
Unlocking
Observability
in
Apache
Kafka-Based
Systems with
OpenTelemetry

whoami

Harry Kimpel

Principal Developer Relations Engineer
New Relic
@harrykimpel

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

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."

Who uses Kafka?

✋ Hands up if you have heard about OpenTelemetry

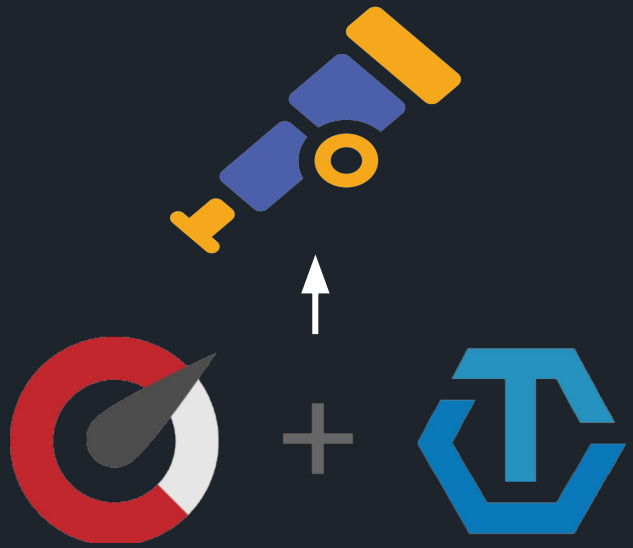
new relic

✋ 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



The Rise of OpenTelemetry

Vendor Neutral

Provides flexibility to change backend

Interoperable

End-to-end visibility with standard instrumentation

Configurable

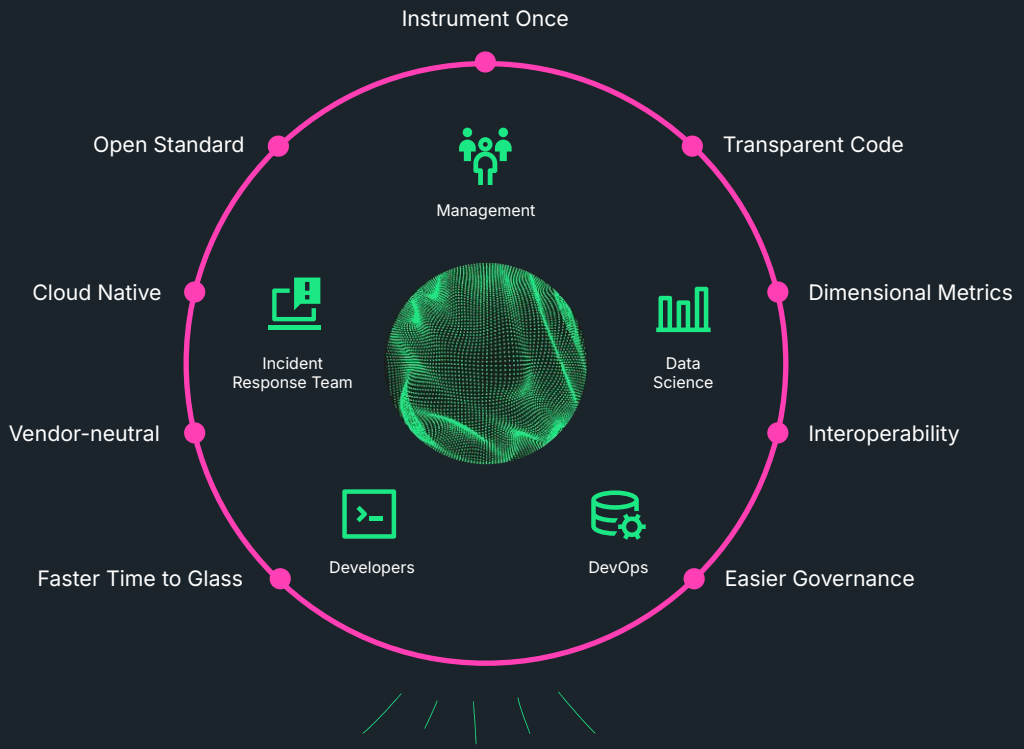
Pick and choose from the pieces what is needed

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

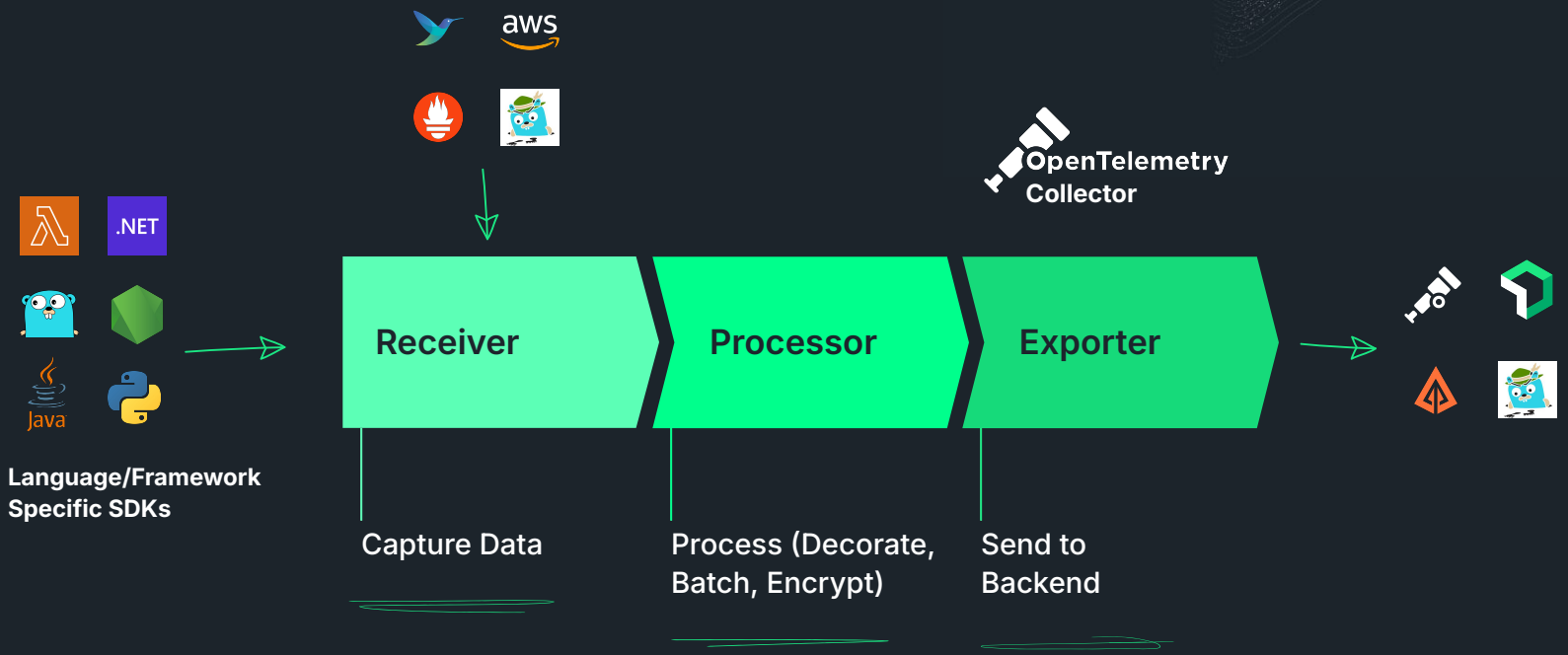
new relic

Why OpenTelemetry?



new relic

Snapshot of OpenTelemetry

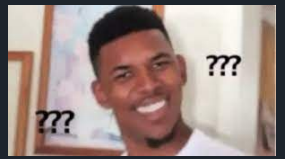


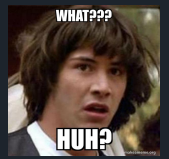
new relic

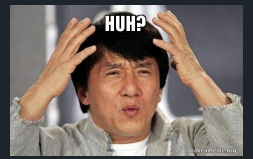
The Rise of OpenTelemetry

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

BUT WAIT ...







new relic

Design Considerations & Guidelines

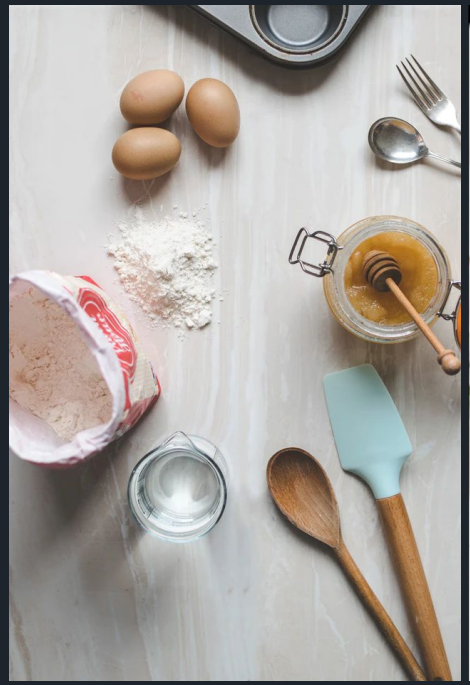
Customize Everything
(manual instrumentation)

Ingredients

Custom Made

Flavours

Java, .NET and others mature



End Product
(auto instrumentation)

Ready to Eat

Packaged

Scalable

Most Languages

new relic

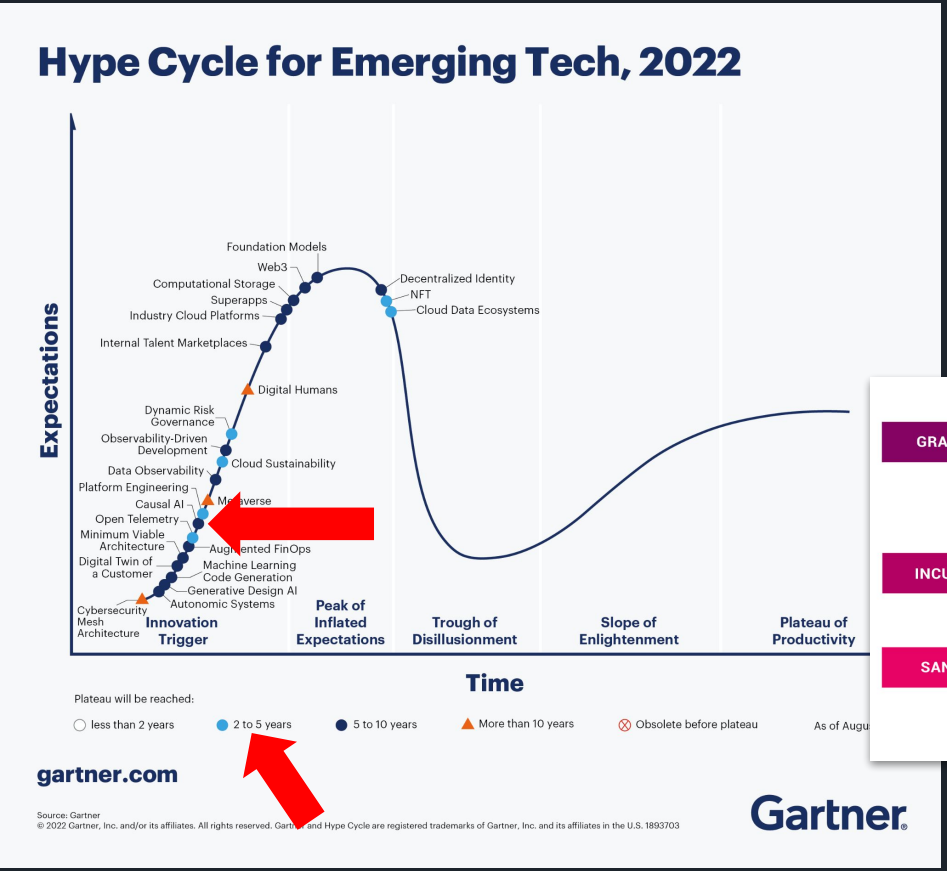
Screenshot from
https://opentelemetry.io/docs/instrumentation/
taken
Ja[illegible]25

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

Design Considerations & Guidelines



Top 6 Questions To Consider

Do you understand the difference between Monitoring vs Observability?

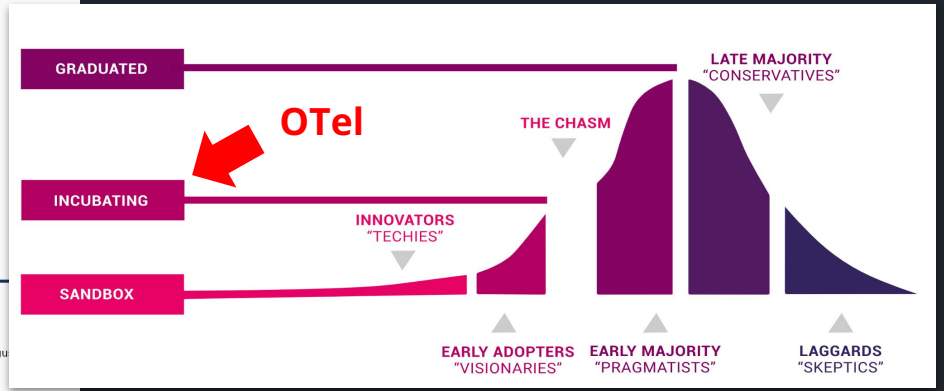
Do you understand the basics of **M**etrics, **L**ogs & **T**races?

Is tracing widely deployed to provide value?

Do you have experience in manual instrumentation?

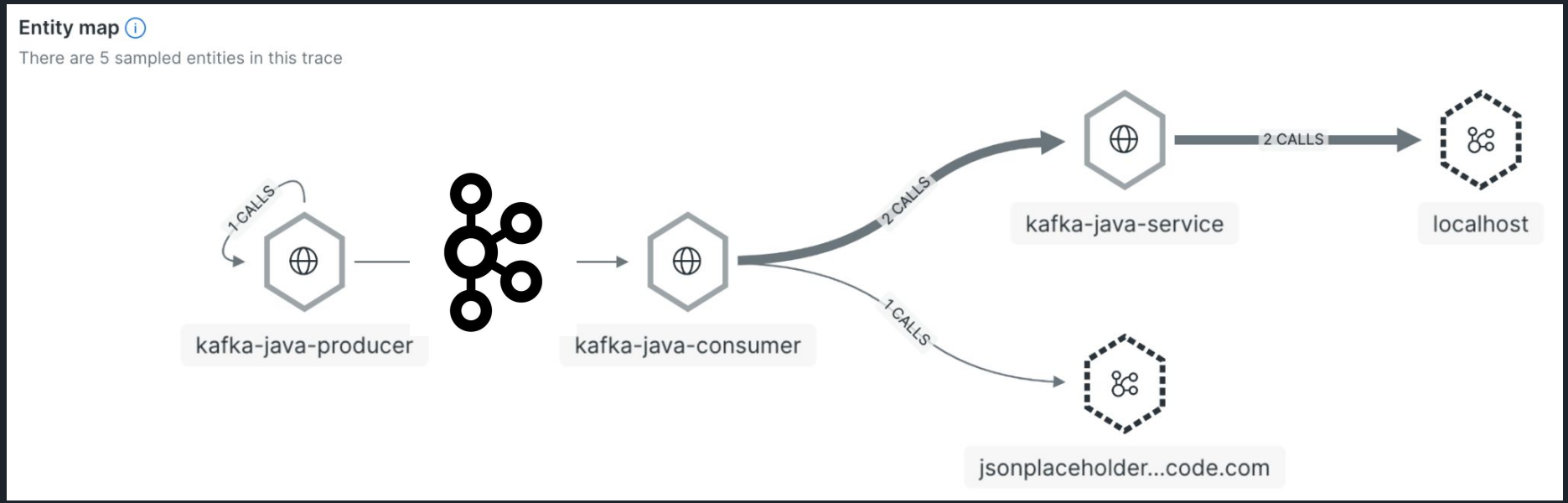
Is your app, friendly to manual instrumentation?

Is your environment or practice tolerant to change?



new relic

Automatic instrumentation with Java



new relic

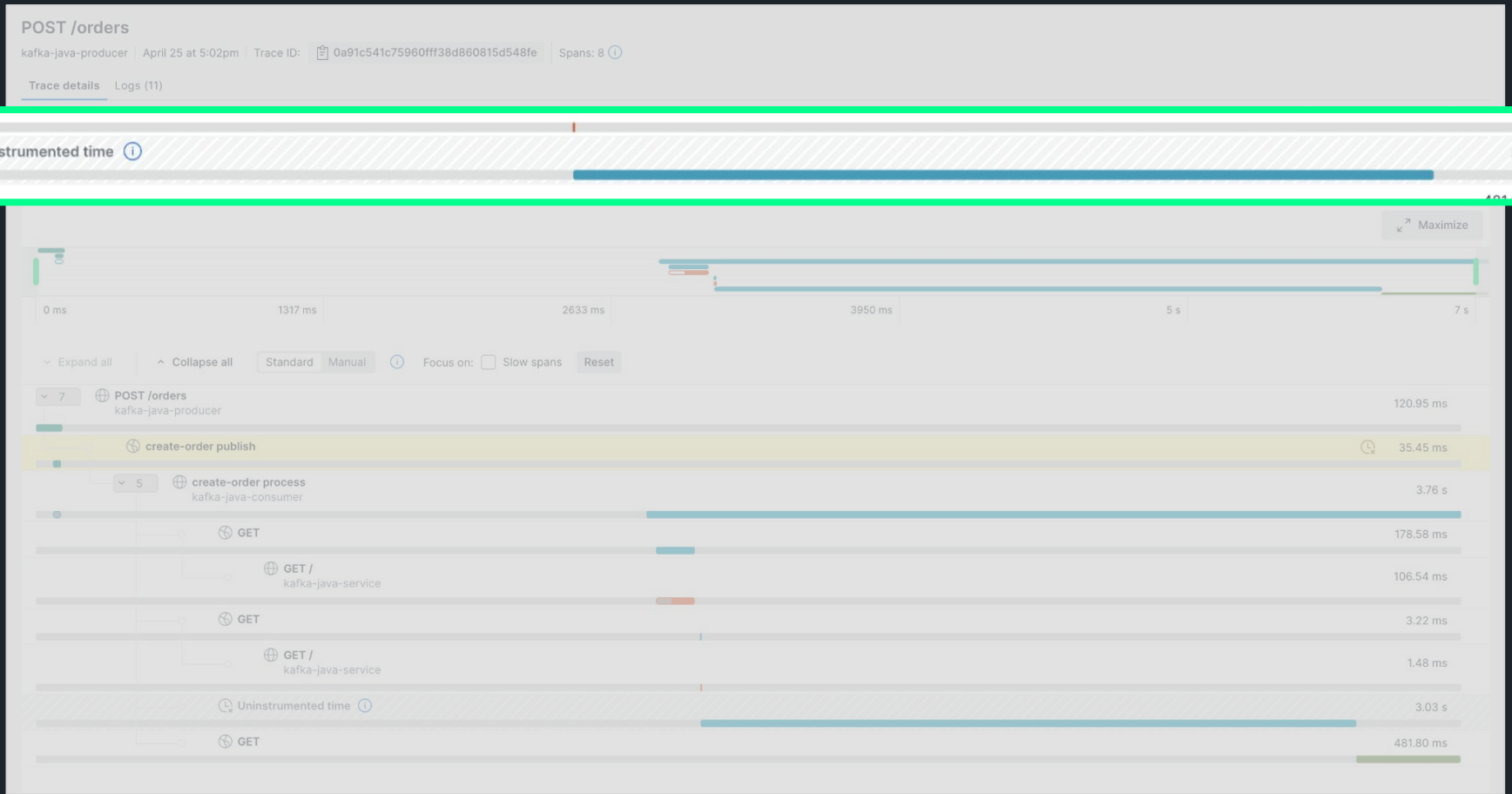
Automatic instrumentation with Java

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
```


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
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 - Kafka producer

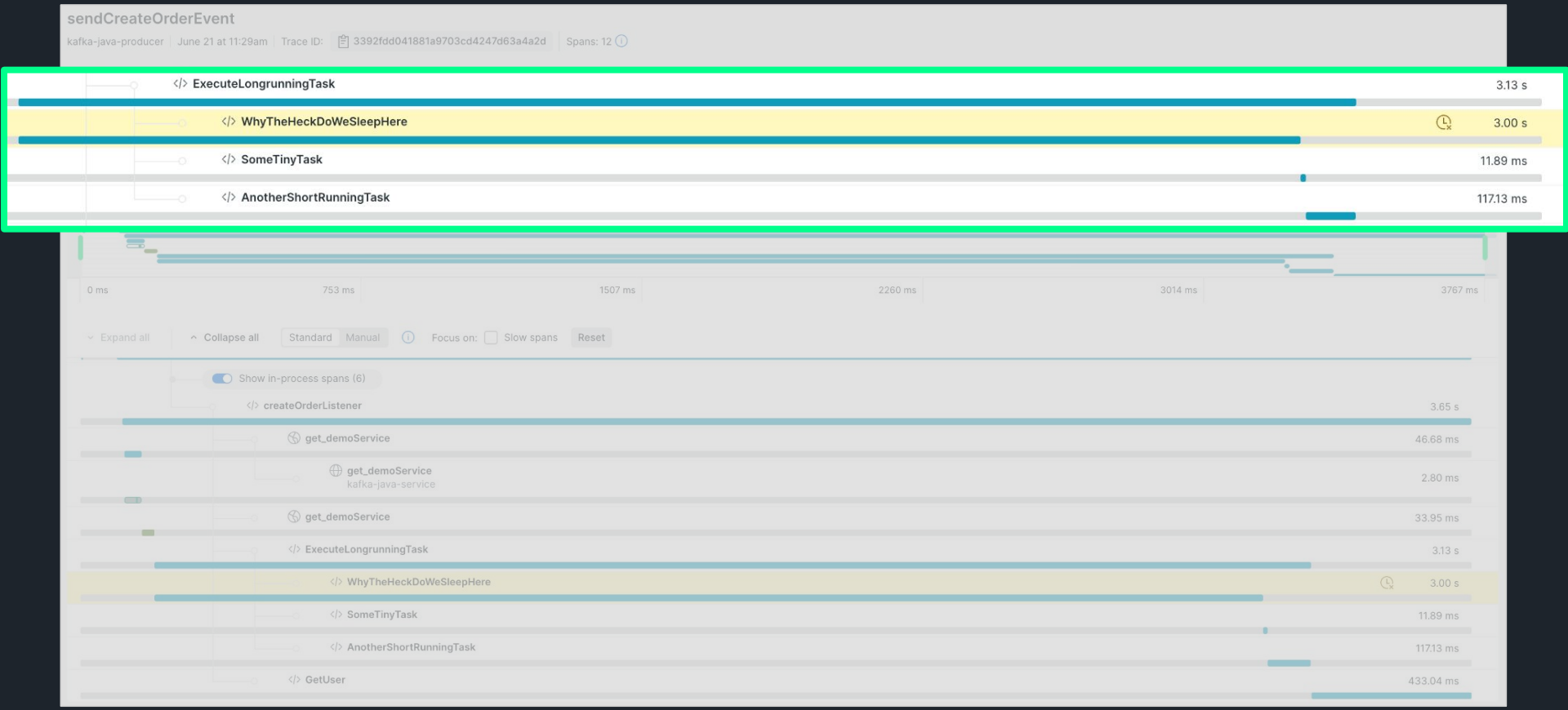
```
Pieces: Comment | Pieces: Explain
22      @Bean
23      public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
24          Map<String, Object> config = new HashMap<>();
25          config.put(
26                  org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,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

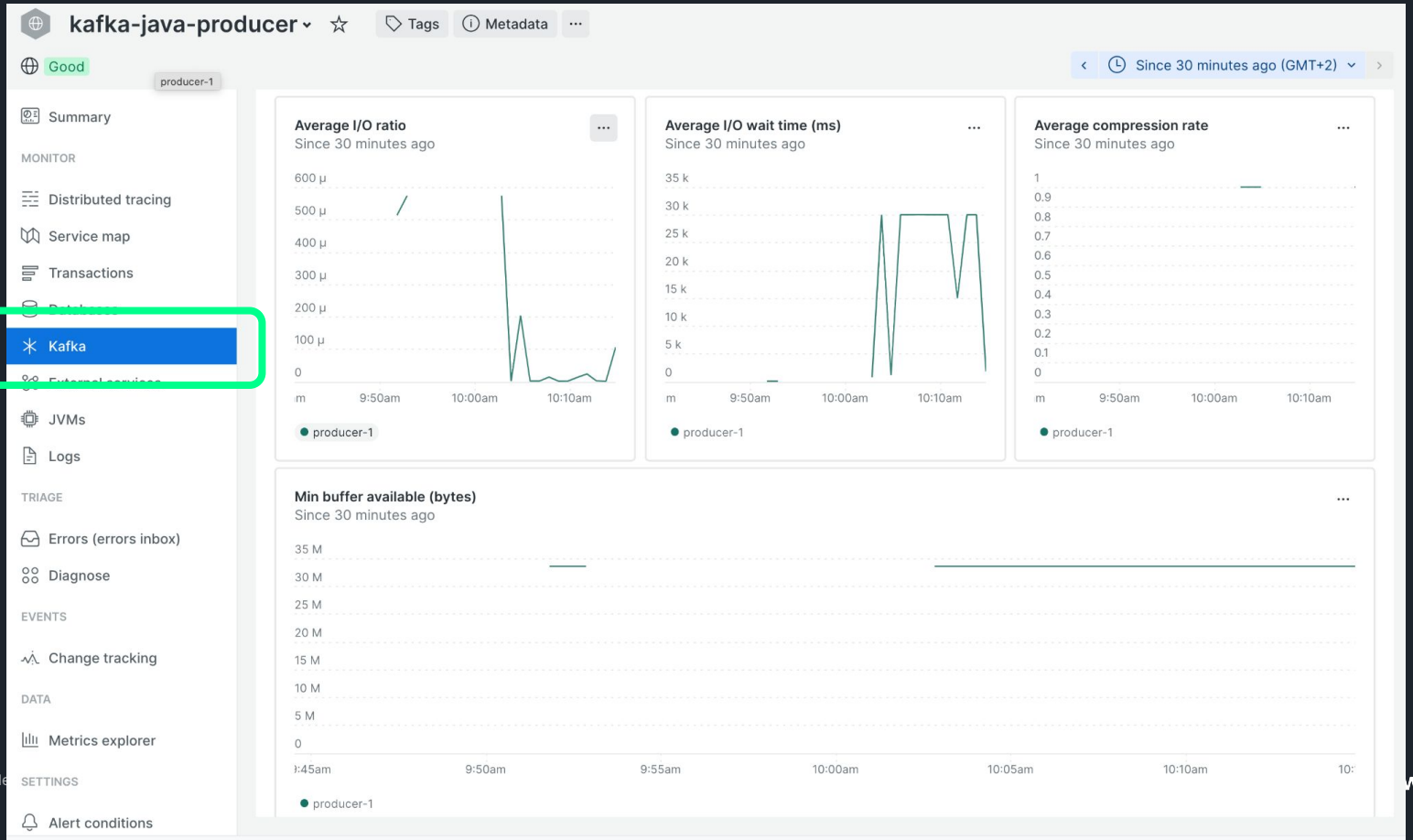


Manual instrumentation with Java - custom spans

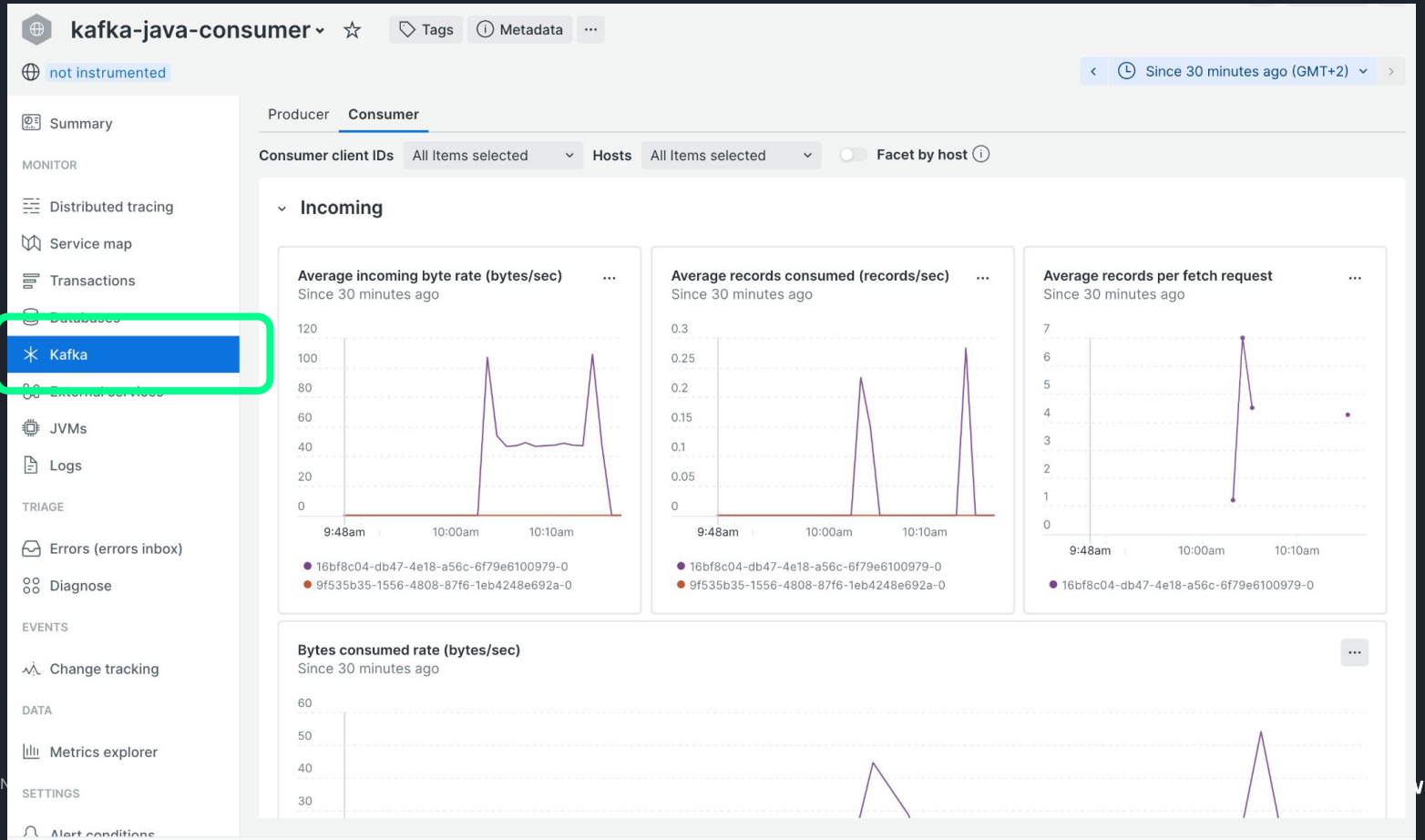
```java
private void ExecuteLongrunningTask(Integer secondsToSleep) {
    Span span = tracer.spanBuilder(spanName:"ExecuteLongrunningTask").startSpan();
    //Make the span the current span
    try {
        Pieces: Comment | Pieces: Explain
        Span sleepSpan = tracer.spanBuilder(spanName:"WhyTheHeckDoWeSleepHere")
                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

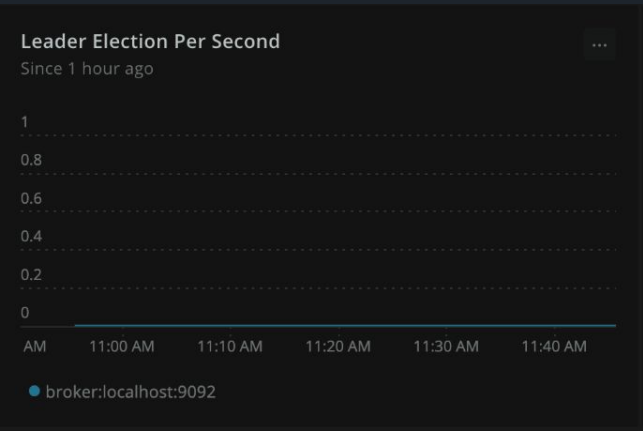
Kafka metrics for producer

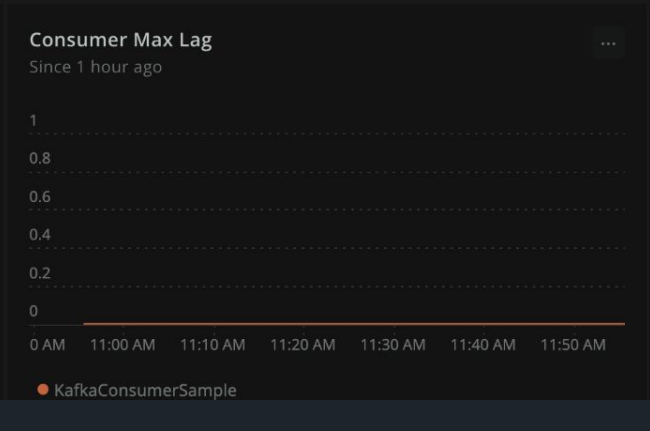


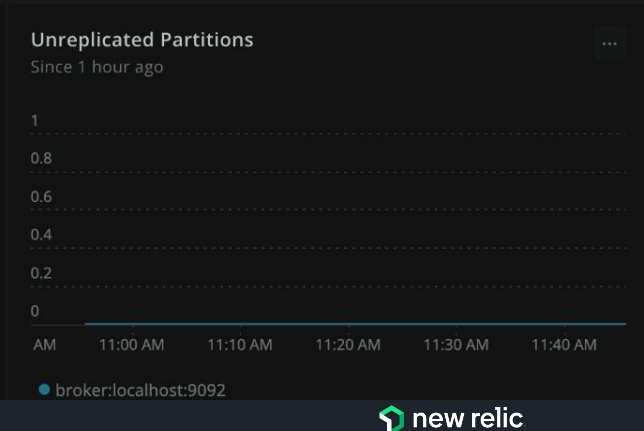
Kafka metrics for consumer



Kafka infrastructure metrics







new relic

Java supported libraries in OpenTelemetry

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

- Disabled instrumentations

Libraries / Frameworks

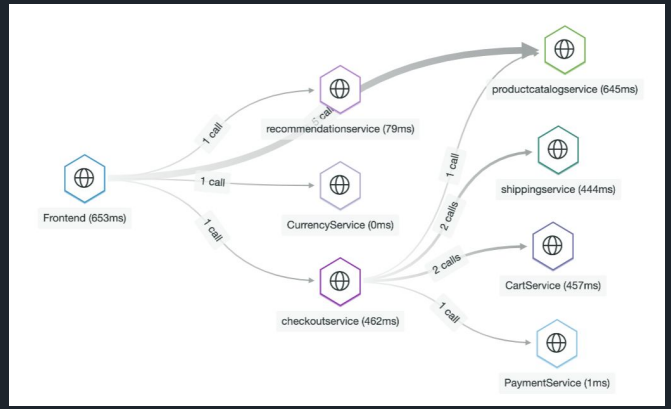
These are the supported libraries and frameworks:

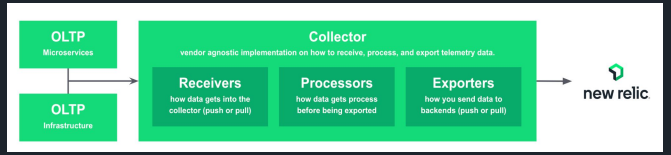
Library/Framework	Auto-instrumented versions	Standalone Library Instrumentation [1]	Semantic Conventions
Akka Actors	2.3+	N/A	Context propagation
Akka HTTP	10.0+	N/A	HTTP Client Spans, HTTP Client Metrics, HTTP Server Spans, HTTP Server Metrics, Provides `http.route` [2]
Alibaba Druid	1.0+	opentelemetry-alibaba-druid-1.0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

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

Reading List

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

Sign Up for Confluent Cloud

Get $400 worth free credits for your first 30 Days

Use Promo Code - POPTOUT000MZG62 to skip the paywall!

Harry Kimpel

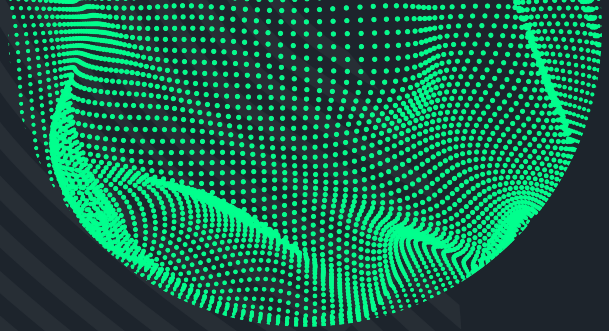
Principal Developer Relations Engineer

New Relic

@harrykimpel

bitly

new relic

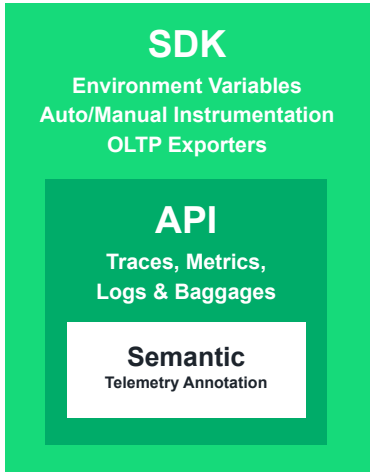


Appendix

some more information

new relic

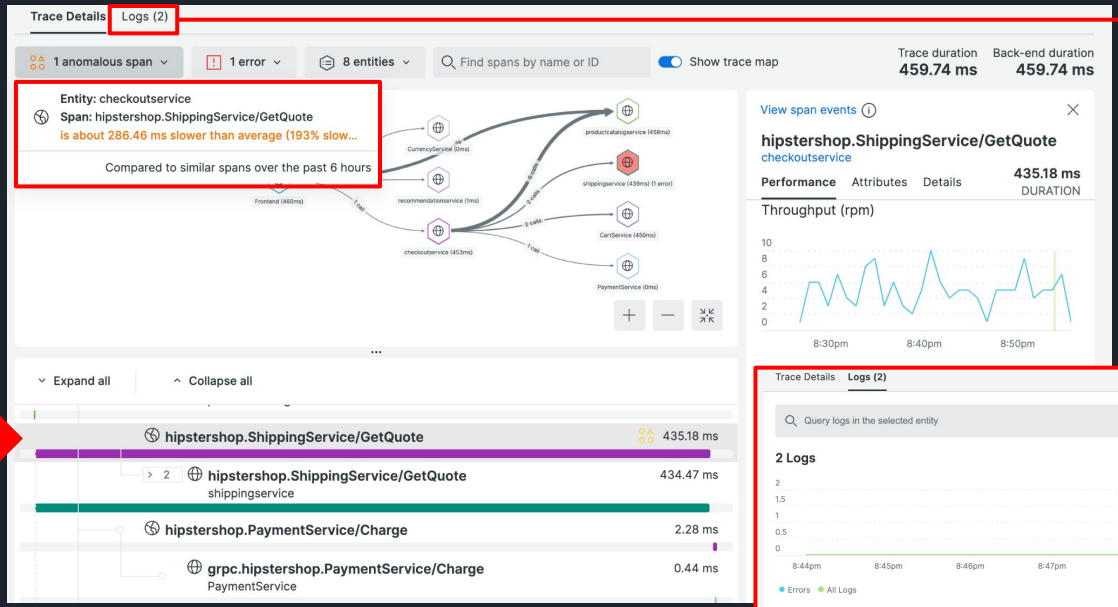
OpenTelemetry - Instrumentation



Core Concepts on Instrumentation

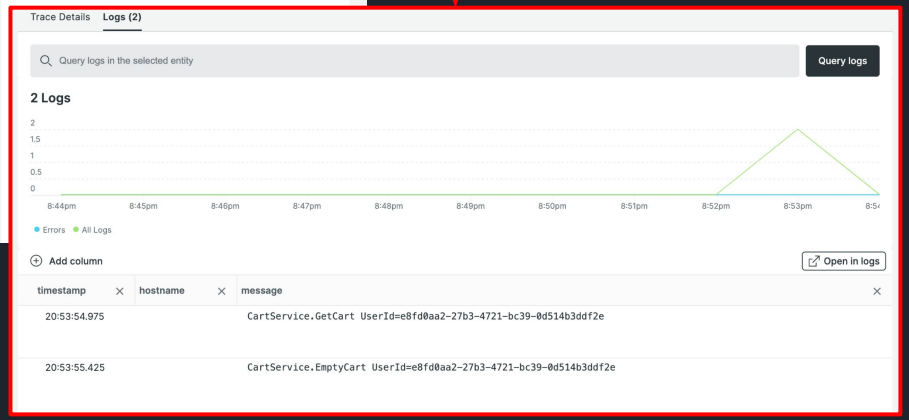
- **Semantic Conventions** - annotate telemetry with attributes specific to the represented operation, such as HTTP calls.
- **API** - standard way to collect instrumentation data.
- **SDK** - language-specific implementation of the API.
    - SDKs incorporate automatic instrumentation for common libraries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



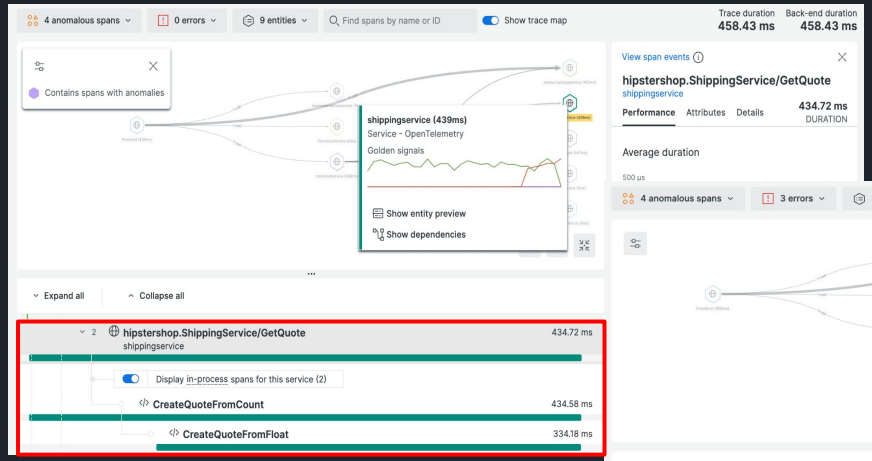
Anomalous Span Detection in Distributed Tracing for OpenTelemetry

Logs in Context (correlate to the right trace)

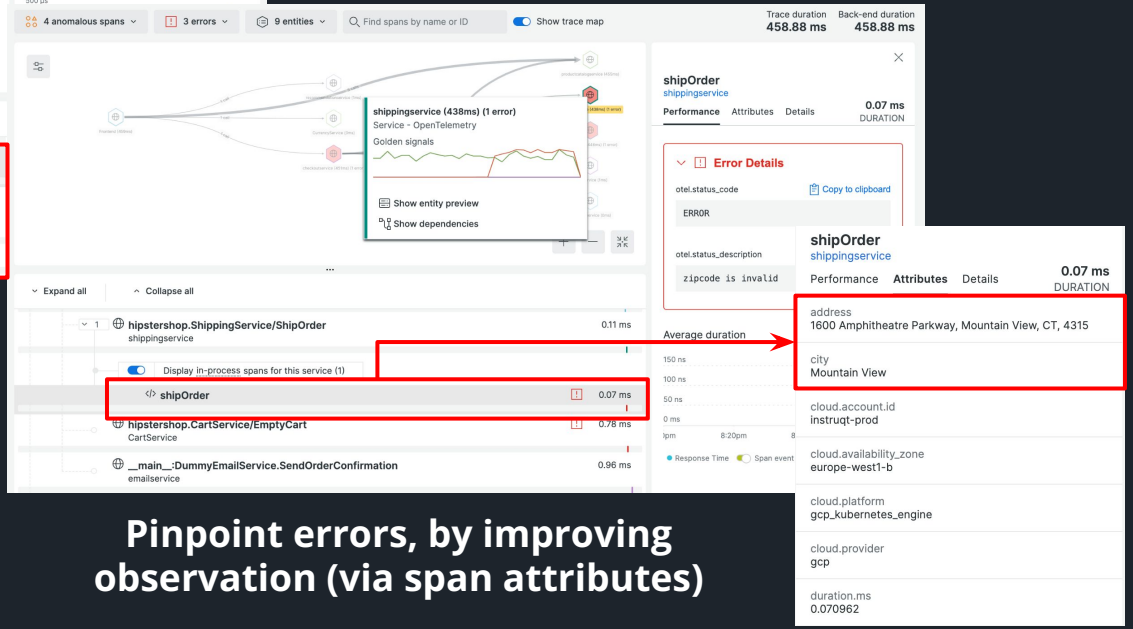


new relic

Instrumentation in Action



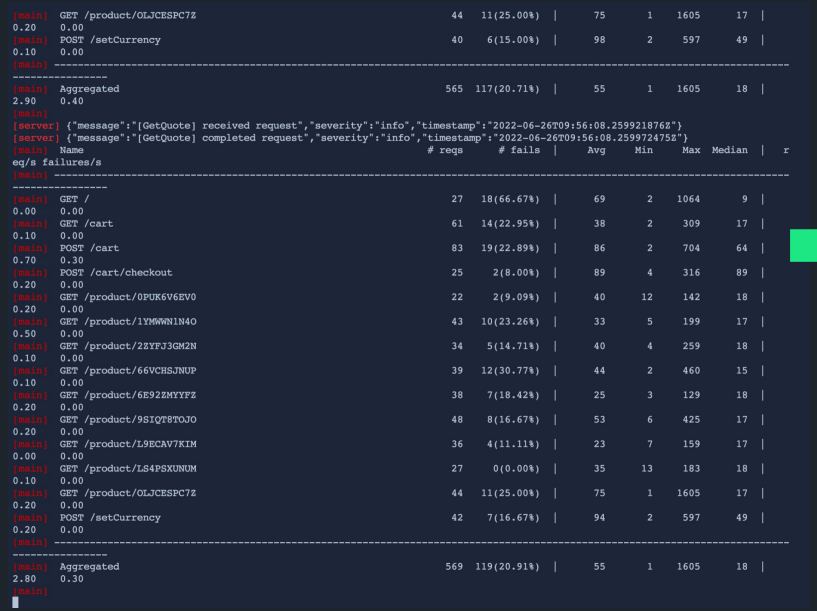
Get deeper into the app stack, by building spans.



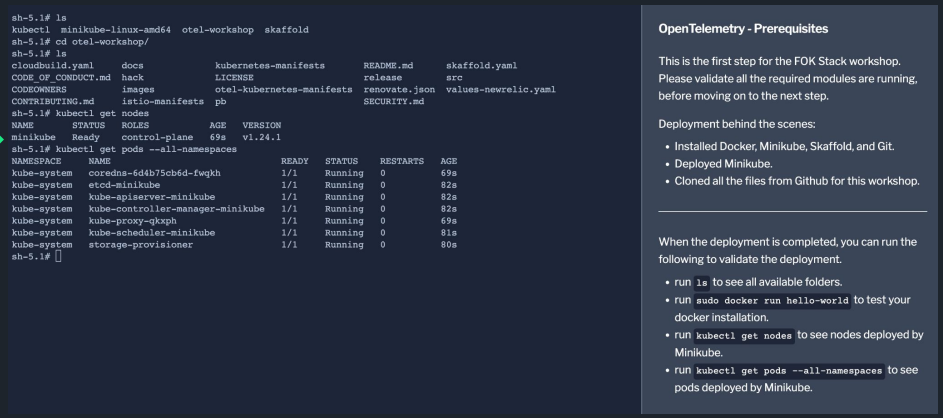
Pinpoint errors, by improving observation (via span attributes)

new relic

OpenTelemetry in Action



Deploy the app, and generate load (successful deployment)



Validate K8s is running, and follow the instructions

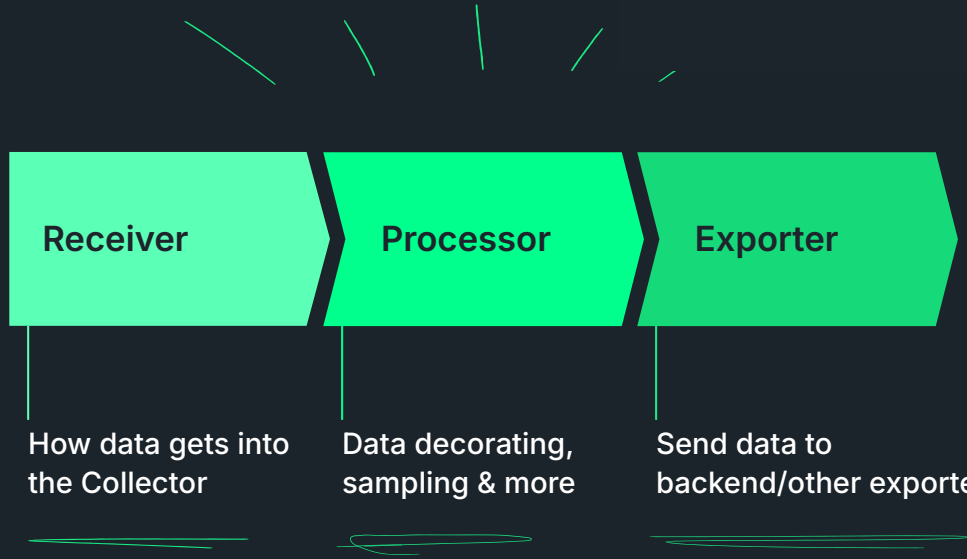
Important - deploy might take up to 10 mins

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

new relic

Collector Component

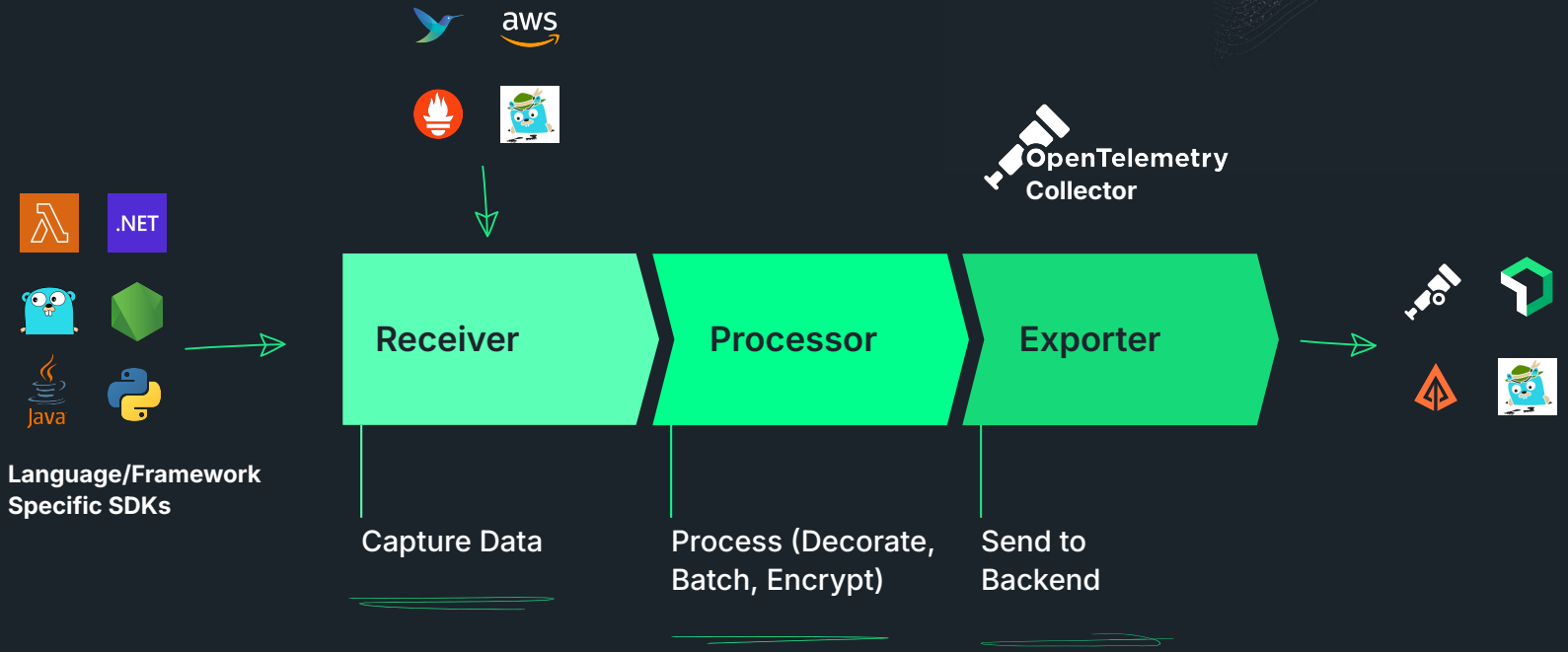
OpenTelemetry



new relic

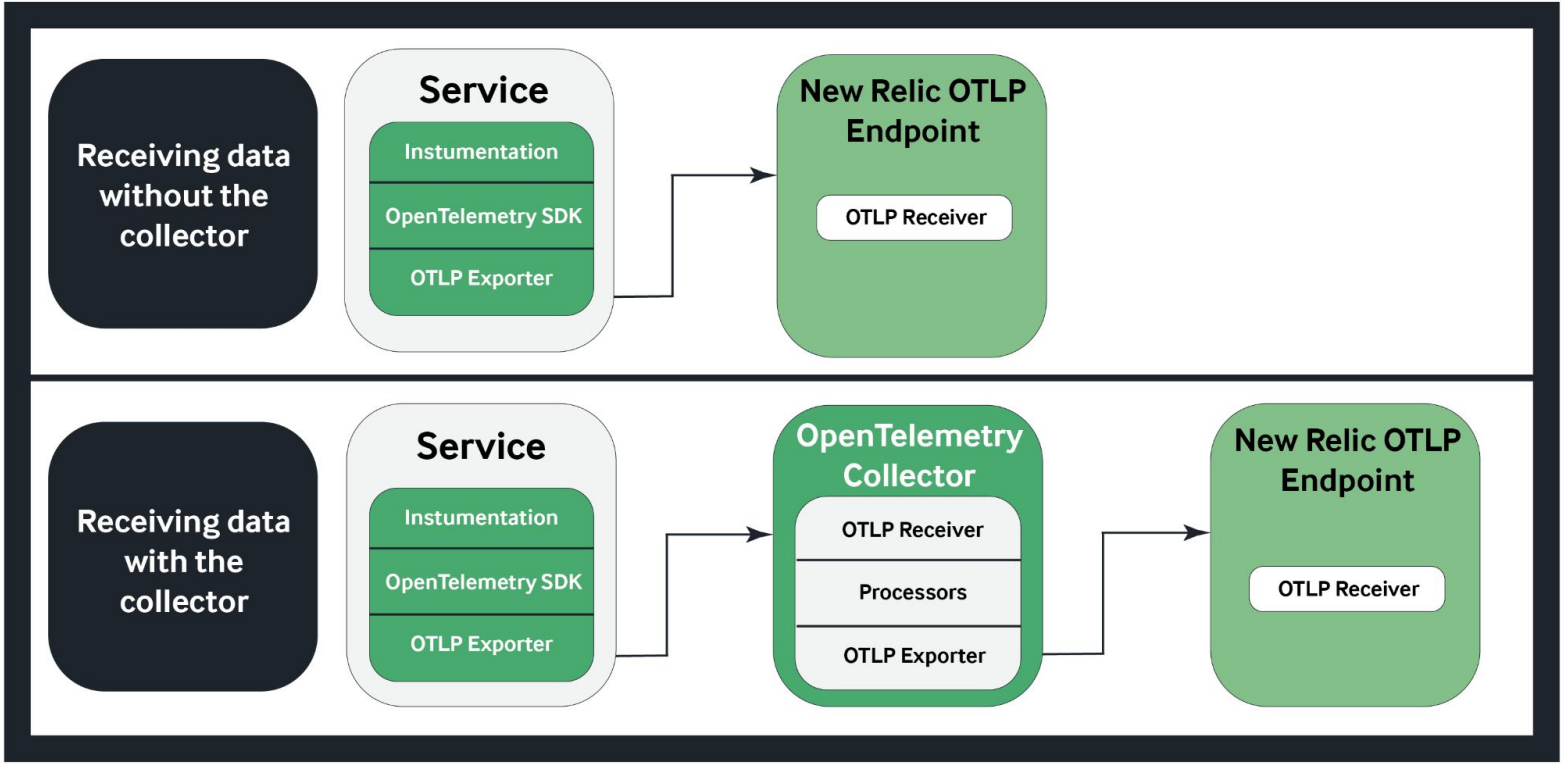
Putting it All Together

OpenTelemetry



new relic

OpenTelemetry - Collector



new relic

OpenTelemetry - Collector

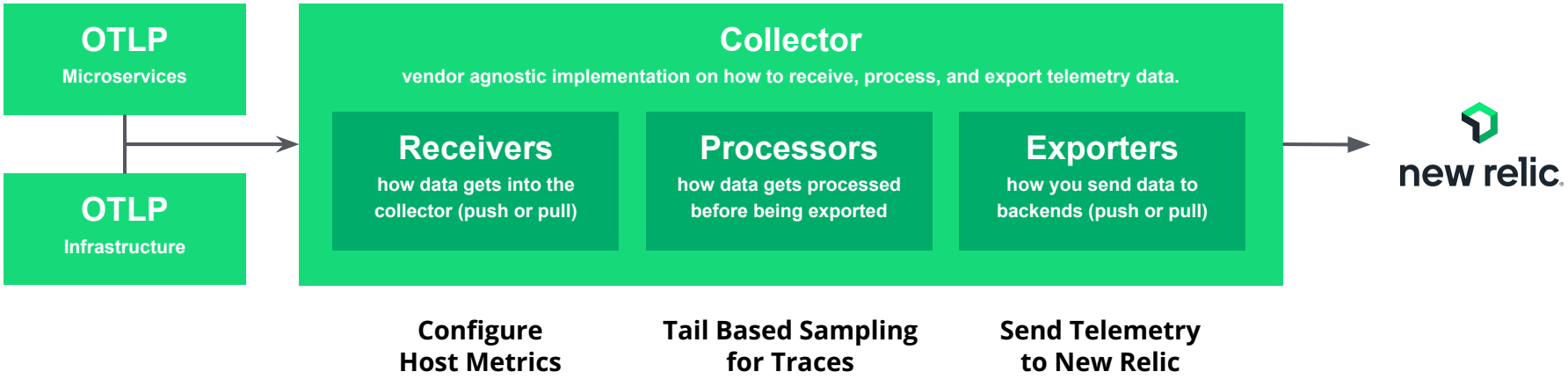
Benefits

- 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

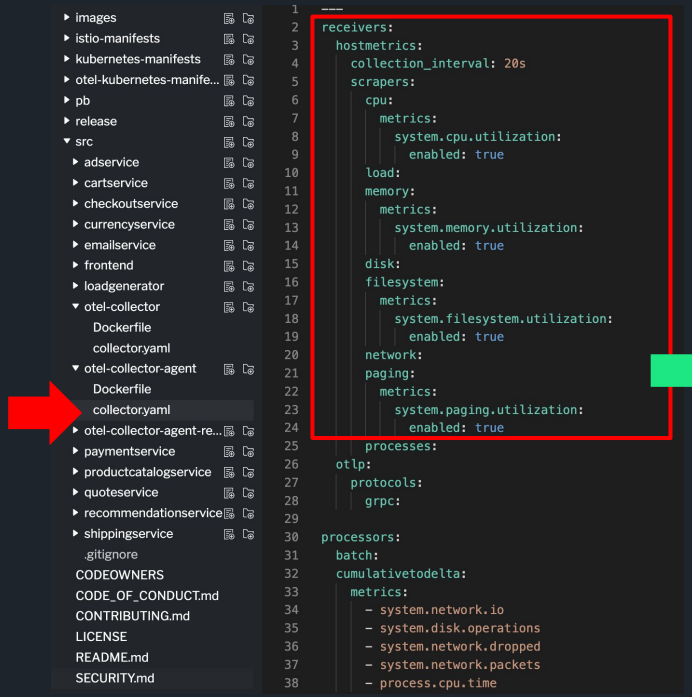
Downsides - nontrivial!

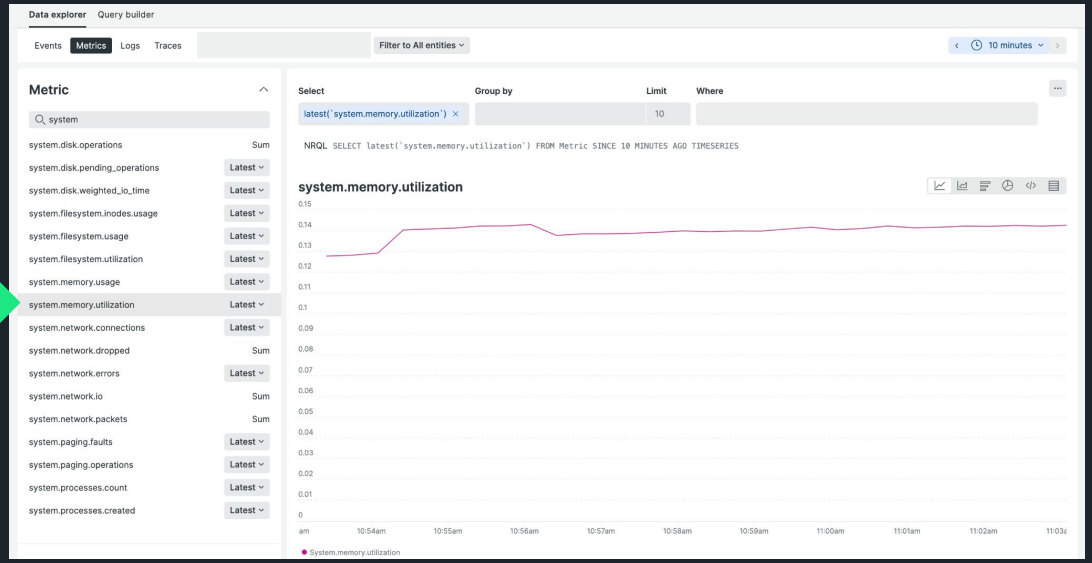
- Complex to scale
  - Deployment patterns
  - Load balancing
- Monitoring the collector is only available with ... the collector itself (at this time)
- Current stability status is “mixed”

OpenTelemetry - Collector



Collector in Action - Host Metrics



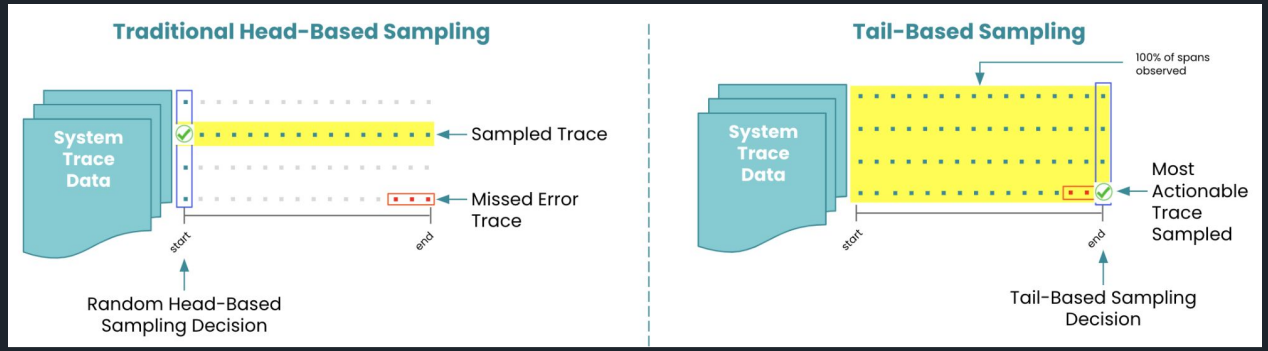


Collect additional infrastructure metrics for OpenTelemtry

new relic

HEAD vs TAIL Sampling

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

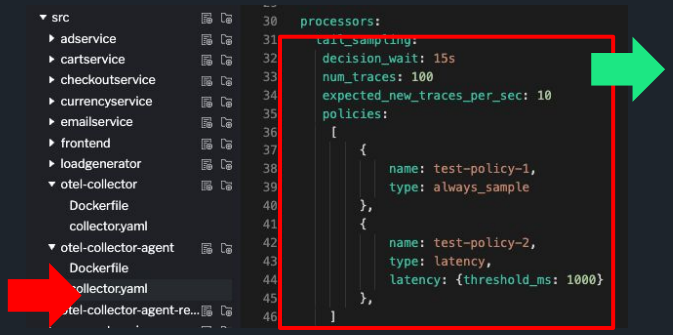


Head-based sampling works well for an overall statistical sampling of requests through a distributed system.

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