new relic

Tracing Kafka with OpenTelemetry

new relic

Tracing Kafka with OpenTelemetry

new relic

new relic

Tracing Kafka with OpenTelemetry

new relic

Safe Harbor

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

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

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

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

whoami

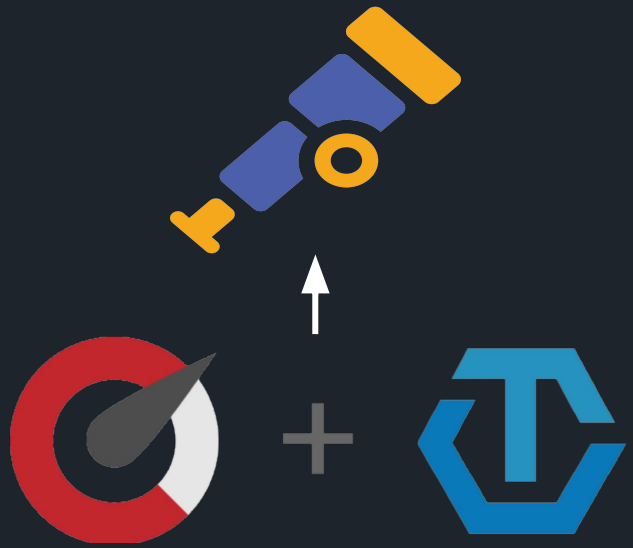
Harry Kimpel

Principal Developer Relations Engineer
New Relic
@harrykimpel

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



The Rise of OpenTelemetry

Vendor Neutral

Provides flexibility to change backend

Interoperable

End-to-end visibility with standard instrumentation

Configurable

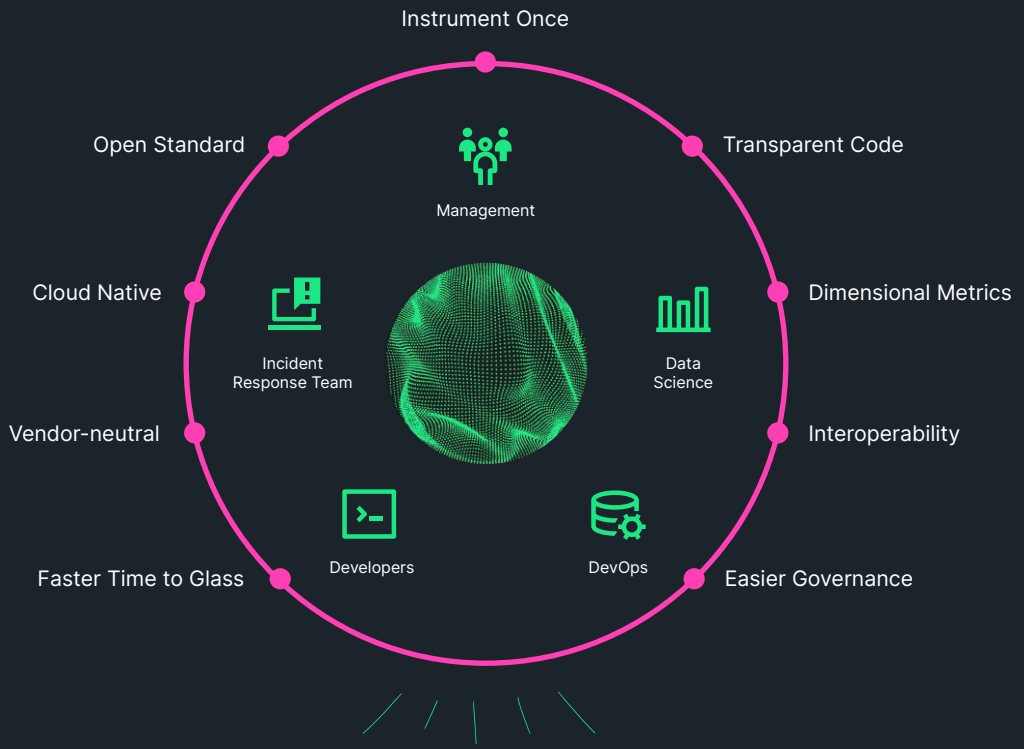
Pick and choose from the pieces what is needed

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

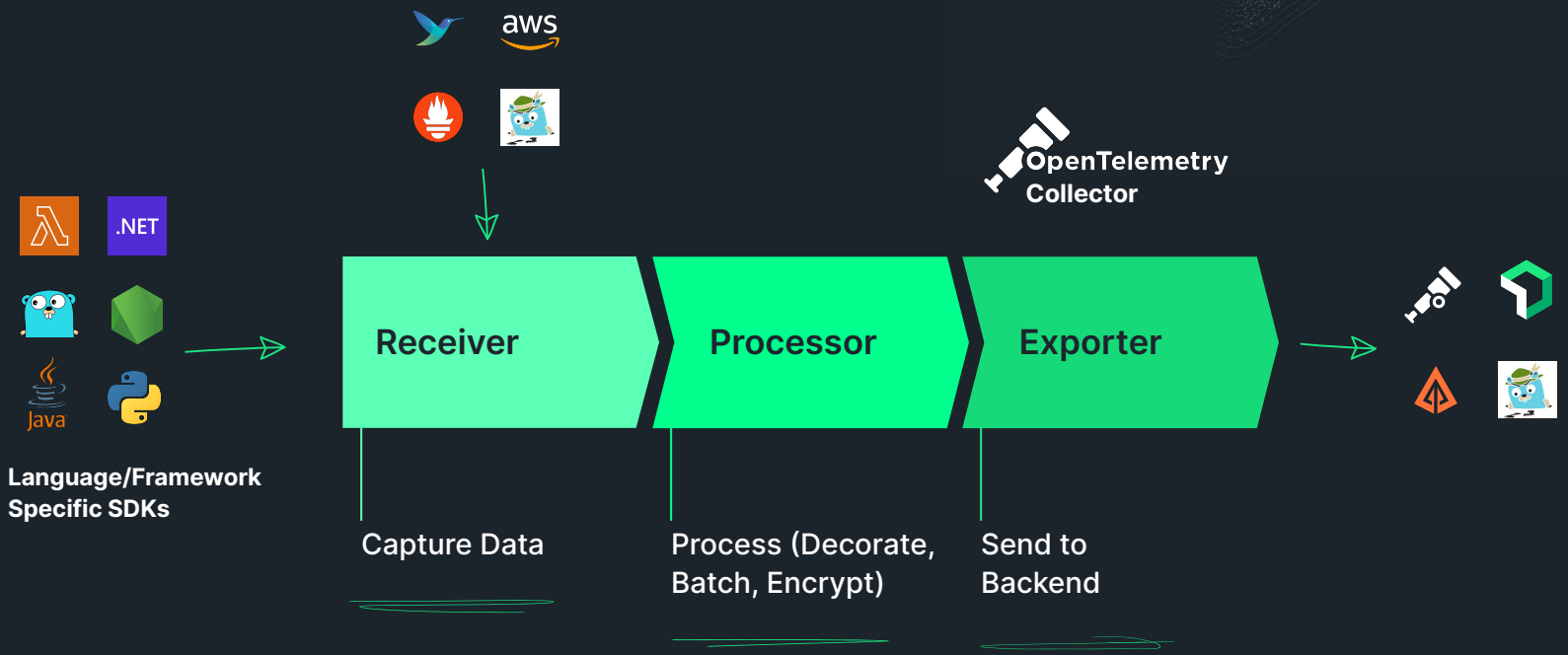
new relic

Why OpenTelemetry?



new relic

Snapshot of OpenTelemetry

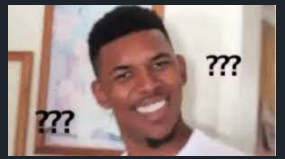


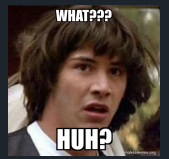
new relic

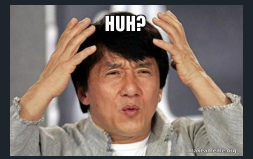
The Rise of OpenTelemetry

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

BUT WAIT ...







new relic

Design Considerations & Guidelines

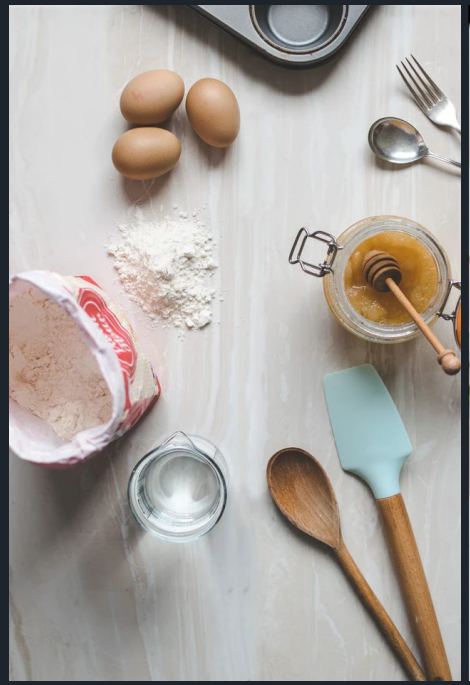
Customize Everything
(aka OTel)

Ingredients

Custom Made

Flavours

Java, .NET and others mature



End Product
(aka NR Agents)

Ready to Eat

Packaged

Scalable

Most Languages

new relic

Design Considerations & Guidelines

Community Supported

Doesn't have SLA/Dedicated Engineering team

Built by NR

SLAs/Support provided

Status & Releases

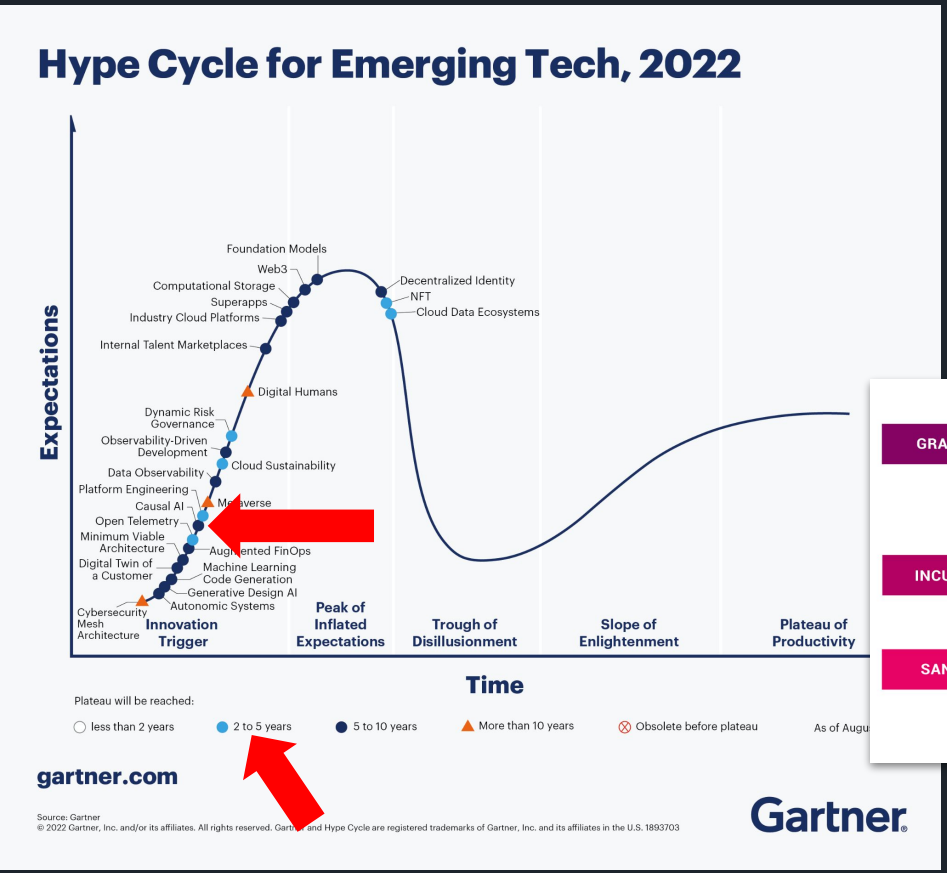
Screenshot from
https://opentelemetry.io/docs/instrumentation/
taken
February 12th, 2023 July 26th, 2024

Status and Releases

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

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

Design Considerations & Guidelines



Top 6 Questions To Consider

Do you understand the difference between Monitoring vs Observability?

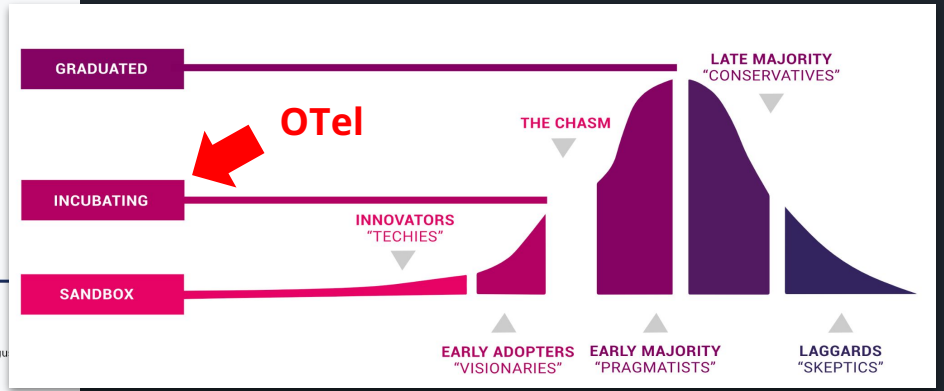
Do you understand the basics of **M**etrics, **L**ogs & **T**races?

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?



new relic

Automatic instrumentation with Java

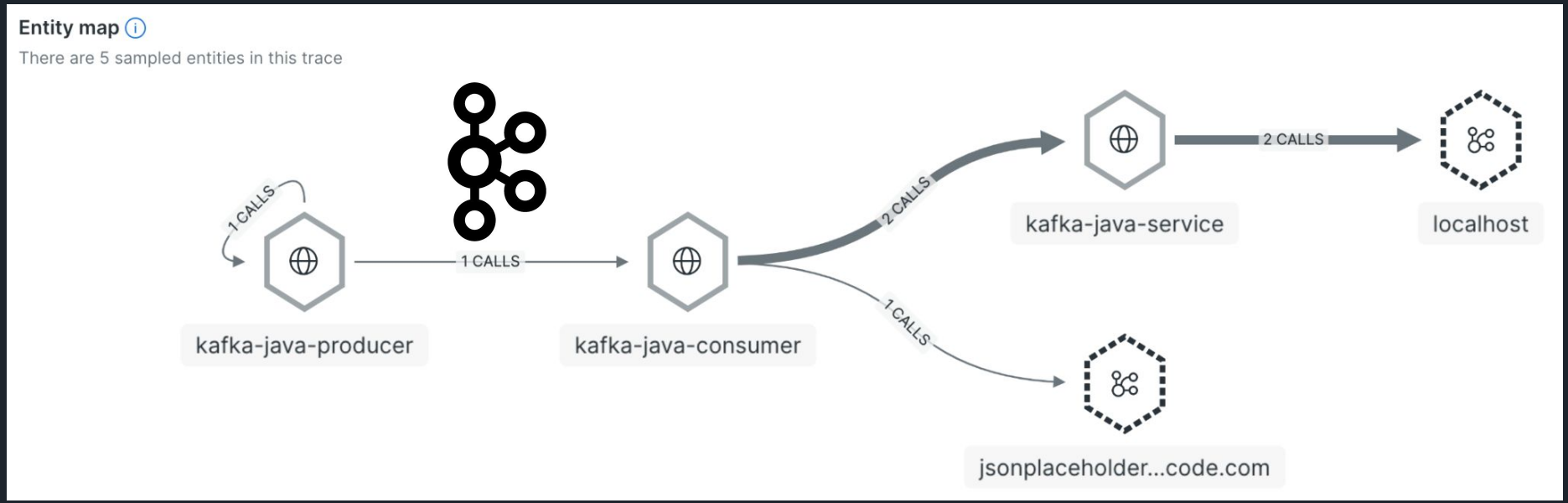
otel-auto-instr > demoConsumer > $ run.sh

```
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
```

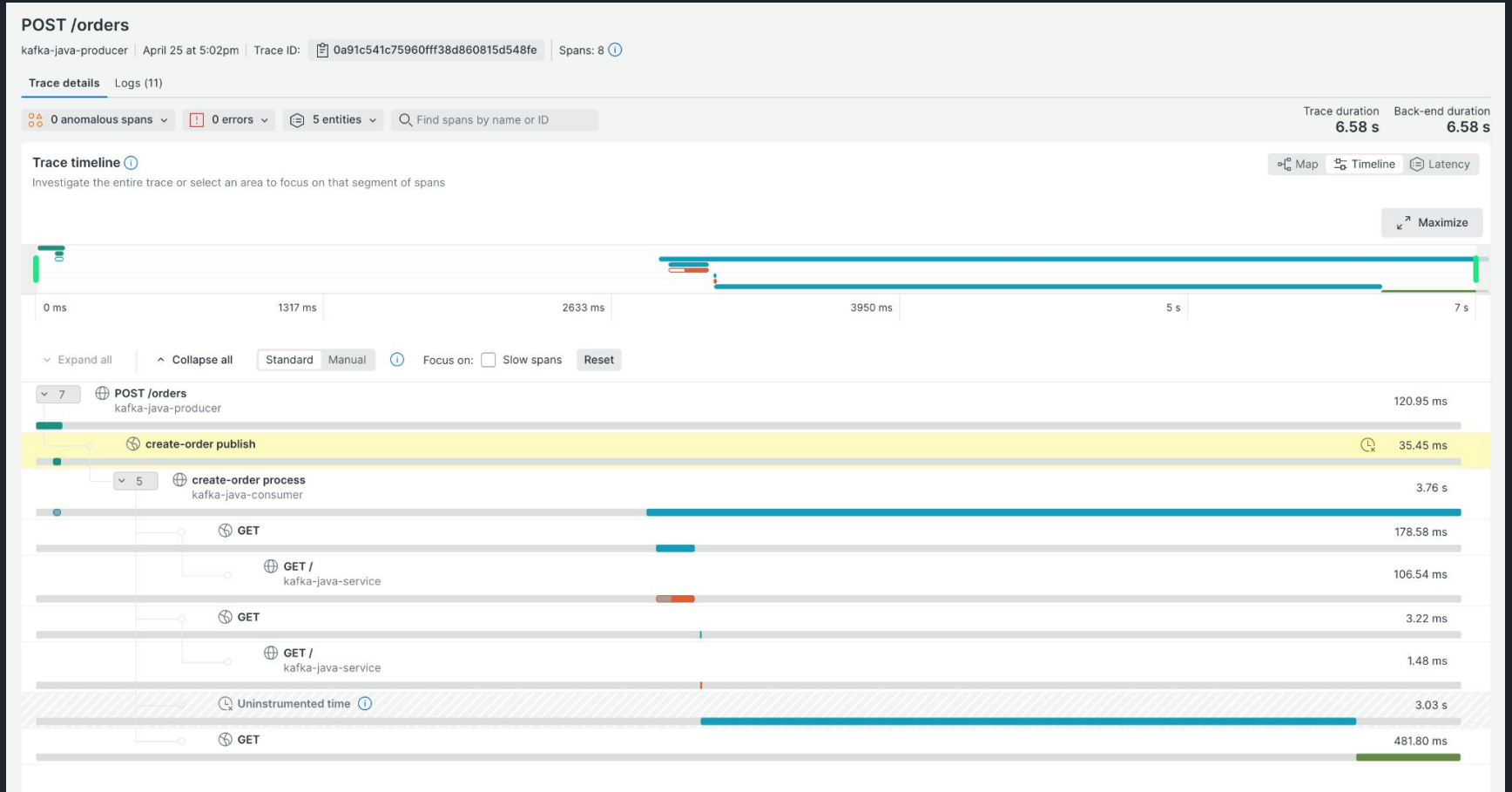

new relic

Automatic instrumentation with Java



new relic

Automatic instrumentation with Java



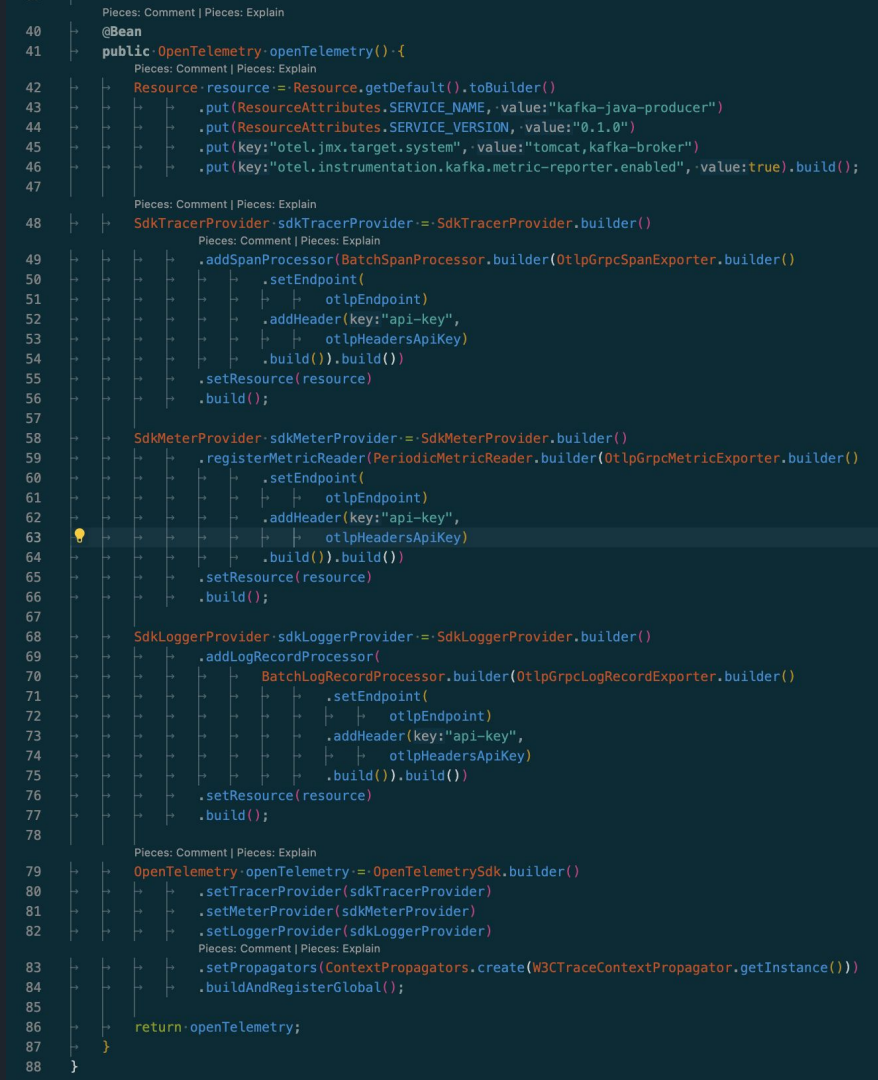
Manual instrumentation with Java

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
@Bean
public OpenTelemetry openTelemetry() {
    Pieces: Comment | Pieces: Explain
    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();

    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();

    SdkMeterProvider sdkMeterProvider = SdkMeterProvider.builder()
            .registerMetricReader(PeriodicMetricReader.builder(OtlpGrpcMetricExporter.builder()
                    .setEndpoint(
                        otlpEndpoint)
                    .addHeader(key:"api-key",
                        otlpHeadersApiKey)
                    .build()).build())
            .setResource(resource)
            .build();

    SdkLoggerProvider sdkLoggerProvider = SdkLoggerProvider.builder()
            .addLogRecordProcessor(
                    BatchLogRecordProcessor.builder(OtlpGrpcLogRecordExporter.builder()
                            .setEndpoint(
                                otlpEndpoint)
                            .addHeader(key:"api-key",
                                otlpHeadersApiKey)
                            .build()).build())
            .setResource(resource)
            .build();

    Pieces: Comment | Pieces: Explain
    OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
            .setTracerProvider(sdkTracerProvider)
            .setMeterProvider(sdkMeterProvider)
            .setLoggerProvider(sdkLoggerProvider)
            Pieces: Comment | Pieces: Explain
            .setPropagators(ContextPropagators.create(W3CTraceContextPropagator.getInstance()))
            .buildAndRegisterGlobal();

    return openTelemetry;
}
}
```

Manual instrumentation with Java - Kafka producer

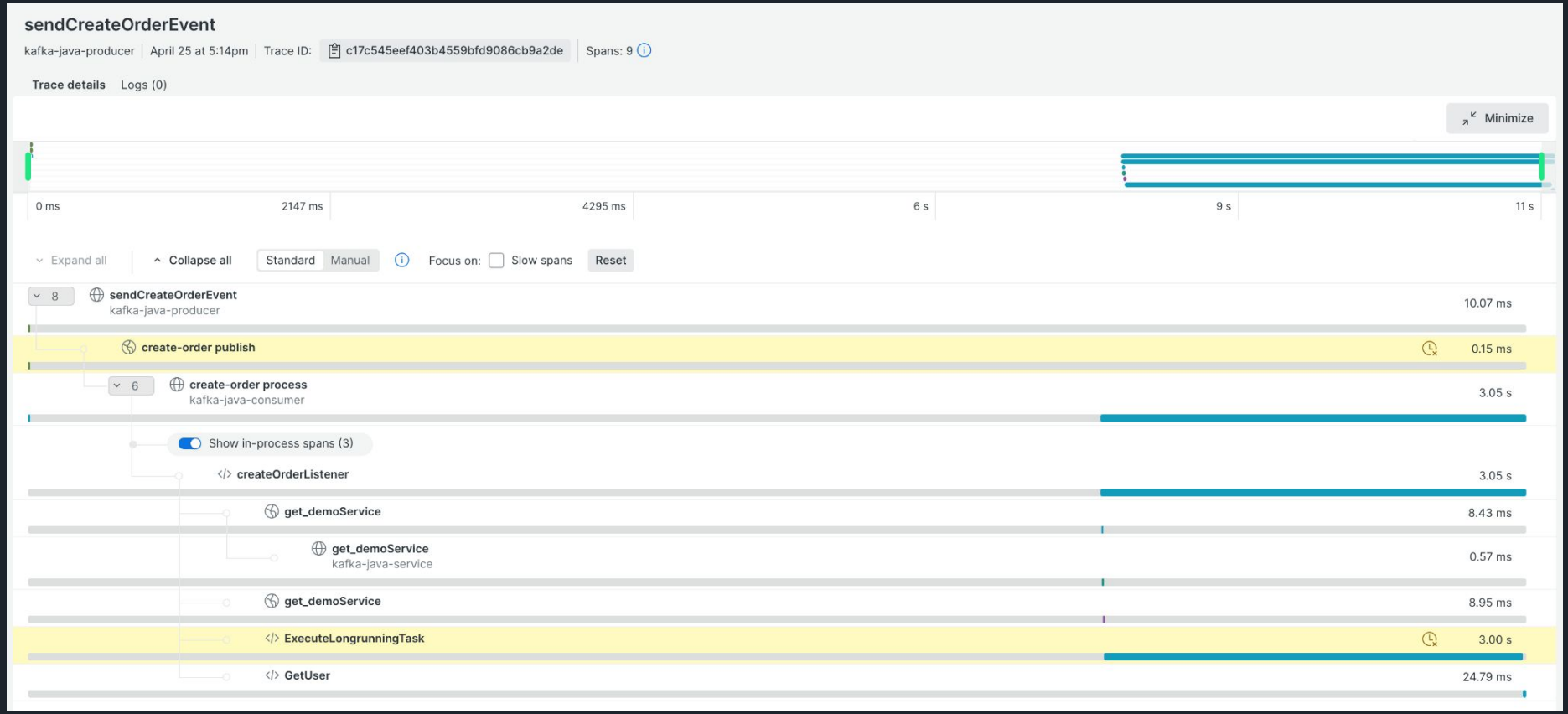
```
Pieces: Comment | Pieces: Explain
22      @Bean
23      public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
24          Map<String, Object> config = new HashMap<>();
25          config.put(
26                  org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
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



Java supported libraries

... see **GitHub repo** for details

- Disabled instrumentations

Libraries / Frameworks

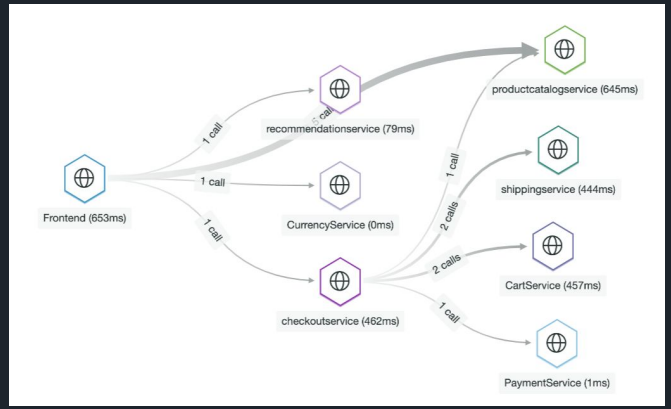
These are the supported libraries and frameworks:

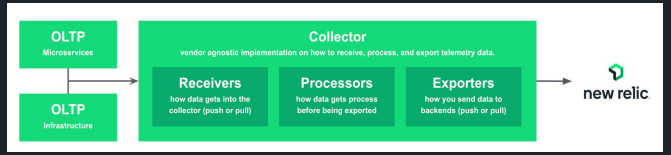
Library/Framework	Auto-instrumented versions	Standalone Library Instrumentation [1]	Semantic Conventions
Akka Actors	2.3+	N/A	Context propagation
Akka HTTP	10.0+	N/A	HTTP Client Spans, HTTP Client Metrics, HTTP Server Spans, HTTP Server Metrics, Provides `http.route` [2]
Alibaba Druid	1.0+	opentelemetry-alibaba-druid-1.0	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]

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



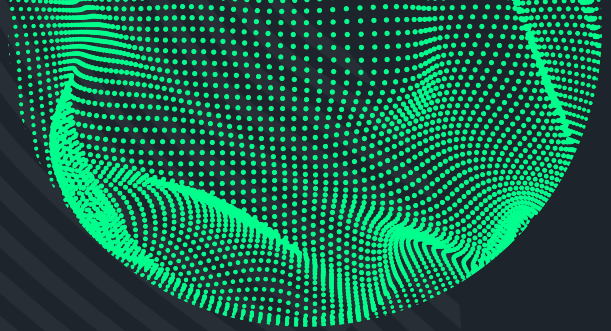


new relic

Reading Materials for OpenTelemetry

Reading List

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

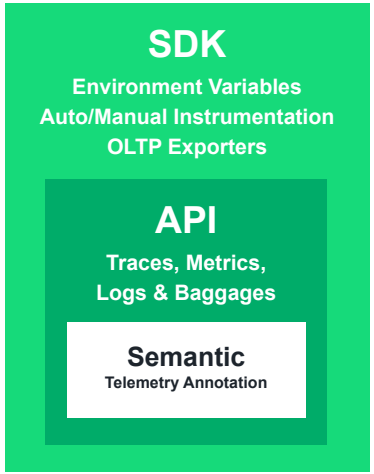


Appendix

some more information

new relic

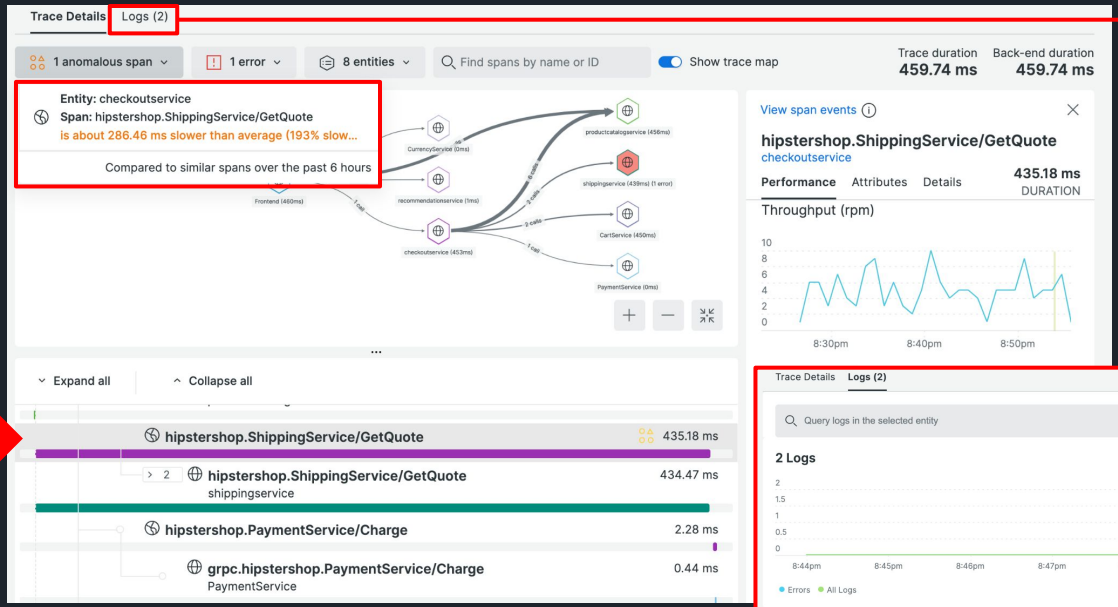
OpenTelemetry - Instrumentation



Core Concepts on Instrumentation

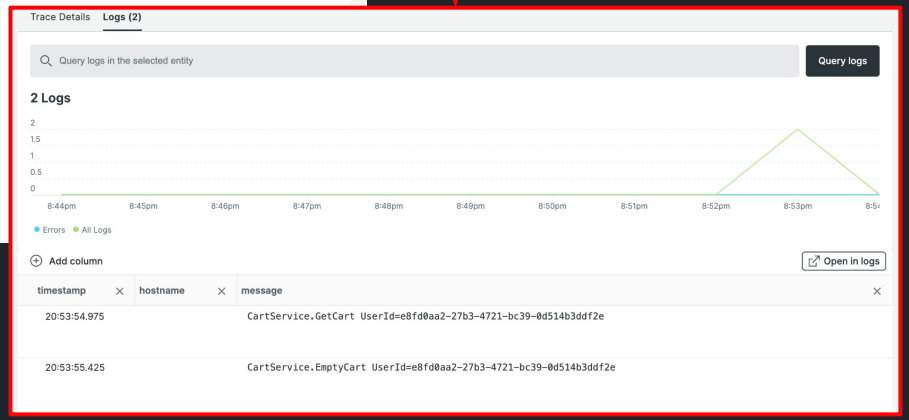
- **Semantic Conventions** - annotate telemetry with attributes specific to the represented operation, such as HTTP calls.
- **API** - standard way to collect instrumentation data.
- **SDK** - language-specific implementation of the API.
    - SDKs incorporate automatic instrumentation for common libraries 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



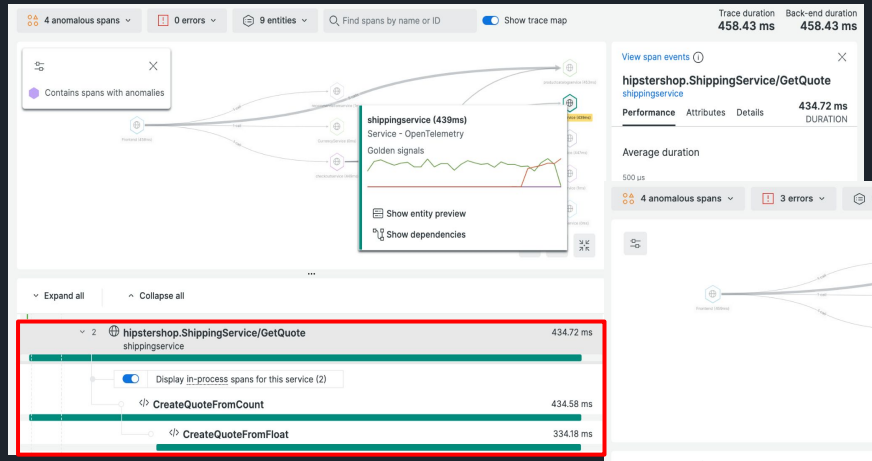
Anomalous Span Detection in Distributed Tracing for OpenTelemetry

Logs in Context (correlate to the right trace)

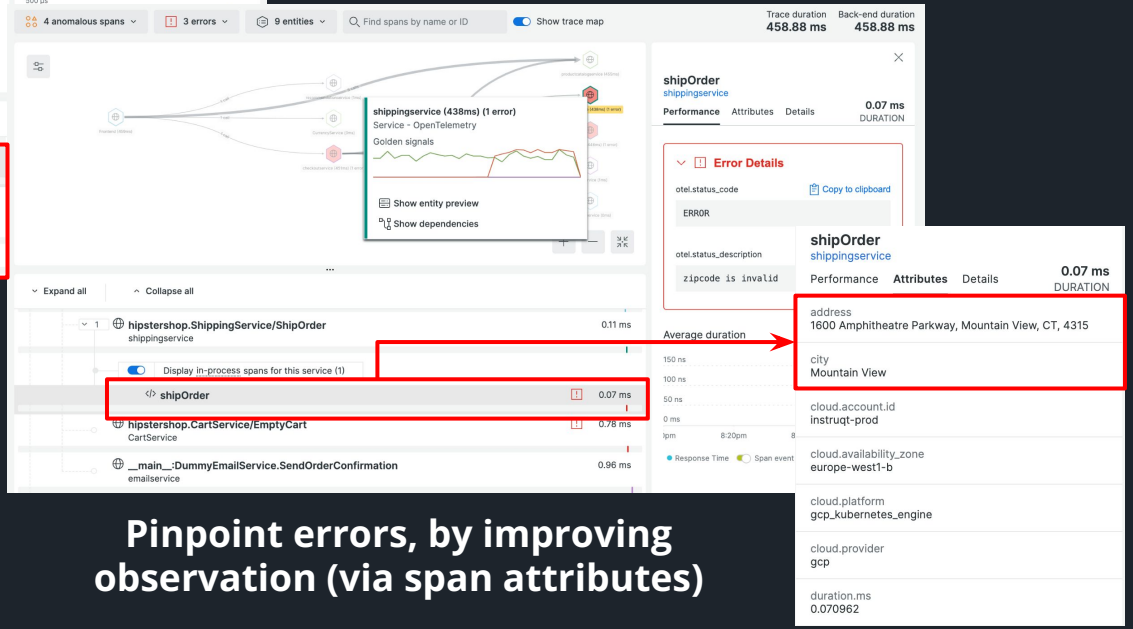


new relic

Instrumentation in Action



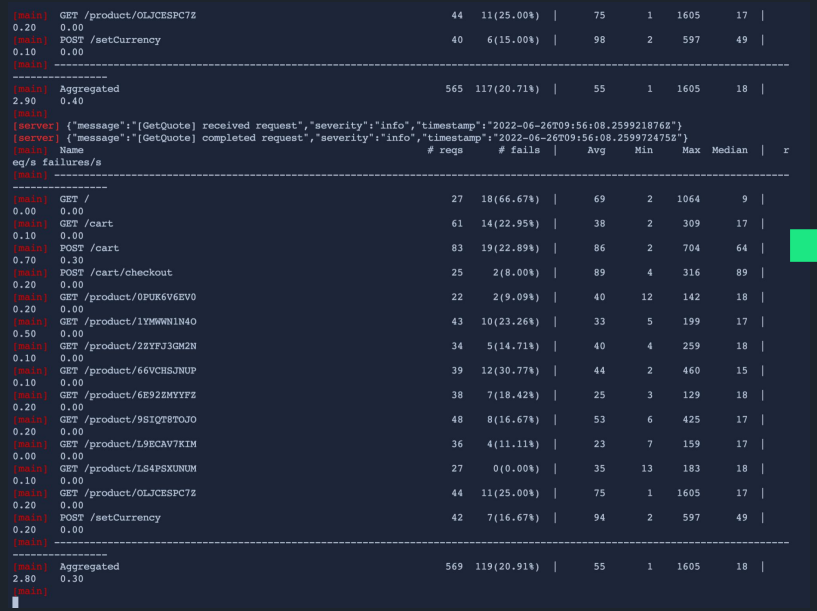
Get deeper into the app stack, by building spans.



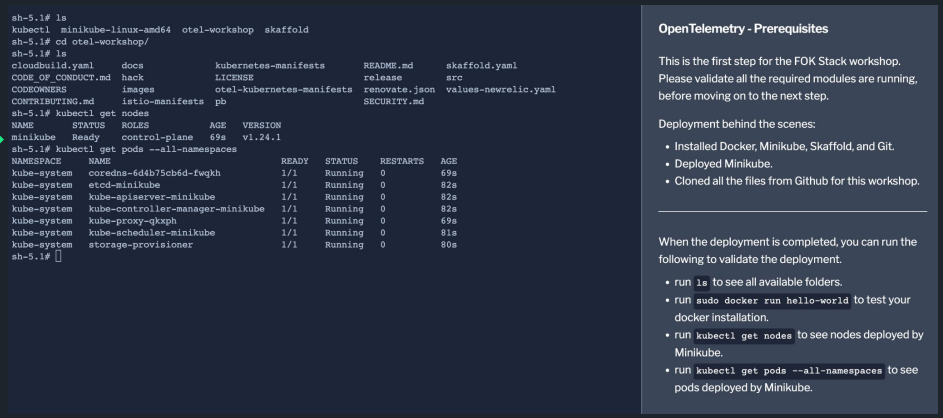
Pinpoint errors, by improving observation (via span attributes)

new relic

OpenTelemetry in Action



Deploy the app, and generate load (successful deployment)



Validate K8s is running, and follow the instructions

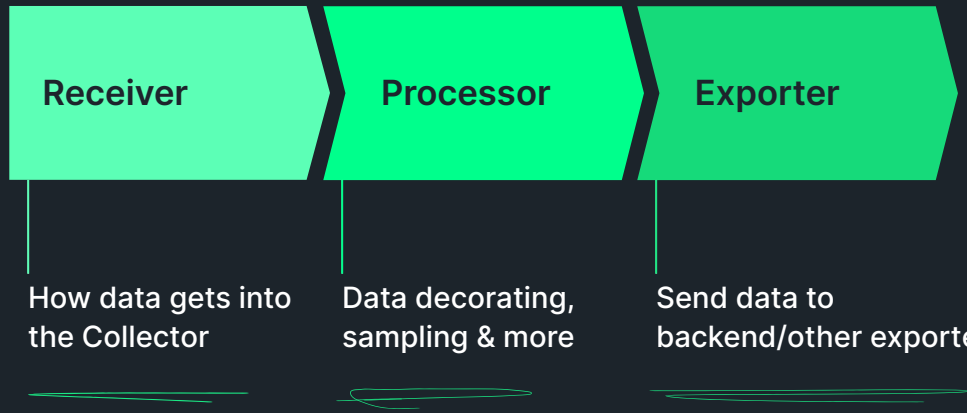
Important - deploy might take up to 10 mins

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

new relic

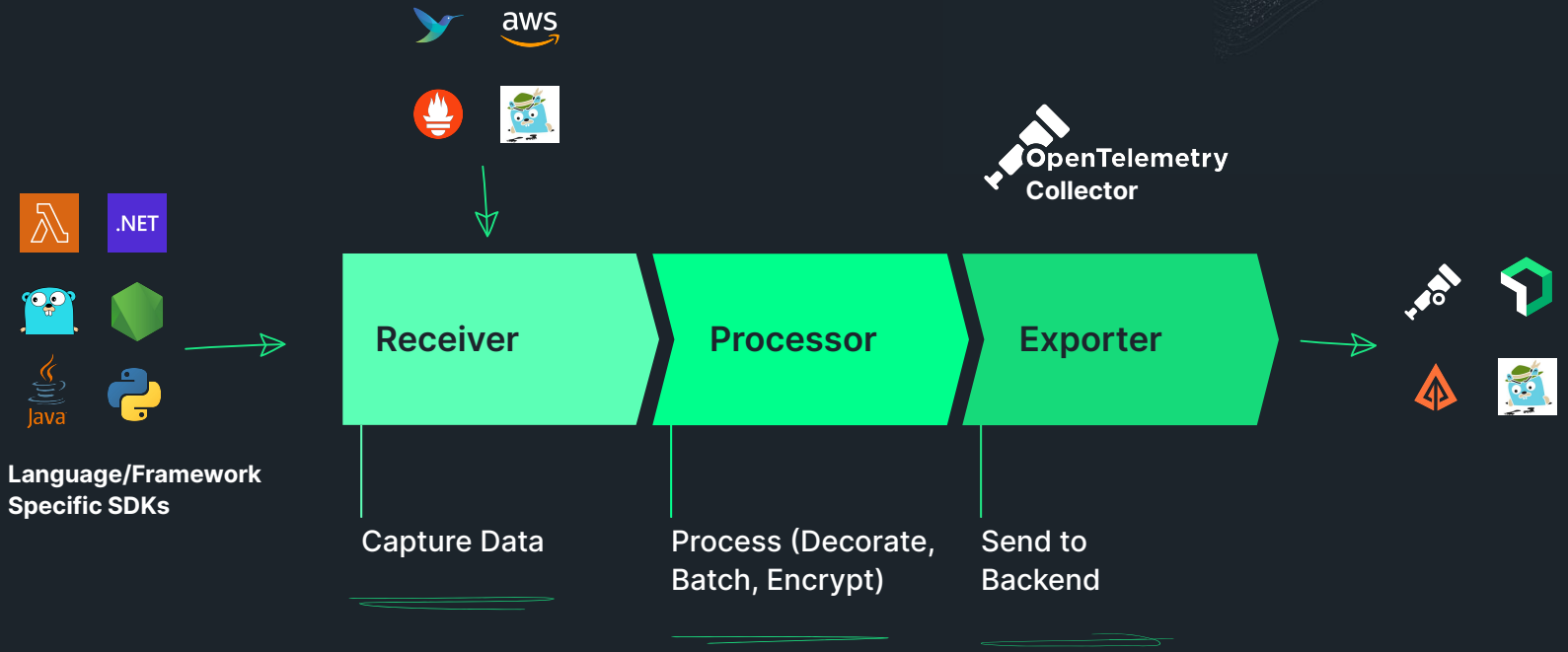
Collector Component

OpenTelemetry



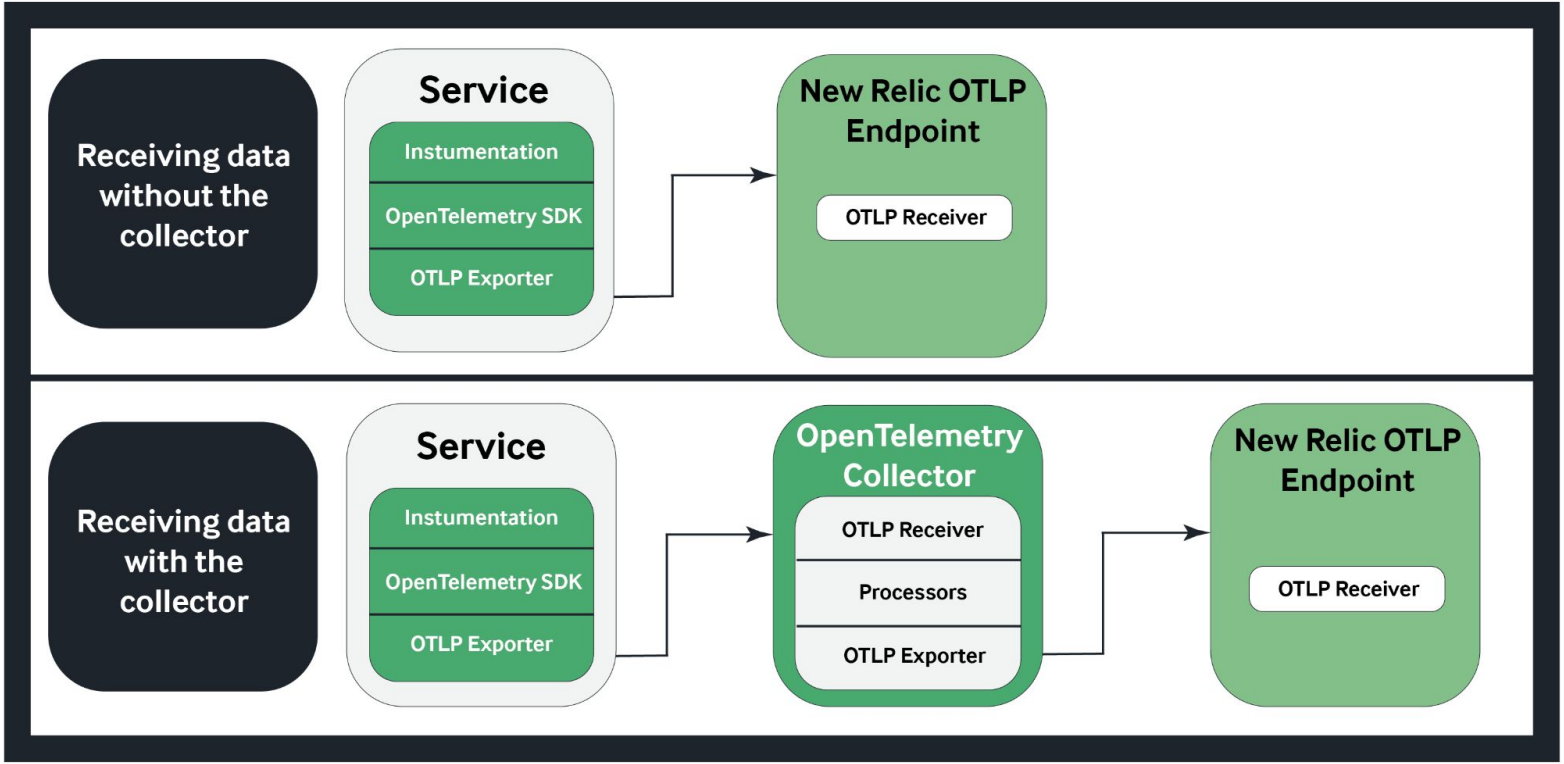
Putting it All Together

OpenTelemetry



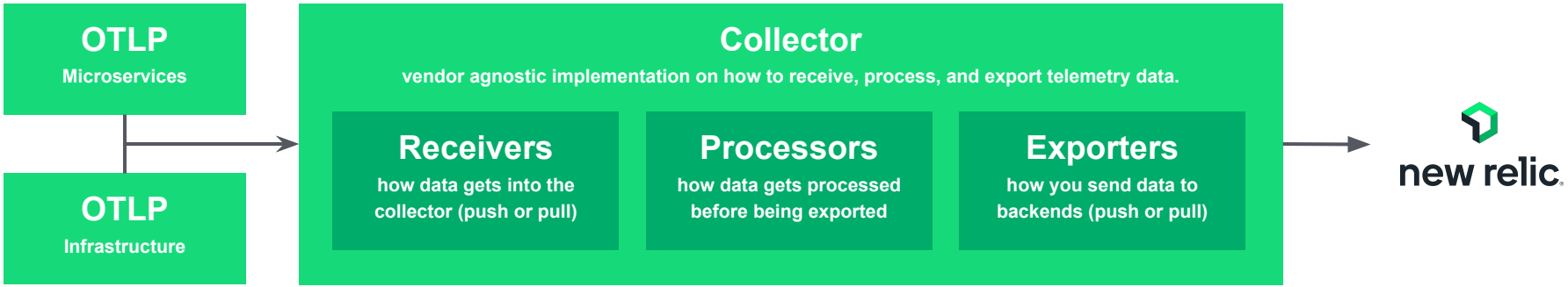
new relic

OpenTelemetry - Collector



new relic

OpenTelemetry - Collector



a **proxy** that processes multiple telemetry formats
(e.g., OTLP, Jaeger, Prometheus, as well as many commercial/proprietary formats)

OpenTelemetry - Collector

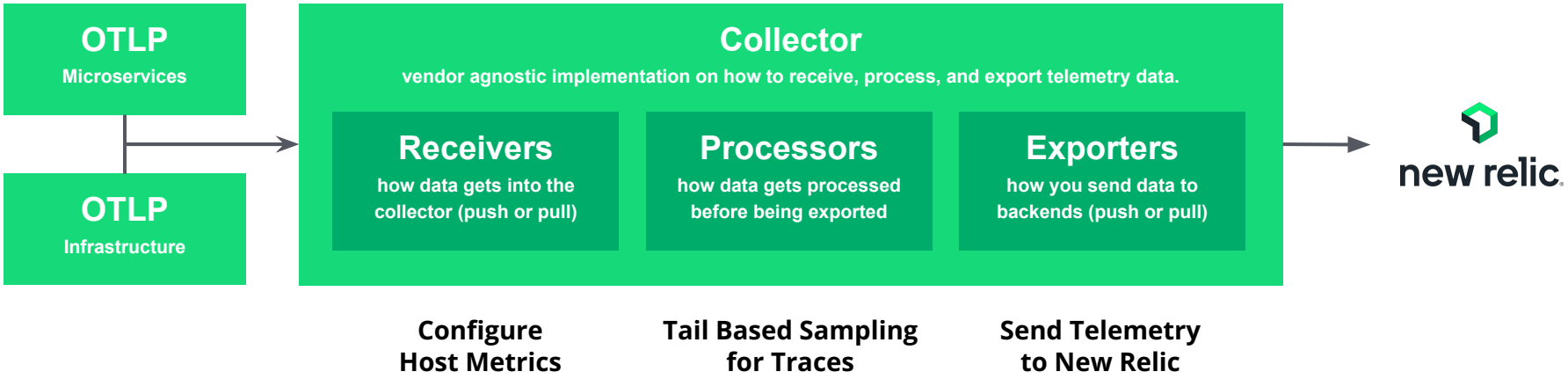
Benefits

- 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

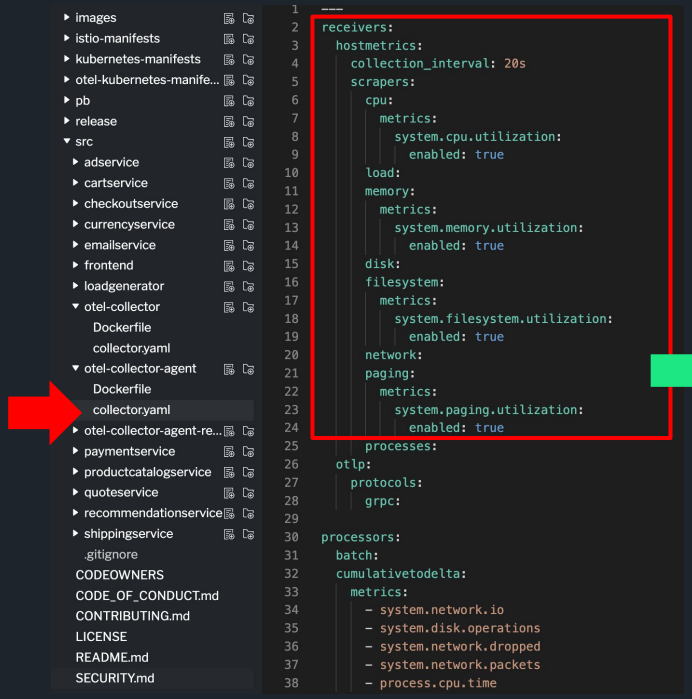
Downsides - nontrivial!

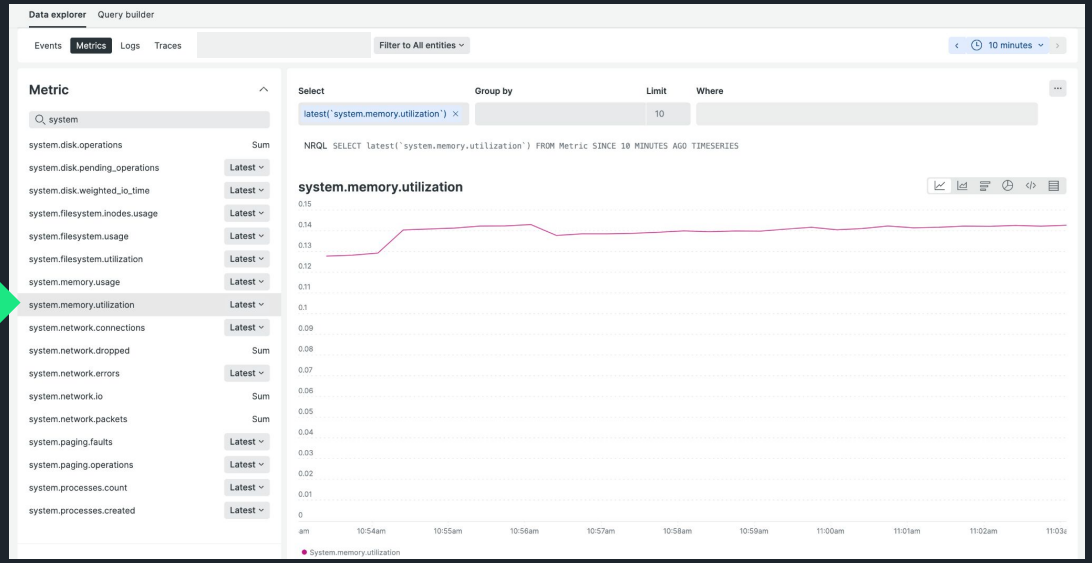
- Complex to scale
  - Deployment patterns
  - Load balancing
- Monitoring the collector is only available with … the collector itself (at this time)
- Current stability status is “mixed”

OpenTelemetry - Collector



Collector in Action - Host Metrics



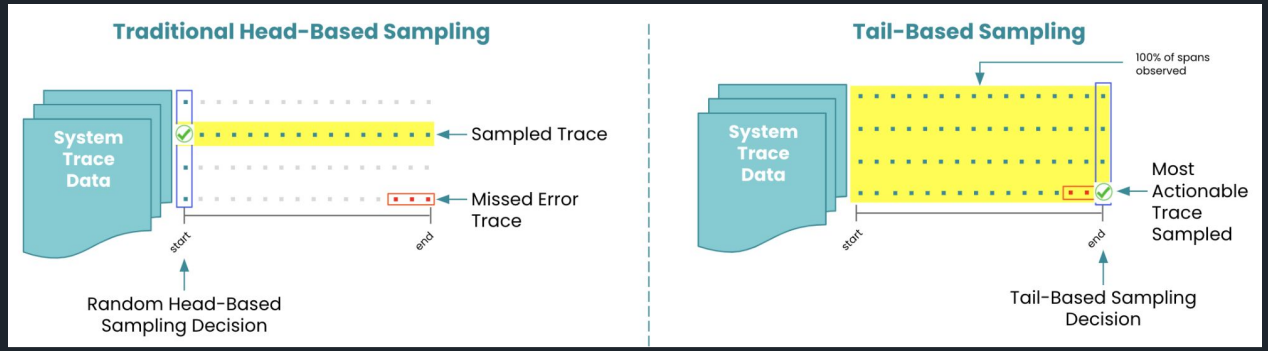


Collect additional infrastructure metrics for OpenTelemtry

new relic

HEAD vs TAIL Sampling

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

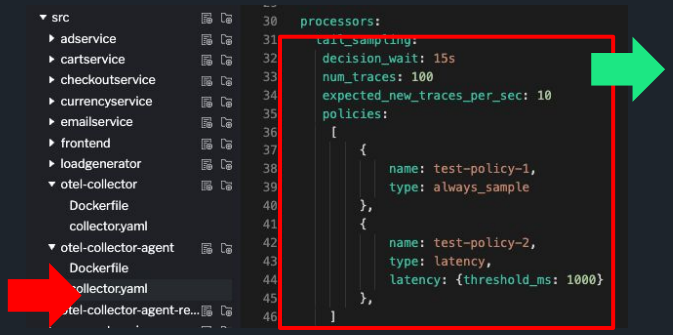


Head-based sampling works well for an overall statistical sampling of requests through a distributed system.

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