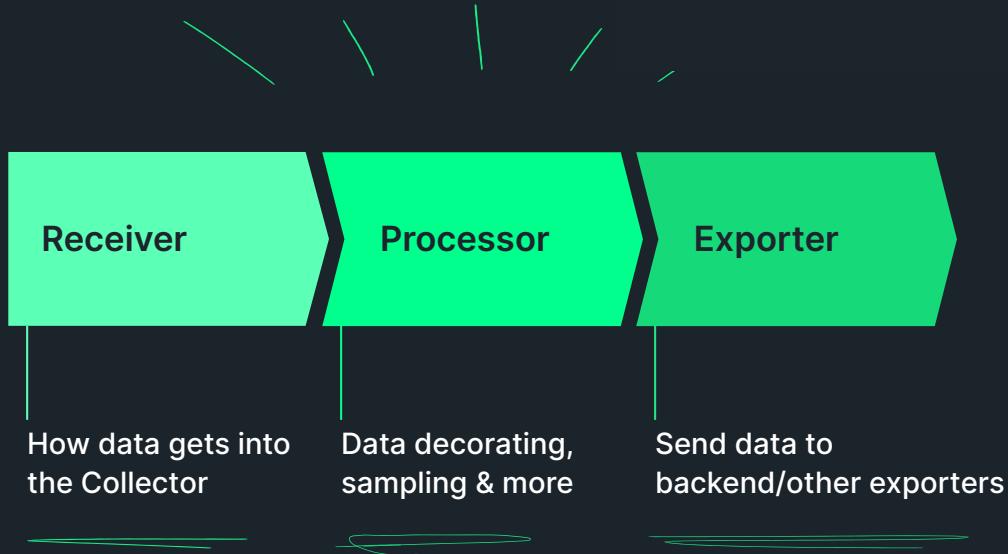
- Installed Docker, Minikube, Skafold, and Git.
- Deployed Minikube.
- Cloned all the files from Github for this workshop.

When the deployment is completed, you can run the following to validate the deployment.

- run `ls` to see all available folders.
- run `sudo docker run hello-world` to test your docker installation.
- run `kubectl get nodes` to see nodes deployed by Minikube.
- run `kubectl get pods --all-namespaces` to see pods deployed by Minikube.

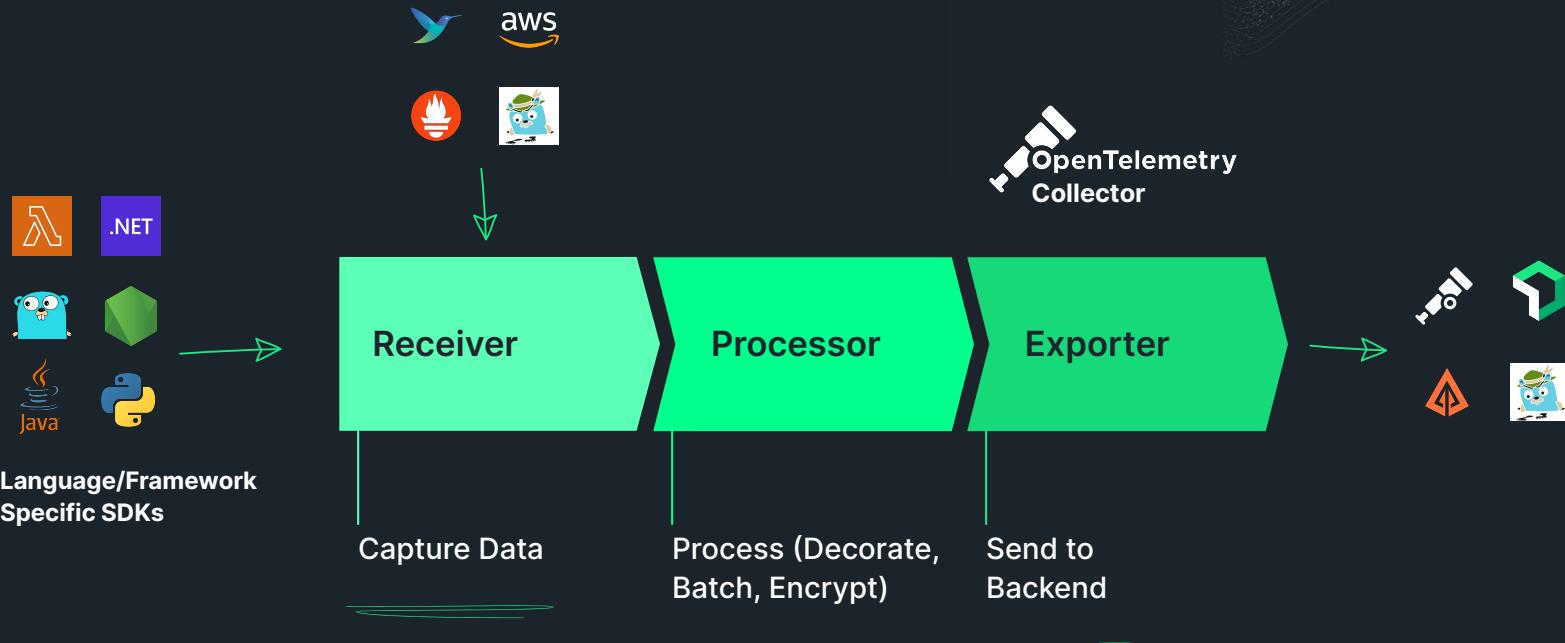
Collector Component

OpenTelemetry

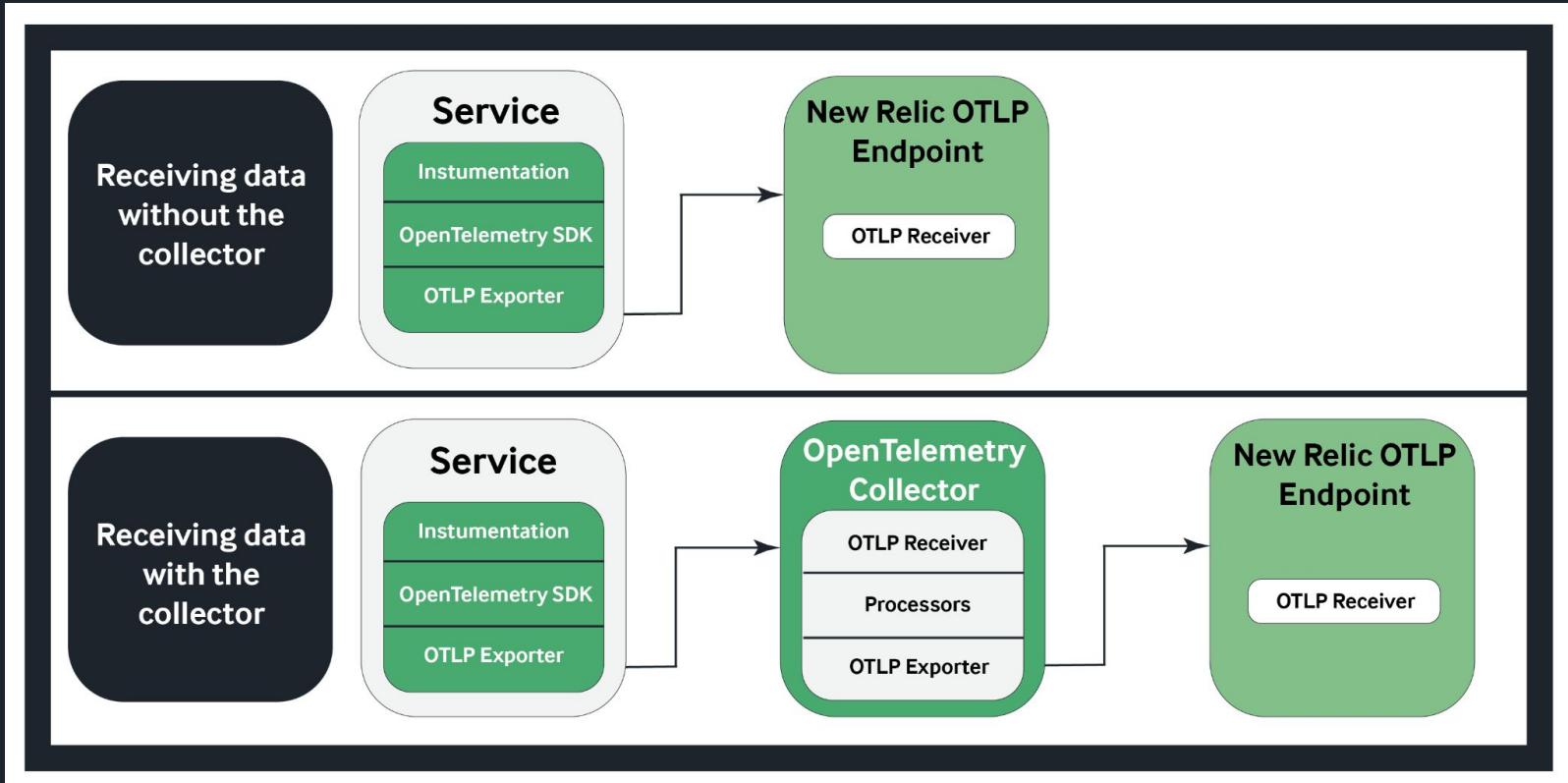


Putting it All Together

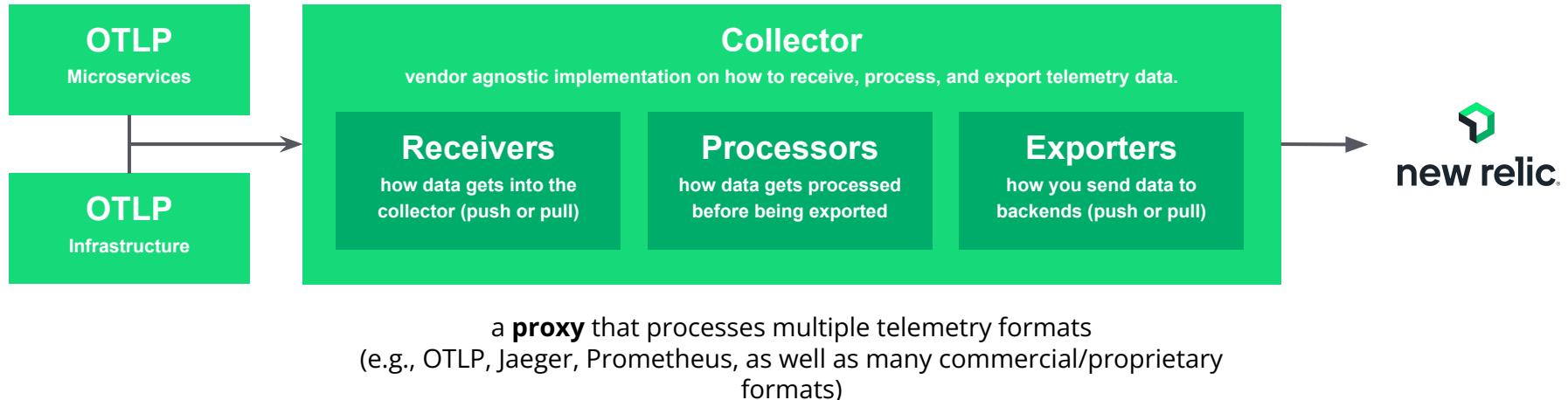
OpenTelemetry



OpenTelemetry - Collector



OpenTelemetry - Collector



OpenTelemetry - Collector

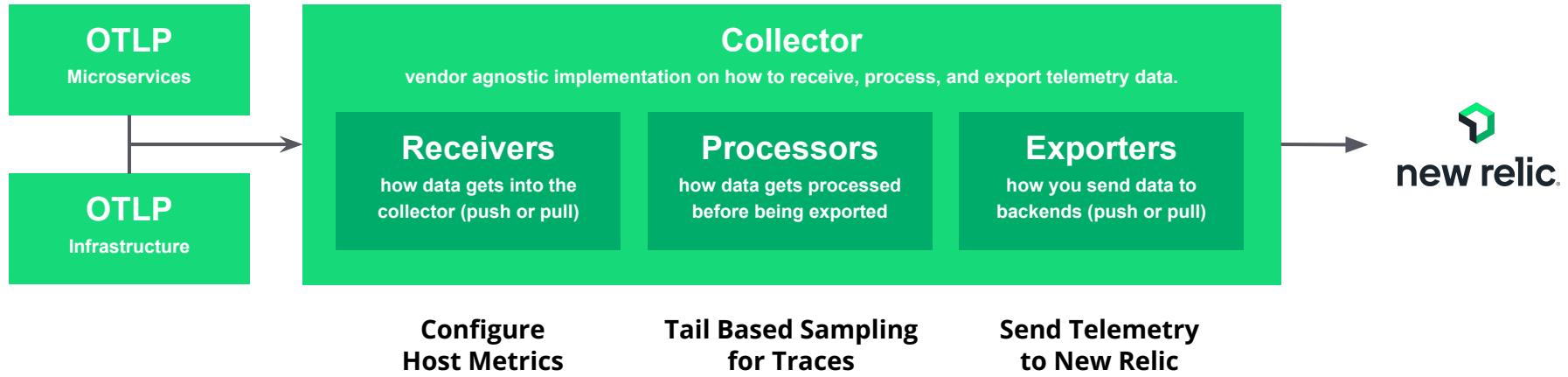
Benefits

- Helps manage ingest costs when using tail sampling
- Reduces app overhead by offloading data management from your apps
- Provides flexibility to handle multiple data formats
- Centralizes configuration of your telemetry pipelines
- Provides ability to export data to multiple backends
- Collects host metrics, e.g., RAM, CPU, and storage capacity
- Provides ability to pull data from monitored systems, e.g., Redis

Downsides - nontrivial!

- Complex to scale
 - Deployment patterns
 - Load balancing
- Monitoring the collector is only available with ... the collector itself (at this time)
- Current stability status is “mixed”

OpenTelemetry - Collector



Collector in Action - Host Metrics

The screenshot illustrates the integration of the OpenTelemetry Collector with the New Relic platform. On the left, a file browser shows the configuration of the OpenTelemetry Collector's `hostmetrics` receiver. A red arrow points from the configuration file to the New Relic interface. A green arrow points from the New Relic interface back to the configuration file, indicating a bidirectional relationship between the two.

File Browser (Left):

```
1 ---  
2 receivers:  
3   hostmetrics:  
4     collection_interval: 20s  
5     scrapers:  
6       cpu:  
7         metrics:  
8           system.cpu.utilization:  
9             enabled: true  
10      load:  
11      memory:  
12        metrics:  
13          system.memory.utilization:  
14            enabled: true  
15      disk:  
16      filesystem:  
17        metrics:  
18          system.filesystem.utilization:  
19            enabled: true  
20      network:  
21      paging:  
22        metrics:  
23          system.paging.utilization:  
24            enabled: true  
25      processes:  
26      otlp:  
27        protocols:  
28          grpc:  
29      processors:  
30        batch:  
31        cumulative_todelta:  
32          metrics:  
33            - system.network.io  
34            - system.disk.operations  
35            - system.network.dropped  
36            - system.network.packets  
37            - process.cpu.time  
38
```

New Relic Data Explorer (Right):

The Data Explorer interface shows a graph of `system.memory.utilization` over time. The graph displays a series of data points starting around 0.12, rising to approximately 0.14, and then fluctuating between 0.13 and 0.15. The X-axis represents time from 10:54am to 11:03am. The Y-axis represents the utilization percentage, ranging from 0 to 0.15.

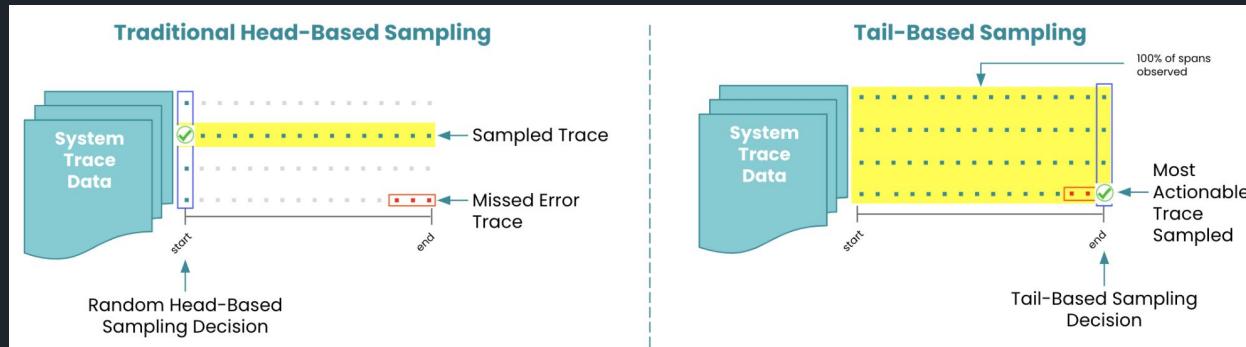
Graph Data:

Time	System Memory Utilization
10:54am	0.12
10:55am	0.14
10:56am	0.13
10:57am	0.14
10:58am	0.13
10:59am	0.14
11:00am	0.13
11:01am	0.14
11:02am	0.13
11:03am	0.14

Collect additional infrastructure metrics
for OpenTelemetry

HEAD vs TAIL Sampling

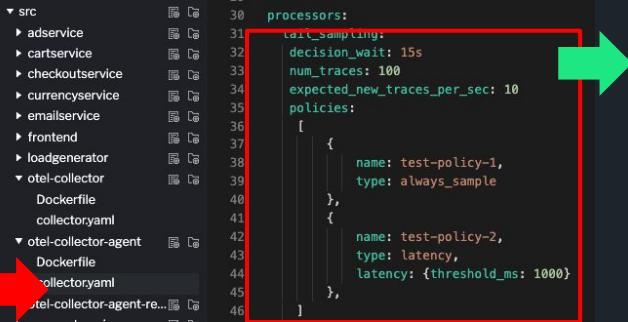
Storing all tracing data is **costly**. Most of the time, you
ONLY NEED THE RIGHT SAMPLING OF DATA



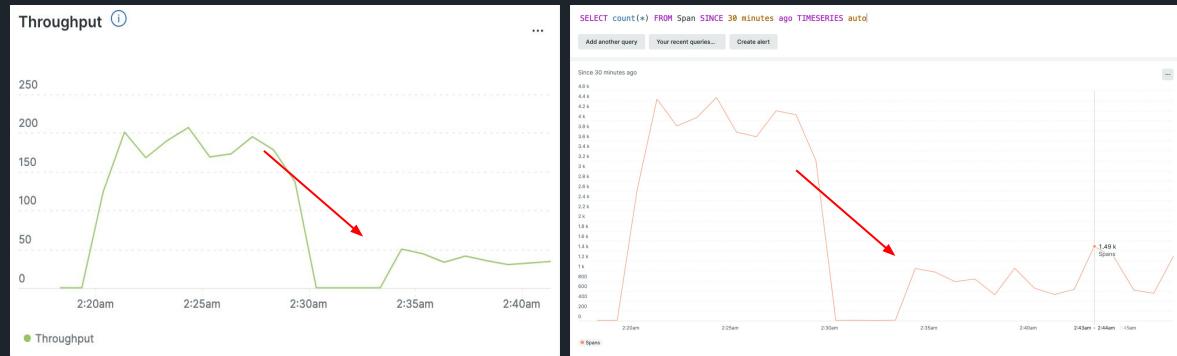
Head-based sampling works well for an overall statistical sampling of requests through a distributed system.

Tail-based sampling is best to decide what to keep, based on isolated, independent portions of the trace data.

Collector in Action - Tail Based Sampling



```
src
  adservice
  cartservice
  checkoutservice
  currencieservice
  emailservice
  frontend
  loadgenerator
  otel-collector
    Dockerfile
    collector.yaml
  otel-collector-agent
    Dockerfile
    collector.yaml
  otel-collector-agent-re...  
  
processors:  
  tail_sampling:  
    decision_wait: 15s  
    num_traces: 100  
    expected_new_traces_per_sec: 10  
    policies:  
      [  
        {  
          name: test-policy-1,  
          type: always_sample  
        },  
        {  
          name: test-policy-2,  
          type: latency,  
          latency: {threshold_ms: 1000}  
        },  
      ]
```



Effects from Tail Based Sampling (Client Side)

New Relic Edge with Infinite Tracing (Server Side)

Fully managed tracing service that observes 100% of your application traces, then provides actionable data so you can solve issues faster.