



Tracing Kafka 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."

What is Tracing?

What is Tracing? Telemetry? Signals?

Traces

The path of a request through your application.

Metrics

A measurement
captured at runtime.

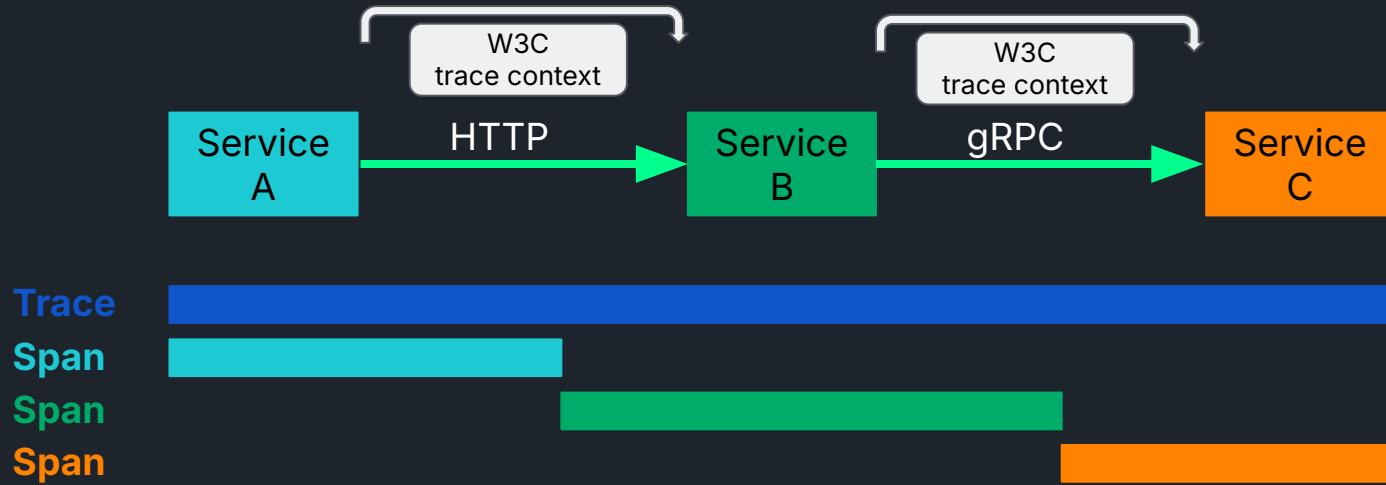
- Avg. CPU
 - Throughput
 - Max. Memory

Logs

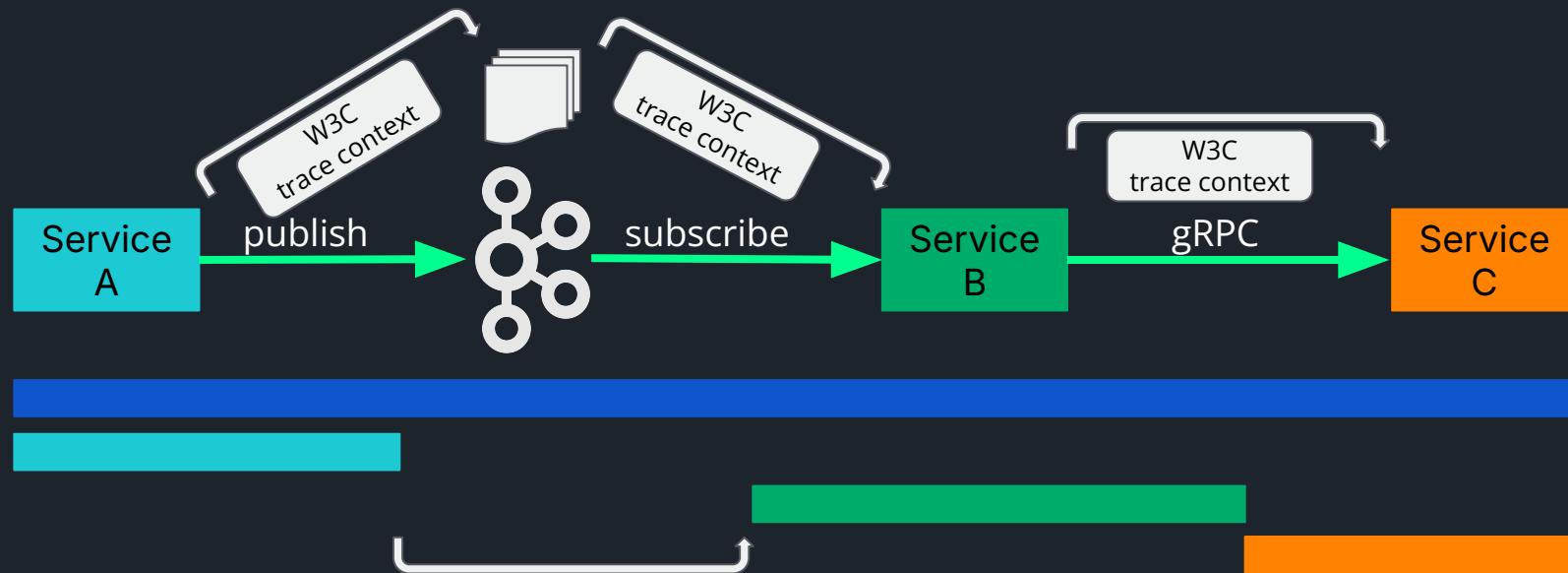
A recording of an event.

Timestamp	X	message	X
08:20:30.519		Startup check warning: Agent version out of date: v12.7.0, in order access up to date feature.	
08:21:16.287		Feb 08 08:21:16 ip-172-17-25-72 dhclient[420]: XMT: Solicit on eth0, interval 11633ms.	
08:21:45.143		Feb 08 08:21:45 ip-172-17-16-8 dhclient[437]: XMT: Solicit on eth0, interval 126620ms.	
08:23:13.216		Feb 08 08:23:13 ip-172-17-25-72 dhclient[420]: XMT: Solicit on eth0, interval 117750ms.	
08:23:51.797		Feb 08 08:23:51 ip-172-17-16-8 dhclient[437]: XMT: Solicit on eth0, interval 128740ms.	
08:29:01.143		Feb 08 08:29:01 ip-172-17-16-149 CRON[264308]: pam_unix(crond-session): session opened for user root.	
08:29:01.148		Feb 08 08:29:01 ip-172-17-16-149 CRON[264308]: (root) CMD (command never - /dev/null &)	
08:29:01.150		Feb 08 08:29:01 ip-172-17-16-149 CRON[264308]: pam_unix(crond-session): session closed for user root.	
08:29:01.990		Feb 08 08:29:01 ip-172-17-16-149 CRON[298986]: pam_unix(crond-session): session opened for user root.	
08:29:01.997		Feb 08 08:29:01 ip-172-17-16-149 CRON[298986]: (root) CMD (command never - /dev/null &)	
08:29:01.998		Feb 08 08:29:01 ip-172-17-16-149 CRON[298986]: pam_unix(crond-session): session closed for user root.	
08:29:01.999		Feb 08 08:29:01 ip-172-17-16-149 CRON[298986]: (root) CMD (command never - /dev/null &)	
08:25:11.019		Feb 08 08:25:11 ip-172-17-21-16 dhclient[420]: XMT: Solicit on eth0, interval 115800ms.	
08:26:00.637		Feb 08 08:26:00 ip-172-17-16-14 dhclient[437]: XMT: Solicit on eth0, interval 12938ms.	
08:26:24.154		Feb 08 08:26:24 ip-172-17-16-14 dhclient[346]: DHCPREQUEST for 172.16.1.8 on eth0 to 172.16.1.1.	
08:26:24.169		Feb 08 08:26:24 ip-172-17-16-14 dhclient[346]: bound to 172.16.1.8 - renew in 163 seconds.	
08:26:37.591		Feb 08 08:26:37 ip-172-16-13-19 sshd[202616]: Invalid user from 54.36.6.19 port 50368	
08:26:45.248		Feb 08 08:26:45 ip-172-16-13-19 sshd[202616]: Connection closed by invalid user 54.36.6.19...p.39	
08:27:04.715		Feb 08 08:27:04 ip-172-17-21-16 dhclient[420]: XMT: Solicit on eth0, interval 112740ms.	
08:27:31.972		Feb 08 08:27:31 ip-172-17-16-13-19 amazon-sns-agent[360]: 2025-02-28 08:27:31.11...	

What is Tracing?



How is Tracing for event-based systems different?



Who uses Kafka?



new relic[®] uses Kafka!

 more than **15 million**
messages per second

 aggregate data rate
approaching **1 Tbps**

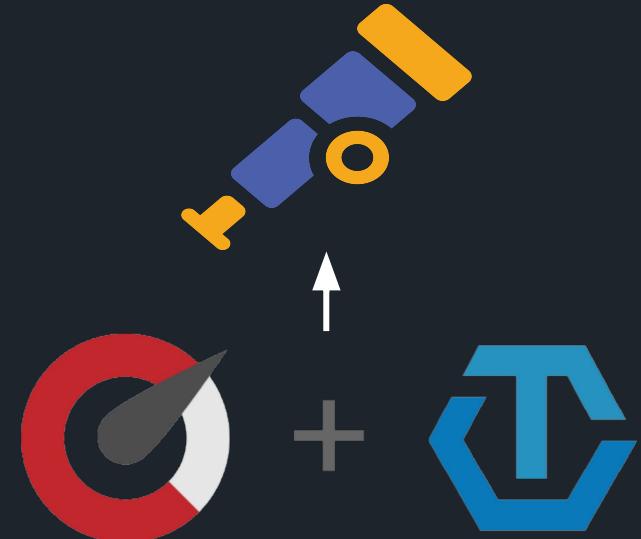
 **Hands up**
if you have heard
about
OpenTelemetry



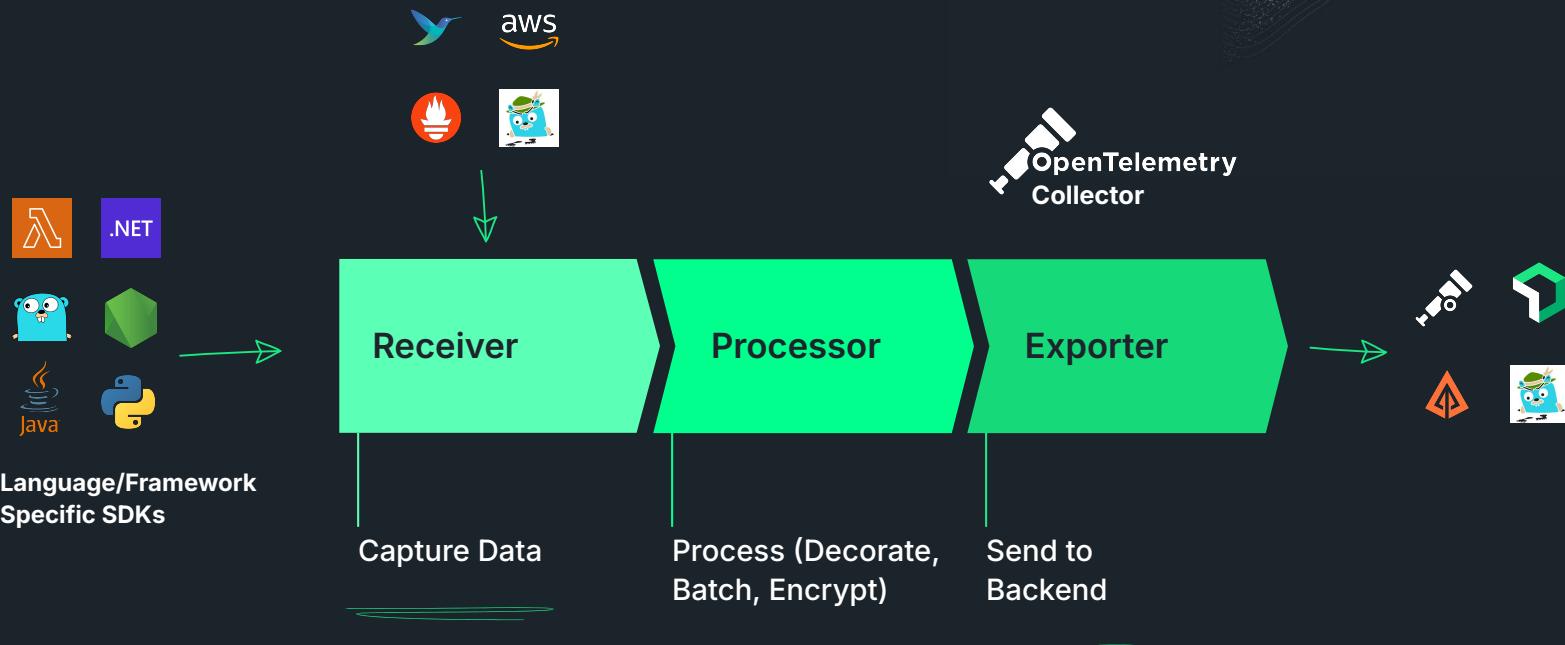
Keep your
hands up
if you use
OpenTelemetry

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



Snapshot of OpenTelemetry

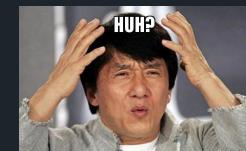


The Rise of OpenTelemetry



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**



Zero-Code
(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

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

↑ Top

• [Disabled instrumentations](#)

Libraries / Frameworks

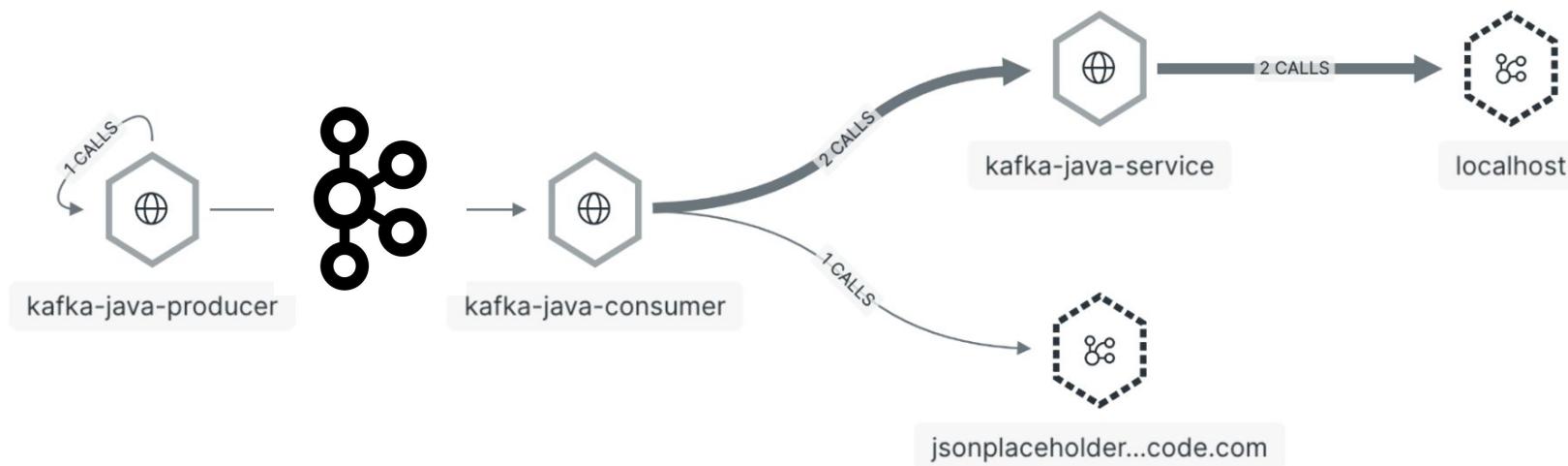
These are the supported libraries and frameworks:

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]

Automatic instrumentation with Java

Entity map i

There are 5 sampled entities in this trace





Automatic instrumentation - aka zero-code instrumentation - with Java

<https://opentelemetry.io/docs/zero-code/>



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
```

POST /orders

kafka-java-producer | February 26 at 7:58am | Trace ID: 9d6027b1fb5326d40bceba687119449e | Spans: 10 ⓘ

Trace details Logs (14)

0 anomalous spans ⓘ

0 errors ⓘ

4 entities ⓘ

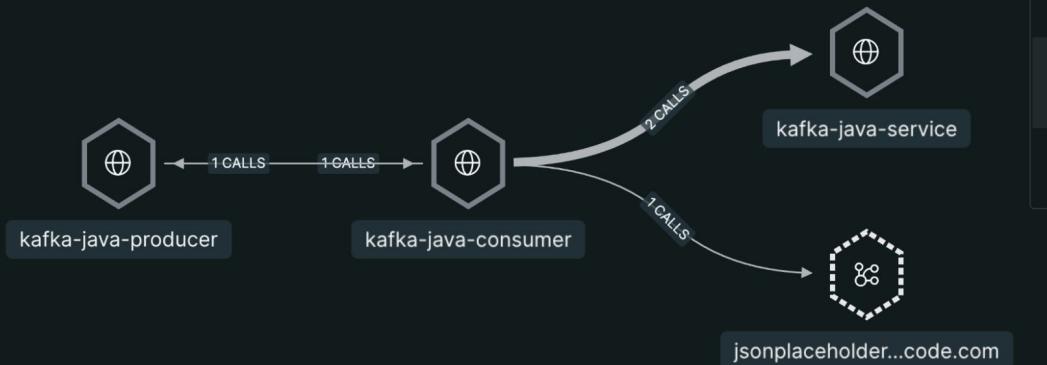
Find spans by name or ID

Trace duration 4402.35 ms Backend duration 4402.35 ms

Entity map ⓘ

There are 4 sampled entities in this trace

Map Timeline Latency



+
-
✖

Expand all

Collapse all

Standard

Manual



Reset

Maximize

> 9 ⚡ POST /orders

kafka-java-producer

20.49 ms

> 7 ⚡ create-order process

kafka-java-consumer

4.37 s

⌚ GET /

kafka-java-service

1.58 ms

⌚ GET /

kafka-java-service

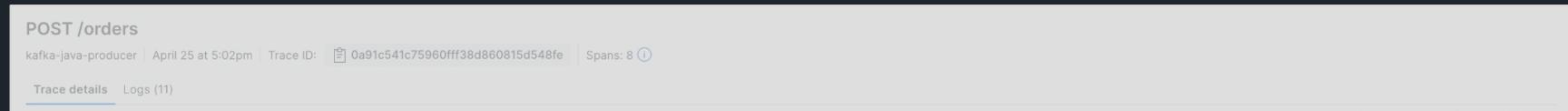
1.25 ms

⌚ user process

kafka-java-producer

11.35 ms

Automatic instrumentation with Java



⌚️ Uninstrumented time ⓘ

This section highlights "Uninstrumented time" with a clock icon and an 'x' symbol.



Manual instrumentation with Java



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



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

```
Resource resource = Resource.getDefault().toBuilder()
```

```
    .put(ResourceAttributes.SERVICE_NAME, "kafka-java-producer")
    .put(ResourceAttributes.SERVICE_VERSION, "0.1.0")
    .put(key:"otel.jmx.target.system", value:"tomcat,kafka-broker")
    .put(key:"otel.instrumentation.kafka.metric-reporter.enabled", value:true).build();
```

```
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(
51  +--                 otlpEndpoint)
52  +--             .addHeader(key:"api-key",
53  +--                 otlpHeadersApiKey)
54  +--             .build()).build())
55  +--         .setResource(resource)
56 }
```

```
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

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(
```

```
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

```
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(
51  +-->                 otlpEndpoint)
52  +-->             .addHeader(key:"api-key",
53  +-->                 otlpHeadersApiKey)
54  +-->         .build()).build())
```

```
OpenTelemetry openTelemetry := OpenTelemetrySdk.builder()
```

```
→     .setTracerProvider(sdkTracerProvider)
→     .setMeterProvider(sdkMeterProvider)
→     .setLoggerProvider(sdkLoggerProvider)
→     .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
→     .buildAndRegisterGlobal();
```

```
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

sendCreateOrderEvent

kafka-java-producer | June 21 at 11:29am | Trace ID: 3392fdd041881a9703cd4247d63a4a2d | Spans: 12 ⓘ

</> ExecuteLongrunningTask

</> WhyTheHeckDoWeSleepHere

</> SomeTinyTask

</> AnotherShortRunningTask

get_demoService
kafka-java-service

2.80 ms

get_demoService

33.95 ms

</> ExecuteLongrunningTask

3.13 s

</> WhyTheHeckDoWeSleepHere

3.00 s ⓘ

</> SomeTinyTask

11.89 ms

</> AnotherShortRunningTask

11713 ms

</> GetUser

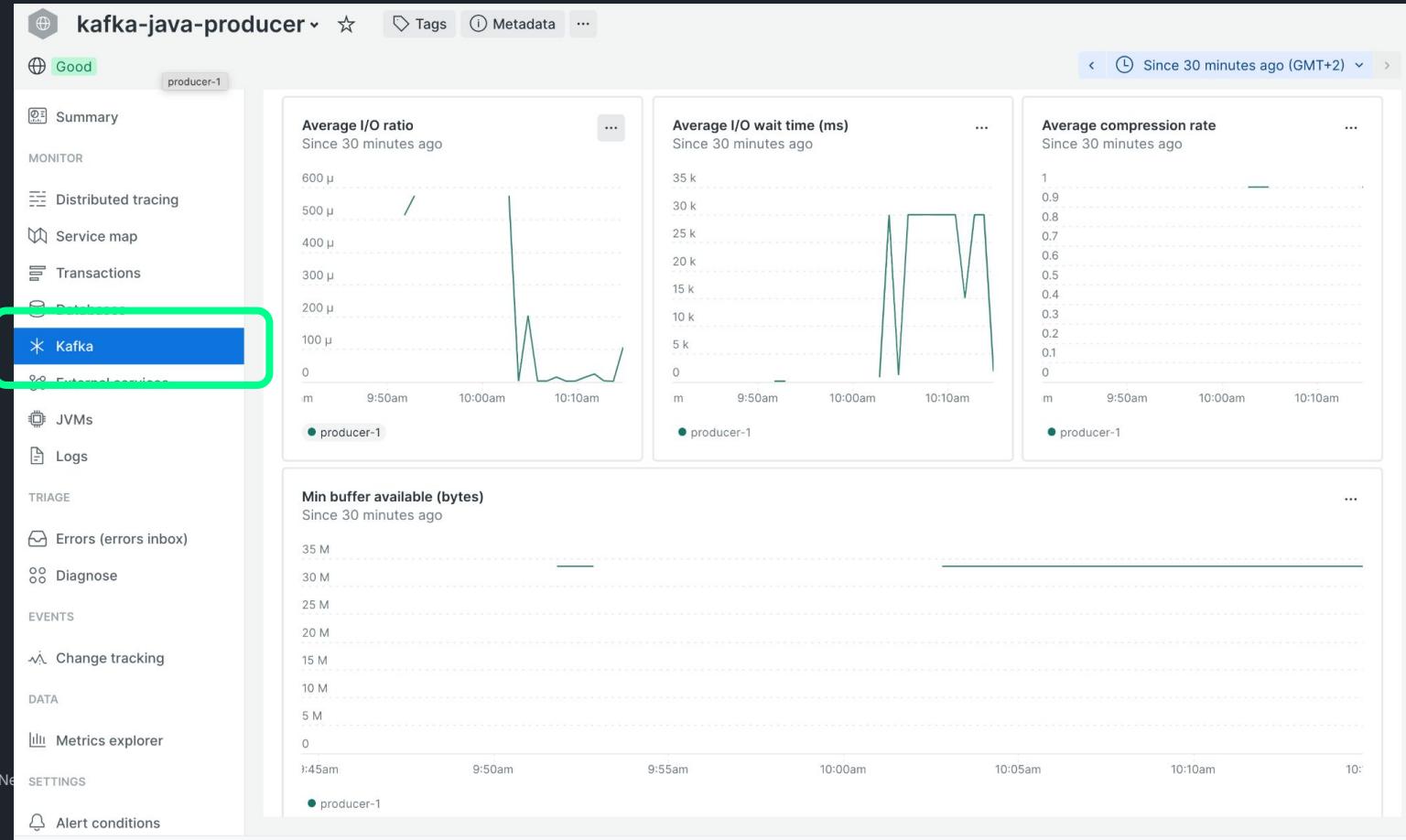
433.04 ms

Manual instrumentation with Java

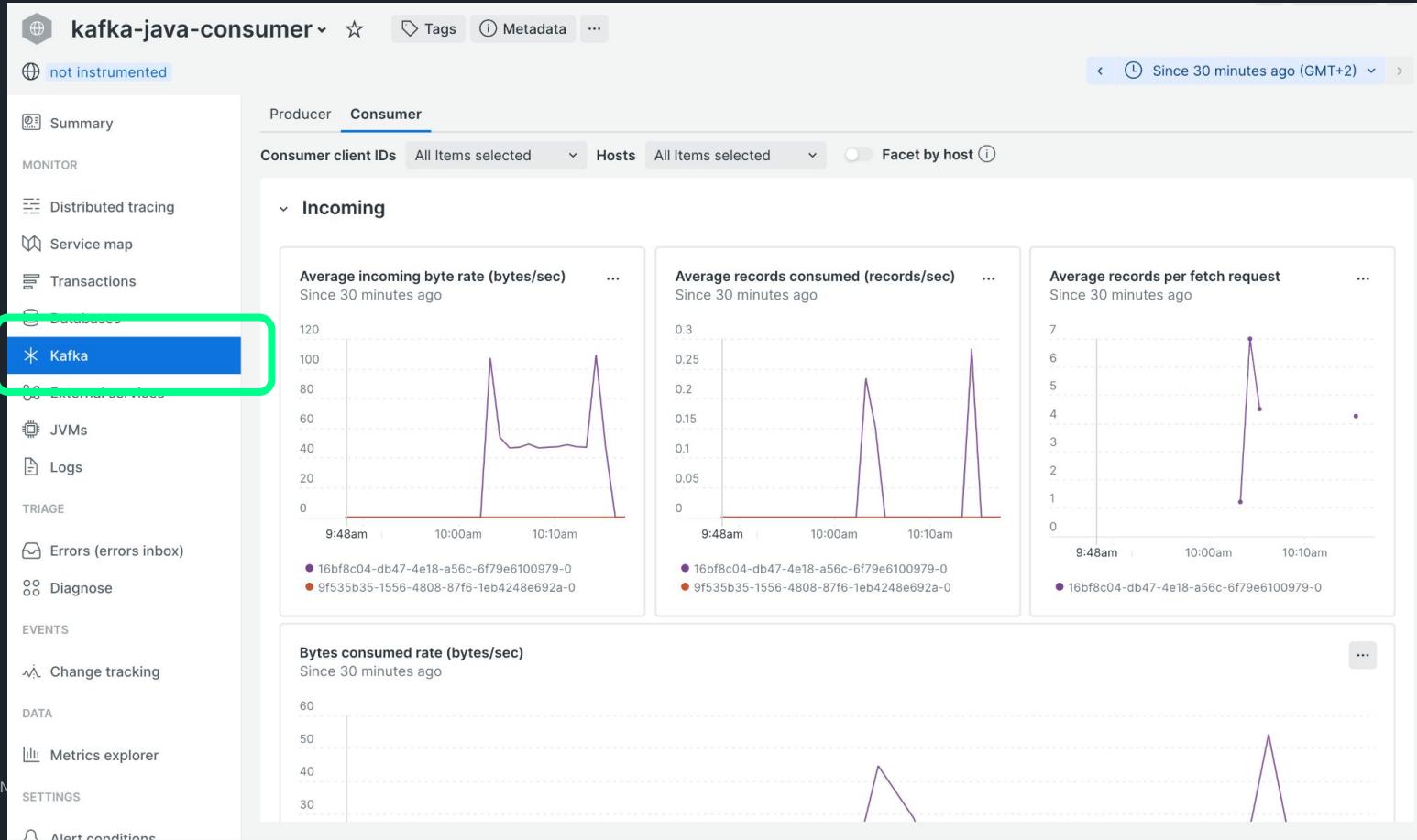
- custom spans

```
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



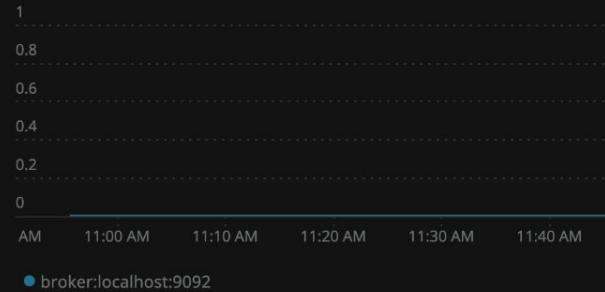
Kafka metrics for consumer



Kafka infrastructure metrics

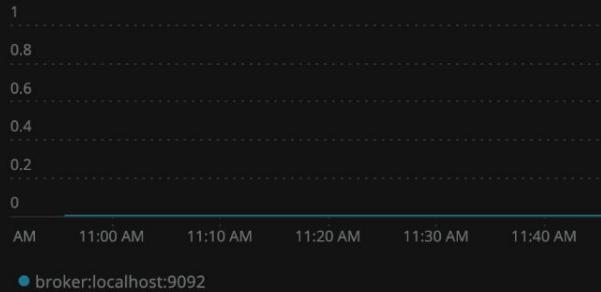
Leader Election Per Second

Since 1 hour ago



Unclean Leader Election Per Second

Since 1 hour ago



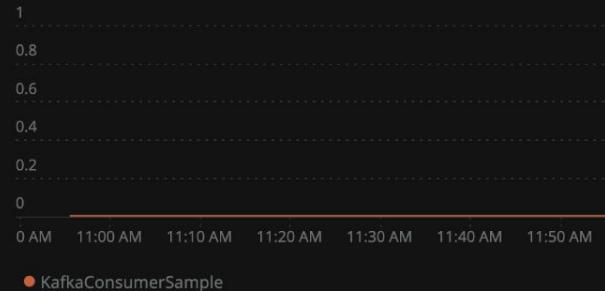
Topic Bytes Written

Since 1 hour ago



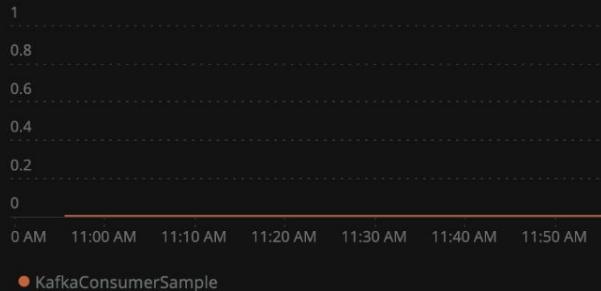
Consumer Minimum Requests Per Second

Since 1 hour ago



Consumer Max Lag

Since 1 hour ago



Unreplicated Partitions

Since 1 hour ago



Other event-based systems?

Microsoft Azure Event Hub

EventHubs.message
eventhub-publish | February 24 at 4:51pm | Trace ID: `fbef513d8a6e2a97b23f55af5cbf74d0` | Spans: 2 ⓘ

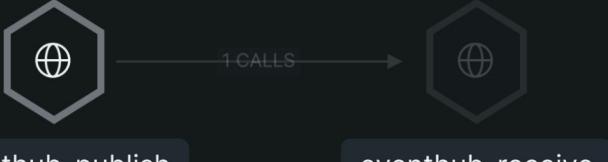
Trace details Logs (0)

0 anomalous spans 0 errors 2 entities Find spans by name or ID

Trace duration Backend duration
1641.08 ms 1641.08 ms

Entity map ⓘ
There are 2 sampled entities in this trace

Map Timeline Latency



eventhub-publish eventhub-receive

...

Expand all Collapse all Standard Manual ⓘ Reset Maximize

1 EventHubs.message eventhub-publish 0.09 ms

EventHubs.process eventhub-receive 0.08 ms

EventHubs.message
eventhub-publish

Performance Attributes Details DURATION **86 µs**

newrelic.source
api.traces.pttp

otel.library.name
azure.core.tracing.ext.opentelemetry_span

otel.library.version
1.0.0b11

process.id
4e47acc63ee788c1

service.name
eventhub-publish

span.kind
producer

telemetry.auto.version
0.51b0

telemetry.sdk.language
python

telemetry.sdk.name
opentelemetry

Google Pub/Sub

otel create

demoProducer | February 24 at 1:12pm | Trace ID: 4a005898f8210570fd1a7a1a2d20b75c | Spans: 5 ⓘ

Trace details Logs (0)

0 anomalous spans 0 errors 2 entities Find spans by name or ID

Entity map ⓘ There are 2 sampled entities in this trace

Map Timeline Latency

otel create demoProducer 254.03 ms DURATION

Attributes

newrelic.source api.traces.otlp

otel.library.name com.google.cloud.pubsub.v1

otel.library.version

process.id 37ab546d9886b5eb

service.name demoProducer

span.kind producer

telemetry.sdk.language java

telemetry.sdk.name opentelemetry

Expand all Collapse all Standard Manual ⓘ Res Maximize

> 4 otel create demoProducer 254.03 ms

> 2 otel-sub subscribe demoConsumer 96.62 ms

© 2025 New Relic, Inc. A new relic

RabbitMQ / (Cloud)AMQP

GET /amqp

web-frontend | February 25 at 9:31pm | Trace ID: 390d46d0e941da55c14ff4ed82e684cc | Spans: 5 ⓘ

Trace details Logs (7)

0 anomalous spans 0 errors 3 entities Find spans by name or ID

Entity map ⓘ

There are 3 sampled entities in this trace.

Map Timeline Latency

web-frontend → api-amqp-producer → api-amqp-consumer

...

Expand all Collapse all Standard Manual ⓘ Reset Maximize

440.35 ms

439.35 ms

438.67 ms

0.01 ms

0.01 ms

438.67 ms

0.01 ms

0.01 ms

queue1 receive

api-amqp-consumer

GET /amqpproducer

api-amqp-producer

438.67 ms DURATION

Performance Attributes Details

instrumentation.provider opentelemetry

name GET /amqpproducer

network.protocol.version 1.1

newRelic.ingestPoint api.traces.otlp

newrelic.source api.traces.otlp

otel.library.name Microsoft.AspNetCore

otel.library.version

parent.id 7c41d15d412ba0aa

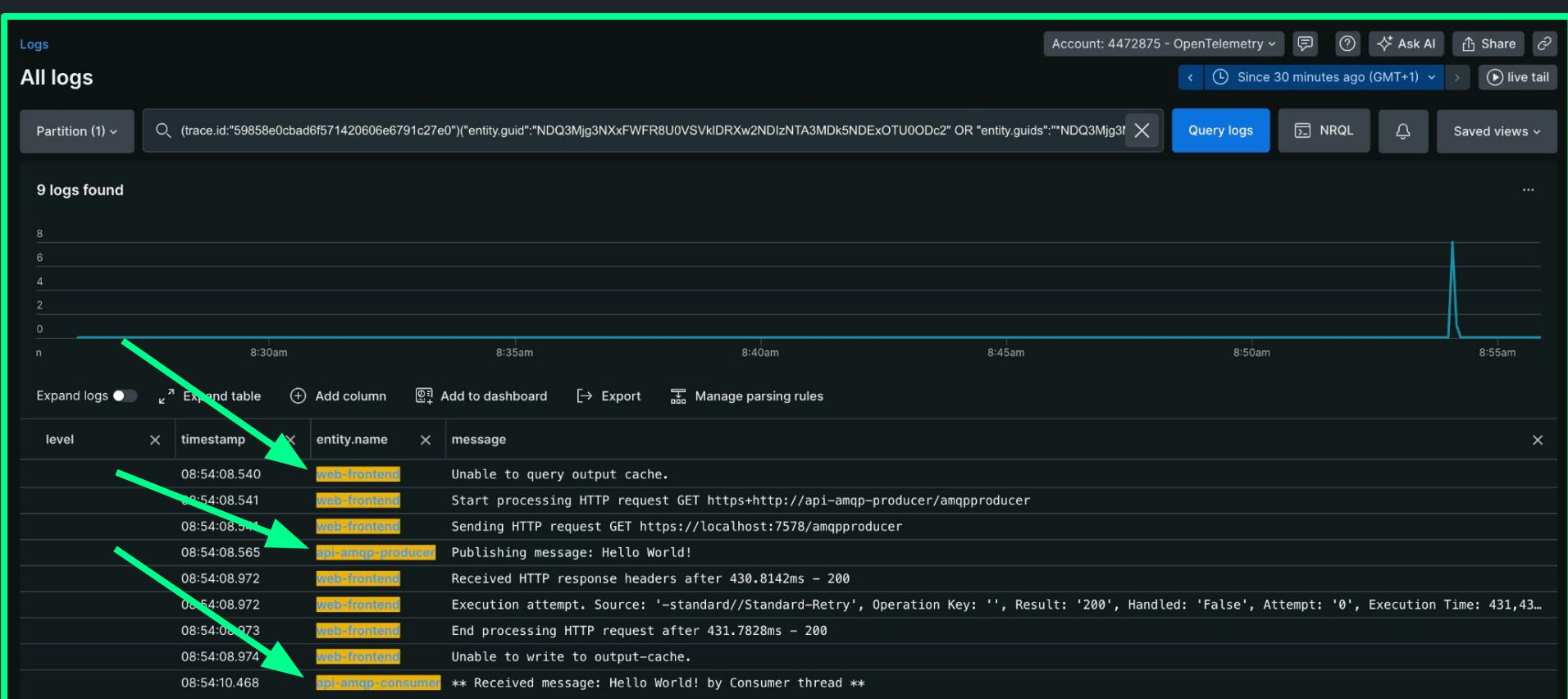
process.id b2ee2d11ab2a80b6

server.address localhost

server.port 7578

© 2024 new relic

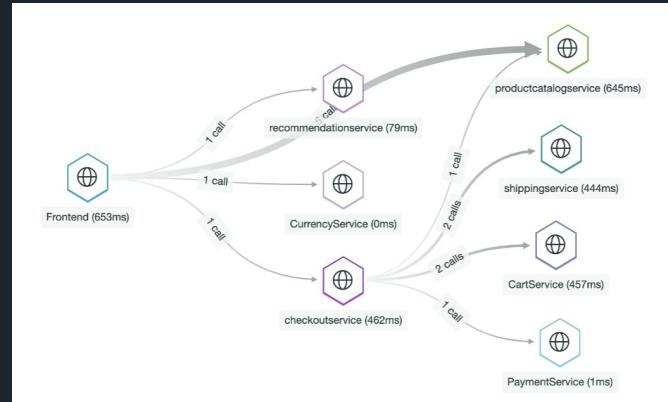
RabbitMQ / (Cloud)AMQP



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>

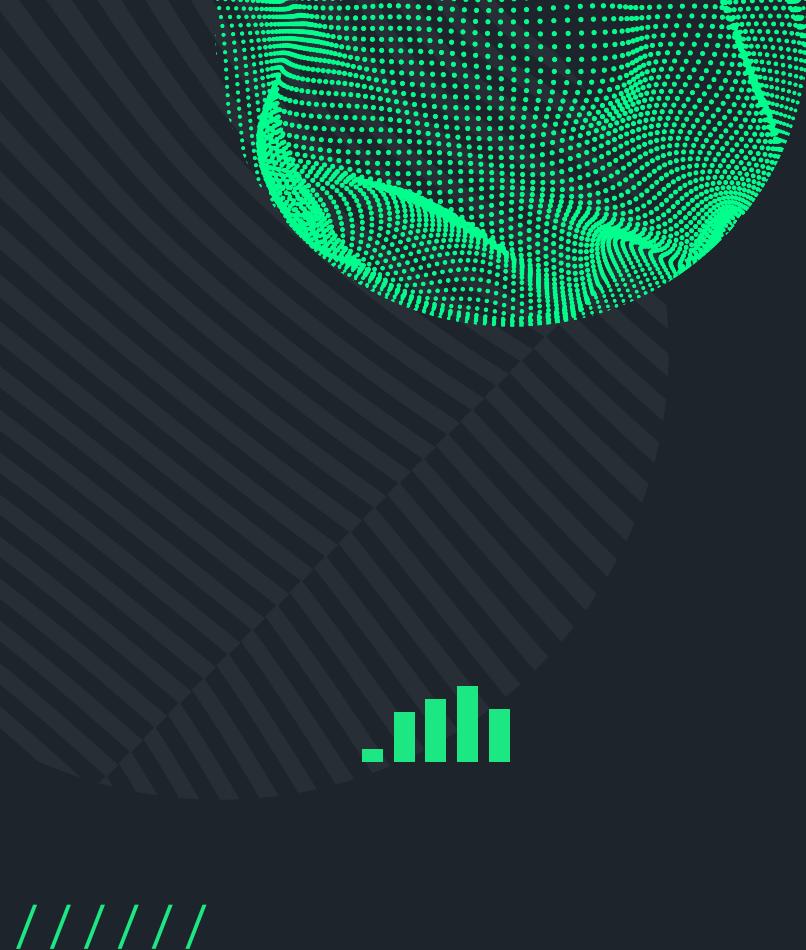
Reading List

- [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](#)



Harry Kimpel
Principal Developer Relations Engineer
New Relic
@harrykimpel

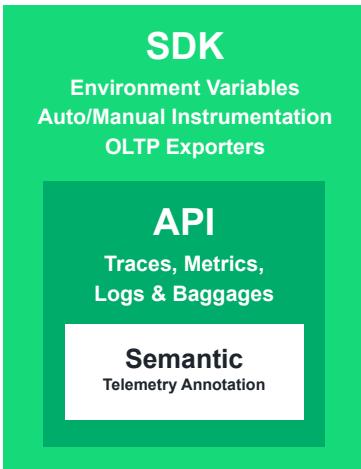




Appendix

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 shows the New Relic distributed tracing interface. A red box highlights the 'Logs (2)' tab in the top navigation bar. Another red box highlights a callout for an 'anomalous span' in the trace map, which points to a detailed view of the span's logs.

Anomalous Span Detection in Distributed Tracing for OpenTelemetry

Logs in Context (correlate to the right trace)

Trace Details Logs (2)

1 anomalous span ▾ 1 error ▾ 8 entities ▾ Find spans by name or ID Show trace map

Entity: checkoutservice
Span: hipstershop.ShippingService/GetQuote
is about 286.46 ms slower than average (193% slow...
Compared to similar spans over the past 6 hours

Trace duration 459.74 ms Back-end duration 459.74 ms

View span events ⓘ

hipstershop.ShippingService/GetQuote
checkoutservice

Performance Attributes Details 435.18 ms DURATION

Throughput (rpm)

8:30pm 8:40pm 8:50pm

Trace Details Logs (2)

Query logs in the selected entity

2 Logs

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

Open in logs

© 2025 New Relic, Inc. All rights reserved.

new relic

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 single trace with a duration of 458.43 ms. It highlights spans for the `hipstershop.ShippingService/GetQuote` service. One span, `CreateQuoteFromCount`, has a duration of 434.58 ms. Another span, `CreateQuoteFromFloat`, has a duration of 334.18 ms. A red box highlights this section.

Top Right Entity Preview: A modal for the `hipstershop.ShippingService/GetQuote` entity shows a timeline with a green line representing "Golden signals". The average duration is 434.72 ms. A red box highlights this section.

Middle Left Trace Map: Shows a trace map for a trace with a duration of 458.88 ms. It highlights spans for the `hipstershop.ShippingService/ShipOrder` service. One span, `shipOrder`, has a duration of 0.07 ms. A red box highlights this section.

Middle Right Entity Preview: A modal for the `shipOrder` entity shows a timeline with a red line representing an error. The average duration is 0.07 ms. A red box highlights this section.

Bottom Left Callout: Text: "Get deeper into the app stack, by building spans."

Bottom Right Callout: Text: "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/1YKWWN1N4O
[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

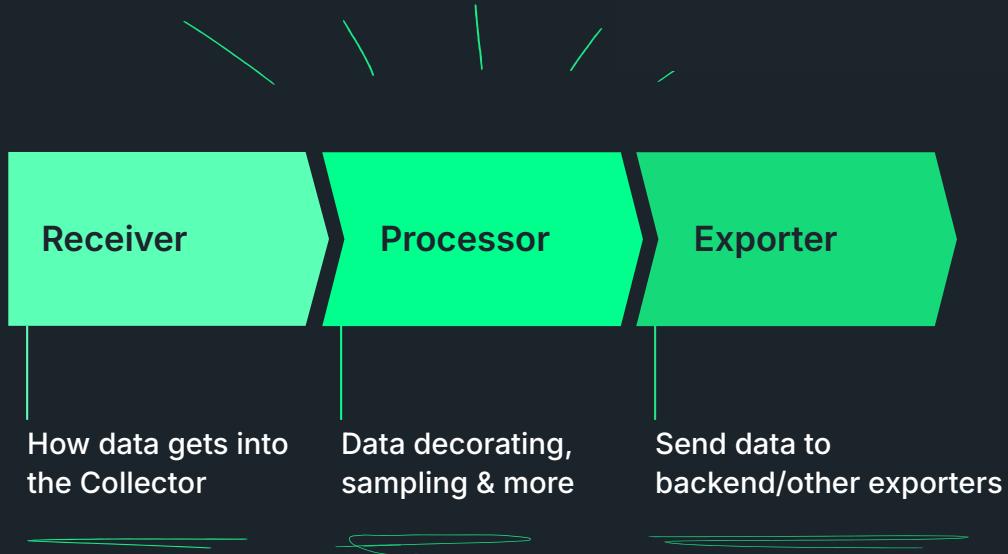
Important - deploy might take up to 10 mins

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

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

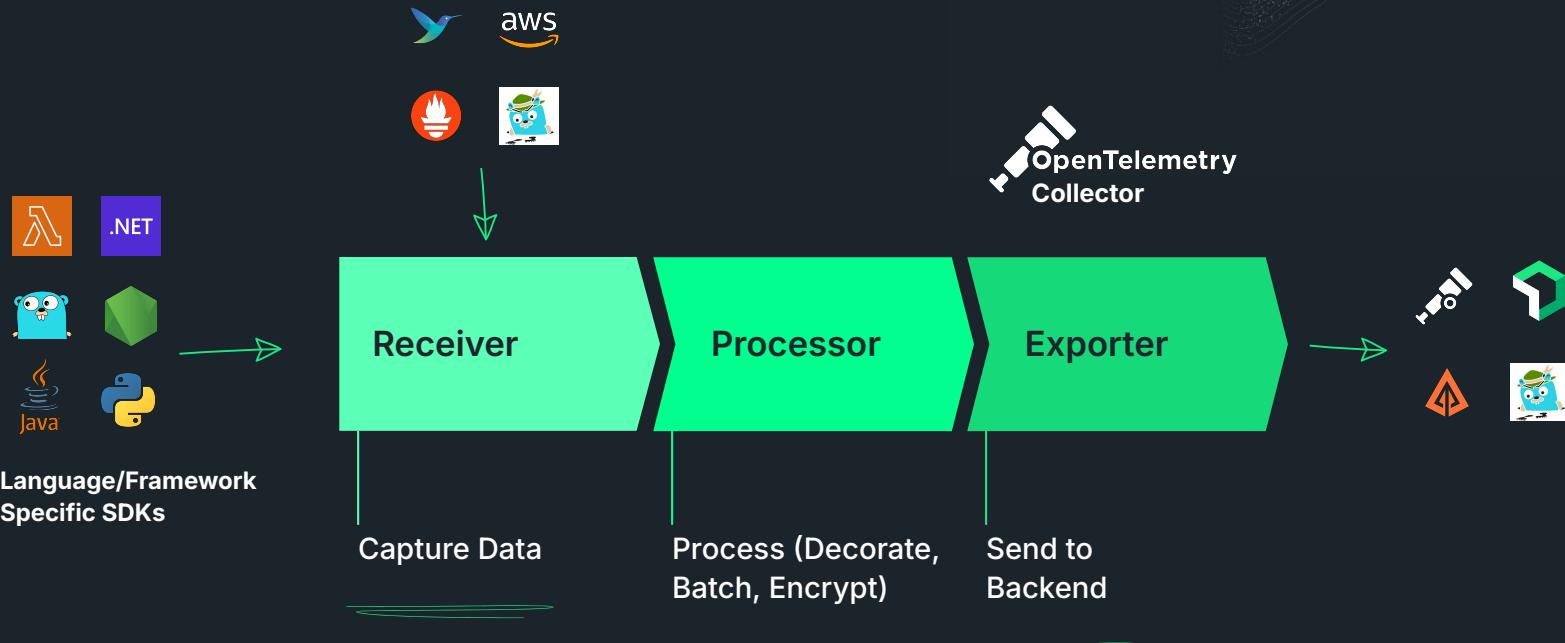
Collector Component

OpenTelemetry

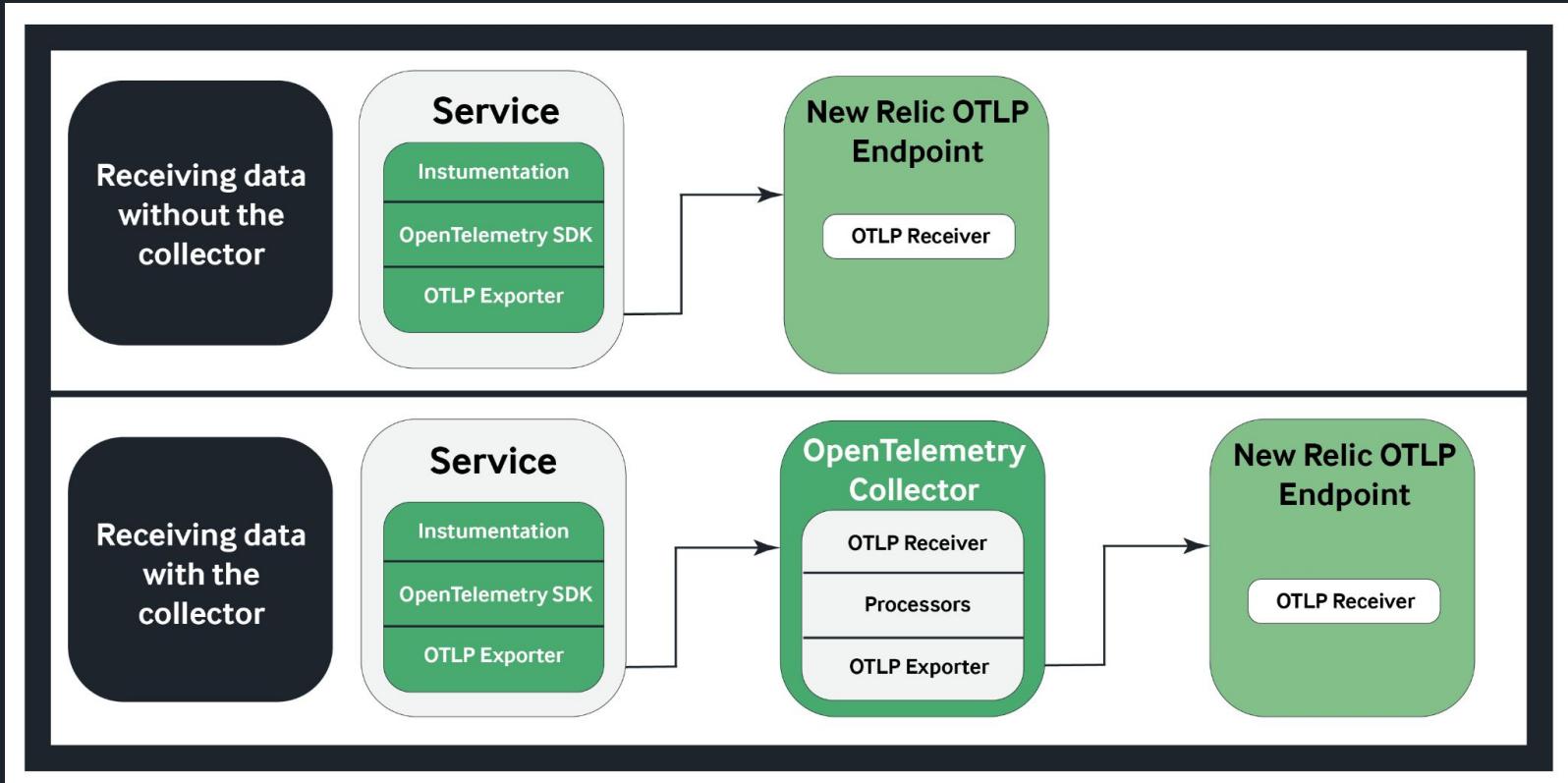


Putting it All Together

OpenTelemetry



OpenTelemetry - Collector



OpenTelemetry - Collector

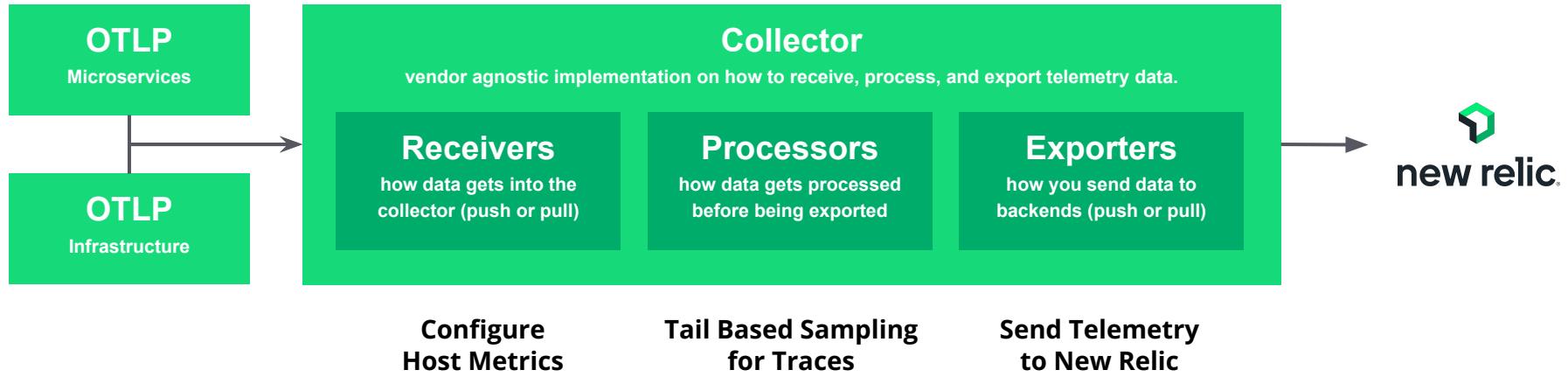
Benefits

- 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 image shows a terminal window on the left and a New Relic Data Explorer interface on the right.

Terminal (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  
30   processors:  
31     batch:  
32   cumulativeedelta:  
33     metrics:  
34       - system.network.io  
35       - system.disk.operations  
36       - system.network.dropped  
37       - system.network.packets  
38       - process.cpu.time
```

A red arrow points from the terminal to the configuration file, and a green arrow points from the configuration file to the Data Explorer interface.

Data Explorer (Right):

The Data Explorer interface shows a query for the latest system memory utilization:

```
NRQL SELECT latest('system.memory.utilization') FROM Metric SINCE 10 MINUTES AGO TIMESERIES
```

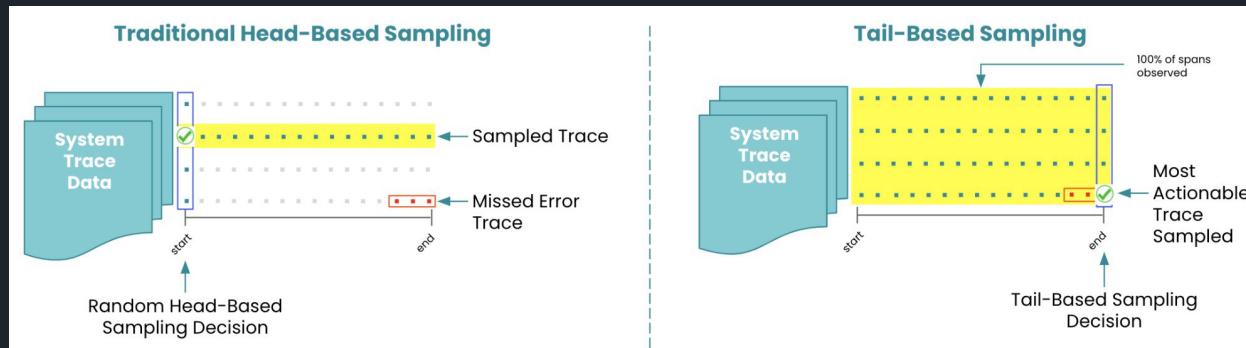
The chart displays the system.memory.utilization metric over time, showing values ranging from approximately 0.12 to 0.15.

Metric	Value
system.memory.utilization	0.15
system.disk.operations	Sum
system.disk.pending_operations	Latest
system.disk.weighted_io_time	Latest
system.filesystem.inodes.usage	Latest
system.filesystem.usage	Latest
system.filesystem.utilization	Latest
system.memory.usage	Latest
system.memory.utilization	Latest
system.network.connections	Latest
system.network.dropped	Sum
system.network.errors	Latest
system.network.io	Sum
system.network.packets	Sum
system.paging.faults	Latest
system.paging.operations	Latest
system.processes.count	Latest
system.processes.created	Latest

Collect additional infrastructure metrics
for OpenTelemetry

HEAD vs TAIL Sampling

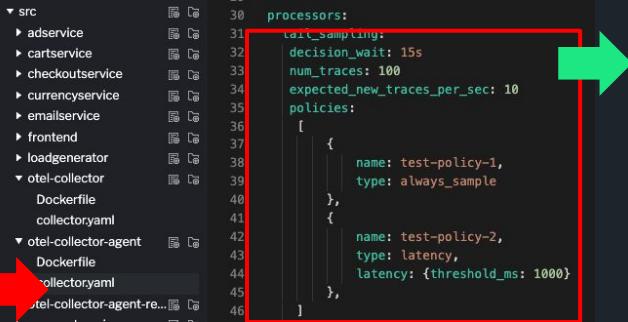
Storing all tracing data is **costly**. Most of the time, you
ONLY NEED THE RIGHT SAMPLING OF DATA



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.