

Как выбрать инструмент для трейсов в .NET?

Николай Савенко

.NET Разработчик, Altenar

Дата

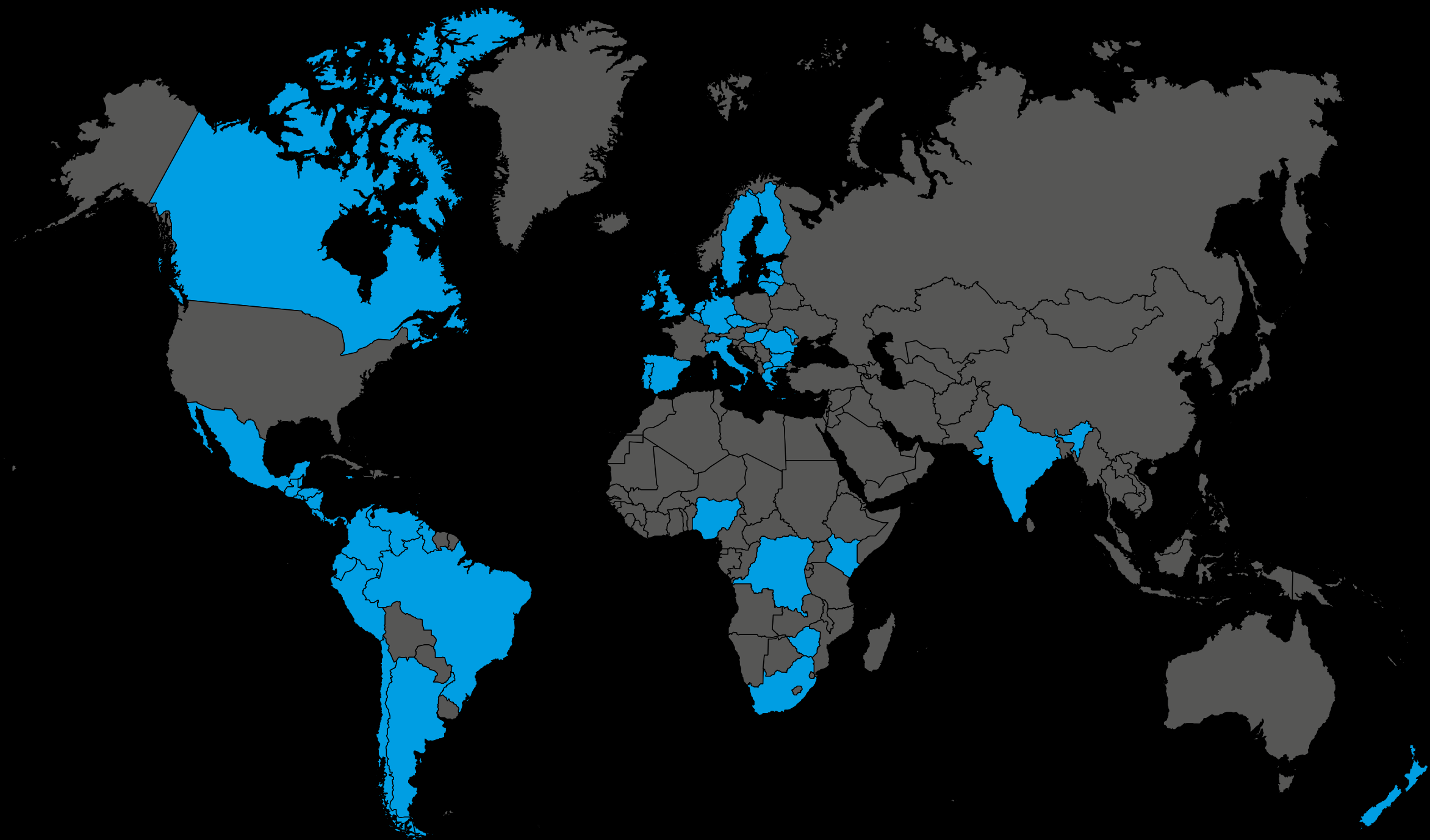
Николай Савенко

.NET Developer



Can break production at scale
Developer 12/10





10+ лет

на зарубежном рынке B2B ПО
для легальных операторов
прогнозов на спорт

780+ сотрудников

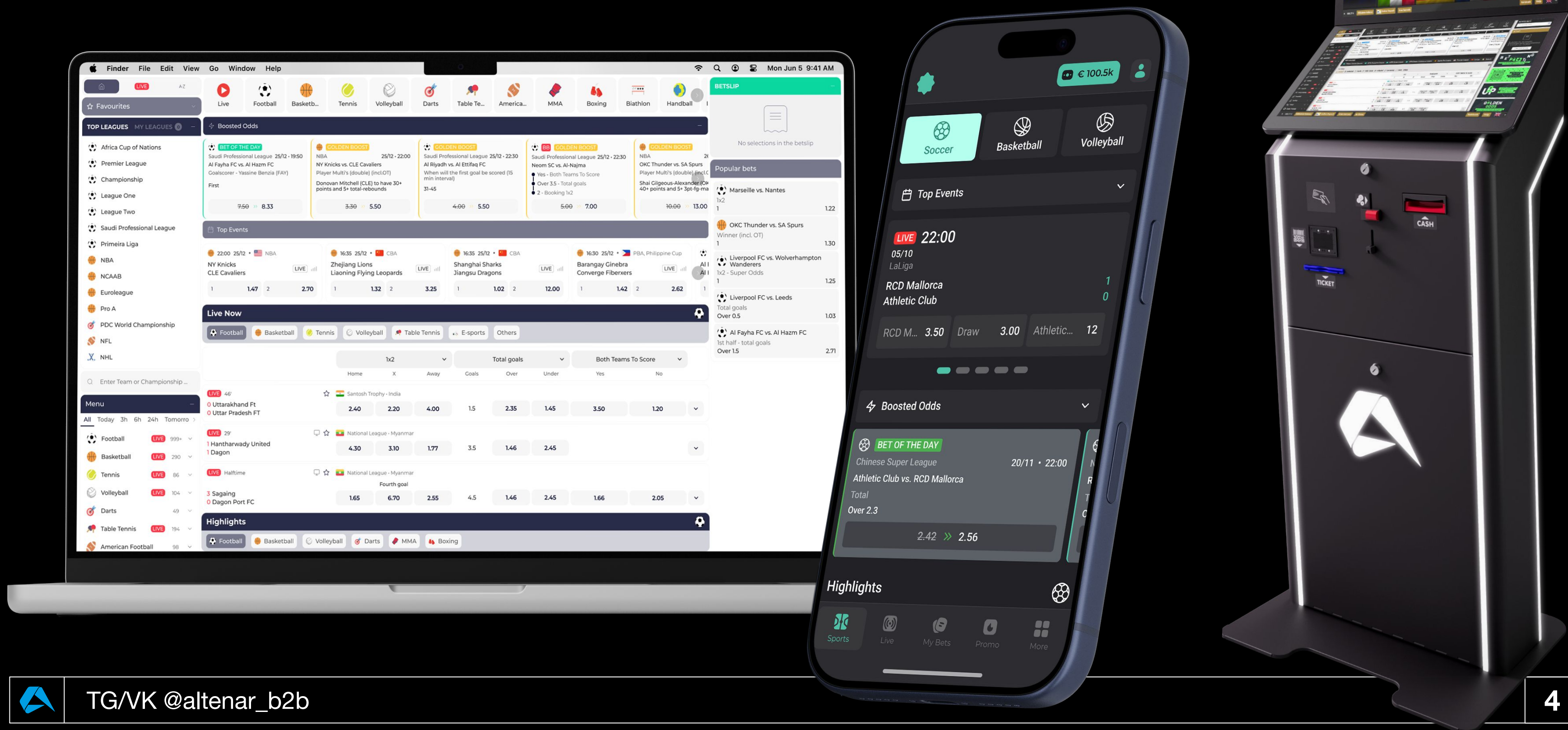
на удаленке и в офисах
в 6 странах

60k+ RPS

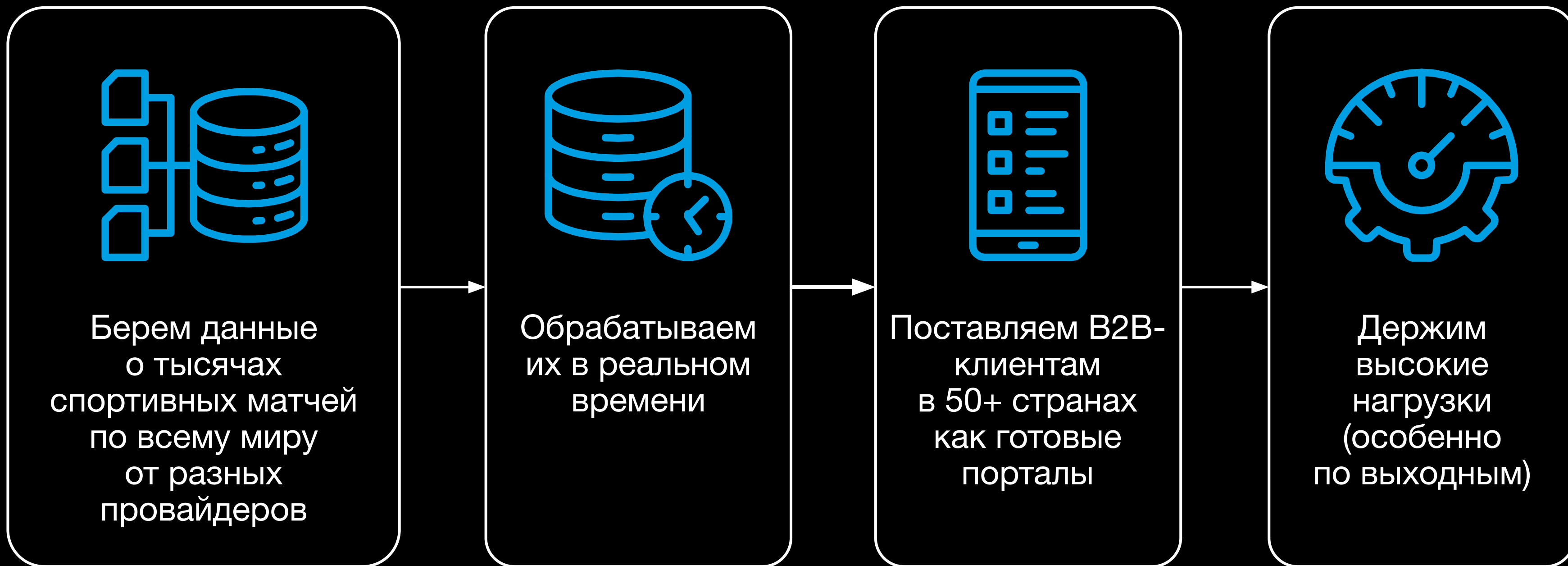
обрабатывают наши API
в дни Лиги чемпионов



Порталы, мобайл, ПО для терминалов



TG/VK @altenar_b2b



Что мы сегодня узнаем?

- Как мониторили раньше?
- Чем отличаются?
 - *ActivitySource* и как его готовить
 - *DiagnosticSource*
 - *EventSource*
- Как внедрить в свой код?
- Как внедрить в чужой код?
- Какие могут быть проблемы?



Зачем трейсы, если есть логи?



Задача логов и трейсов

Логи:

- События в одном сервисе
- Одиночные операции или состояния
- Временная метка и уровень
- Нет информации о зависимостях между сервисами.

Трейсы:

- Связанные шаги в разных частях приложения.
- Parent-child связи между span'ами.
- Визуализируют всю цепочку выполнения.

Пример:

“Transfer 200 funds from user A to user B”.

Пример:

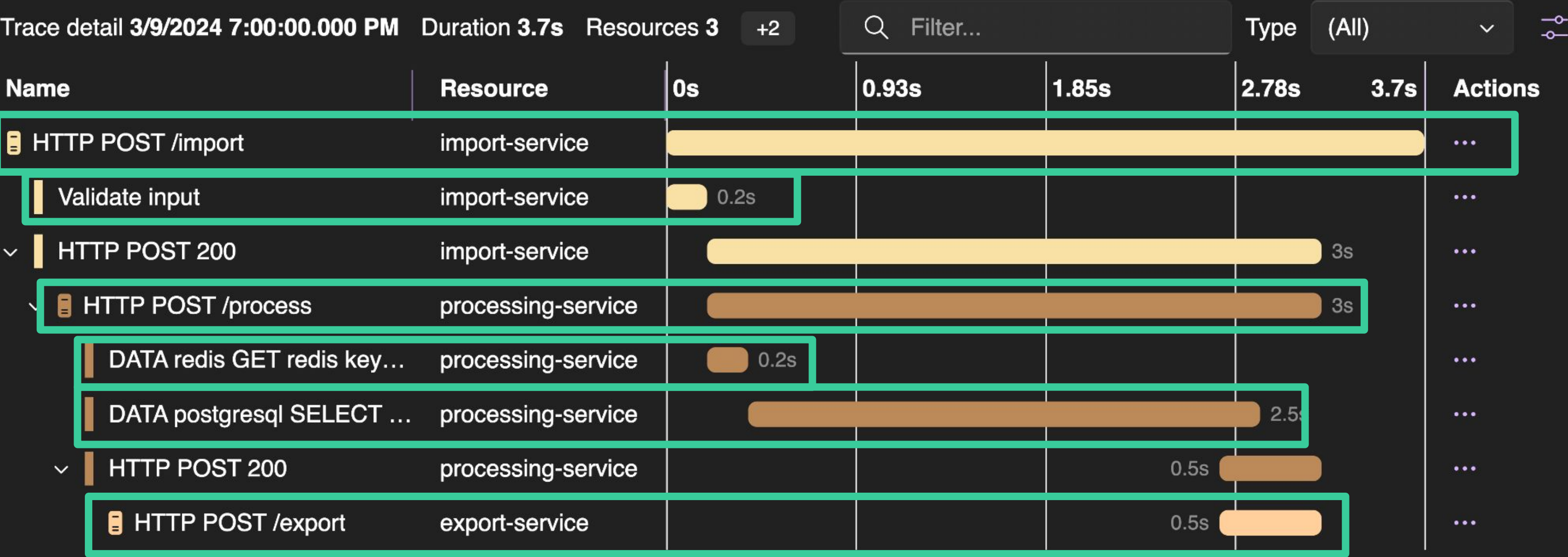
“HTTP запрос → обработка в сервисе → вызов БД → отправка сообщения в очередь”.

Чего мы хотим от трейсинга?

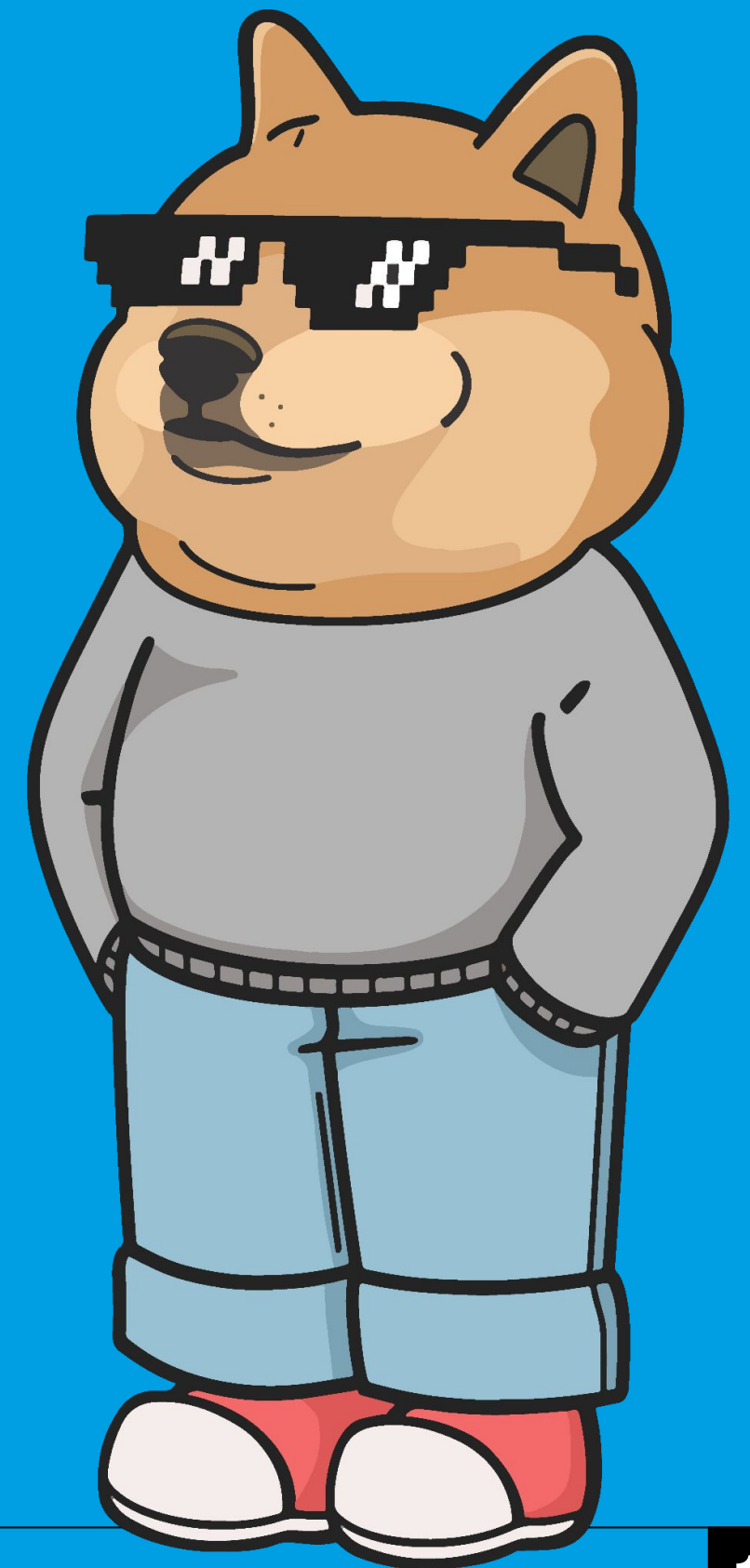
- Что за операция?
- Длительность?
- Связь с другими операциями?
- Распределенность
- Визуализация

Ожидания: 😎

import-service: HTTP POST /import eba71a7

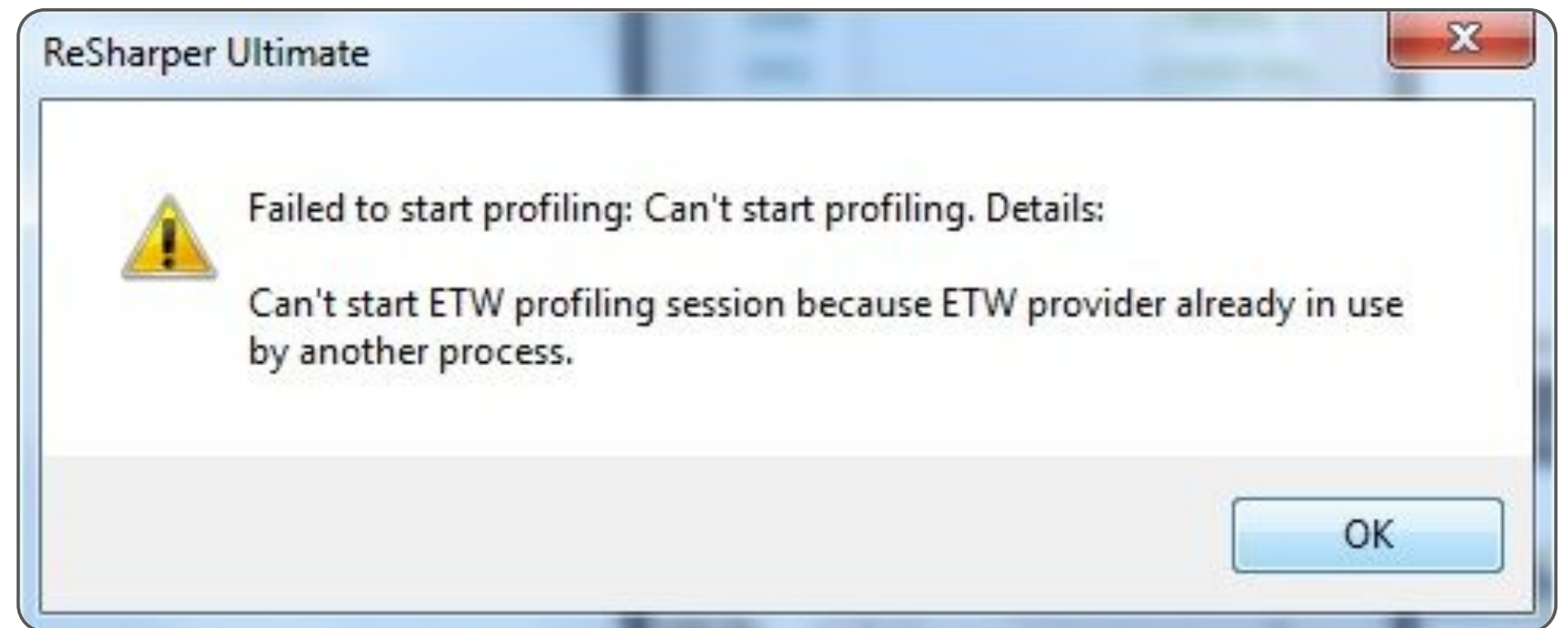


ЧТО МЫ ИМЕЕМ В .NET?



EventSource — когда трава была зеленой...

- ETW = Event Tracing for Windows
- Single Host
- Low Level API



EventSource — только события

```
[EventSource(Name = "MyCompany-MyLibrary")]
public sealed class MyEventSource : EventSource
{
    public static readonly MyEventSource Log = new();

    [Event(1, Message = "Processing with foo={0}, bar={1}", Level = EventLevel.Informational)]
    public void SomeWork(string foo, int bar) ⇒ WriteEvent(1, foo, bar);
}

static async Task DoSomeWork(string foo, int bar)
{
    MyEventSource.Log.SomeWork(foo, bar);

    await StepOne();
    await StepTwo();
}
```



Создаем свой EventListener

```
class ConsoleWriterEventListener : EventListener
{
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        if (eventSource.Name == "MyCompany-MyLibrary")
        {
            EnableEvents(eventSource, EventLevel.Informational);
        }
    }

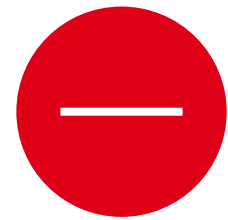
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        Console.WriteLine(eventData.Timestamp + " " + eventData.EventName);
    }
}
```

Создаем экземпляр Listener

```
public static void Main(string[] args)
{
    ConsoleWriterEventListener listener = new ConsoleWriterEventListener();

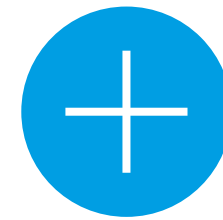
    DemoEventSource.Log.SomeWork("Hello!", "World!");
}
```

EventSource



Недостатки

- Не предназначен для distributed tracing
- Много шаблонного кода
- Не для бизнес событий



Преимущества

- низкий overhead
- устоявшийся API

Порешали вопросы?

- Что за операция? 
- Длительность? 
- Связь с другими операциями? 
- Распределенность 
- Визуализация 

DiagnosticSource — шаг вперед

```
static readonly DiagnosticListener listener = new("MyCompany.MyLibrary");

static async Task DoSomeWork(string foo, int bar)
{
    if (listener.IsEnabled("SomeWork"))
    {
        var payload = new { foo, bar };
        listener.Write("SomeWork.Start", payload);

        await StepOne();
        await StepTwo();

        listener.Write("SomeWork.Stop", payload);
    }
    else
    {
        await StepOne();
        await StepTwo();
    }
}
```

Как подписаться на нужный?

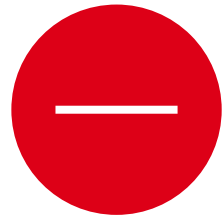
```
DiagnosticListener.AllListeners.Subscribe(listener =>
{
    if (listener.Name == "MyCompany.MyLibrary")
    {
        listener.Subscribe(new MyDiagnosticObserver());
    }
});
```

Как подписаться на нужный?

```
class MyDiagnosticObserver : IObserver<KeyValuePair<string, object>>
{
    public void OnCompleted() ⇒ throw new NotImplementedException();
    public void OnError(Exception error) ⇒ throw new NotImplementedException();

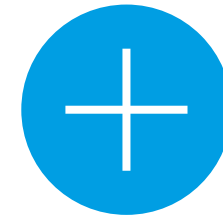
    public void OnNext(KeyValuePair<string, object> value)
    {
        var payload = (MyPayload)value.Value;
        Console.WriteLine($"Example {payload.GetType().Name} ");
    }
}
```

DiagnosticSource



Недостатки

- Нет встроенной поддержки иерархической трассировки



Преимущества

- Можно передавать сложные структуры данных
- Есть начало и конец

Стало лучше?

- Что за операция? 
- Длительность? 
- Связь с другими операциями? 
- Распределенность 
- Визуализация 

ActivitySource



ActivitySource — актуальный стандарт

```
static readonly ActivitySource source = new("MyCompany.MyLibrary");
```

```
static async Task DoSomeWork(string foo, int bar)  
{
```

```
    using Activity activity = source.StartActivity("SomeWork");
```

```
    activity?.SetTag("foo", foo);
```

```
    activity?.SetTag("bar", bar);
```

```
    await StepOne();
```

```
    await StepTwo();
```

```
}
```



Как подписаться?

```
var listener = new ActivityListener
{
    ShouldListenTo = (source) => source.Name == "MyCompany.MyProduct",
    Sample = (ref options) => ActivitySamplingResult.AllData,
    ActivityStarted = activity => Console.WriteLine($"Started: {activity.OperationName}"),
    ActivityStopped = activity => Console.WriteLine($"Stopped: {activity.OperationName}"),
};

// Register
ActivitySource.AddActivityListener(listener);
```

Почему Span называются Activity?



Note

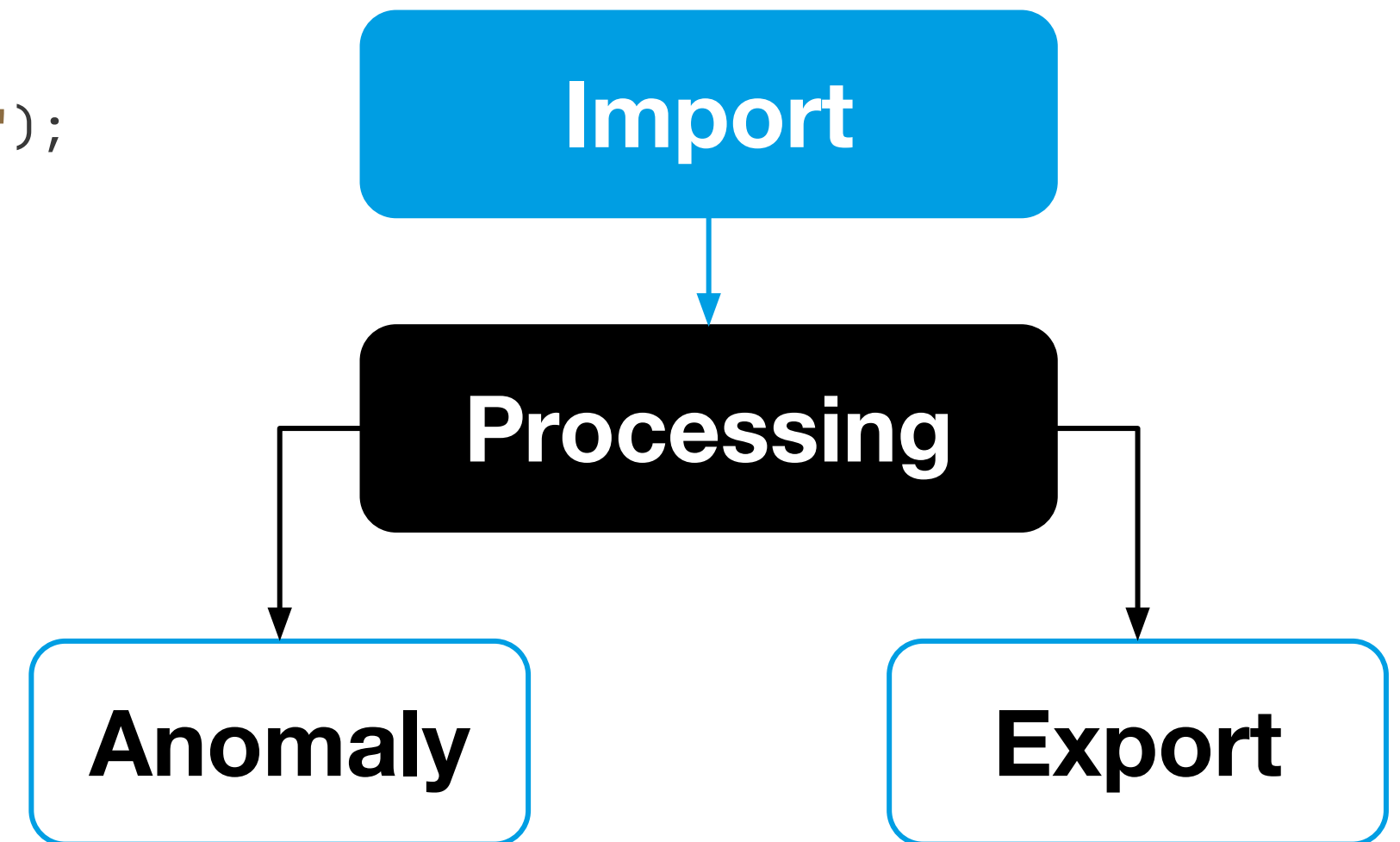
Another common industry name for units of work in a distributed trace are 'Spans'. .NET adopted the term 'Activity' many years ago, before the name 'Span' was well established for this concept.

Связанность разных Activity

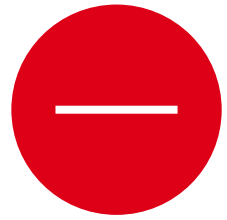
```
using var parent = source.StartActivity("Import");  
using var processing = source.StartActivity("Processing");
```

```
using var anomaly = source.StartActivity(  
    "Anomaly",  
    parentContext: processing.Context);
```

```
using var export = source.StartActivity(  
    "Export",  
    parentContext: processing.Context);
```

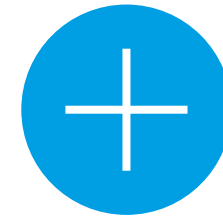


ActivitySource



Недостатки

- Неявный контекст - Activity.Current
- Нет “Trace” как объекта
- Сложность тестирования



Преимущества

- Поддержка Distributed Tracing
- Простота внедрения
- Otel из коробки



А сейчас все закрыли?

- Что за операция? 
- Длительность? 
- Связь с другими операциями? 
- Распределенность 
- Визуализация 

Distributed Tracing



Все Activity связаны через TraceContext



Где находится контекст?

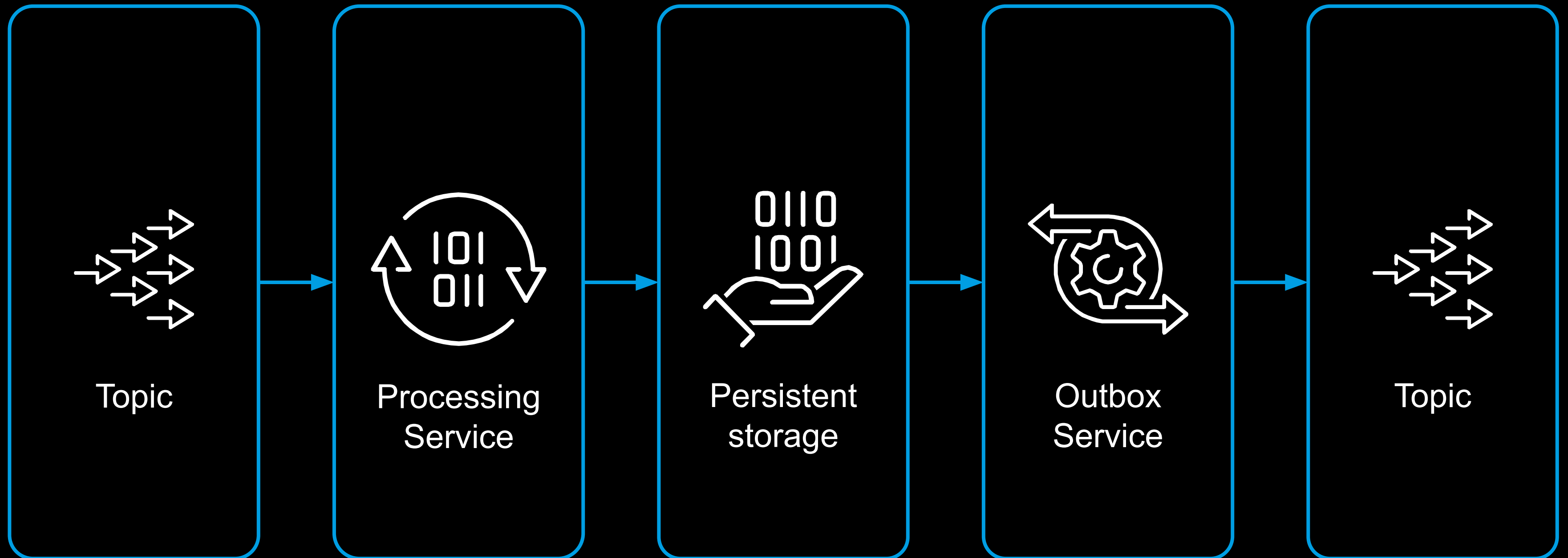
- HTTP Headers
- Message Headers
- Database Columns
- etc...



Context Propagation



Transactional Outbox наносит ответный удар



Как достичь Context Propagation?

- TraceParent
 - *version-traceid-spanid-flags*
- Example
 - *00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01*

// Service A

```
var traceparent = Activity.Current.Id;
```

// Service B

```
if (ActivityContext.TryParse(traceparent, tracestate: null, out var parentContext))  
{  
    using var activity = MyActivitySource.StartActivity(  
        "SendMessage",  
        ActivityKind.Producer,  
        parentContext);  
    activity?.SetTag("messaging.system", "kafka");  
}
```



Почти победа

- Что за операция? 
- Длительность? 
- Связь с другими операциями? 
- Распределенность 
- Визуализация 

Визуализация?



На любой вкус и цвет



Vendor Lock

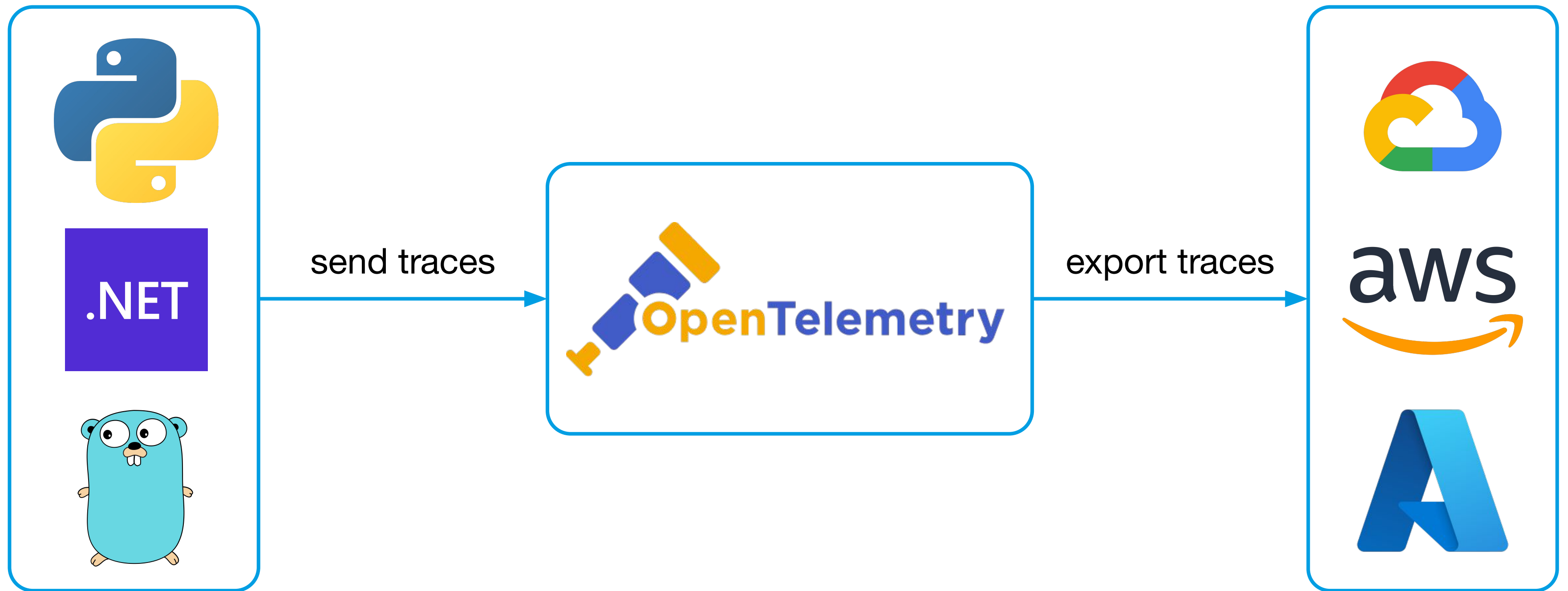


Один протокол, чтоб править всеми

- Logs
- Metrics
- Traces



ActivitySource + OpenTelemetry = ❤️

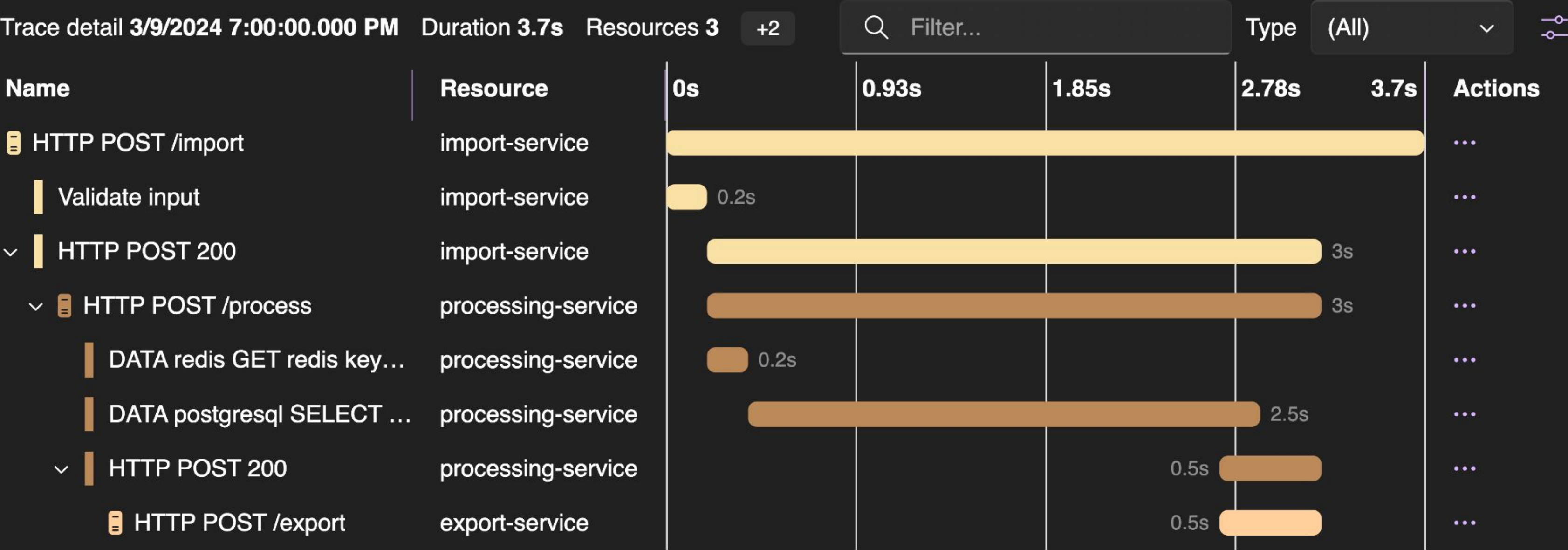


OpenTelemetry + Google Cloud Trace

Name Collapse all	Service/workload ?	0s86.285ms172.57ms258.854ms345.13
▼ Trace ID 0a49adced662b8bf09cf10accf0a36f...		345.139ms
└─▼ message_handle	LineBlocking	69.483ms
└─└─▼ SendMessage	Generic.Outbox.Service.Kafka	10ms
└─└─└─▼ ProduceAndRetryIfNeededAsync	Generic.Outbox.Service.Kafka	7.317ms
└─└─└─└─ ConsumeAndRetryIfNeededAsync	MatchBlocker	19.6μs
└─└─▼ SendMessage	Generic.Outbox.Service.Kafka	10.328ms
└─└─└─▼ ProduceAndRetryIfNeededAsync	Generic.Outbox.Service.Kafka	7.663ms
└─└─└─└─ ConsumeAndRetryIfNeededAsync	Export.Markets.Soccer	14.2μs

Ожидания: 😎

import-service: HTTP POST /import eba71a7



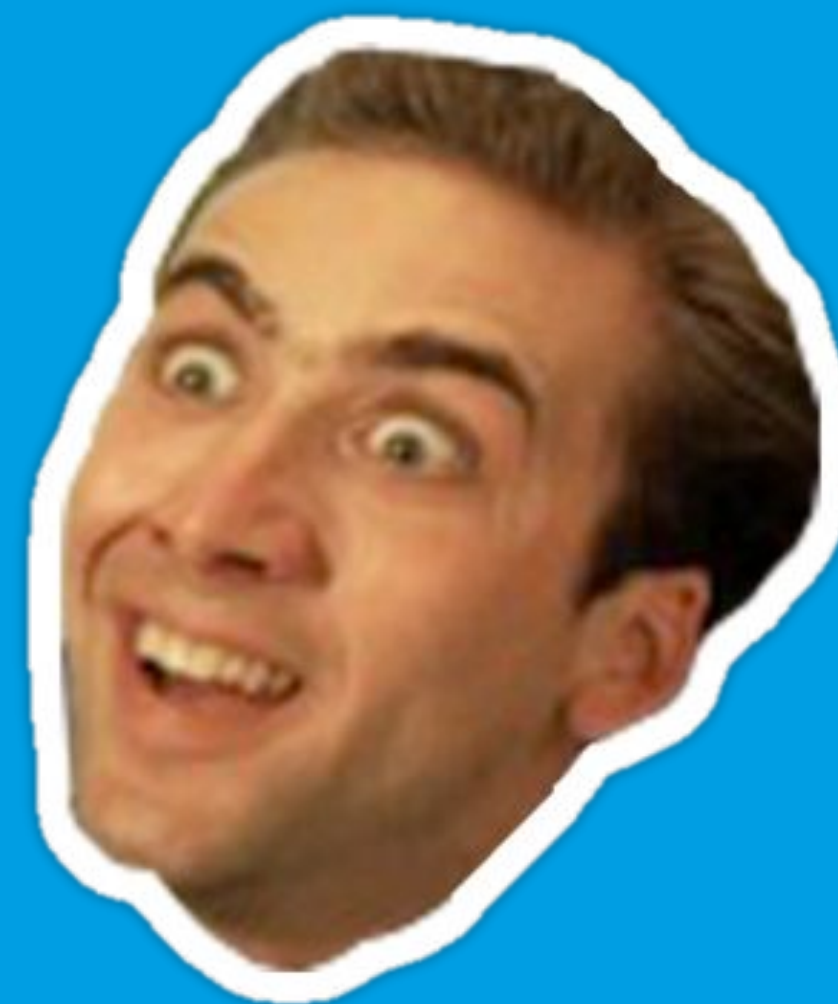
А дальше что?

- Что за операция? 
- Длительность? 
- Связь с другими операциями? 
- Распределенность 
- Визуализация 

Хотим трейдинг для gRPC, EF, HttpClient



Как обстоят дела в реальности?



Все уже перешли на ActivitySource?

До сих пор на DiagnosticSource:

- Entity Framework Core
- System.Net.Http.HttpClient
 - Активно создает Activity, но не через ActivitySource
- SqlClient
- MongoDB.Driver
- gRPC
 - до .NET 7 использует DiagnosticSource
- MassTransit, MediatR, Hangfire, Quartz.NET, etc...



Сценарий 1: Компонент использует ActivitySource

```
builder.Services.AddOpenTelemetry()  
    .WithTracing(builder =>  
    {  
        builder  
            .AddSource("ComponentName")  
            .AddOtlpExporter(); // или другой экспортер  
    });
```

Сценарий 2: DiagnosticSource и есть библиотека инструментов

```
builder.Services.AddOpenTelemetry()  
    .WithTracing(builder =>  
    {  
        builder  
            .AddXyzInstrumentation()  
            .AddOtlpExporter();  
    });
```

Сценарий 3: 🙄 Нет готовой библиотеки инструментации

```
DiagnosticListener.AllListeners.Subscribe(listener =>
{
    if (listener.Name == "Component.Name")
    {
        listener.Subscribe(new MyDiagnosticObserver());
    }
});
```

Сценарий 3: 🤪 Нет готовой библиотеки инструментации

```
// MyDiagnosticObserver.cs
public void OnNext(KeyValuePair<string, object?> value)
{
    using var activity = MySource.StartActivity("SomeOperation",
ActivityKind.Internal);
    activity?.SetTag("custom.tag", value.Key);
}
```

Сценарий 4: 😐 Компонент использует EventSource

```
class CrunchEventListener : EventListener
{
    static readonly ActivitySource source = new("MyCompany.MyLibrary");

    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        if (eventSource.Name == "MyCompany-MyLibrary")
        {
            EnableEvents(eventSource, EventLevel.Informational);
        }
    }

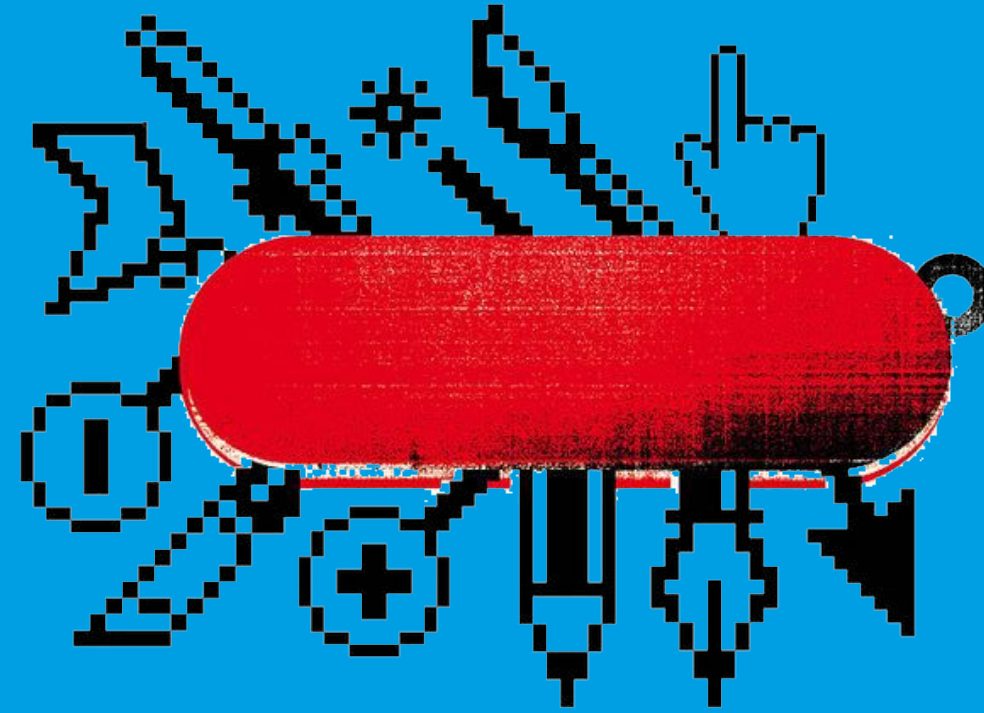
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        using var activity = source.StartActivity("SomeWork");
    }
}
```

Сценарий 5: 🤔
Компонент
не использует
ничего



Сценарий 5.1: 🙌 Есть доступ к коду

```
static readonly ActivitySource MySource = new("MyApp.Custom");  
  
using var activity = MySource.StartActivity("ProcessSomething",  
ActivityKind.Internal);  
activity?.SetTag("custom.key", "value");
```



Auto-Instrumentation



Сценарий 5.2: 🙅 Нет доступа к коду

- `dotnet add [<PROJECT>] package OpenTelemetry.AutoInstrumentation`
- Dockerfile
- OpenTelemetry Operator





```
FROM mcr.microsoft.com/dotnet/sdk:8.0
# install OpenTelemetry .NET Automatic Instrumentation
ARG OTEL_VERSION=1.12.0
ENV OTEL_DOTNET_AUTO_HOME=/otel-dotnet-auto
ADD
https://github.com/open-telemetry/opentelemetry-dotnet-instrumentation/releases/download/v${OTEL_VERSION}/otel-dotnet-auto-install.sh otel-dotnet-auto-install.sh
RUN apt-get update && apt-get install -y unzip && \
    sh otel-dotnet-auto-install.sh
RUN chmod +x /otel-dotnet-auto/instrument.sh

WORKDIR /app
COPY . .
RUN dotnet publish -c Release -o out
```

IL injection

CLR загружает агент
как Profiler

Profiler подписывается на
события JIT компиляции

Profiler переписывает
сгенерированный IL

IL выполняется

```
CORECLR_ENABLE_PROFILING=1  
CORECLR_PROFILER={guid}  
CORECLR_PROFILER_PATH=OpenTelemetry.  
AutoInstrumentation.Native.so  
OTEL_DOTNET_AUTO_HOME=/opt/opentelem  
etry/instrumentation/net
```



Что поддерживается?

- ASP.NET Core
- ASP.NET (System.Web)
- HttpClient / System.Net.Http
- gRPC
- SQL Client
- Entity Framework Core
- Redis (StackExchange.Redis)
- MongoDB.Driver
- etc...



И на фронте МОЖНО?



Blazor Server — проecto ASP.NET Core

```
builder.Services.AddOpenTelemetry()  
    .WithTracing(tracing => tracing  
        .AddAspNetCoreInstrumentation()  
        .AddSource("MyBlazorApp")  
        .AddOtlpExporter());
```

Blazor Server — просто ASP.NET Core

```
// В компоненте .razor
@inject ActivitySource ActivitySource

@code {
    private async Task HandleClick()
    {
        using var activity = ActivitySource.StartActivity("Button.Click");
        activity?.SetTag("component", nameof(Counter));
        await SomeService.DoWorkAsync();
    }
}
```

Blazor WASM — «браузерная» история

```
// Program.cs (Blazor WASM)
builder.Services.AddOpenTelemetry()
    .WithTracing(tracing => tracing
        .AddSource("MyBlazorWasmApp")
        .AddHttpClientInstrumentation() // инструментирует HttpClient-вызовы к API
        .AddOtlpExporter(opts =>
        {
            opts.Endpoint = new Uri("https://your-gateway/v1/traces");
            opts.Protocol =
                OtlpExportProtocol.HttpProtobuf; // gRPC не работает в браузере!
        }
    ));
```

Что обернуть в ActivitySource?



Отдельный ActivitySource

- Ваших библиотек транспорта
- Ваших shared-core библиотек
- Крупных подсистем

- `System.Data.SqlClient`
- `Microsoft.Data.SqlClient`
- `Azure.Messaging.ServiceBus`
- `Azure.Messaging.EventHubs`
- `System.Net.Http`



Отдельный Activity

- Получение
- Отправка
- Storage: БД запросы
- Internal
 - заметные стадии пайплайна

- . System.Net.Sockets.Socket.Connect
- . System.Net.Sockets.Socket.Accept
- . Grpc.Net.Client.GrpcOut
- . Grpc.AspNetCore.Server.GrpcIn
- . Azure.Messaging.ServiceBus.Send
- . Azure.Messaging.ServiceBus.Process
- . Azure.Messaging.EventHubs.Send
- . Azure.Messaging.EventHubs.Process
- . Microsoft.Data.SqlClient.Execute
- . System.Data.SqlClient.Execute

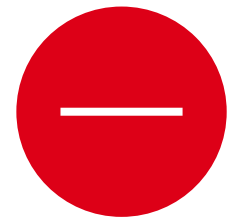
Хочу потрогать здесь и сейчас



Вариант «в лоб»

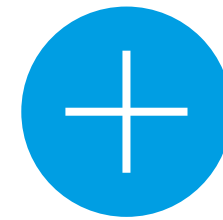
- Jaeger
- Zipkin
- ELK





Недостатки

- Немножко конфигов
- Не все можно запустить



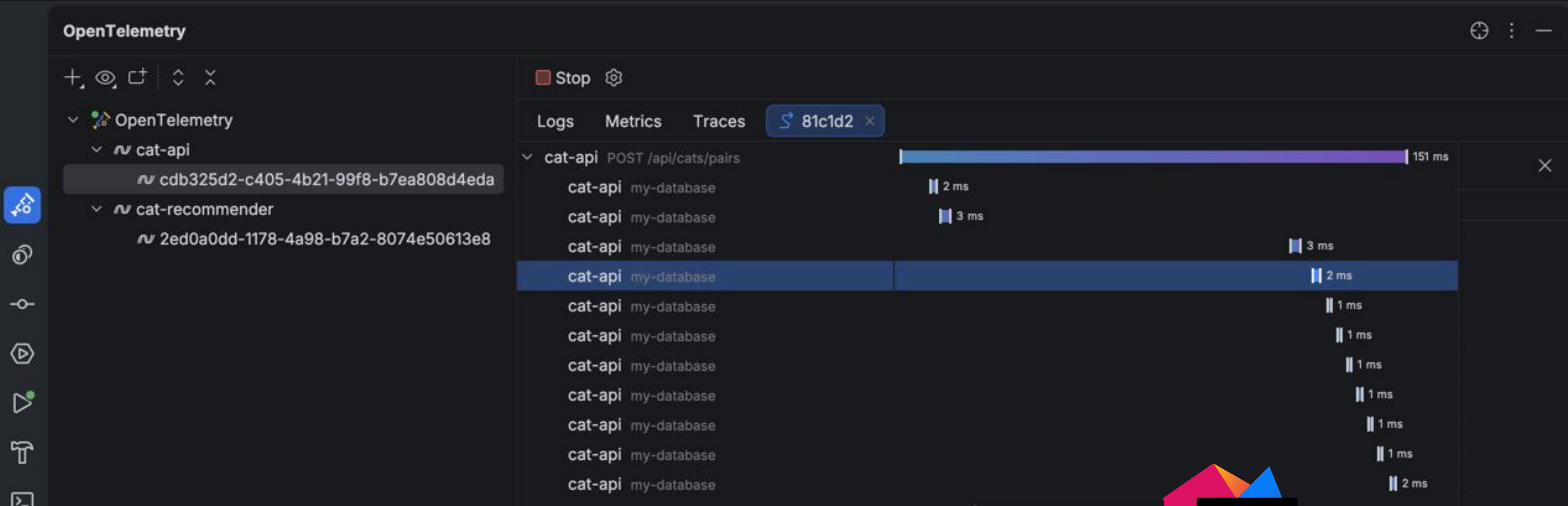
Преимущества

- Инфра похожа на реальную
- Воспроизводимость

Jetbrains Rider



Rider Plugin



Rider Plugin MCP

- get_log_records
- get_spans
- get_services
- get_service_map



Ran 2 commands, read Telemetry.cs, **used** a tool >

There есть `get_spans` и `get_services` — попробую посмотреть трейсы.

Used rider: execute tool >

Нашёл трейсы через `get_spans` — сервис `OtelRiderDemo.Api`, 13 спанов за последний прогон:

HTTP-запросы:

- `GET /` → 200 (78ms)
- `GET /favicon.ico` → 404
- `GET /orders` → 404 (такого роута нет)
- `GET /inventory/{id:int}` — 6 запросов, id=123:
 - 2× 404 (маршрут смэтчился, но ответ 404 — видимо, явный `NotFound`)
 - 4× 200

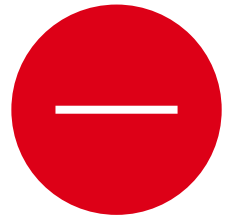
Внутренние спаны `inventory.lookup` (дочерние к запросам `/inventory/123`) показывают нестабильный результат для одного и того же `inventory.id=123`:

- `out_of_stock` (2 раза)
- `available`, qty=5 / 65 / 96 (3 раза, разное количество каждый раз)

Похоже на намеренную имитацию через `FailureInjectionOptions.cs` — случайные failure/latency инъекции для демонстрации трейсинга. Хотите, гляну код `InventoryEndpoints.cs`, чтобы подтвердить логику рандомизации?

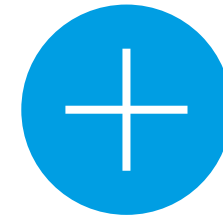


Rider Plugin



Недостатки

- Переписывает переменные



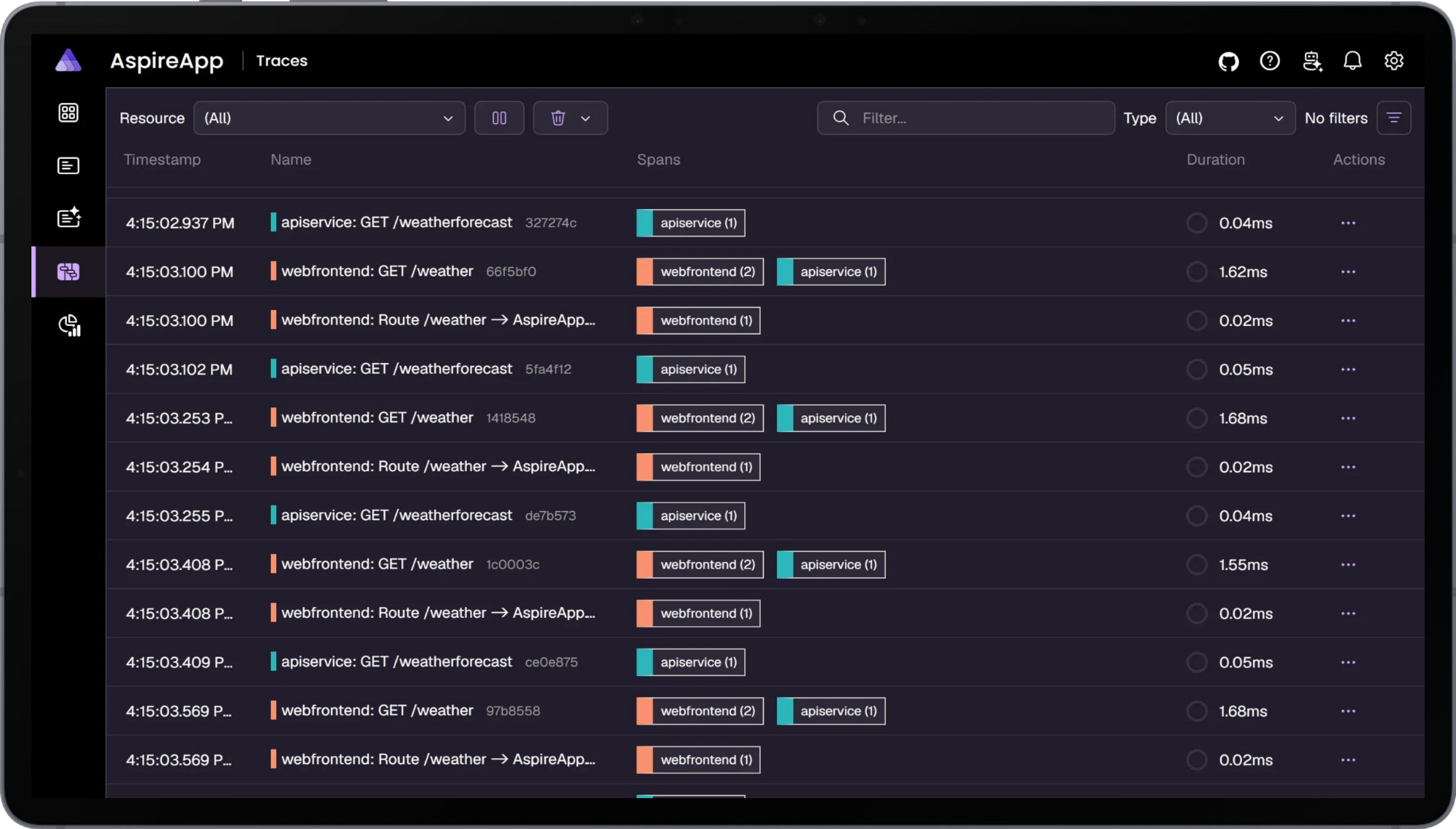
Преимущества

- Переписывает переменные
- Можно свой Collector
- MCP

Aspire Dashboard



Aspire Dashboard



**Так он мне
понравился**



Aspire MCP & Skill

- list_resources
- list_structured_logs
- list_traces
- list_console_logs



Log Trace Correlation



Базовый минимум



fields.SourceContext	ADF.Service.MatchBlocker.Logic.Hand
fields.SpanId	637e90cec1dcd8c1
fields.TraceId	e83b71ab53210e8c3e3be9d97ea92229
index_timestamp	2026-08-20T10:56:11.367025185Z

Serilog Enrichers

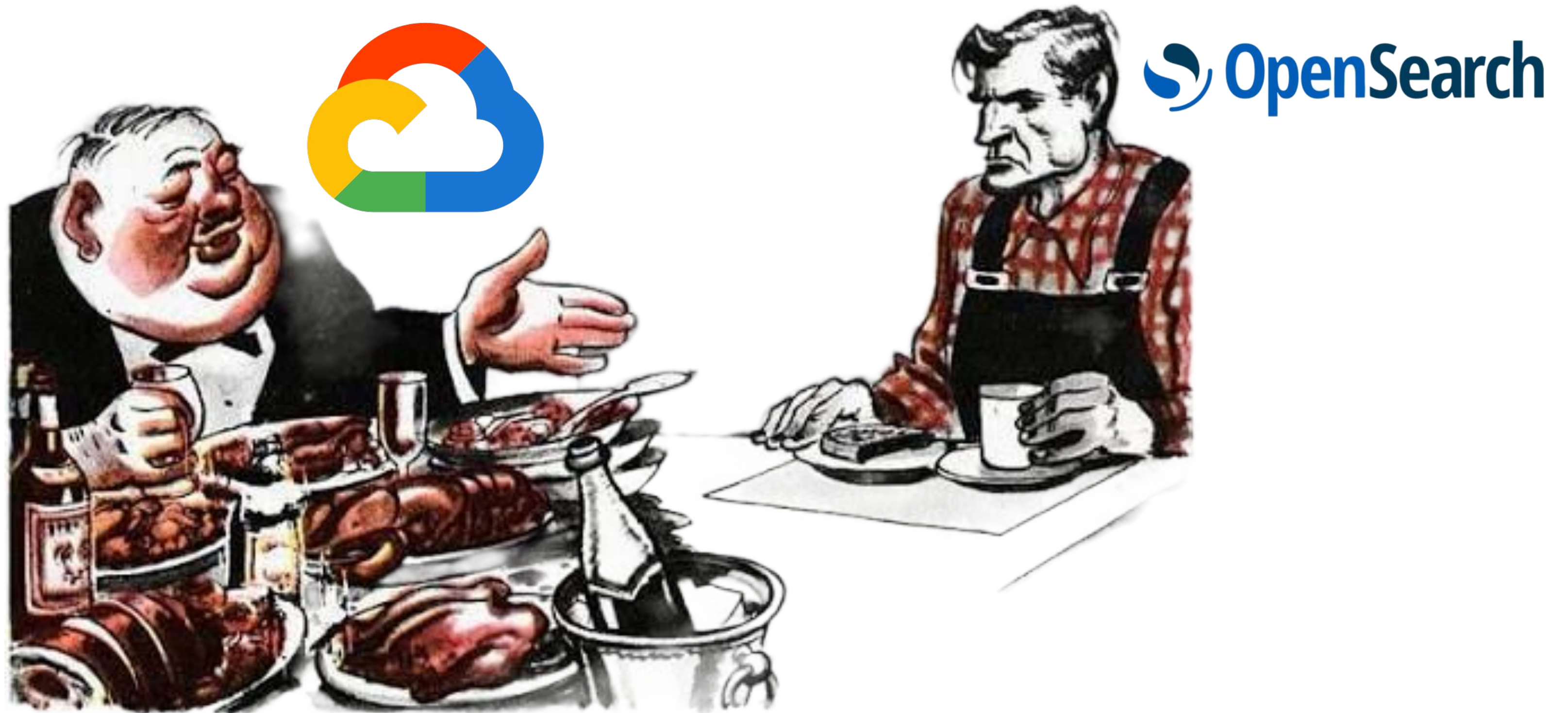
```
public sealed class ActivityTraceEnricher : ILogEventEnricher
{
    public void Enrich(LogEvent logEvent, ILogEventPropertyFactory propertyFactory)
    {
        var activity = Activity.Current;

        logEvent.AddPropertyIfAbsent(
            propertyFactory.CreateProperty("TraceId", activity.TraceId.ToString()));
        logEvent.AddPropertyIfAbsent(
            propertyFactory.CreateProperty("SpanId", activity.SpanId.ToString()));
    }
}
```

Logs+Traces у одного вендора



Свое дешевле роднее





Grafana Data Sources

- Elasticsearch
- Loki

- Jaeger
- Tempo
- Zipkin



Trace

Processing: Consume

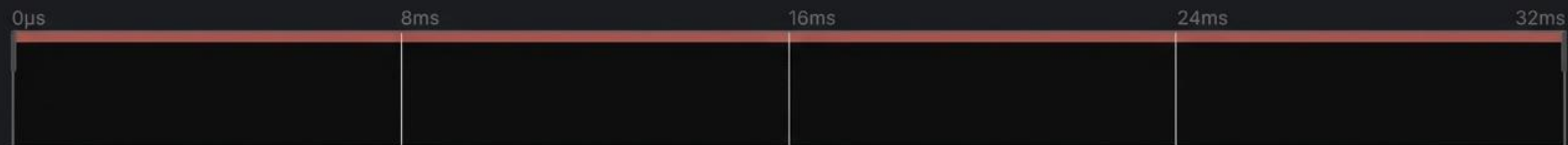
[Feedback](#)[Share](#)Trace ID [09407812cf53901048abb7cc74a06938](#)

Start time 2026-09-22 23:19:11.699 (12 minutes ago)

Duration 32ms

Services 1

Overview



Filters

Critical path

Errors

High latency

↑

↓

1 spans ⓘ

☒ Show all spans

Service & Operation

||

0

8ms

16ms

24ms

32ms

☒ Processing Consume (32ms)[Explore the logs for this in split view](#)

No volume information available for the current queries and time range.

Logs

Total lines: 19

Search fields by name

☒ Show log level

Fields

☐ _id	100%
☐ _index	100%
☐ _source	100%
☐ _type	0%
☐ fields.AppVersion	100%
☐ fields.Attempt	22%
☐ fields.Identifier	90%
☐ fields.KafkaPartition	11%
☐ fields.KafkaPartitionOffset	11%
☐ fields.KafkaTopic	11%
☐ fields.MachineName	100%
☐ fields.MatchId	58%
☐ fields.Scope	100%
☐ fields.ServiceName	100%
☐ fields.SourceContext	100%
☐ fields.SourceId	100%

```
← | : 2026-08-21 05:52:12.697 The offset is successfully committed
: 2026-08-21 05:52:12.697 Committed the offset
: 2026-08-21 05:52:12.697 INFO Report has been handled
: 2026-08-21 05:52:12.694 DEBUG Transaction committed successfully
: 2026-08-21 05:52:12.694 DEBUG Report changes handled
: 2026-08-21 05:52:12.694 DEBUG Instance disposed
: 2026-08-21 05:52:12.692 DEBUG Changes sent to database
: 2026-08-21 05:52:12.690 DEBUG Snapshot for match has been updated
: 2026-08-21 05:52:12.690 DEBUG Odds change report has been added to outbox
: 2026-08-21 05:52:12.688 The message is successfully consumed
: 2026-08-21 05:52:12.688 DEBUG Received new match report
: 2026-08-21 05:52:12.688 DEBUG Transaction created
: 2026-08-21 05:52:12.661 The offset is successfully committed
: 2026-08-21 05:52:12.661 Committed the offset
: 2026-08-21 05:52:12.661 INFO Update successfully processed
: 2026-08-21 05:52:12.658 WARN Skipped an unsupported message
: 2026-08-21 05:52:12.658 DEBUG Instance disposed
: 2026-08-21 05:52:12.657 The message is successfully consumed
: 2026-08-21 05:52:12.657 DEBUG Transaction created
```



A photograph of a person in a dark winter coat digging a deep pit in a snowy forest. The person is using a shovel to clear snow from around a dark, cylindrical object, possibly a barrel or a large container, which is partially buried in the snow. The background shows a dense forest of bare trees under a bright sky.

Копнем глубже

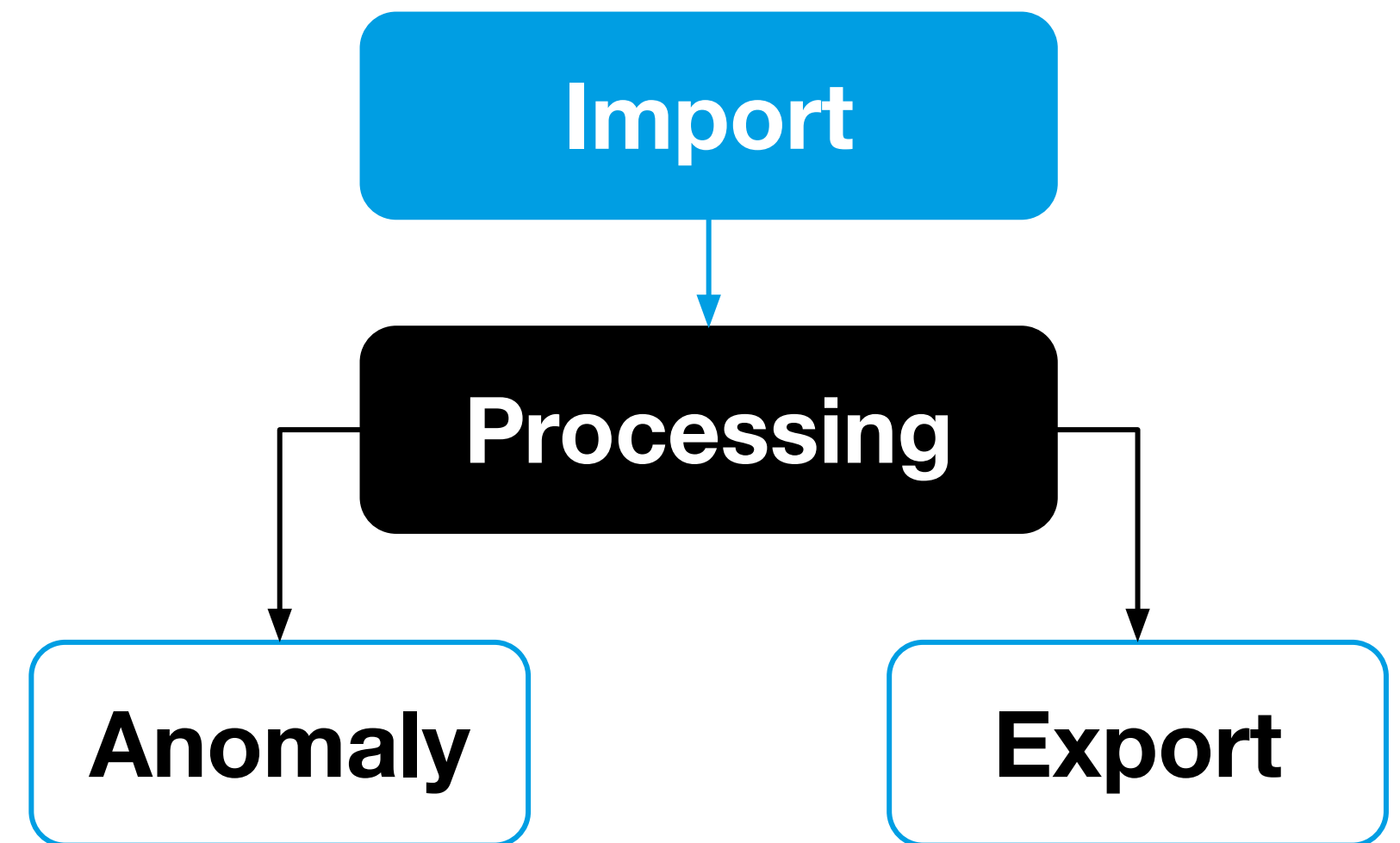
Context Propagation++



Хотим доп. данные

- match_id
- championship_id
- team_id

- user_id
- request_id
- tenant_id
- etc



A как же Tags?

Consume

Service: **adf-service-export-livestatistics**

Duration: 1.67ms

Start Time: 0µs (14:3

Library Name: **ADF.Library.Transport.Kafka**

Span attributes

messaging.kafka.attempt	"1"	
messaging.kafka.message.key	"411693"	
messaging.kafka.offset	"622850"	
messaging.operation.type	"receive"	
messaging.source.name	"soccer-match-data-live"	
messaging.source.partition.id	"19"	
messaging.system	"kafka"	

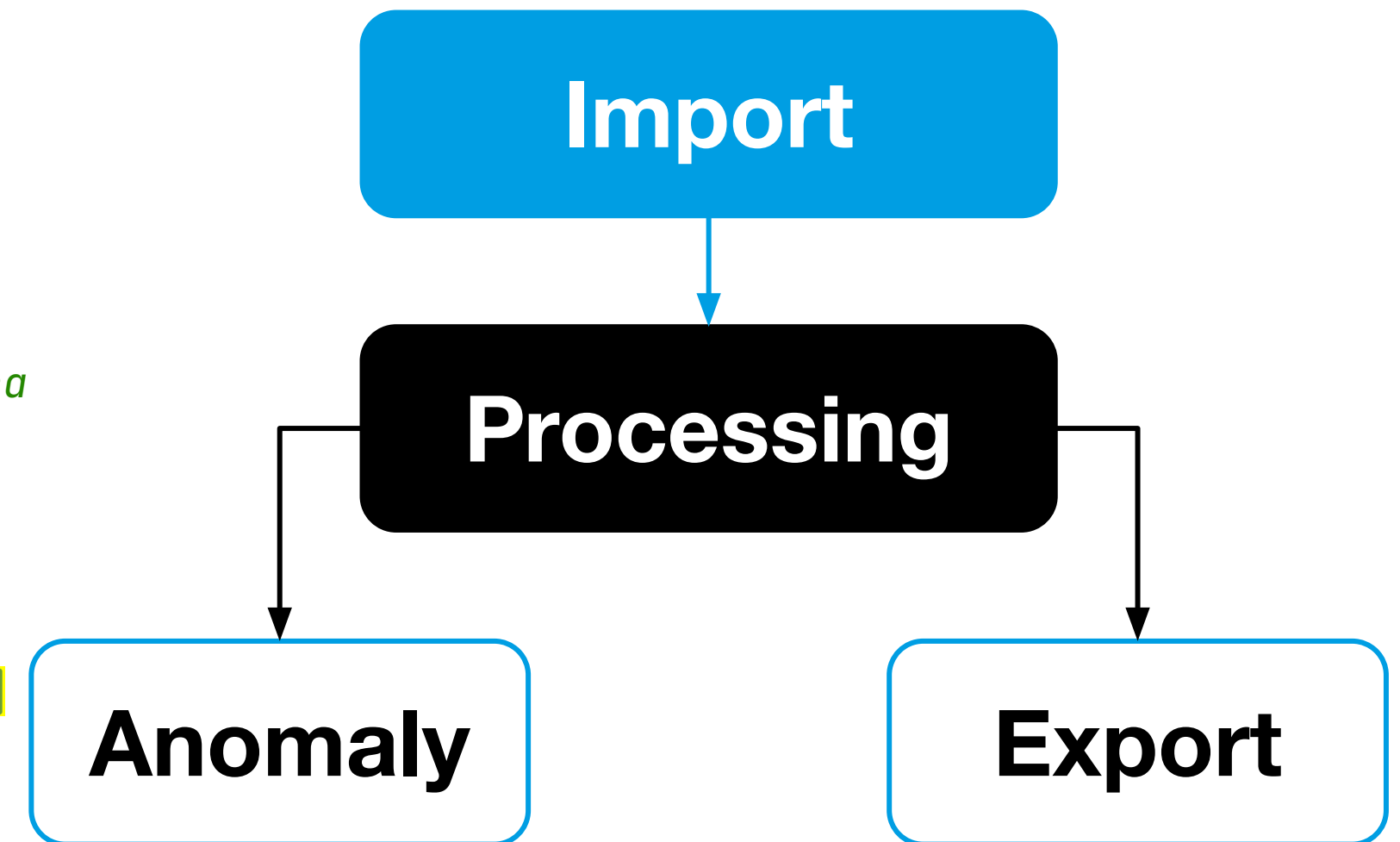
Baggage

```
var source = new ActivitySource("asd");

using var parent = source.StartActivity("ParentOperation");
parent?.AddBaggage("userId", "12345");

// Дочерняя Activity – например, при вызове другого метода/сервиса
using var child = source.StartActivity("ChildOperation");
// child наследует Current как родителя автоматически

Console.WriteLine(child?.GetBaggageItem("userId")); // "12345" ✓
```



Что выбрать?

Activity Tags:

- Детали конкретной операции
 - http.method
 - http.status_code
 - http.url
- Бизнес-метрика конкретной операции
 - order.total_amount
 - order.items_count
 - payment.method

Baggage:

- Сквозная идентификация
 - userId
 - tenantId
- Feature flags
 - experiment.checkout_flow
 - feature.new_pricing_engine

W3C Baggage

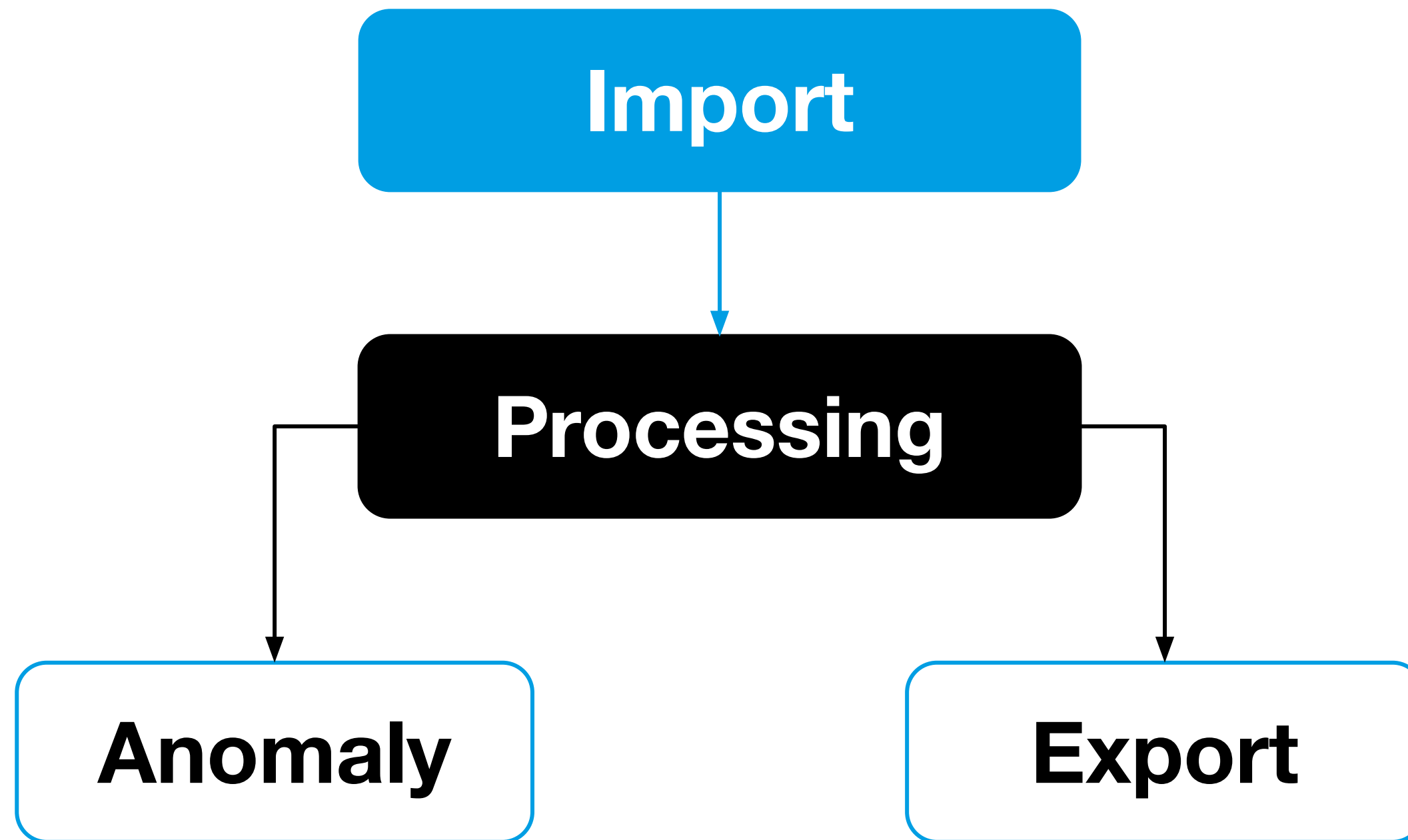
baggage: `userId=123, region=us-west`



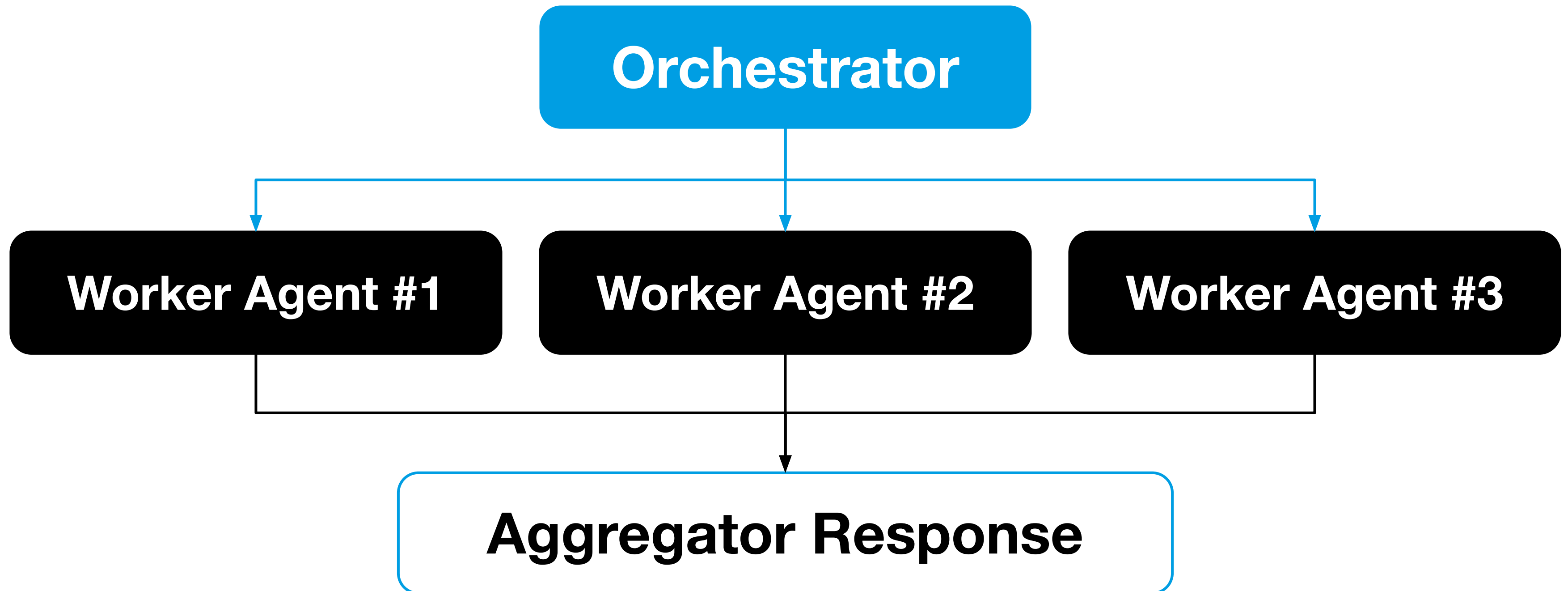
У нас сложный процесс



Упрощенная схема

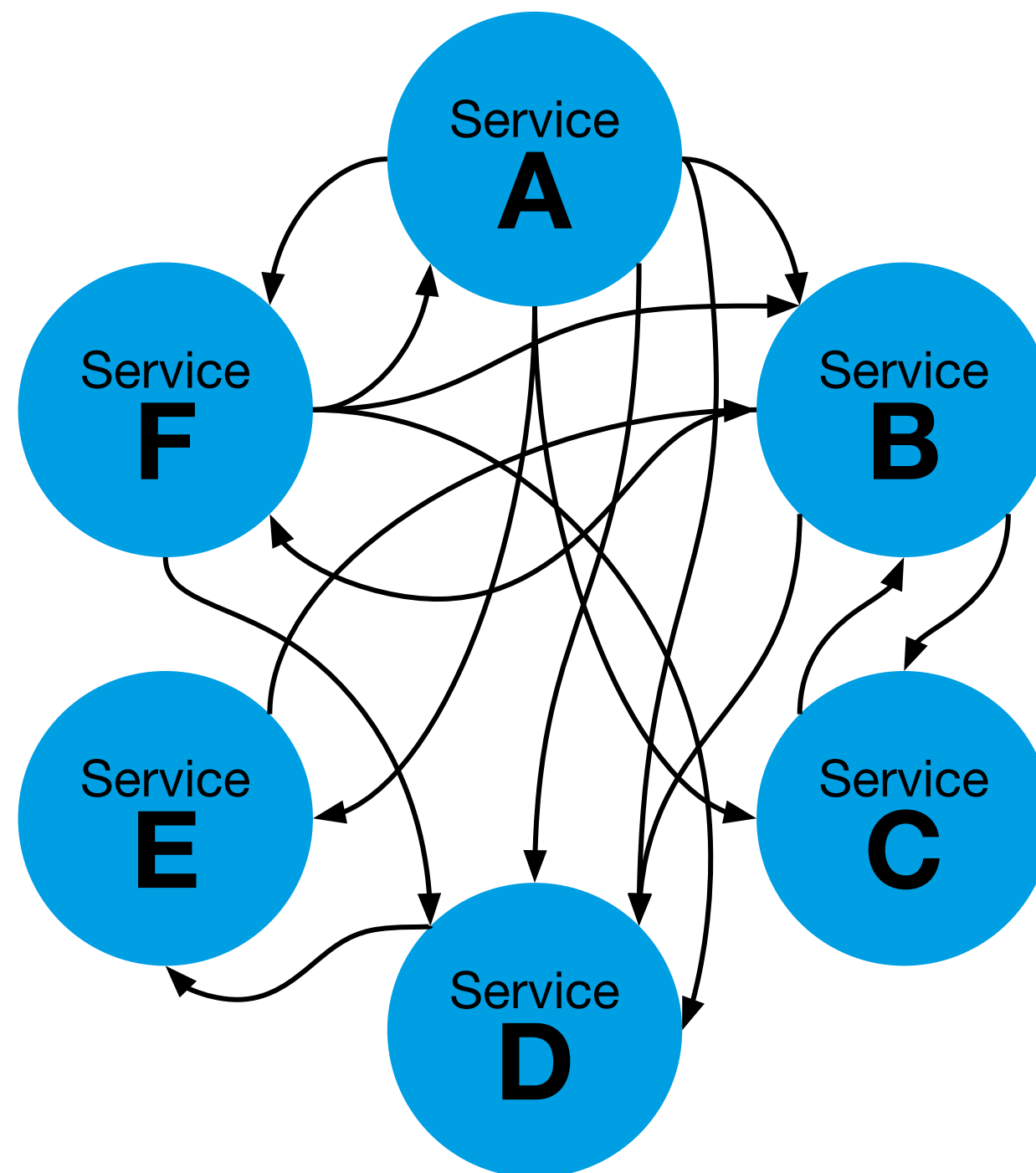


1 parent не прокатит



Кошмар трейсинга

Choreography



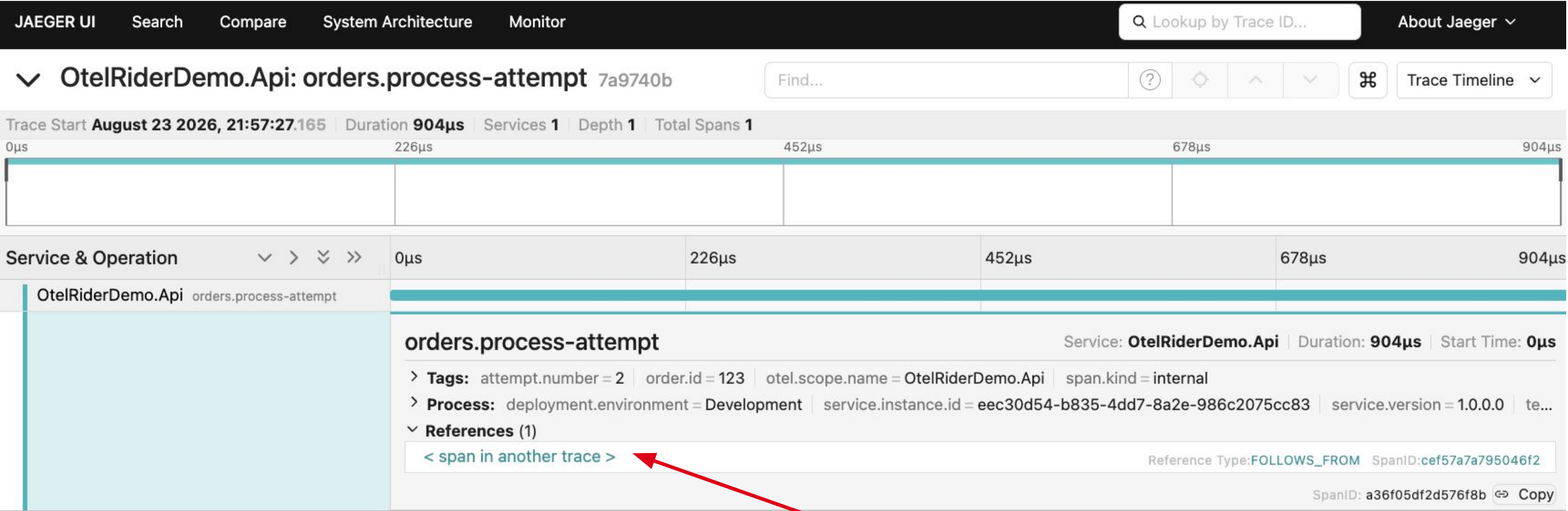
Корреляция между разными Trace

```
var link = new ActivityLink(someOtherContext);  
using var linked = activitySource.StartActivity(  
    "LinkedOperation",  
    ActivityKind.Internal,  
    parentContext: default,  
    links: new[] { link });
```

Почти никто не
поддерживает



Jaeger moret



Sampling



Давайте трейсить вообще всё!

- Google — \$0.20/million spans
- Azure — \$2.30–\$2.76 per GB
- Amazon — \$0.50 per 1 million traces



Готовые стратегии Sampling

- `AlwaysOnSampler` — всегда записывать
- `AlwaysOffSampler` — никогда не записывать
- `TraceIdRatioBasedSampler(double ratio)`
- `ParentBasedSampler(Sampler root)`

```
builder.Services.AddOpenTelemetry()  
    .WithTracing(builder =>  
    {  
        builder  
            .AddSource("ComponentName")  
            .SetSampler(new AlwaysOnSampler())  
            .AddOtlpExporter()  
    });
```

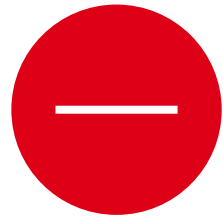
Head-Based Sampling

- TraceParent
 - *version-traceid-spanid-flags*
- Example
 - *00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01*

↑

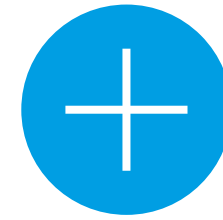
sampled flag (1 = да, 0 = нет)

Head-Based Sampling



Недостатки

- Много точек конфигурации
- Ограниченная гибкость



Преимущества

- Дёшево
- Низкий overhead
- Политики на C#

Tail-Based Sampling

```
# otel-collector-config.yaml
```

```
processors:
```

```
  tail_sampling:
```

```
    decision_wait: 10s
```

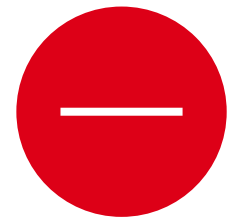
```
    policies:
```

```
      - name: errors-policy
```

```
        type: status_code
```

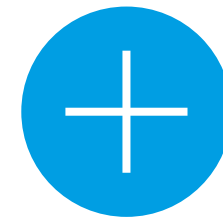
```
        status_code: { status_codes: [ ERROR ] }
```

Tail-Based Sampling



Недостатки

- Стоимость инфры
- Необходимость роутинга между коллекторами
- Задержка в принятии решения



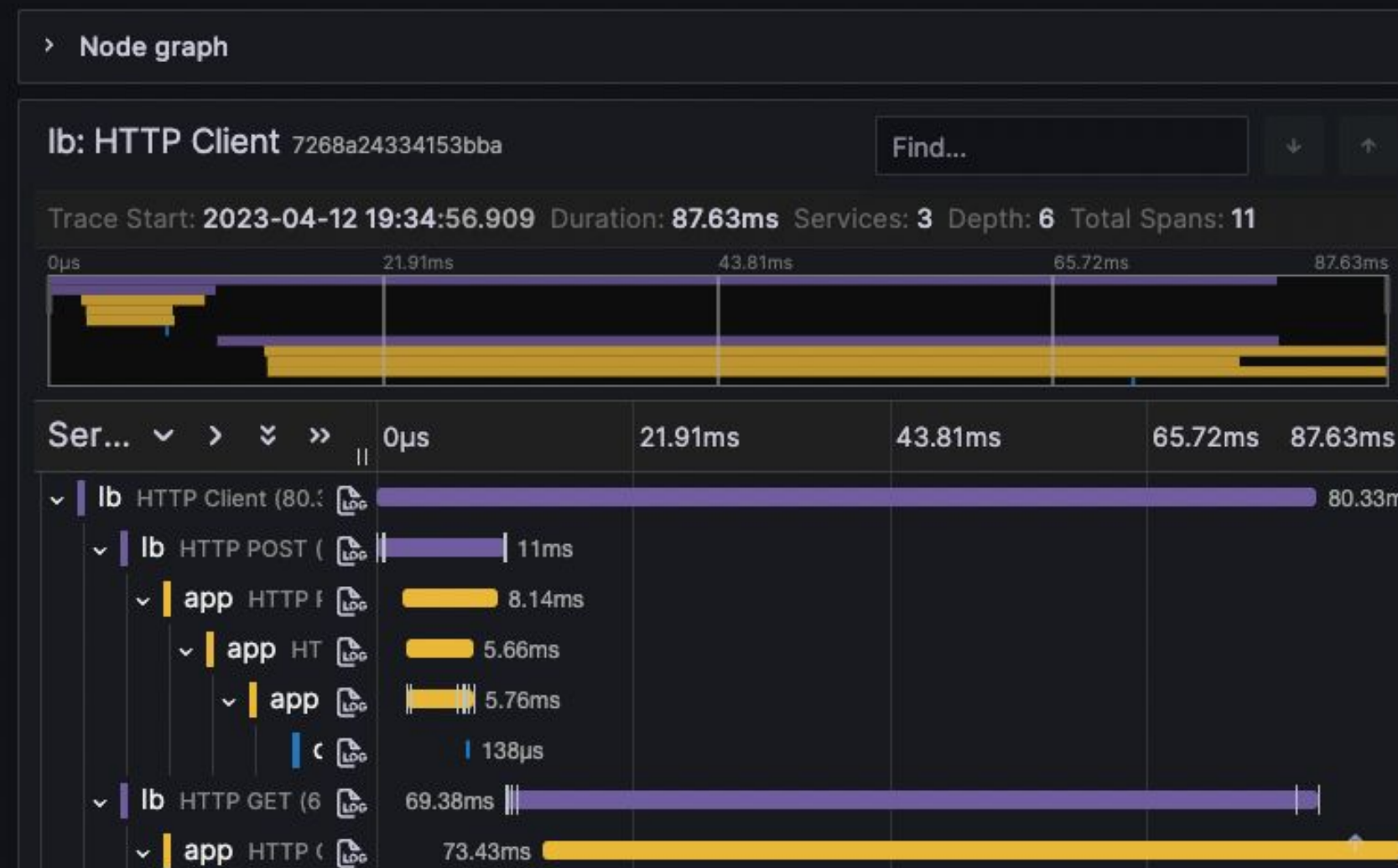
Преимущества

- Гарантированно ловит важное
- Доступ ко всему трейсу

Traces+Metrics



The screenshot shows the Tempo UI's query editor. At the top, there's a header bar with the Tempo logo and a 'Close' button. Below the header, the editor area has a tab labeled 'A' and '(Tempo)'. The 'Query type' dropdown is set to 'TraceQL'. Below this, there's a text input field containing the query '7268a24334153bba'. To the right of the input field is a 'Documentation' link. At the bottom of the editor, there's a section for 'Options' with a 'Limit: 20' setting. The bottom of the image shows three buttons: '+ Add query', 'Query history', and 'Inspector'.



Exemplars

```
var meterProvider = Sdk.CreateMeterProviderBuilder()  
    .SetResourceBuilder(resource)  
    .AddMeter("MyCompany.MyProduct.MyLibrary")  
    .SetExemplarFilter(ExemplarFilterType.TraceBased)  
    .AddOtlpExporter()  
    .Build();
```

Exemplars

```
var meter = new Meter("MyApp.Metrics");  
var histogram = meter.CreateHistogram<double>("http.request.duration");  
  
app.Use(async (context, next) =>  
{  
    using var activity = activitySource.StartActivity("HandleRequest");  
    var sw = Stopwatch.StartNew();  
  
    await next();  
  
    histogram.Record(sw.Elapsed.TotalSeconds);  
});
```

Не все
поддерживают



Чувствительные данные



Выводы

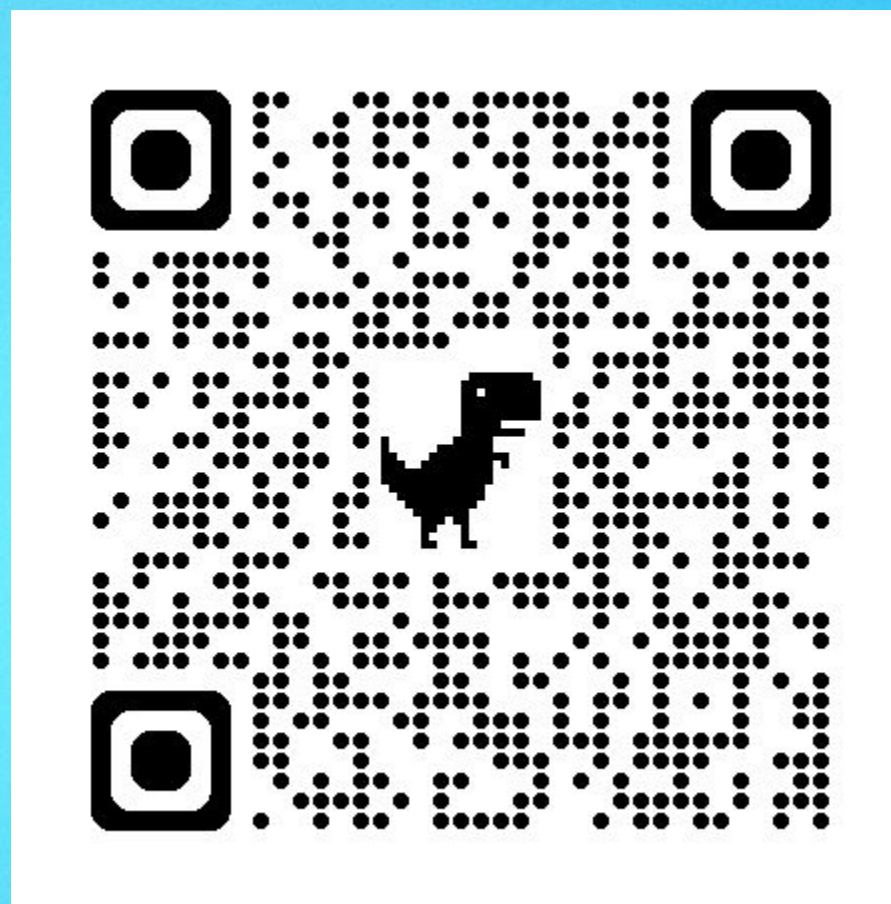
- Современные версии .NET прекрасно интегрируются с Otel
- Используйте готовые инструментации везде, где это возможно
- При необходимости можно легко дополнить свой код трейсами
- Если нет доступа к коду, то можно использовать Auto-Instrumentation
- Span Links бесполезны в реальной среде
- Используйте Sampling если хотите сэкономить

Что еще изучить?

- OpenTelemetry для самых маленьких
(<https://www.youtube.com/watch?v=X3faF3xw3m8>)
- OpenTelemetry Auto-Instrumentation
(<https://opentelemetry.io/docs/zero-code/dotnet>)
- .NET distributed tracing
(<https://learn.microsoft.com/en-us/dotnet/core/diagnostics/distributed-tracing>)



Спасибо!



@nikolay_savenko