
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

new relic

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?

new relic

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

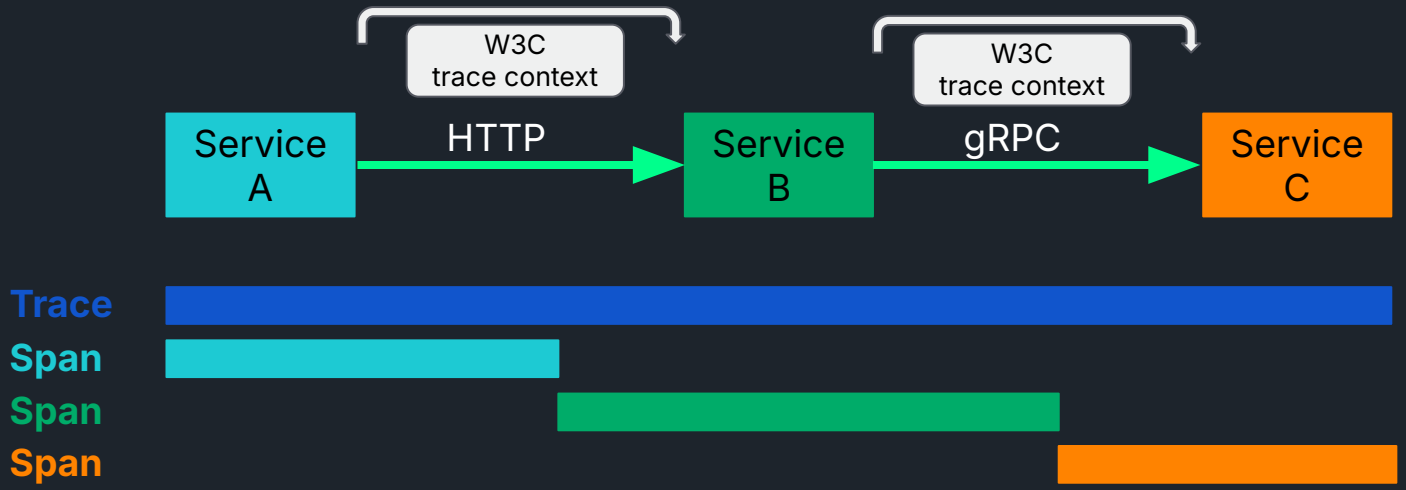
Logs

A recording of an event.

timestamp	message
08:20:30.519	Startup check warning: Agent version out of date: v12.7.0, in order access up to date feature…
08:21:16.827	Feb 28 08:21:16 ip-172-31-25-72 dhclient[420]: XMT: Solicit on eth0, interval 116330ms.
08:21:45.143	Feb 28 08:21:45 ip-172-16-1-8 dhclient[437]: XMT: Solicit on eth0, interval 126620ms.
08:23:13.216	Feb 28 08:23:13 ip-172-31-25-72 dhclient[420]: XMT: Solicit on eth0, interval 117750ms.
08:23:51.707	Feb 28 08:23:51 ip-172-16-1-8 dhclient[437]: XMT: Solicit on eth0, interval 128740ms.
08:25:01.143	Feb 28 08:25:01 ip-172-16-1-149 CRON[264368]: pam_unix(cron:session): session opened for user…
08:25:01.148	Feb 28 08:25:01 ip-172-16-1-149 CRON[264369]: (root) CMD (command -v debian-sa1 > /dev/null &…
08:25:01.150	Feb 28 08:25:01 ip-172-16-1-149 CRON[264368]: pam_unix(cron:session): session closed for user…
08:25:01.990	Feb 28 08:25:01 ip-172-16-1-139 CRON[289985]: pam_unix(cron:session): session opened for user…
08:25:01.997	Feb 28 08:25:01 ip-172-16-1-139 CRON[289986]: (root) CMD (command -v debian-sa1 > /dev/null &…
08:25:01.998	Feb 28 08:25:01 ip-172-16-1-139 CRON[289985]: pam_unix(cron:session): session closed for user…
08:25:11.019	Feb 28 08:25:11 ip-172-31-25-72 dhclient[420]: XMT: Solicit on eth0, interval 113600ms.
08:26:00.637	Feb 28 08:26:00 ip-172-16-1-8 dhclient[437]: XMT: Solicit on eth0, interval 131920ms.
08:26:24.154	Feb 28 08:26:24 ip-172-16-1-8 dhclient[346]: DHCPREQUEST for 172.16.1.8 on eth0 to 172.16.1.1…
08:26:24.154	Feb 28 08:26:24 ip-172-16-1-8 dhclient[346]: DHCPACK of 172.16.1.8 from 172.16.1.1
08:26:24.169	Feb 28 08:26:24 ip-172-16-1-8 dhclient[346]: bound to 172.16.1.8 -- renewal in 1363 seconds.
08:26:37.591	Feb 28 08:26:37 ip-172-16-1-139 sshd[290216]: Invalid user from 54.36.8.139 port 50368
08:26:45.248	Feb 28 08:26:45 ip-172-16-1-139 sshd[290216]: Connection closed by invalid user 54.36.8.139 …
08:27:04.715	Feb 28 08:27:04 ip-172-31-25-72 dhclient[420]: XMT: Solicit on eth0, interval 112740ms.
08:27:31.172	Feb 28 08:27:31 ip-172-16-1-139 amazon-ssm-agent.amazon-ssm-agent[376]: 2025-02-28 08:27:31.1…

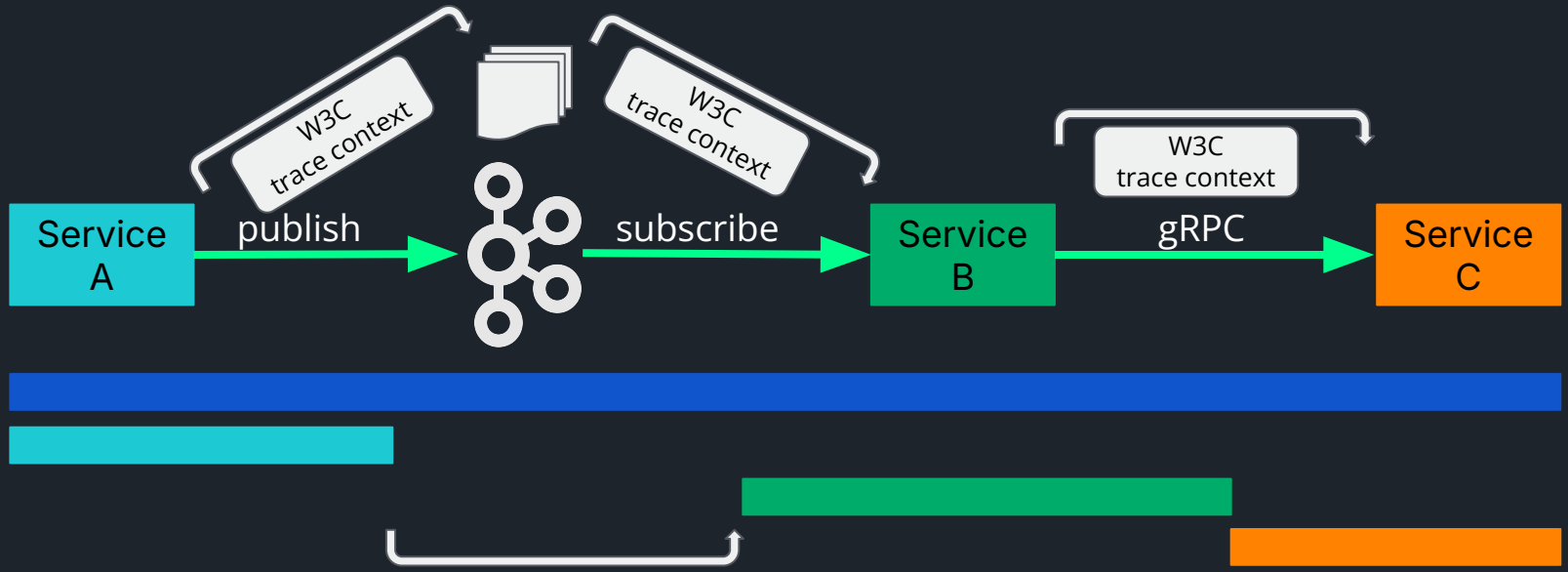
new relic

What is Tracing?



new relic

How is Tracing for event-based systems different?



new relic

Who uses Kafka?

new relic

new relic® uses Kafka!

more than **15** million messages per second

aggregate data rate approaching **1** Tbps

new relic

✋ Hands up if you have heard about OpenTelemetry

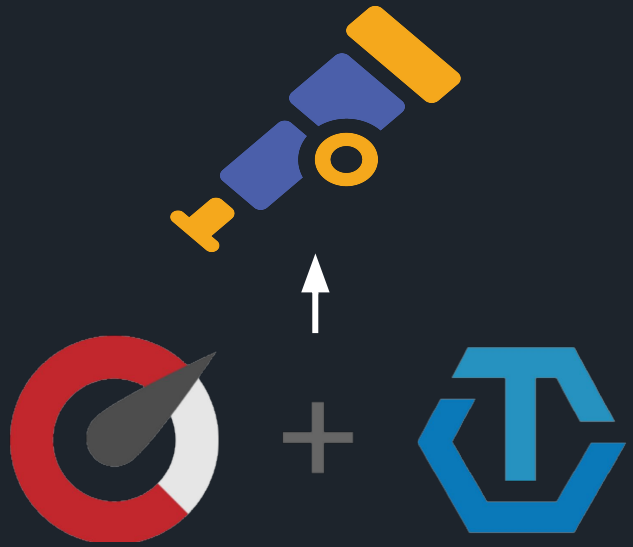
new relic

✋ Keep your hands up if you use OpenTelemetry

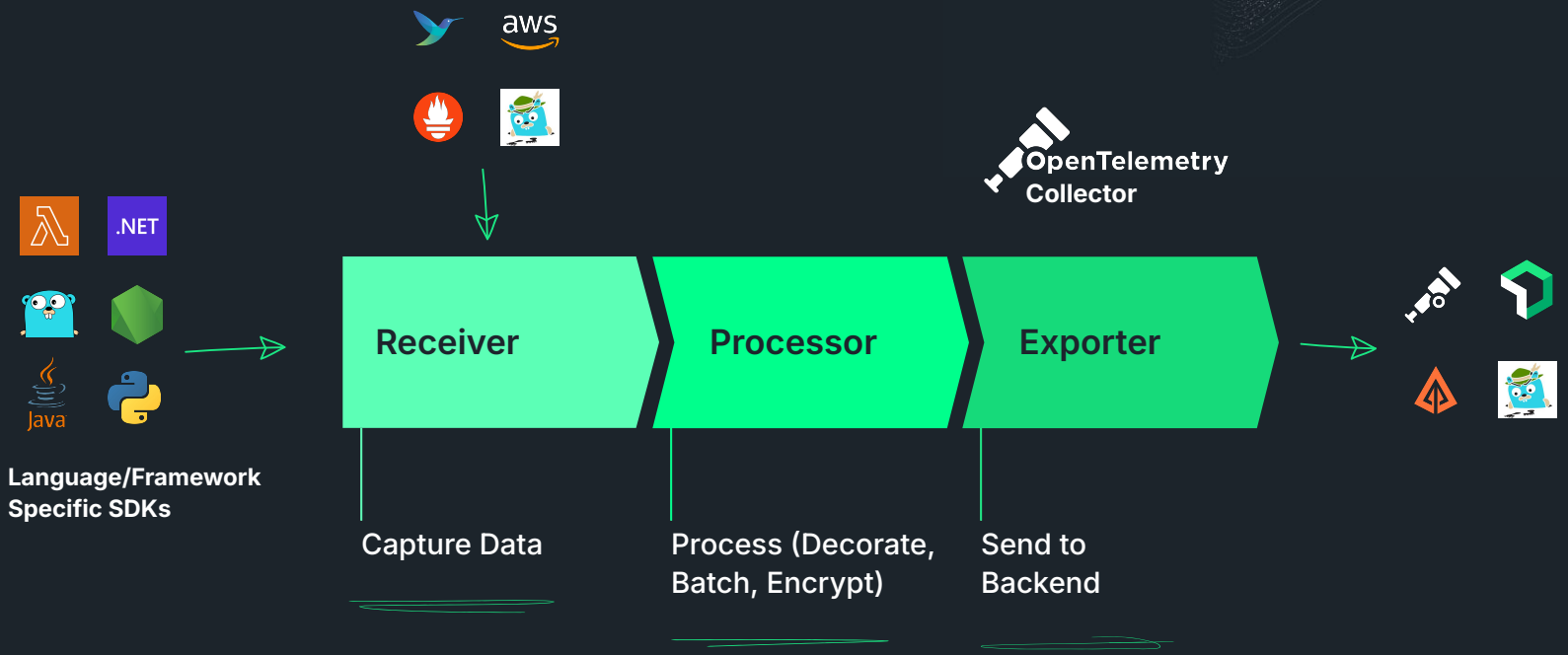
new relic

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

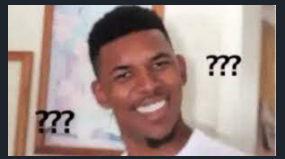


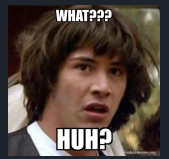
new relic

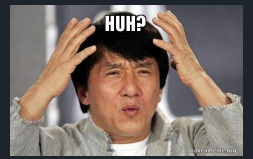
The Rise of OpenTelemetry

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

BUT WAIT ...







new relic

Design Considerations & Guidelines

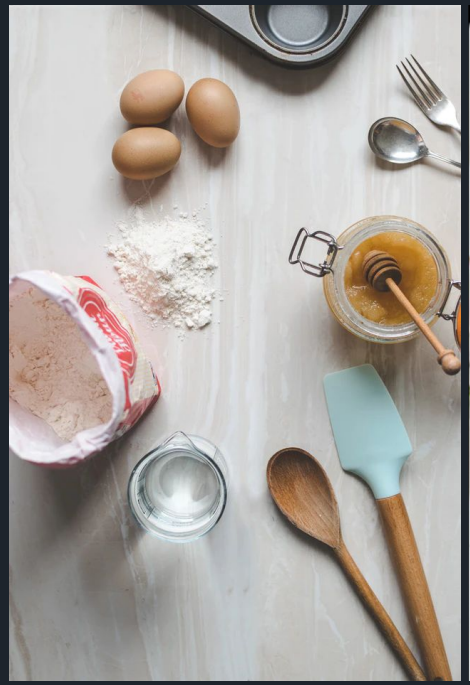
Customize Everything
(manual instrumentation)

Ingredients

Custom Made

Flavours

Java, .NET and others mature



Zero-Code
(auto instrumentation)

Ready to Eat

Packaged

Scalable

Most Languages

new relic

Screenshot from
https://opentelemetry.io/docs/instrumentation/
taken
Ja[illegible]25

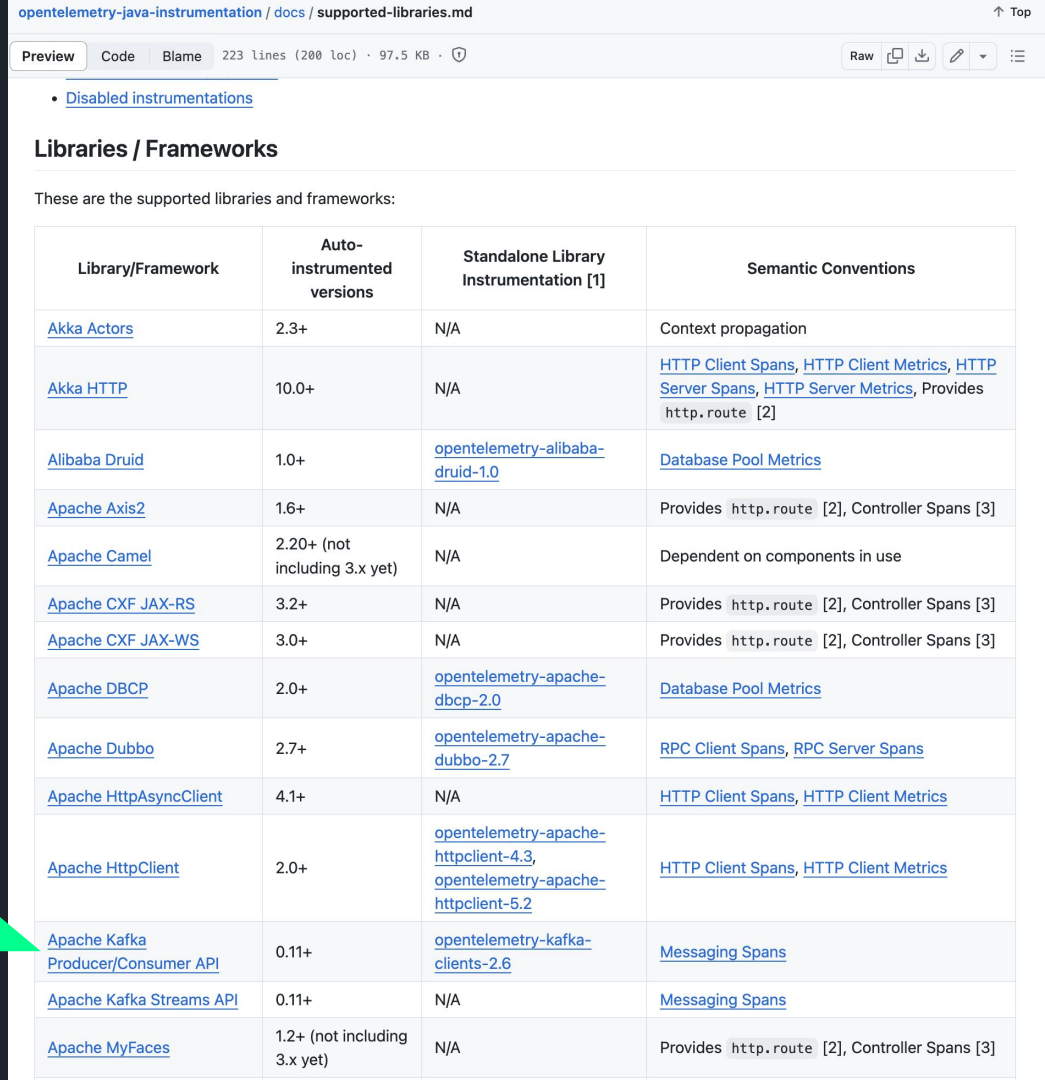
Status and Releases

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 ↑ Top

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

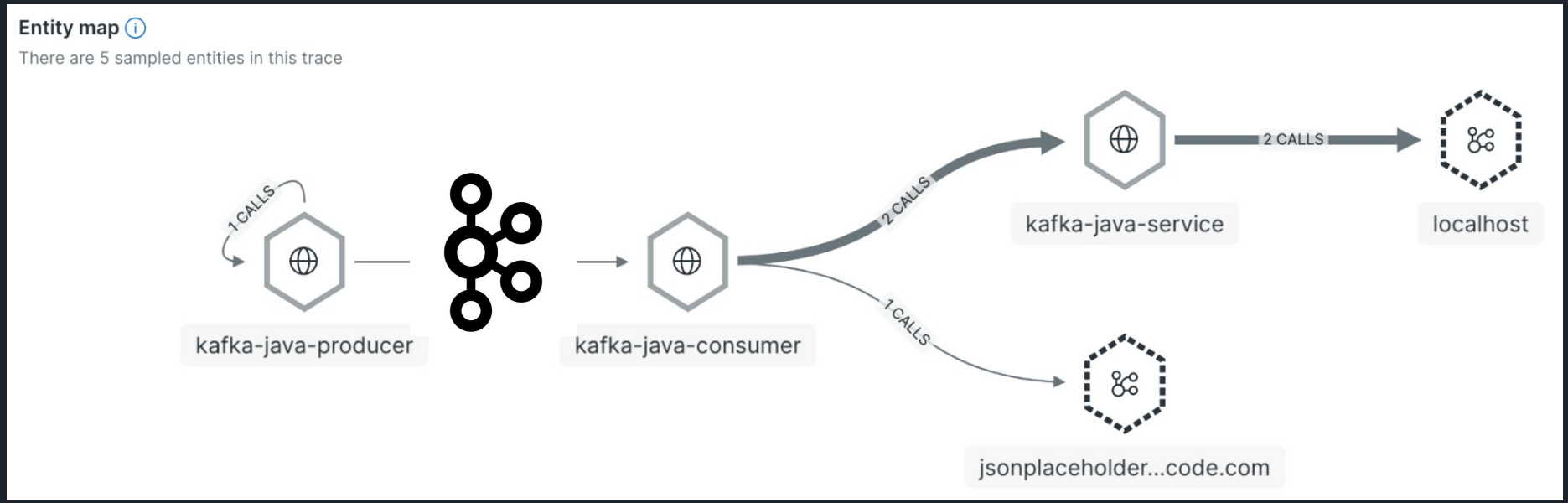
- Disabled instrumentations

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 `http.route` [2]
Alibaba Druid	1.0+	opentelemetry-alibaba-druid-1.0	Database Pool Metrics
Apache Axis2	1.6+	N/A	Provides `http.route` [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 `http.route` [2], Controller Spans [3]
Apache CXF JAX-WS	3.0+	N/A	Provides `http.route` [2], Controller Spans [3]
Apache DBCP	2.0+	opentelemetry-apache-dbcp-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 `http.route` [2], Controller Spans [3]

Automatic instrumentation with Java



new relic

Automatic instrumentation - aka zero-code instrumentation - with Java

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

new relic

Automatic instrumentation with Java

otel-auto-instr > demoConsumer > $ run.sh

```
export JAVA_TOOL_OPTIONS="-javaagent:/Users/hkimpel/projects/kafka/java-local-test/otel-auto-instr/opentelemetry-javaagent.jar"
export OTEL_TRACES_EXPORTER=otlp
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
#·US·region
export OTEL_EXPORTER_OTLP_ENDPOINT='https://otlp.nr-data.net'
#·EU·region
#export·OTEL_EXPORTER_OTLP_ENDPOINT='https://otlp.eu01.nr-data.net'
export OTEL_EXPORTER_OTLP_HEADERS="api-key=NEW_RELIC_LICENSE_KEY"
export OTEL_SERVICE_NAME="kafka-java-consumer"
export OTEL_SERVICE_VERSION="0.1.0"

./mvnw spring-boot:run
```


new relic

POST /orders

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

Trace details | Logs (14)

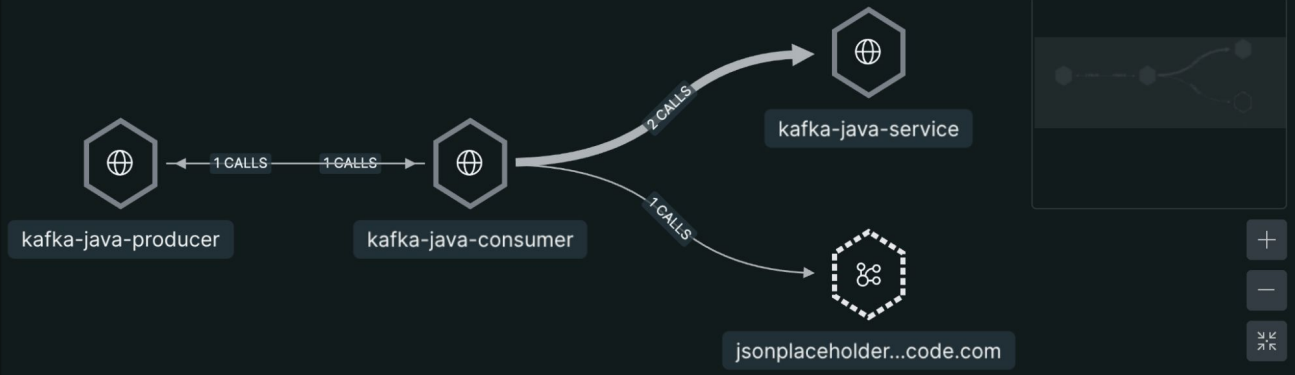
0 anomalous spans | 0 errors | 4 entities | Find spans by name or ID

Trace duration 4402.35 ms | Backend duration 4402.35 ms

Entity map

There are 4 sampled entities in this trace

Map | Timeline | Latency

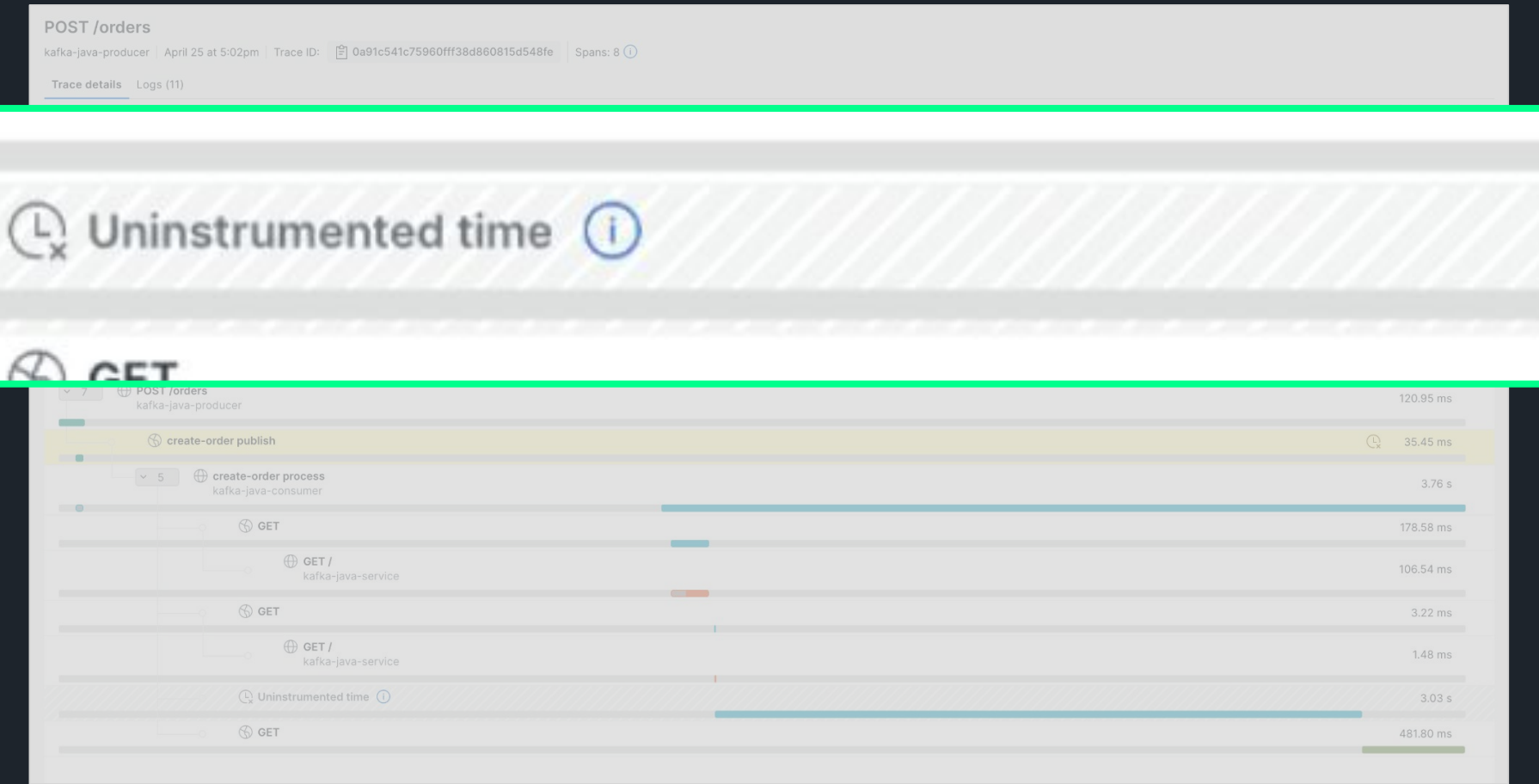


Expand all | Collapse all | Standard | Manual | Reset | Maximize

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

new relic

Automatic instrumentation with Java



Manual instrumentation with Java

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

new relic

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

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();
```

Manual instrumentation with Java

```
OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
    .setTracerProvider(sdkTracerProvider)
    .setMeterProvider(sdkMeterProvider)
    .setLoggerProvider(sdkLoggerProvider)
    .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
    .buildAndRegisterGlobal();
```

Manual instrumentation with Java - Kafka producer

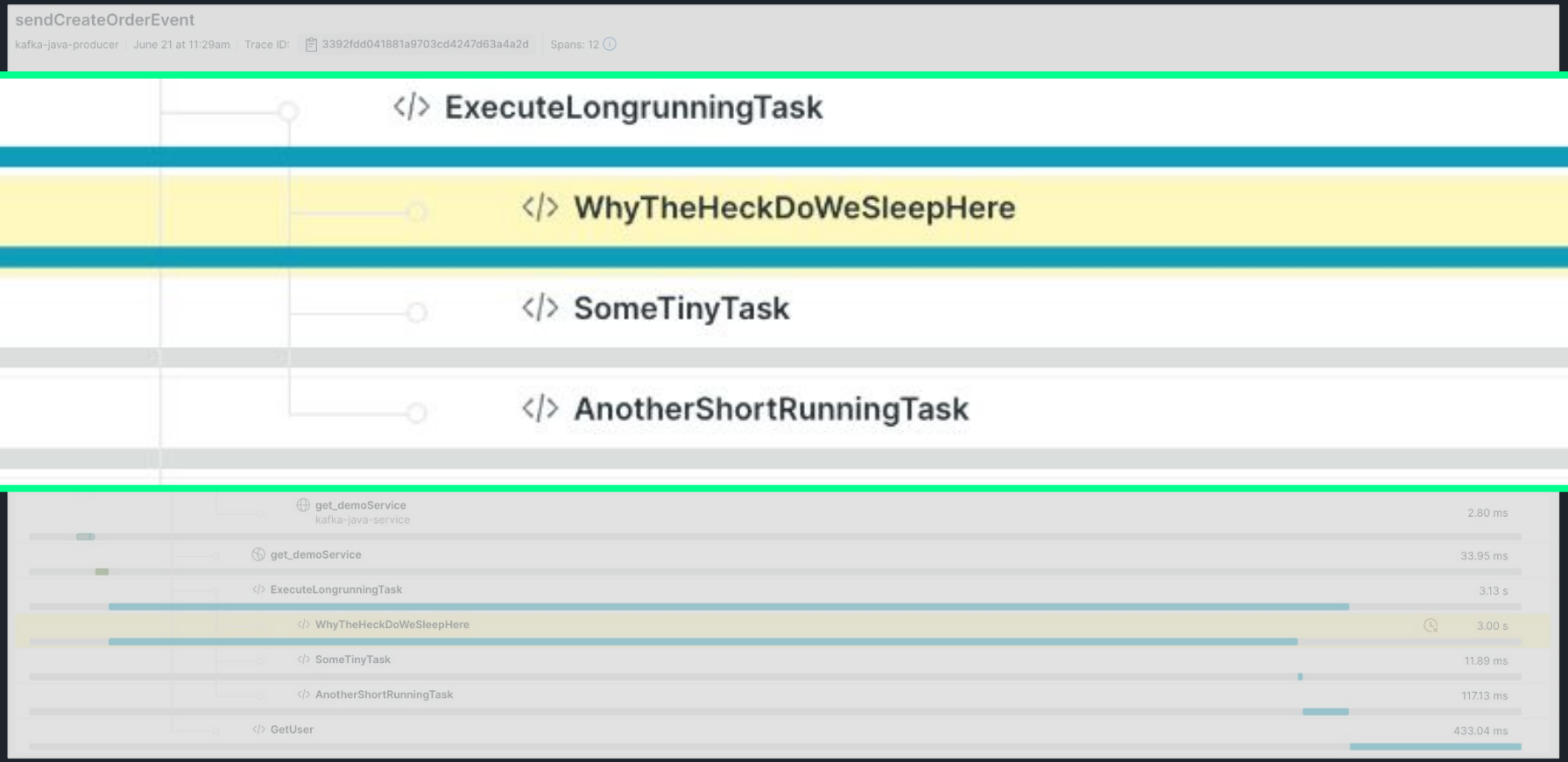
```
Pieces: Comment | Pieces: Explain
22      @Bean
23      public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
24          Map<String, Object> config = new HashMap<>();
25          config.put(
26                  org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,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


new relic

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<>();
          Pieces: Comment | Pieces: Explain
38        props.put(
39            ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
40            TracingConsumerInterceptor.class.getName());
          Pieces: Comment | Pieces: Explain
41        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
          Pieces: Comment | Pieces: Explain
42        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
          Pieces: Comment | Pieces: Explain
43        props.put(ConsumerConfig.CLIENT_ID_CONFIG, UUID.randomUUID().toString());
          Pieces: Comment | Pieces: Explain
44        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, value:StringSerializer.class);
45        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, value:JsonSerializer.class);
          Pieces: Comment | Pieces: Explain
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

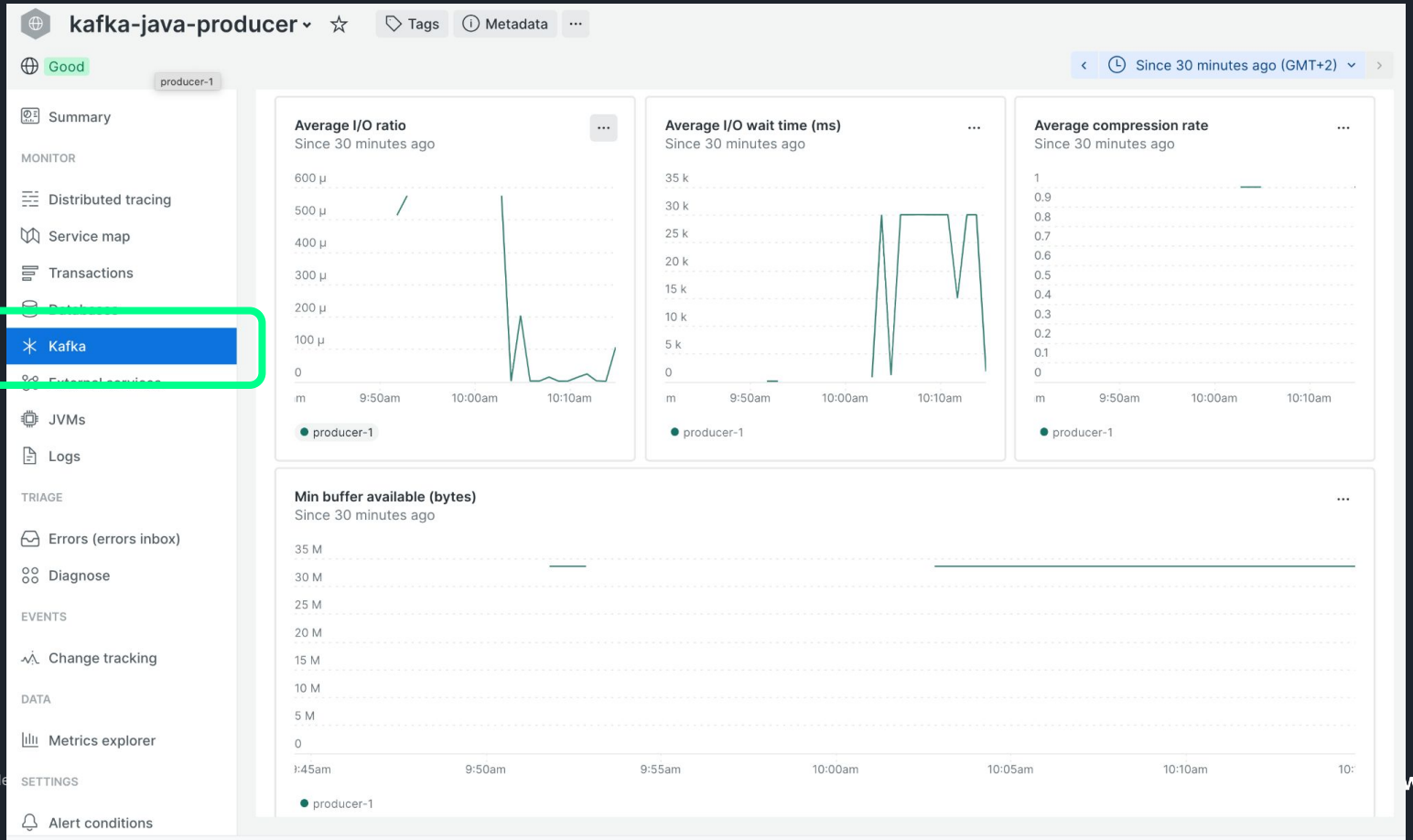


Manual instrumentation with Java - custom spans

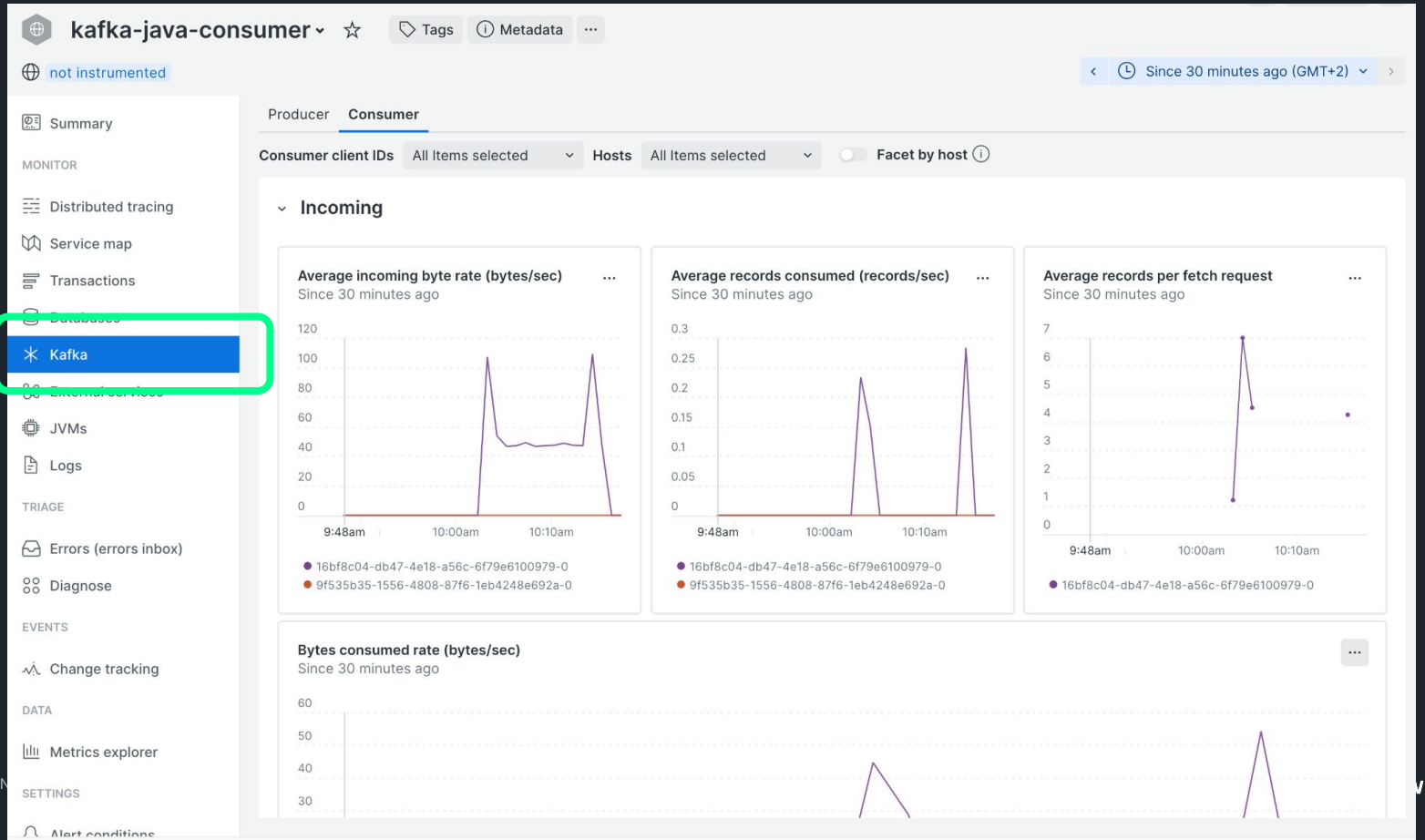
```java
private void ExecuteLongrunningTask(Integer secondsToSleep) {
    Span span = tracer.spanBuilder(spanName:"ExecuteLongrunningTask").startSpan();
    //Make the span the current span
    try {
        Pieces: Comment | Pieces: Explain
        Span sleepSpan = tracer.spanBuilder(spanName:"WhyTheHeckDoWeSleepHere")
                Pieces: Comment | Pieces: Explain
                .setParent(Context.current().with(span))
                .startSpan();
        try {
            Pieces: Comment | Pieces: Explain
            sleepSpan.setAttribute(key:"secondsToSleep", secondsToSleep);
            Pieces: Comment | Pieces: Explain
            Thread.sleep(secondsToSleep * 1000);
            Pieces: Comment | Pieces: Explain
            log.info("Executed some long running task that took " + secondsToSleep + " seconds to run.");

        } finally {
            Pieces: Comment | Pieces: Explain
            sleepSpan.end();
        }

        Pieces: Comment | Pieces: Explain
        SomeTinyTask(span);

        Pieces: Comment | Pieces: Explain
        AnotherShortRunningTask(span);
    } catch (Exception t) {
        span.recordException(t);
        //throw t;
    } finally {
        span.end();
    }
```


new relic

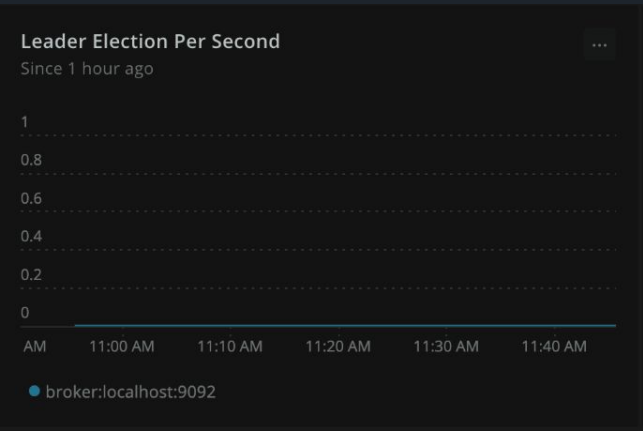
Kafka metrics for producer

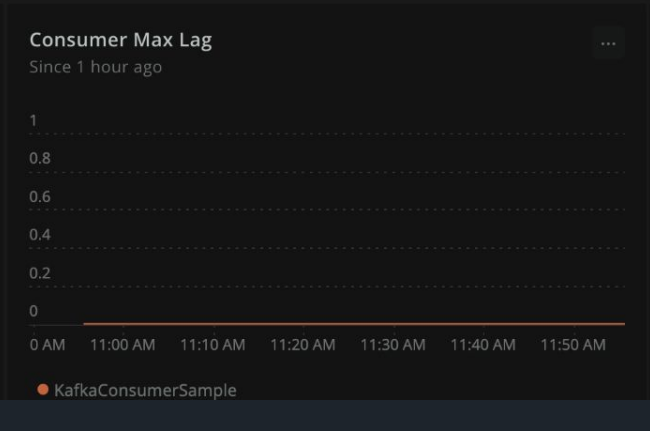


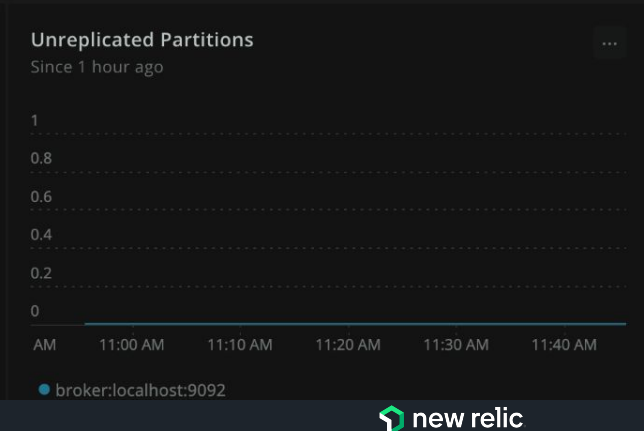
Kafka metrics for consumer



Kafka infrastructure metrics



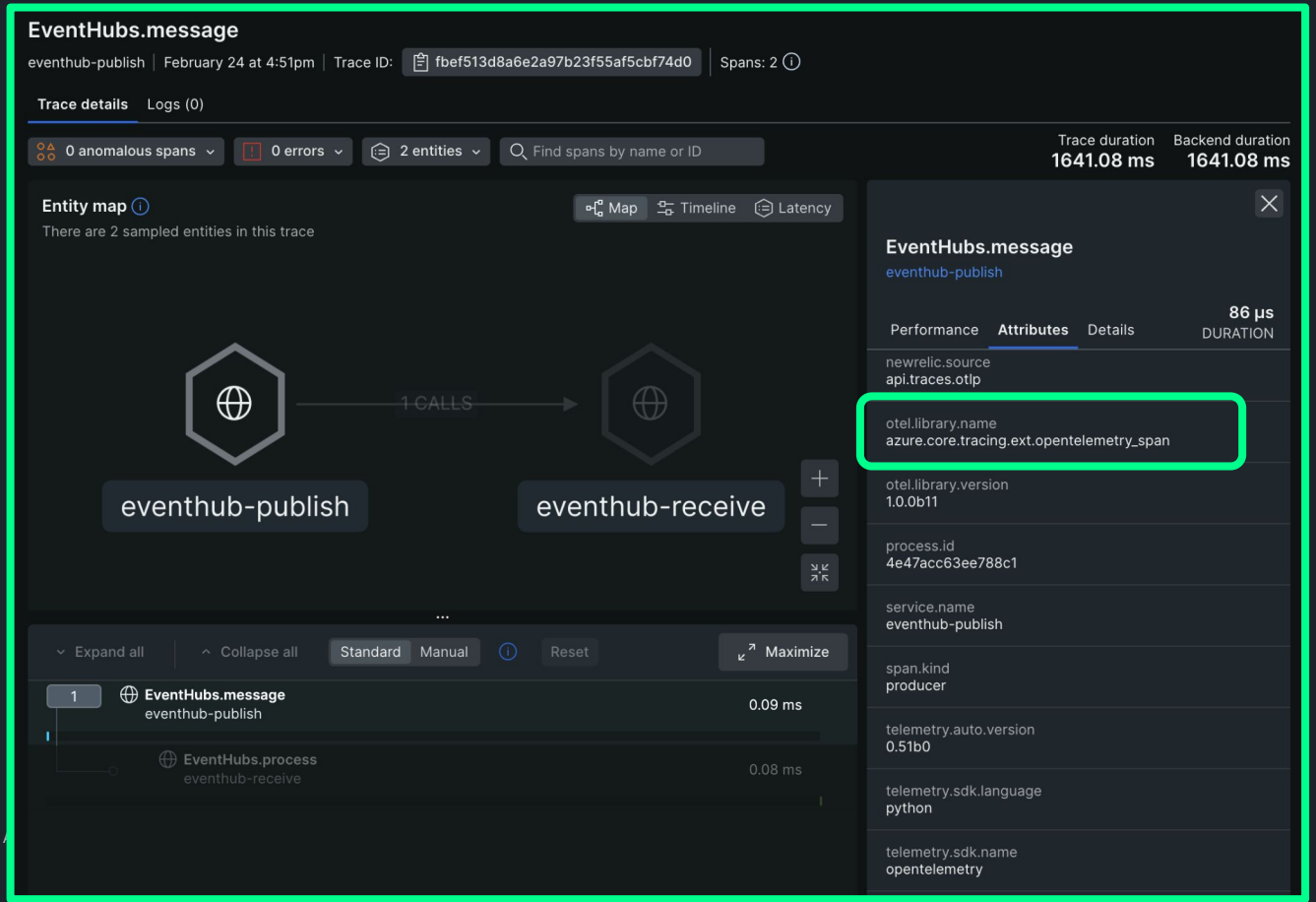




new relic

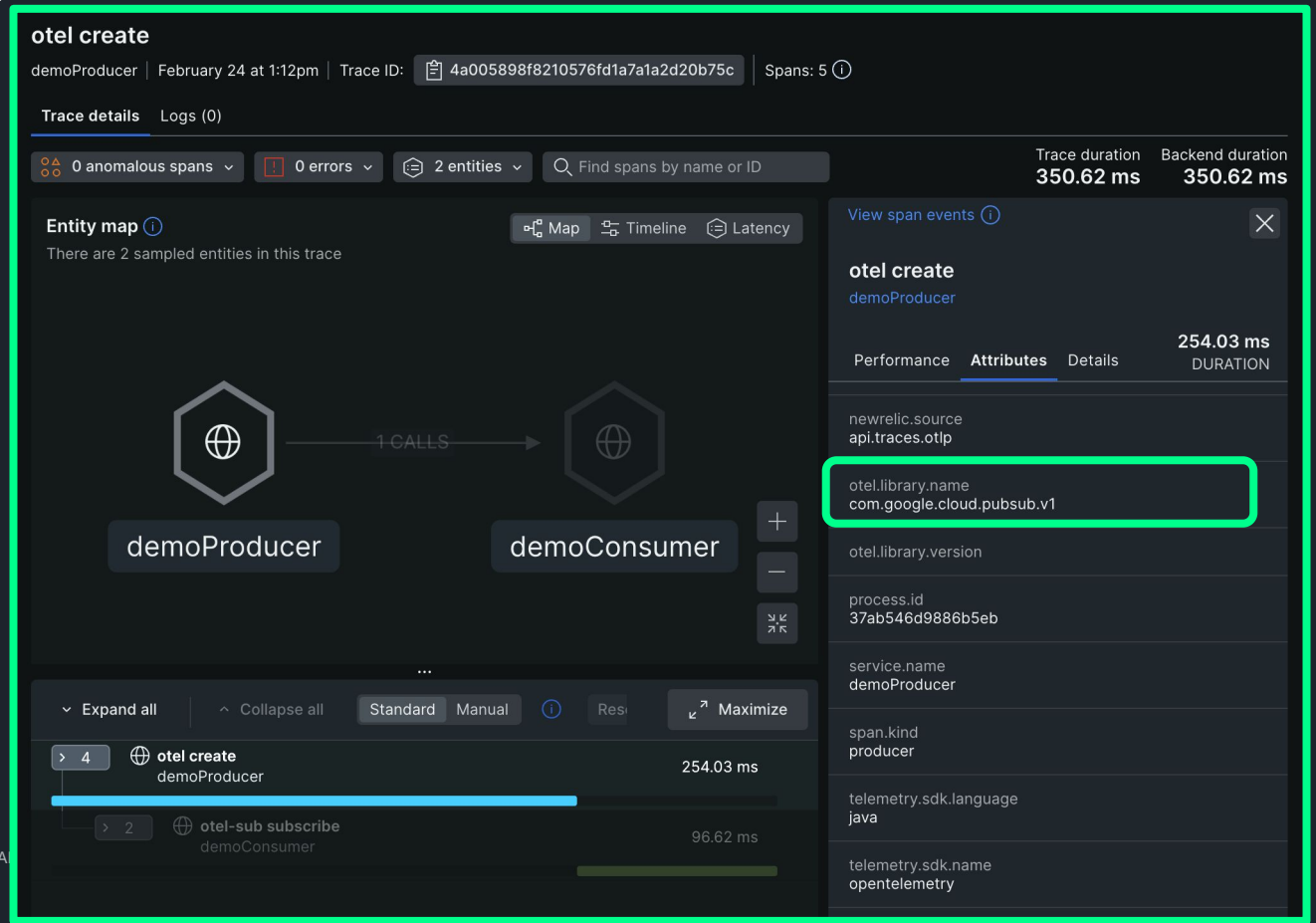
Other event-based systems?

Microsoft Azure Event Hub



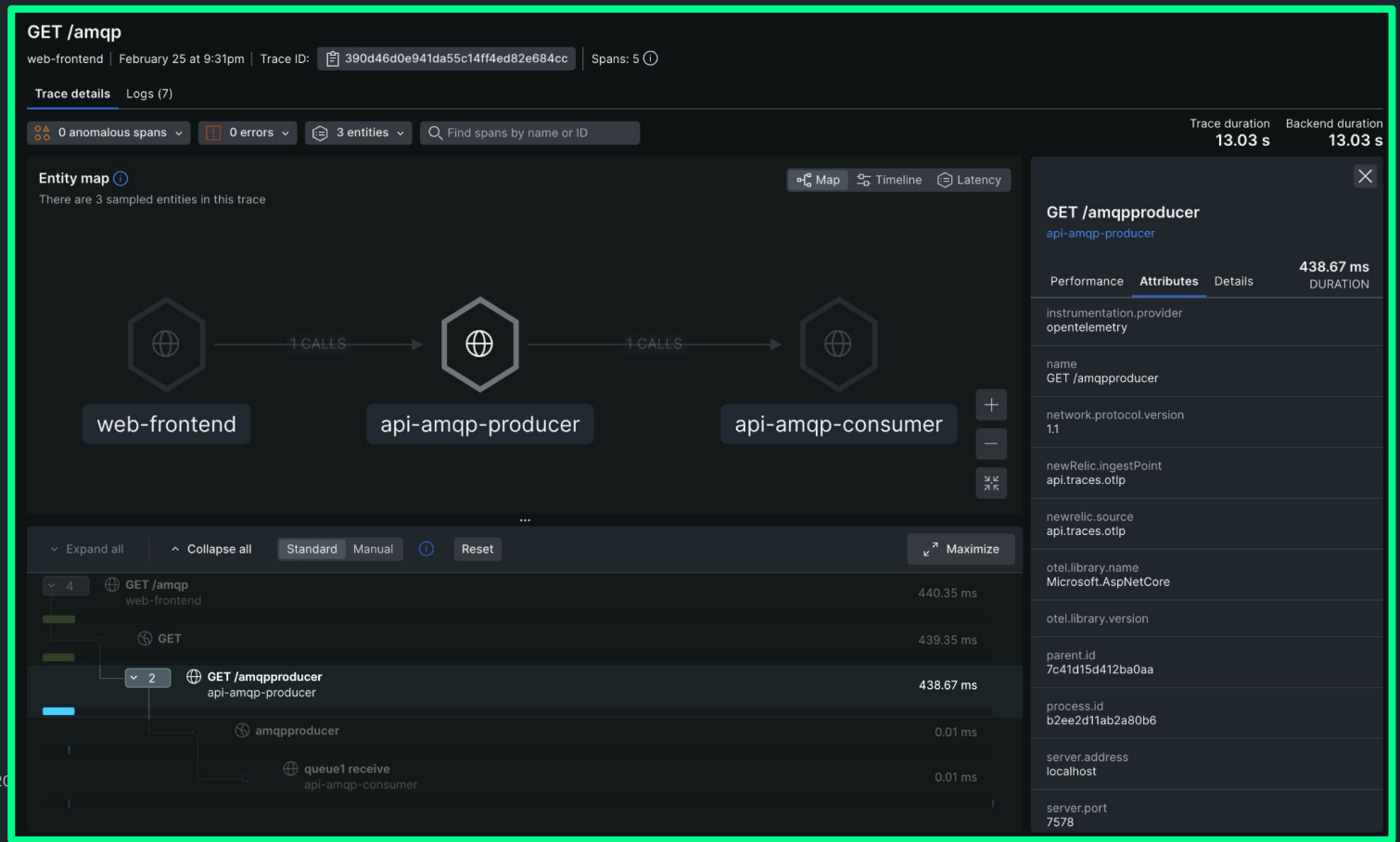
new relic

Google Pub/Sub



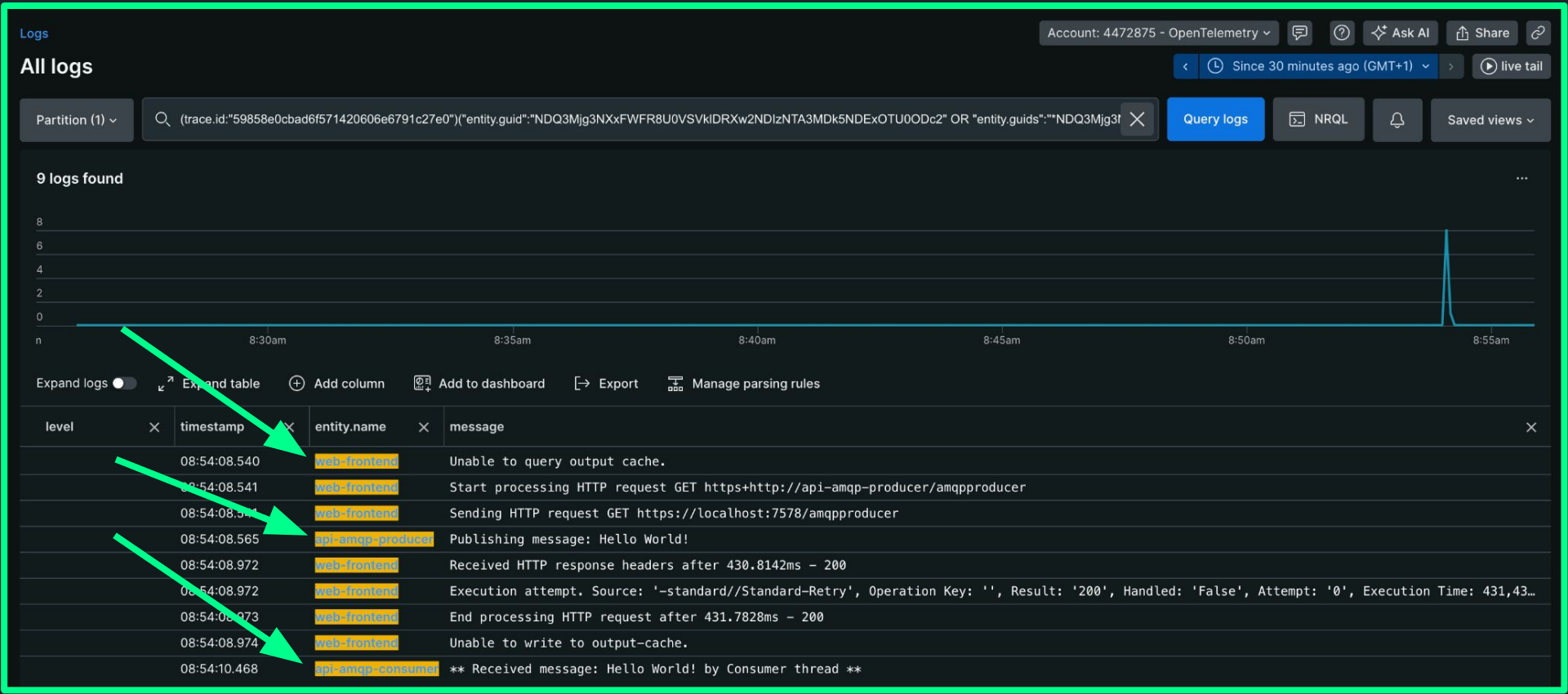
new relic

RabbitMQ / (Cloud)AMQP



new relic

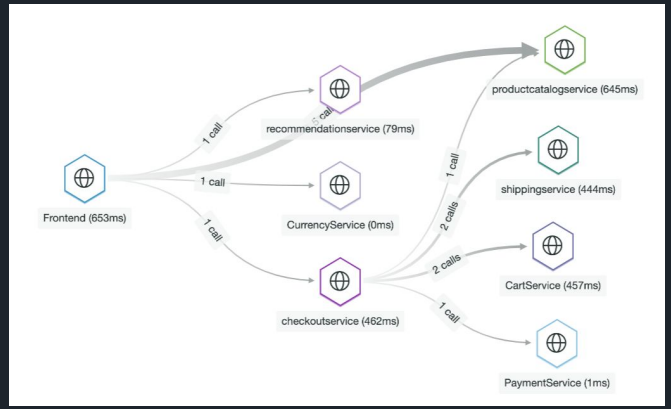
RabbitMQ / (Cloud)AMQP

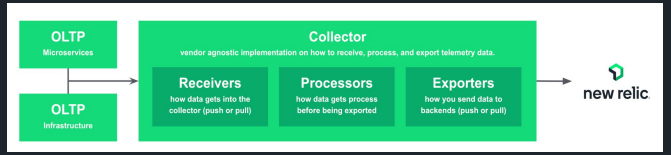


Summary

- 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





new relic

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

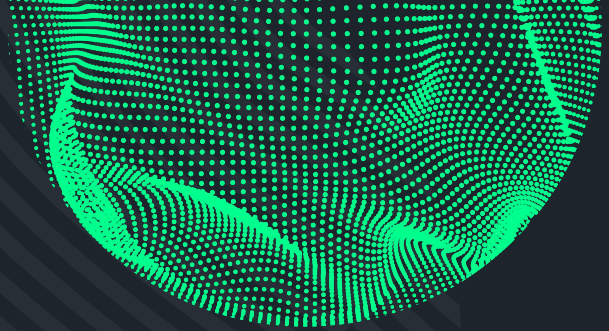
Harry Kimpel

Principal Developer Relations Engineer

New Relic

@harrykimpel

new relic

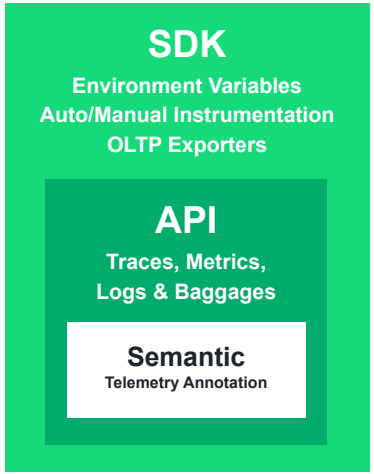


Appendix

some more information

new relic

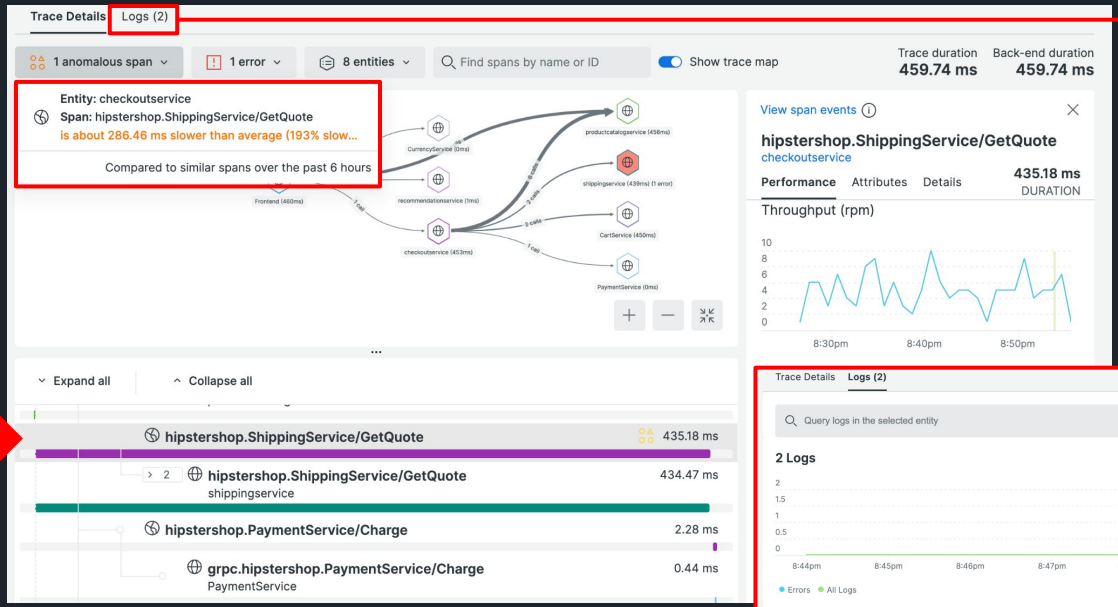
OpenTelemetry - Instrumentation

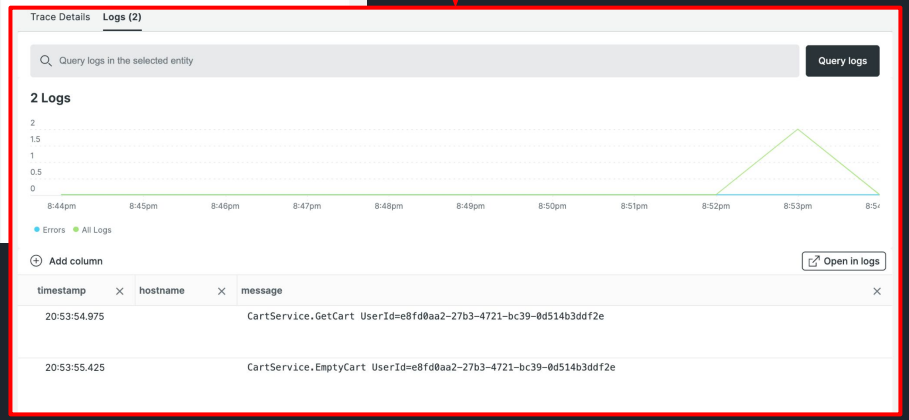


Core Concepts on Instrumentation

- **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



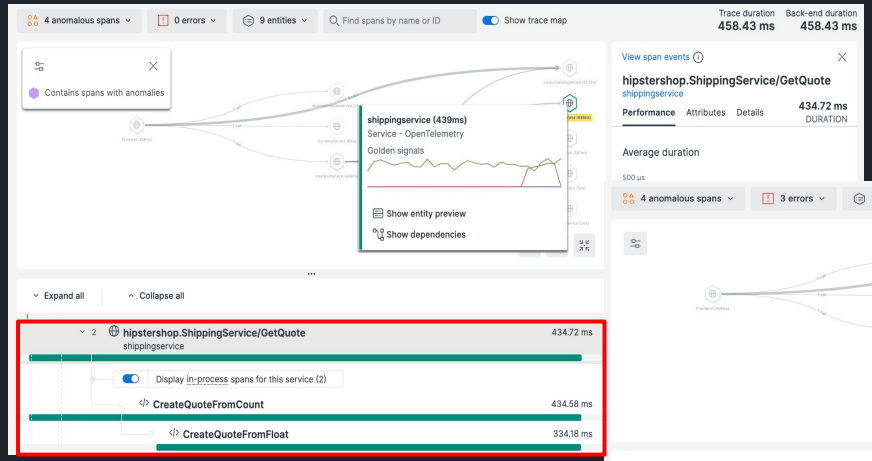


Logs in Context
(correlate to the right trace)

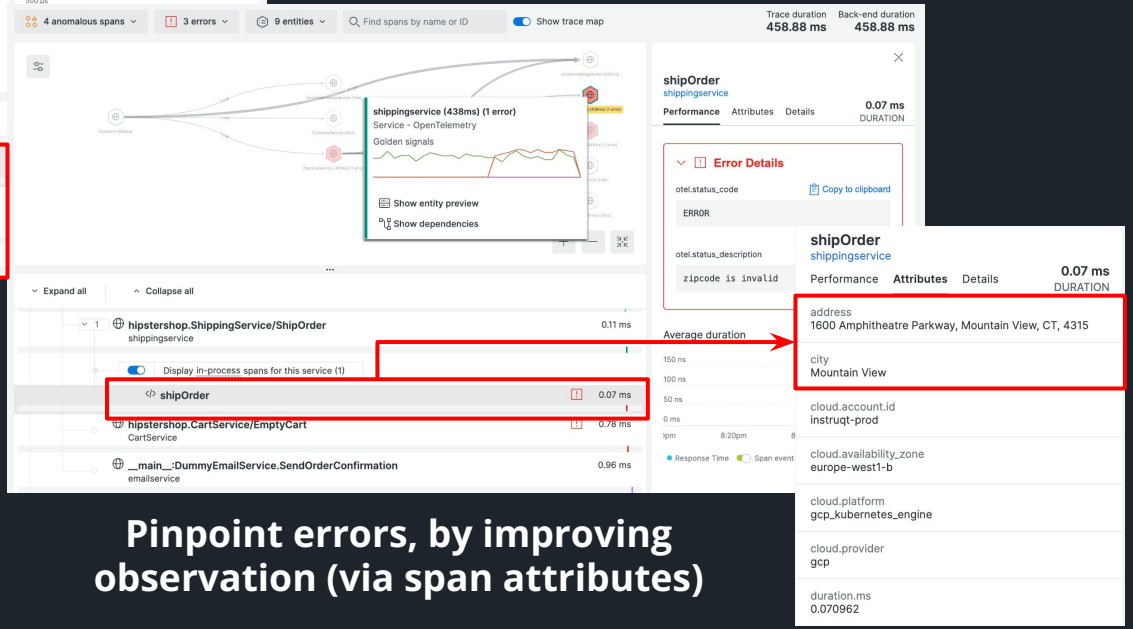
Anomalous Span Detection in Distributed Tracing for OpenTelemetry

new relic

Instrumentation in Action



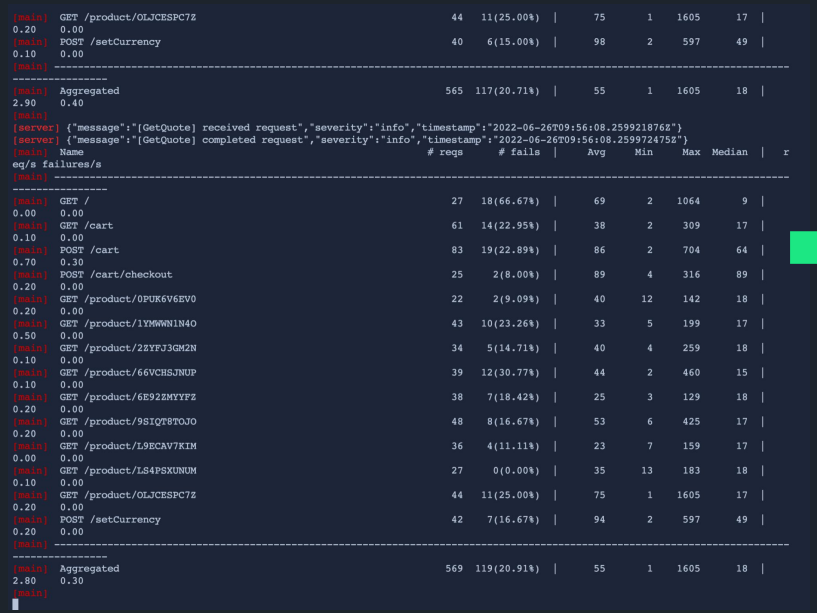
Get deeper into the app stack, by building spans.



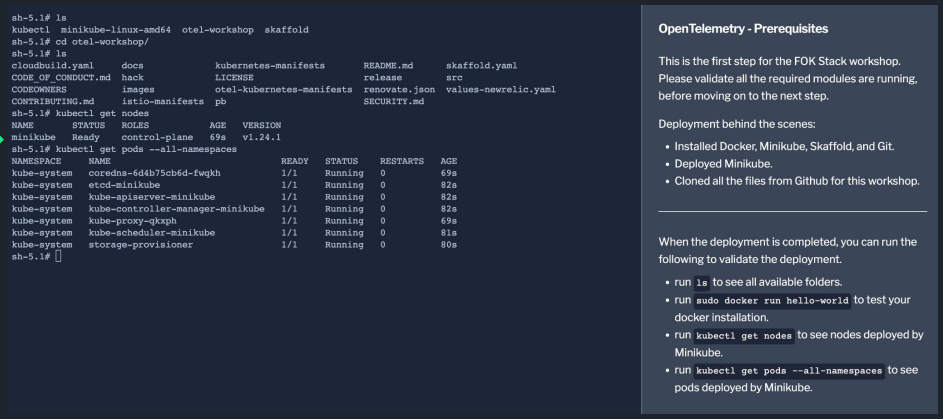
Pinpoint errors, by improving observation (via span attributes)

new relic

OpenTelemetry in Action



Deploy the app, and generate load (successful deployment)



Validate K8s is running, and follow the instructions

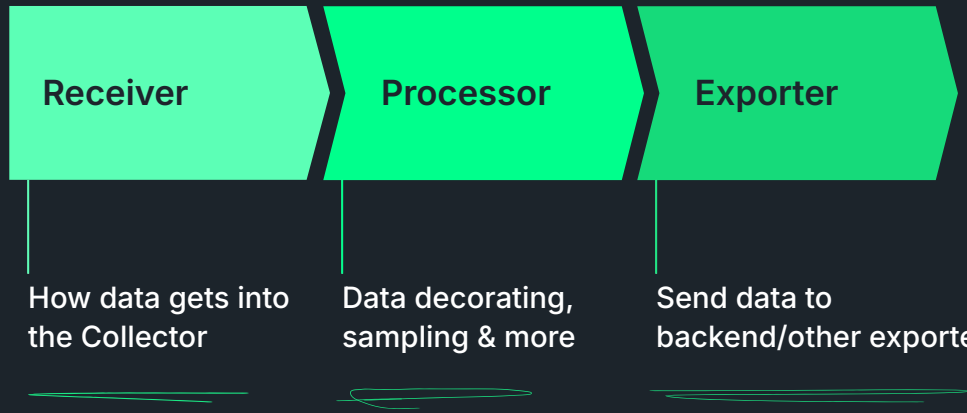
Important - deploy might take up to 10 mins

Need the env var? Go **HERE**.

new relic

Collector Component

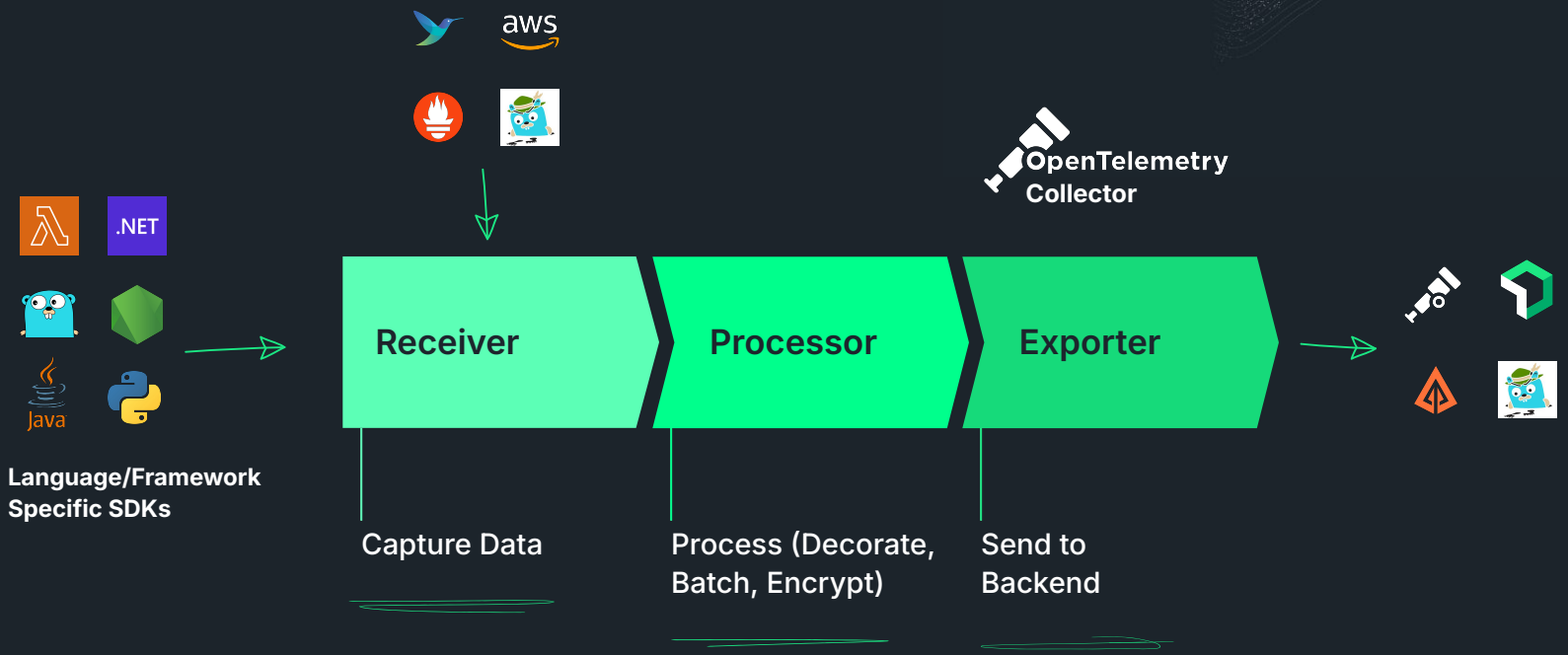
OpenTelemetry



new relic

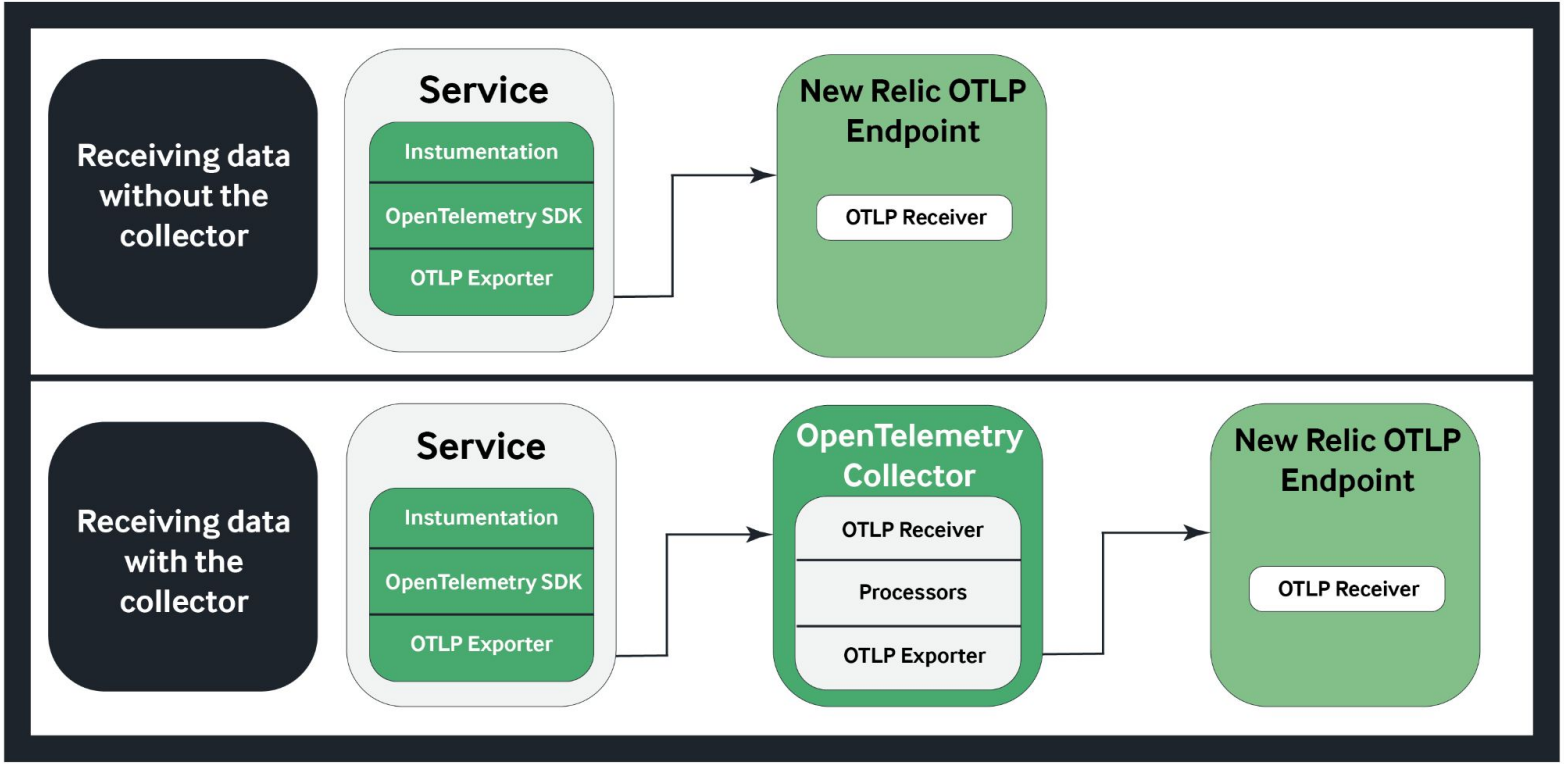
Putting it All Together

OpenTelemetry



new relic

OpenTelemetry - Collector



new relic

OpenTelemetry - Collector

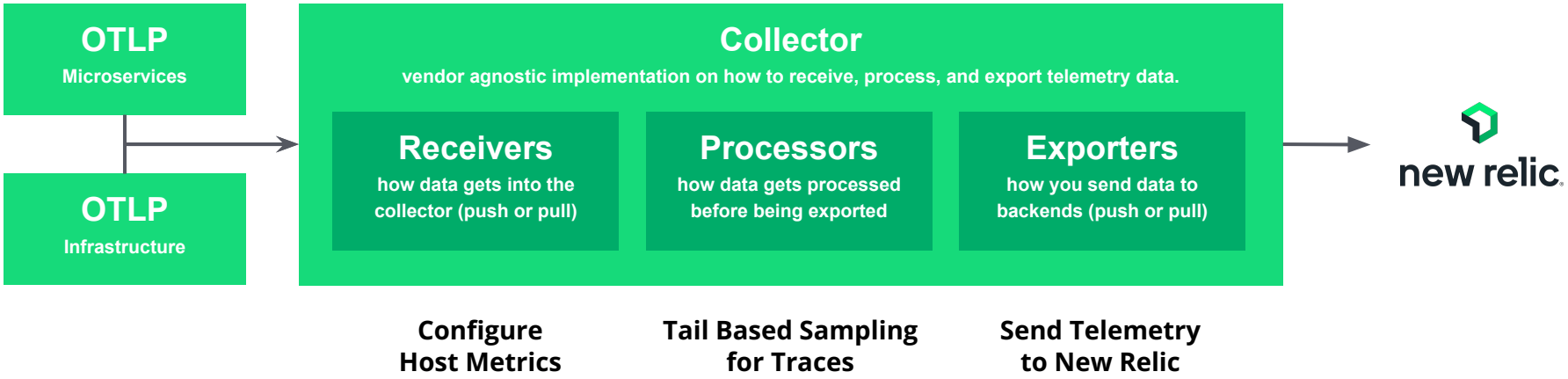
Benefits

- 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

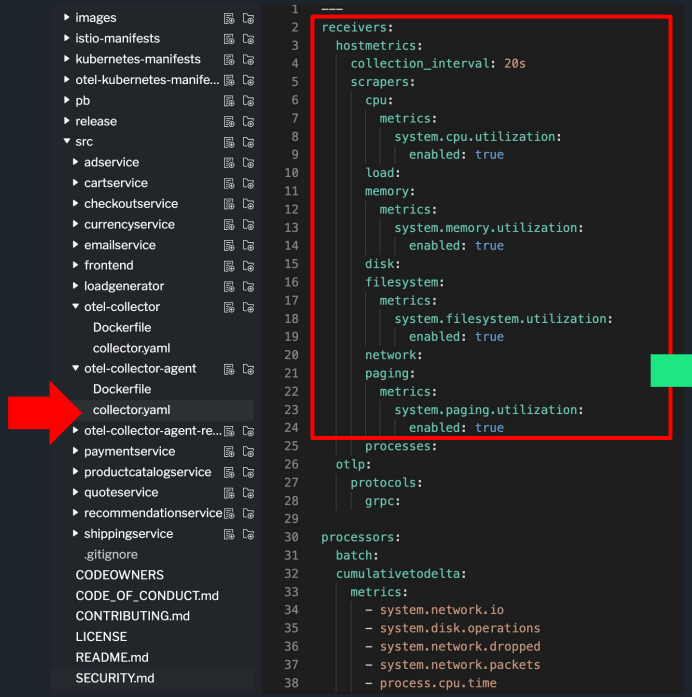
Downsides - nontrivial!

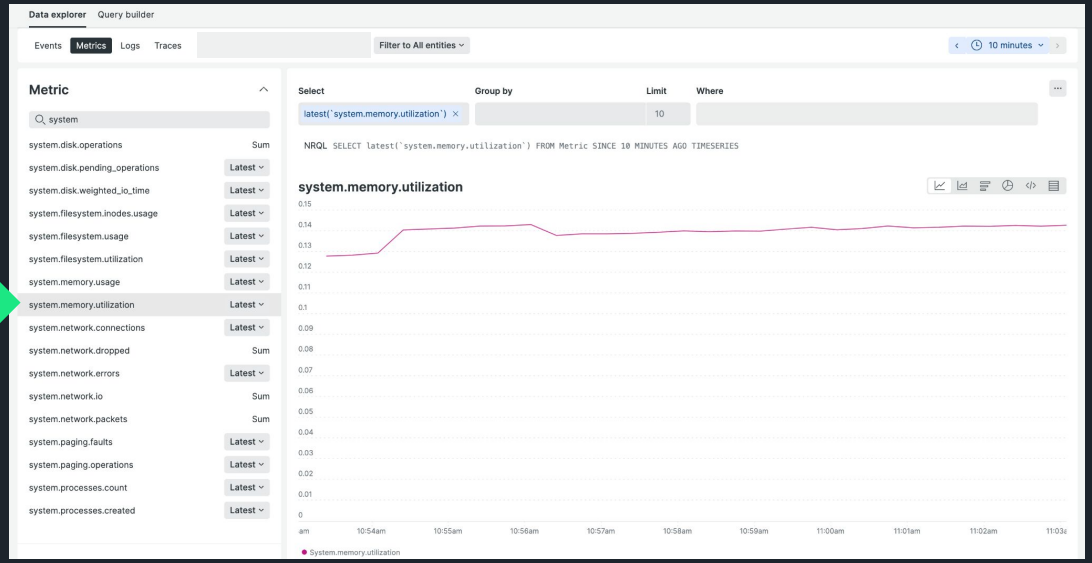
- Complex to scale
  - Deployment patterns
  - Load balancing
- Monitoring the collector is only available with ... the collector itself (at this time)
- Current stability status is “mixed”

OpenTelemetry - Collector



Collector in Action - Host Metrics



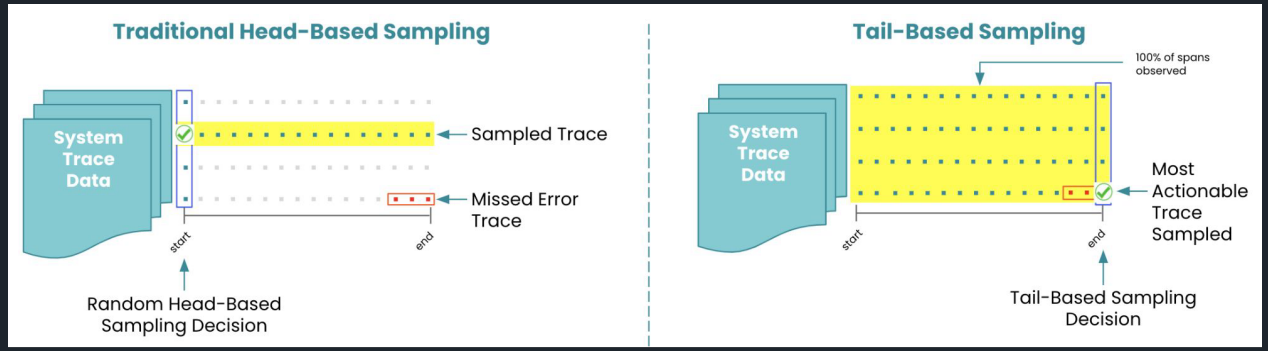


Collect additional infrastructure metrics for OpenTelemtry

new relic

HEAD vs TAIL Sampling

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

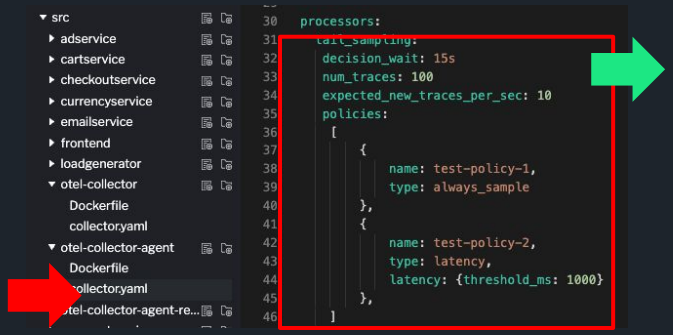


Head-based sampling works well for an overall statistical sampling of requests through a distributed system.

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

new relic

Collector in Action - Tail Based Sampling



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.

new relic