



Unlocking Observability in Apache Kafka-Based Systems with OpenTelemetry



whoami



Harry Kimpel

Principal Developer Relations Engineer

New Relic

@harrykimpel

//////

- Passionate software craftsman
- Microsoft developer ecosystem
- With New Relic since 2017
- Hiker, climber, biker, runner, swimmer, skier





Request:

"Based on what you know about me. draw a picture of what you think my **current life looks like**"

Response:

"Here's an illustration capturing your **developer life** surrounded by the **inspiring Bavarian Alps**. It shows a blend of productivity and natural inspiration—a high-tech setup with coding on screens, **observability dashboards**, and the beauty of **snowy peaks** outside."



Who uses Kafka?

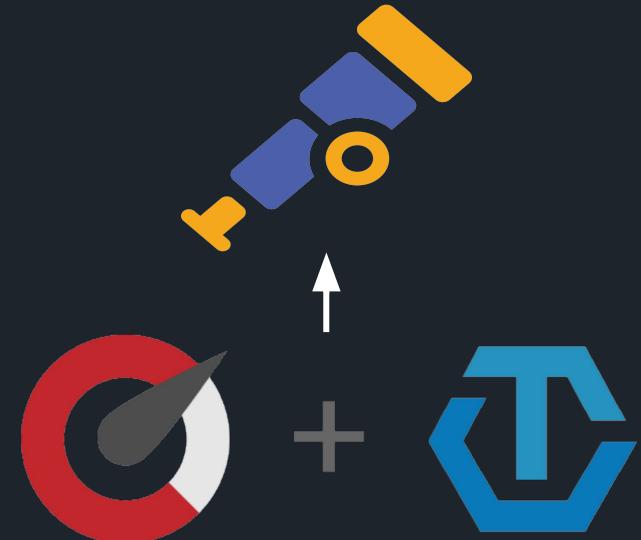
 **Hands up**
if you have heard
about
OpenTelemetry



Keep your
hands up
if you use
OpenTelemetry

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

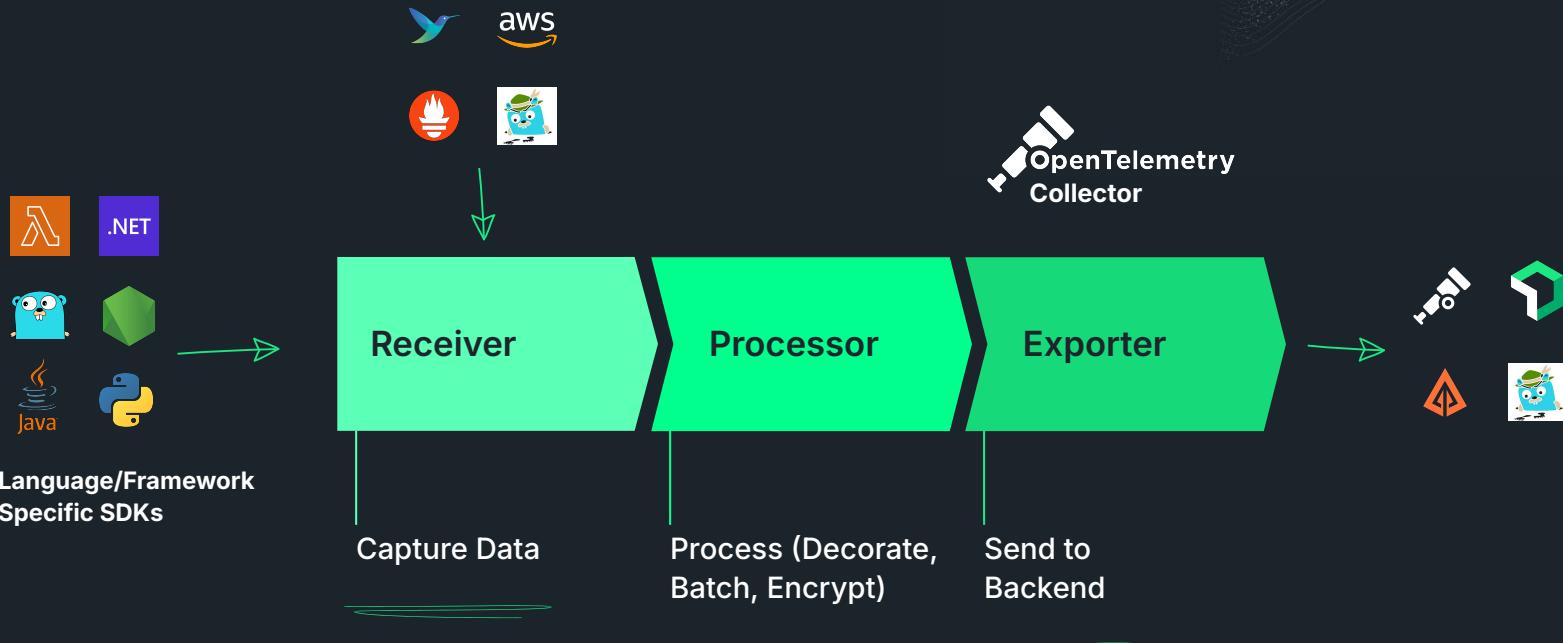
By 2025, **70%** of new cloud-native application monitoring will use open-source instrumentation

⁽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**
(manual instrumentation)

Ingredients

Custom Made

Flavours

**Java, .NET
and others
mature**



**End
Product**
(auto instrumentation)

Ready to Eat

Packaged

Scalable

**Most
Languages**



Screenshot from
<https://opentelemetry.io/docs/instrumentation/>
taken
January 23, 2025

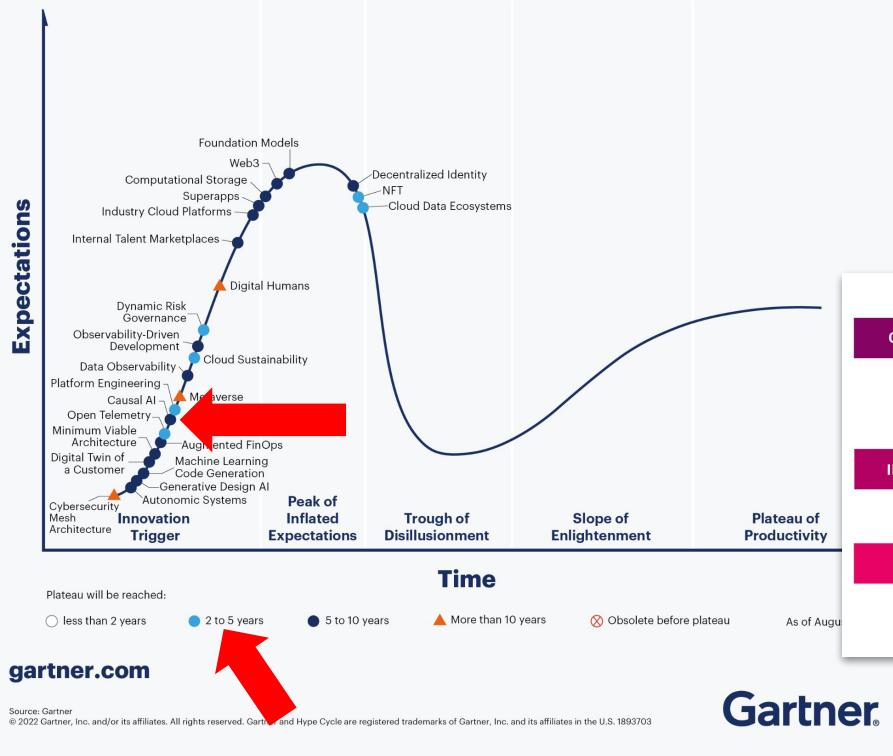
Status and Releases

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	Development	Development
Go	Stable	Stable	Beta
Java	Stable	Stable	Stable
JavaScript	Stable	Stable	Development
PHP	Stable	Stable	Stable
Python	Stable	Stable	Development
Ruby	Stable	Development	Development
Rust	Beta	Beta	Beta
Swift	Stable	Development	Development

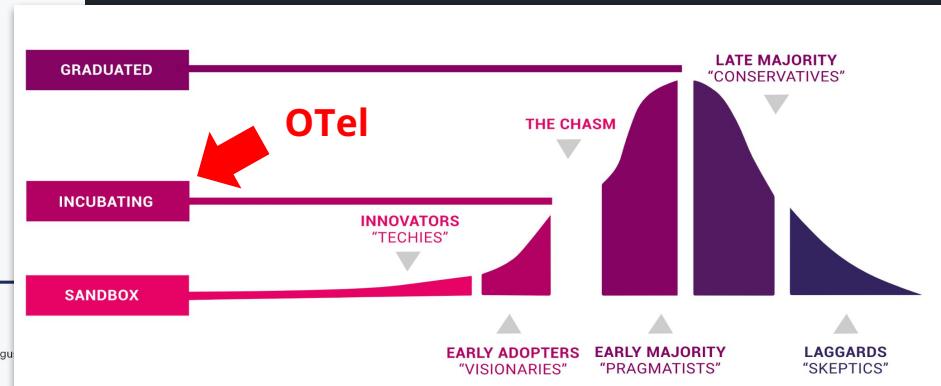
Design Considerations & Guidelines

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?



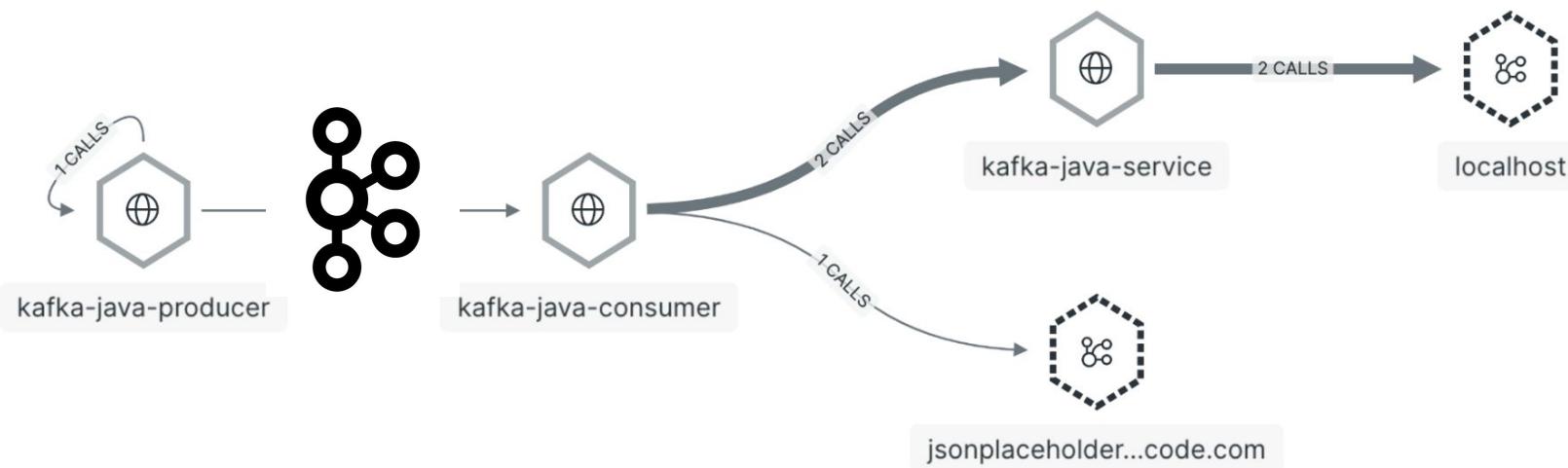
Gartner

new relic

Automatic instrumentation with Java

Entity map i

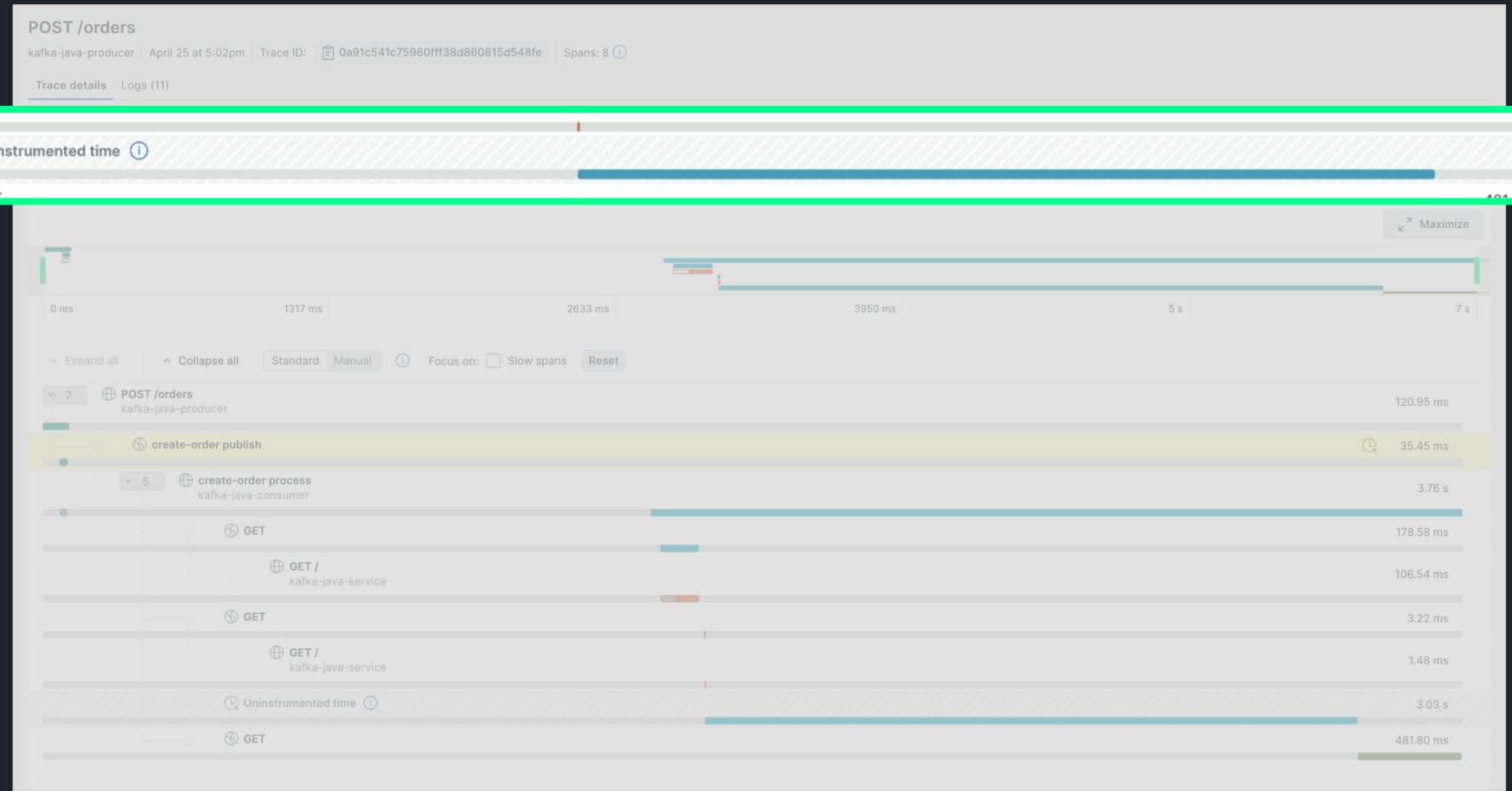
There are 5 sampled entities in this trace



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



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

SdkTracerProvider.sdkTracerProvider = SdkTracerProvider.builder()

Pieces: Comment | Pieces: Explain

```
.addSpanProcessor(BatchSpanProcessor.builder(OtlpGrpcSpanExporter.builder()
    .setEndpoint(
        otlpEndpoint)
    .addHeader(key:"api-key",
        otlpHeadersApiKey)
    .build()).build())
.setResource(resource)
.build();
```

```
Pieces: Comment | Pieces: Explain
40  @Bean
41  public OpenTelemetry openTelemetry() {
42      Resource resource = Resource.getDefault().toBuilder()
43          .put(ResourceAttributes.SERVICE_NAME, "kafka-java-producer")
44          .put(ResourceAttributes.SERVICE_VERSION, "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(
```

```
    .setTracerProvider(sdkTracerProvider)
    .setMeterProvider(sdkMeterProvider)
    .setLoggerProvider(sdkLoggerProvider)
    .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
    .buildAndRegisterGlobal());
86
87 }
88 }
```

Manual instrumentation with Java - Kafka producer

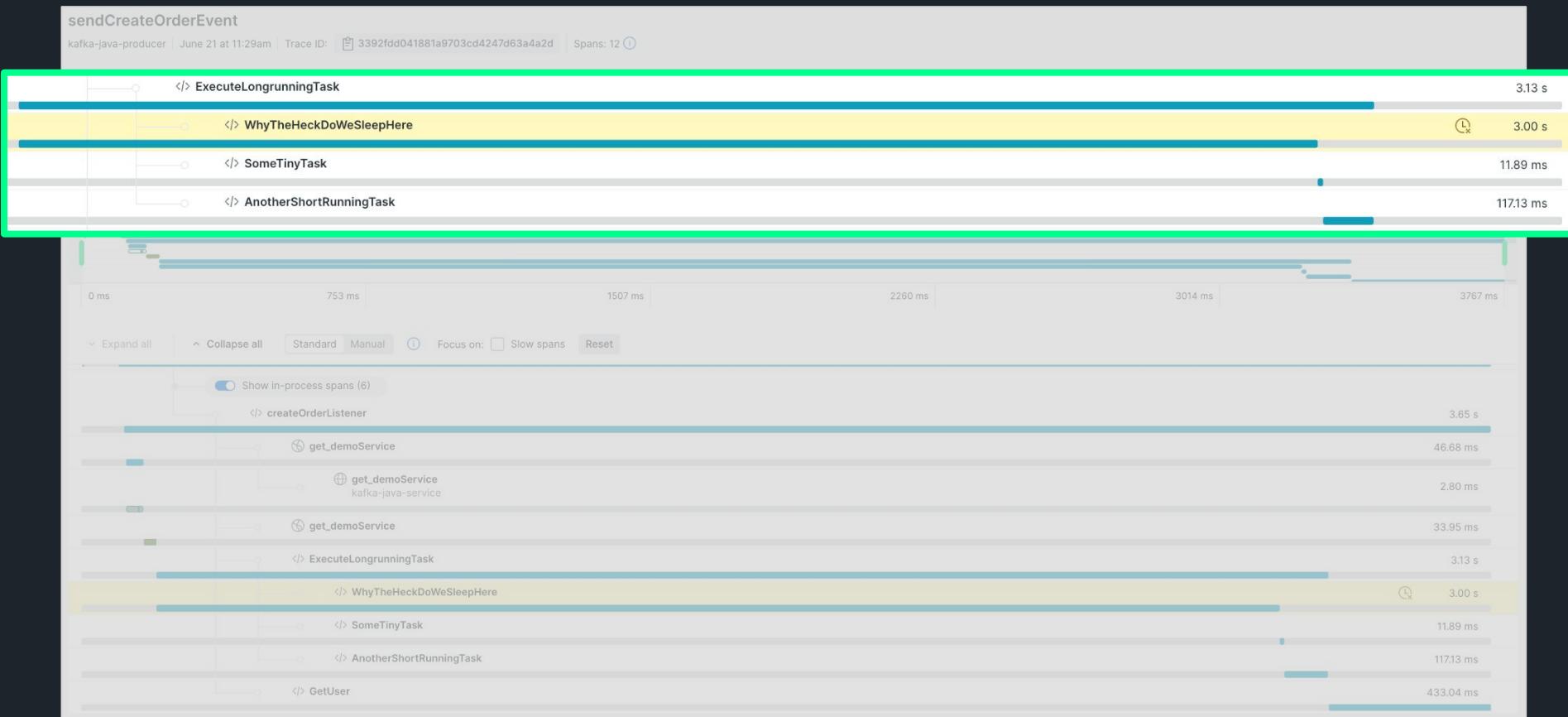
```
Pieces: Comment | Pieces: Explain
22  ...
23  @Bean
24  public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
25      Map<String, Object> config = new HashMap<>();
26      config.put(
27          org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
28          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      ...
34      value: JsonSerializer.class);
35      return new DefaultKafkaProducerFactory(config);
36  }
```

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

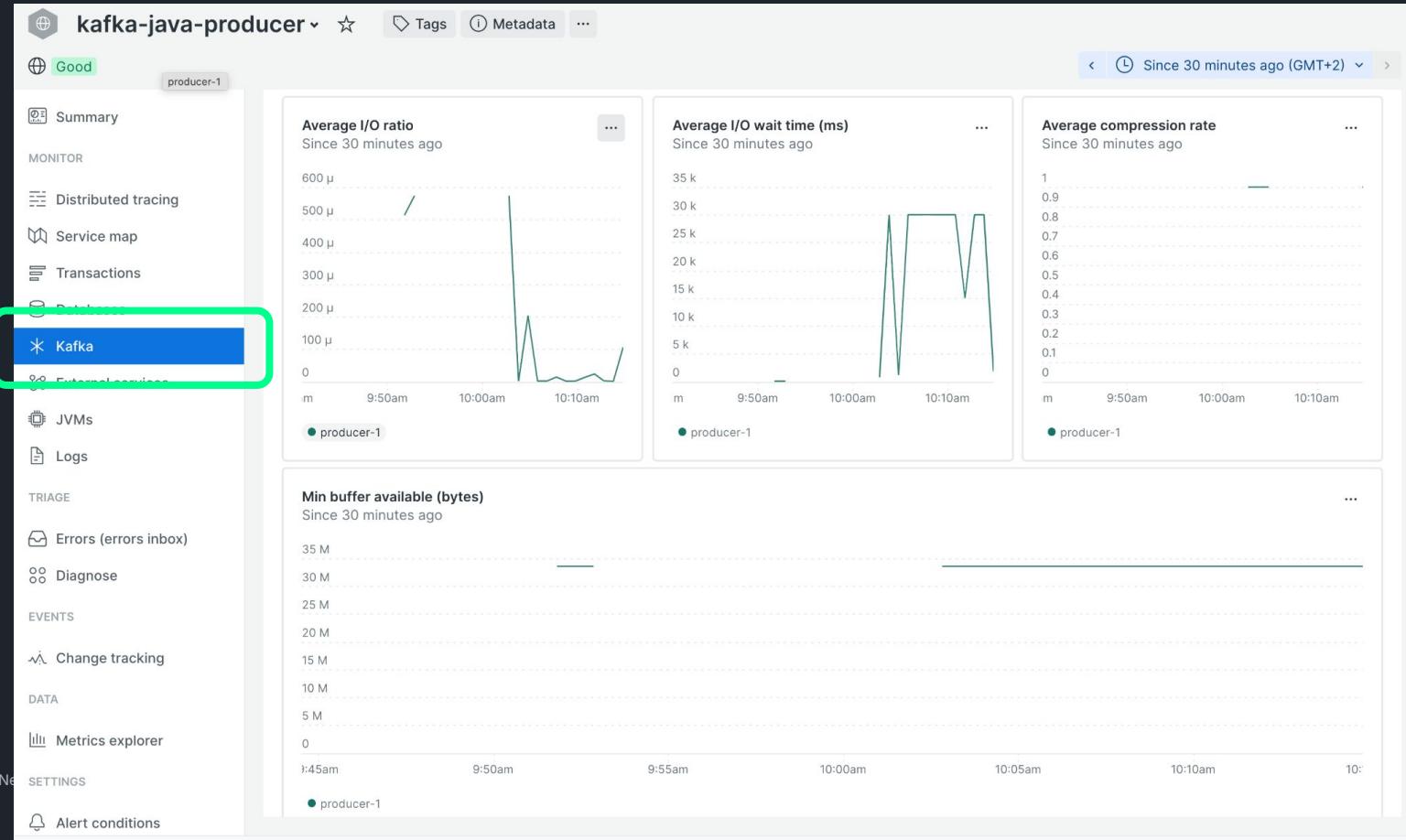


Manual instrumentation with Java

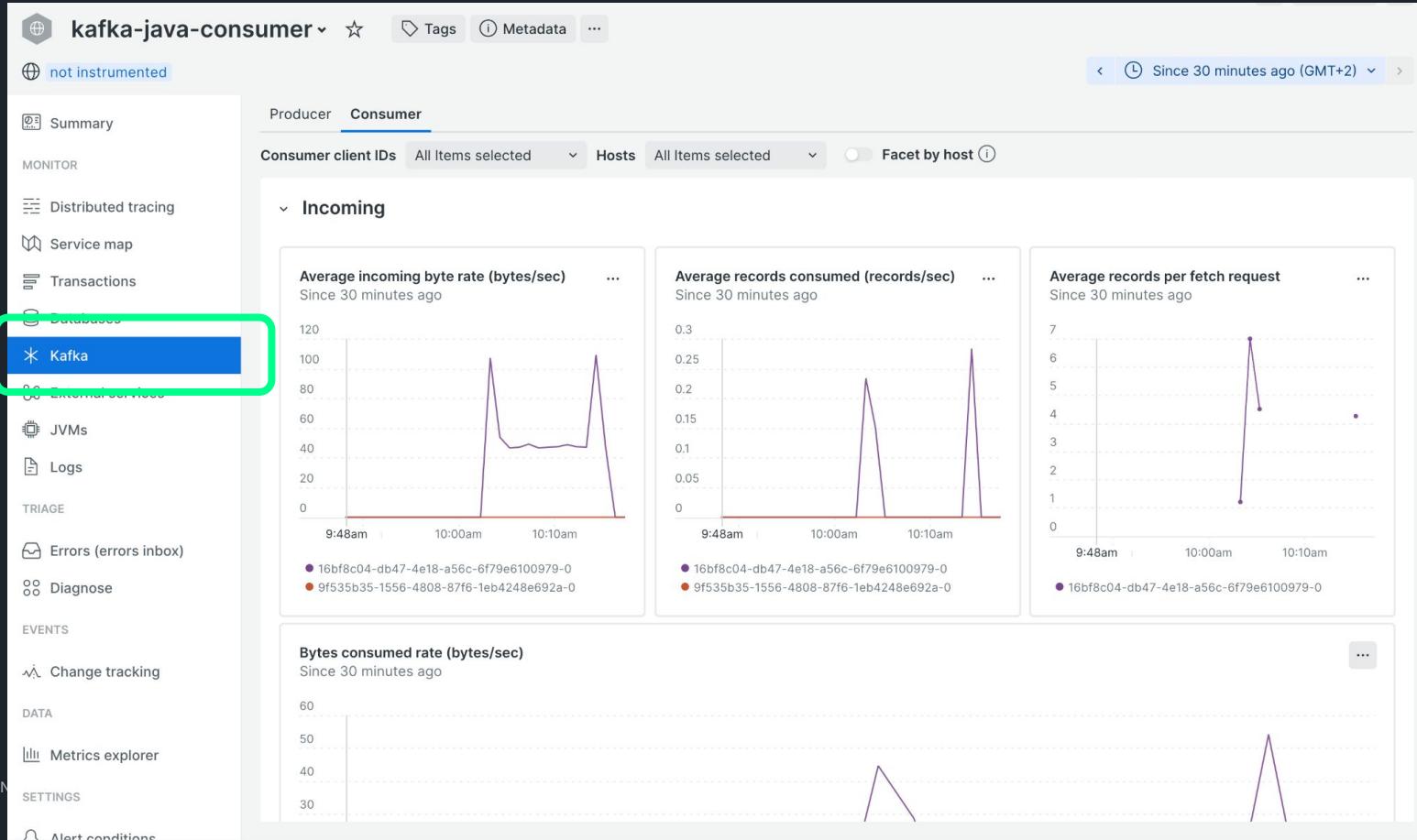
- custom spans

```
65  private void ExecuteLongrunningTask(Integer secondsToSleep) {  
66      Span span = tracer.spanBuilder(spanName:"ExecuteLongrunningTask").startSpan();  
67      // Make the span the current span  
68      try {  
69          // Pieces: Comment | Pieces: Explain  
70          Span sleepSpan = tracer.spanBuilder(spanName:"WhyThe Heck Do We Sleep Here")  
71              .setParent(Context.current().with(span))  
72              .startSpan();  
73          // Pieces: Comment | Pieces: Explain  
74          sleepSpan.setAttribute(key:"secondsToSleep", secondsToSleep);  
75          Thread.sleep(secondsToSleep * 1000);  
76          log.info("Executed some long running task that took " + secondsToSleep + " seconds to run.");  
77      } finally {  
78          sleepSpan.end();  
79      }  
80      // Pieces: Comment | Pieces: Explain  
81      SomeTinyTask(span);  
82      // Pieces: Comment | Pieces: Explain  
83      AnotherShortRunningTask(span);  
84  } catch (Exception t) {  
85      span.recordException(t);  
86      // throw t;  
87  } finally {  
88      span.end();  
89  }
```

Kafka metrics for producer



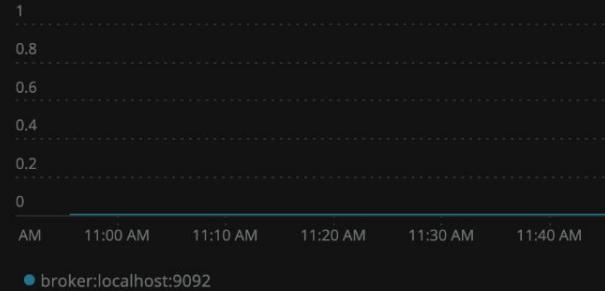
Kafka metrics for consumer



Kafka infrastructure metrics

Leader Election Per Second

Since 1 hour ago



Unclean Leader Election Per Second

Since 1 hour ago



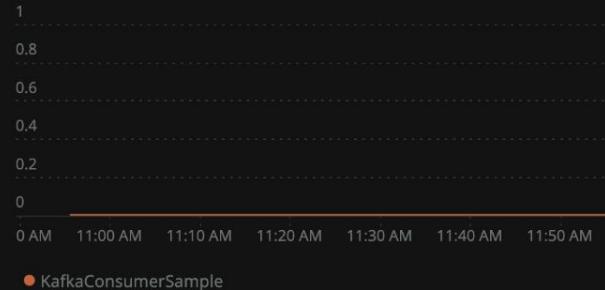
Topic Bytes Written

Since 1 hour ago



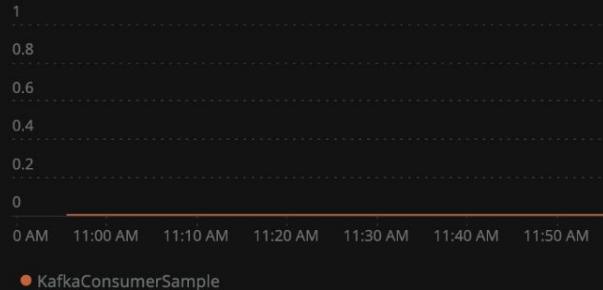
Consumer Minimum Requests Per Second

Since 1 hour ago



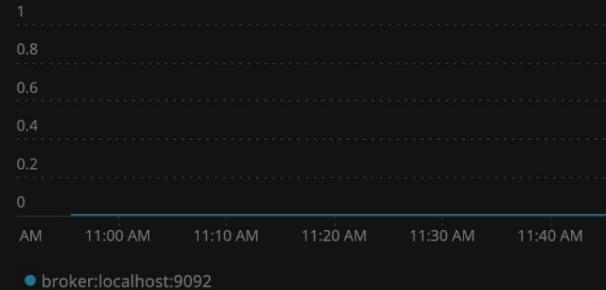
Consumer Max Lag

Since 1 hour ago



Unreplicated Partitions

Since 1 hour ago



Java supported libraries in OpenTelemetry

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

opentelemetry-java-instrumentation / docs / supported-libraries.md

Preview Code Blame 223 lines (200 loc) · 97.5 KB ·

Raw

• [Disabled instrumentations](#)

Libraries / Frameworks

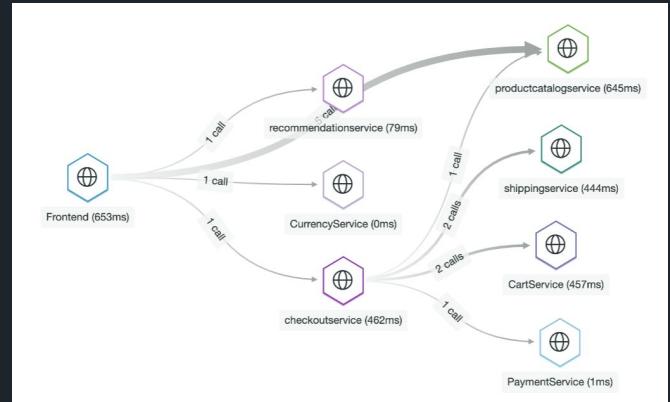
These are the supported libraries and frameworks:

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

Sample application <https://github.com/harrykimpel/java-kafka-otel-producer-consumer>

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](#)

Sign Up for Confluent Cloud

Get \$400 worth free credits
for your first 30 Days

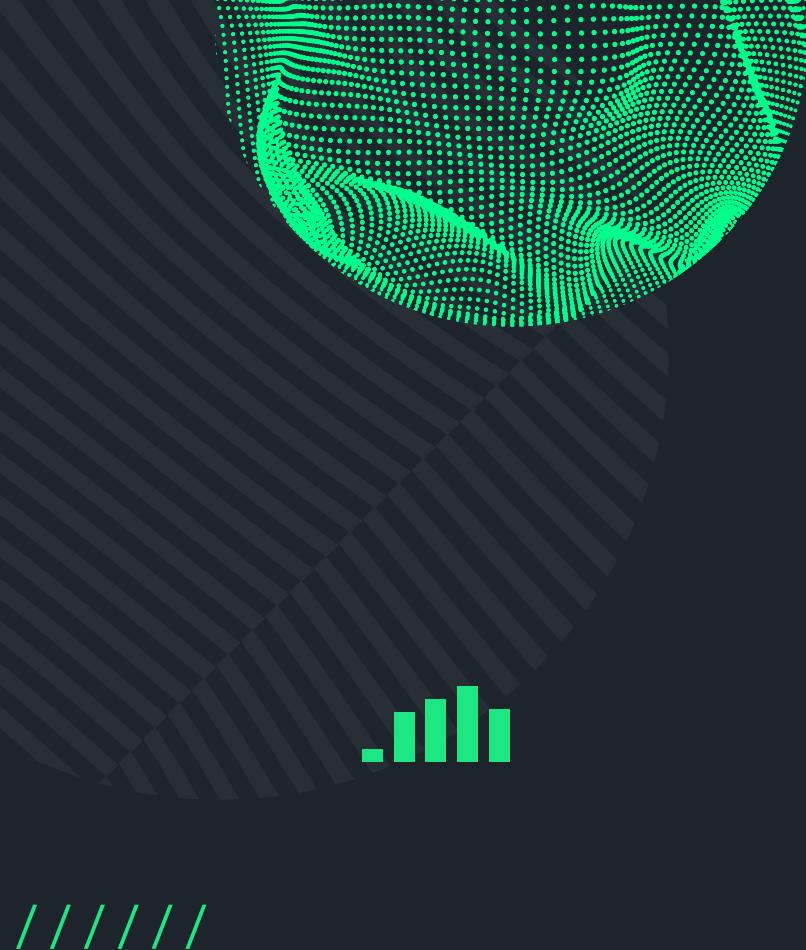
**Use Promo Code - POPTOUT000MZG62
to skip the paywall!**





Harry Kimpel
Principal Developer Relations Engineer
New Relic
@harrykimpel

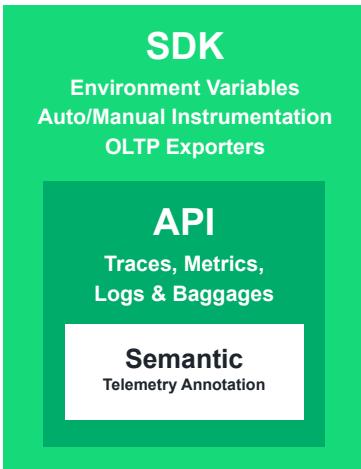




Appendix

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

The screenshot illustrates the New Relic distributed tracing interface, specifically focusing on the 'Logs' tab.

Anomalous Span Detection: A red box highlights a specific span in the trace map and the main timeline. The span is identified as `hipstershop.ShippingService/GetQuote` and is labeled as "about 286.46 ms slower than average (193% slow...)" compared to similar spans over the past 6 hours. This span is highlighted in purple in the main timeline.

Logs in Context: A red box highlights the right-hand panel, which shows logs for the selected entity. The title indicates the span `hipstershop.ShippingService/GetQuote` and the entity `checkoutservice`. The panel displays a performance chart for throughput (rps) over time, followed by a log table with two entries:

timestamp	hostname	message
20:53:54.975		CartService.GetCart UserId=e8fd0aa2-27b3-4721-bc39-0d514b3ddf2e
20:53:55.425		CartService.EmptyCart UserId=e8fd0aa2-27b3-4721-bc39-0d514b3ddf2e

A red arrow points from the 'Logs' tab in the top navigation bar to the 'Logs' section in the right-hand panel.

Logs in the Timeline: A red box highlights the bottom section of the interface, which shows logs integrated directly into the trace timeline. The logs are color-coded by type: blue for errors and green for all logs. The logs correspond to the entries shown in the detailed logs panel above.

Text Labels:

- 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 visibility into application stacks.

Top Left Trace Map: Shows a trace map for a shipping service call. It highlights 4 anomalous spans and 9 entities. The main span, `hipstershop.ShippingService/GetQuote`, has a duration of 434.72 ms. A red box highlights this span and its child spans: `CreateQuoteFromCount` (434.58 ms) and `CreateQuoteFromFloat` (334.18 ms).

Top Right Entity Preview: A modal for the same span shows a timeline of events with "Golden signals" and a graph of average duration over time.

Middle Left Trace Map: Shows a trace map for a `shipOrder` call. It highlights 4 anomalous spans and 9 entities. The main span has a duration of 0.07 ms. A red box highlights the `shipOrder` span.

Middle Right Entity Preview: A modal for the `shipOrder` span shows error details: `zipCode is invalid`. An arrow points to the `address` attribute value: `1600 Amphitheatre Parkway, Mountain View, CT, 4315`.

Bottom Text:

- 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]
```



```
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  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-5dd4b75bd-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]
```

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.

Validate K8s is running, and follow the instructions

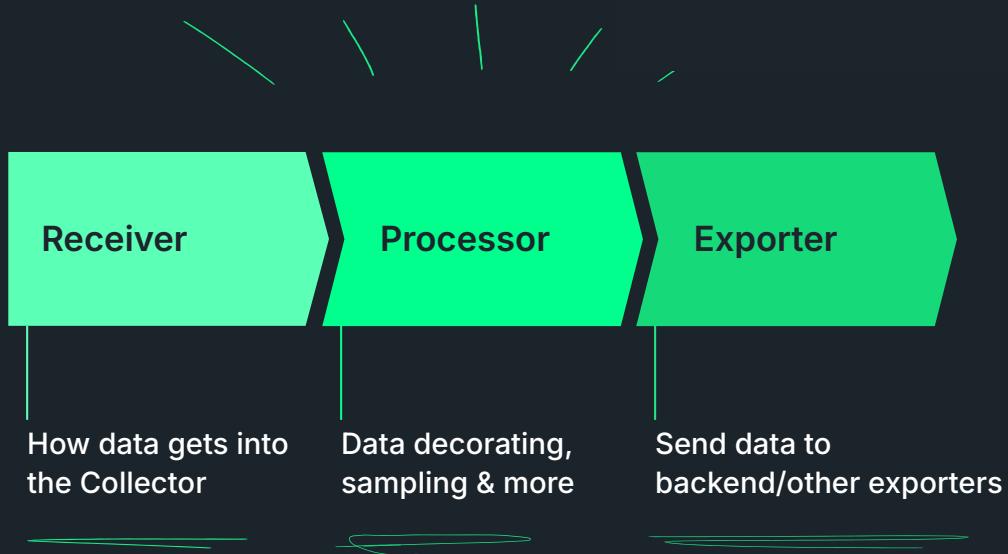
Important - deploy might take up to 10 mins

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

**Deploy the app, and generate load
(successful deployment)**

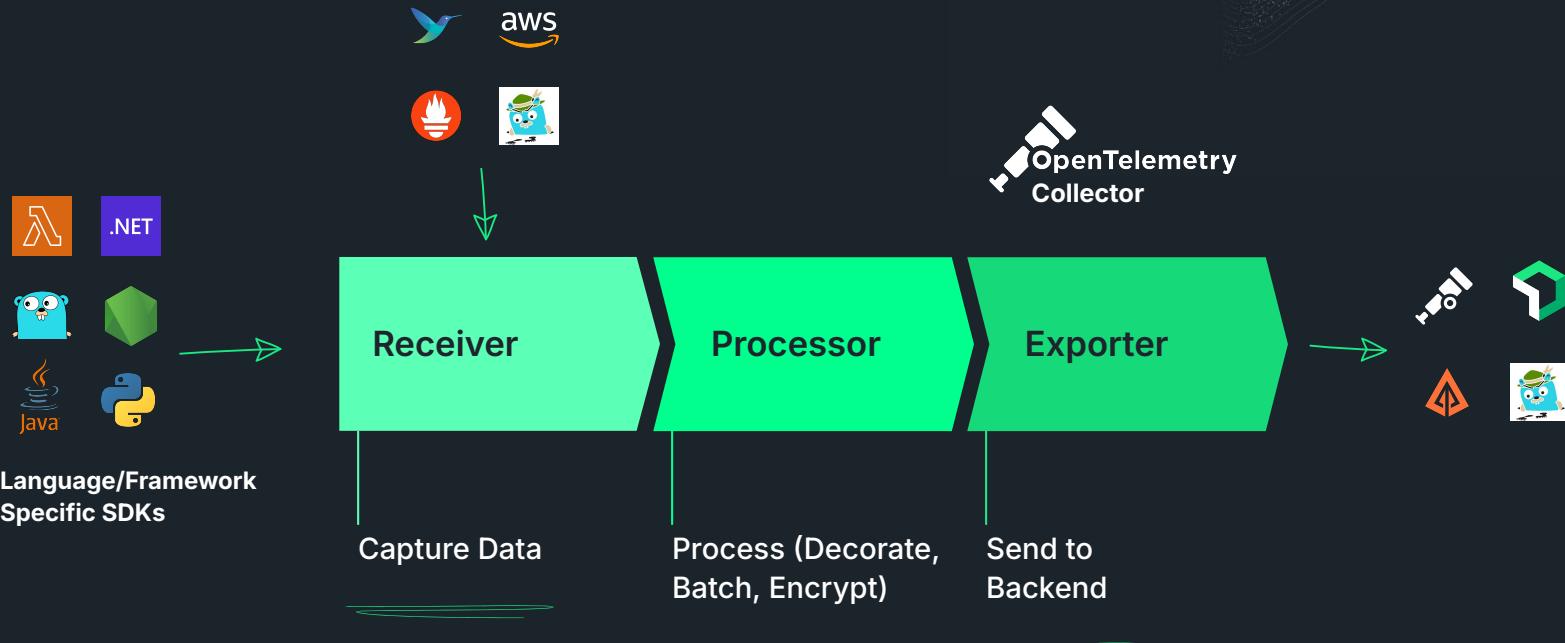
Collector Component

OpenTelemetry

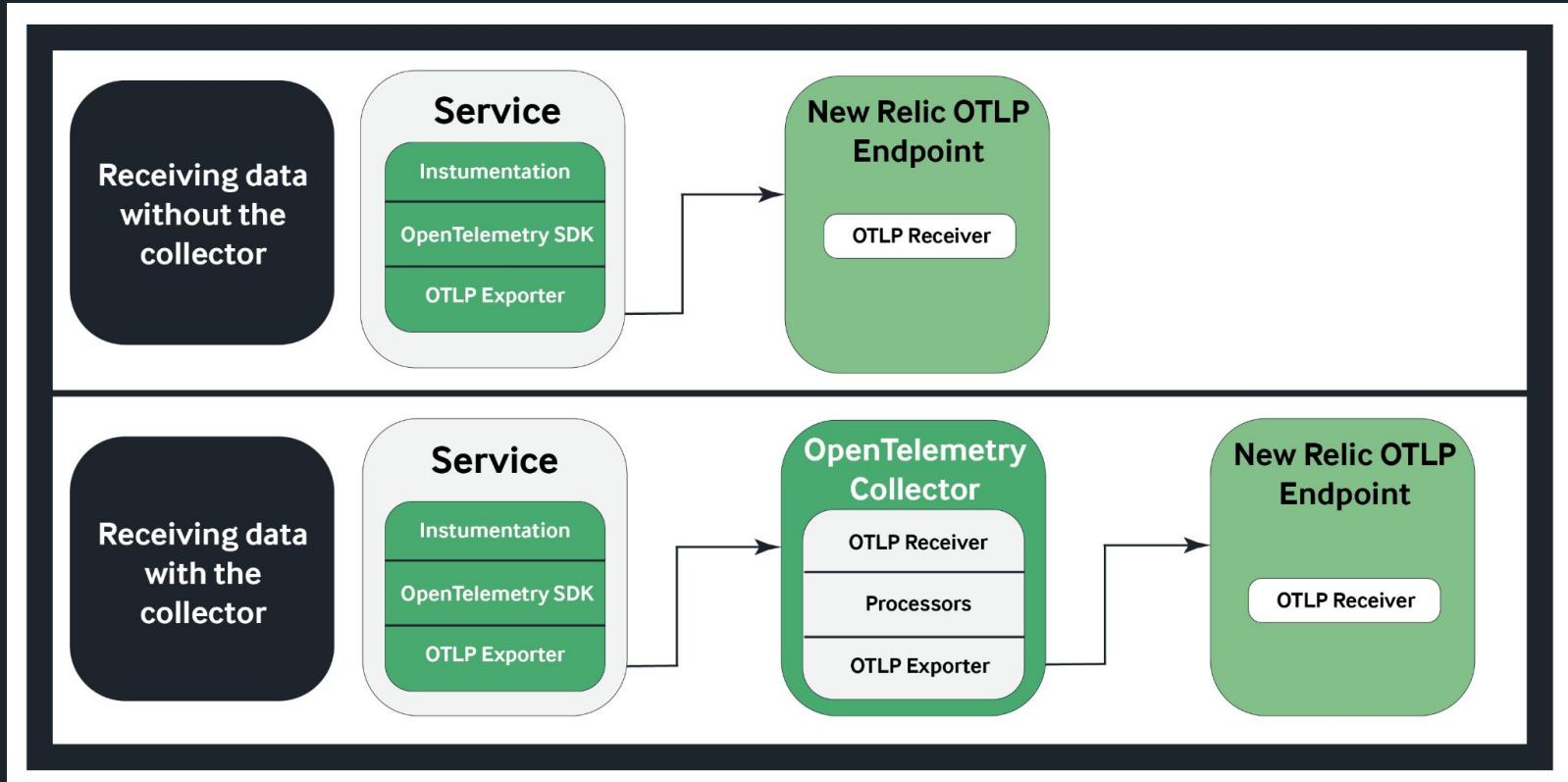


Putting it All Together

OpenTelemetry



OpenTelemetry - Collector



OpenTelemetry - Collector

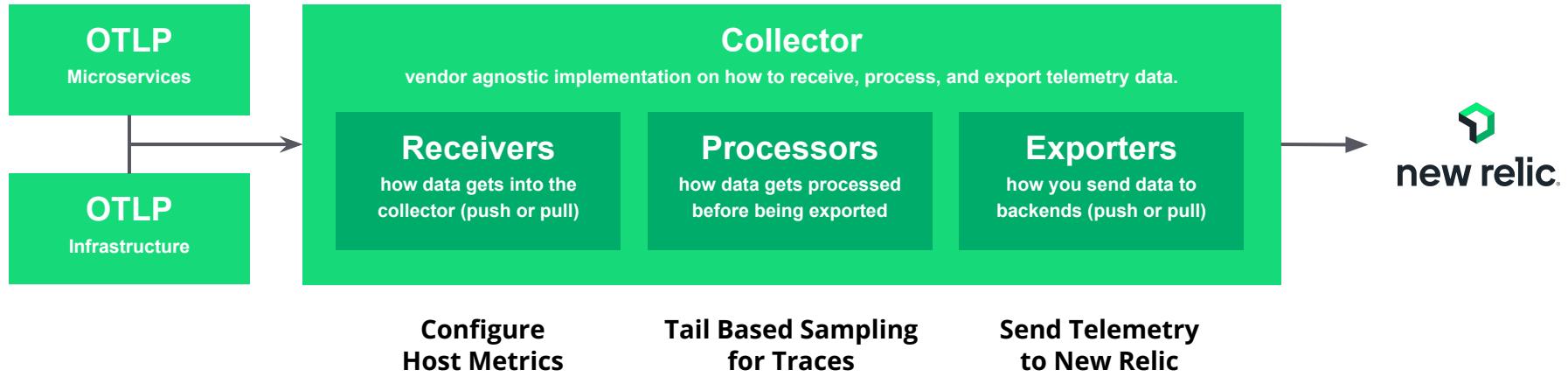
Benefits

- 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, and a green arrow points from the New Relic interface back to the configuration file, indicating a bidirectional relationship.

OpenTelemetry Collector Configuration (hostmetrics receiver):

```
receivers:  
  hostmetrics:  
    collection_interval: 20s  
    scrapers:  
      cpu:  
        metrics:  
          system.cpu.utilization:  
            enabled: true  
      load:  
      memory:  
        metrics:  
          system.memory.utilization:  
            enabled: true  
      disk:  
      filesystem:  
        metrics:  
          system.filesystem.utilization:  
            enabled: true  
      network:  
      paging:  
        metrics:  
          system.paging.utilization:  
            enabled: true  
      processes:  
      otlp:  
        protocols:  
          grpc:  
      processors:  
        batch:  
        cumulative_todelta:  
          metrics:  
            - system.network.io  
            - system.disk.operations  
            - system.network.dropped  
            - system.network.packets  
            - process.cpu.time
```

New Relic Data Explorer Metrics View:

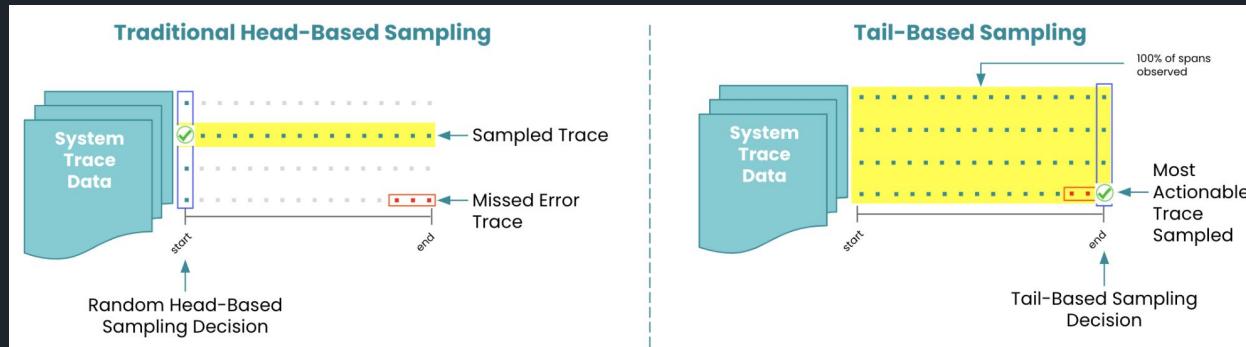
The Data Explorer interface displays the collected host memory utilization metric. The query is set to `latest('system.memory.utilization')` with a limit of 10. The chart shows the metric over time, with a pink line representing the latest data point. The Y-axis ranges from 0 to 0.15, and the X-axis shows time from 10:54am to 11:03am.

Metric	Value
system.memory.utilization	~0.14

Collect additional infrastructure metrics
for OpenTelemetry

HEAD vs TAIL Sampling

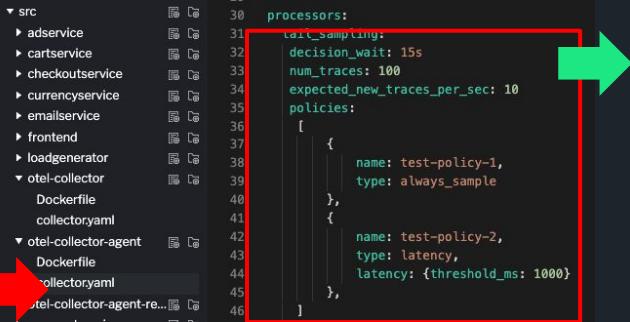
Storing all tracing data is **costly**. Most of the time, you
ONLY NEED THE RIGHT SAMPLING OF DATA



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



```
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.