

# 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."



# Who uses Kafka?



# new relic<sup>®</sup> uses Kafka!

 more than **15 million**  
messages per second

 aggregate data rate  
approaching **1 Tbps**

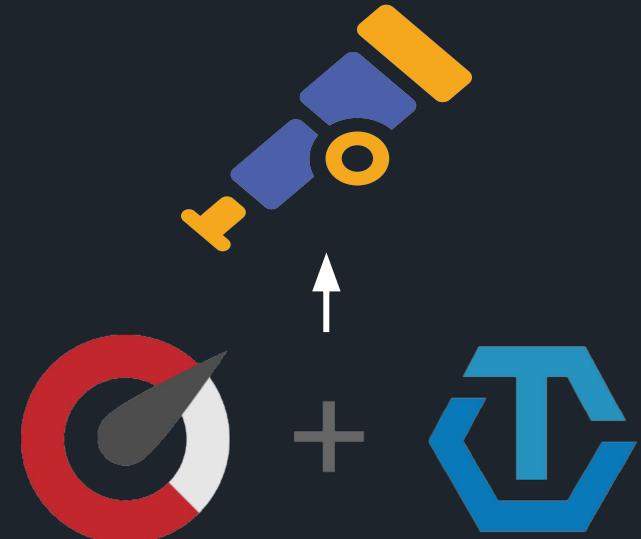
 **Hands up**  
**if you have heard**  
**about**  
**OpenTelemetry**



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



# The Rise of OpenTelemetry



## Vendor Neutral

Provides flexibility to change backend



## Interoperable

End-to-end visibility with standard instrumentation



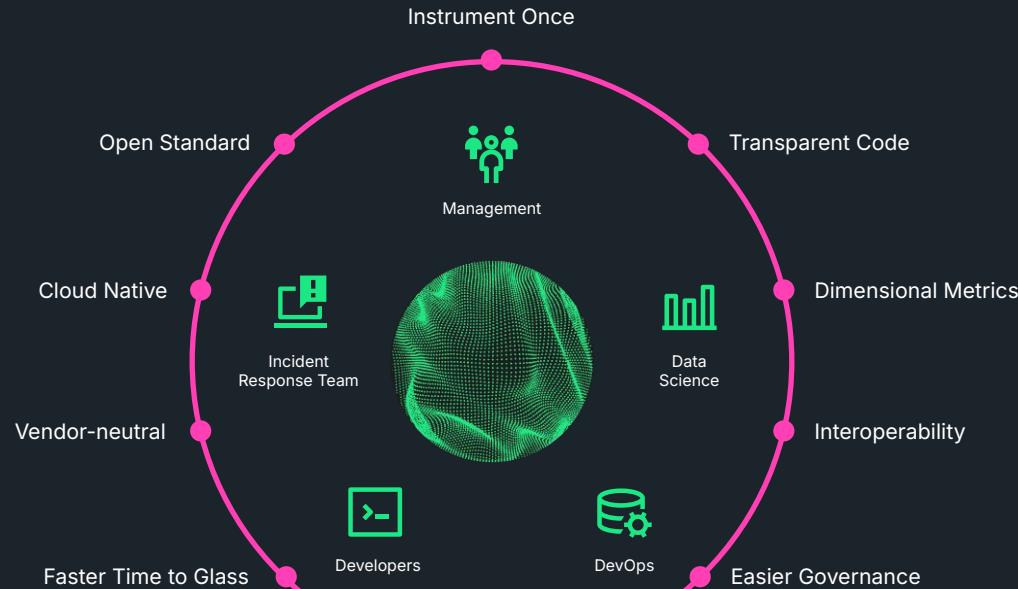
## Configurable

Pick and choose from the pieces what is needed

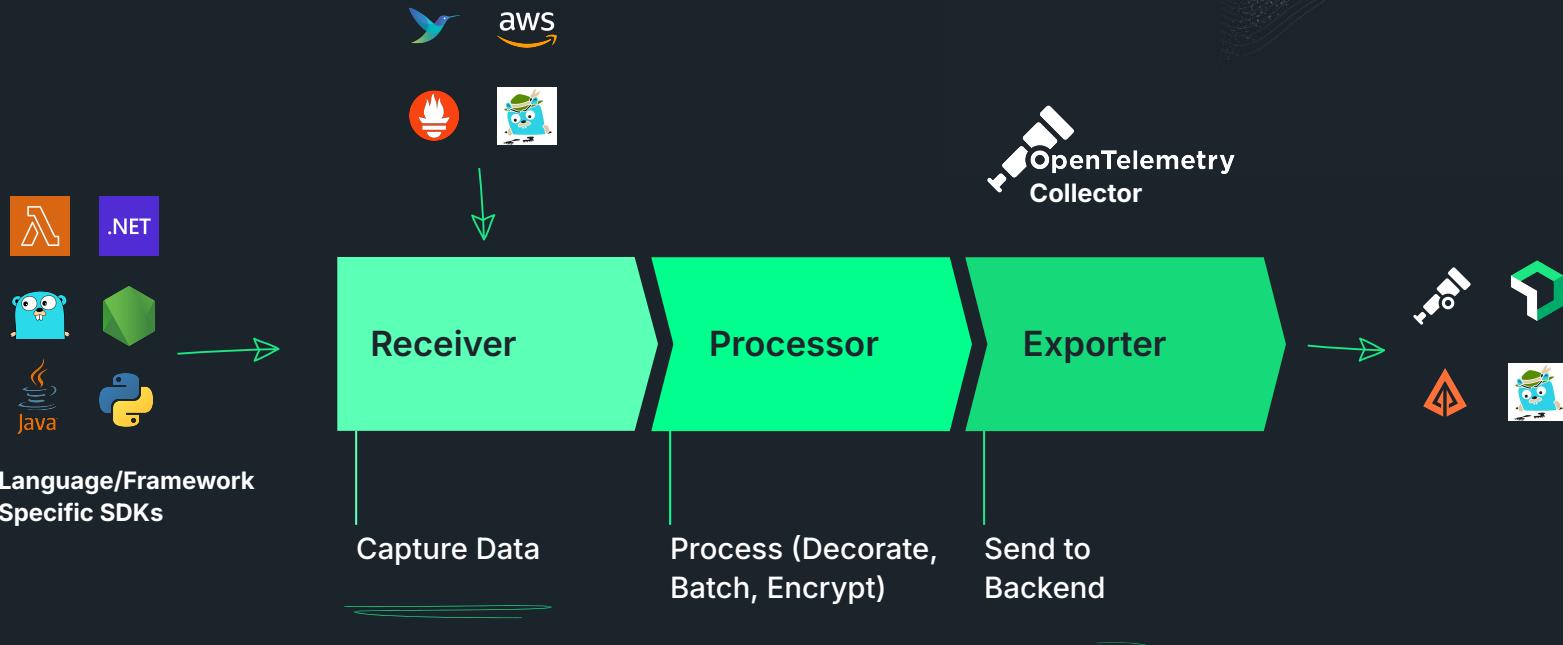
By 2025, **70%** of new cloud-native application monitoring will use open-source instrumentation

<sup>(1)</sup> Source: Gartner Magic Quadrant 2021 for Application Performance Monitoring - [link](#)

# Why OpenTelemetry?



# Snapshot of OpenTelemetry

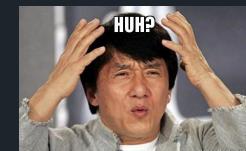
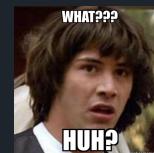


# The Rise of OpenTelemetry



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

**BUT WAIT ...**



# Design Considerations & Guidelines

**Customize  
Everything**  
(manual instrumentation)

Ingredients

Custom Made

Flavours

**Java, .NET  
and others  
mature**



**End  
Product**  
(auto instrumentation)

Ready to Eat

Packaged

Scalable

**Most  
Languages**



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

## Status and Releases

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

Language	Traces	Metrics	Logs
<a href="#">C++</a>	Stable	Stable	Stable
<a href="#">C#/NET</a>	Stable	Stable	Stable
<a href="#">Erlang/Elixir</a>	Stable	Development	Development
<a href="#">Go</a>	Stable	Stable	Beta
<a href="#">Java</a>	Stable	Stable	Stable
<a href="#">JavaScript</a>	Stable	Stable	Development
<a href="#">PHP</a>	Stable	Stable	Stable
<a href="#">Python</a>	Stable	Stable	Development
<a href="#">Ruby</a>	Stable	Development	Development
<a href="#">Rust</a>	Beta	Beta	Beta
<a href="#">Swift</a>	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](#)

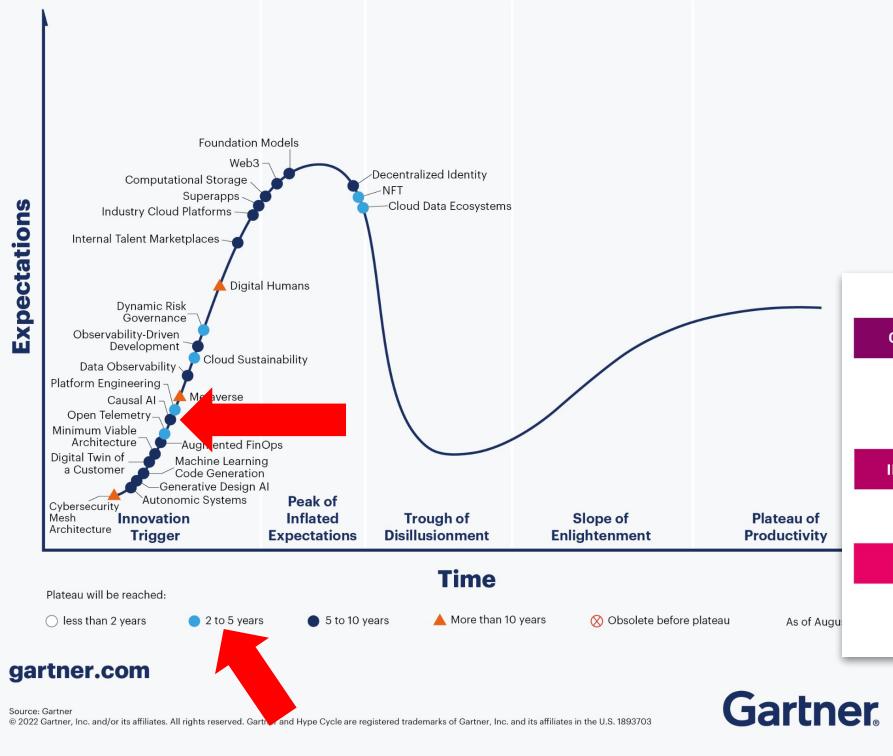
## Libraries / Frameworks

These are the supported libraries and frameworks:

Library/Framework	Auto-instrumented versions	Standalone Library Instrumentation [1]	Semantic Conventions
<a href="#">Akka Actors</a>	2.3+	N/A	Context propagation
<a href="#">Akka HTTP</a>	10.0+	N/A	<a href="#">HTTP Client Spans</a> , <a href="#">HTTP Client Metrics</a> , <a href="#">HTTP Server Spans</a> , <a href="#">HTTP Server Metrics</a> , Provides <code>http.route</code> [2]
<a href="#">Alibaba Druid</a>	1.0+	<a href="#">opentelemetry-alibaba-druid-1.0</a>	<a href="#">Database Pool Metrics</a>
<a href="#">Apache Axis2</a>	1.6+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
<a href="#">Apache Camel</a>	2.20+ (not including 3.x yet)	N/A	Dependent on components in use
<a href="#">Apache CXF JAX-RS</a>	3.2+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
<a href="#">Apache CXF JAX-WS</a>	3.0+	N/A	Provides <code>http.route</code> [2], Controller Spans [3]
<a href="#">Apache DBCP</a>	2.0+	<a href="#">opentelemetry-apache-dbc-2.0</a>	<a href="#">Database Pool Metrics</a>
<a href="#">Apache Dubbo</a>	2.7+	<a href="#">opentelemetry-apache-dubbo-2.7</a>	<a href="#">RPC Client Spans</a> , <a href="#">RPC Server Spans</a>
<a href="#">Apache HttpAsyncClient</a>	4.1+	N/A	<a href="#">HTTP Client Spans</a> , <a href="#">HTTP Client Metrics</a>
<a href="#">Apache HttpClient</a>	2.0+	<a href="#">opentelemetry-apache-httpclient-4.3</a> , <a href="#">opentelemetry-apache-httpclient-5.2</a>	<a href="#">HTTP Client Spans</a> , <a href="#">HTTP Client Metrics</a>
<a href="#">Apache Kafka Producer/Consumer API</a>	0.11+	<a href="#">opentelemetry-kafka-clients-2.6</a>	<a href="#">Messaging Spans</a>
<a href="#">Apache Kafka Streams API</a>	0.11+	N/A	<a href="#">Messaging Spans</a>
<a href="#">Apache MyFaces</a>	1.2+ (not including 3.x yet)	N/A	Provides <code>http.route</code> [2], Controller Spans [3]

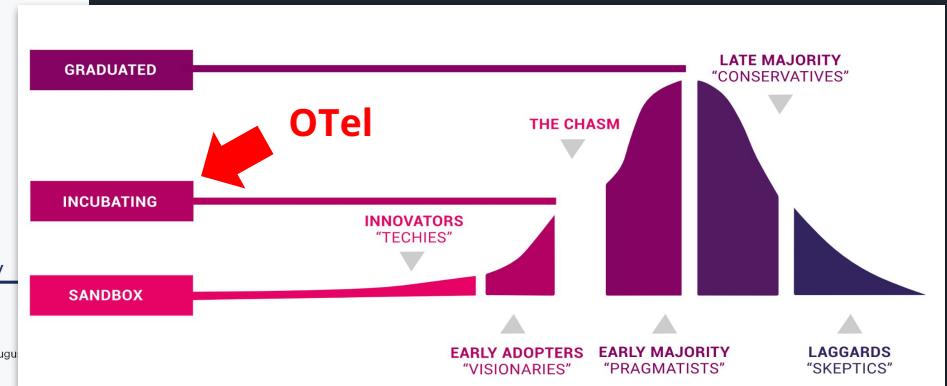
# Design Considerations & Guidelines

## Hype Cycle for Emerging Tech, 2022



## Top 6 Questions To Consider

- Do you understand the difference between Monitoring vs Observability?
- Do you understand the basics of Metrics, Logs & Traces?
- Is tracing widely deployed to provide value?
- Do you have experience in manual instrumentation?
- Is your app, friendly to manual instrumentation?
- Is your environment or practice tolerant to change?



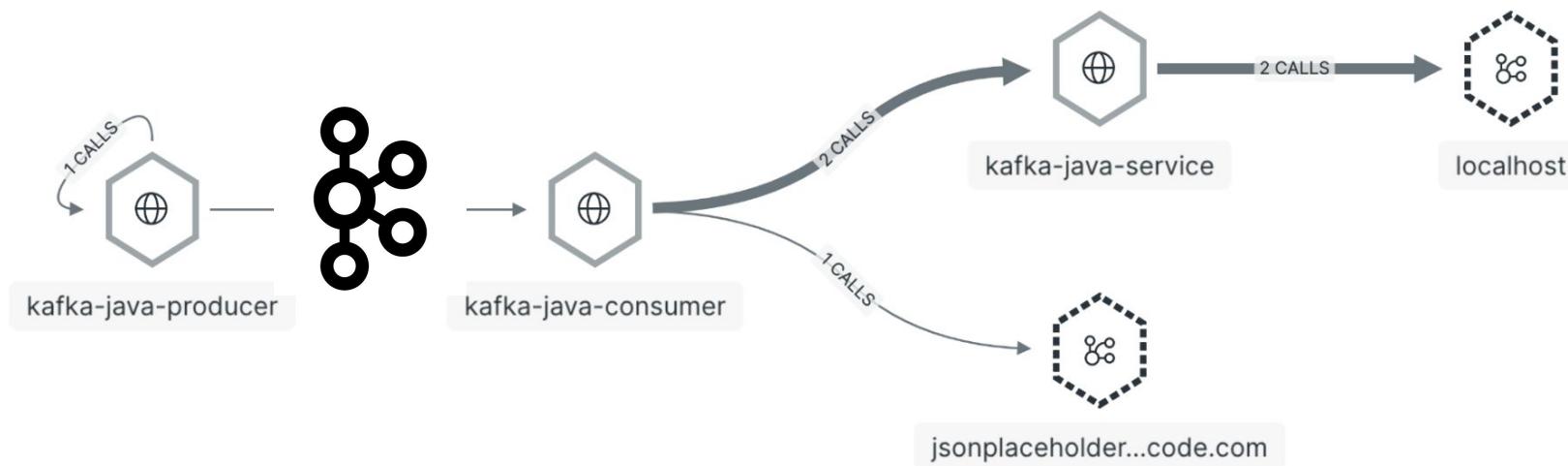
Gartner

new relic

# Automatic instrumentation with Java

## Entity map i

There are 5 sampled entities in this trace



# Automatic instrumentation - aka zero-code instrumentation - with Java

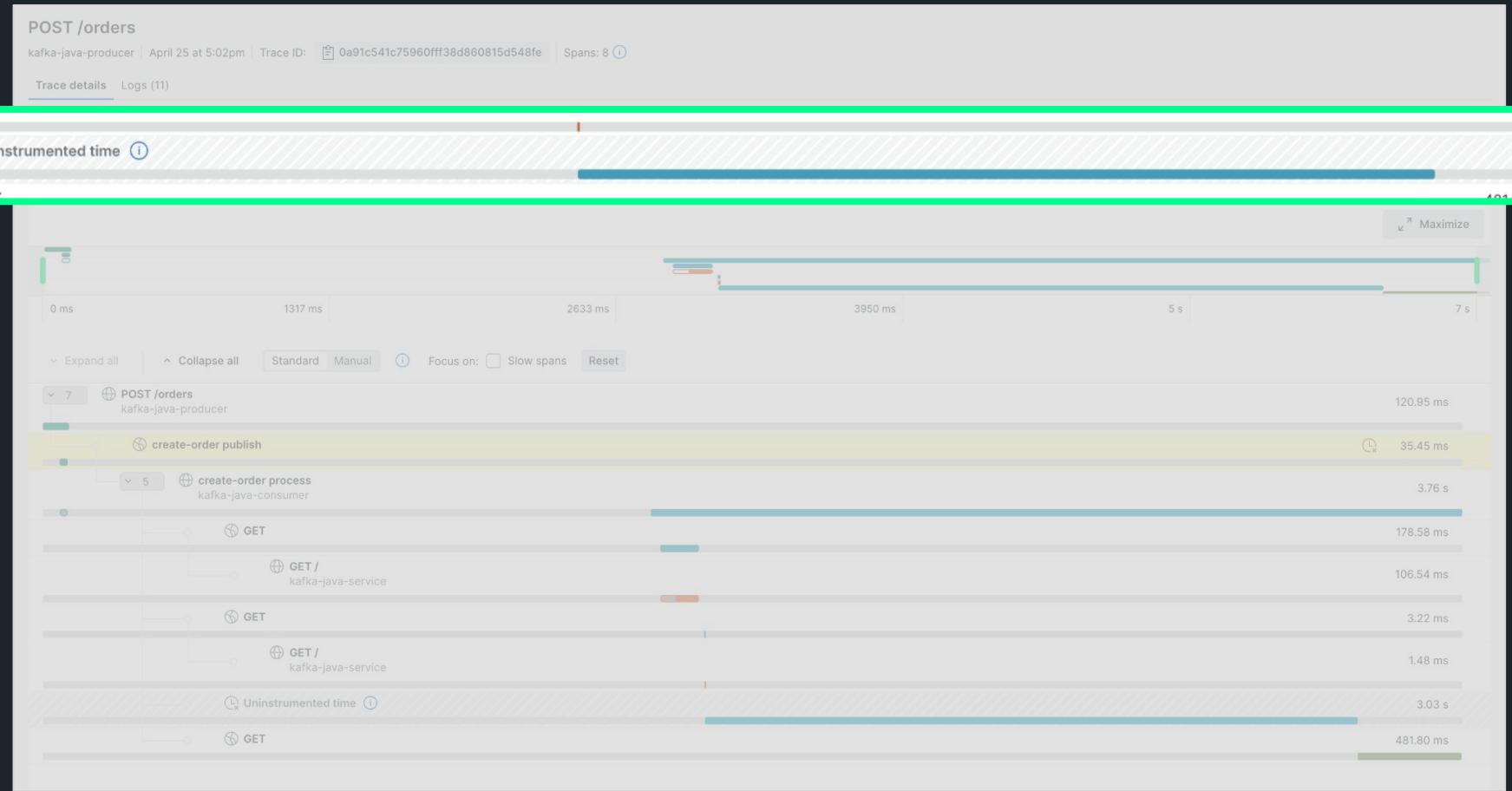
<https://opentelemetry.io/docs/zero-code/>



# 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
```

# Automatic instrumentation with Java



# 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, .value:"kafka-java-producer")
44  +-->     .put(ResourceAttributes.SERVICE_VERSION, .value:"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

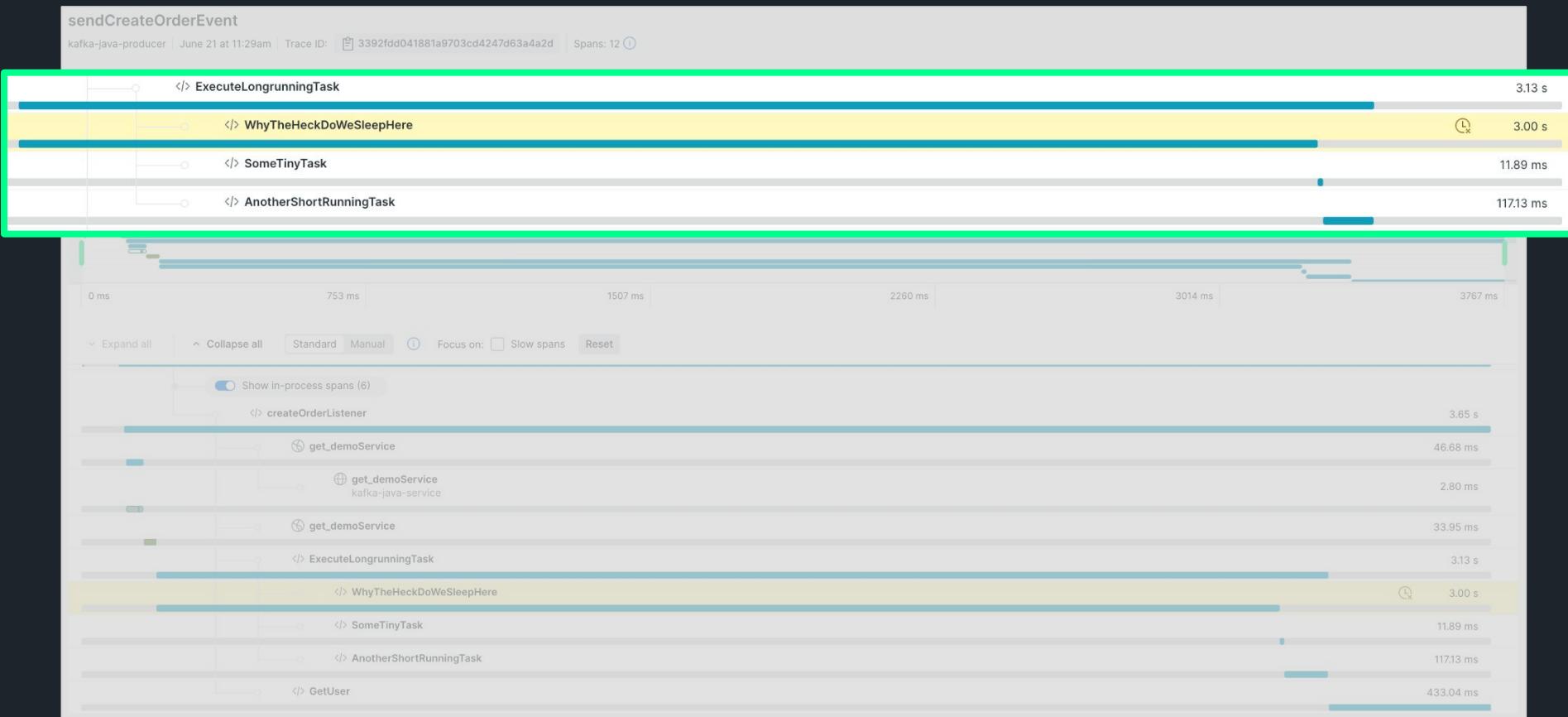
```
Pieces: Comment | Pieces: Explain
22  ...
23  @Bean
24  public <K, V> ProducerFactory<K, V> createOrderProducerFactory() {
25      Map<String, Object> config = new HashMap<>();
26      config.put(
27          org.apache.kafka.clients.producer.ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
28        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

# Manual instrumentation with Java

## - Kafka consumer

```
Pieces: Comment | Pieces: Explain
35     ...@Bean("orderConsumerFactoryNotificationService")
36     public ConsumerFactory<String, Order> createOrderConsumerFactory() {
37         Map<String, Object> props = new HashMap<>();
38         props.put(
39             ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
40             TracingConsumerInterceptor.class.getName());
41         props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
42         props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
43         props.put(ConsumerConfig.CLIENT_ID_CONFIG, UUID.randomUUID().toString());
44         props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, value: StringSerializer.class);
45         props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, value: JsonSerializer.class);
46         props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, value: false);
47
48         return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
49             new JsonDeserializer<>(targetType: Order.class));
50     }
```

# Manual instrumentation with Java

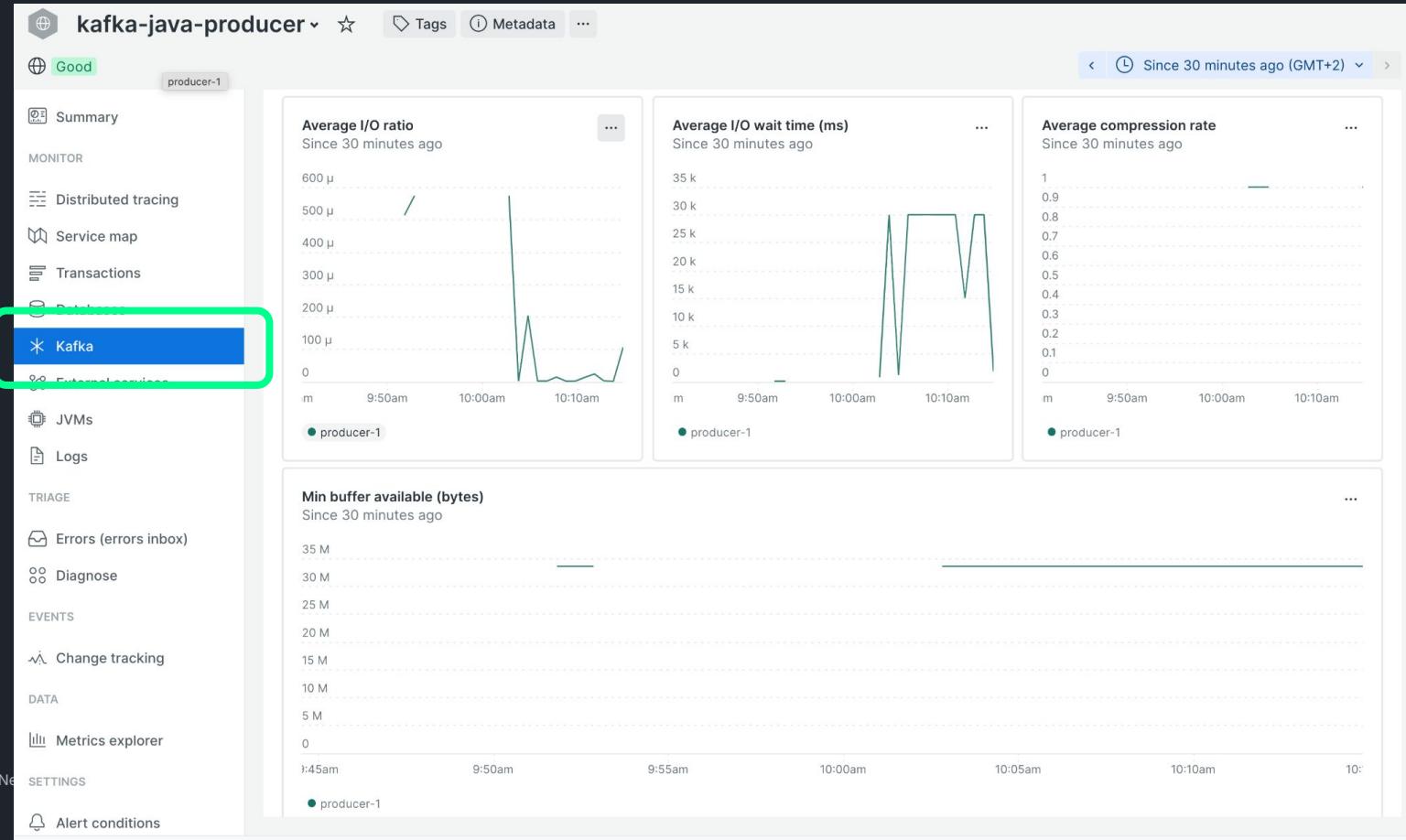


# Manual instrumentation with Java

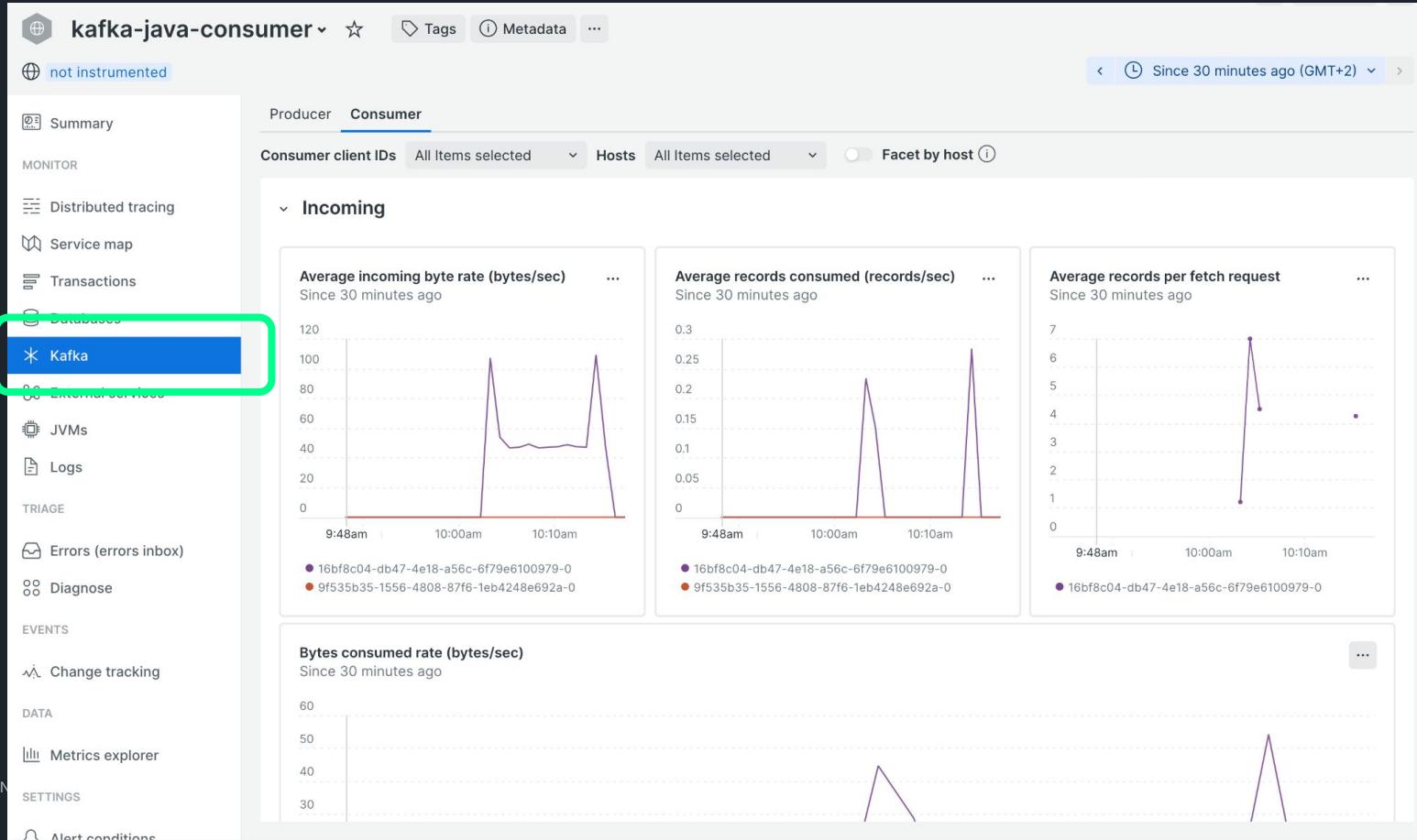
## - custom spans

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

# Kafka metrics for producer



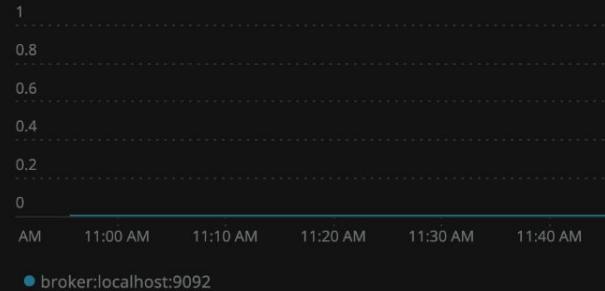
# Kafka metrics for consumer



# Kafka infrastructure metrics

## Leader Election Per Second

Since 1 hour ago



## Unclean Leader Election Per Second

Since 1 hour ago



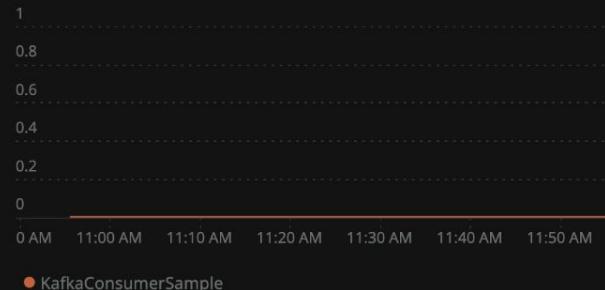
## Topic Bytes Written

Since 1 hour ago



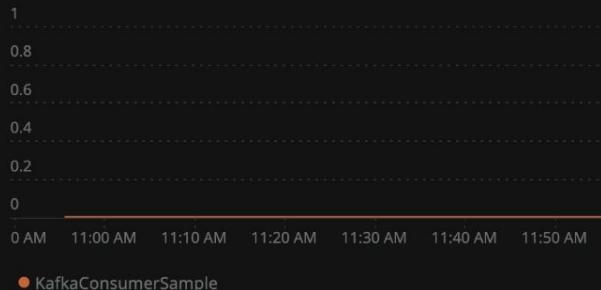
## Consumer Minimum Requests Per Second

Since 1 hour ago



## Consumer Max Lag

Since 1 hour ago



## Unreplicated Partitions

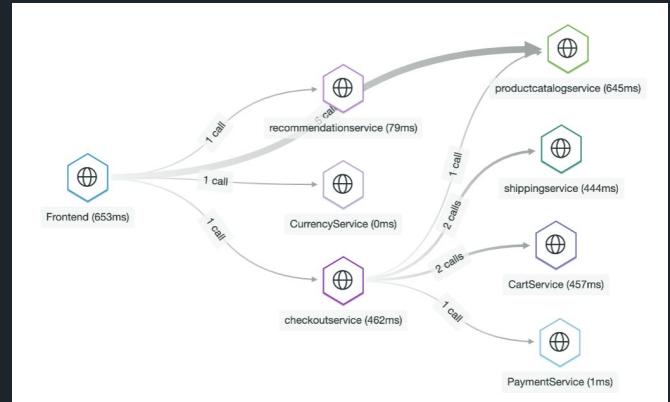
Since 1 hour ago



# Summary

- Exciting times for open source observability!  
**Be mindful about the maturity status**, and plan ahead on the adoption of OpenTelemetry.
- The collector is a very useful tool that also comes with challenges, especially scalability.
- Instrumentation can include traces, logs, and/or metrics to improve observations.
- New Relic is actively investing in OpenTelemetry, and helping engineers do their best work based on **data, not opinions**.

The Stack



# Reading Materials for OpenTelemetry

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

## Reading List

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

# Sign Up for Confluent Cloud

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

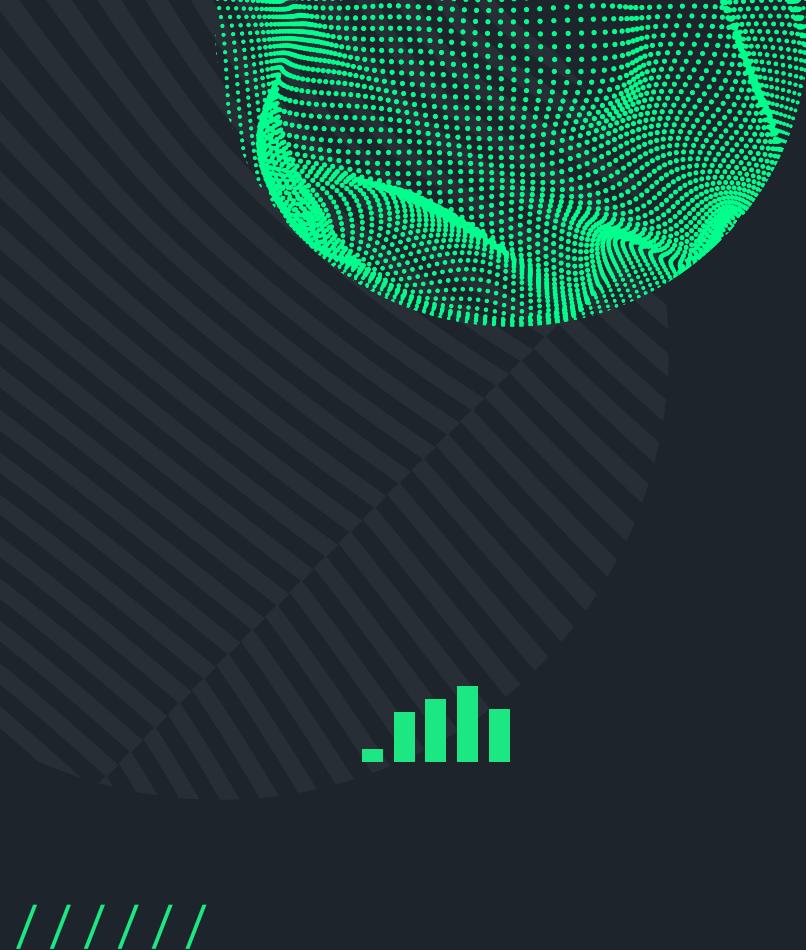
**Use Promo Code - POPTOUT000MZG62  
to skip the paywall!**





**Harry Kimpel**  
Principal Developer Relations Engineer  
**New Relic**  
**@harrykimpel**





# Appendix

## some more information

# Kafka Best Practices

[https://newrelic.com/blog/best-practices/  
kafka-consumer-config-auto-commit-data-loss](https://newrelic.com/blog/best-practices/kafka-consumer-config-auto-commit-data-loss)



# Kafka Best Practices - Partitioning 1/2

**Understand the data rate of your partitions to ensure you have the correct retention space.**

Number of messages



\*

Avg. message size



= Data retention space

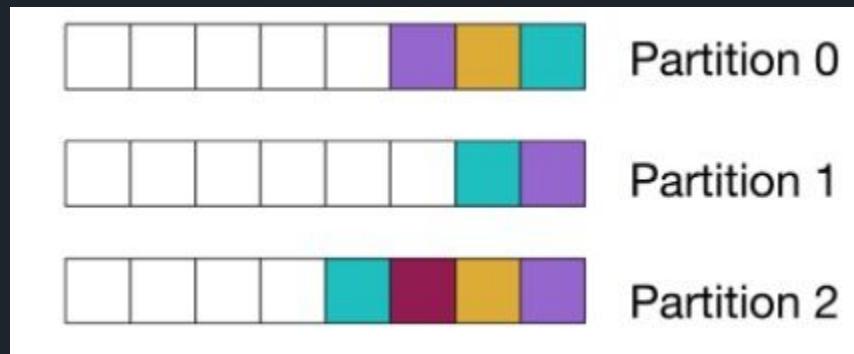


minimum performance, a single consumer needs to support, without lagging



# Kafka Best Practices - Partitioning 2/2

**Unless you have architectural needs that require you to do otherwise, use random partitioning when writing to topics.**



- Evenest spread of load for consumers
- Thus makes scaling the consumers easier
- Particularly suited for stateless or “embarrassingly parallel” services
- Using the default partitioner while not manually specifying a partition or a message key

# Kafka Best Practices - Consumers 1/4

**If your consumers are running versions of Kafka older than 0.10, upgrade them.**

- In version 0.8.x, consumers use Apache ZooKeeper for consumer group coordination
- a number of known bugs can result in long-running rebalances or even failures of the rebalance algorithm, aka "rebalance storms"
- During a rebalance, one or more partitions are assigned to each consumer in the consumer group
- In a rebalance storm, partition ownership is continually shuffled among the consumers, preventing any consumer from making real progress on consumption

# Kafka Best Practices - Consumers 2/4

## Tune your consumer socket buffers for high-speed ingest

- In Kafka 0.10.x, the parameter is `receive.buffer.bytes`, defaults to 64kB
- In Kafka 0.8.x, the parameter is `socket.receive.buffer.bytes`, defaults to 100kB
- Default values are **too small** for high-throughput environments
- Particularly if the network's bandwidth-delay product between the broker and the consumer is larger than a local area network (LAN)
- For high-bandwidth networks (10 Gbps or higher) with latencies of 1 millisecond or more, consider setting the `socket buffers` to 8 or 16 MB
- If memory is scarce, consider 1 MB
- You can also use a value of `-1`, which lets the underlying operating system tune the buffer size based on network conditions
  - However, the automatic tuning might not occur fast enough for consumers that need to start "hot."

# Kafka Best Practices - Consumers 3/4

## Design high-throughput consumers to implement back-pressure when warranted

- Better to consume only what you can process efficiently
  - rather than it is to consume so much that your process grinds to a halt and then drops out of the consumer group
- Consumers should consume into fixed-sized buffers (see the [Disruptor pattern](#)), preferably off-heap if running in a Java virtual machine (JVM)
- A fixed-size buffer will prevent a consumer from pulling so much data onto the heap that the JVM spends all of its time performing garbage collection instead of the work you want to achieve - which is processing messages.

Disruptor pattern: <https://lmax-exchange.github.io/disruptor/files/Disruptor-1.0.pdf>

# Kafka Best Practices - Consumers 4/4

**When running consumers on a JVM, be wary of the impact that garbage collection can have on your consumers**

- For example, long garbage collection pauses can result in dropped ZooKeeper sessions or consumer-group rebalances
- The same is true for brokers, which risk dropping out of the cluster if garbage collection pauses are too long

# Kafka Best Practices - Producer 1/4

## Configure your producer to wait for acknowledgments

- this is how the producer knows that the message has actually made it to the partition on the broker
- In Kafka 0.10.x, the setting is `acks`; in 0.8.x, it's `request.required.acks`
- Kafka provides fault-tolerance via replication so the failure of a single node or a change in partition leadership does not affect availability
- If you configure your producers without `acks` (otherwise known as "fire and forget"), messages can be silently lost

# Kafka Best Practices - Producer 2/4

## Configure `retries` on your producers

- The default value is `3`, which is often too low
- The right value will depend on your application
  - For applications where data-loss cannot be tolerated, consider `Integer.MAX_VALUE` (effectively, infinity)
  - This guards against situations where the broker leading the partition isn't able to respond to a produce request right away

# Kafka Best Practices - Producer 3/4

## For high-throughput producers, tune buffer sizes

- Particularly `buffer.memory` and `batch.size` (which is counted in bytes)
- Because `batch.size` is a per-partition setting, producer performance and memory usage can be correlated with the number of partitions in the topic
- The values depend on several factors:
  - Producer data rate (both the size and number of messages)
  - Number of partitions you are producing to
  - Amount of memory you have available
- Keep in mind that larger buffers are not always better
  - because if the producer stalls for some reason (say, one leader is slower to respond with acknowledgments), having more data buffered on-heap could result in more garbage collection

# Kafka Best Practices - Producer 4/4

## Instrument your application to track metrics

- Such as
  - Number of produced messages
  - Average produced message size
  - Number of consumed messages
  - ...

# Kafka Best Practices - Broker 1/10

## Compacted topics require memory and CPU resources on your brokers

- Log compaction needs both heap (memory) and CPU cycles on the brokers to complete successfully
- Failed log compaction puts brokers at risk from a partition that grows unbounded
- Tune `log.cleaner.dedupe.buffer.size` and `log.cleaner.threads` on your brokers
  - Keep in mind, these values affect heap usage on the brokers
- If broker throws an `OutOfMemoryError` exception, it will shut down and potentially lose data
- Buffer size and thread count will depend on both:
  - the number of topic partitions to be cleaned and
  - the data rate and key size of the messages in those partitions
- As of Kafka version 0.10.2.1, monitoring the log-cleaner log file for `ERROR` entries is the surest way to detect issues with log cleaner threads.

# Kafka Best Practices - Broker 2/10

## Monitor your brokers

- Network throughput
  - transmit (TX)
  - receive (RX)
- Disks
  - disk I/O
  - disk space
- CPU usage

Capacity planning is a key part of maintaining cluster performance

# Kafka Best Practices - Broker 3/10

## Distribute partition leadership among brokers in the cluster

- Leadership requires a lot of network I/O resources
- For example, when running with replication factor 3
  - a leader must receive the partition data
  - transmit two copies to replicas
  - plus transmit to however many consumers want to consume that data
  - So, in this example, being a leader is at least four times as expensive as being a follower in terms of network I/O used
- Leaders may also have to read from disk; followers only write.

# Kafka Best Practices - Broker 4/10

**Don't neglect to monitor your brokers for in-sync replica (ISR) shrinks, under-replicated partitions, and unpreferred leaders**

- These are signs of potential problems in your cluster
- For example
  - frequent ISR shrinks for a single partition
  - can indicate that the data rate for that partition exceeds the leader's ability to service the consumer and replica threads

# Kafka Best Practices - Broker 5/10

## Modify the Apache Log4j properties as needed

- Kafka broker logging can use an excessive amount of disk space
- However, don't forgo logging completely
- Broker logs can be the best - and sometimes only - way to reconstruct the sequence of events after an incident.

# Kafka Best Practices - Broker 6/10

**Either disable automatic topic creation or establish a clear policy regarding the cleanup of unused topics**

- For example
  - if no messages are seen for x days
  - consider the topic defunct and remove it from the cluster
  - this will avoid the creation of additional metadata within the cluster that you'll have to manage

# Kafka Best Practices - Broker 7/10

**For sustained, high-throughput brokers, provision sufficient memory to avoid reading from the disk subsystem**

- Partition data should be served directly from the operating system's file system cache whenever possible
- However, this means you'll have to ensure your consumers can keep up
  - a lagging consumer will force the broker to read from disk

# Kafka Best Practices - Broker 8/10

**For a large cluster with high-throughput service level objectives (SLOs), consider isolating topics to a subset of brokers**

- How you determine which topics to isolate will depend on the needs of your business
- For example
  - if you have multiple online transaction processing (OLTP) systems using the same cluster
  - isolating the topics for each system to distinct subsets of brokers
  - can help to limit the potential blast radius of an incident

# Kafka Best Practices - Broker 9/10

**Using older clients with newer topic message formats, and vice versa, places extra load on the brokers**

- ... as they convert the formats on behalf of the client
- Avoid this whenever possible

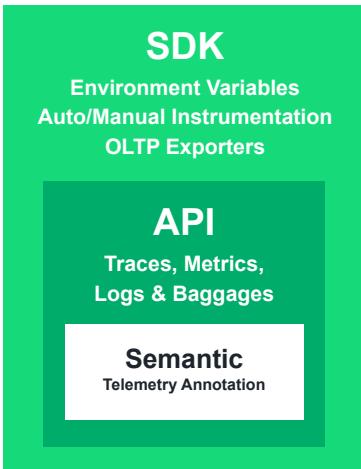
# Kafka Best Practices - Broker 10/10

**Don't assume that testing a broker on a local desktop machine is representative of the performance you'll see in production**

- Testing over a loopback interface to a partition using replication factor 1 is a very different topology from most production environments
- The network latency is negligible via the loopback and the time required to receive leader acknowledgements can vary greatly when there is no replication involved

# More information

# OpenTelemetry - Instrumentation



## Core Concepts on Instrumentation

- **Semantic Conventions** - annotate telemetry with attributes specific to the represented operation, such as HTTP calls.
- **API** - standard way to collect instrumentation data.
- **SDK** - language-specific implementation of the API.
  - SDKs incorporate automatic instrumentation for common libraries and frameworks for your application.
- **OpenTelemetry Protocol (OTLP)** - used to send data to your backend Observability platform of choice.
- **Specification ("spec")** - provides blueprints for all of the above to bring standardization across all languages.

# Instrumentation in Action

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

**Anomalous Span Detection:** A red box highlights a callout for an 'anomalous span' named `hipstershop.ShippingService/GetQuote`, which is described as being about 286.46 ms slower than average (193% slow). This span is part of a trace involving multiple entities: Frontend, CurrencyService, RecommendationService, CheckoutService, ShippingService, CartService, and PaymentService. The trace map shows the flow of data between these services. A red arrow points from this section to the main trace details table below.

Entity	Span	Duration
Frontend	Frontend	460ms
CurrencyService	CurrencyService	0ms
RecommendationService	recommendationservice	1ms
CheckoutService	checkoutservice	43.0ms
ShippingService	hipstershop.ShippingService/GetQuote	435.18 ms
CartService	CartService	450ms
PaymentService	PaymentService	0ms

**Logs in Context:** A red box highlights the 'Logs' tab for the `hipstershop.ShippingService/GetQuote` entity. It shows two log entries:

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

A red arrow points from this section to the bottom right corner of the interface.

**Annotations:** The text 'Anomalous Span Detection in Distributed Tracing for OpenTelemetry' is displayed at the bottom left, and 'Logs in Context (correlate to the right trace)' is displayed at the bottom right.

# 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 window for the `hipstershop.ShippingService/GetQuote` entity shows a timeline chart for the `shippingService` (439ms) with a duration of 434.72 ms. It includes a "Golden signals" graph and tabs for Performance, Attributes, and Details.

**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 window for the `hipstershop.ShippingService/ShipOrder` entity shows a timeline chart for the `shipOrder` service (438ms) with 1 error. The error is described as "zipcode is invalid". It includes tabs for Performance, Attributes, and Details. An arrow points from the error message to the "Error Details" section of the modal.

**Bottom Text:**

- Get deeper into the app stack, by building spans.**
- Pinpoint errors, by improving observation (via span attributes)**

# OpenTelemetry in Action

```
[main] GET /product/OLJCESPC7Z          44  11(25.00%) |  75   1  1605  17 |
[main] 0.20  0.00
[main] POST /setCurrency               40  6(15.00%)  |  98   2  597  49 |
[main] 0.10  0.00
[main] -----
[main] Aggregated                      565 117(20.71%) |  55   1  1605  18 |
[main] 2.90  0.40
[main]
[server] {"message":"[GetQuote] received request","severity":"info","timestamp": "2022-06-26T09:56:08.259921876Z"}
[server] {"message":"[GetQuote] completed request","severity":"info","timestamp": "2022-06-26T09:56:08.259972475Z"}
[main] Name
eq/a failures/s
[main] -----
[main] GET /
0.00  0.00
[main] GET /cart
0.10  0.00
[main] POST /cart
0.70  0.30
[main] GET /cart/checkout
0.20  0.00
[main] GET /product/OPUR6V6EVO
0.20  0.00
[main] GET /product/1YKWWN1N4O
0.50  0.00
[main] GET /product/2ZYFJ3GM2N
0.10  0.00
[main] GET /product/66VCHSJNUP
0.10  0.00
[main] GET /product/6E92ZMYYFZ
0.20  0.00
[main] GET /product/9SIQTBTOJO
0.20  0.00
[main] GET /product/L9ECAV7KIM
0.00  0.00
[main] GET /product/L54PSXNUJM
0.10  0.00
[main] GET /product/OLJCESPC7Z
0.20  0.00
[main] POST /setCurrency
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.json  src
CODEOWNERS        images    otel-kubernetes-manifests  renovate.json  values-newrelic.yaml
CONTRIBUTING.md  istio-manifests  pb                  SECURITY.md
sh-5.1# kubectl get nodes
NAME           STATUS    ROLES   AGE   VERSION
minikube       Ready    control-plane   69s   v1.24.1
sh-5.1# kubectl get pods --all-namespaces
NAMESPACE     NAME           READY   STATUS    RESTARTS   AGE
kube-system   coredns-5dd4b75bd-fwqkh   1/1    Running   0          69s
kube-system   etcd-minikube   1/1    Running   0          62s
kube-system   kube-apiserver-minikube  1/1    Running   0          62s
kube-system   kube-controller-manager-minikube  1/1    Running   0          62s
kube-system   kube-proxy-4k6t6      1/1    Running   0          59s
kube-system   kube-scheduler-minikube  1/1    Running   0          81s
kube-system   storage-provisioner  1/1    Running   0          80s
sh-5.1# [REDACTED]
```

## OpenTelemetry - Prerequisites

This is the first step for the FOK Stack workshop. Please validate all the required modules are running, before moving on to the next step.

## Deployment behind the scenes:

- Installed Docker, Minikube, Skafold, and Git.
- Deployed Minikube.
- Cloned all the files from Github for this workshop.

When the deployment is completed, you can run the following to validate the deployment.

- run `ls` to see all available folders.
- run `sudo docker run hello-world` to test your docker installation.
- run `kubectl get nodes` to see nodes deployed by Minikube.
- run `kubectl get pods --all-namespaces` to see pods deployed by Minikube.

**Validate K8s is running, and follow the instructions**

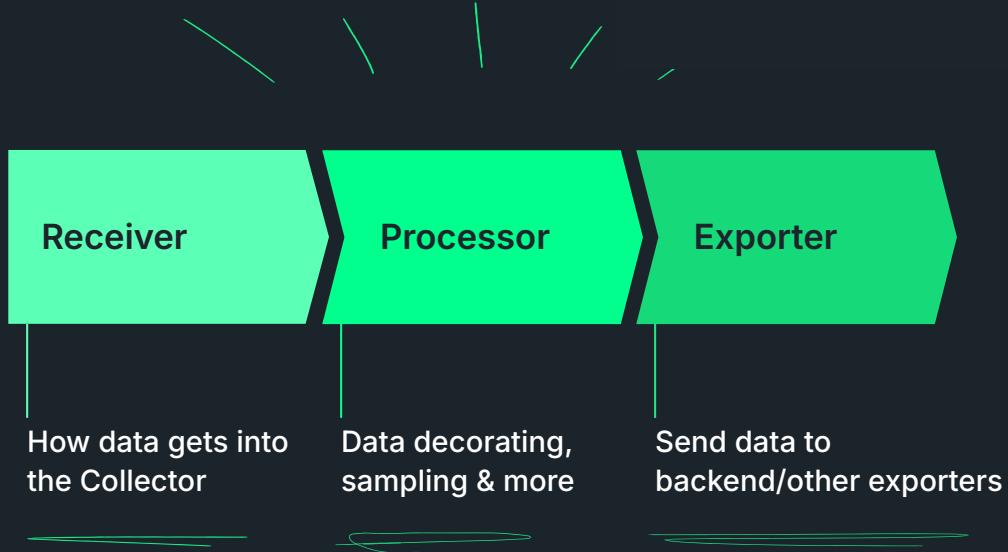
**Important** - deploy might take up to 10 mins

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

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

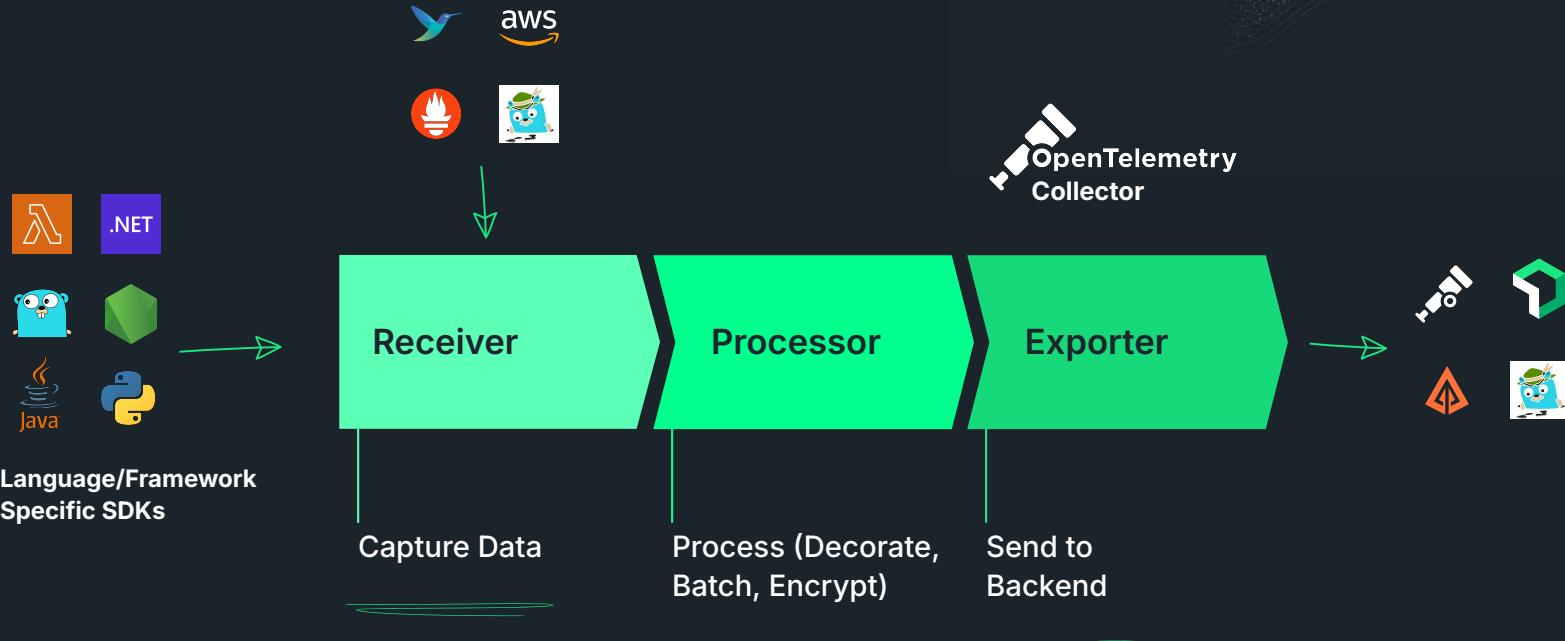
# Collector Component

OpenTelemetry

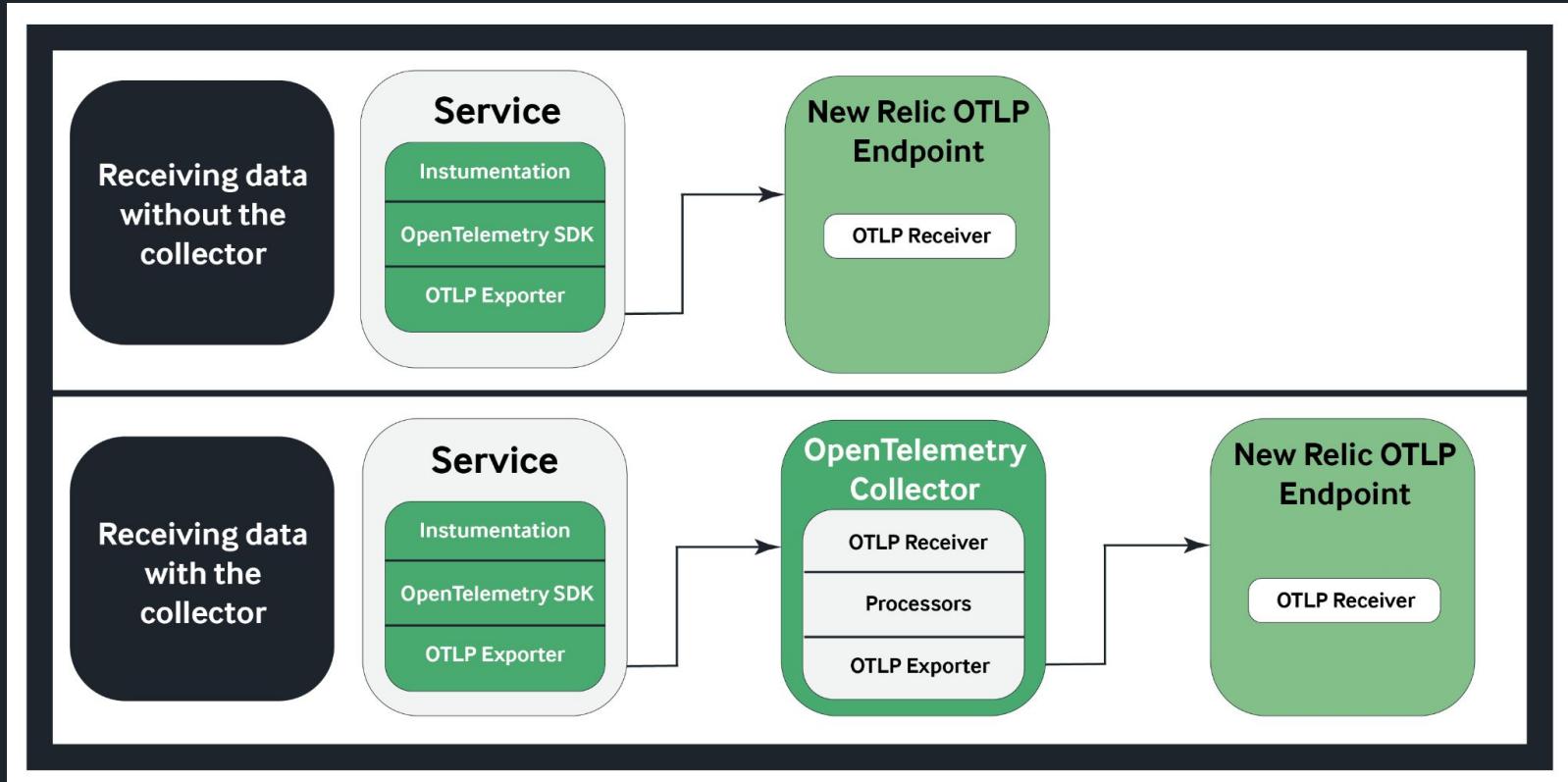


# Putting it All Together

# OpenTelemetry



# OpenTelemetry - Collector



# OpenTelemetry - Collector

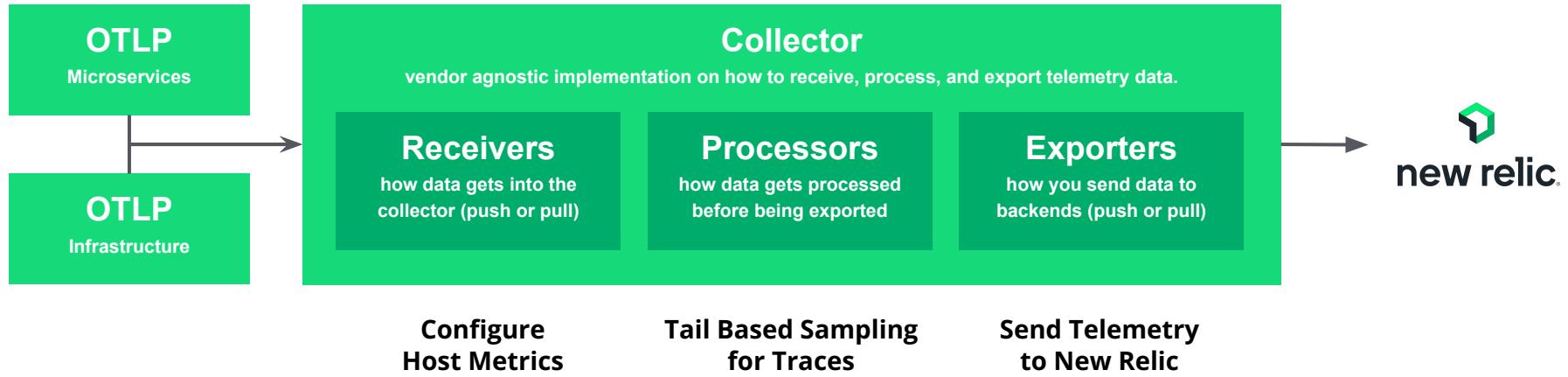
## Benefits

- Helps manage ingest costs when using tail sampling
- Reduces app overhead by offloading data management from your apps
- Provides flexibility to handle multiple data formats
- Centralizes configuration of your telemetry pipelines
- Provides ability to export data to multiple backends
- Collects host metrics, e.g., RAM, CPU, and storage capacity
- Provides ability to pull data from monitored systems, e.g., Redis

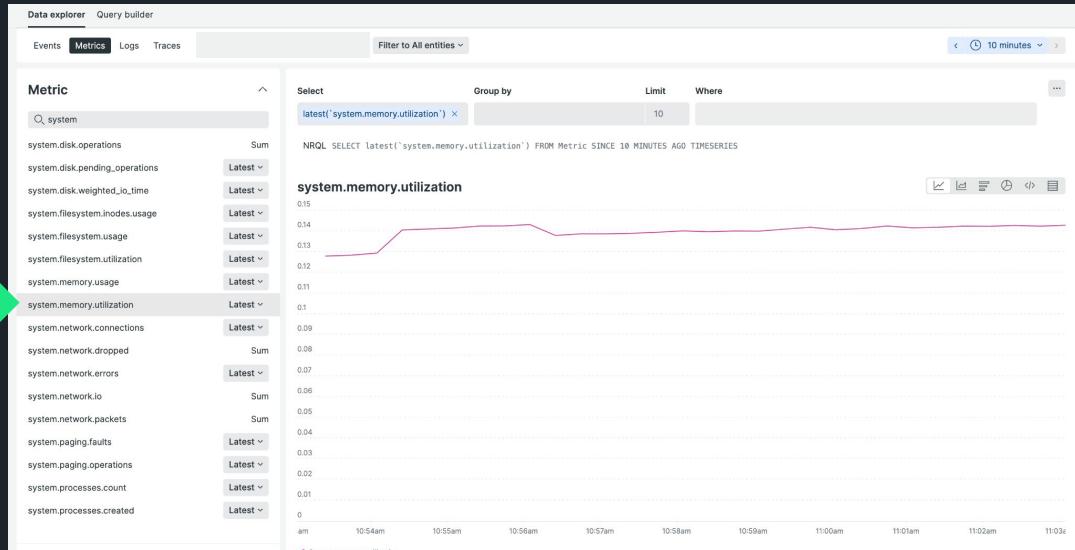
## Downsides - nontrivial!

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

# OpenTelemetry - Collector



# Collector in Action - Host Metrics



The screenshot shows the New Relic Data Explorer interface. On the left, there is a file tree with several configuration files. A red arrow points from the file tree to a red box highlighting a portion of the configuration file. A green arrow points from the red box to the Data Explorer interface.

**File Tree:**

- images
- istio-manifests
- kubernetes-manifests
- otel-kubernetes-manifests
- pb
- release
- src
  - adservice
  - cartservice
  - checkoutservice
  - currencybservice
  - emailservice
  - frontend
  - loadgenerator
  - otel-collector
    - Dockerfile
    - collectoryaml
  - otel-collector-agent
    - Dockerfile
    - collectoryaml
  - otel-collector-agent-re...
  - paymentservice
  - productcatalogservice
  - quotesservice
  - recommendationservice
  - shippingservice
  - .gitignore
- CODEOWNERS
- CODE\_OF\_CONDUCT.md
- CONTRIBUTING.md
- LICENSE
- README.md
- SECURITY.md

**Configuration File (highlighted by red box):**

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

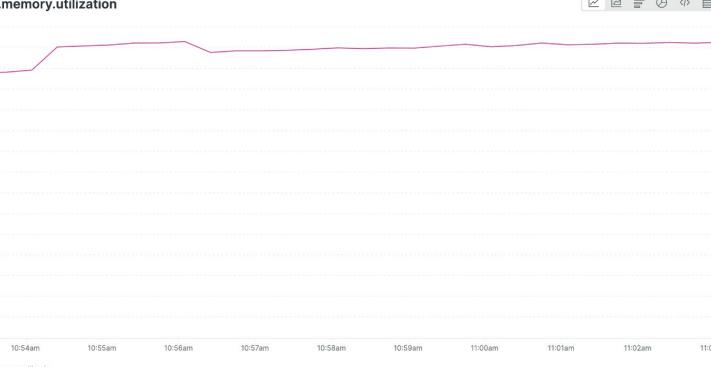
**Data Explorer Interface:**

Events Metrics Logs Traces

Select: latest('system.memory.utilization') Group by: Limit: 10 Where:

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

system.memory.utilization



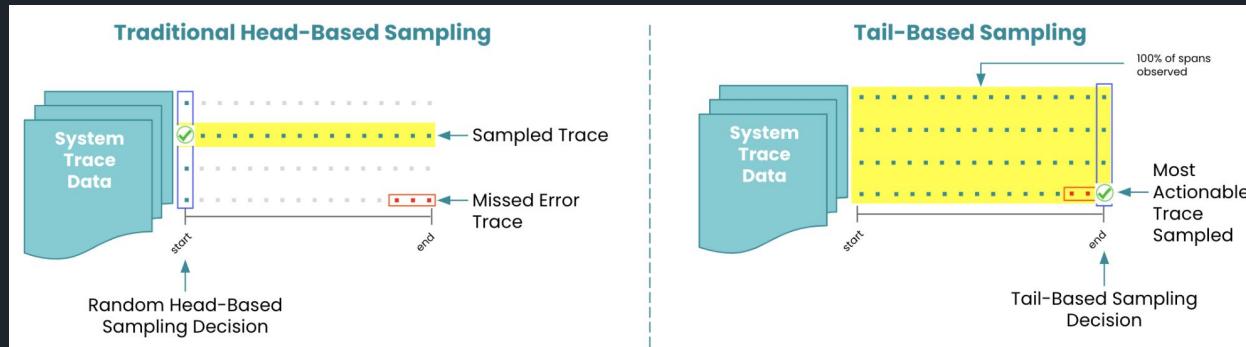
Time	System Memory Utilization
10:54am	0.12
10:55am	0.13
10:56am	0.14
10:57am	0.13
10:58am	0.13
10:59am	0.13
11:00am	0.13
11:01am	0.13
11:02am	0.13
11:03am	0.13

System.memory.utilization

Collect additional infrastructure metrics  
for OpenTelemetry

# HEAD vs TAIL Sampling

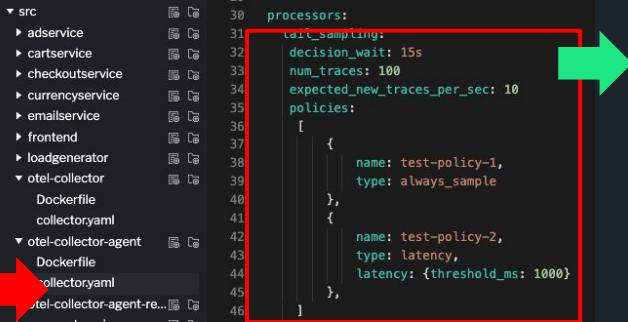
Storing all tracing data is **costly**. Most of the time, you  
**ONLY NEED THE RIGHT SAMPLING OF DATA**



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

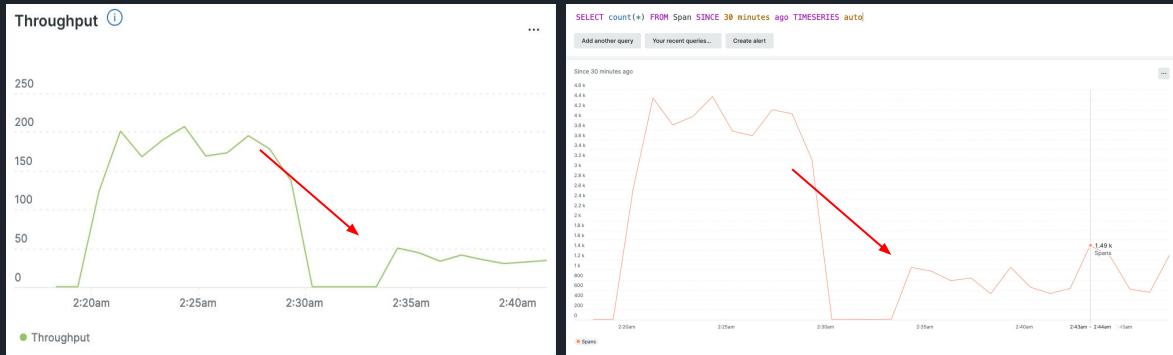
**Tail**-based sampling is best to decide what to keep, based on isolated, independent portions of the trace data.

# 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.