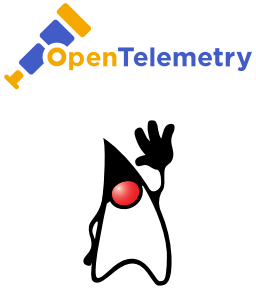


OpenTelemery Tracing: Od zera do Bohatera

Sebastian Kozak

Plan Na tą prezentację

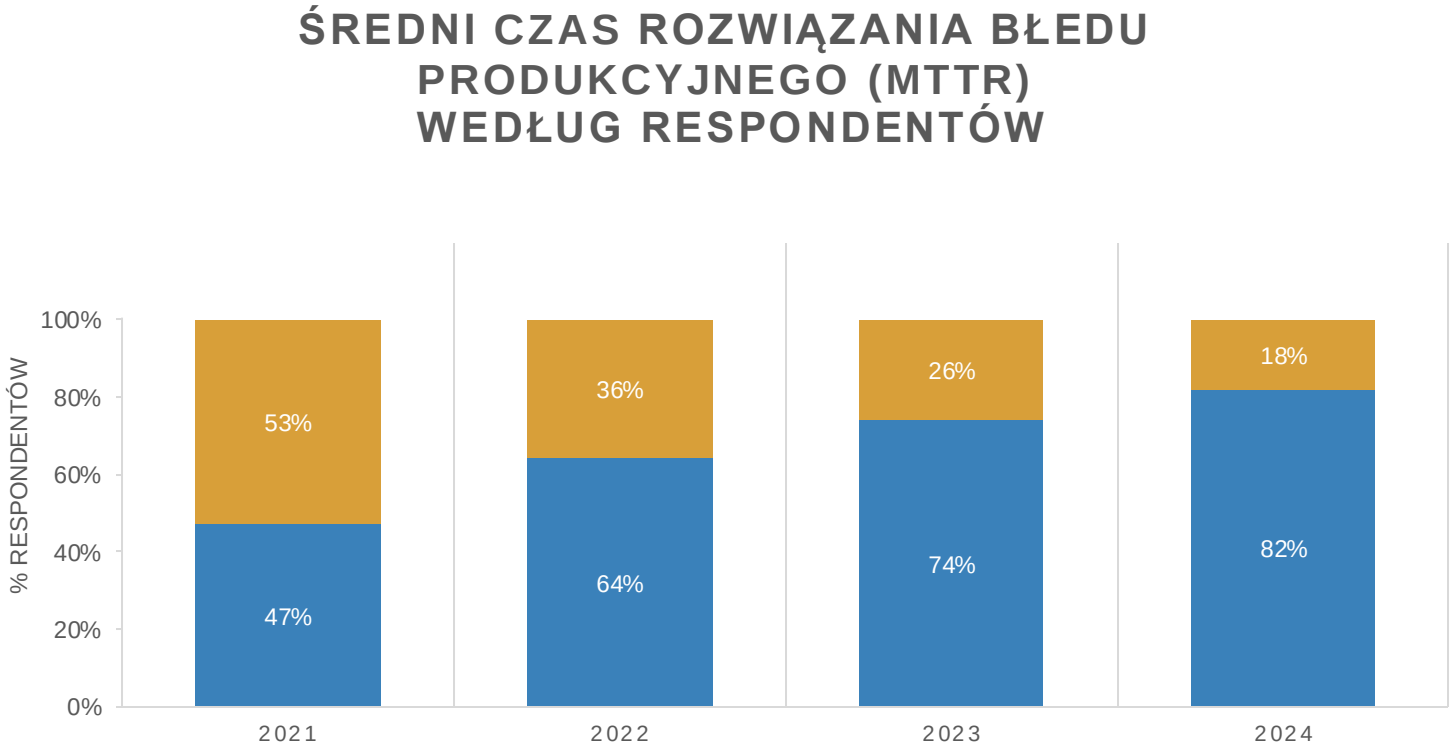
Cz. 1 ~ 50’

- Problem namierzenia problemu w systemach rozproszonych
- Czym jest Tracing i jak działa
- Dlaczego OpenTelemetry?
- Jak działa agent jvm + Demo
- Krótkie Demo tracingu w ELK
- Przykłady z pola bitwy
- Jak tego użyć w Twoim projekcie (agent)

Cz. 1 ~ 40’

- Jak tego użyć w Twoim projekcie (sdk/api)
- Problem za dużej ilości danych I jak to rozwiązać
- OpenTelemetry collector: Co to jest i po co mi to potrzebne?
- Gdzie tracing nie działa z pudełka? I jak to naprawić?
- Tracing jako pomoc w poznawaniu systemu

Czas rozwiązywania błędów Produkcyjnych 2021-2024 według logz.io



Na podstawie danych z
https://logz.io/wp-content/uploads/2022/04/Pulse-Report-2022.pdf
https://logz.io/devops-pulse-2023/
https://logz.io/observability-pulse-2024/#executive-summary

Coraz bardziej skomplikowane systemy, więcej ruchomych części

Trudność namierzenia przyczyny

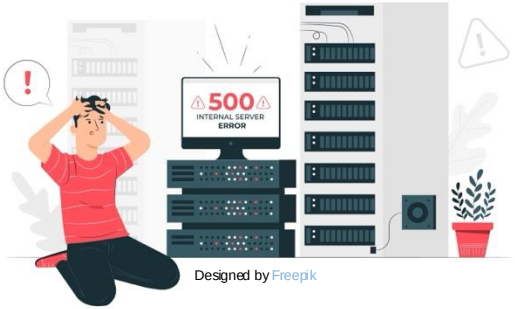
Jakie możemy mieć błędy w systemie?

Jawne (Explicit)

- Coś nie działa, dostajemy 5xx, timeout, NPE…

Ukryte (Implicit)

- Coś działa wolno (czasami)
- Coś działa źle (czasami)

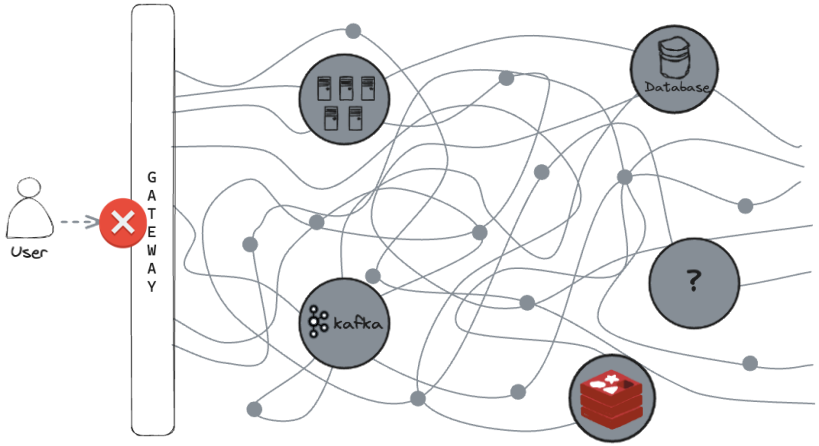


Problem Jawny:

http 500: Internal Server Error

500: Internal server error

Nasz system



Połączenia/requesty

Serwis / DB / Cache / Kolejka itp..

Dostaliśmy zgłoszenie o błędzie i co dalej?

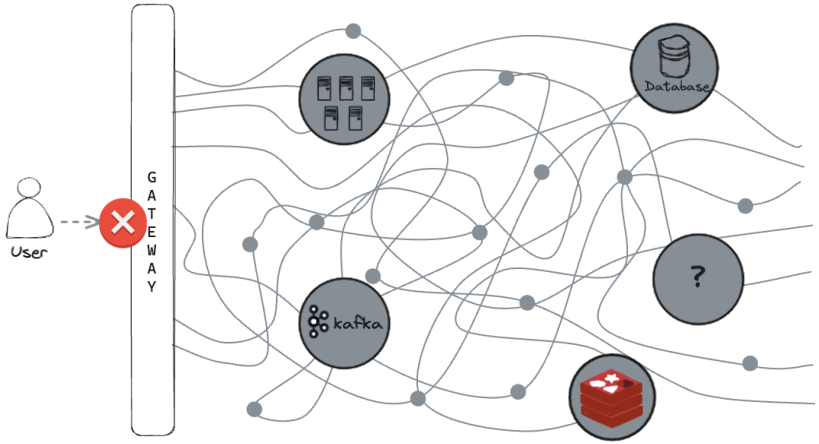
- Gdzie nie działa?
- Co nie działa

- Trzeba wiedzieć gdzie szukać
- Miejsc, gdzie może pójść nie tak może być wiele
- Czy na pewno ten błąd, który jest w logach dotyczy mojego przypadku ?
- Syzyfowa praca powtarzana w kółko przy każdym błędzie

= $$

jak ułatwić sobie życie i skrócić czas szukania błędu?

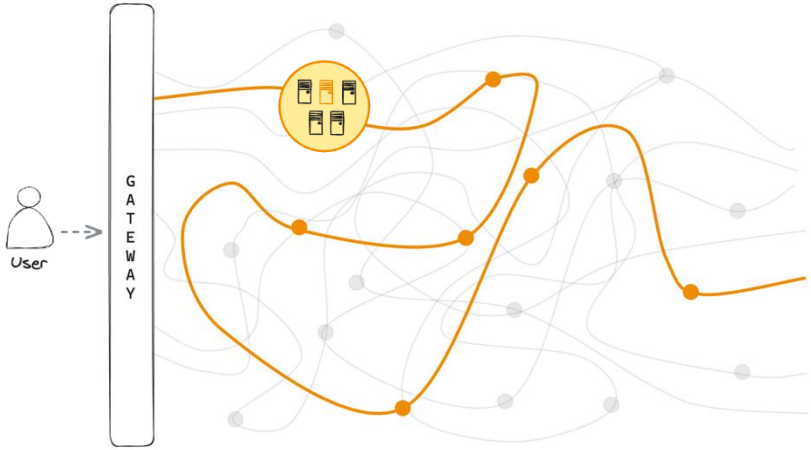
Nasz system



Połączenia/requesty

Serwis / DB / Kolejka itp..

A gdyby tak wprowadzić unikalny identyfikator zapytania, Np. TraceId?



Indywidualna ścieżka naszego zapytania

Serwis / DB / Kolejka … przetwarzające zapytanie

Co zyskujemy ?

- Odsianie szumu
- Skoncentrowanie się na danych skorelowanych z problemem np. logach

Problem Ukryty:
System działa wolno/czasami

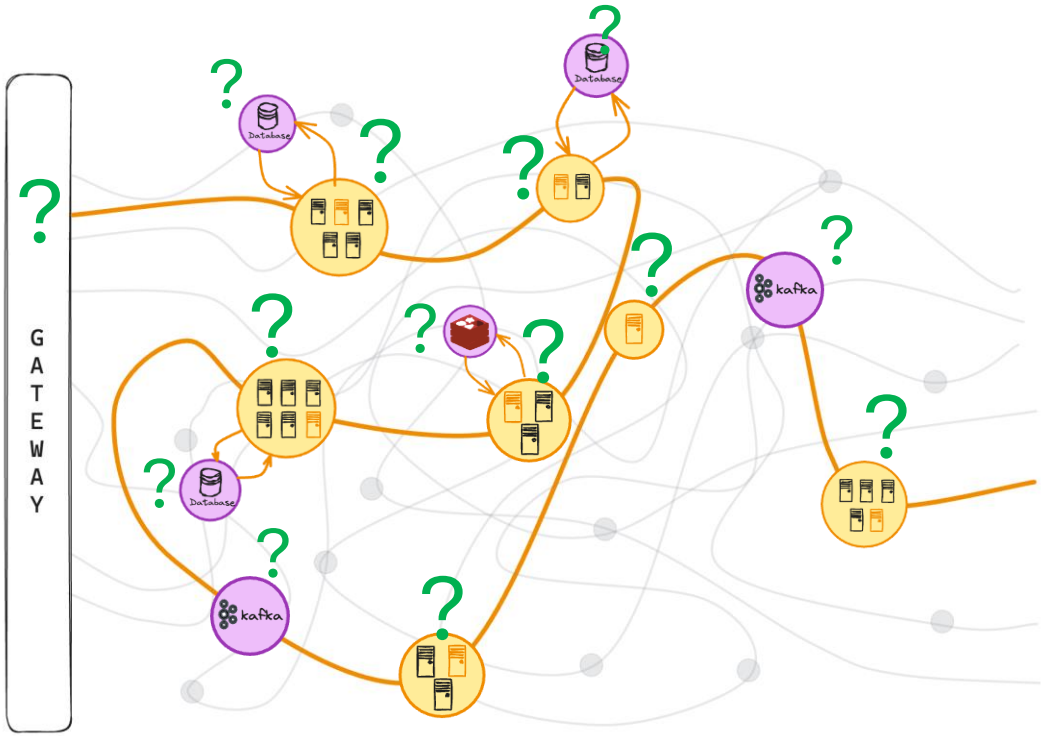
System działa wolno

- U mnie działa
- Klientowi działało wolno o 12,
  my sprawdziliśmy o 14 I było ok
- Załóżmy, że w zgłoszeniu dostalśmy traceid
  - Analizować timestampy w logach
  - Mamy za mało logów
  - Dodać więcej logów
    + deploy
    + próba odtworzenie sytuacji
  - Dodanie jakichś logów start/koniec
  - Da się, ale jakim kosztem?
    Powtarzamy to przy każdym błędzie
- Na jakie pytanie szukamy odpowiedzi ?

  Gdzie jest wolno?

  Dlaczego jest wolno?

trace ze zgłoszenia

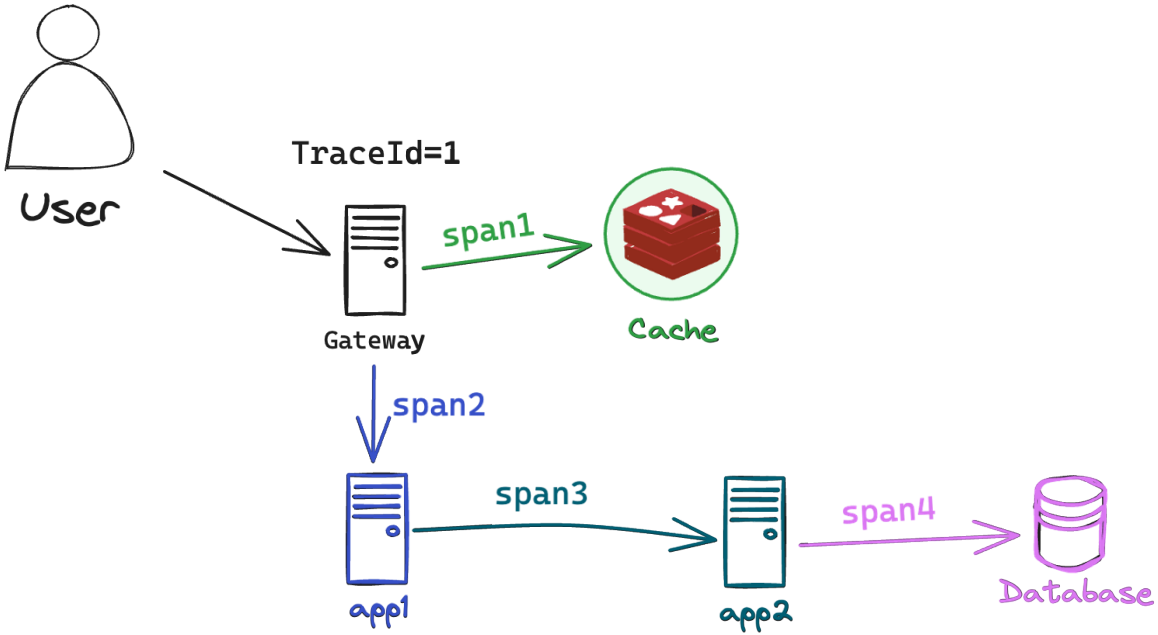


Jak ustalić gdzie jest wolno i dlaczego jest wolno?

Mamy już wyizolowany trace (flow przez nasz system).

Widać jednak, że potrzebujemy dodatkowego wymiaru.

A gdyby tak oprócz traceId, również oznaczyć operację zewnętrzną (spanId)?

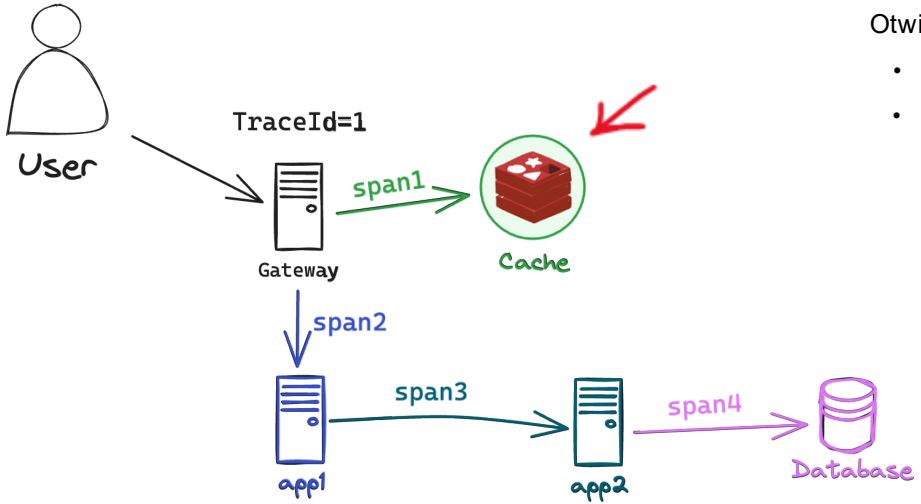


Co nam to daje ?

Dla każdego spanu możemy indywidualnie oznaczyć

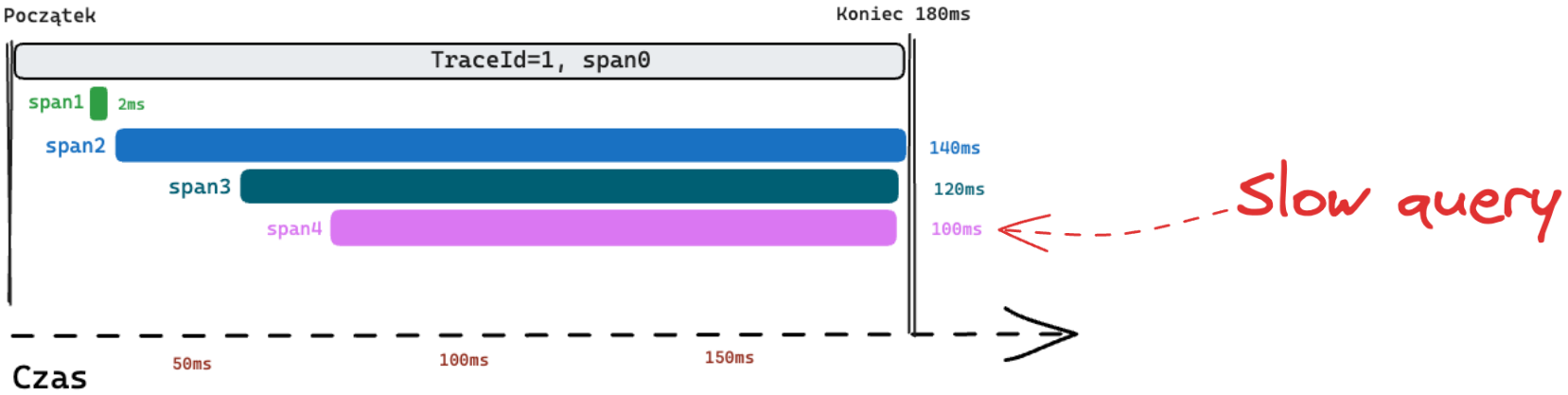
- Start Operacji
- Koniec Operacji
- Dodatkowe dane kontekstowe:
  - Zapytanie do bazy danych
  - Dane HTTP
    (Nagłówki, url, parametry)
  - Dane kafki
    (topik, key, offset wiadomości)
  - Informacje o wątku (nr wątku)
  - Informacje domenowe np. UserId, TenantId

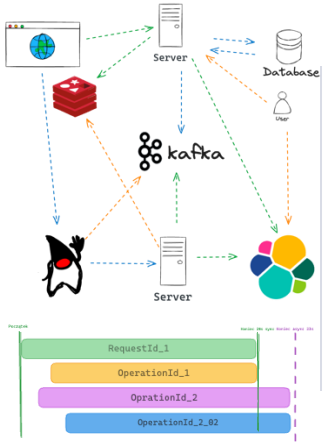
Mamy indywidualne operacje w ramach trace i co dalej ?



Otwiera to nam nowe możliwości:

- Wizualizacja całego flow (timeline)
- Interakcję z zebranymi danymi np.
  - Pokaż mi trace, które trwały > 10s oraz brał w nich udział userId = 2 w piątek
  - Pokaż logi operacji =spanId4 & trace=1
  - Pokaż mi operacje, które trwają najdłużej w moim systemie wraz z parametrami
  - …




Server
Database
User
kafka
Server
RequestId_1
OperationId_1
OprationId_2
OperationId_2_02

Tracing

3 fazy tracingu

I
Instrumentacja operacji
(Jak oznaczyć)

- Http
- kolejki
- Bazy danych
- Sockety
- ….

(Attributes)
Np. TraceId, SpanId
service name
start, koniec,
db query,
http parameters,
Kafka topics
Nazwa wątku

II
Propagacja
(Jak to wysłać)

- Między sewisami app
- Wewnątrz aplikacji
- Do serwera wizualizacji

III
Rekonstrukcja z wizualizacja
(Jak to połączyć I zaprezentować)

- Server analizujący
- Wizualizacja
- Język zapytań
- Agregacje
- Alerty
- Obserwowanie anomalii
- ….

Dlaczego OpenTelemetry ?

Kiedyś i dziś

Apm/tracing*

Pierwsze podejście do standaryzacji

Drugie podejście do standaryzacji

Jaeger

Elastic
APM (Opbeat)

OpenCensus
(początkowo Google)

Lightstep

Czas

2008+

2016

2019

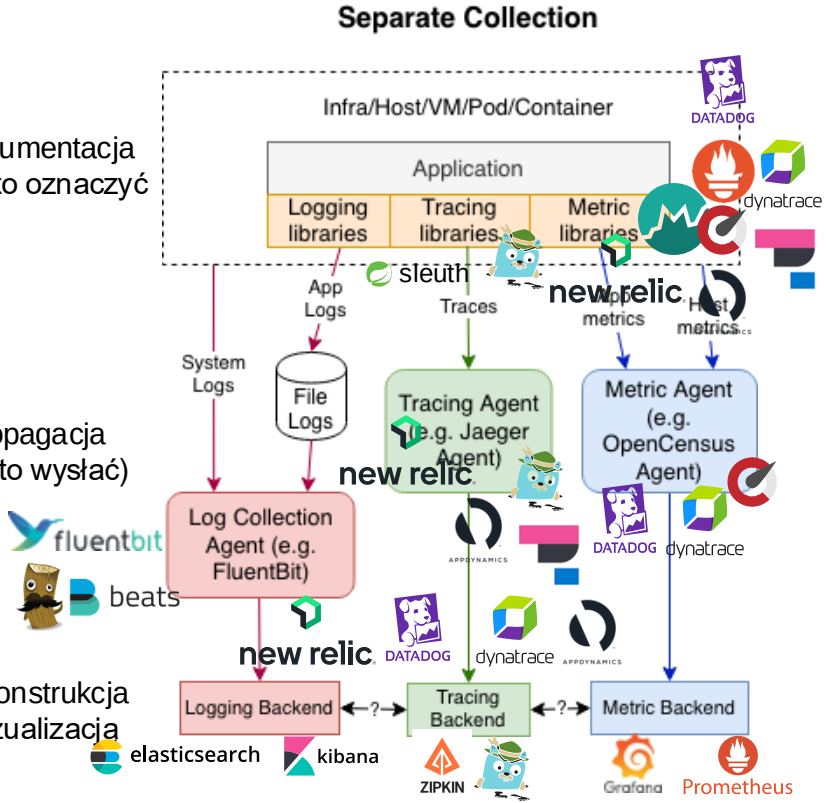
Kiedyś i dziś (logi, tracing, metryki) JVM

Wszystko dostarczane przez vendora/ów

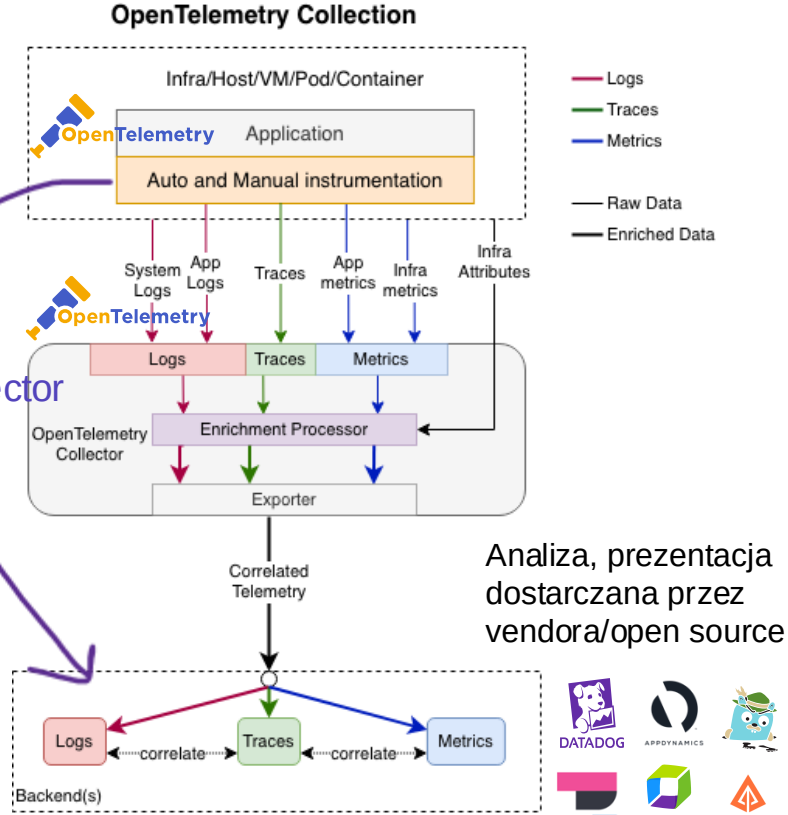
Instrumentacja (Jak to oznaczyć

Propagacja (Jak to wysłać)

Rekonstrukcja wizualizacja



Skip Collector



Analiza, prezentacja dostarczana przez vendora/open source

DATADOG APPDYNAMICS dynatrace ZIPKIN Grafana new relic.

https://github.com/open-telemetry/opentelemetry-specification/tree/main/specification/logs

- Sygnały
  - Traces
  - Metrics
  - Logs
  - Constant profiling
  - RUM

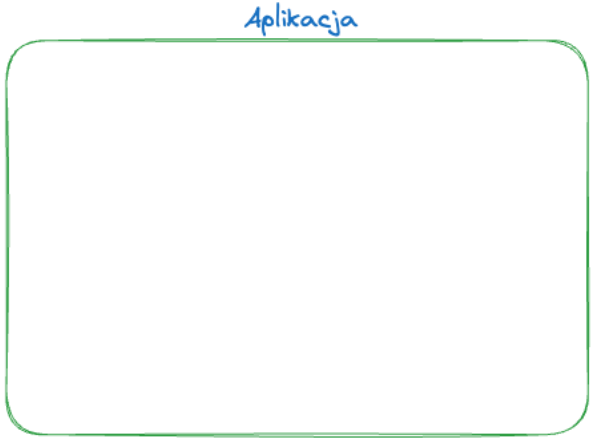
- Spójny Protokół wymiany OTLP
  (pozwala eksperymentować z różnymi rozwiązaniami)
- OpenSource
- Niezależne od vendora
- Bez GUI i analiz
(tutaj wkraczają vendorzy i OpenSource)
- Oficjalne wsparcie dla wielu popularnych języków

Language APIs & SDKs

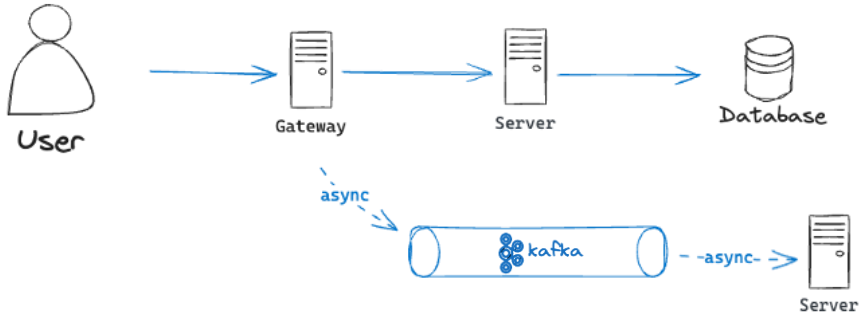
For the development status, or maturity level, of a language API or SDK, see the following table:

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	Alpha	Alpha
Swift	Stable	Development	Development

Jak traceId i spanId są propagowane?



Wewnątrz systemu



Pomiędzy systemami

http: Headery

Mesagging : Headery/metadata

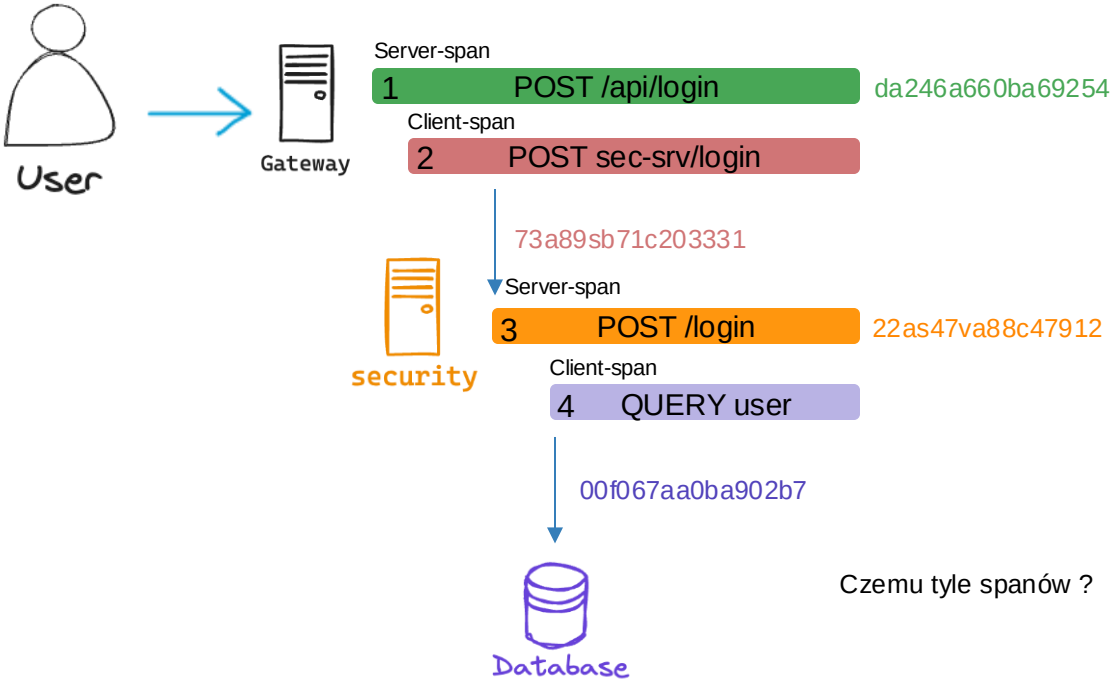
jdbc : connection level / call level atributes

…

Propagacja pomiędzy systemami

traceparent: `0af7651916cd43dd8448eb211c80319c-da246a660ba69254`

- `0af7651916cd43dd8448eb211c80319c` → TraceId
- `da246a660ba69254` → SpanId (parentSpan)



1) Jest pierwszym serwisem, więc tworzy nowy traceId oraz pierwszy SpanId. (traceParent)

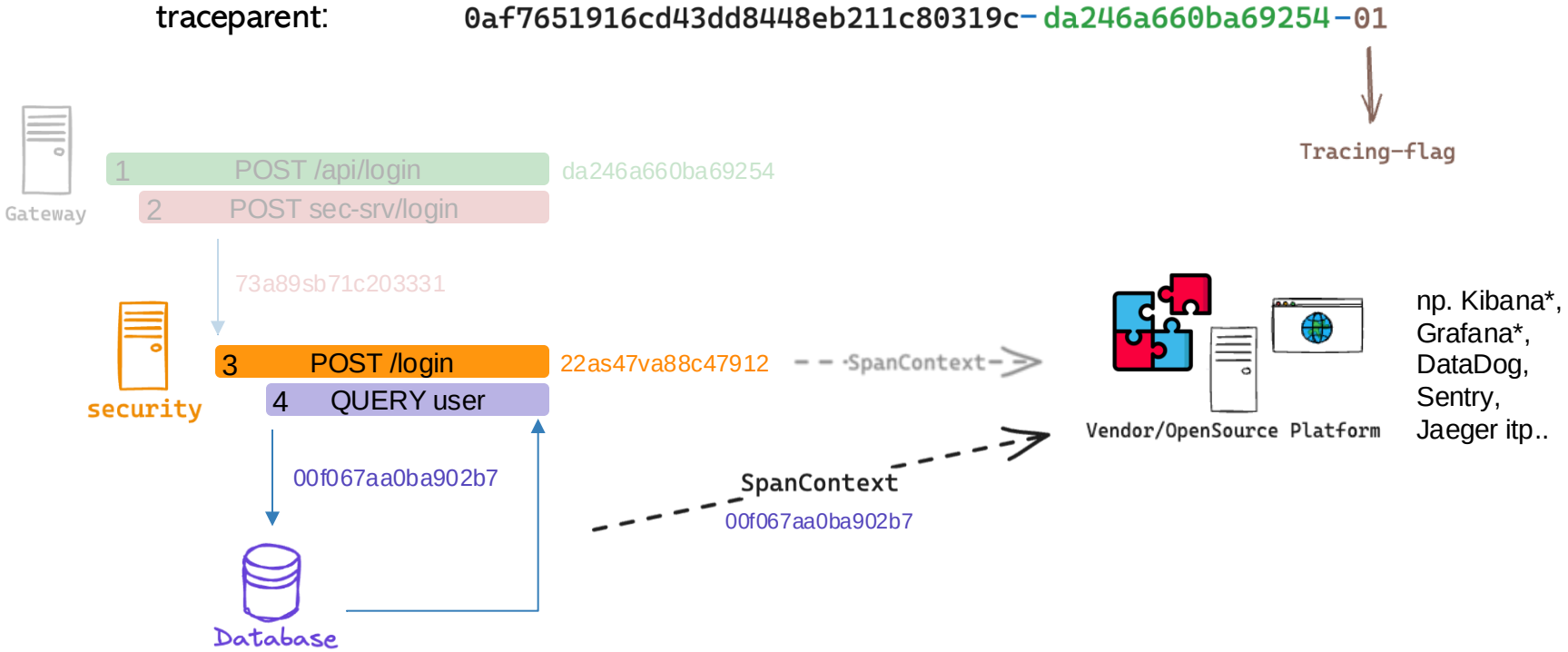
2) Tworzy nowy span operacji http na podstawie ParentSpan

3) Odbiera traceparent
Na podstawie parentSpan tworzy nowy SpanId

4) Tworzy nowy span operacji db na podstawie parentSpan

Czemu tyle spanów ?

Co dalej ze zbieranymi danymi?

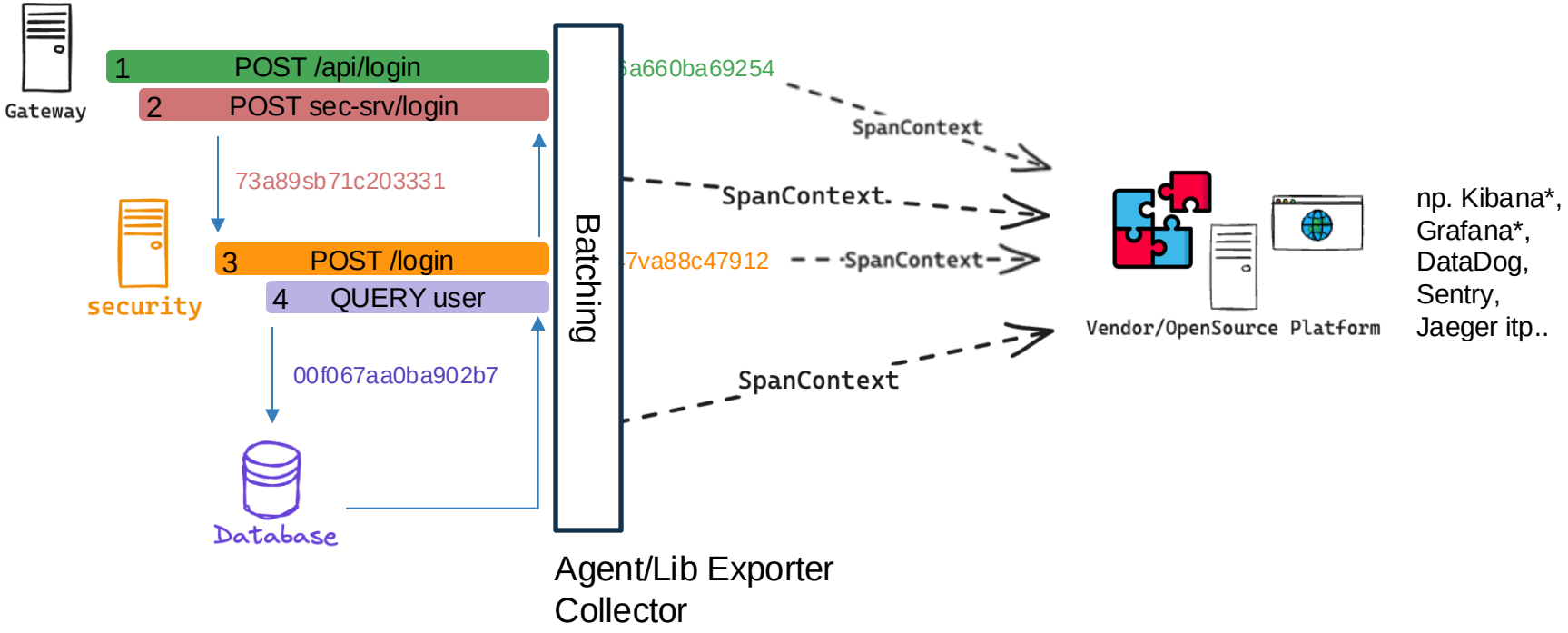


Wewnątrz spanu bazodanowego

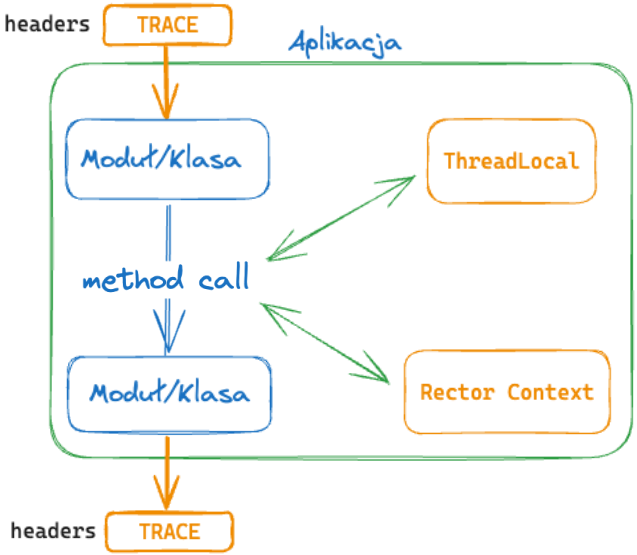

```
{
  "traceId": "0af7651916cd43dd8448eb211c80319c",
  "spanId": "00f067aa0ba902b7",
  "parentSpanId": "22as47va88c47912",
  "name": "find security.userLogin",
  "kind": 3,
  "startTimeUnixNano": "1708196704346846888",
  "endTimeUnixNano": "1708196704350470888",
  "attributes": [
    {
      "key": "db.operation",
      "value": {
        "stringValue": "find"
      }
    },
    {
      "key": "db.mongodb.collection",
      "value": {
        "stringValue": "users"
      }
    },
    {
      "key": "db.connection_string",
      "value": {
        "stringValue": "mongodb://host.docker.internal:27017"
      }
    },
    {
      "key": "db.statement",
      "value": {
        "stringValue": "{\"find\": \"users\", \"filter\": {\"username\": \"johndoe\", \"passwordHash\": \"hashed_password_value\"},
      }
    },
    {
      "key": "thread.id",
      "value": {
        "intValue": "44"
      }
    },
```

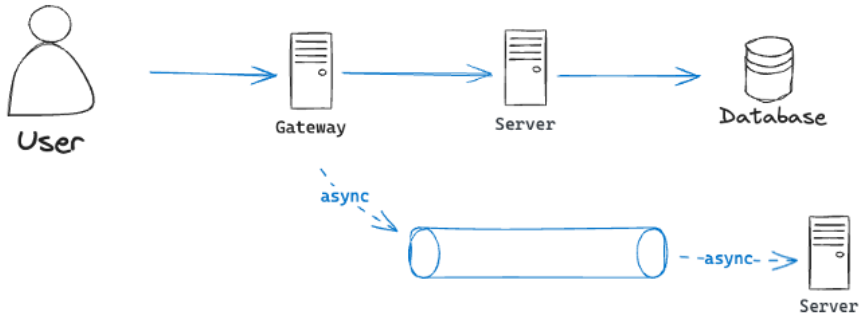
Co dalej ze zbieranymi danymi?

traceparent: `0af7651916cd43dd8448eb211c80319c-da246a660ba69254-01`



Jak te dane są propagowane





Wewnątrz systemu

Thread-per-request: ThreadLocal

Reactor: Context propagation

Pomiędzy systemami

http: Headery

Mesagging : Headery/metadata

jdbc : connection level / call level atributes

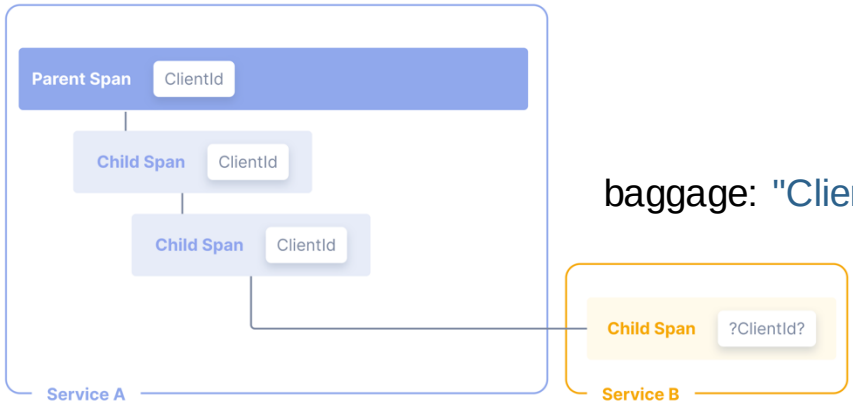
Więcej w linku poniżej: …

https://github.com/open-telemetry/opentelemetry-java-instrumentation/blob/main/docs/supported-libraries.md#libraries--frameworks

Standard propagacji W3C

(TraceContxt) traceparent: `00-0af7651916cd43dd8448eb211c80319c-da246a660ba69254-01`

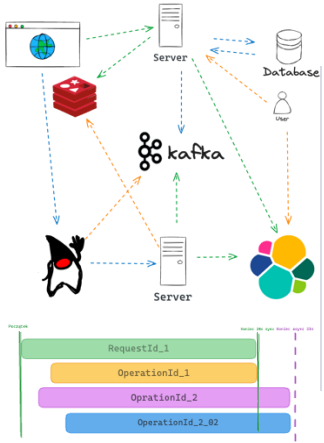
- `00` — Format Version
- `0af7651916cd43dd8448eb211c80319c` — TraceId
- `da246a660ba69254` — SpanId
- `01` — Tracing-flag



baggage: "ClientId=24c3dee4-7696-4b07-a2e3-6e83ef682b3a"

Span Attributes vs Baggage?

Baggage są propagowane zawsze więc jak wołamy zewnętrzne serwisy to też je wyślemy, trzeba uważać na wrażliwe dane

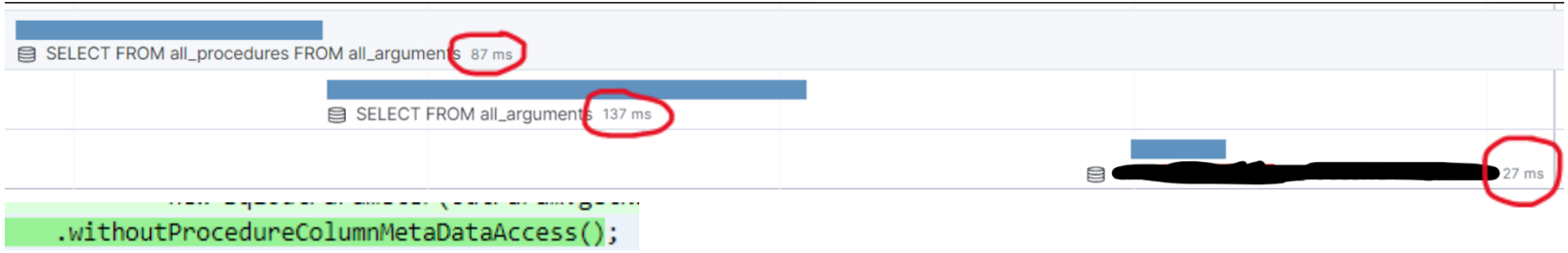


OpenTelemetry Tracing DEMO

trace timeline w ELK

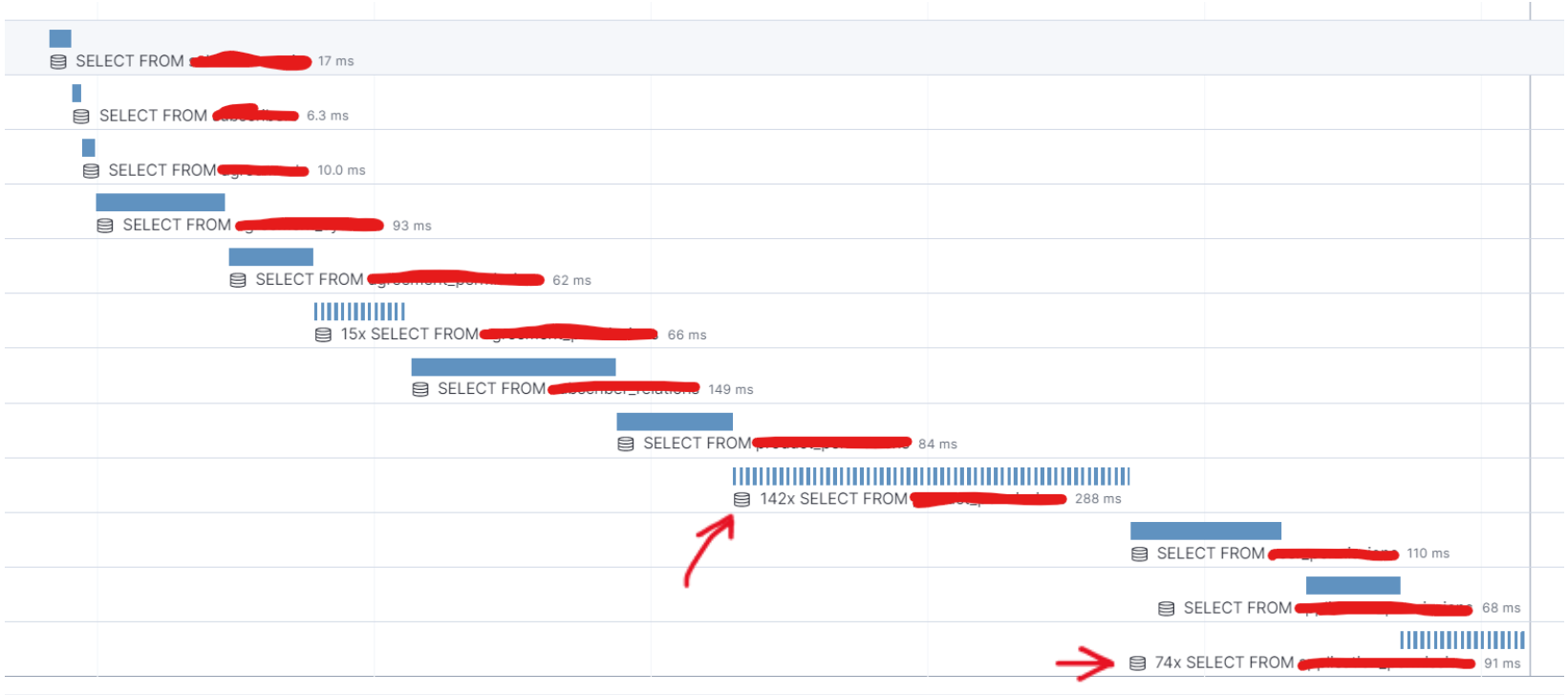
Niektóre problemy, które tracing pomógł rozwiązać w projektach, w których pracowałem

- Przy wykorzystywaniu procedur bazodanowych dodatkowy call do bazy (za każdym razem)



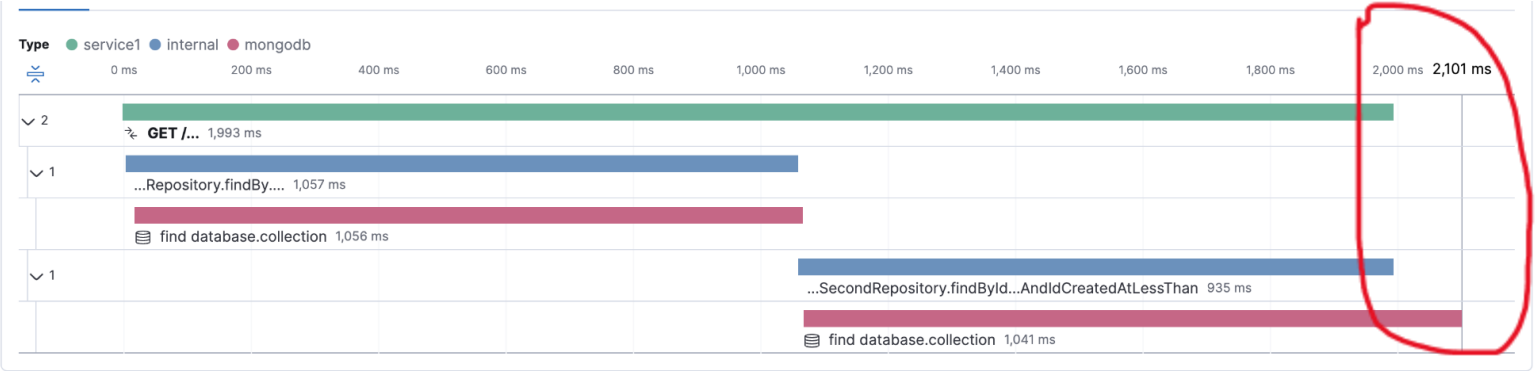
Niektóre problemy, które tracing pomógł rozwiązać w projektach, w których pracowałem

- Turbo N+1



Niektóre problemy, które tracing pomógł rozwiązać w projektach, w których pracowałem

- Degradacja szybkości przetwarzania requestów wraz z ilością danych w bazie (dla niektórych klientów)
- Odcinanie requestu przez Gateway, ze względu na za długi czas trwania

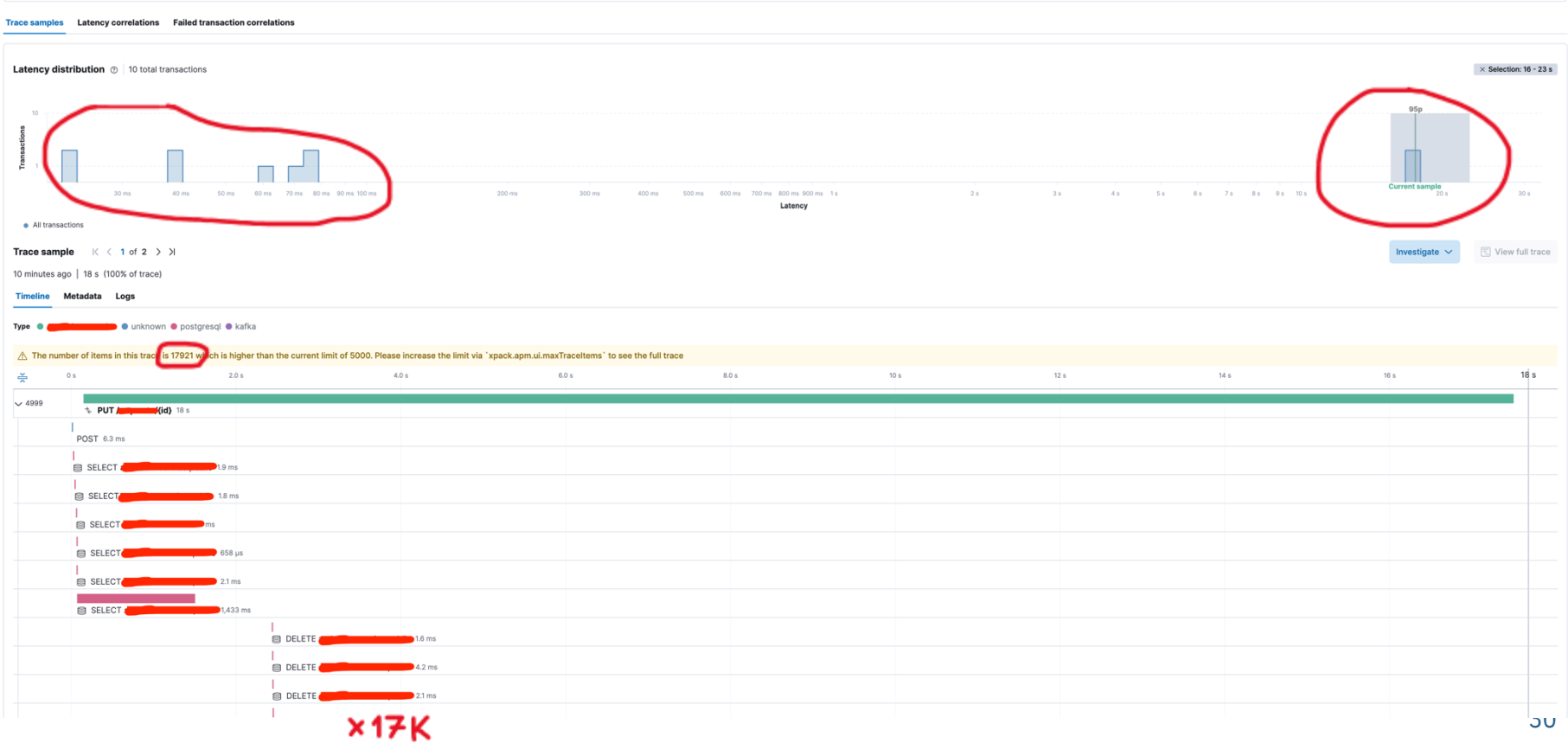


Niektóre problemy, które tracing pomógł rozwiązać w projektach, w których pracowałem

- Wolne przetwarzanie requestu po tracigu wyszło na to, że to serwis walidacji
  - Wyciek pamięci
  - Nieoptymalna logika w silniku używająca Groovy.eval (nr. Wątku z trace + VisualVm)

Niektóre problemy, które tracing pomógł rozwiązać w projektach, w których pracowałem

- Niedeterministyczny problem z testami e2e (timeout po 10s)



Jak tego użyć w Twoim projekcie ?

Jak to wszystko poskładać i użyć w Twoim projekcie ?

BlackBox (automatuczna)

- Agent
- framework np.
  - micrometer Tracing Otel bridge
  - opentelemetry-spring-boot-starter
  - …

WhiteBox (ręczna)

- otel instrumentation api/sdk
- Micrometer api
- …

Mixed (automatyczna + ręczna)

BlackBox inne języki

- Agent (Java, C#)
- Monkey Patching (Js, Python)
- Interpreter extension (PHP)
- eBPF (Go)

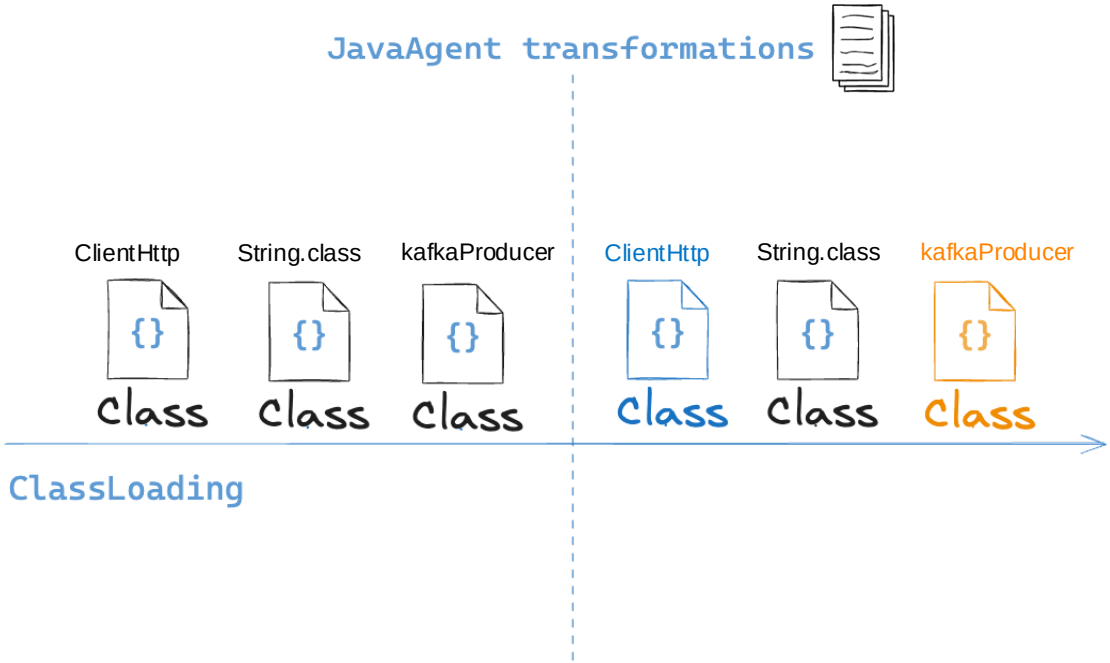
Agent w tej chwili posiada więcej instrumentacji niż oficjalne Otel sdk

Agent

(BlackBox)

Czym jest i co robi agent?

- To specjalny komponent napisany w java, pozwalający modyfikować bytecode aplikacji podczas jej działania
- Wykorzystuje Oficjalne JVM Instrumentation API, do wzbogacania klas w czasie ich ładowania



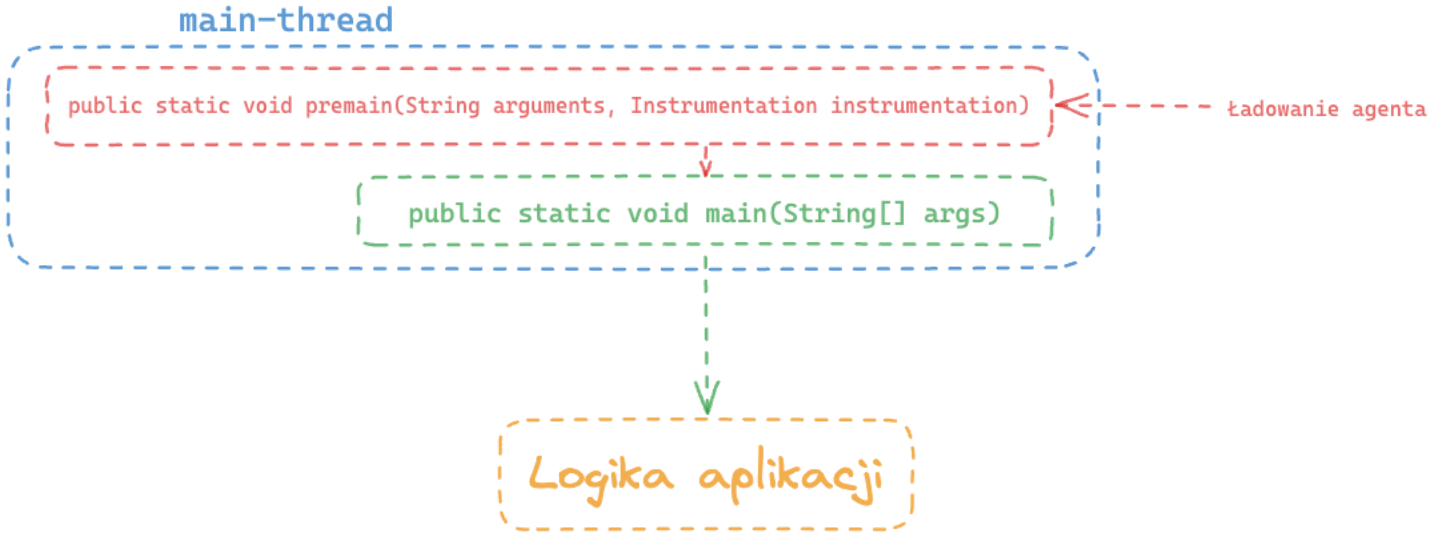
Przykłady:

- TestCovrage
- Mockito Inline Mock
- AspectJ
- VisualVm
- APMy

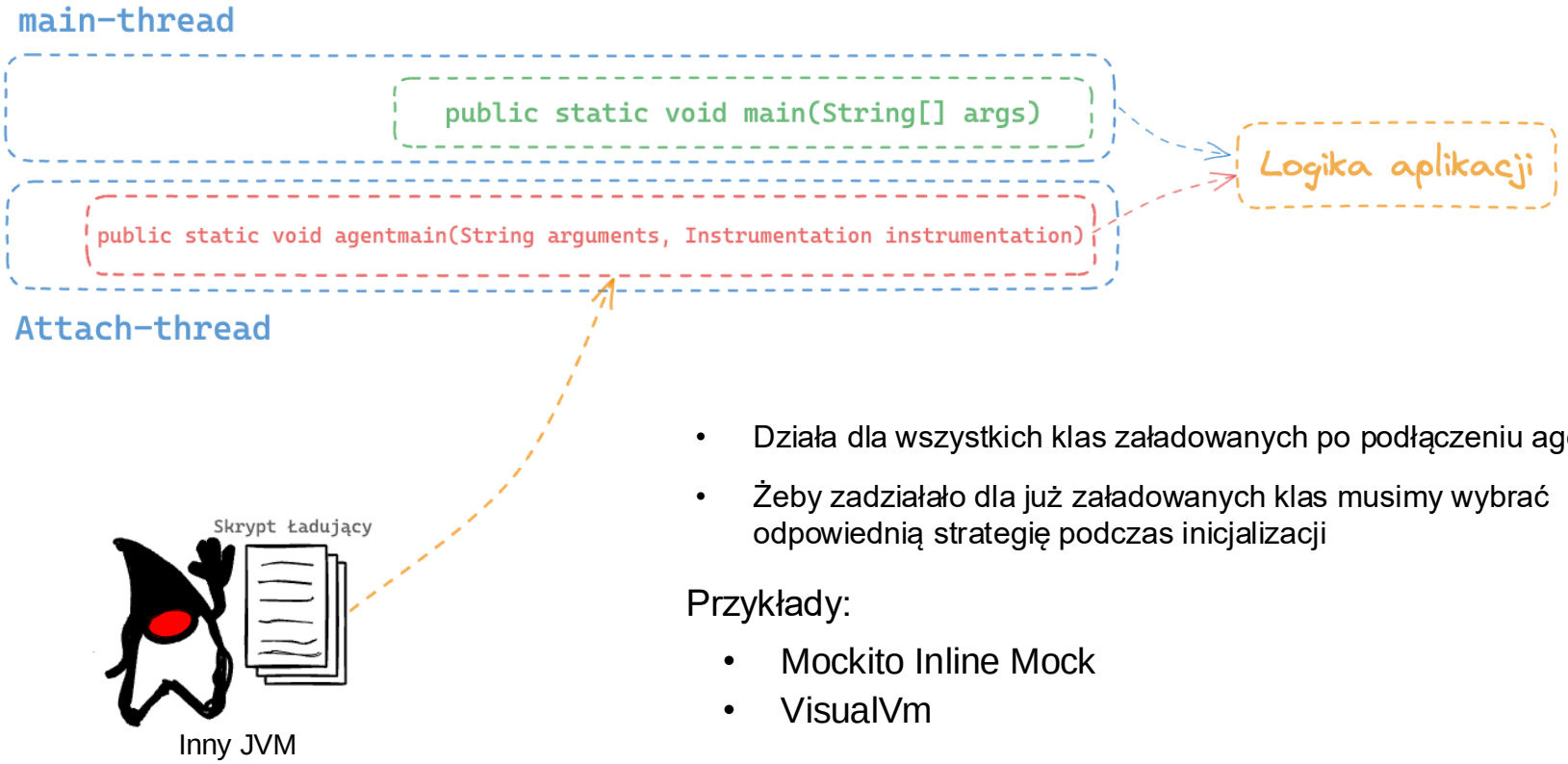
Podłączenie agenta przy starcie aplikacji

java –javaagent otel-agent.jar application.jar

gdy JVM startuje nasz program



Podłączenie agenta w runtime



- Działa dla wszystkich klas załadowanych po podłączeniu agenta
- Żeby zadziałało dla już załadowanych klas musimy wybrać odpowiednią strategię podczas inicjalizacji

Przykłady:

- Mockito Inline Mock
- VisualVm

Przykładowy kod agenta

```
public class Main {  Complexity is 5 Everything is cool!
    public static void premain(String arguments, Instrumentation instrumentation) {  kozaks *

        new AgentBuilder.Default()

            // Filtruje klasy, które mają zostać zainstrumentowane (DispatcherServlet)
            .type(ElementMatchers.nameEndsWith("DispatcherServlet"))  Narrowable

            // Określa, jak zmodyfikować wybrane klasy.
            .transform(( Builder<?> builder,  TypeDescription type,  ClassLoader cl,  JavaModule module,  ProtectionDomain pd) ->

                // informacje o tym co ma być wstrzyknięte do klasy
                builder.visit(Advice.to(ResponseHeaderAddingAdvice.class)

                    // Wybiera metodę doDispatch w DispatcherServlet, jako miejsce, gdzie zostanie dodany kod.
                    .on(ElementMatchers.named("doDispatch")))
            )  Extendable

            // po tej operacji JVM monitoruje ładowanie klas i stosuje określone zmiany.
            .installOn(instrumentation);
    }
```

Metoda Sringa

> Process the actual dispatching to the handler.
>
> The handler will be obtained by applying the servlet's HandlerMappings in order. The HandlerAdapter will be obtained by querying the servlet's installed HandlerAdapters to find the first that supports the handler class.
>
> All HTTP methods are handled by this method. It's up to HandlerAdapters or handlers themselves to decide which methods are acceptable.
>
> Params: `request` – current HTTP request
> `response` – current HTTP response
>
> Throws: `Exception` – in case of any kind of processing failure

```
@SuppressWarnings("deprecation")
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {
```

Przykładowy kod agenta

```
public static class ResponseHeaderAddingAdvice {  1 usage  kozaks *
                                                                    Complexity is 4 Everything is cool!

    private ResponseHeaderAddingAdvice() {  no usages  kozaks
    }

    @Advice.OnMethodEnter() // wywoływane przed wejściem do metody doDispatch  no usages  kozaks *
    public static void doDispatchEnter(@AllArguments(readOnly = false, typing = DYNAMIC) Object[] args) {

        final ResponseFacade response = (ResponseFacade) args[1];

        if (response != null) {
            final String generatedTraceId = UUID.randomUUID().toString();
            response.addHeader("X-Custom-TraceId", generatedTraceId); // dodajemy nagłówek do odpowiedzi
        }
    }
}
```

Agent Demo

Tip #1

Dodanie TraceId oraz SpanId do logów (Agent automatycznie wrzuca je do MDC)

Spring

```
logging:
  pattern:
    correlation: "[ %X{trace_id:-},%X{span_id:-} ] "
```

Logback

```
<encoder>
    <pattern>....%X{trace_id:-} %X{span_id:-}...</pattern>
</encoder>
```

```
2024-06-09T15:56:44.553Z  INFO 1 --- [analyzes] [ntainer#0-0-C-1] [ 204e459ada2ab46561e7a82f4ca6ee2a,9cdc54bc67ab3b00 ] c.p.c.s.a.a.CreditCardApplicationSink    : Message consumed
2024-06-09T15:56:44.553Z  INFO 1 --- [analyzes] [ntainer#0-0-C-1] [ cfa5369cce2fd8044a5927239e0d804f,313f4a1ef44bc0dd ] c.p.c.s.a.a.CreditCardApplicationSink    : Consuming message {"id":"CCARD-204
","type":"ID_CARD","pesel":"84283119106","income":11415,"number":"IRH2YU8M","sourceId":"CCARD-20447","sourceType":"CreditCardApplication","creditLimit":50000.0,"destination":"application-processed-
ate":"APPROVED"}
```

Jeśli wystawiamy API restowe, to warto dodać traceId (np. jako errorId) do błędu http

```
{
  ”code": 500,
  "errorMessage": "Unexpected Error occured please try again latter",
  "errorId": "6f9ca1f1499db1ebd978cc8115a8a6ba"
}
```

Koniec Cz. I Q&A

Kilka linków, jak by ktoś nie dotrwał do końca ;)

Ankieta: https://forms.gle/WYdy2VsRp1SkNYNa8

Kod i slajdy: *https://cupofcodes.pl/talks/*

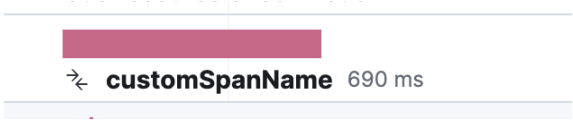
Co jeśli Chcemy wystartować własny span/zmodyfikować istniejący? (whiteBox)

```
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-instrumentation-annotations</artifactId>
    <version>1.32.0</version>
</dependency>
```

Aby z tego skorzystać:

- Działający Agent
- Lub opentelemetry-spring-boot-starter - działa tylko z beanami springa

```
1 usage   kozaks *
@WithSpan(value = "customSpanName", kind = SpanKind.SERVER)
public Mono<FraudResult> checkFraud(@SpanAttribute FraudCheck fraudCheck) {
    if (isUnderAge(fraudCheck.dateOfBirth())) {
        return Mono.just(FraudResult.underAge());
    }
```



labels

labels._deployment_environment	dev"
labels.code_function	checkFraud
labels.code_namespace	pl.com.kozak.telemetry.creditwise.fraud.check.FraudCheckService
labels.fraudCheck	FraudCheck[name=John Doe, email=johndoe@example.com, lastName=null, dateOfBirth=1985-05-01, street=123 Main St, city=Anytown, zip=12345]

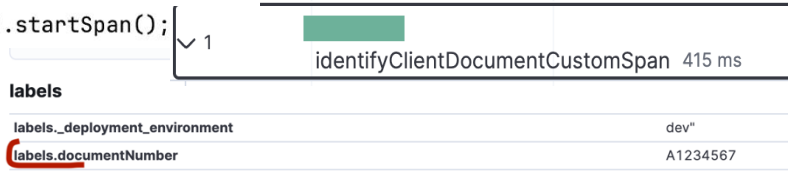
Co jeśli Chcemy wystartować własny span, a adnotacje to za mało?

Manualnie

```
<dependency>
   <groupId>io.opentelemetry</groupId>
   <artifactId>opentelemetry-api</artifactId>
   <version>1.32.0</version>
</dependency>
```

```
1 private final Tracer tracer;
```

```
1 usage   SKozak *
public IdentificationResult identifyClientDocument(DocumentToVerification document) {
2    Span span = tracer.spanBuilder("identifyClientDocumentCustomSpan").startSpan();
3    try (Scope scope = span.makeCurrent()) {
6        span.setAttribute("documentNumber", document.getNumber());

         log.info("trying to identify client documet {}", document);
         final IdentificationResult identificationResult = identityRegistryClient.verifyDoc
         log.info("client document verified with response {}", identificationResult);
         return identificationResult;
     } catch (Exception e) {
         log.error("error while identifying client document", e);
4        span.recordException(e);
         throw e;
     } finally {
5        span.end();
     }
```



Skąd wziąć instancje OpenTelemetry lub tracer ?

```
<dependency>
   <groupId>io.opentelemetry</groupId>
   <artifactId>opentelemetry-api</artifactId>
   <version>1.32.0</version>
</dependency>
```

Gdy nie mamy żadnego z powyższych

GlobalOpenTelemetry.*get*(); zwróci No-op

Agent

```
OpenTelemetry openTelemetry = GlobalOpenTelemetry.get();
```

```
final Tracer tracer = openTelemetry.
        getTracer("creditcards-identity", "1.0.0");
```

OpenTelemetry sdk

```
private final OpenTelemetry openTelemetry =
AutoConfiguredOpenTelemetrySdk.initialize().getOpenTelemetrySdk();
```

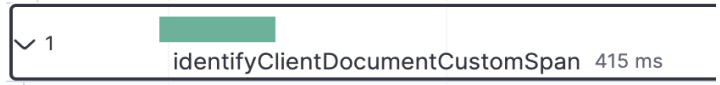
opentelemetry-spring-boot-starter

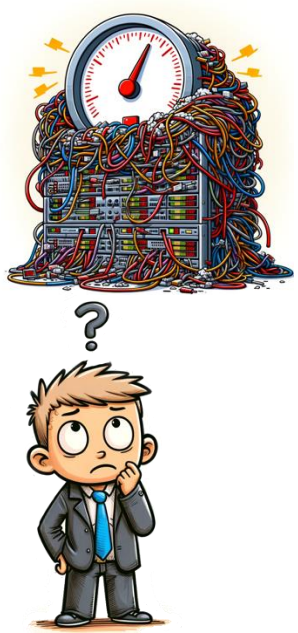
```
@Autowired
OpenTelemetry openTelemetry; // lub przez konstruktor
```

Co jeśli Chcemy wystartować własny span, a adnotacje to za mało?

Adnotacje + Manual

```
1 @WithSpan(value = "identifyClientDocumentCustomSpan")  1 usage   SKozak *
  public IdentificationResult identifyClientDocument(DocumentToVerification document) {
      //acquiring span
2     Span currentSpan = Span.current();
      //setting attributes
3     currentSpan.setAttribute("documentNumber", document.getNumber());

      log.info("trying to identify client documet {}", document);
      final IdentificationResult identificationResult = identityRegistryClient.verifyDocument(document);
      log.info("client document verified with response {}", identificationResult);
      return identificationResult;
  }
```





Za dużo danych

Redukcja ilości danych

Filtrowanie - dobry początek, ale to za mało

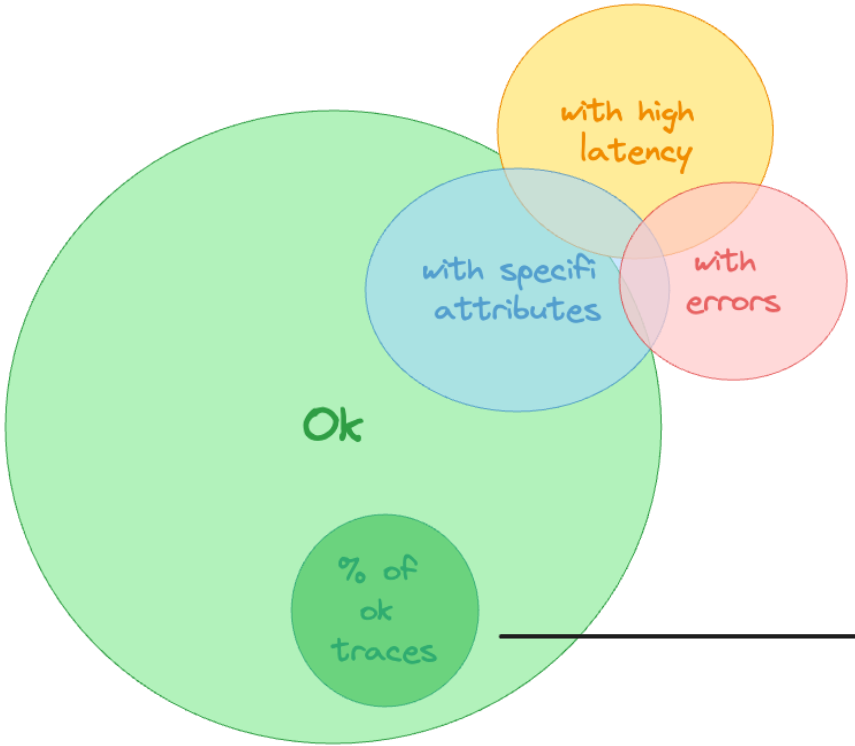
Head-based sampling - Najprostrza opcja (decyzja na początku śledzenia)

- Zawsze
- Nigdy
- Procentowo np. 0.20 (20%)
- poleganie na fladze w traceparent (00 lub 01)
- …

Tail-based sampling – trudniejsza opcja, jednak dająca dużo większe możliwości

Sampling – tail based

Czy potrzebujemy cały ten ruch ?



Czy Wystarczy nam to ?

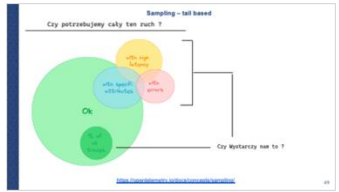
https://opentelemetry.io/docs/concepts/sampling/

Sampling

Head-based - Najprostrza opcja (decyzja na początku śledzenia)

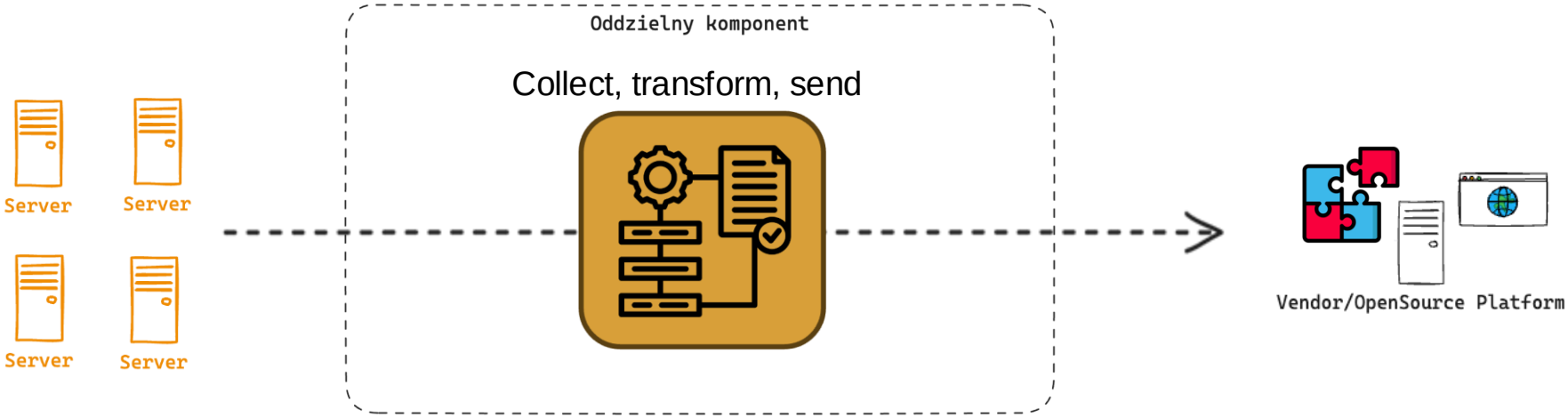
- Zawsze
- Nigdy
- Procentowo np. 0.20 (20%)
- polega na fladze w traceparent (00 lub 01)
- polega na fladze w traceparent, ale zawsze wyłączony dla tego serwisu
- polega na fladze w traceparent, ale procentowo

Tail-based – trudniejsza opcja, jednak dająca dużo większe możliwości



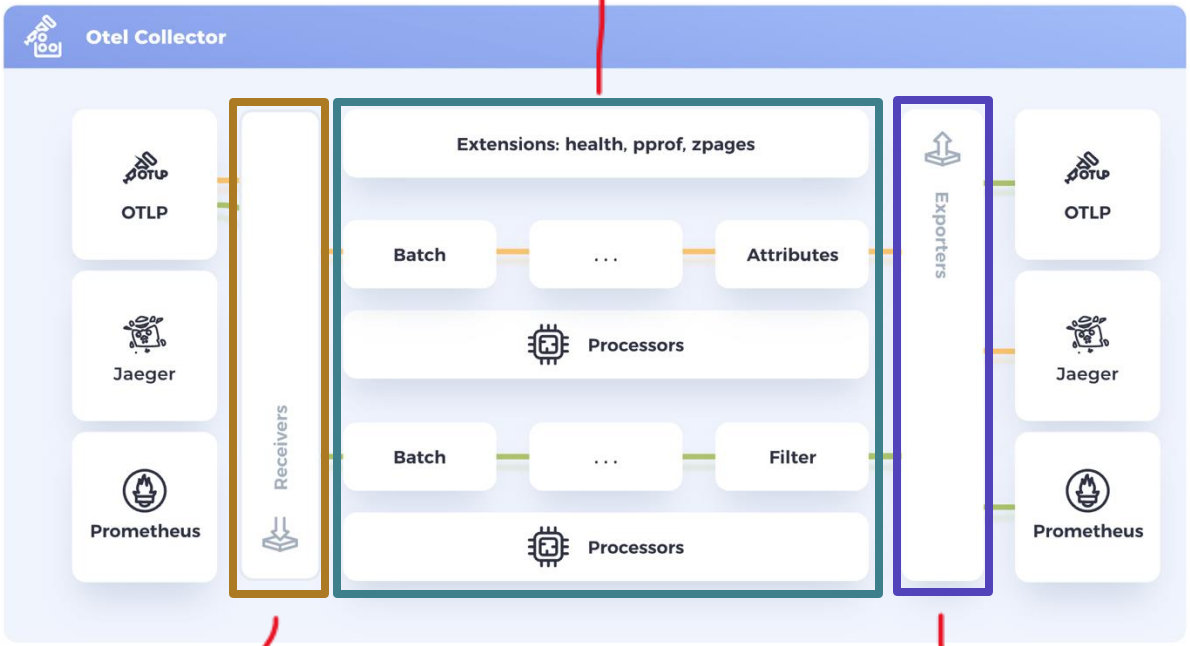
- decyzja gdy mamy zebrany cały trace
- Ile czasu czekać na cały trace (na decyzję) ?
- Gdzieś przez ten czas trzeba trzymać dane (więcej ramu)/ czy może buffor na dysk

Collector


Server
Server
Server
Server
Oddzielny komponent
Collect, transform, send
Vendor/OpenSource Platform

Collector

procesowanie Batchowanie/rename/filtrowanie dodawanie atrybutów



Odbieranie informacji i dostosowywanie ich do do protokołu OTLP

Wyjście w standardzie OTLP
Lub dostosowanie do vendora

Konfiguracja Collectora, yaml

Definiowanie komponentów

```yaml
receivers:
  otlp:
    protocols:
      grpc:
      http:
  hostmetrics:
    collection_interval: 60s
    scrapers:
      cpu:
      memory:
```

```yaml
processors:
  batch:
```

```yaml
exporters:
  otlp/elastic:
    endpoint: http://apm-server:8200
    tls:
      insecure: true
    headers:
      # Elastic APM Server secret token
      Authorization: "Bearer test321test"
#  kafka:
#    brokers:
#      - kafka:9092
#    protocol_version: 2.0.0
#    encoding: otlp_json
#    topic: traces
```

Definiowanie pipeline

```yaml
service:
  pipelines:
    traces:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ otlp/elastic ]
    metrics:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ otlp/elastic ]
    logs:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ otlp/elastic ]
```

Zastosowanie Kolektora

Tail-based sampling - processor

```
processors:
  tail_sampling:
    decision_wait: 10s
    num_traces: 100
    expected_new_traces_per_sec: 10
    policies:
      [
        {
          name: test-policy-1,
          type: status_code,
          status_code: { status_codes: [ ERROR, UNSET ] }
        },
        {
          name: test-policy-2,
          type: latency,
          latency: { threshold_ms: 5000 }
        },
        {
          name: test-policy-3,
          type: string_attribute,
          string_attribute: { key: clientId, values: [ 12314155, 1245256322 ] }
        },
      ]
```

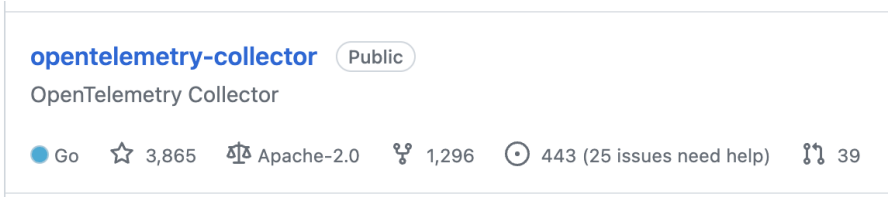
Przykładowe Inne zastosowanie kolektora

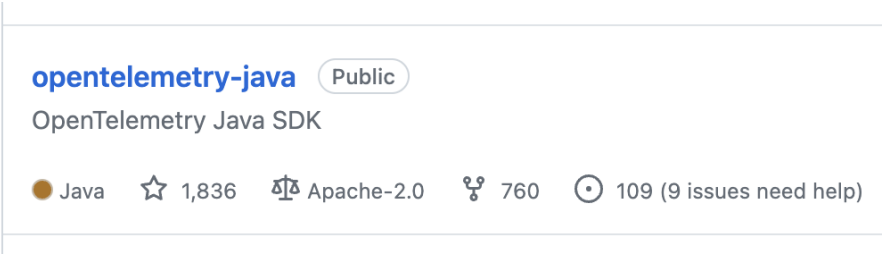
- Filtrowanie niechcianych trace np.. Healthcheck, swagger
- Wzbogacanie np.. o dane poda/infry
- Security (np. obfuskacja wrażliwych danych
- Dodatkowe metryki
  Scrap oss services (Mongo, redis, kafka itp.)
  Jmx, files, syslog

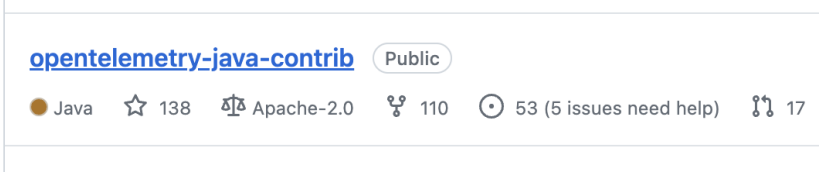
https://github.com/open-telemetry/opentelemetry-collector-contrib

https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/processor/tailsamplingprocessor/README.md

Collector vs Collecotr contrib







- Fundament OpenTelemetry – stabilny i zgodny ze specyfikacją.
- Oferuje kluczowe funkcje, takie jak podstawowe API, SDK, kolektor, standardowe instrumentacje bibliotek i integracje.
- Główne repozytorium, wspierane i utrzymywane z długoterminową stabilnością.

- Rozszerzenie ekosystemu – eksperymentalne i dodatkowe funkcjonalności.
- Zawiera mniej popularne lub specyficzne integracje, niestandardowe instrumentacje i prototypy.
- Aktywnie rozwijane przez społeczność – przydatne, ale czasem mniej stabilne.

Tail sampling - tradeoffs

Wszystkie spany w trace muszą trafić na tą samą instancję

Większe zużycie procesora i pamięci niż w stateless operacjach

Może być konieczne odpowiednie skalowanie collektorów oraz dodanie loadBalancerów

Przerwany trace

Czyli, kiedy automat nie działa

Gdzie tracing się gubi ?

Wszędzie tam gdzie nie jesteśmy w stanie przesłać traceparent wysyłający/odbierający

- Startowanie wątku, ręczne lub przez executor service

Ręczne startowanie wątku

```
new Thread(() -> log.info("Sleeping in new thread"))
        .start();
```

```
[analyzes] [ntainer#1-0-C-1] [ eadabfc9740f87839f2b46a06d3a872e,3c3368e6f0fa01ac ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Consuming message
[analyzes] [       Thread-3] [ , ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Sleeping in new thread
```

```
final Context propagatedContext = Context.current();
new Thread(() -> {
                                                    ){

}).start();
```

extract trace

restore trace

Technika ta przyda nam się również później

```
[analyzes] [ntainer#1-0-C-1] [ eadabfc9740f87839f2b46a06d3a872e,3c3368e6f0fa01ac ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Message consumed
[analyzes] [       Thread-4] [ eadabfc9740f87839f2b46a06d3a872e,3c3368e6f0fa01ac ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Sleeping with trace in new thread
```

Najprościej

```
final ExecutorService executorService = Context.taskWrapping(new ScheduledThreadPoolExecutor(10));
executorService.submit(() -> log.info("Sleeping with trace in new thread"));
```

```
[ntainer#1-0-C-1] [ dbca3f94a659acb07b6ada2eba1a72e3,4089703e143eb790 ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Message consumed
[pool-8-thread-1] [ dbca3f94a659acb07b6ada2eba1a72e3,4089703e143eb790 ] c.p.c.s.a.c.sink.CreditLimitSumarySink   : Sleeping with trace in new thread
```

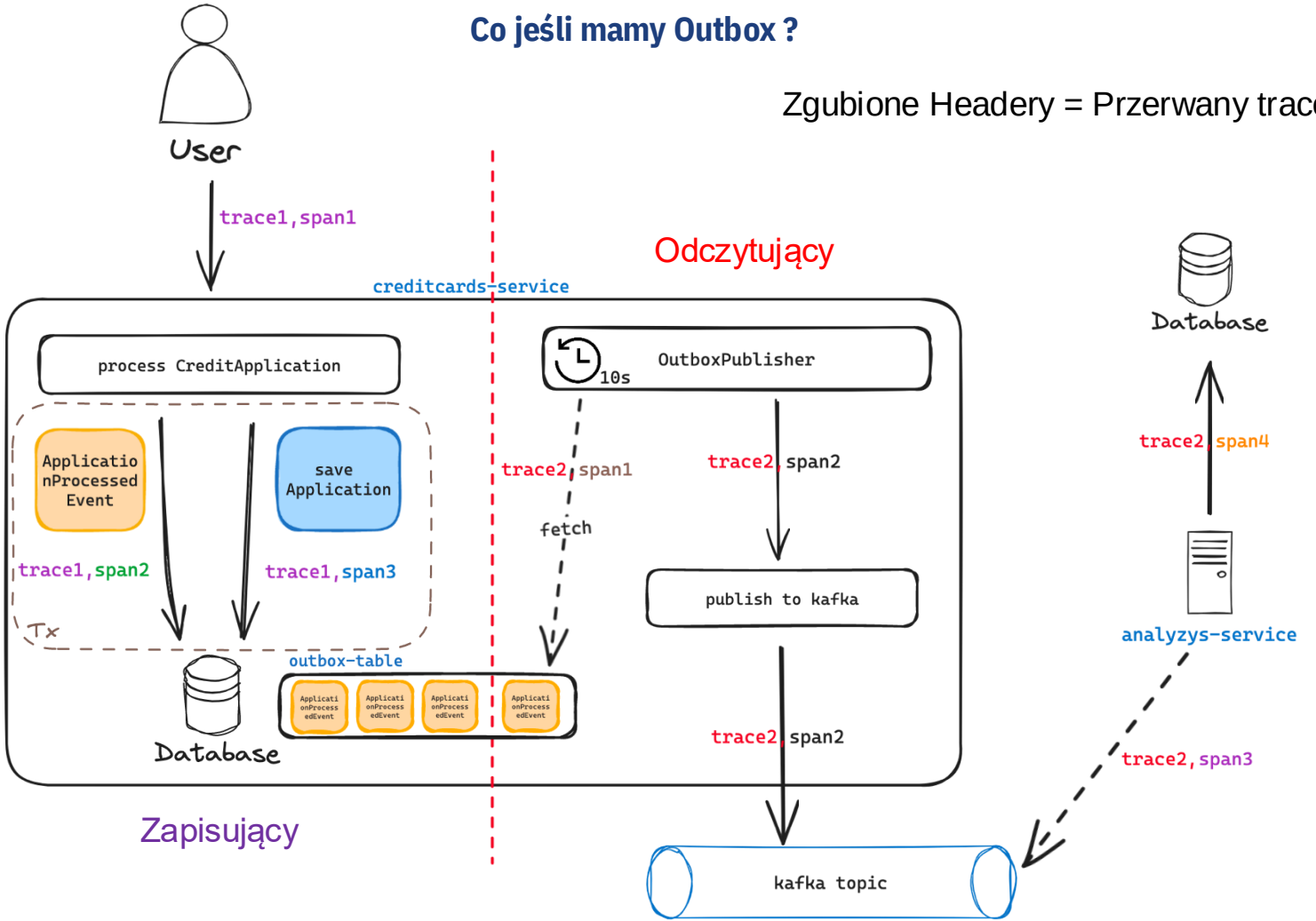
Gdzie tracing się gubi ?

Wszędzie tam gdzie nie jesteśmy w stanie przepropagować traceparent dalej

- Ręczne startowanie wątku
- Outboxy/Inbox (Transakcyjne wysyłanie wiadomości)

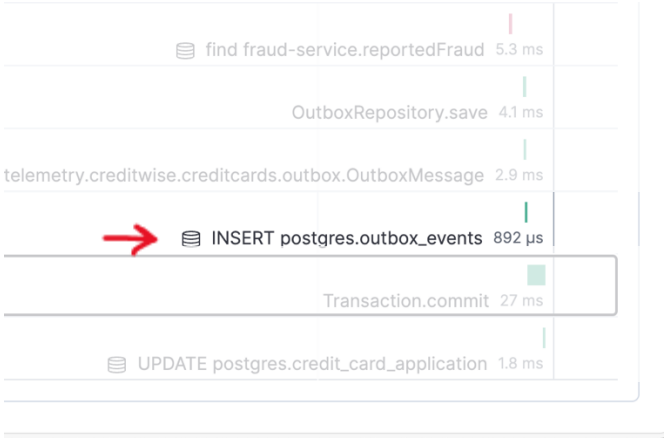
Co jeśli mamy Outbox ?

Zgubione Headery = Przerwany trace

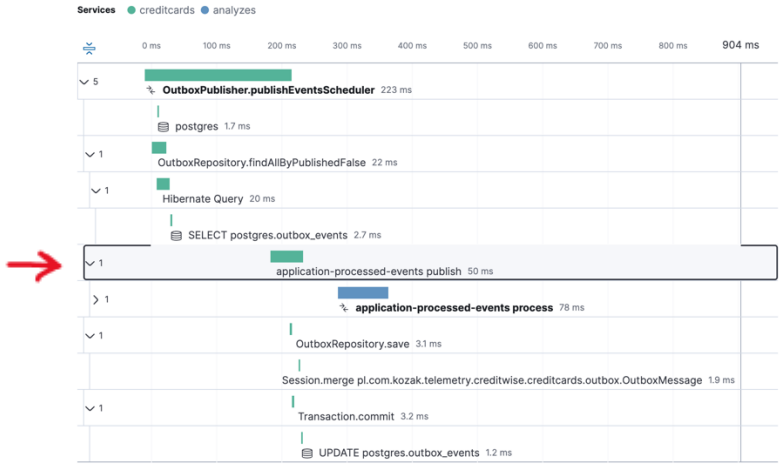


Co jeśli mamy Outbox ?
– jak to wygląda na vizualizacji

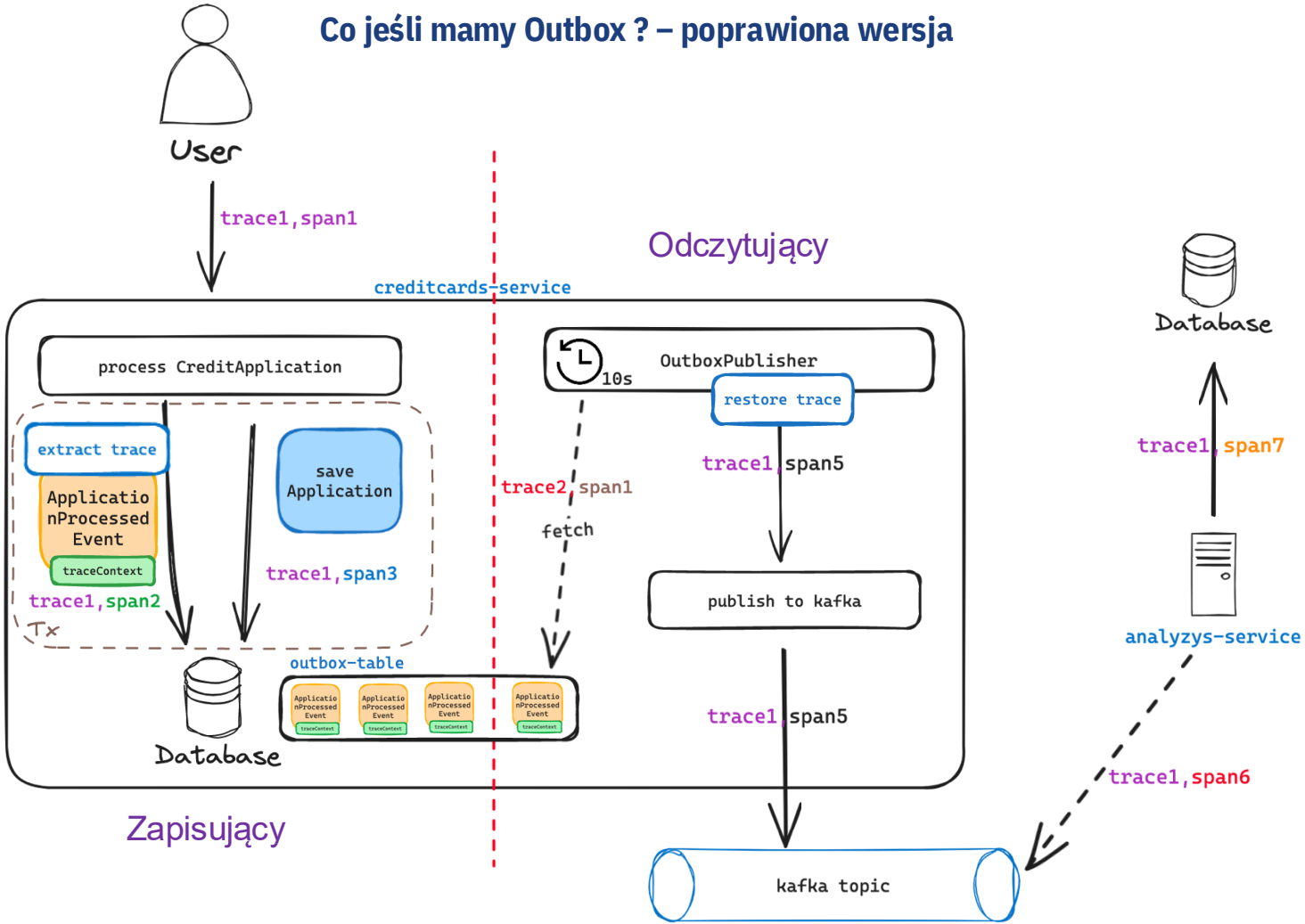
Trace1



Trace2



Co jeśli mamy Outbox ? – poprawiona wersja



Wyciąganie TraceContext

```java
private void saveToDatabase(OutboxEvent outboxEvent) {  1 usage  kozaks *
    final Map<String, String> map = new HashMap<>();
    final TextMapPropagator textMapPropagator = openTelemetry.getPropagators().getTextMapPropagator();
    textMapPropagator.inject(Context.current(), map, Map::put);
    log.info("current Tracing data: {}", map);
```

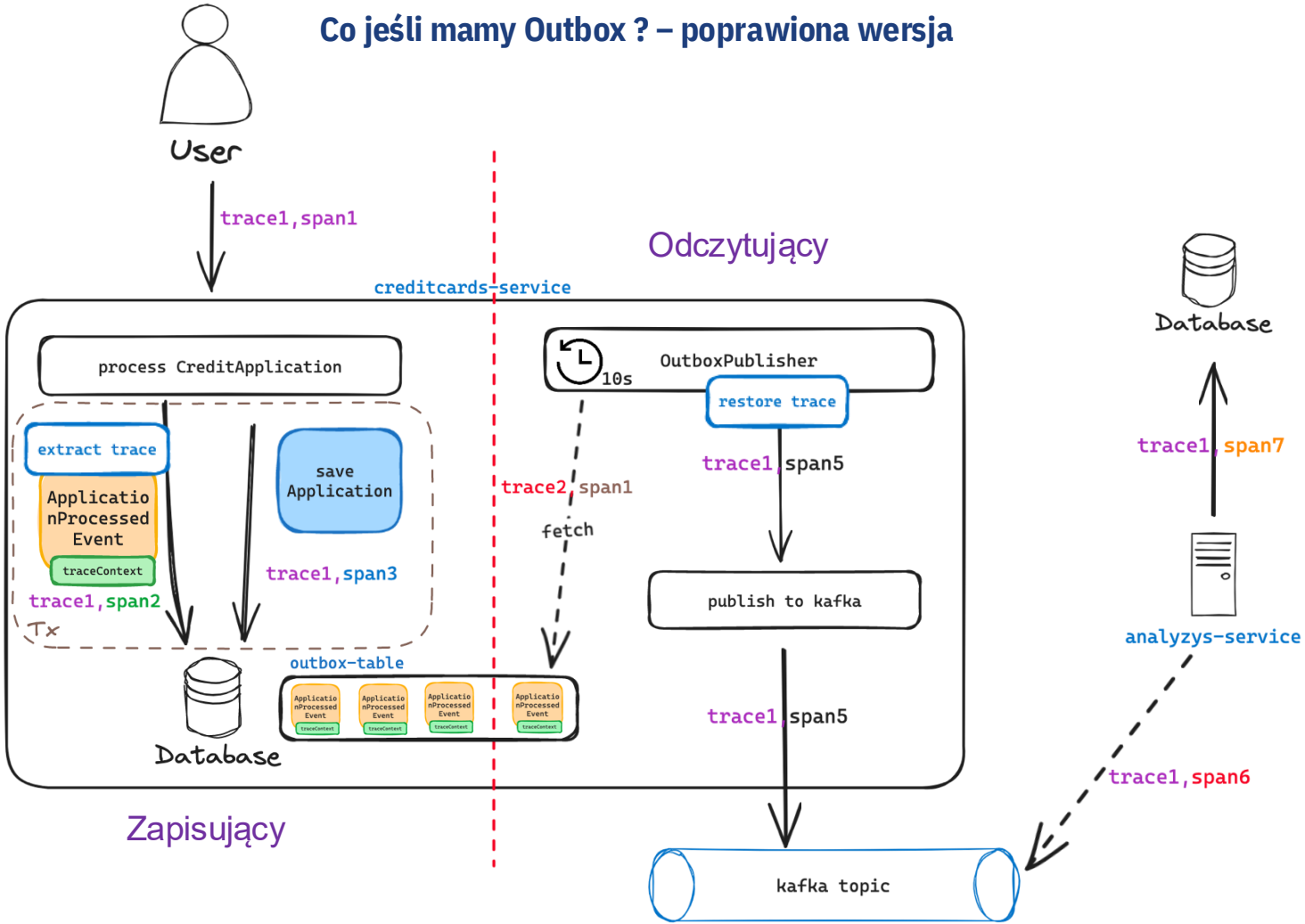
Lub

```java
(passedMap, key, value) -> passedMap.put(key, value));
```

```java
    try {
        OutboxMessageBuilder outboxMessageBuilder = OutboxMessage.builder();
        //....
        outboxMessageBuilder.tracingData(objectMapper.writeValueAsString(map));
        //....
        final OutboxMessage outboxMessage = outboxMessageBuilder.build();
        manualOutboxRepository.save(outboxMessage);
    } catch (JsonProcessingException e) {
        throw new RuntimeException(e);
    }
}
```

	id	source_id	tracing_data	source_type	destination
1	2	CCARD-2	{"traceparent":"00-59f7acb9d9fd6e39687c5a3b3b52fc01-ea24216eb5267c96-01"}	CreditCardApplication	application-processed-events
2	3	CCARD-3	{"traceparent":"00-e458e4e2261b140e6be9978a9f98f79e-ca5fb23140d5a3d7-01"}	CreditCardApplication	application-processed-events
3	4	CCARD-5	{"traceparent":"00-c36795213b280d4779cceffe907a71b0-6e7991dcd42ef91b-01"}	CreditCardApplication	application-processed-events

Co jeśli mamy Outbox ? – poprawiona wersja



Przywracanie TraceContext

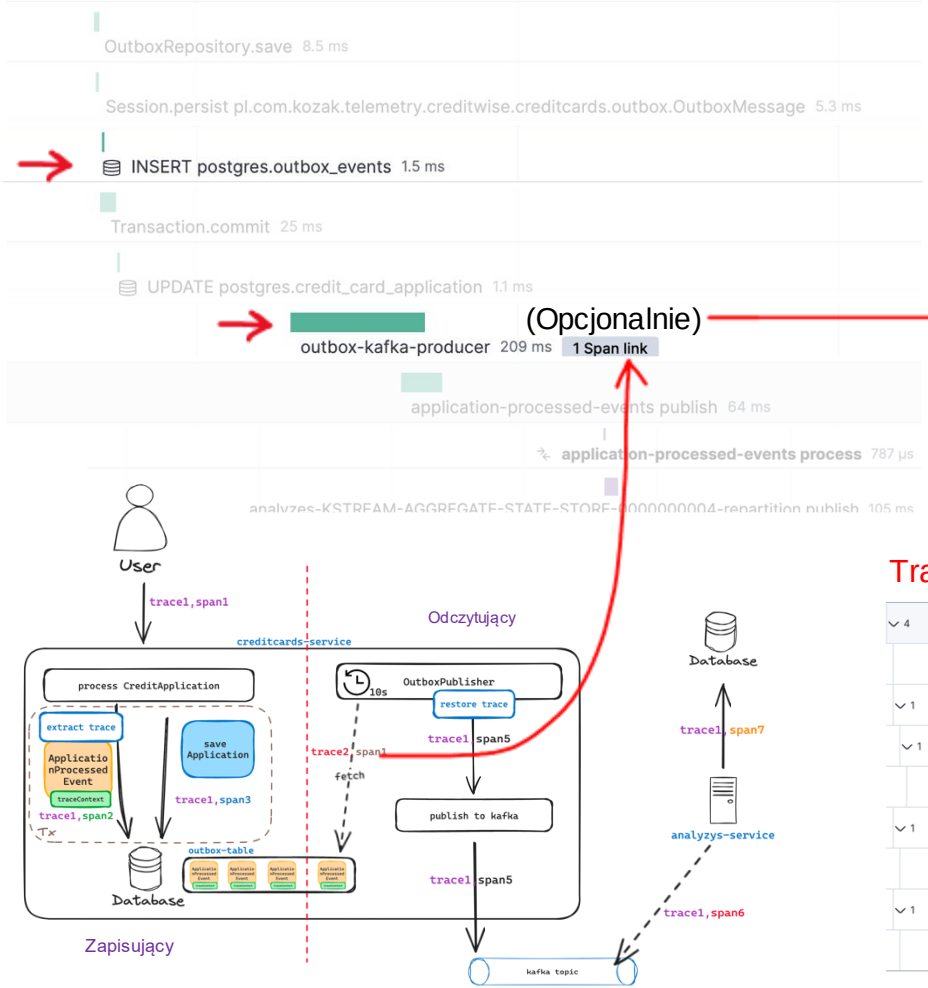
```java
@Transactional  kozaks
@Scheduled(fixedDelay = 5000) // 5 seconds
public void publishEventsScheduler() throws JsonProcessingException {
    List<OutboxMessage> events = outboxRepository.findAllByPublishedFalse(PageRequest.of(0, 20)).getContent();
    for (OutboxMessage event : events) {
        sendEvent(event);
        event.markPublished();
        outboxRepository.save(event);
    }
}
```

```java
private void sendEvent(OutboxMessage event) throws JsonProcessingException {  1 usage  kozaks *
    TextMapGetter<Map<String, String>> getter =
        new TextMapGetter<>() {  kozaks
            @Override  kozaks
            public String get(Map<String, String> carrier, String key) {
                if (carrier.containsKey(key)) {
                    return carrier.get(key);
                }
                return null;
            }

            @Override  kozaks
            public Iterable<String> keys(Map<String, String> carrier) { return carrier.keySet(); }
        };

1 Map<String, String> map = objectMapper.readValue(event.getTracingData(), new TypeReference<HashMap<String, String>>() {});
2 Context extractedContext = textMapPropagator.extract(Context.current(), map, getter);
3 final SpanContext currentSpanToLink = Span.current().getSpanContext();

4 try (Scope unused = extractedContext.makeCurrent()) {
5     Span serverSpan = tracer.spanBuilder("outbox-kafka-producer")
            .addLink(currentSpanToLink)
            .setSpanKind(SpanKind.INTERNAL)
            .startSpan();
        try {
            kafkaTemplate.send(event.getDestination(), event.getKey(), event.getPayload());
        } catch (Exception e) {
            serverSpan.recordException(e);
            throw e;
        } finally {
            serverSpan.end();
        }
    }
}
```

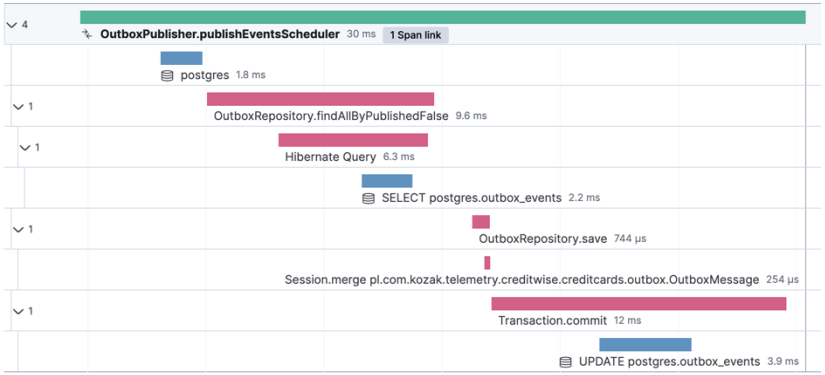
outbox-kafka-producer 970 µs **1 Span link**

Poprawiony outbox, kontunuacja trace manualnie

Trace1



Trace2



W skrócie

Dlaczgo nie zadziałało automatycznie?

- Przy zapisie do bazy, brak instrumentacji, która zapisze też traceContext
- Przy odczycie z bazy, brak instrumentacji, która przywróci traceContext

Jak to naprawić ?

zapis

1) Pobierz aktualny trace `extract trace`

2) Serializuj i zapisz TraceContext

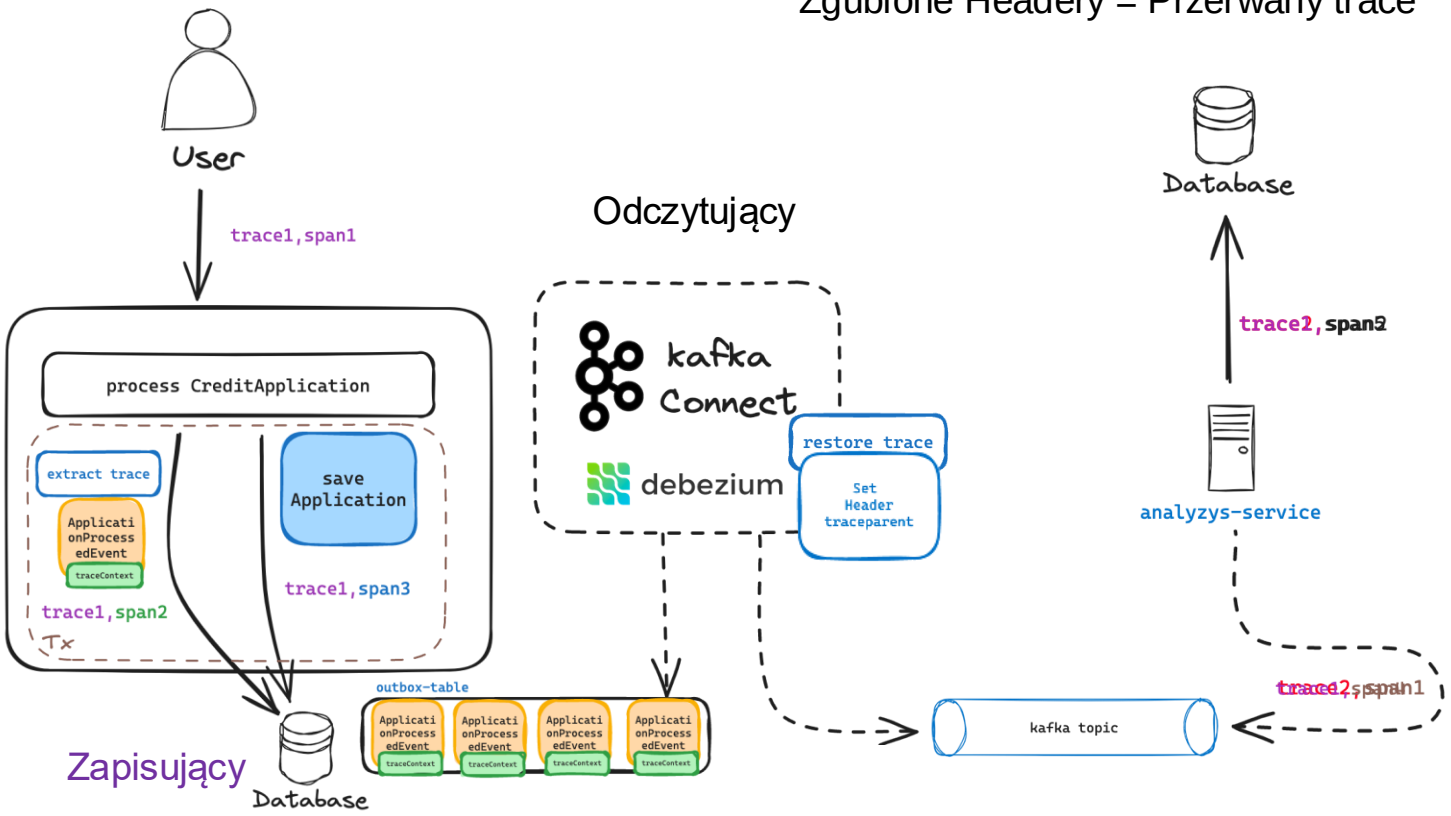
odczyt

4) Deserializuj/odczytaj TraceContext

5) Przywróć TraceContext oznaczając go jako aktualnie obowiązujący `restore trace` w wątku przetwarzającym operację

Co jeśli mamy Outbox – debezium/kafkaconnect?

Zgubione Headery = Przerwany trace



Debezium ustawianie headera

```
curl -X POST \
     -H "Content-Type: application/json" \
     --data '
     {
       "name": "outbox-connector",
       "config": {
         "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
         "database.hostname": "db",
         "database.port": "5432",
         "database.user": "postgres",
         "database.password": "testPassword",
         "database.dbname": "postgres",
         "database.server.name": "dbserver1",
         "table.include.list": "public.debezium_outbox_events",
         "topic.prefix": "outbox",
         "plugin.name": "pgoutput",
         "transforms": "outbox",
         "transforms.outbox.type": "io.debezium.transforms.outbox.EventRouter",
         "transforms.outbox.table.fields.additional.placement": "trace:header:traceparent",
         "transforms.outbox.table.expand.json.payload": "true",
         "key.converter": "org.apache.kafka.connect.storage.StringConverter",
         "value.converter": "org.apache.kafka.connect.json.JsonConverter",
         "value.converter.schemas.enable": "false"
       }
     }
     ' \
     http://localhost:8083/connectors -w "\n"
```

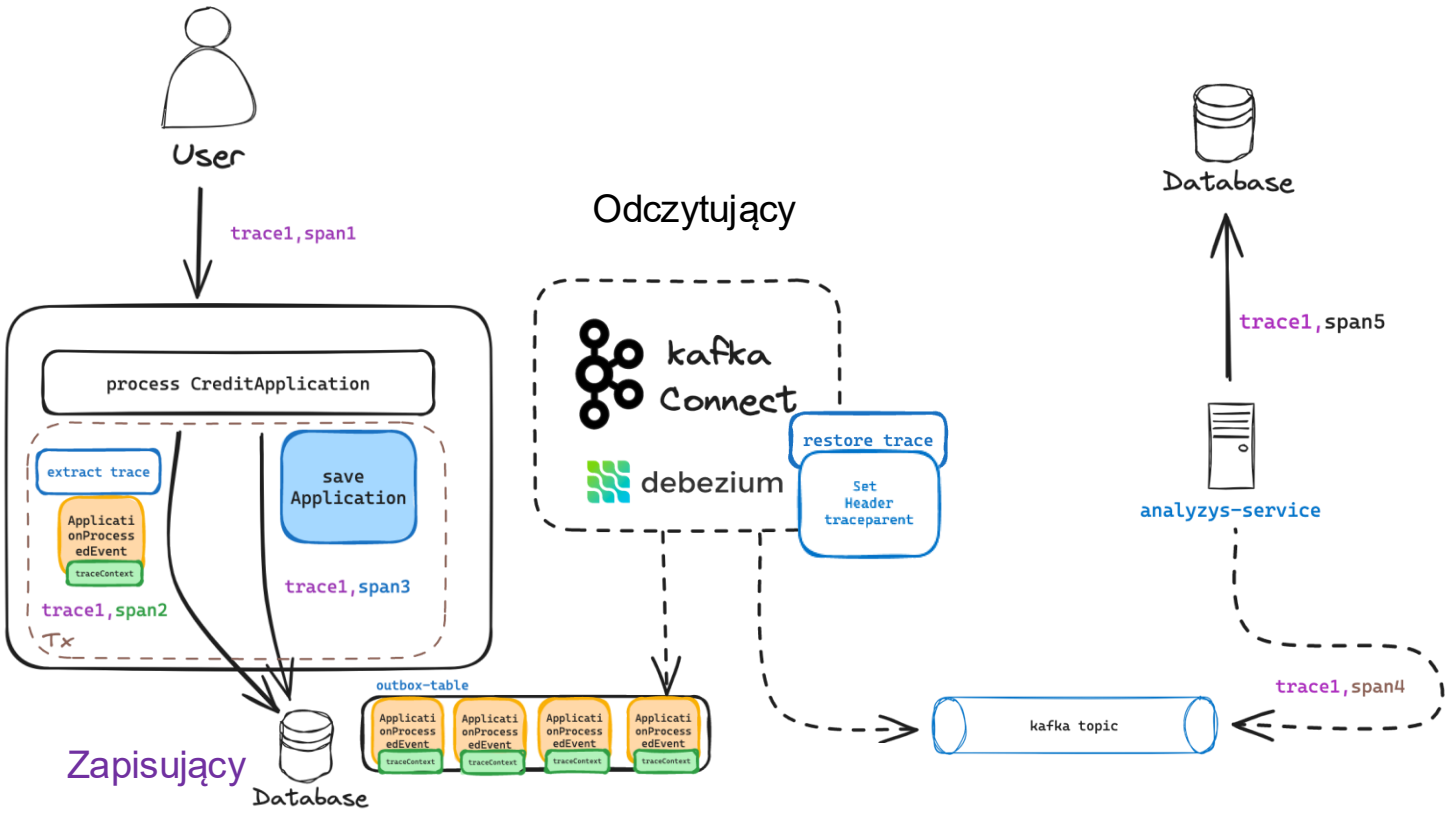
source

target

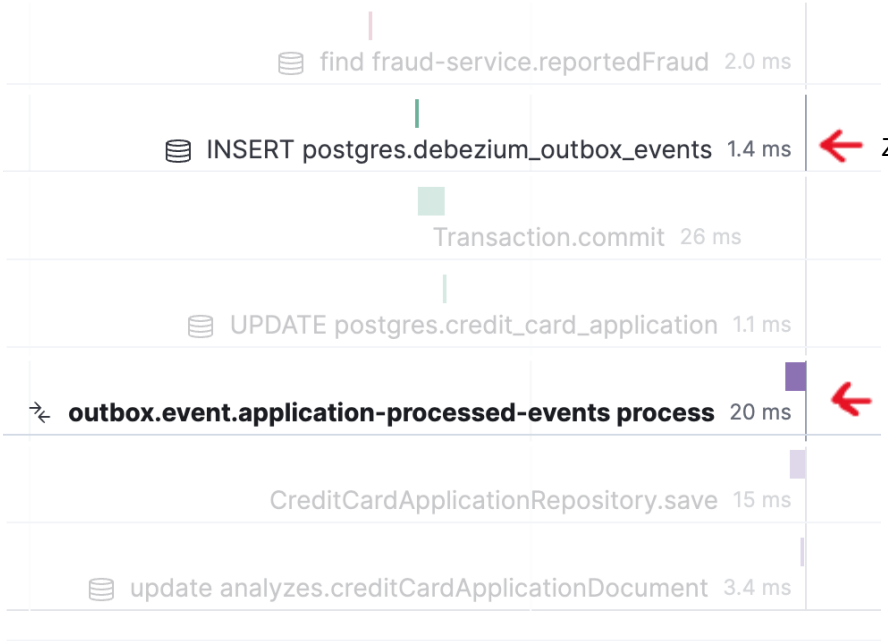
Wyślij w wiadomości
Nowy header
Nazwa: traceparent
Wartość: taka jak w kolumnie „trace”

aggregateid	payload	trace
92431543123	{"id": "CCARD-1183", "key": "92431543123", "type": "ID_CARD", "pesel": "92431	00-302cdec1a85cd5978aaf03d135f16a77-4b7ea1e0abe48857-01
92431543123	{"id": "CCARD-1183", "key": "92431543123", "type": "ID_CARD", "pesel": "92431	00-302cdec1a85cd5978aaf03d135f16a77-4b7ea1e0abe48857-01
92431543123	{"id": "CCARD-1185", "key": "92431543123", "type": "ID_CARD", "pesel": "92431	00-bd5e639509e4eaf666b2602817fd3cef-fa6ca76d0e83d9e1-01

Co jeśli mamy Outbox – debezium/kafkaconnect?



Co jeśli mamy Outbox – debezium/kafkaconnect?



Zapis do outboxa

Odczytanie z kafki przez serwis

Jeśli nie potrzebujemy mierzyć/widzieć czasu procesowania na debezium ten sposób nam wystarczy

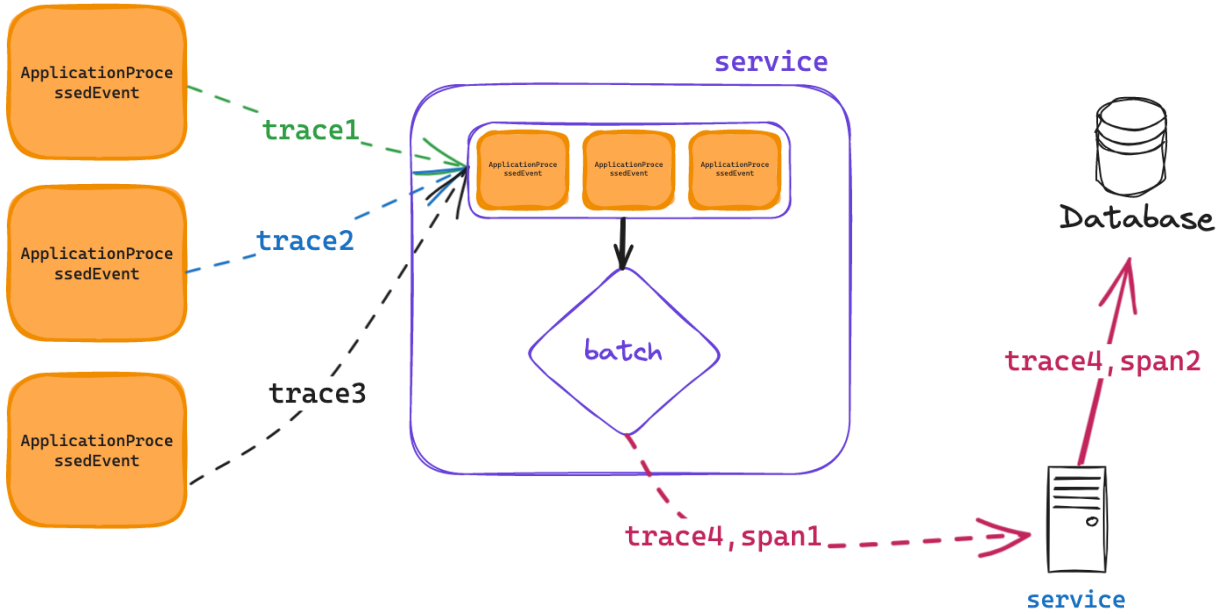
Gdzie tracing się gubi ?

Wszędzie tam gdzie nie jesteśmy w stanie przepropagować traceparent dalej

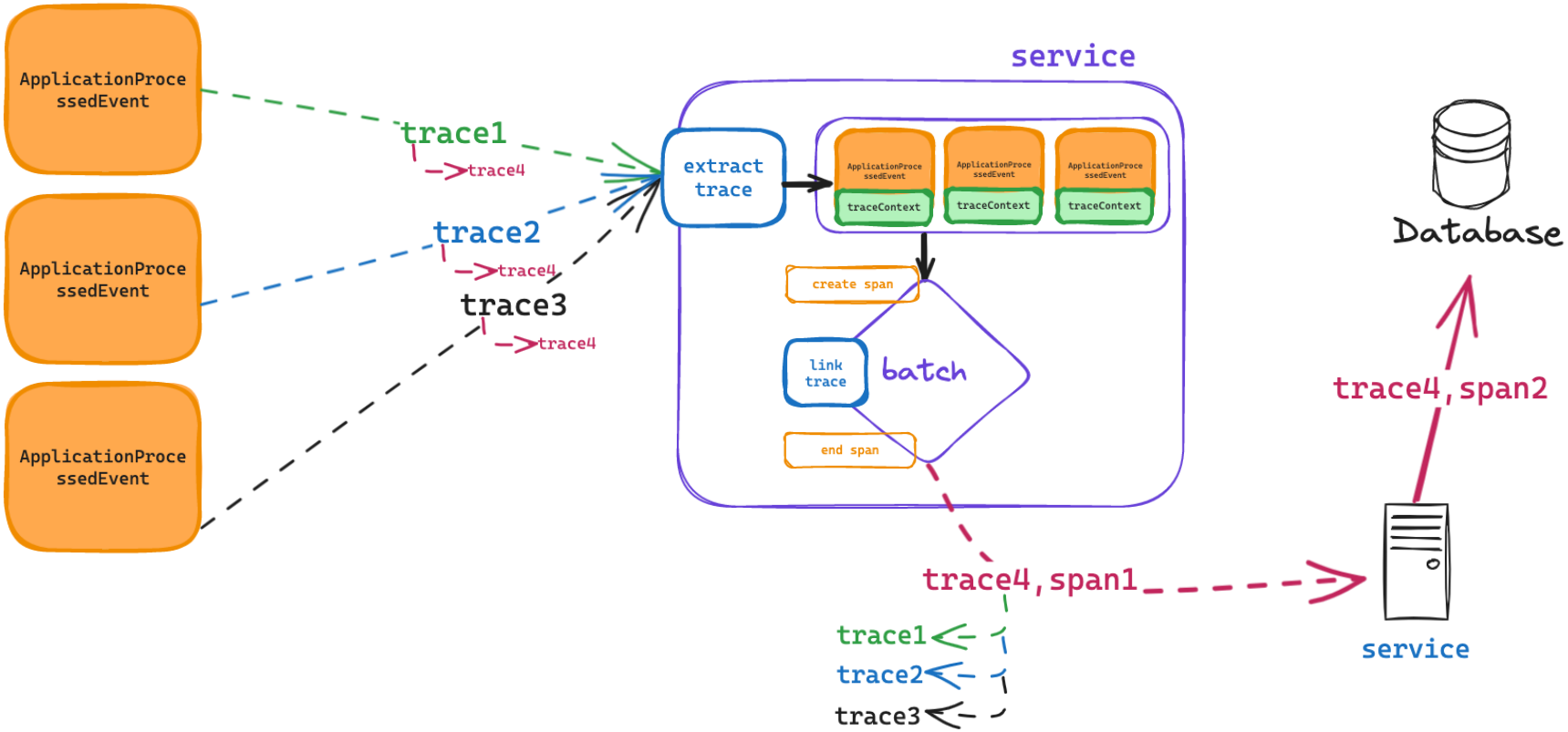
- Ręczne startowanie wątku
- Outboxy/Inbox (Transakcyjne wysyłanie wiadomości)
- Procesy batchowe

Co jeśli mamy proces batchowy i chcemy wiedzieć co się działo przed nim? - automat

Własne rozwiązanie:

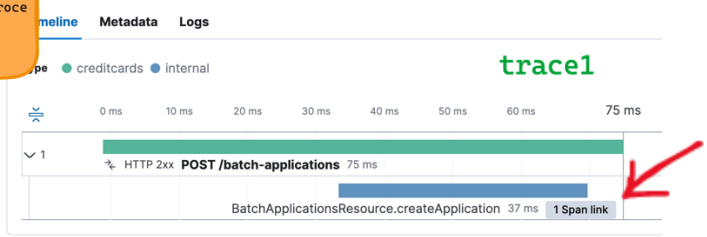


Co jeśli mamy proces batchowy? – automat + manual

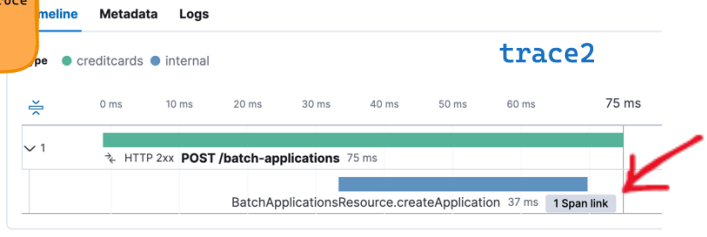


Co jeśli mamy proces batchowy? – automat + manual - kibana

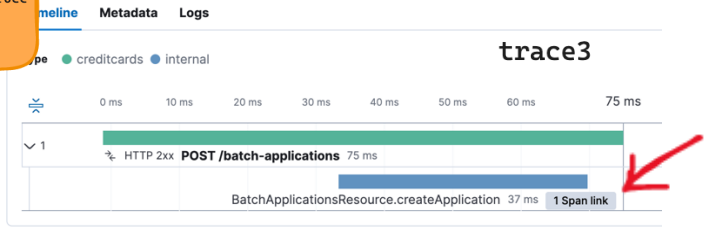
ApplicationProcessedEvent

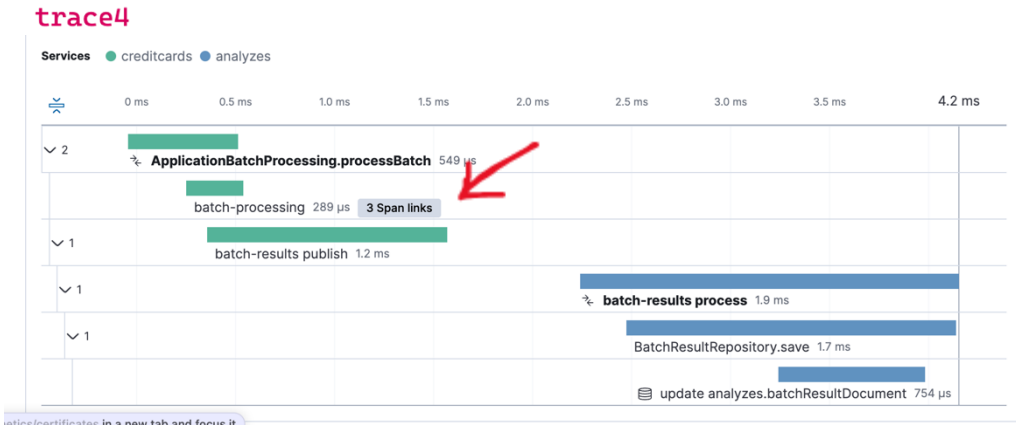


ApplicationProcessedEvent



ApplicationProcessedEvent





Gdzie tracing się gubi ?

Wszędzie tam gdzie nie jesteśmy w stanie przepropagować traceparent dalej

- Ręczne startowanie wątku
- Outboxy (Transakcyjne wysyłanie wiadomości)
- Procesy batchowe
- Statefull kafka stream Ktable

Co jeśli mamy Kafka Streams?

Streamy Stateless :

- Działają z automatu gdy mamy agenta
- Dlatego, że nie serializujemy/deserializujemy kontekstu do roksDB

Streamy Statfull – tutaj nie jest już tak różowo, trace jest przerwany w interakcji z roksDB

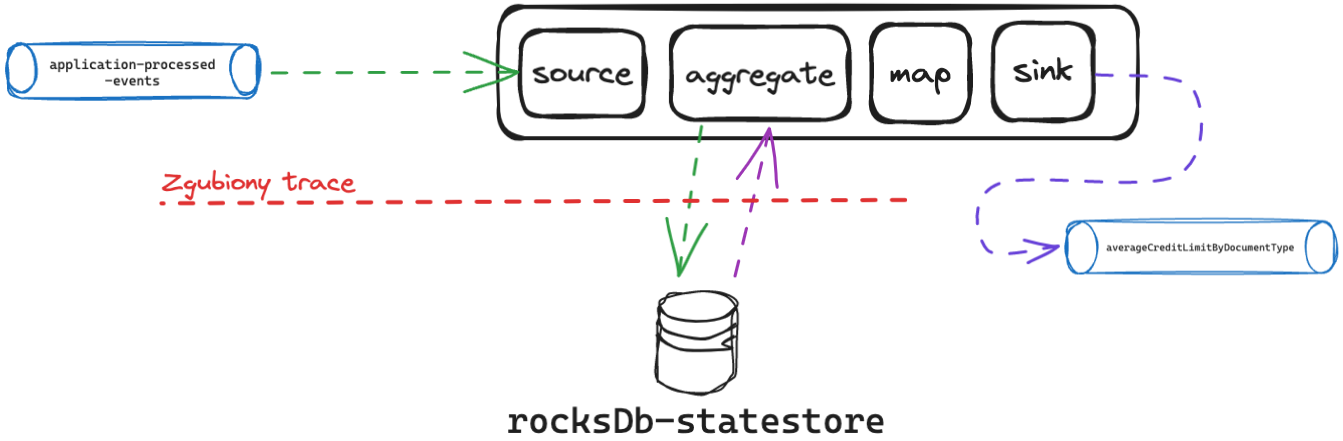
- Agregacje
- Agregacje w oknie czasowym
- Procesory zapisujące i odczytujące z stateStore

Co jeśli mamy statefull Kafka Streams?

Streamy Statefull Normalna agregacja – Auto instrumentacja

trace1 – – –

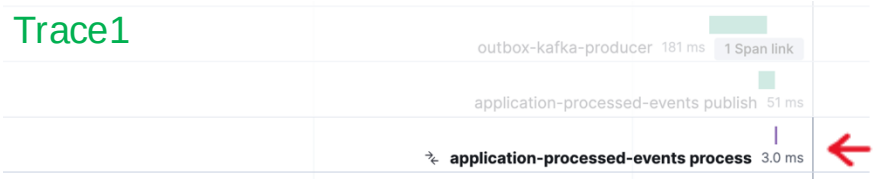
trace2 – – –



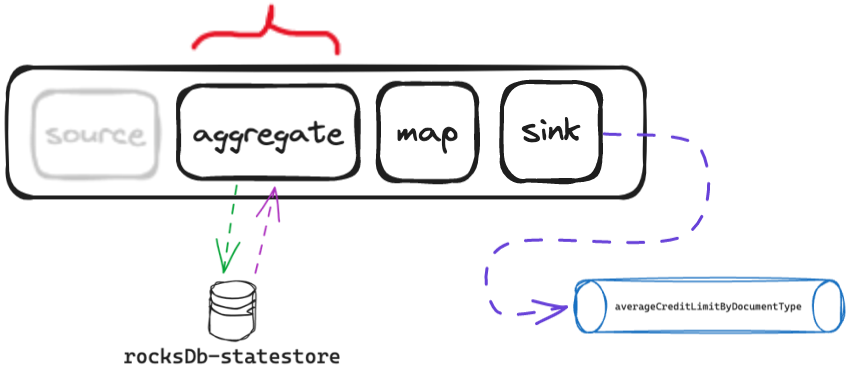
Co jeśli mamy statefull Kafka Streams ? - automat

Streamy Statfull – Auto instrumentacja

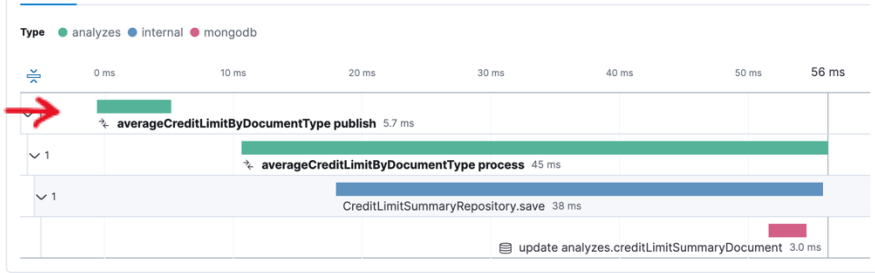
Trace1



Brak trace



Trace2

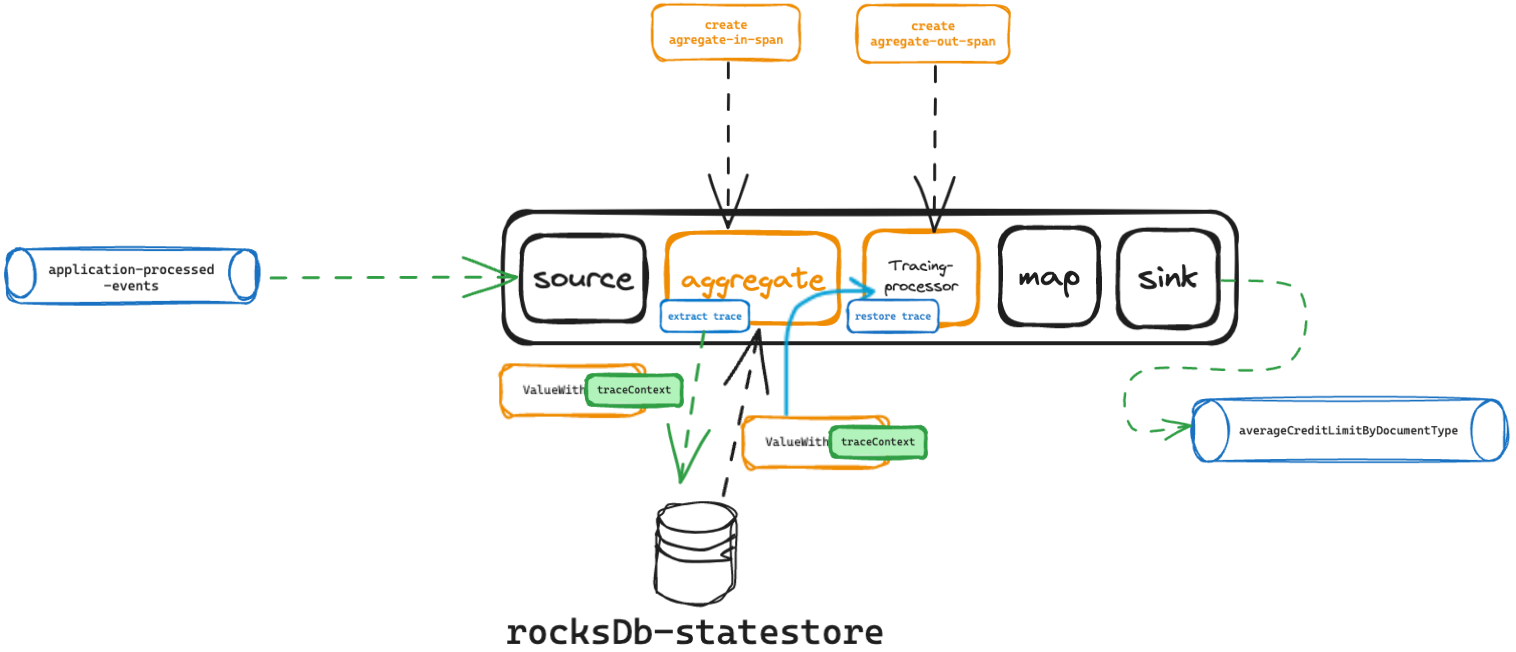


Co jeśli mamy statefull Kafka Streams? - poprawiony

Streamy Statfull – Auto + manual

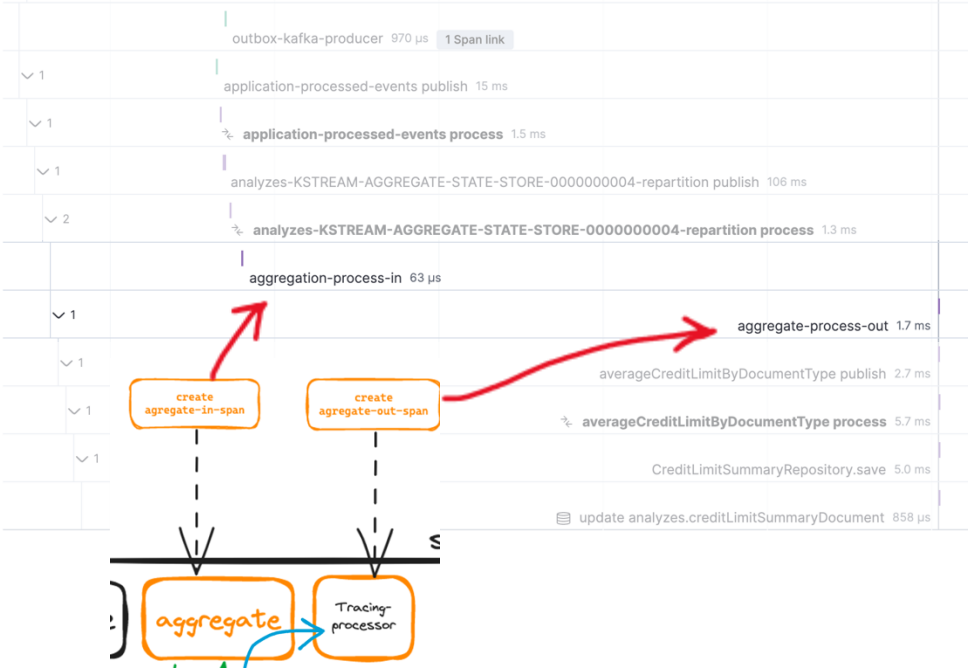
trace1 – – –

instrumentacja ręczna ———



Co jeśli mamy Kafka Streams? – wznowienie trace ręczne

Streamy Statfull – Auto + manual



Co jeśli mamy Kafka Streams?

Streamy Statfull – agregacje czasowe

- Trudne ☺

Gdzie tracing się gubi ?

Wszędzie tam gdzie nie jesteśmy w stanie przepropagować traceparent dalej

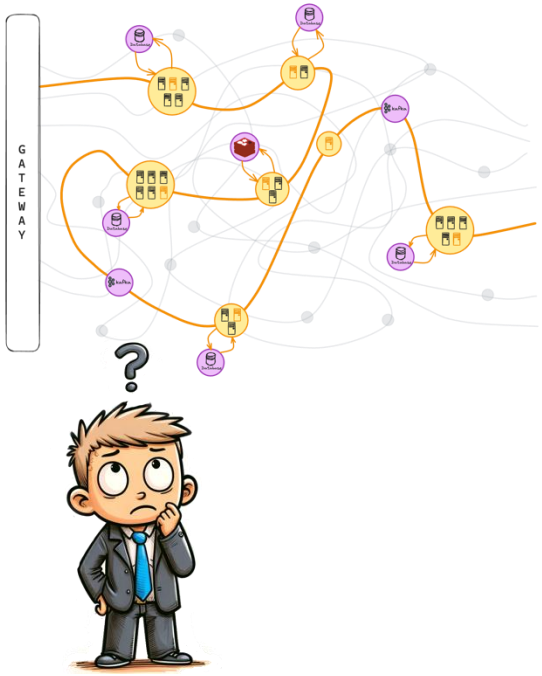
- Ręczne startowanie wątku
- Outboxy (Manualne i CDC np. debezium)
- Procesy batchowe
- Statefull kafka stream Ktable
- Gdy nie mamy instrumentacji z pudełka np. dla rsocketa
- Sagi

Podsumowując: wszędzie tam, gdzie tracimy metadane (traceparent) zapis/odczyt, producer/consumer

Czy zawsze musze dorabiać tą ręczna instrumentację ?

To oczywiście zależy ☺

- Czy jest to ważny flow, który chcesz monitorować od początku do końca?
- Jeśli nie, to może nie warto spalać na to czasu
- Głównym celem nie jest śledzenie długo trwających procesów np. ze względu na tailsampling, lub retencję (choć jak ktoś potrzebuje to można)

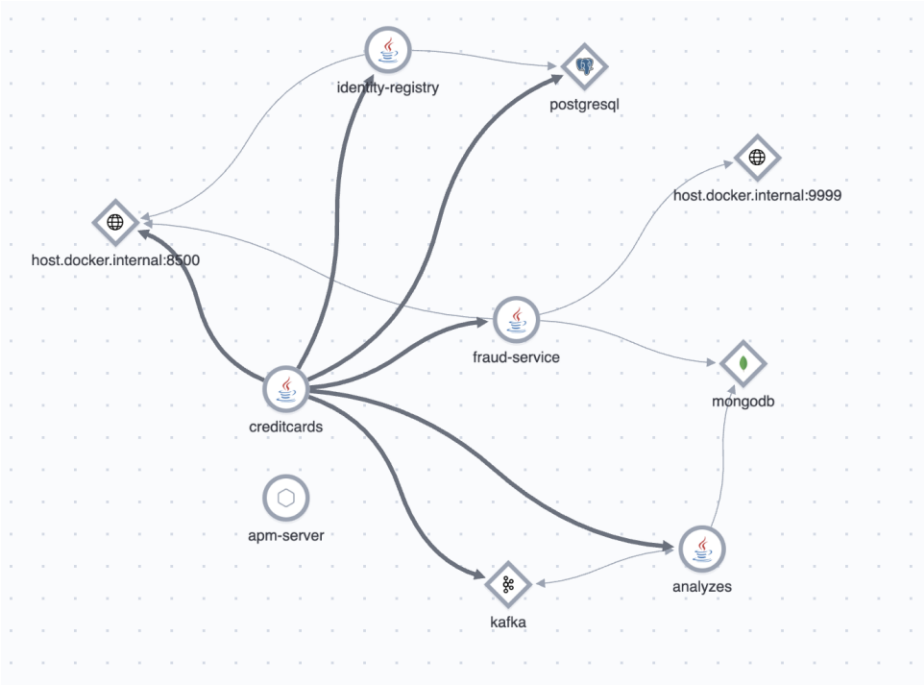


Nowa osoba w projekcie

OpenTelemetry jako pomoc w poznawaniu systemu

Dependency Map

Kibana



Grafana Tempo

Nowa osoba w projekcie

- Lepsze zrozumienie przepływów danych

  Widoczność całego **trace'a** (np. od klienta do bazy danych i z powrotem) pomaga zrozumieć, które części systemu są kluczowe dla danej funkcji.

- Łatwiejsze Debuggowanie problemów

  Zamiast analizować tysiące logów, nowa osoba może wykorzystać trace do szybkiej identyfikacji źródła błędów lub wąskich gardeł, oraz zobaczyć logi skorelowane z interesującego flow.

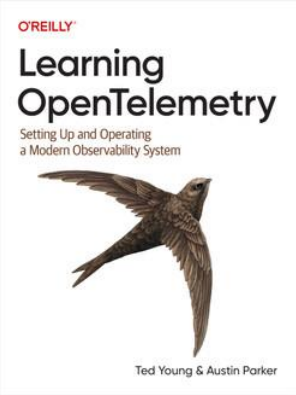
- Poznawanie zależności pomiędzy komponentami

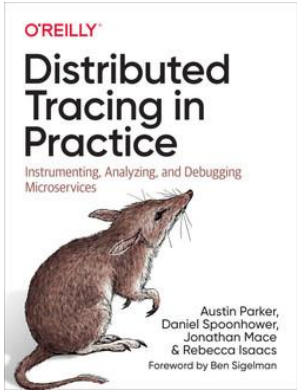
  Dzięki tracingowi można zobaczyć hierarchię wywołań (dependency chain), co pomaga nowej osobie szybk zorientować się, jakie serwisy komunikują się ze sobą i jakie dane są przesyłane.

Gdzie sięgnąć po więcej?

- *https://opentelemetry.io/docs/*
- *https://opentelemetry.io/blog/*
- *https://opentelemetry.io/docs/demo/*

- Książki





Gdzie mnie znaleźć ?

Blog: https://cupofcodes.pl

Linkedin: https://linkedin.com/in/kozaks/

Q&A & Ankieta

https://forms.gle/WYdy2VsRp1SkNYNa8

Kod: *https://cupofcodes.pl/talks/*

Dziękuję za uwagę ☺