# Spring Kafka beyond the basics

Lessons learned on our Kafka journey at **ING Bank**

**SPRING I/O BARCELONA – MAY 27TH 2022**

Tim van Baarsen


ING

# Who am I?

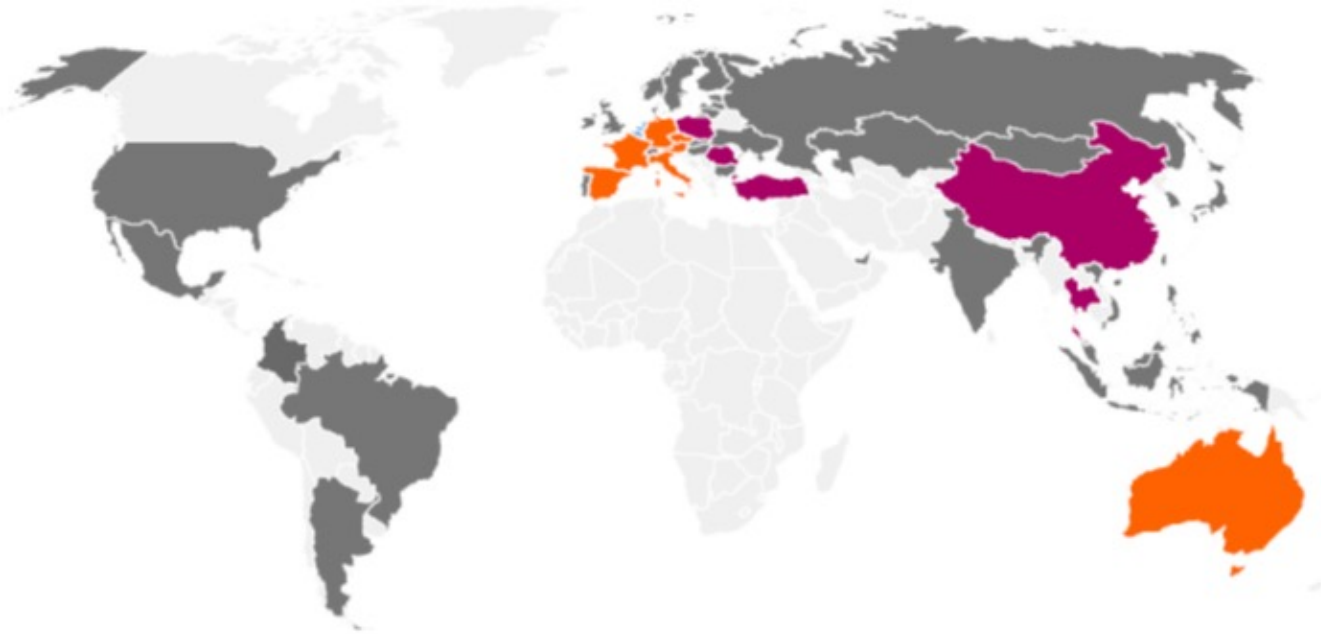
**Tim van Baarsen**
Software Engineer @ **ING Netherlands**

The Netherlands


Amsterdam

Team Dora

# ING is active in more than 40 counties



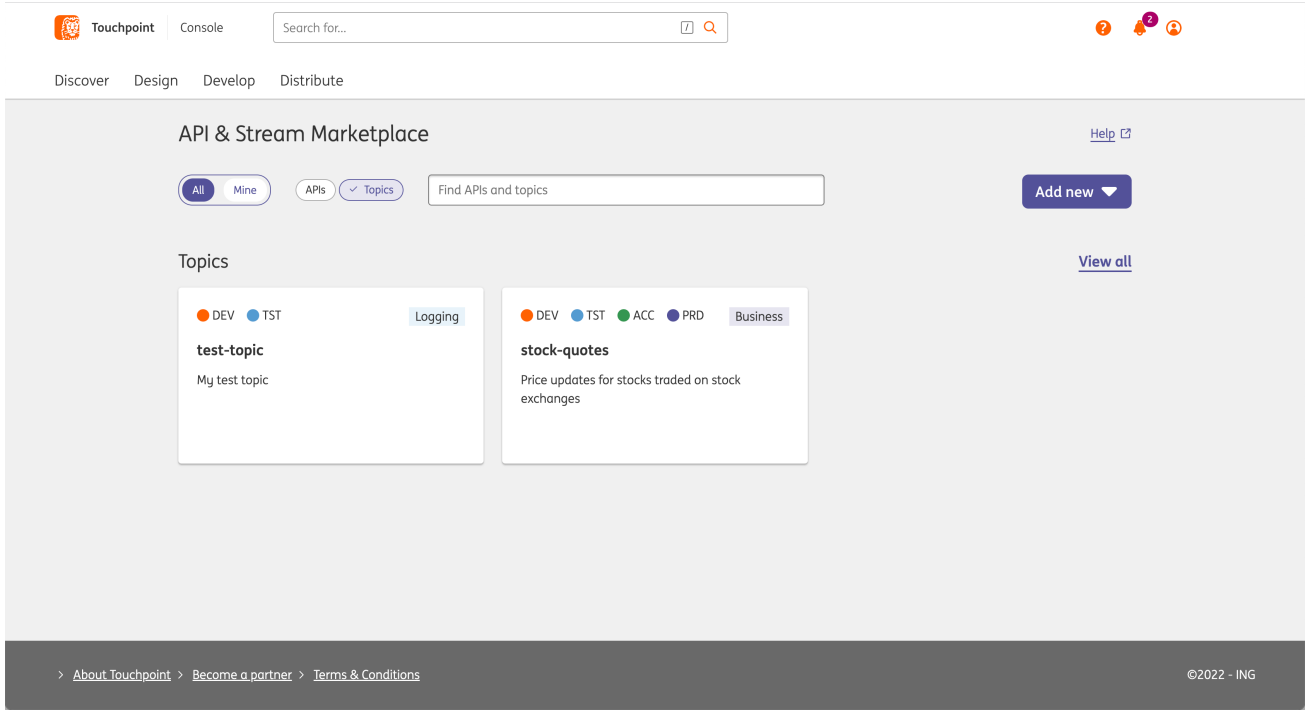

**Disclaimer**: Please note that ING Bank does not have a banking license in the US and is therefore not permitted to conduct banking activities in the US.

# Kafka @ ING

## Frontrunners in Kafka

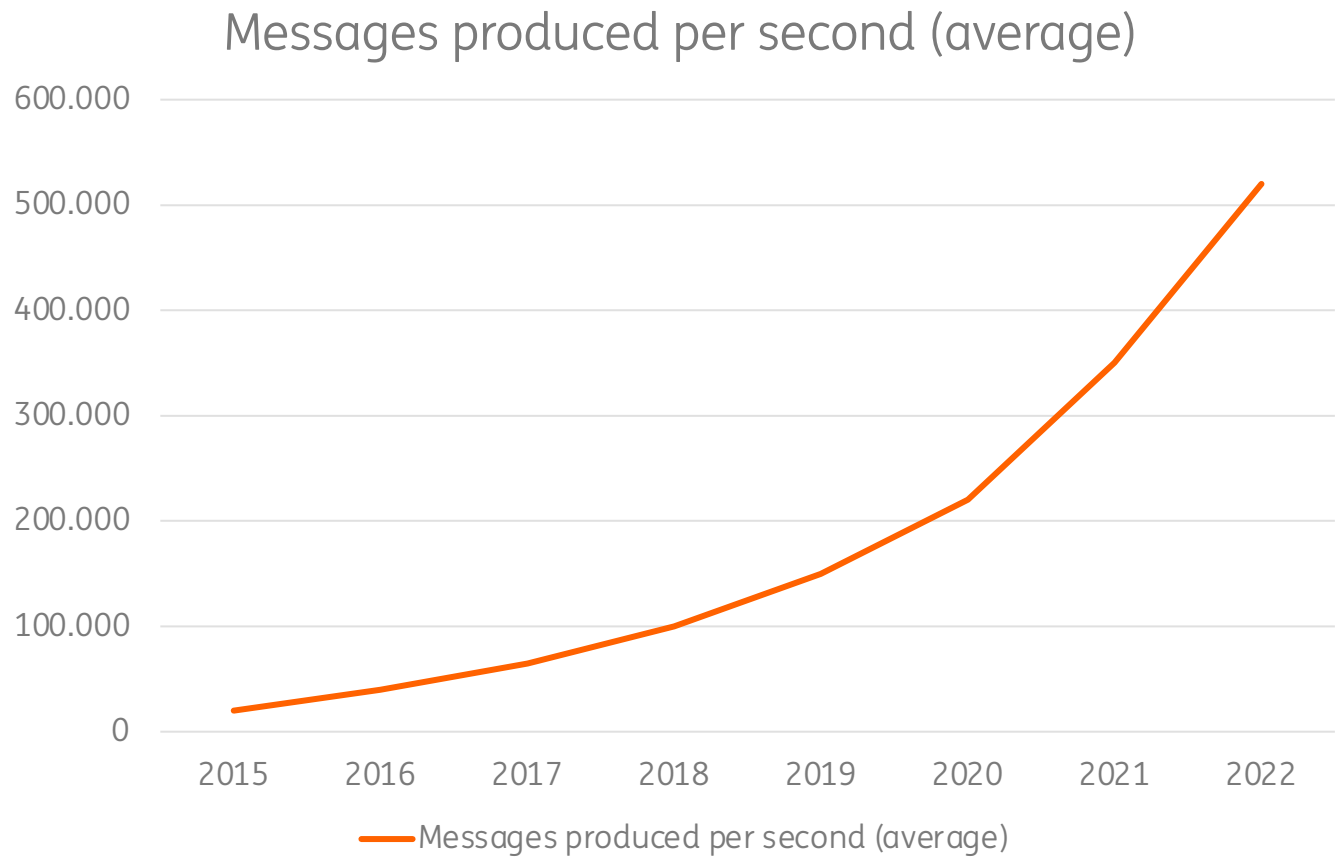
Running in production:

- 7 years
- 5000+ **topics**
- Serving 850+ Development **teams**
- Self service topic management



# Kafka @ ING

## Traffic is growing with 10%+ monthly






ING

# Agenda

- Kafka in a nutshell
- Spring Kafka essentials

- Scenario: Poison pill / Deserialization exceptions
- Scenario: Lack of exception handling

- Testing
- Monitoring

- Wrap-up
- Questions

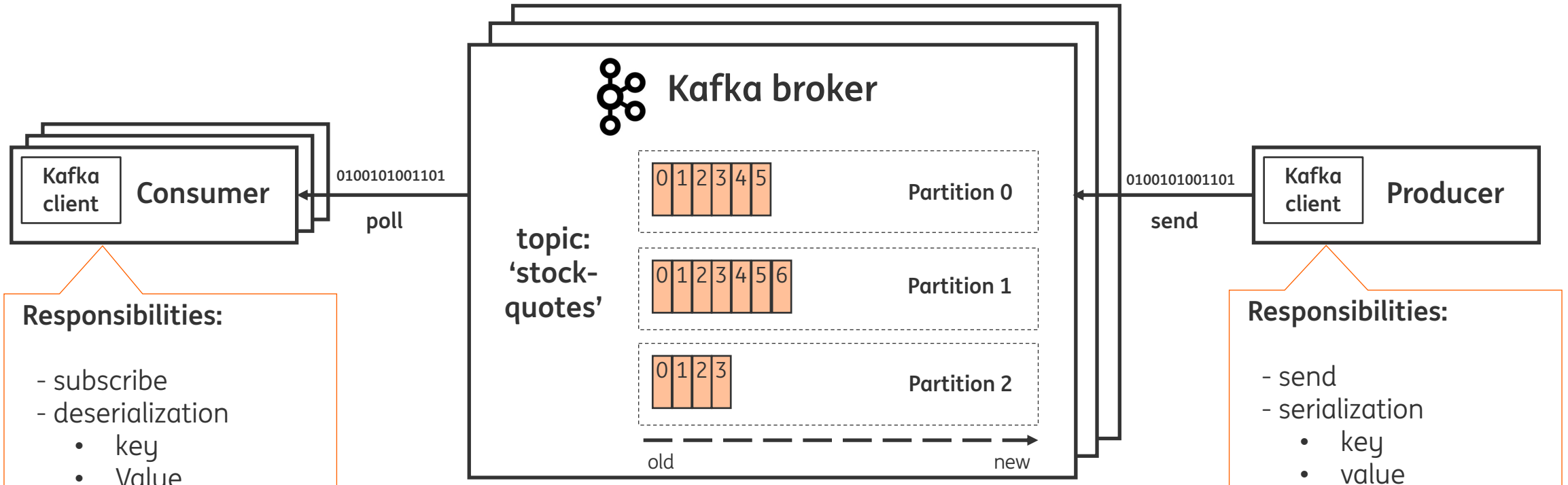
# Kafka in a nutshell

**Responsible for:**

- Append only log
- Distribute
- Replicate

**Not responsible for:**

- Type checking
- Schema validation
- Other constraints




**Responsibilities:**

- subscribe
- deserialization
  - key
  - Value
- heartbeat

**Responsibilities:**

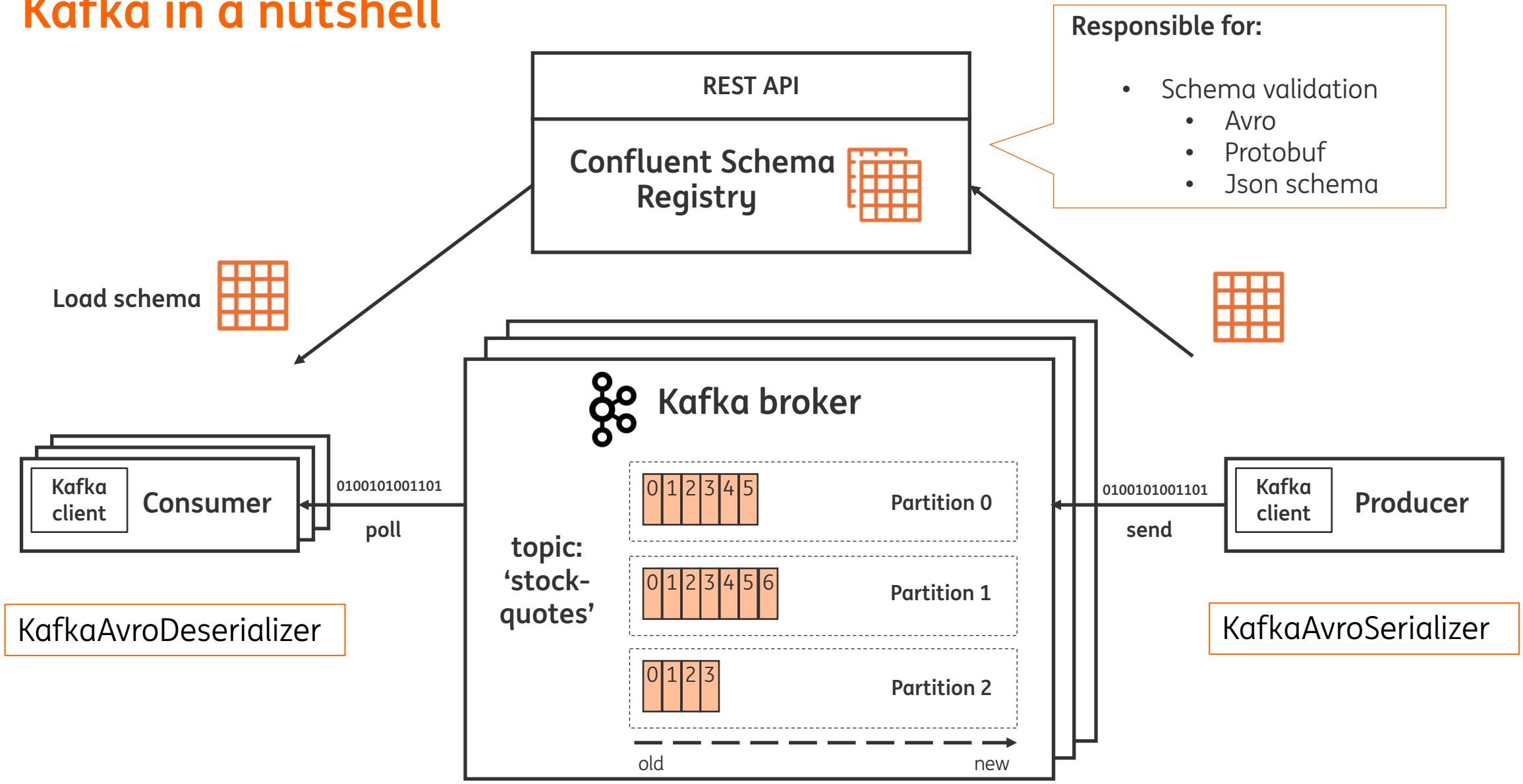
- send
- serialization
  - key
  - value

Data in a Kafka topic are just stored as bytes!

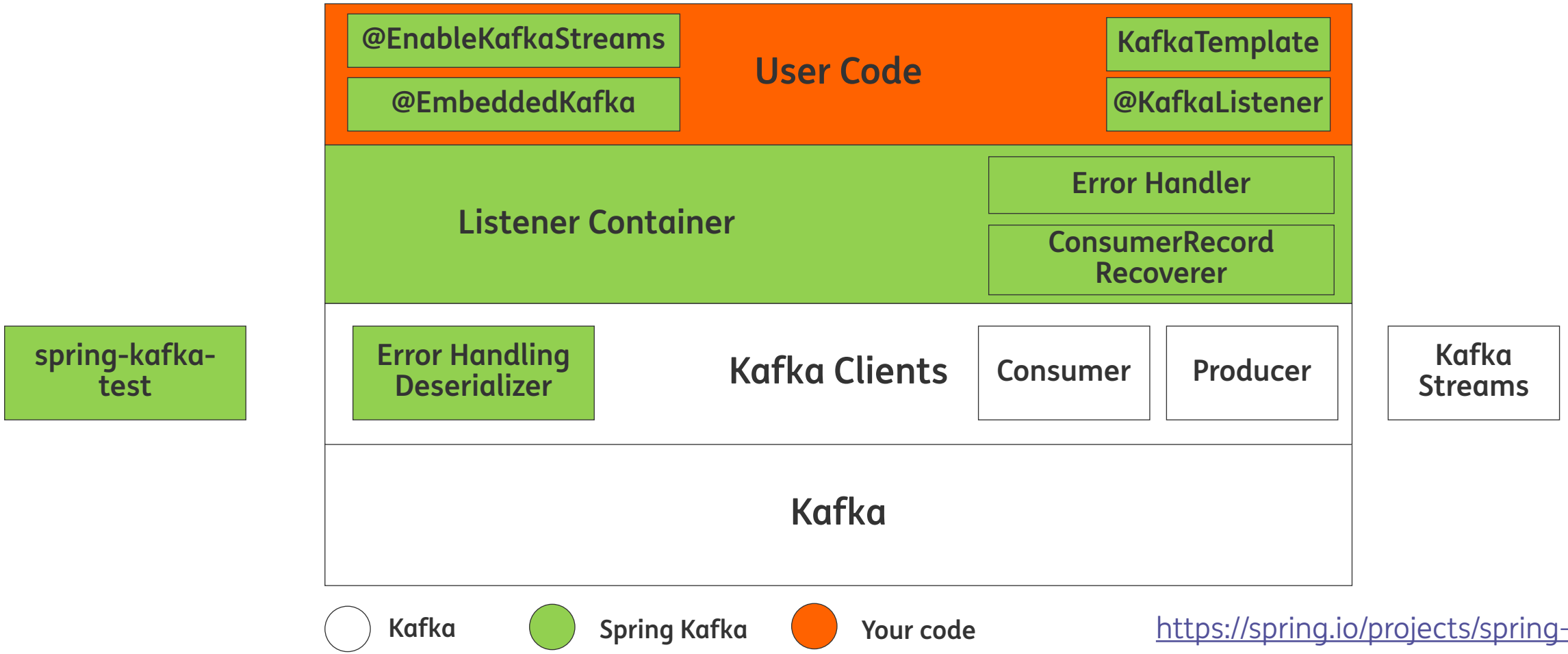


ING

# Kafka in a nutshell



# Spring for Apache Kafka - essentials




https://spring.io/projects/spring-kafka



ING

# Spring Kafka & Spring Boot - Producer

```java
@Component
@Slf4j
public class StockQuoteProducer {

    @Autowired
    private KafkaTemplate<String, StockQuote> kafkaTemplate;

    public void produce(StockQuote stockQuote) {
        kafkaTemplate.send("stock-quotes", stockQuote.getSymbol(), stockQuote);
        log.info("Produced stock quote: {}", stockQuote);
    }
}
```

# Spring Kafka & Spring Boot - Consumer

```
@Component
@Slf4j
public class StockQuoteConsumer {

    @KafkaListener(topics = "stock-quotes")
    public void on(StockQuote stockQuote,
                   @Header(KafkaHeaders RECEIVED_PARTITION_ID) String partition) {
        log.info("Consumed from partition: {} value: {}", partition, stockQuote);
    }
}
```

# Spring Kafka & Spring Boot – Kafka streams

```
@Configuration
@EnableKafkaStreams
public class KafkaStreamsConfig {

    @Bean
    public KStream<String, StockQuote> kStream(StreamsBuilder streamsBuilder) {

        KStream<String, StockQuote> branchedStream = new KafkaStreamBrancher<String, StockQuote>()
            .branch((key, value) -> value.getExchange().equalsIgnoreCase("NYSE"), kStream -> kStream.to("stock-quotes-nyse"))
            .branch((key, value) -> value.getExchange().equalsIgnoreCase("NASDAQ"), kStream -> kStream.to("stock-quotes-nasdaq"))
            .branch((key, value) -> value.getExchange().equalsIgnoreCase("AMS"), kStream -> kStream.to("stock-quotes-ams"))
            .defaultBranch(kStream -> kStream.to("stock-quotes-exchange-other"))
            .onTopOf(streamsBuilder.stream("stock-quotes"));

        return branchedStream;
    }
}
```

# Spring Kafka & Spring Boot – Configuration (application.yml)

```yaml
spring:
  application:
    name: producer-application

  kafka:
    bootstrap-servers: localhost:9092

    producer:
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer
      client-id: ${spring.application.name}

    properties:
      schema.registry.url: http://localhost:8081
```

# After the honeymoon phase is over

Traps

Lessons learned
The hard way!

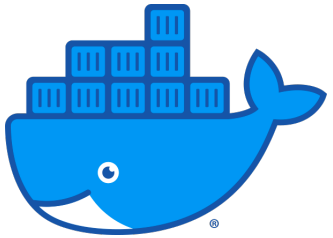
Tips & Tricks

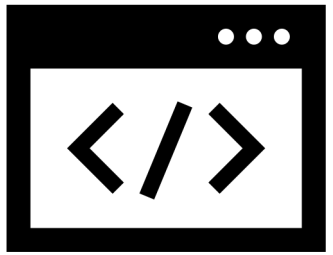
Apply in your own project(s)

Share with your fellow developers

# 🚀 Tip: Local development setup

- **Docker**
  - Kafka Cluster
  - Zookeeper
  - Confluent Schema Registry
- **CLI Tools**
  - Kafka CLI, Confluent CLI, Kafka cat
- **UI Tools**
  - Confluent Control Center
  - Kafka UI
  - Conduktor
  - etc





# 🧨 Scenario: ‘Poison Pill’ (Deserialization exception)

## What is a Poison Pill?

A **record** that **always fails** when **consumed**,
no matter how many times it is attempted.

## Different forms:

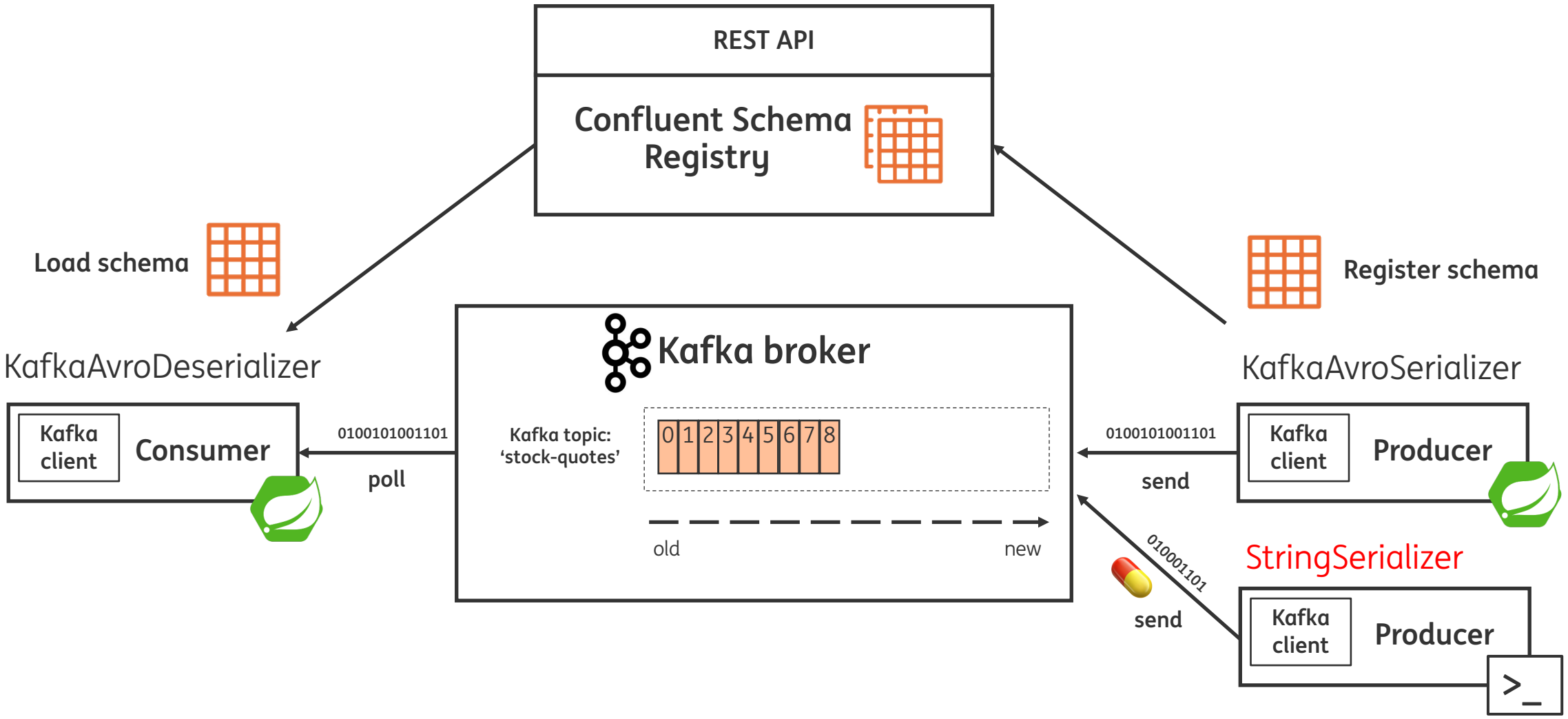
**Corrupted record**

**Deserialization failure**



ING

# Scenario: ‘Poison Pill’ (Deserialization exception)





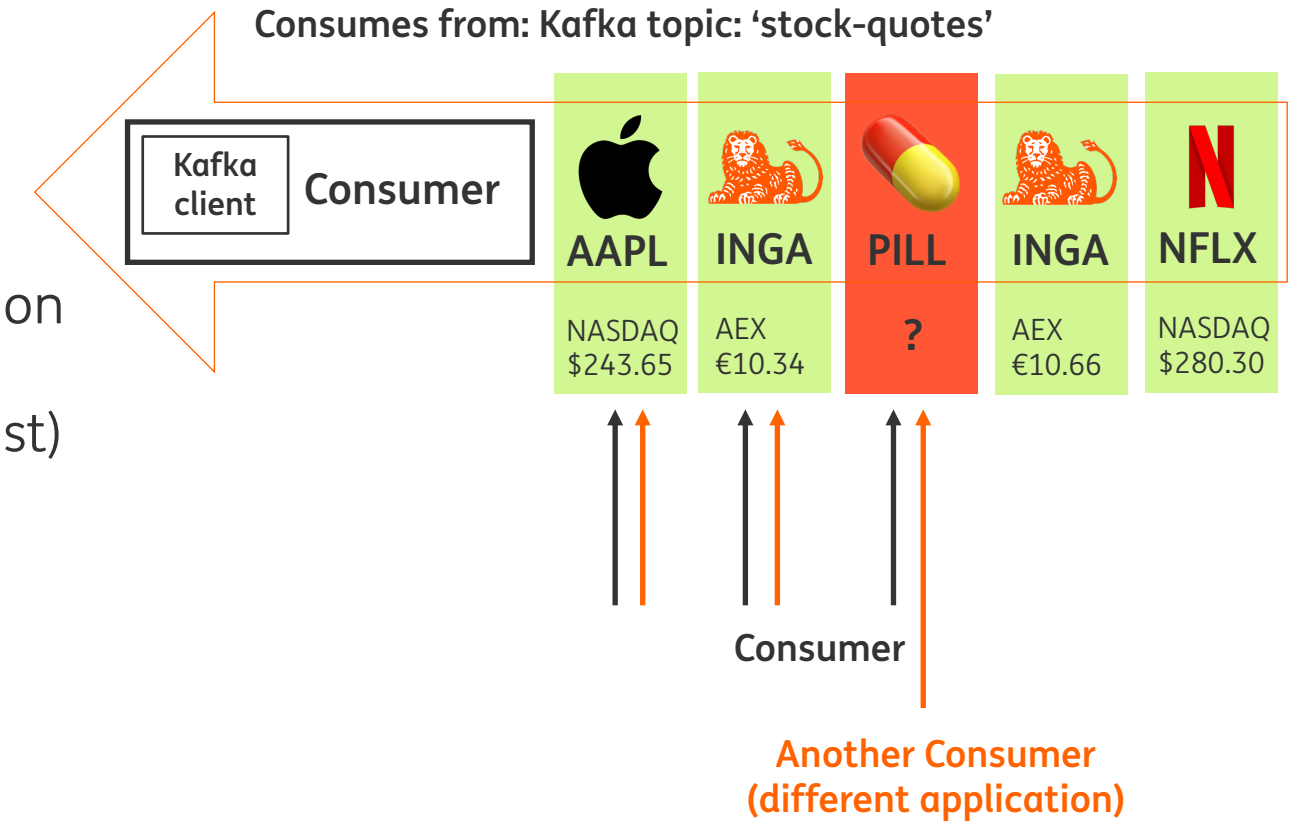

ING

# Scenario: 'Poison Pill' (Deserialization exception)

**Scenario:**

- **Consumer** of topics
- Someone **produced** a '**poison**' message
  - Consumer fails to deserialize

- **Consequence**
  - Blocks consumption of the topic/partition
  - application can't 'swallow the pill'
  - Try again and again and again (very fast)
  - Log line for every failure



# 🧨 Scenario: 'Poison Pill' (Deserialization exception)

- **Result**: log file will grow very fast, flood your disc
- **Impact**: High

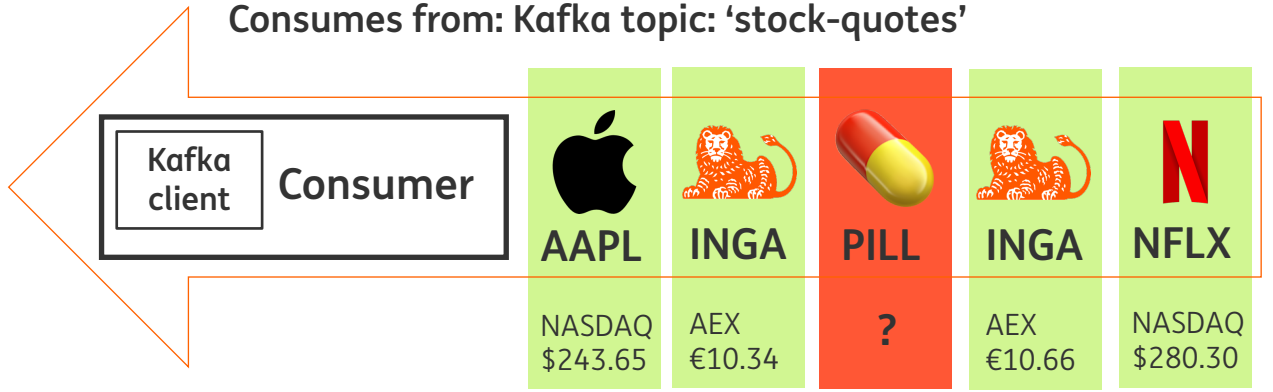
- **How to survive this scenario?**

👎
- Wait until the retention period of the topic has passed

👎
- Change consumer group

👎
- Manually / Programmatically update the offset

👍
- Configure ErrorHandlingDeserializer (Provided by Spring Kafka)






ING

# Scenario: ‘Poison Pill’ - Demo

# 🧨 Scenario: Lack of proper exception handling

**Scenario:**

- Consumer
- Exception is thrown in the method handling the message

**Result (by default)**

- Records that fail are:
  - Retried
  - Logged
  - We move on to the next one

**Consequence**

- You lose that message!
- Not acceptable in many use-cases!

```
@KafkaListener(topics = "stock-quotes")
public void on(StockQuote stockQuote) {
   if ("KABOOM".equalsIgnoreCase(stockQuote.getSymbol())) {
      throw new RuntimeException("Whoops something went wrong...");
   }
}
```

#  Scenario: Lack of proper exception handling

**Impact**

- Dependents on your use-case

**How to survive this scenario?**

- Replace / configure DefaultErrorHandler by:
  - CommonLoggingErrorHandler
  - ContainerStoppingErrorHandler
- Use backoff strategy for recoverable exceptions
- Configure ConsumerRecordRecoverer
  - Dead letter topic
  - Implement your own recoverer

# Scenario: Lack of proper exception handling - Demo



ING

# Testing

- Support for Integration testing **@EmbeddedKafka**

```
@EmbeddedKafka
@SpringJUnitConfig
public class EmbeddedKafkaIntegrationTest {

    @Autowired
    private EmbeddedKafkaBroker embeddedKafkaBroker;

    @Test
    public void yourTestHere() throws Exception {
        // TODO implement ;)
    }
}
```



ING

# Testing – Alternatives to EmbeddedKafka

- Test containers
- Focus on unit test first
- Kafka streams
  - Topology test driver

# Testing – Test topology test driver

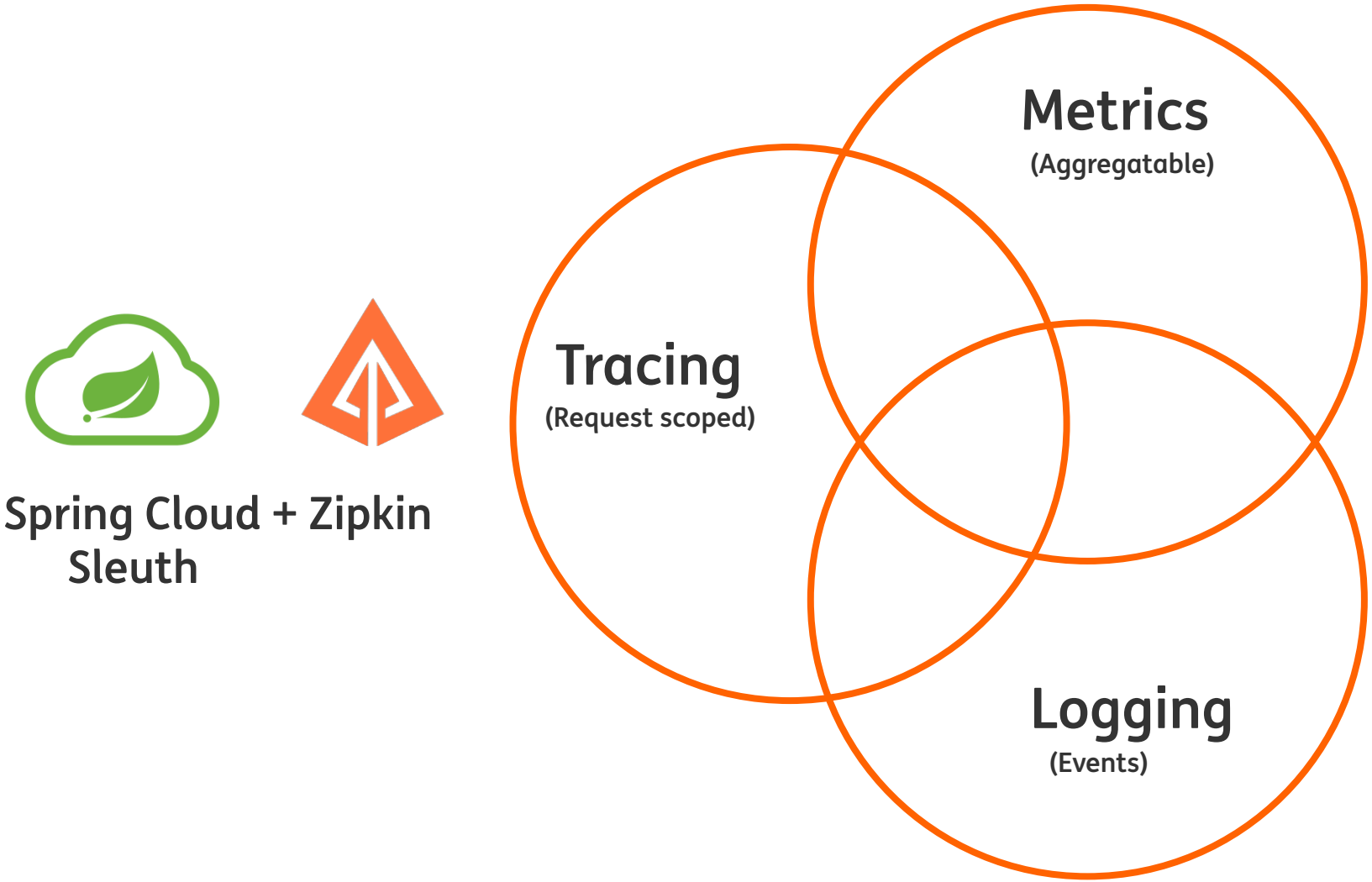
```
@Test
void stockQuoteFromAmsterdamStockExchangeEndUpOnTopicQuotesAmsTopic() {
    StockQuote stockQuote = new StockQuote("INGA", "AMS", "10.99", "EUR",
                                           "Description", Instant.now());

    stockQuoteInputTopic.pipeInput(stockQuote.getSymbol(), stockQuote);

    assertThat(stockQuoteAmsOutputTopic.isEmpty()).isFalse();
    assertThat(stockQuoteAmsOutputTopic.getQueueSize()).isEqualTo(1L);
    assertThat(stockQuoteAmsOutputTopic.readValue()).isEqualTo(stockQuote);

    assertThat(stockQuoteNyseOutputTopic.isEmpty()).isTrue();
    assertThat(stockQuoteNasdaqOutputTopic.isEmpty()).isTrue();
    assertThat(stockQuoteOtherOutputTopic.isEmpty()).isTrue();
}
```

# Monitoring - Three Pillars observability



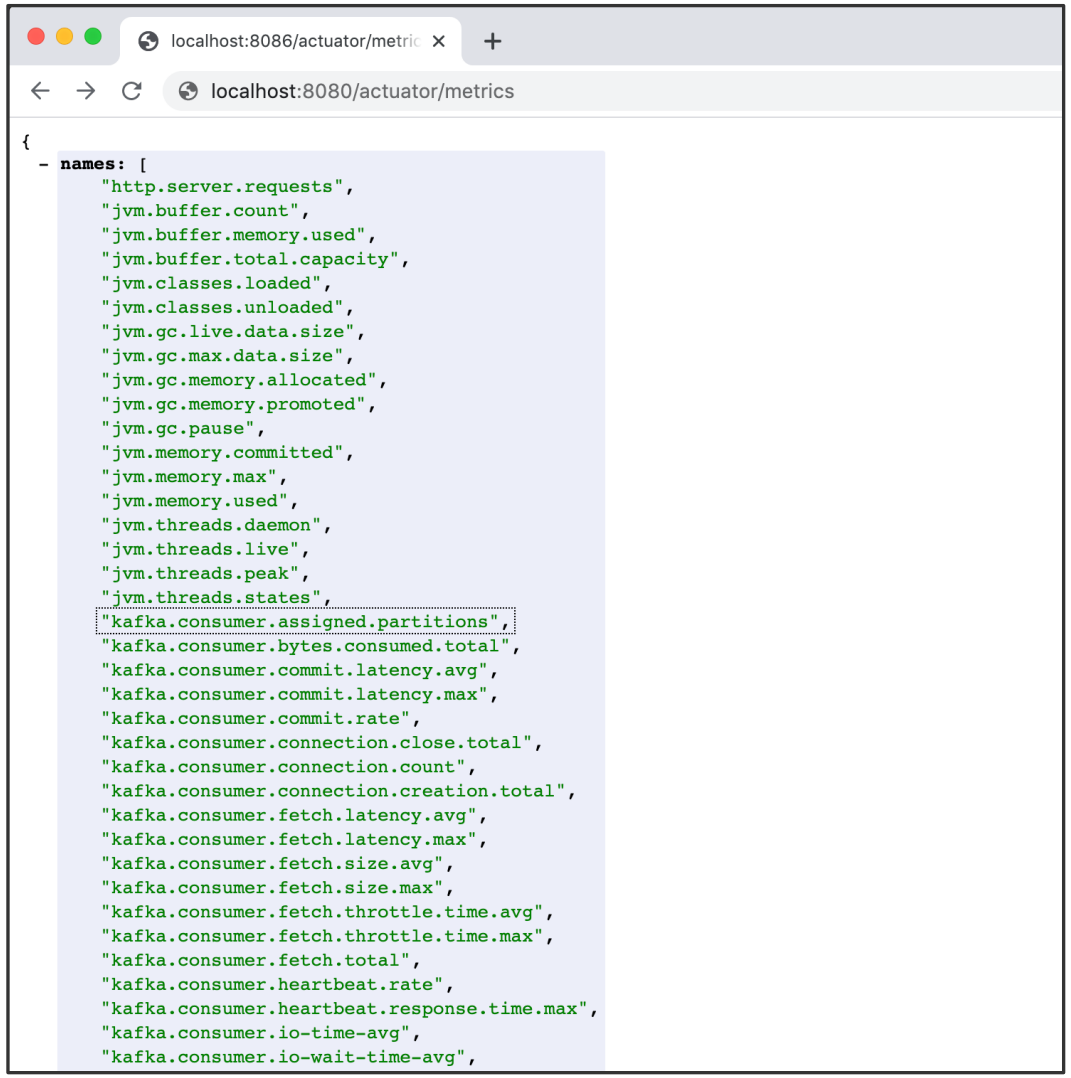

Micrometer + Prometheus

Spring Cloud + Zipkin Sleuth

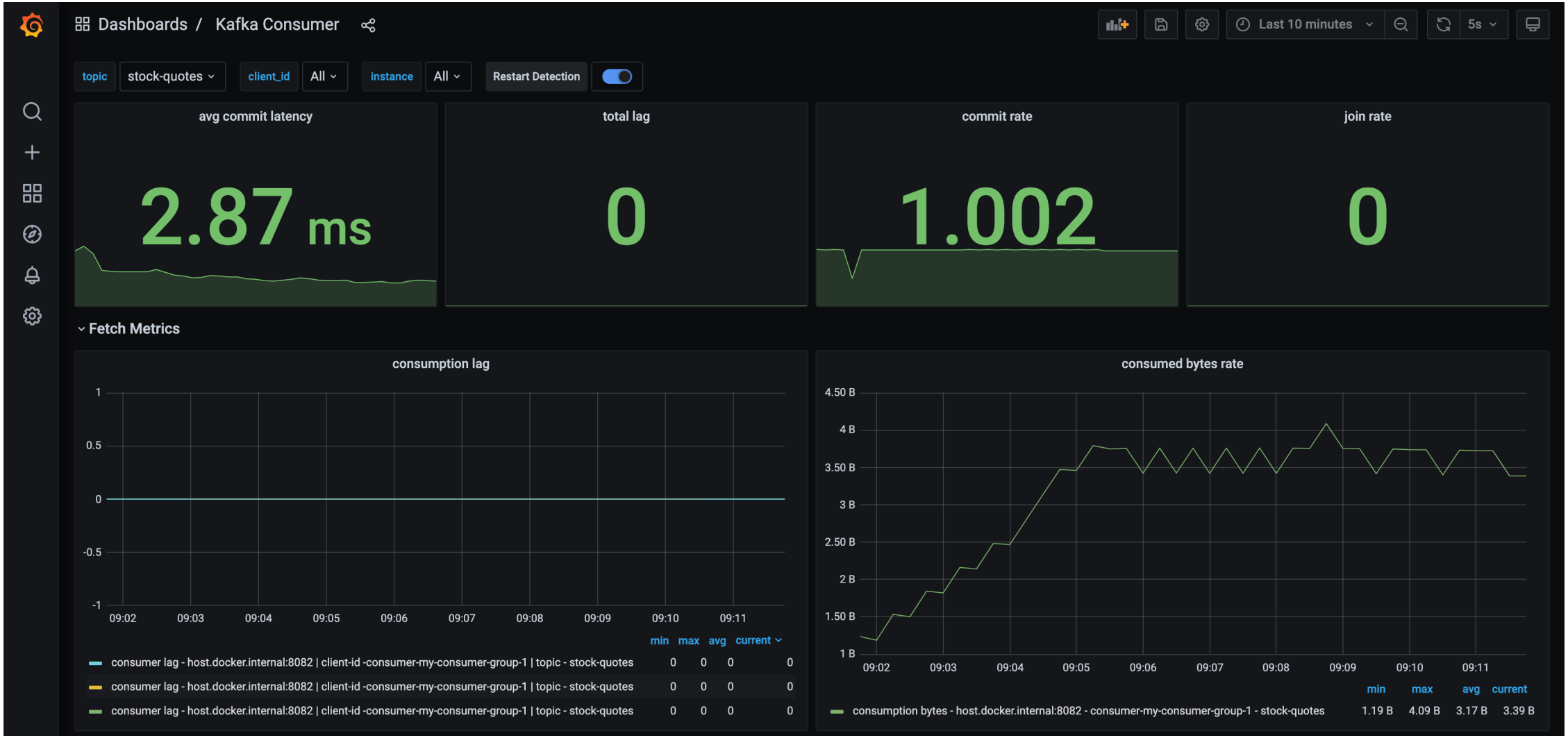
# Monitoring – Metrics

- **Out of the box Kafka metrics**
  - consumers
  - producer
  - streams

- Micrometer
- Spring Boot Actuator Metrics

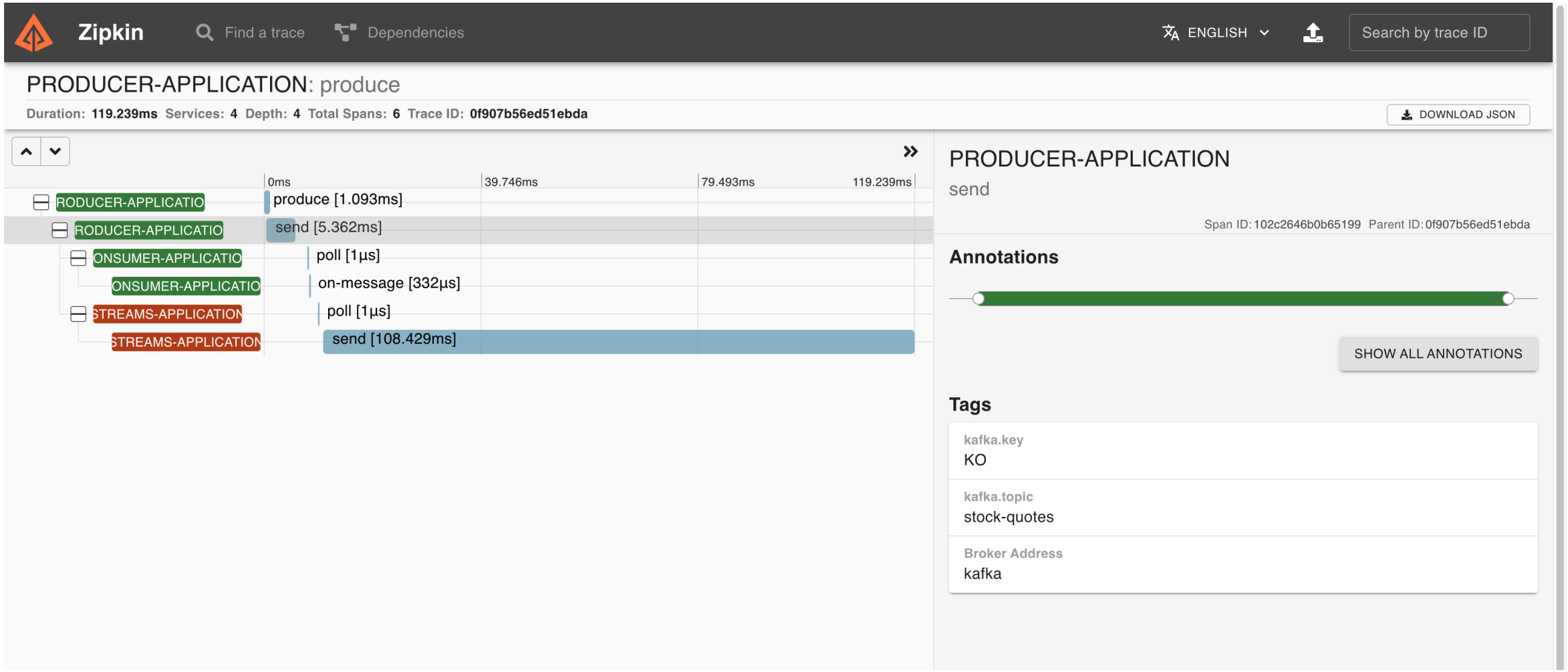


# Monitoring – Metrics



# Monitoring – Distributed tracing

# Monitoring – Distributed tracing



# 🎓 Lessons learned

- Invest time in your **local development** environment (fast feedback loop)
- Use **Spring Kafka** but also understand the core **Kafka APIs**
- Consumer:
  - Expect the unexpected!
    - Handle Deserialization exceptions a.k.a. ‘poison pills’
    - Proper exception handling
    - Validate incoming data
- Producer:
  - Don’t change your serializers
  - Leverage Apache Avro + Confluent Schema registry
  - Don’t break compatibility for your consumers!

# 🎓 Lessons learned

- Security
  - Who can produce data to your topics?
  - Who can consume?

- Monitor your topics, consumer groups & applications in production
  - Micrometer
  - Spring Cloud Sleuth

- Don’t overdo integration test!

# Questions

# Thanks for joining my talk!

@TimvanBaarsen


https://www.medium.com/@TimvanBaarsen

https://github.com/j-tim/spring-io-barcelona-2022-spring-kafka-beyond-the-basics