Observability Day
NORTH AMERICA

KubeCon + CloudNativeCon
Salt Lake City, Utah

Observability Day
NORTH AMERICA

Enhancing Asynchronous Communication Observability with OpenTelemetry

Liudmila Molkova, Microsoft
Shivanshu Raj Shrivastava, SigNoz

About the Speakers

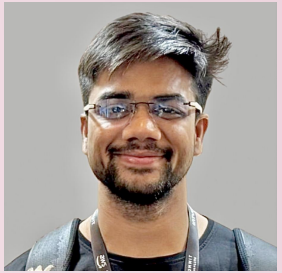
Observability Day
NORTH AMERICA

November 12, 2024
Salt Lake City

Liudmila Molkova

- Principal Software Engineer
- Microsoft
- Working on Azure client libraries and developer experiences
- Member OTeL Technical Committee
- OTeL SemConv maintainer

- Twitter: @ne_skazu
- Linkedin: lmolkova
- Github: lmolkova



Shivanshu Raj Shrivastava

- Founding Engineer
- SigNoz (YC W21)
- Building open source OTeL native observability products
- Member and contributor OpenTelemetry
- CNCF Ambassador

- Twitter: @shivanshu1333
- Linkedin: shivanshu1333
- Github: shivanshuraj1333

Observability Day
NORTH AMERICA

Enhancing Asynchronous Communication Observability with OpenTelemetry

Liudmila Molkova, Microsoft
Shivanshu Raj Shrivastava, SigNoz

Async messaging: what can go wrong?

Common problems

- Message Loss
- High Latency and Throughput Issues
- Dead Letter Queues
- Monitoring Blind Spots
- Throttling and Quota Limits
- Message Duplication
- Message Ordering Issues
- Consumer Lag and Back-pressure
- Broker Failures
- Network Partitions and Latency
- Security and Authentication Issues
- Partition bloating

Messaging queues are Complex!

Managing them is **pain** for cluster managers who are running queues at scale :(

Messaging Queues

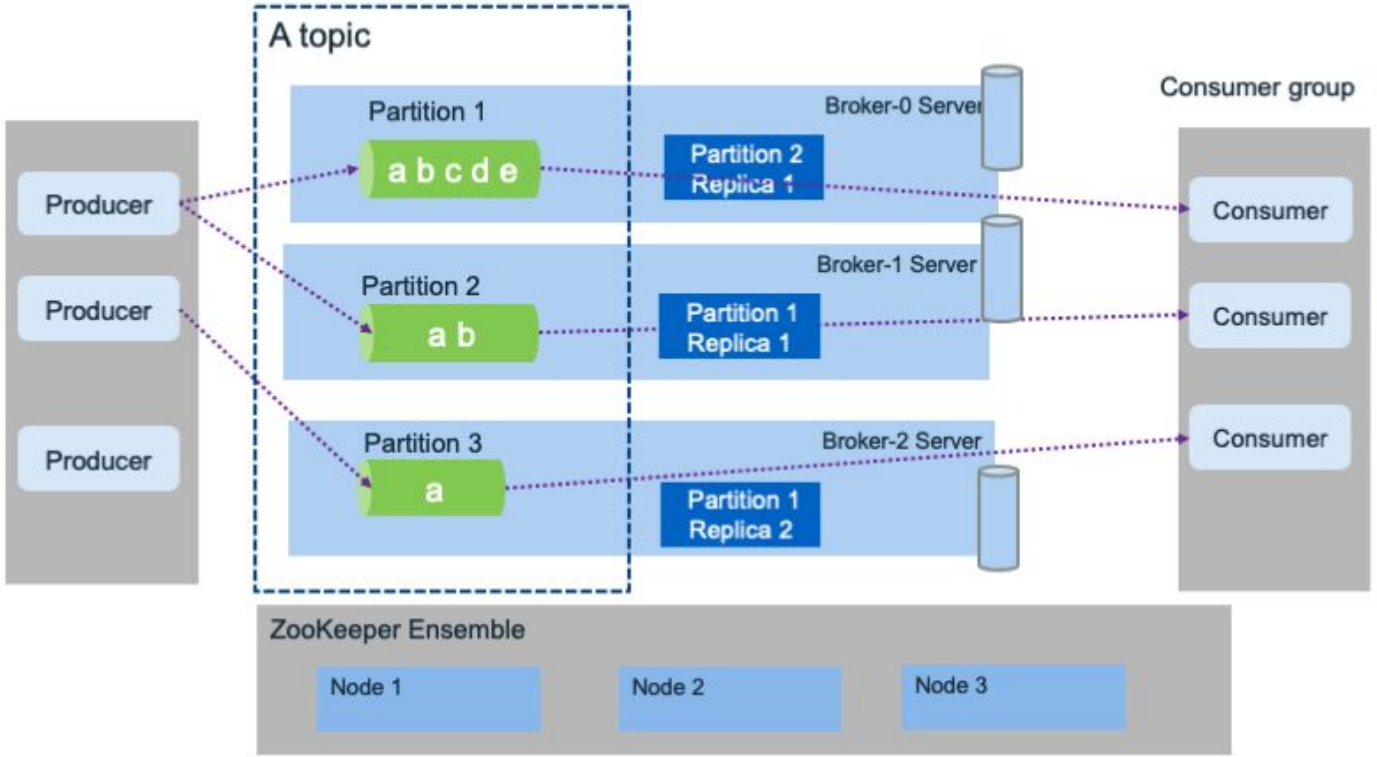
Developers often use **default** configurations, while fine tuning with **deeper observability** can help them **scale better**

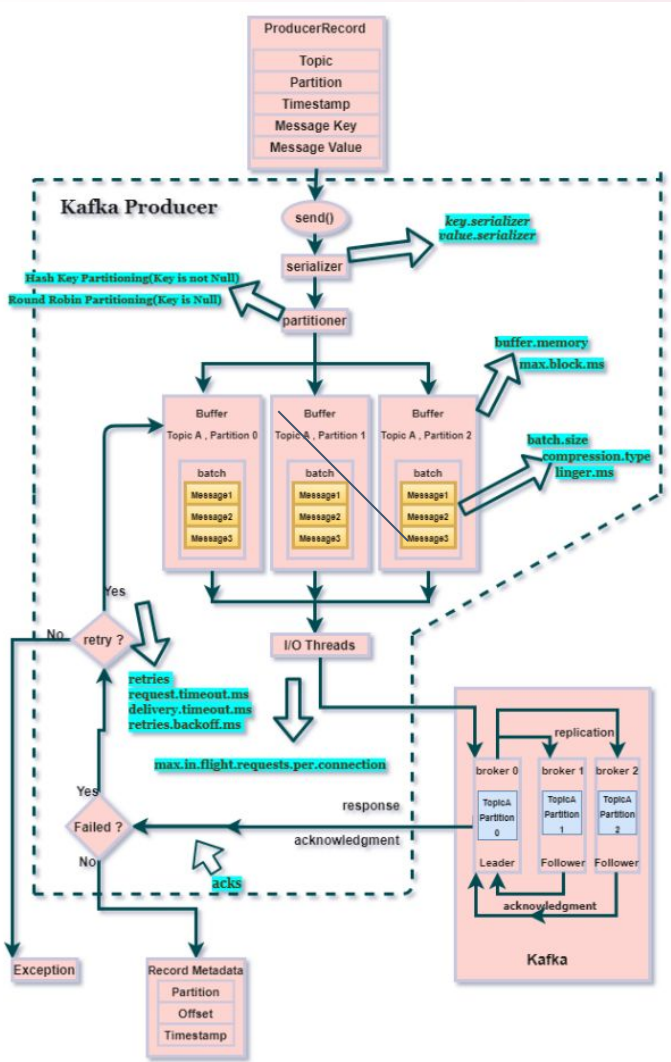
Knowing **when** and **how** things are breaking, can **help in fixing this faster** :)

Let's see how the magic happens inside a queue

Our favourite Kafka

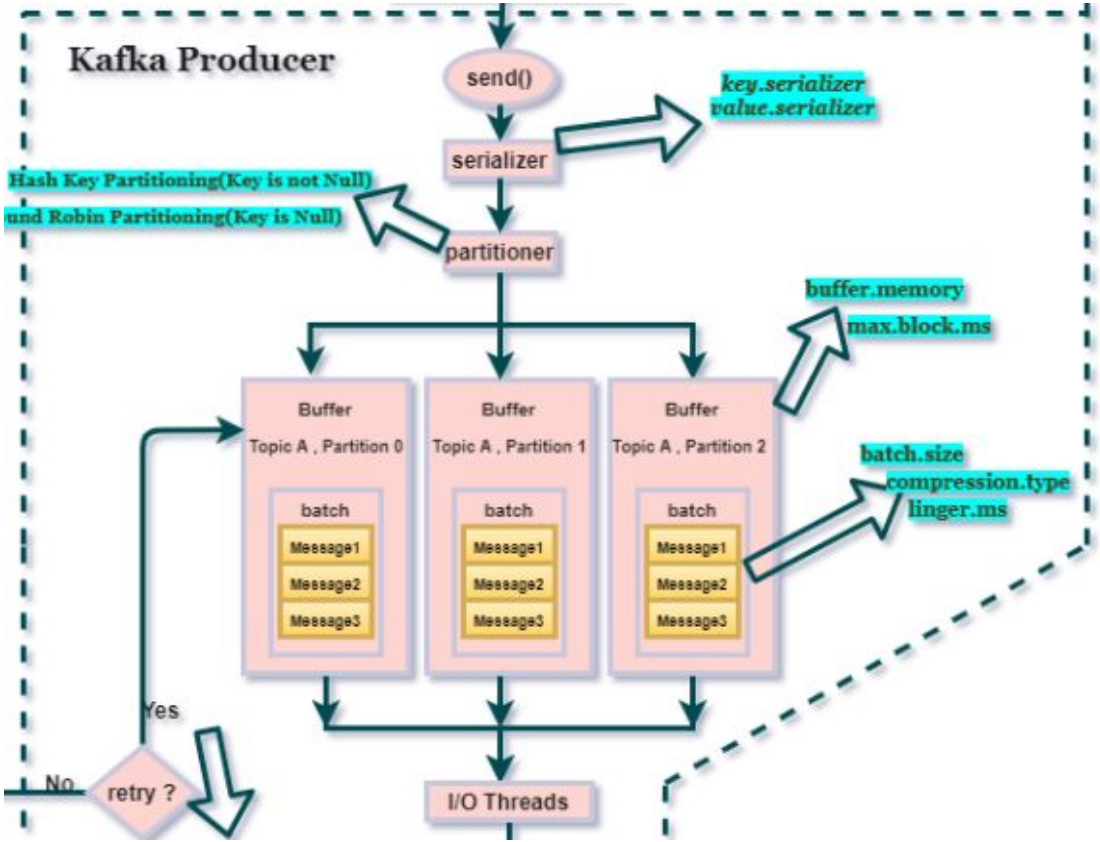
Messaging Queues




ProducerRecord
Topic
Partition
Timestamp
Message Key
Message Value
Kafka Producer
send()
key.serializer
value.serializer
serializer
Hash Key Partitioning(Key is not Null)
Round Robin Partitioning(Key is Null)
partitioner
buffer.memory
max.block.ms
Buffer
Topic A , Partition 0
Buffer
Topic A , Partition 1
Buffer
Topic A , Partition 2
batch
Message1
Message2
Message3
batch.size
compression.type
linger.ms
Yes
No
retry ?
I/O Threads
retries
request.timeout.ms
delivery.timeout.ms
retries.backoff.ms
max.in.flight.requests.per.connection
replication
broker 0
broker 1
broker 2
TopicA Partition 0
TopicA Partition 1
TopicA Partition 2
Leader
Follower
Follower
Yes
Failed ?
response
acknowledgment
No
acks
acknowledgment
Kafka
Exception
Record Metadata
Partition
Offset
Timestamp

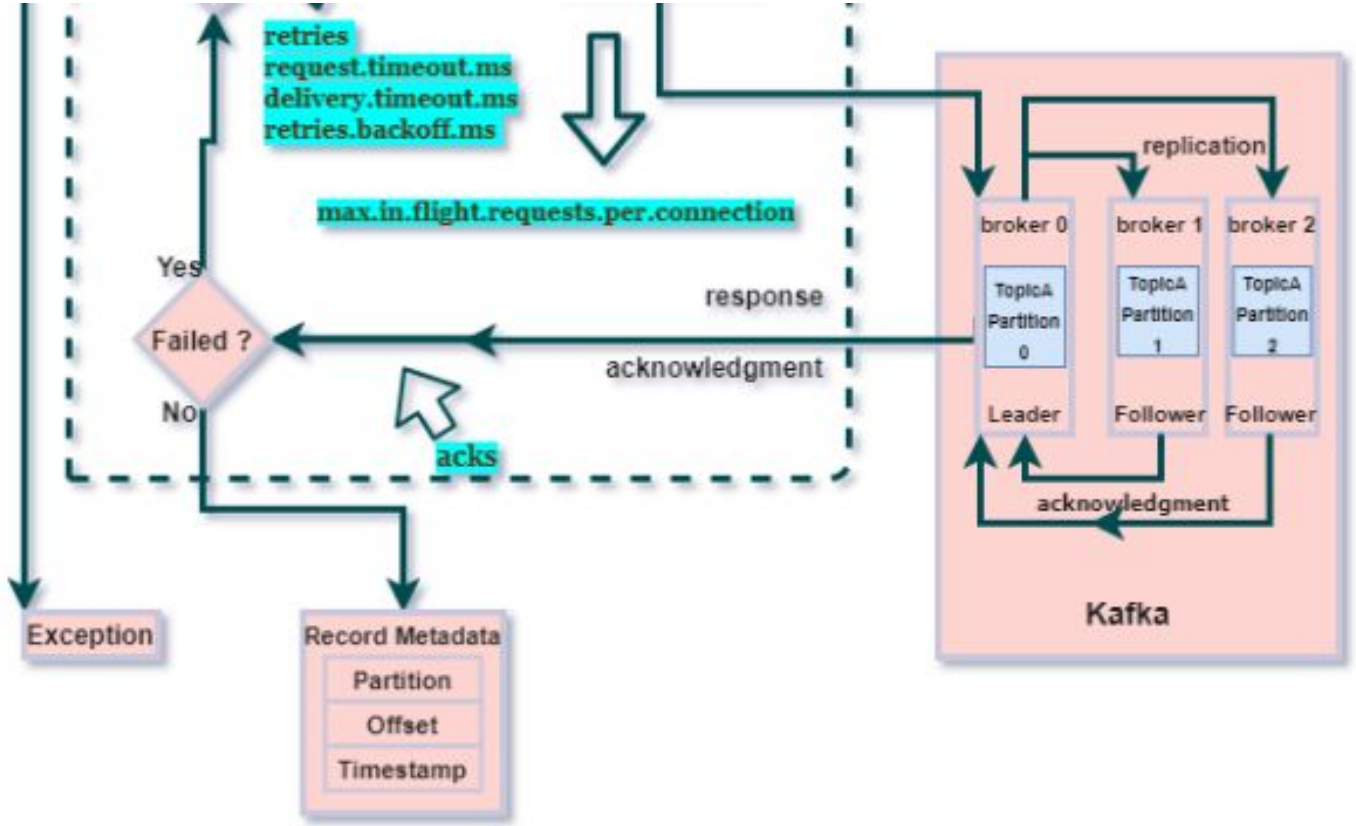
Complex!

Messaging Queues



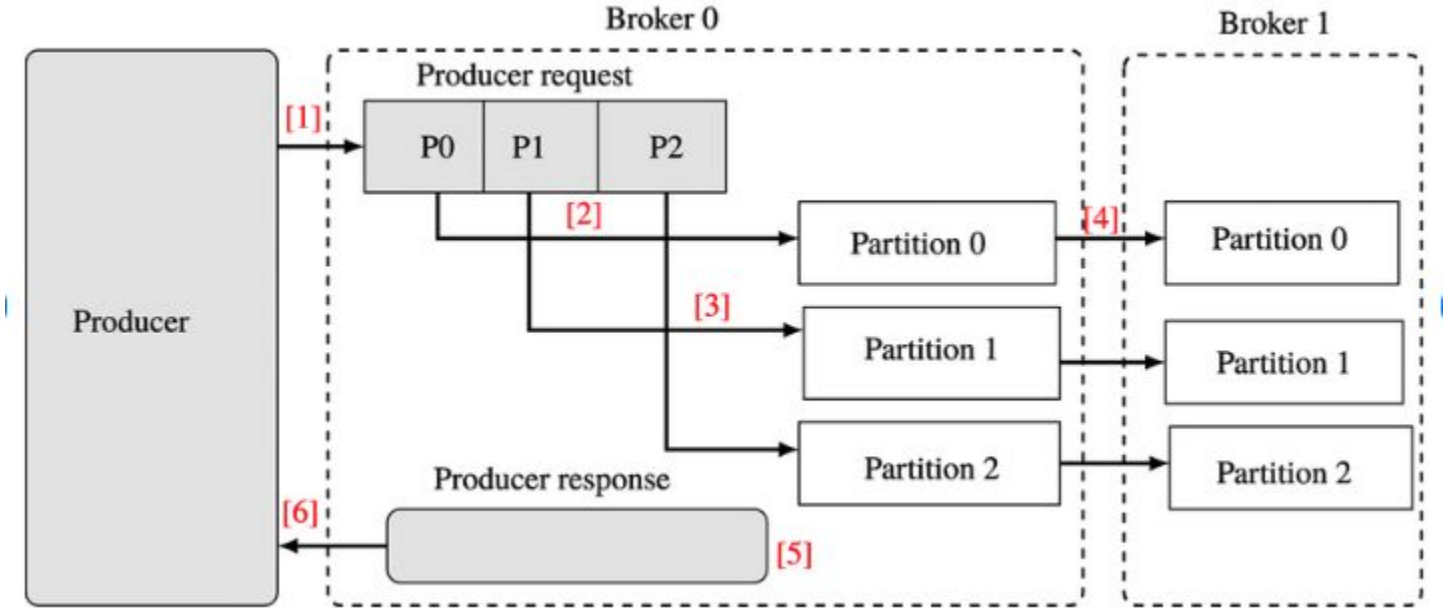
Complex!!

Messaging Queues



Complex!!!

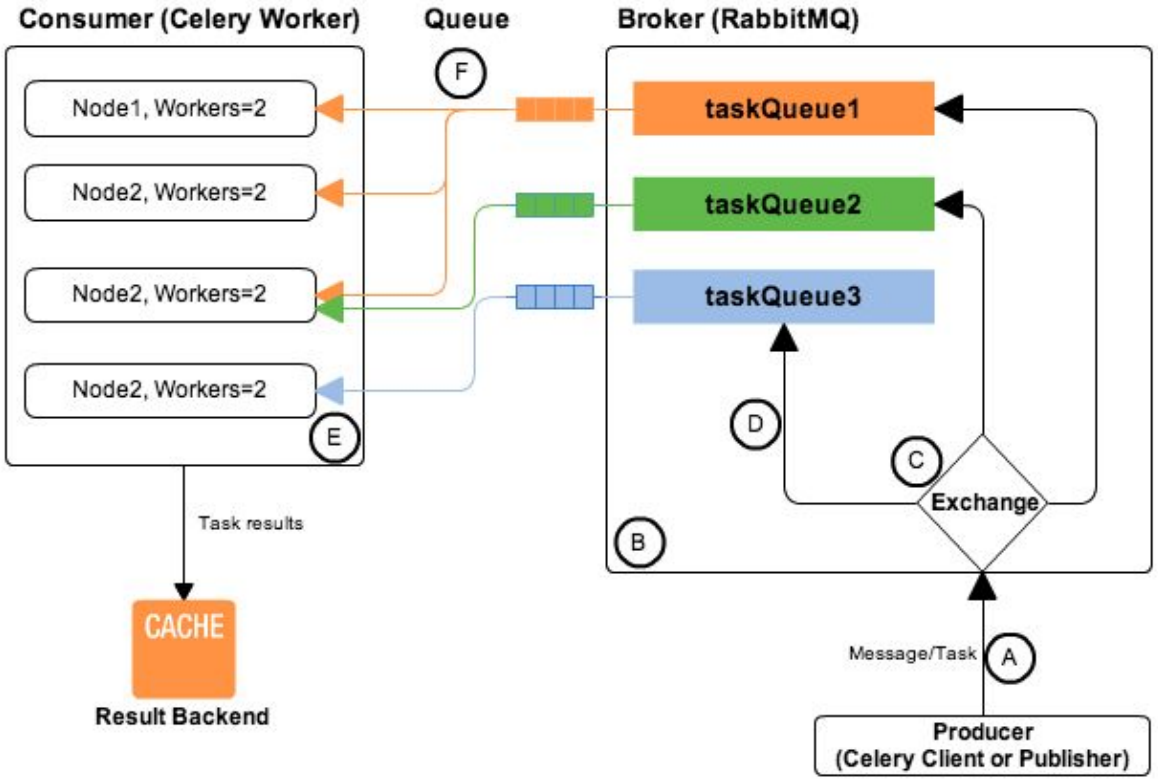
Messaging Queues



LGTM :)

Celery + RabbitMQ

Messaging Queues



There's a queue for every human :|

Messaging Queues

1. RabbitMQ
2. Azure Event Hubs
3. Apache Kafka
4. ActiveMQ
5. Amazon MSK
6. Amazon SQS
7. Redis
8. ZeroMQ
9. IBM MQ
10. Google Cloud Pub/Sub
11. NATS
12. NSQ
13. Strimzi

Some OSS, some vendor managed, some proprietary

TBH, many more….

What you (don't) see on broker metrics

Common problems

- **Kafka does not directly expose producer-side latency metrics per partition.**
  - No one ever knows why there is high producer latency.
  - There is no visibility if there's a problem with Kafka itself or producer.
  - Difficulty tracking producer throughput across different topics
  - Complex to monitor batch sizes and compression ratios
  - Complex to find the correlation between different partitions

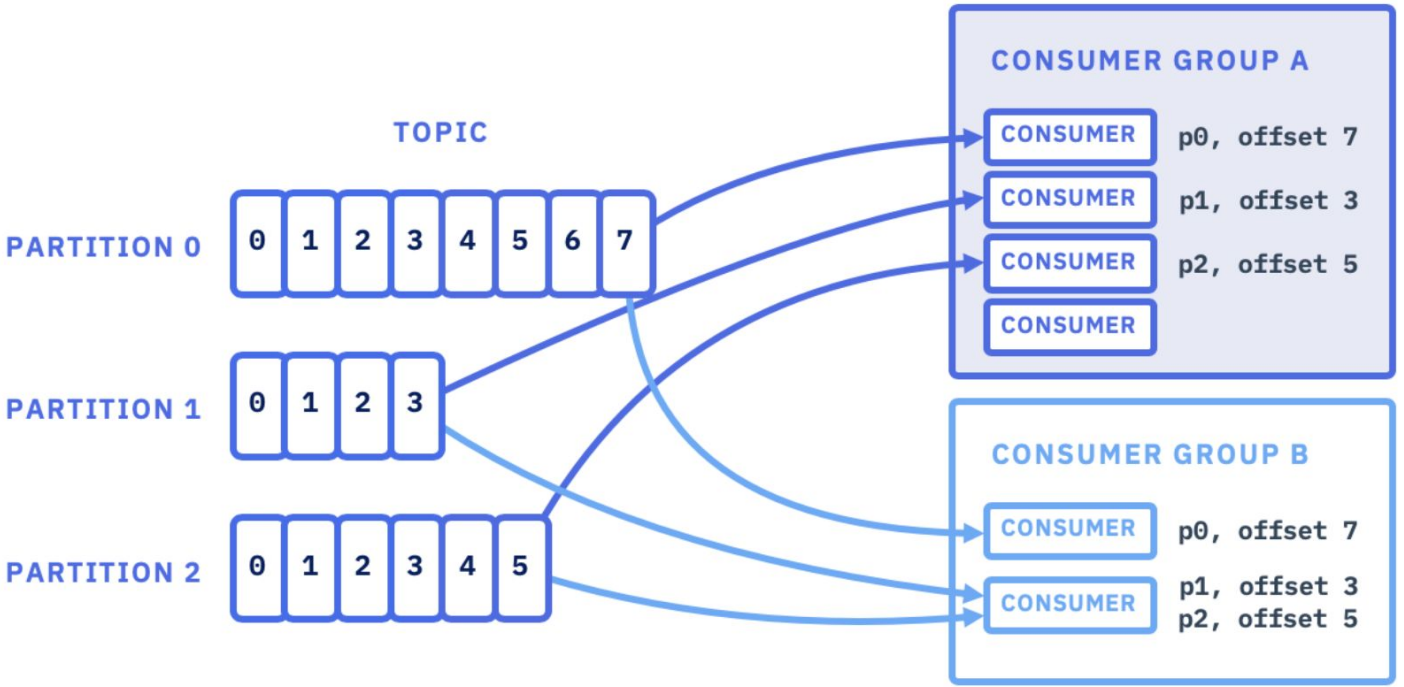
Common problems

- **End-to-End Latency (How much time it took for producer to send a message and a consumer to acknowledge it)**
  - Difficult to track processing time per message
  - No direct correlation between consumer performance and resource utilization
  - Complex to correlate broker metrics with client-side issues

Common problems

- **Consumer Group Lag correlation with spans**
  - No visibility of which partition is causing the lag?
  - Is the lag caused by consumer application, network problems, or imbalances between producer and consumer performance.
  - Manual implementation needed for trace context propagation.
  - Limited visibility into consumer group rebalancing events

Common problems

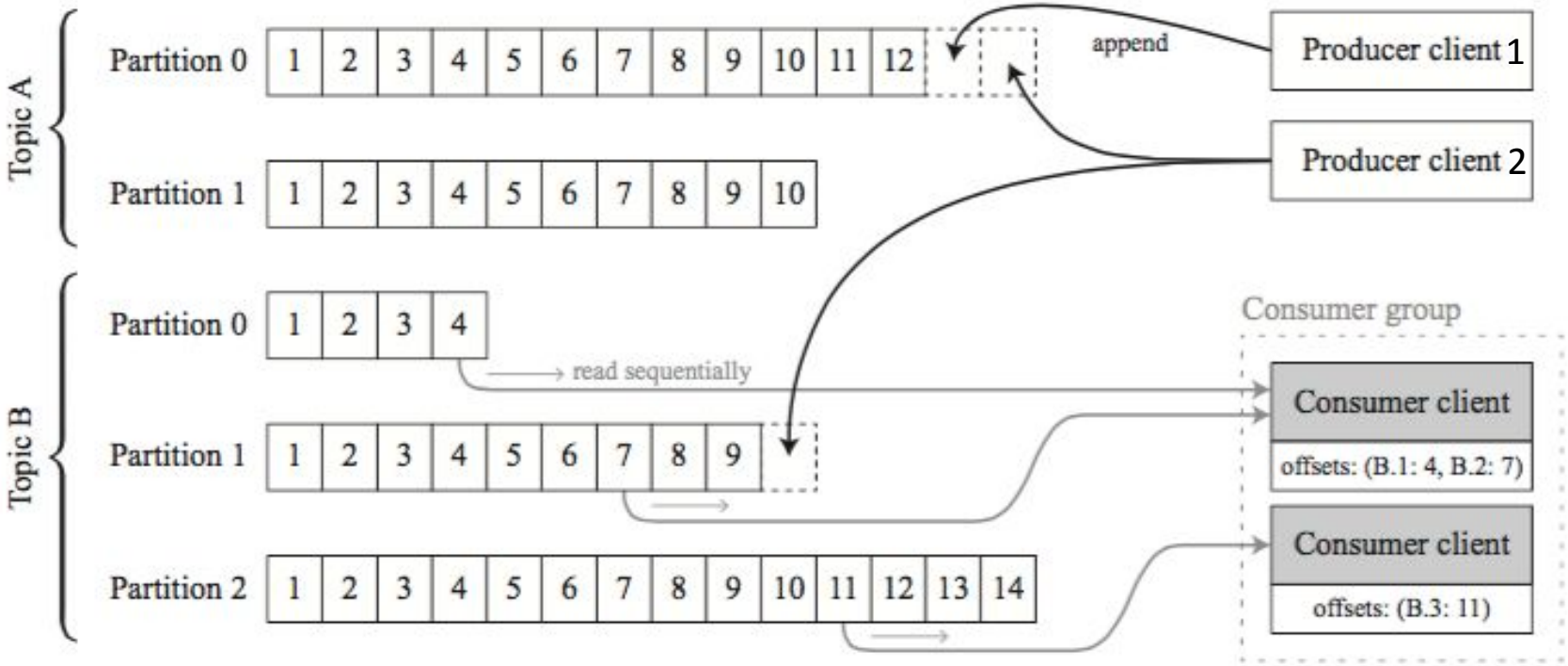


Common problems

- **Infrastructure Monitoring Gaps**
  - Complex relationship between JVM metrics and Kafka performance
  - Difficult to correlate network issues with message delivery problems
  - Limited visibility into disk I/O patterns
  - Difficult to track partition growth patterns

Noisy neighbour

Messaging Queues



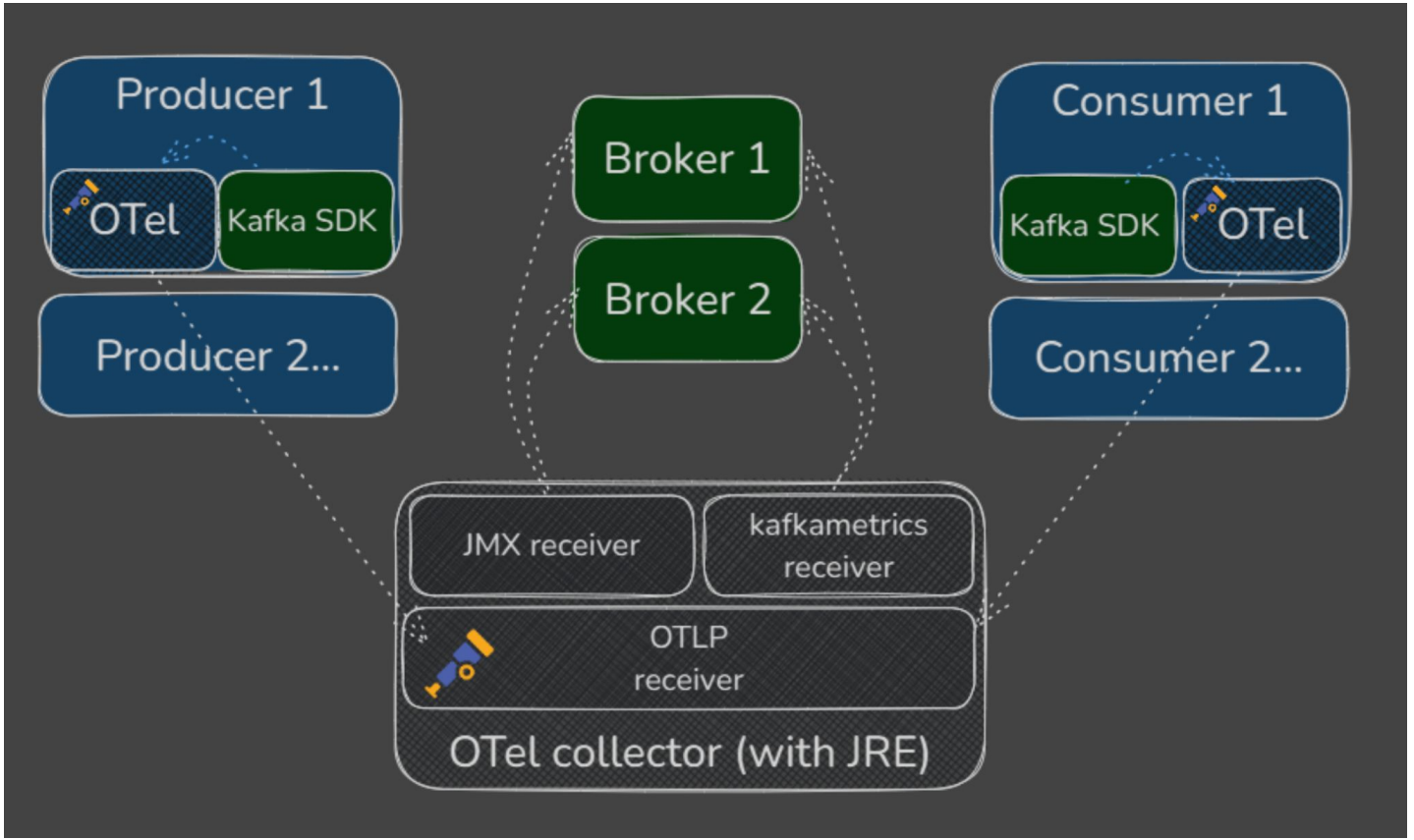
Messaging Queues

Metrics -> Trace
Trace -> Logs
Trace -> Metrics

All these correlations are hard to achieve natively in queues

How to get more insights?

Kafka OTel setup



Sample collector config: https://github.com/lmolkova/obs-day-messaging/blob/main/configs/otel-collector-config.yml

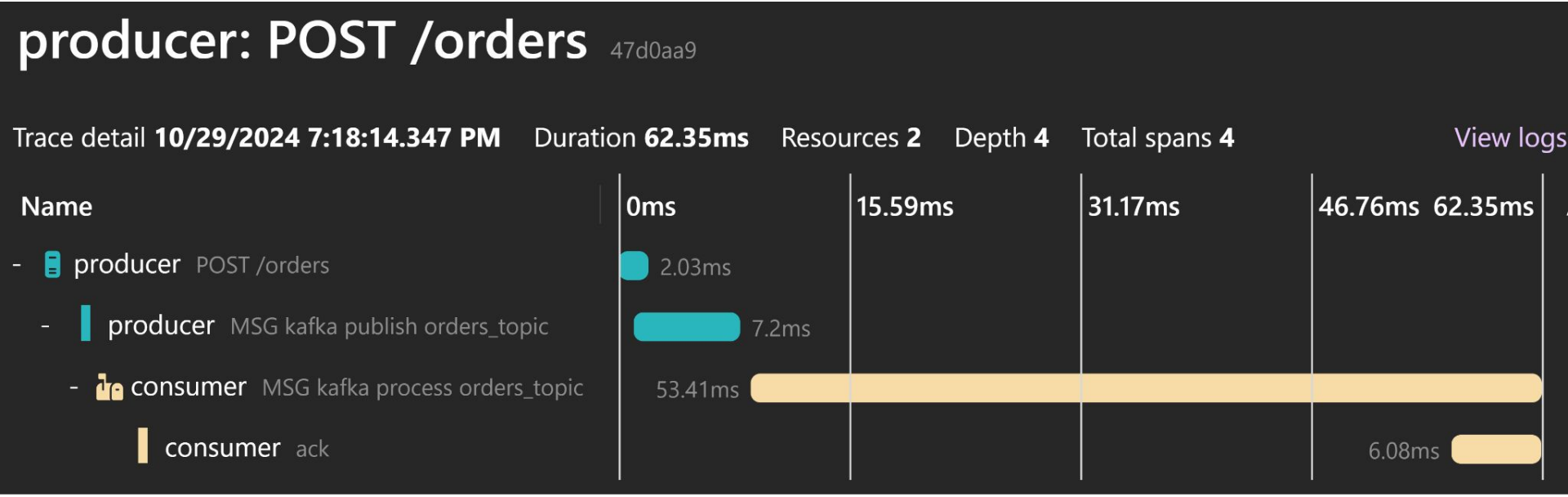
Client-side: Open Telemetry

- **Stable API** with ability to export telemetry anywhere
- **Instrumentations** for popular clients and frameworks
- **Semantic conventions** to unify telemetry format

Distributed tracing in messaging queues

Observability Day
NORTH AMERICA

- Useful in debugging individual flows and root cause issues
- Each operation is represented as span
- Trace context is propagated with the message through the messaging queues



More context in span attributes

Name	Value
Name	orders_topic publish
Kind	Producer
messaging.client_id	kafka-producer-producer-1
messaging.destination.name	orders_topic
messaging.destination.partition.id	2
messaging.kafka.message.offset	79342
messaging.operation	publish
messaging.system	kafka

Check out https://github.com/open-telemetry/semantic-conventions/blob/main/docs/messaging/messaging-spans.md

Messaging metrics

- Useful to understand **overall state and health** of the system
- **Easy** to store and query
- Very **low performance overhead**
- Telemetry is **aggregated** on client and exported with **low-cardinality** dimensions

OTel client metrics

- **Operation duration** histograms
- Sent, received, processed **message count**
- **Extra dimensions** to filter and break things down
- More to come: lag, delivery count, etc
- Implemented in a **few instrumentations** so far

Check out https://github.com/open-telemetry/semantic-conventions/blob/main/docs/messaging/messaging-metrics.md for the details!

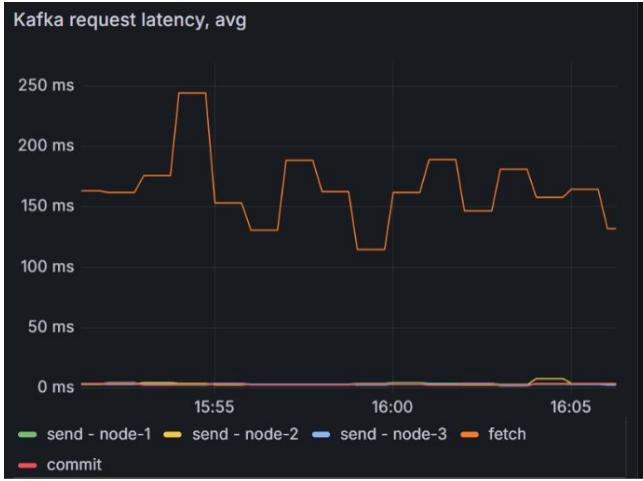
Name	Instrument Type	Unit (UCUM)	Description
`messaging.client.operation.duration`	Histogram	`s`	Duration of messaging operation initiated by a producer or consumer client.

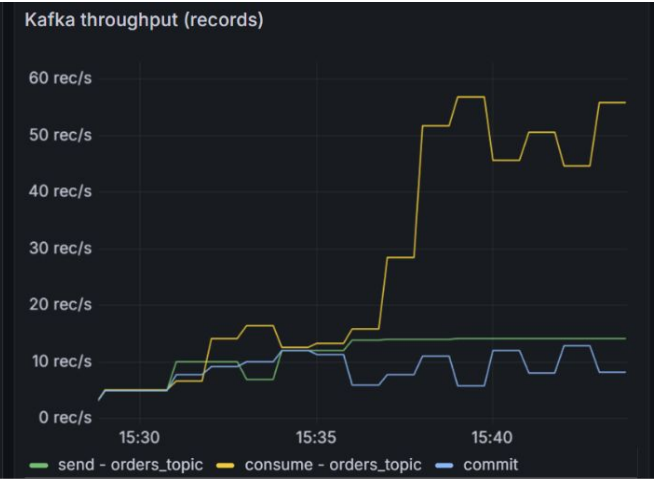
Attribute	Type	Description	Examples
`error.type`	string	Describes a class of error the operation ended with.	`amqp:decode-error`; `KAFKA_STORAGE_ERROR`; `channel-error`
`messaging.consumer.group.name`	string	The name of the consumer group with which a consumer is associated.	`my-group`; `indexer`
`messaging.destination.name`	string	The message destination name.	`MyQueue`; `MyTopic`
`messaging.destination.partition.id`	string	The identifier of the partition messages are sent to or received from, unique within the `messaging.destination.name`.	`1`
`messaging.operation.name`	string	The system-specific name of the messaging operation.	`send`; `receive`; `ack`
`messaging.operation.type`	string	A string identifying the type of the messaging operation.	`create`; `send`; `receive`
`messaging.system`	string	The messaging system as identified by the client instrumentation.	`activemq`; `aws_sqs`; `eventgrid`
`server.address`	string	Server domain name if available without reverse DNS lookup; otherwise, IP address or Unix domain socket name.	`example.com`; `10.1.2.80`; `/tmp/my.sock`
`server.port`	int	Server port number.	`80`; `8080`; `443`

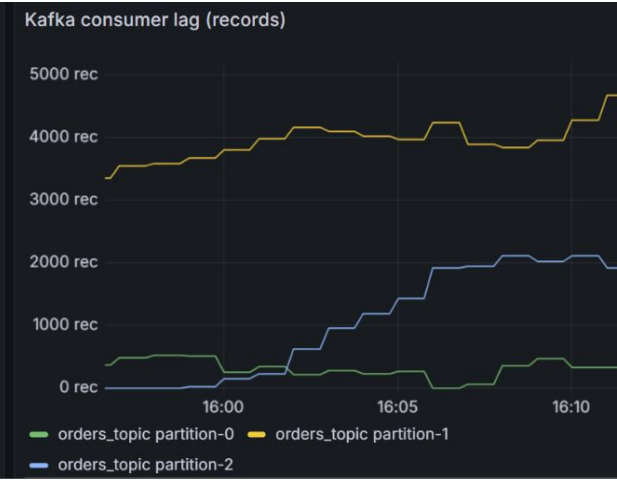
- messaging.operation.name
- messaging.system
- messaging.batch.message_count
- messaging.consumer.group.name
- messaging.destination.anonymous
- messaging.destination.name
- messaging.destination.subscription.name
- messaging.destination.template
- messaging.destination.temporary
- messaging.operation.type
- messaging.client.id
- messaging.destination.partition.id
- messaging.message.conversation_id
- messaging.message.id
- messaging.message.body.size
- messaging.message.envelope.size

Kafka client 'native' metrics

- **Throughput**: records, requests, IO, etc
- **Latency**: request, IO, internal queues
- Connectivity, authentication, buffers, batching and more

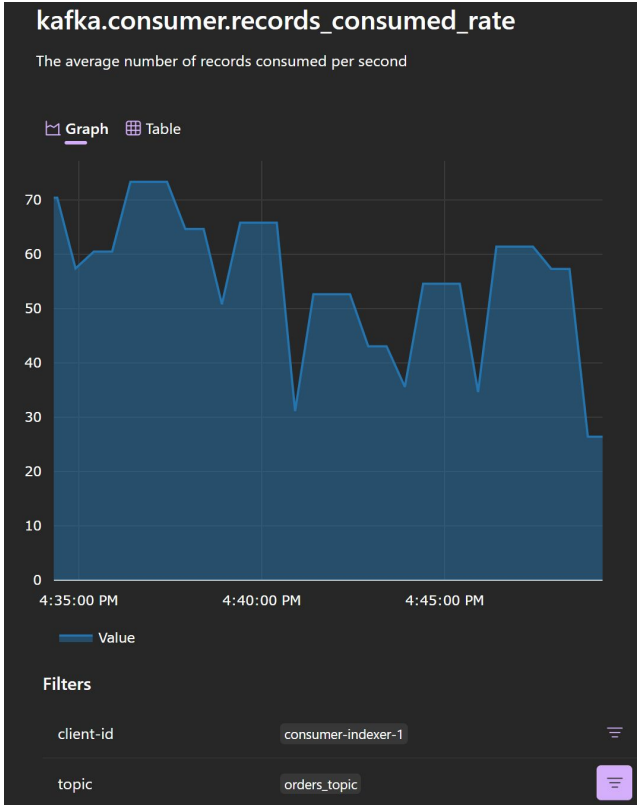






Kafka client 'native' metrics

- Client and language-specific
  - Instrumentations pull metrics from Kafka SDKs
  - Some metrics come through broker plugin (KIP-714)
- **Don't provide enough context**
  - One common dimension: `client_id`
  - Some metrics include: `node_id`, `topic`, `partition`

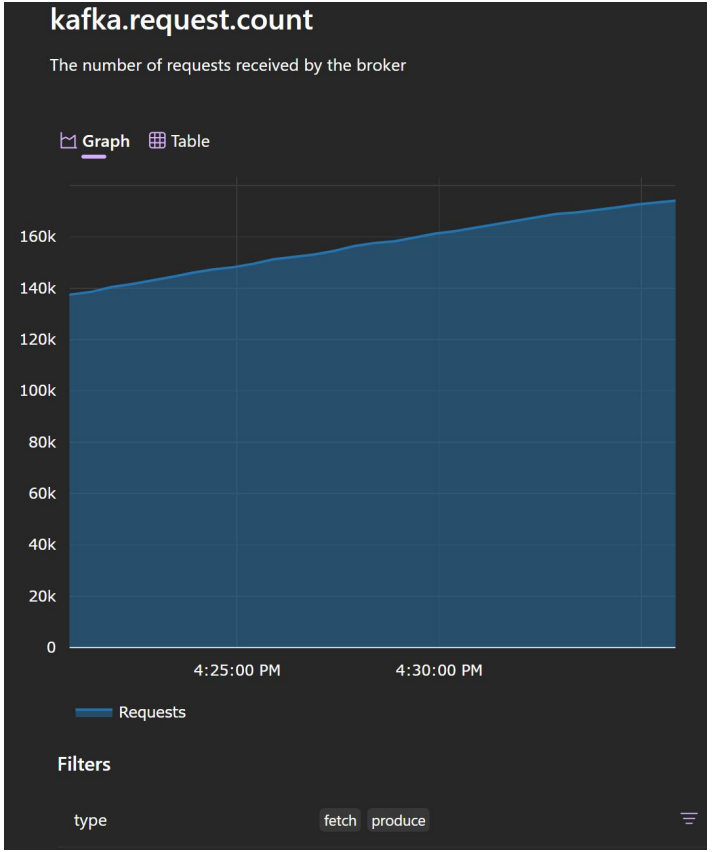


Existing Kafka metrics reported by OTel instrumentations: Java, .NET

Kafka broker 'native' metrics

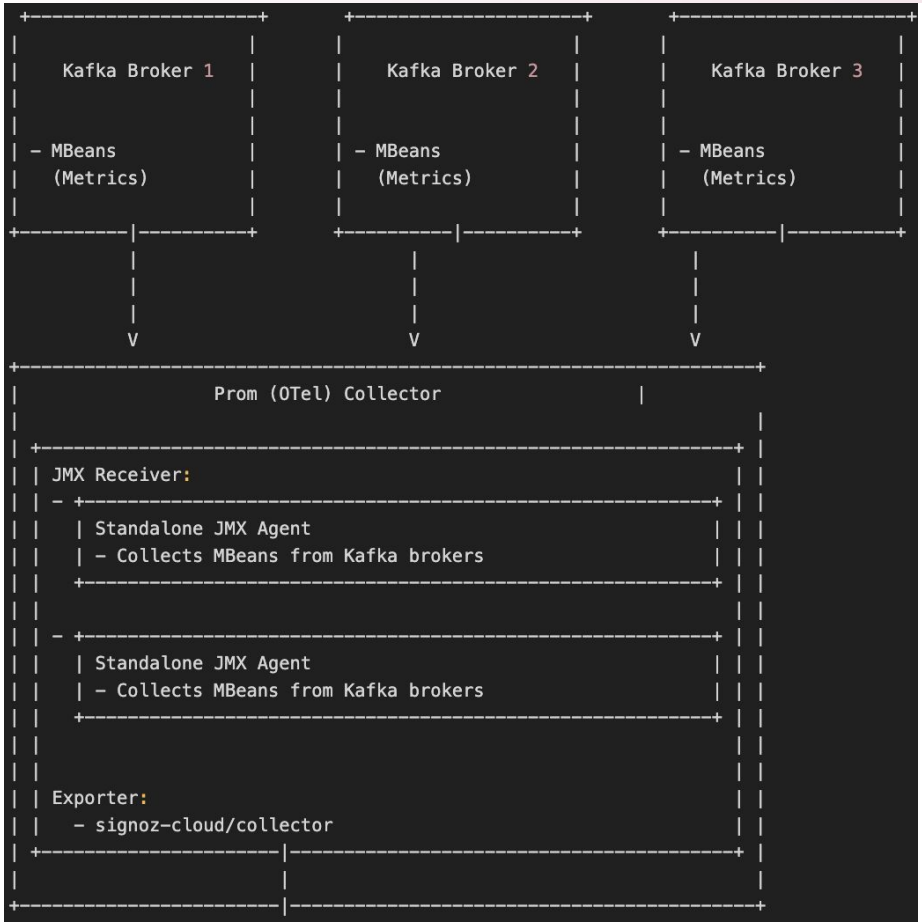
- Throughput, latency, elections, offsets, etc
- Available via JMX and kafka-metrics receivers in OTel collector

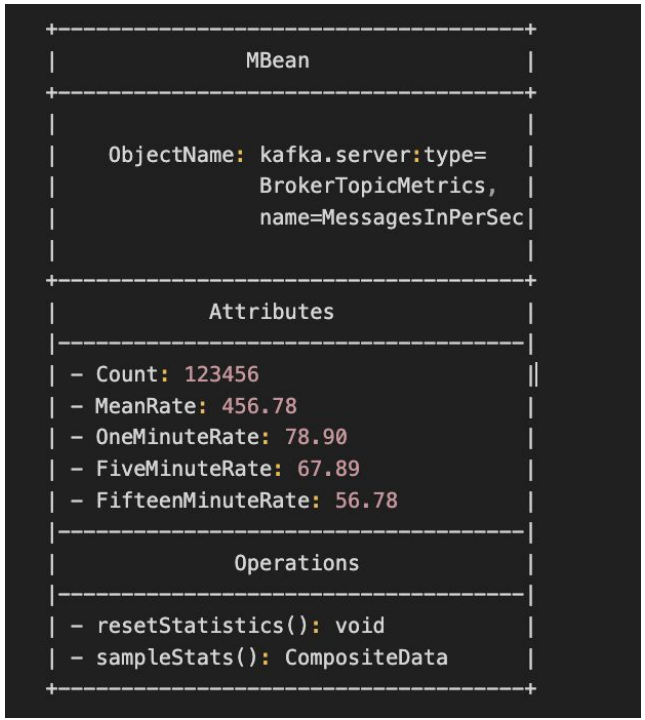
- **Still not enough context**



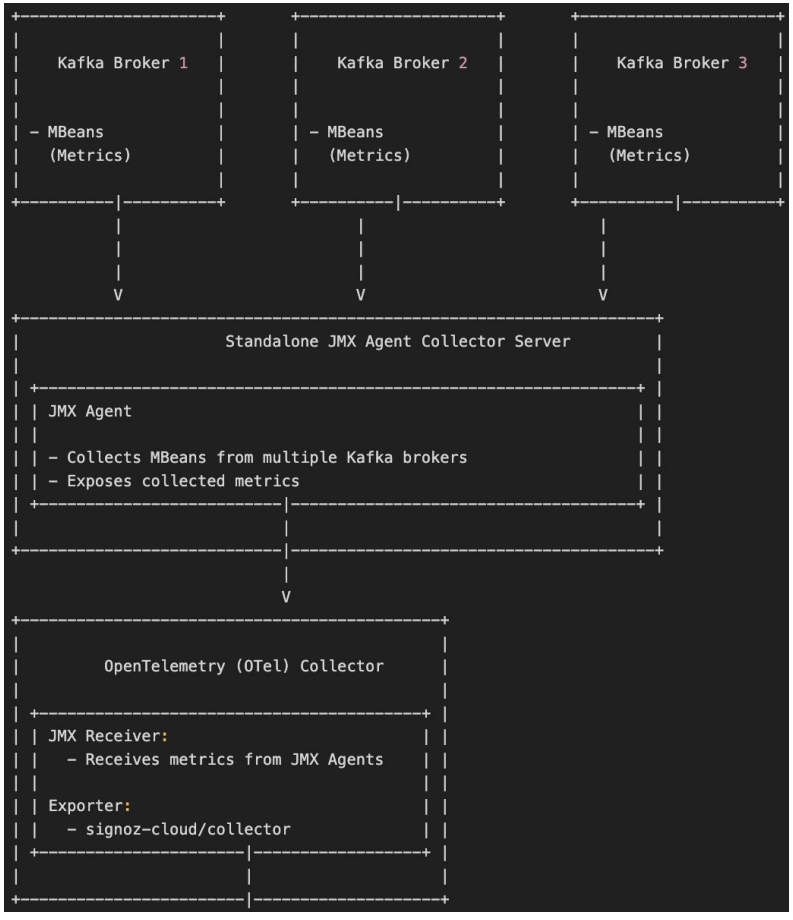
Existing Kafka metrics reported by OTel instrumentations: Java, .NET

How to get deeper insights?

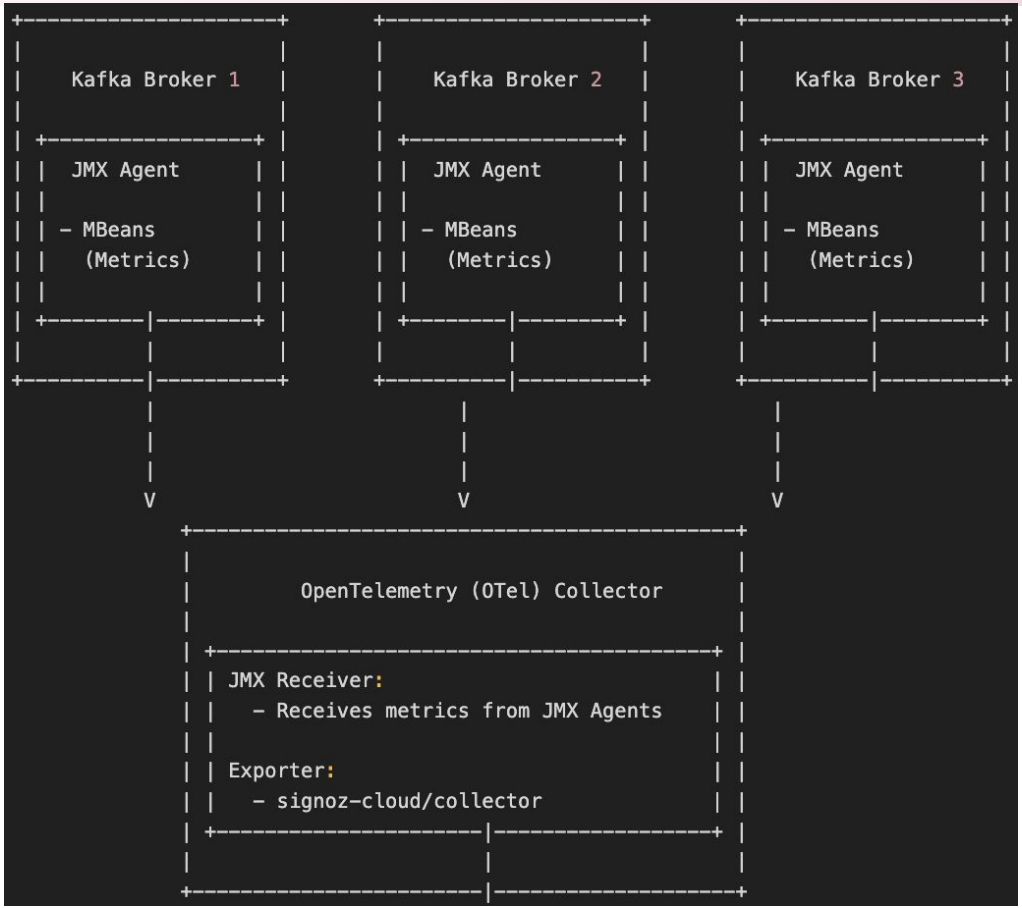




How to get deeper insights?



How to get deeper insights?



Demo!

Call to action

- Participate in OTel Messaging Semantic Conventions
  - Define new metrics, add attributes or conventions for new systems
- Improve existing instrumentations
  - Available in Java, Python, .NET, node.js, and other languages
- Ask for native instrumentation in your favorite Kafka client library

Questions?

Thanks for joining!

November 12, 2024
Salt Lake City