



OBSERVABILITY СЕРВИСОВ

Унификация телеметрии для
логов, метрик и трассировок
Начын Д.

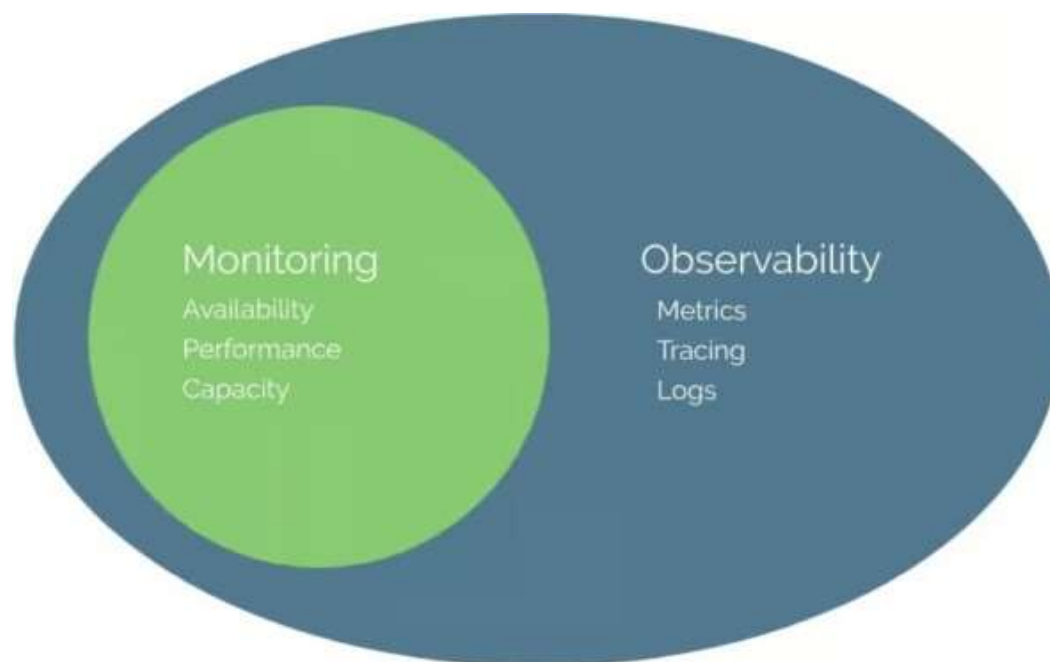
OBSERVABILITY

Observability — это способность системы предоставлять подробную информацию о своём внутреннем состоянии на основе внешних сигналов, таких как логи, метрики и трассировки.

В отличие от стандартного мониторинга, который лишь фиксирует заранее определённые показатели, observability помогает обнаруживать скрытые проблемы, которые сложно предугадать заранее.

«Observability. Новый взгляд на качество ПО» - Лаборатория Качества

МОНИТОРИНГ VS. OBSERVABILITY



МОНИТОРИНГ VS. OBSERVABILITY

Мониторинг:

- *Фокус* — Отслеживание заранее заданных метрик и пороговых значений
- *Подход* — Реактивный — сигнализирует о проблемах, когда они уже произошли
- *Пример из практики ЛК* — Когда мы замечали, что процент ошибок превышает определённый уровень, система посылала уведомления. Но зачастую этого было недостаточно для понимания, что именно пошло не так

МОНИТОРИНГ VS. OBSERVABILITY

Observability:

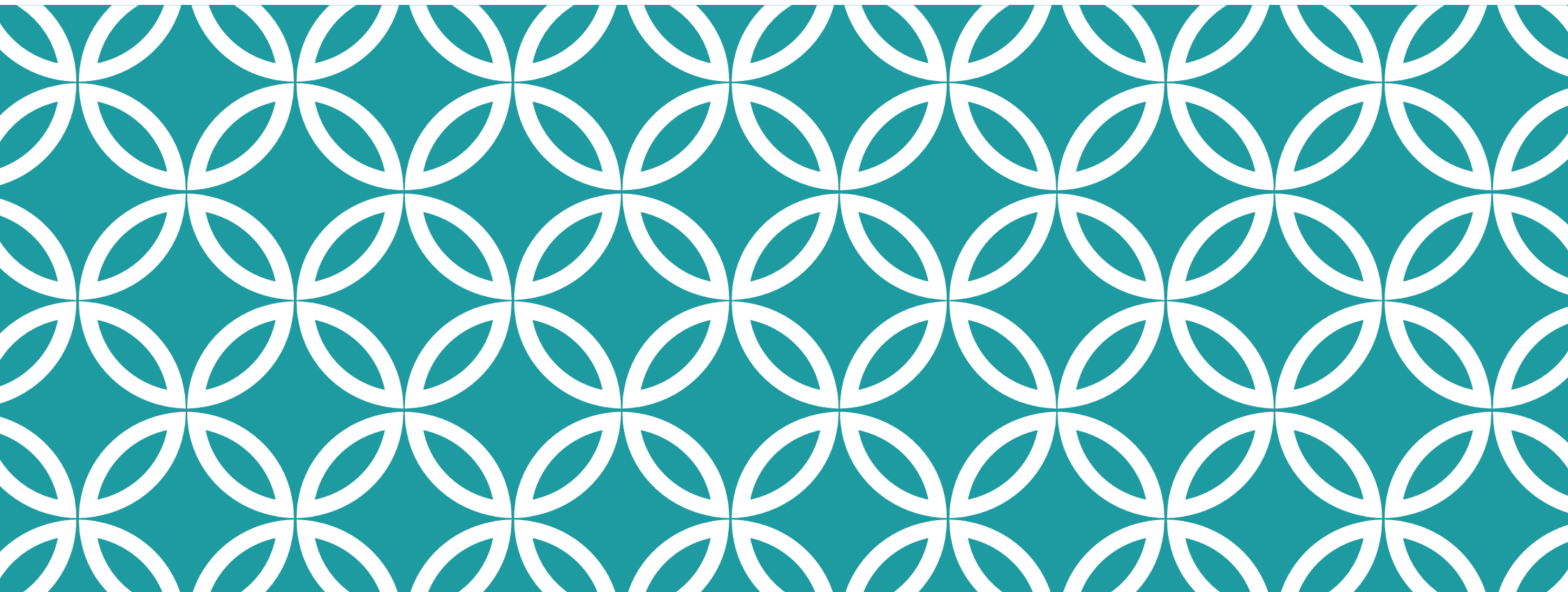
- *Фокус* — Глубокий анализ системы через сбор логов, метрик и трассировок, позволяющий понять, почему произошёл сбой
- *Подход* — Проактивный — даёт возможность обнаруживать «неизвестные неизвестности» и анализировать сложные взаимосвязи
- *Пример из практики ЛК* — В одном из проектов у нас была проблема с задержками в микросервисной архитектуре. Благодаря интеграции трассировок мы смогли увидеть полный жизненный цикл HTTP-запроса и обнаружить узкое место, которое ранее не попадало в поле зрения традиционных инструментов мониторинга

ПРЕИМУЩЕСТВА OBSERVABILITY

- ✓ **Быстрая диагностика.** Наблюдаемость позволяет моментально определить корень проблемы
- ✓ **Повышение качества продукта.** Глубокое понимание работы системы помогает обнаруживать даже скрытые баги, что приводит к более стабильным релизам
- ✓ **Оптимизация расходов**
- ✓ **Непрерывная обратная связь из Production-среды** позволяет вносить улучшения в режиме реального времени, делая процесс разработки более гибким и адаптивным
- ✓ **Обоснованное принятие решений.** Подробные данные и аналитика дают возможность не только выявлять проблемы, но и оптимизировать архитектуру и процессы, что особенно ценно для руководителей и владельцев бизнеса

ЛОГОВ В ФАЙЛ ДОСТАТОЧНО

- Как часто вы смотрите в Production-логи?
- Текстовые логи в файле линейны, их нельзя отсортировать, сгруппировать
- В большинстве случаев логи с типом *Warning* игнорируются, смотрим в логи когда произошла критическая ошибка
var rule = pipelineData.Find(x => ...) ?? new EmptyRule() // Fix Null Ref Exception.
- На сколько вам удобно просматривать логи. *Dozzle* или «*docker cp my_container:/var/log/app.log ./app.log*» ?
- На сколько быстро находите причину бага или требовались ли делать дампы production БД, чтоб воспроизвести баг?



ЛОГИ

Logs
Metrics
Traces

ЛОГИ

Журналы (логи) — это события, которые записывает работающее программное обеспечение. Например:

- Регистрируют, когда пользователь обращается к системе
- Регистрируют критическую ошибку
- Регистрируют трассировку стека ошибки времени выполнения
- Иногда используют для дебага кода =) и забывают удалить из GIT
`console.log("<3")`
`console.debug("AAAAAAAAAA, не работает!!!!!!!!!!!!!!")`

СТРУКТУРА ЛОГОВ

Большинство разработчиков стараются придерживаться общих правил написания логов. К примеру, практически любой лог имеет:

- системную информацию (время и дата события, ID события и другая служебная информация);
- уровень лога;
- текст сообщения (например, сообщения об ошибке);
- контекст (дополнительная информация).

ОБЫЧНЫЕ ЛОГИ

{**Timestamp**:yyyy-MM-dd HH:mm:ss.fff zzz} | {**Level**:u3} | {**SourceContext**} | {**Message**:lj}{NewLine}{**Exception**}

2025-05-21 23:47:07.821 +07:00 | INF | Project1.ConnectionBackgroundService | Start Main connection service...

[23:51:30 INF] Start processing HTTP request POST http://localhost:5000/api/warehouse?*

[23:51:30 INF] Sending HTTP request POST http://localhost:5000/api/warehouse?*

[23:51:30 INF] Received HTTP response headers after 2.9172ms - 500

[23:51:30 INF] End processing HTTP request after 6.8081ms - 500

[23:51:30 INF] Executed endpoint 'HTTP: POST /api/orders'

[23:51:30 **ERR**] Error: Response status code does not indicate success: 500 (Internal Server Error).. **TraceId**: 66a57aeb6e4109e2cbede9f8946ad69b System.Net.Http.**HttpRequestException**: Response status code does not indicate success: 500 (Internal Server Error).

СТРУКТУРИРОВАННЫЕ ЛОГИ

Label filters

service_name ▾ = ▾ EShop.OrderService ▾ × +

```
> 2025-05-21 23:51:39.798 {
  "body": "Entity saved Order",
  "traceid": "72de3bb1f80f5dc53851b3e1bc572bff",
  "spanid": "7e407312ecfc9d1d",
  "severity": "Information",
  "attributes": {
    "ConnectionId": "0HNC0GRMH8V2M",
    "RequestId": "0HNC0GRMH8V2M:00000010",
    "RequestPath": "/api/orders",
    "entity": "Order",
    "message_template.text": "Entity saved {entity}"
  },
  "resources": {
    "service.name": "EShop.OrderService",
    "telemetry.sdk.language": "dotnet",
    "telemetry.sdk.name": "serilog",
    "telemetry.sdk.version": "4.1.1+b894851cb04a4203707a1243bf0f66ff6c278be8"
  },
  "instrumentation_scope": {
    "name": "EShop.Database.AppDbContext"
  }
}
```

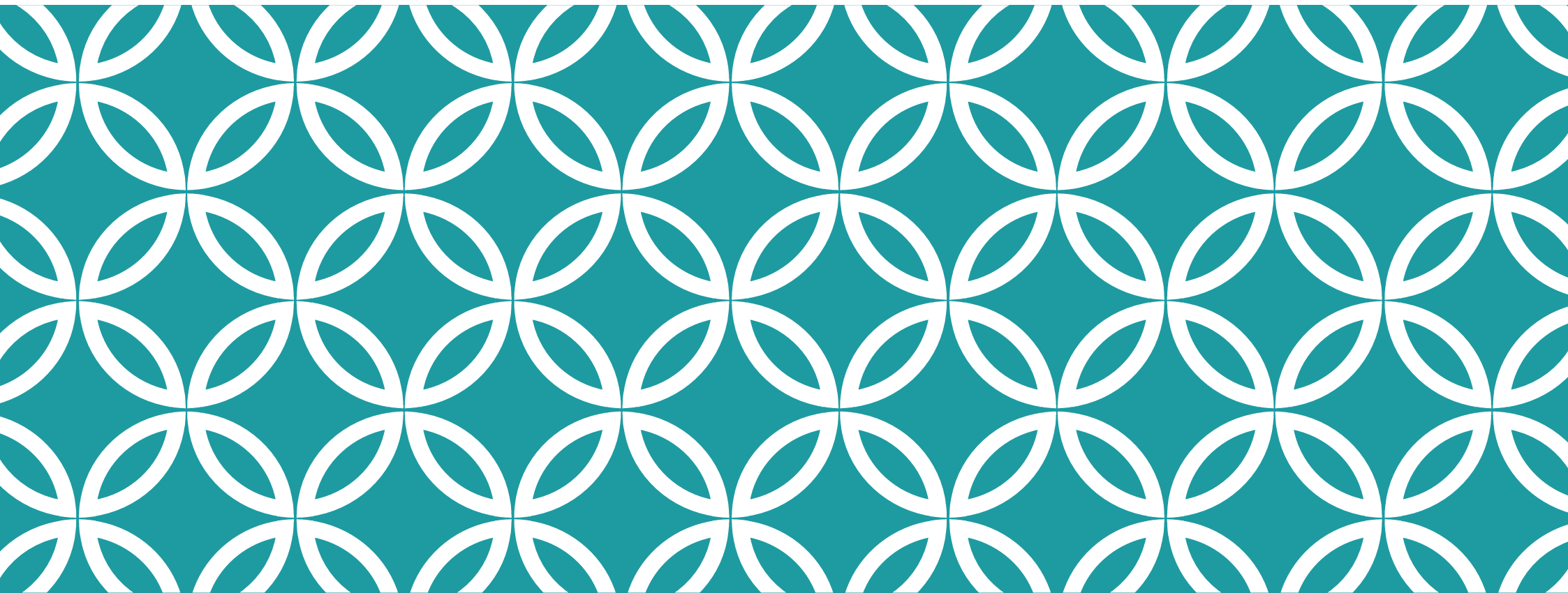
```
    "telemetry.sdk.name": "serilog",
    "telemetry.sdk.version": "4.1.1+b894851cb04a4203707a1243bf0f66ff6c278be8"
  },
  "instrumentation_scope": {
    "name": "EShop.Middlewares.ExceptionHandler"
  }
}
```

Fields

🔍	🔍	🔍	🔍	🔍	EnvNameTest	EShop
🔍	🔍	🔍	🔍	🔍	detected_level	ERROR
🔍	🔍	🔍	🔍	🔍	exporter	OTLP
🔍	🔍	🔍	🔍	🔍	job	EShop.OrderService
🔍	🔍	🔍	🔍	🔍	level	ERROR
🔍	🔍	🔍	🔍	🔍	service_name	EShop.OrderService

СТРУКТУРИРОВАННЫЕ ЛОГИ

```
> 2025-05-21 23:51:31.423 {
  "body": "Error: Response status code does not indicate success: 500 (Internal Server Error).. TraceId: f8e4cdc30e31051c6042c2c91d018cda",
  "traceid": "f8e4cdc30e31051c6042c2c91d018cda",
  "spanid": "537ba86f61b40c43",
  "severity": "Error",
  "attributes": {
    "ConnectionId": "0HNC0GRMH8V2M",
    "Message": "Response status code does not indicate success: 500 (Internal Server Error).",
    "RequestId": "0HNC0GRMH8V2M:0000000A",
    "RequestPath": "/api/orders",
    "TraceId": "f8e4cdc30e31051c6042c2c91d018cda",
    "exception.message": "Response status code does not indicate success: 500 (Internal Server Error).",
    "exception.stacktrace": "System.Net.Http.HttpRequestException: Response status code does not indicate success: 500 (Internal Server Error)
    at EShop.Client.EShopClient.ReserveProduct(Int32 productId, Int32 quantity) in C:\\Users\\nachy\\RiderProjects\\OpenTelemetryExample\\EShop\\EShop\\Clients\\EShopClient.cs:line 17
    at Microsoft.AspNetCore.Mvc.ApplicationParts.ApplicationPartManager.InvokeAsync(InvocationContext context) in C:\\Users\\nachy\\RiderProjects\\OpenTelemetryExample\\EShop\\EShop\\Services\\OrderService.cs:line 17
    at Microsoft.AspNetCore.Routing.EndpointMiddleware.Invoke(HttpContext httpContext, JsonSerializerOptions jsonOptions) in C:\\Users\\nachy\\RiderProjects\\OpenTelemetryExample\\EShop\\EShop\\Middlewares\\ExceptionMiddleware.cs:line 12",
    "exception.type": "System.Net.Http.HttpRequestException",
    "message_template.text": "Error: {Message}. TraceId: {TraceId}"
  },
  "resources": {
    "service.name": "EShop.OrderService",
    "telemetry.sdk.language": "dotnet",
    "telemetry.sdk.name": "serilog",
    "telemetry.sdk.version": "4.1.1+b894851cb04a4203707a1243bf0f66ff6c278be8"
  },
  "instrumentation_scope": {
    "name": "EShop.Middlewares.ExceptionMiddleware"
  }
}
```



МЕТРИКИ

Logs
Metrics
Traces

МЕТРИКИ

Метрики — это количественное измерение какого-то свойства системы в определенный момент ее работы, т.е. метрики отвечают на вопросы «когда?» и «сколько?».

- Сколько ресурсов процессора было использовано за последний час?
- Сколько дискового пространства потребляется?
- Какая пропускная способность была использована?
- Сколько раз ты открыл приложение Wildberries?
- Сколько раз ты положил товар в корзину, но не купил его...

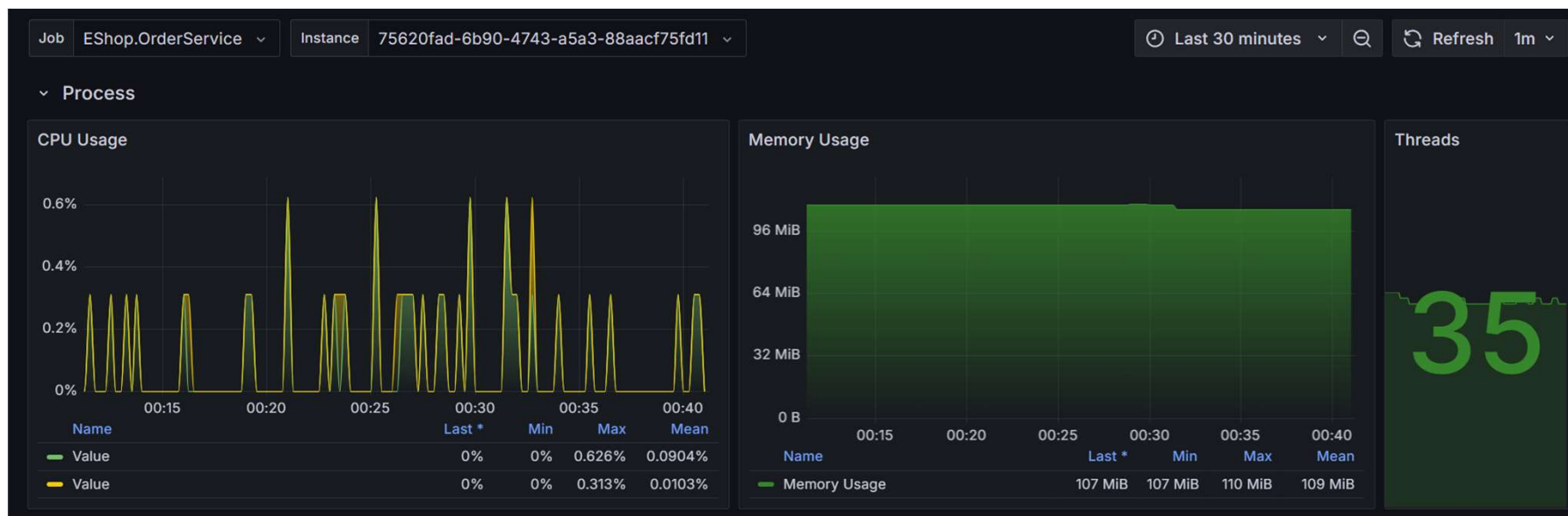
КАТЕГОРИИ МЕТРИК

- **Системные метрики** — показывают количество потребляемых системных ресурсов, например, процессорное время или оперативная память. Такие метрики обычно предоставляются операционной системой.
- **Метрики приложения** — это набор неких стандартных метрик, которые можно собирать с инфраструктурных компонентов вашего приложения, например, сборщик мусора или веб-фреймворка.
- А вот **Бизнес метрики** сами собой не появятся. Они уже зависят от бизнес логики вашего приложения, и только вы сами или бизнес решает, что необходимо считать.

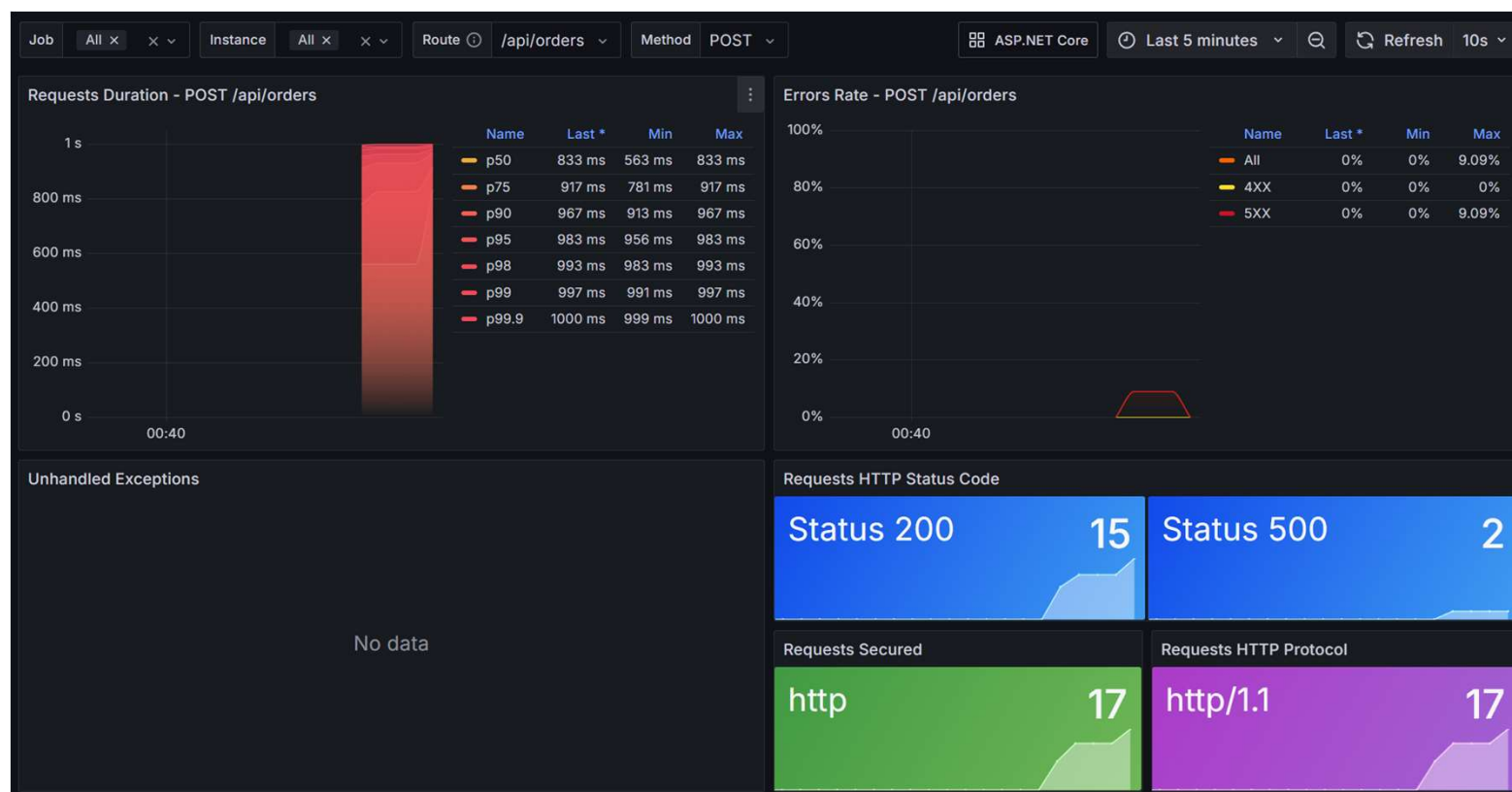
ТИПЫ МЕТРИК

- **Counter** — Подсчет монотонно возрастающих значений. Значение только увеличивается (нельзя уменьшать).
- **UpDownCounter** — Подсчет значений, которые могут увеличиваться и уменьшаться.
- **Gauge (Измеритель)** — Измерение текущего значения в определенный момент времени. (Температура сервера)
- **Histogram (Гистограмма)** — Анализ распределения значений (например, времени ответа).
 - Позволяет вычислять перцентили (P95, P99), среднее, сумму

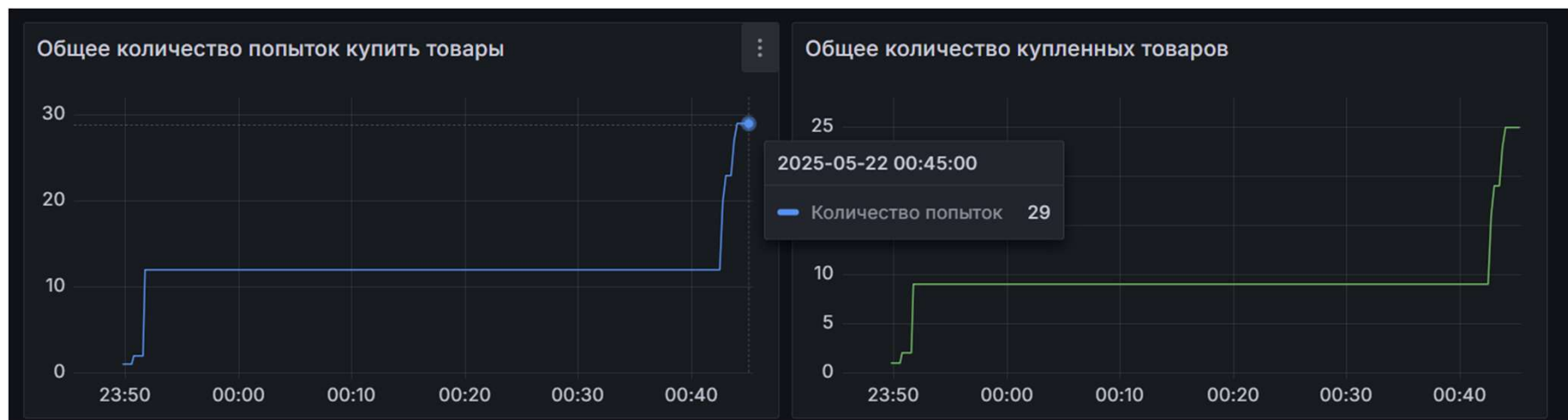
МЕТРИКИ ПРИЛОЖЕНИЯ



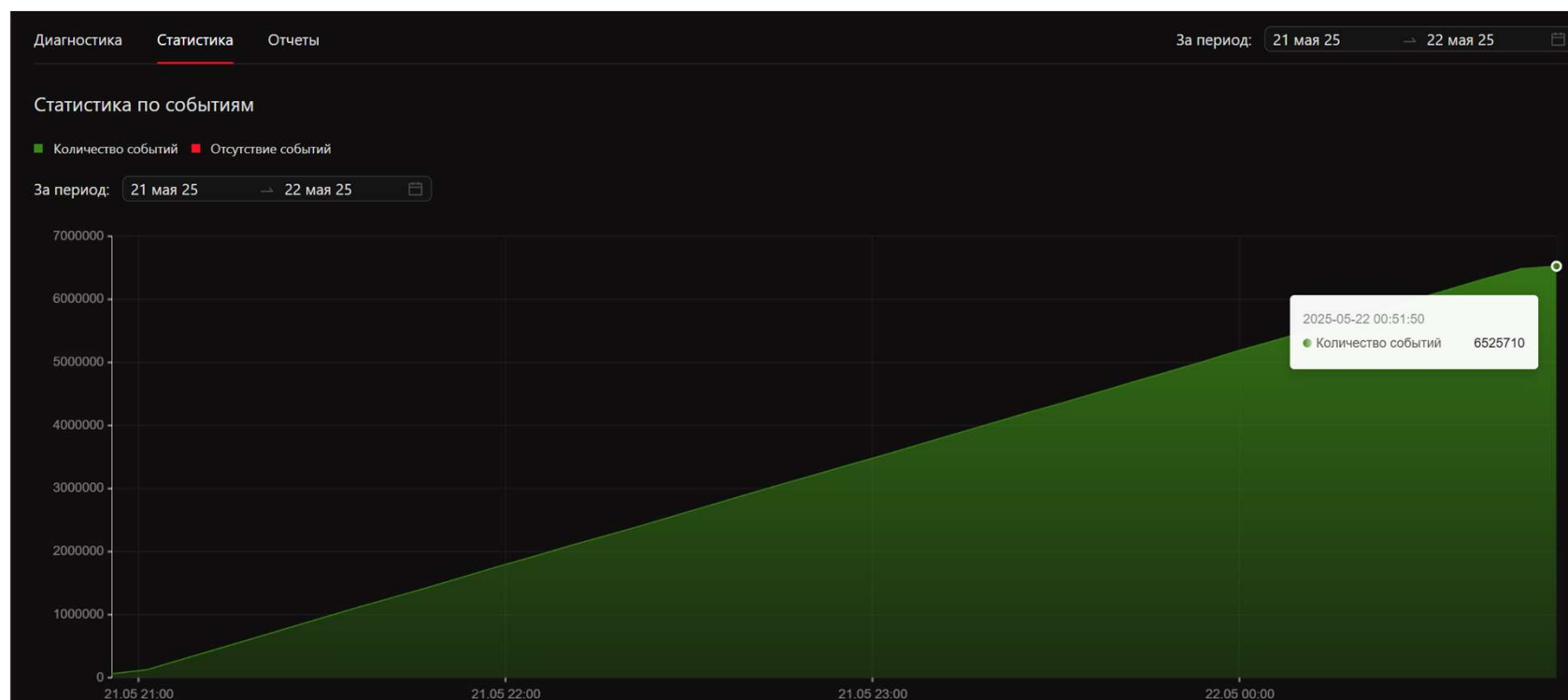
МЕТРИКИ ПРИЛОЖЕНИЯ

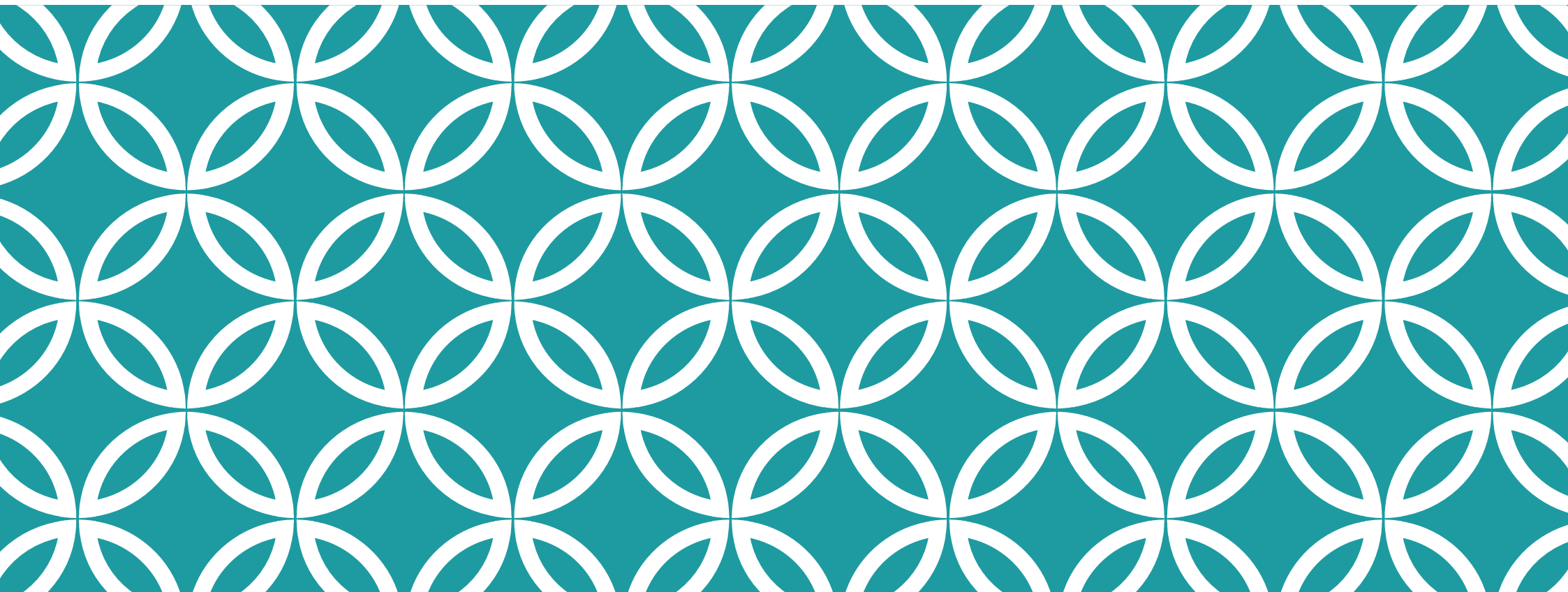


БИЗНЕС МЕТРИКИ



БИЗНЕС МЕТРИКИ





ТРАССИРОВКИ

Logs
Metics
Traces

ТРАССИРОВКИ

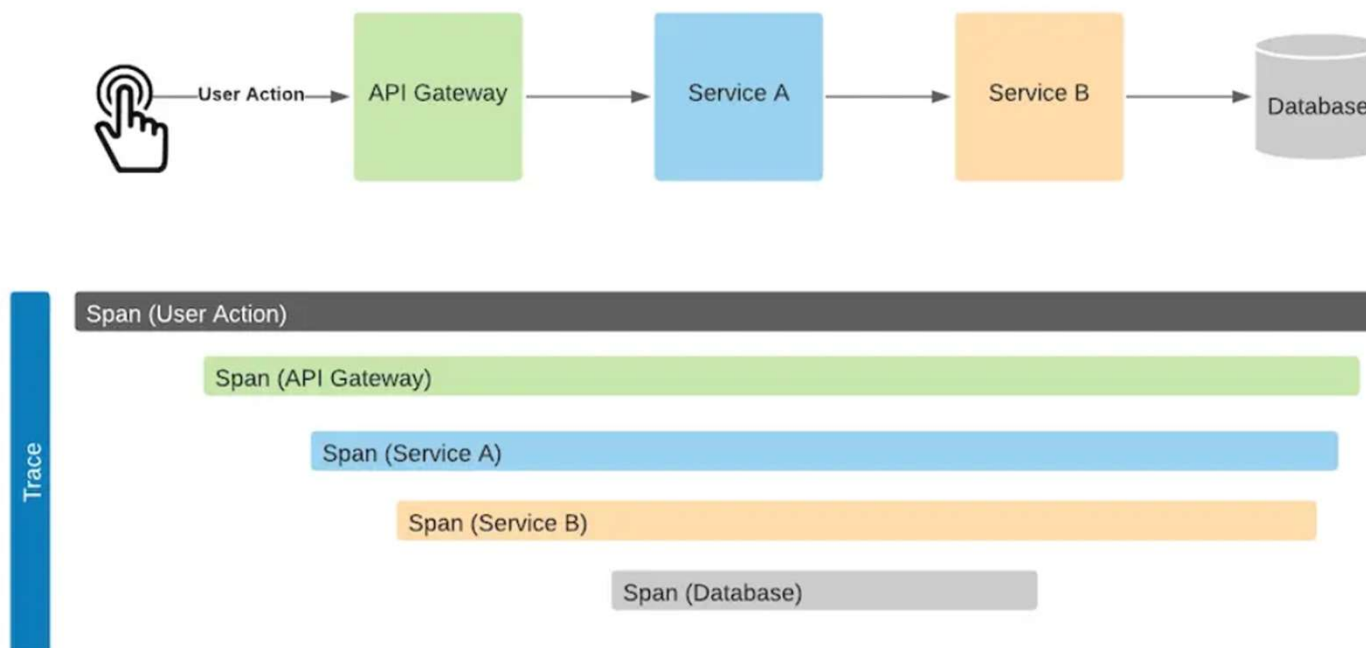
Простым языком - это пошаговое выполнение программы.

Каждый день разработчики пользуются трассировкой во время отладки или профилирования. Но для этого приложение собирается в debug-режиме, и разработчик вмешивается в работу приложения.

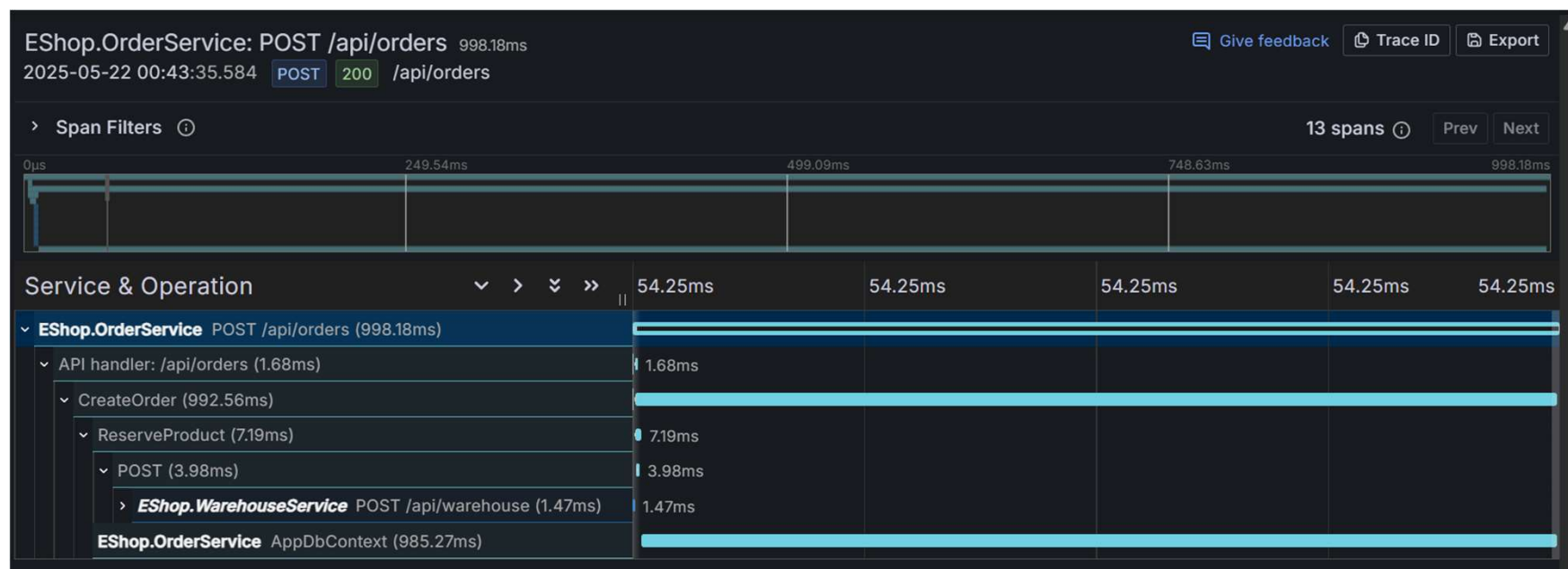
На проде мы так сделать не можем. Для сбора трассировок во время работы приложения их нужно явно создавать, точно так же, как это делается с логами.

КЛЮЧЕВЫЕ СЛОВА ТРАССИРОВКИ

SPAN & TRACE



ТРАССИРОВКИ TRACE + SPAN



ТРАССИРОВКИ. EVENTS & TAGS

reserveProduct

Service: EShop.WarehouseService

Duration: 536.1µs

Start Time: 6.6ms (00:43:35.591)

Kind: internal | Status: unset

Library Name: warehouse-service

> Span attributes: order.product_id = 1 | order.quantity = 3 | product.name = Keyboard | product.pric...

> Resource attributes: EnvNameTest = EShop | host.arch = amd64 | host.id = 8f124cf3-18bf-4dd7-b...

Events (2)

6.94ms (Start reserving prod...)

event name"Start reserving product"

timestamp1747849415591

quantity3

6.95ms (Product reserved)

productNameKeyboard

quantity3

Event timestamps are relative to the start time of the full trace.

ТРАССИРОВКИ

JAEGER UI

Search

Compare

System Architecture

Monitor

Q Lookup by Trace ID...

About Jaeger

←

▼ EShop.OrderService: POST /api/orders dd275b2

Find...

?

×

⌘

Trace Statistics

Trace Start May 22 2025, 00:43:41.359

Duration 824.56ms

Services 2

Depth 8

Total Spans 13

Trace Statistics

Group By: Service Name

No item selected

Color by: Count

Group	Count	Total	Avg	Min	Max	ST Total	ST Avg	ST Min	ST Max	ST in Duration
EShop.OrderService	6	2466.82ms	411.14ms	1.63ms	824.56ms	1641.68ms	273.61ms	0.01ms	822.93ms	199.1%
EShop.WarehouseService	7	2.8ms	0.4ms	0.02ms	1.34ms	2.01ms	0.29ms	0.02ms	1ms	0.24%

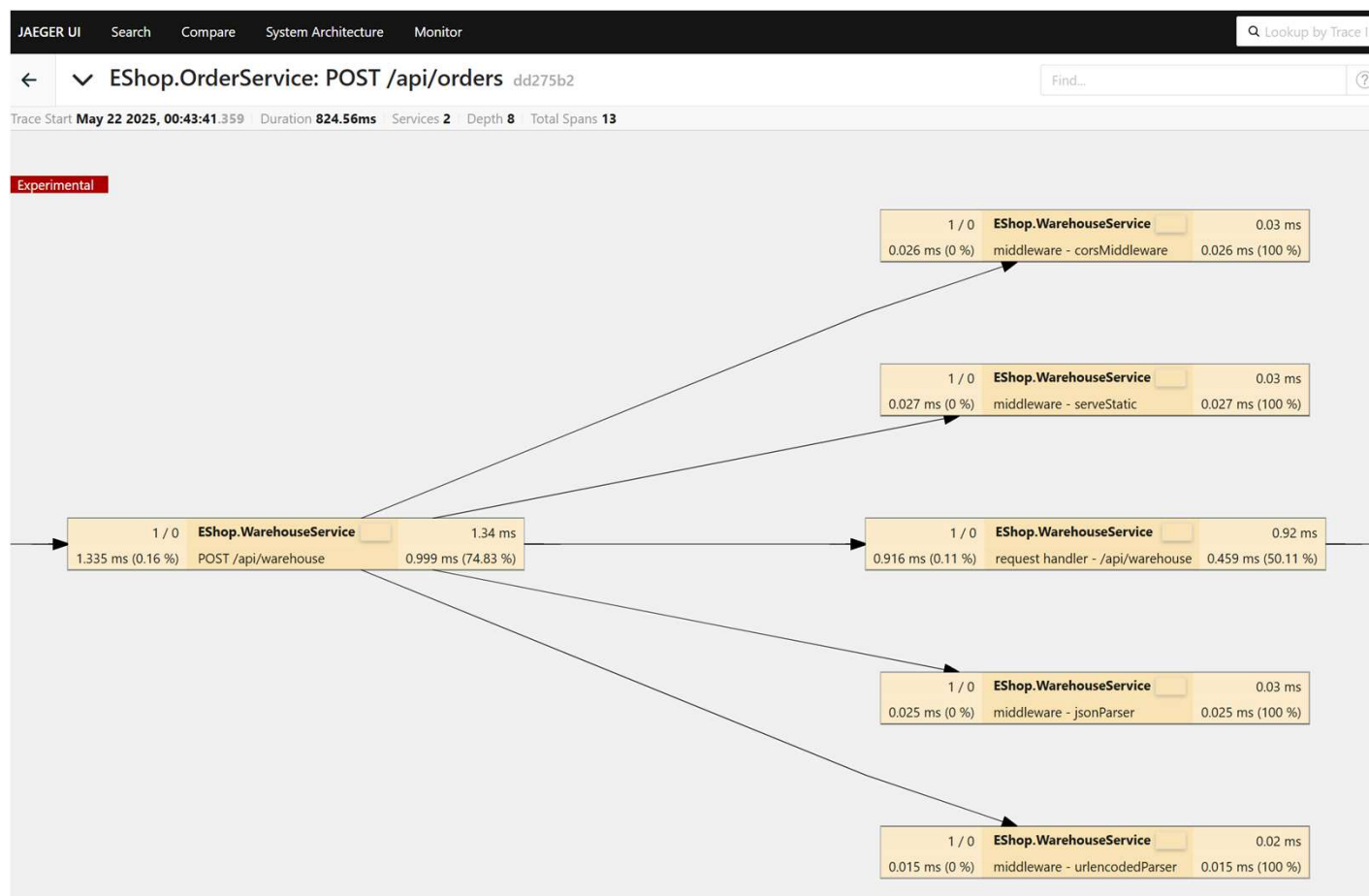
<

1

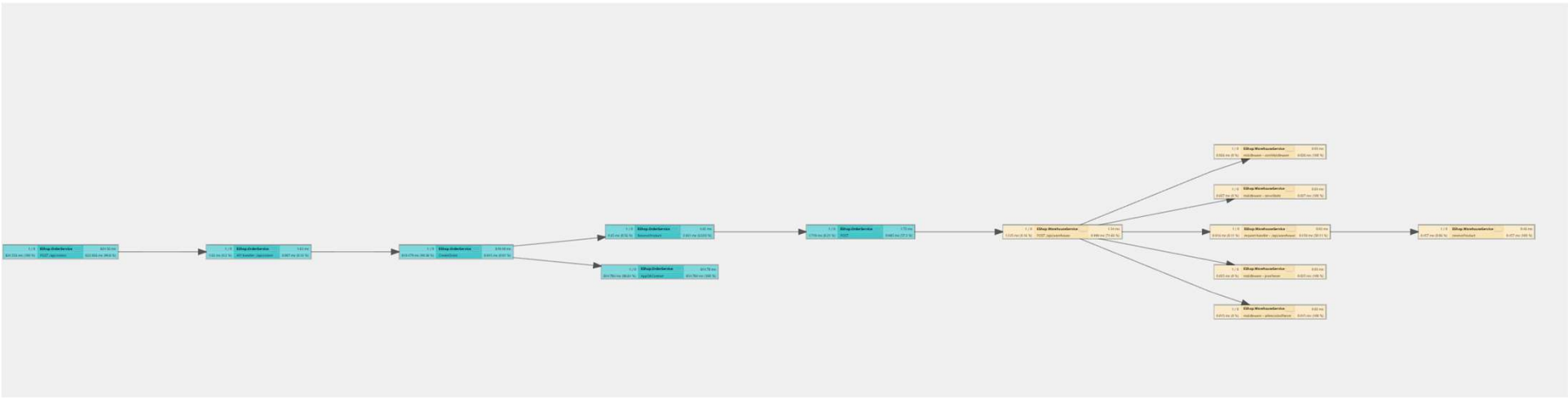
>

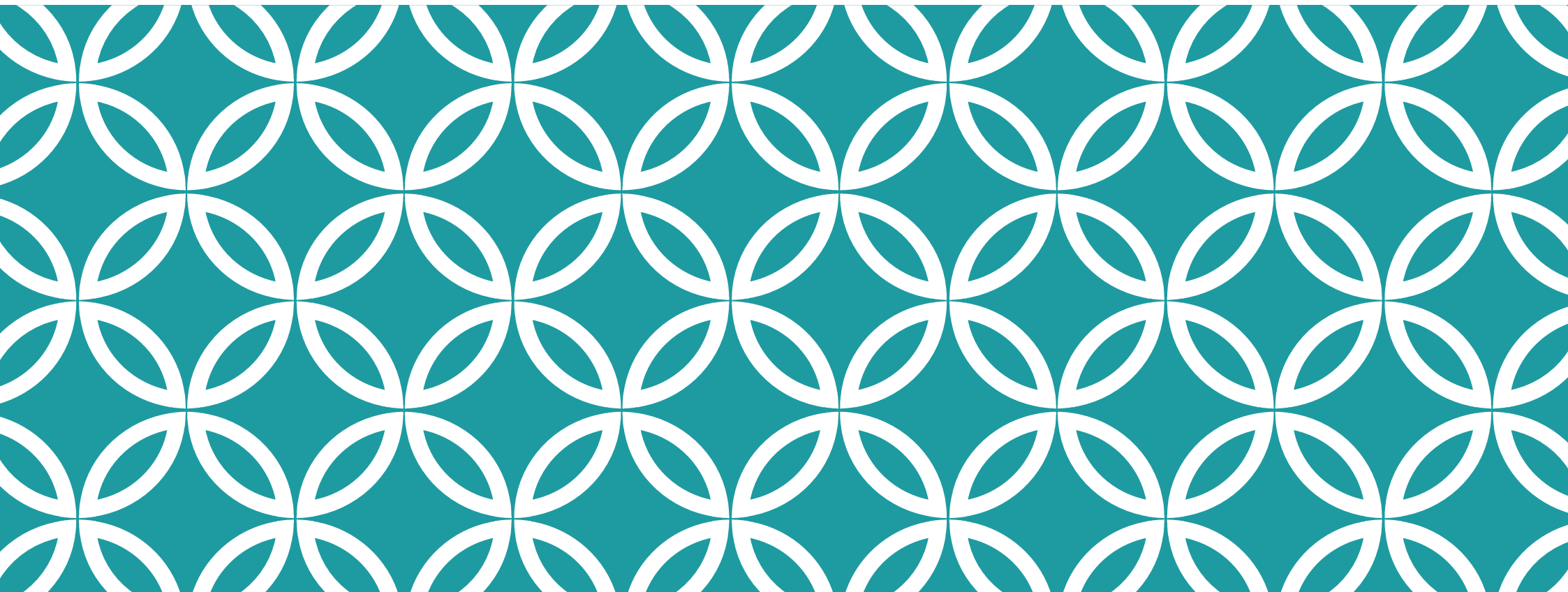
10 / page

ТРАССИРОВКИ



ТРАССИРОВКИ

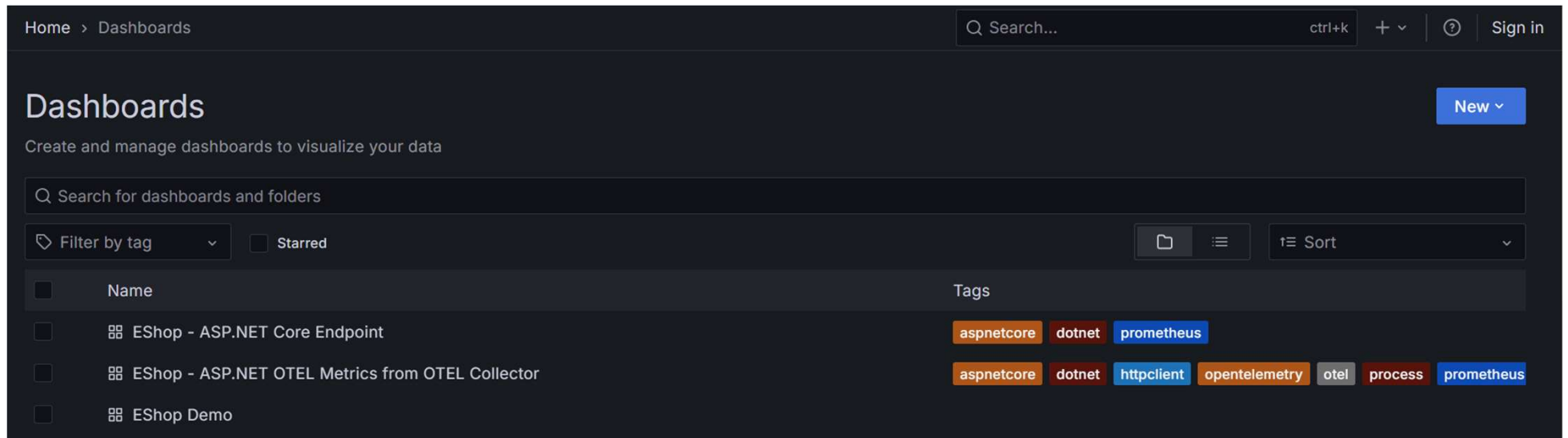




КОМБО MTL

Logs
Metics
Traces

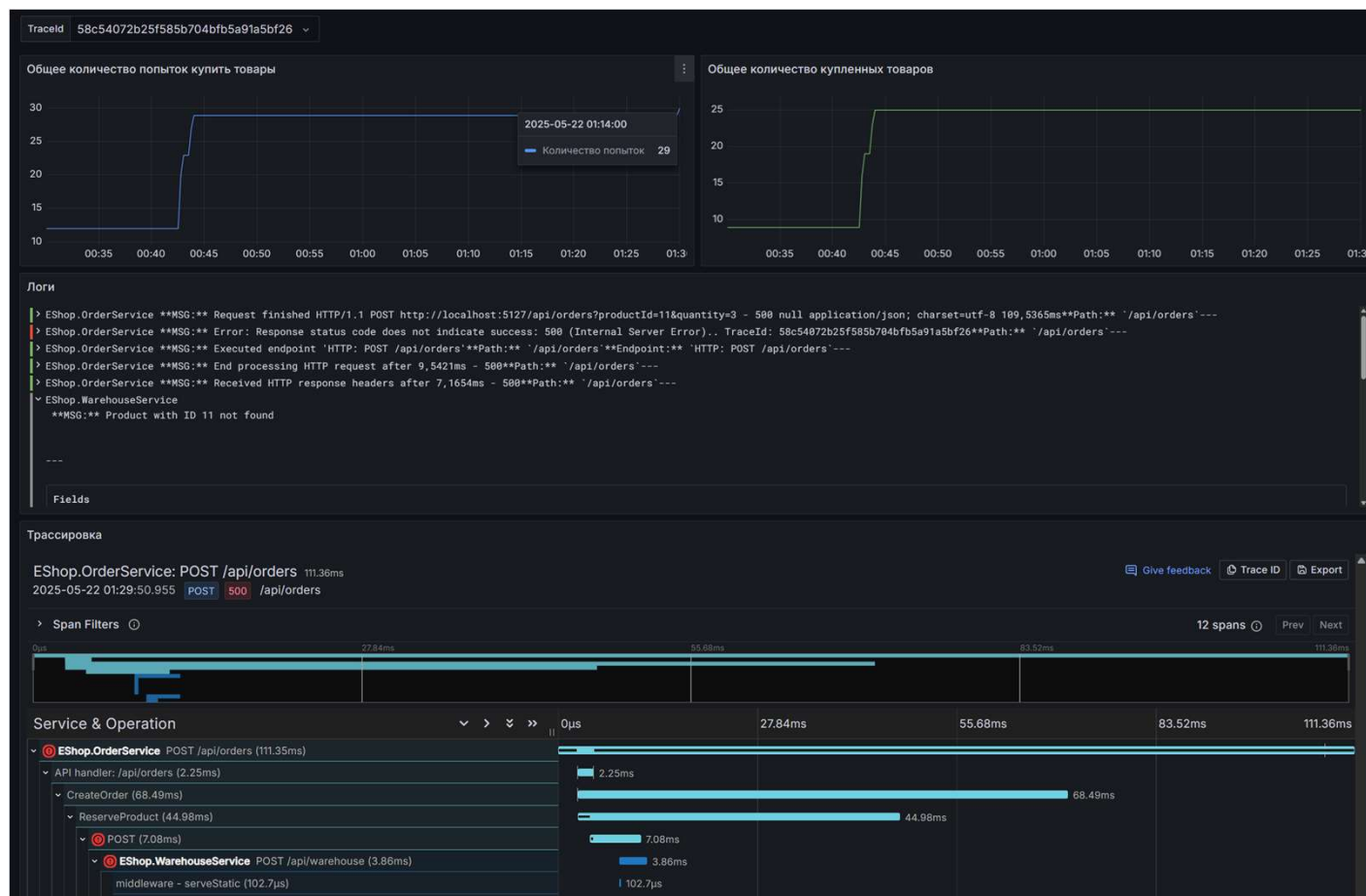
METICS + TRACES + LOGS = DASHBOARD



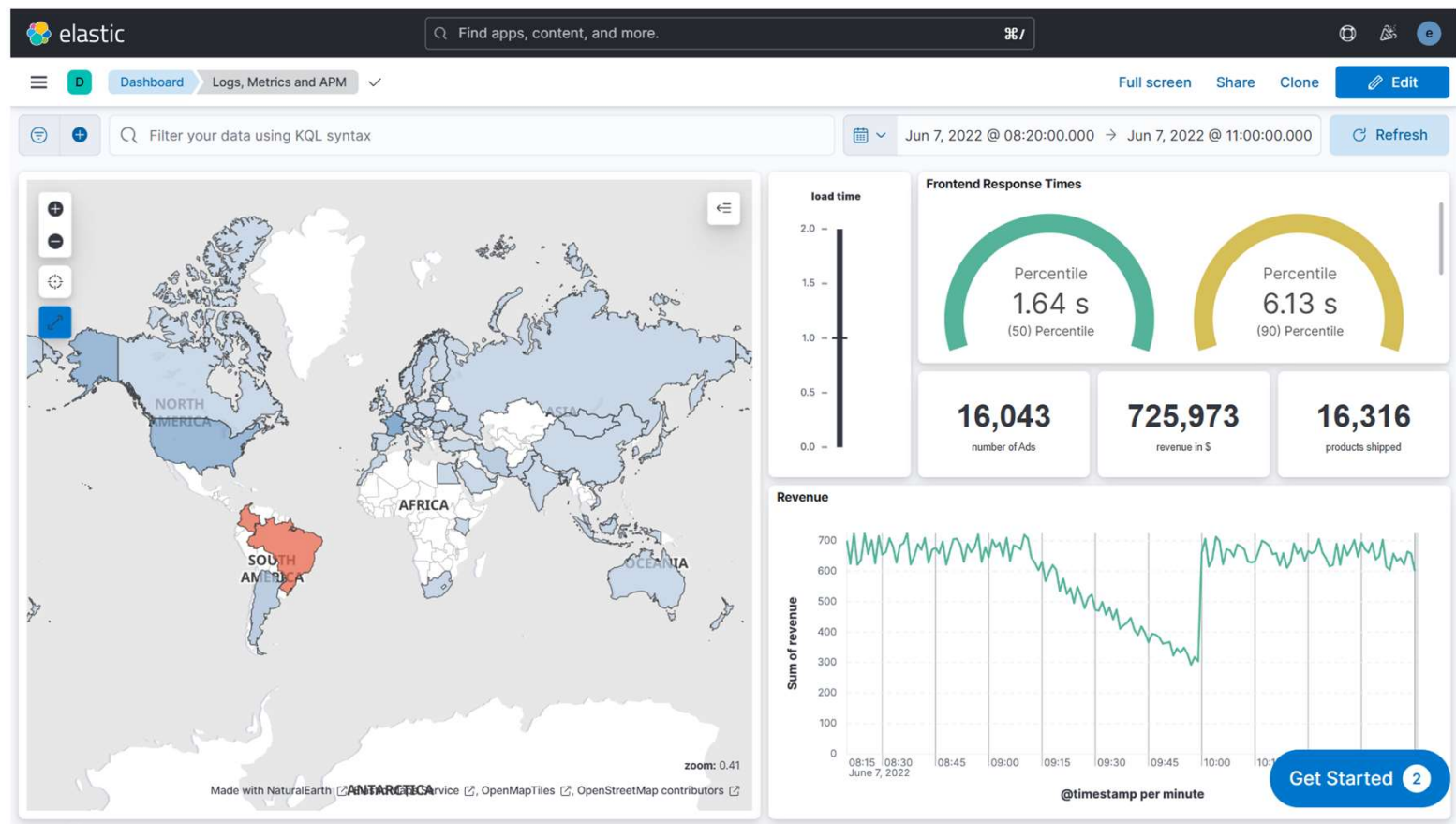
The screenshot shows the Grafana Dashboards interface. At the top, there's a navigation bar with 'Home > Dashboards', a search bar, and user controls. Below this, the 'Dashboards' section has a 'New' button and a description: 'Create and manage dashboards to visualize your data'. A search bar for dashboards and folders is present. Below the search bar, there are filters for 'Filter by tag' and 'Starred'. A table lists three dashboards with their names and associated tags.

Name	Tags
EShop - ASP.NET Core Endpoint	aspnetcore dotnet prometheus
EShop - ASP.NET OTEL Metrics from OTEL Collector	aspnetcore dotnet httpclient opentelemetry otel process prometheus
EShop Demo	

METRICS + TRACES + LOGS = DASHBOARD



MTL → АНАЛИЗ ДАННЫХ



ДЛЯ КОГО OBSERVABILITY

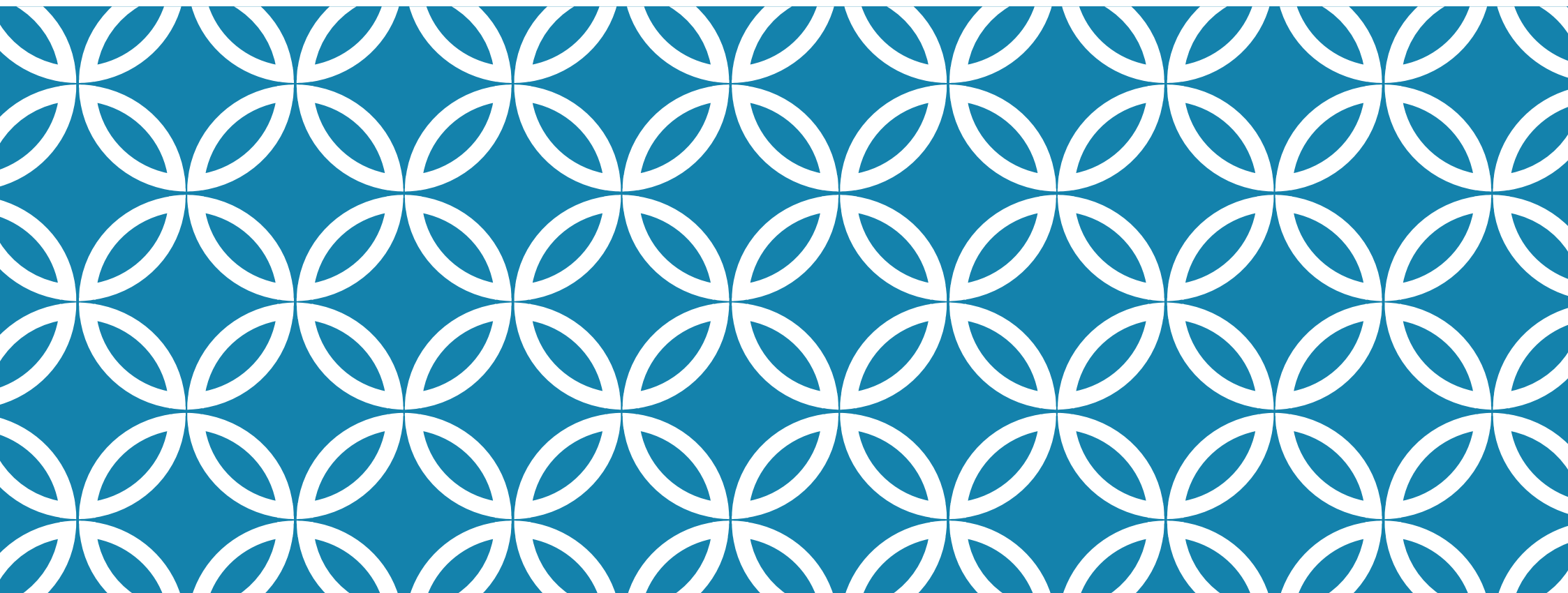
Коротко → для всех (контроль процесса, визуал)

Разработчики → realtime отладка production и т. д

Тестировщики → повышение качества продукта (в bug тикетах присылают `traceld`, могут изучить `Trace` и самостоятельно определить проблему, построение графиков качества системы)

Бизнес → оптимизация расходов \$\$\$, обоснованное принятие решений, похвастаться перед заказчиками

Бизнес метрики → аналитики, маркетологи, финансисты, менеджеры, BI — все, кому нужна понятная аналитика



ВНЕДРЕНИЕ OBSERVABILITY

Унификация телеметрии для
логов, метрик и трассировок

ЗООПАРК ИНСТРУМЕНТОВ

Инструменты для логирования:

- Microsoft App Insights & Azure Monitor
- ELK Stack (Elasticsearch + Logstash + Kibana)
- Grafana Loki
- Fluentd / Fluent Bit
- Graylog
- SigNoz
- Datadog / New Relic
- ClickHouse ...

ЗООПАРК ИНСТРУМЕНТОВ

Инструменты для метрик:

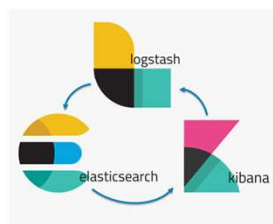
- Prometheus
- VictoriaMetrics
- Thanos / Cortex
- Graphite
- InfluxDB
- Microsoft App Insights & Azure Monitor
- ELK Stack (Elasticsearch + Logstash + Kibana)...

ЗООПАРК ИНСТРУМЕНТОВ

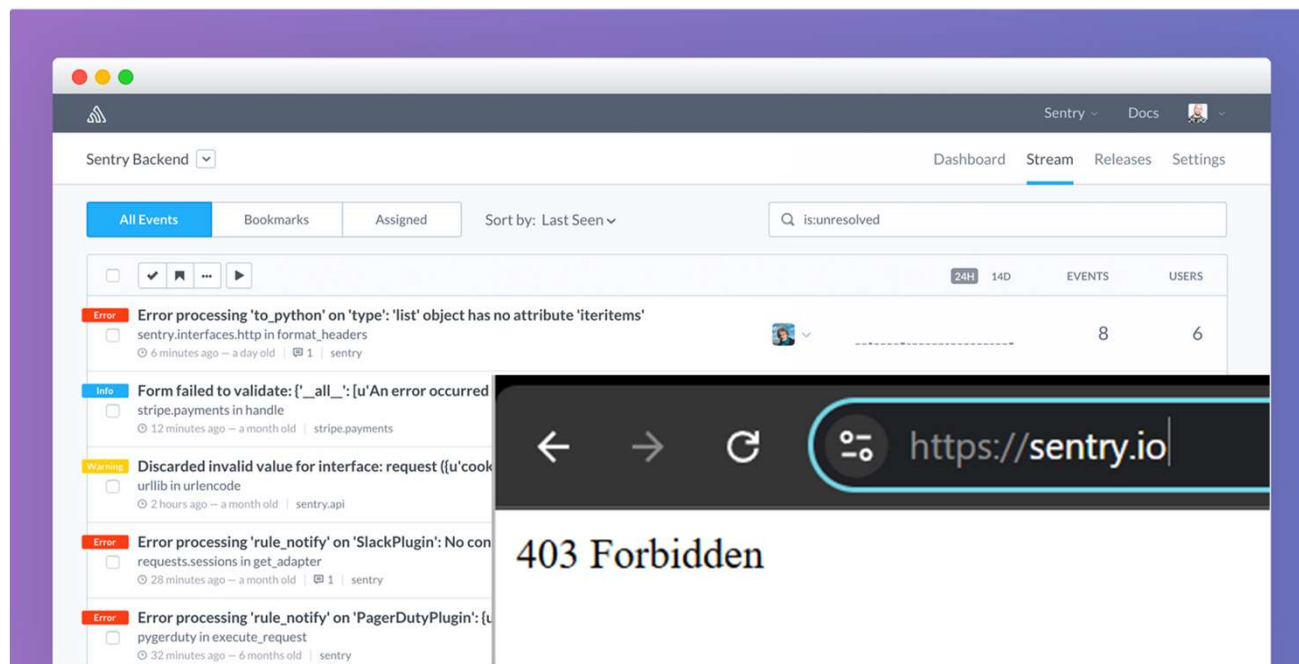
Инструменты для трассировок:

- Jaeger
- Zipkin
- SkyWalking
- AWS X-Ray
- Microsoft App Insights & Azure Monitor
- Grafana Tempo...

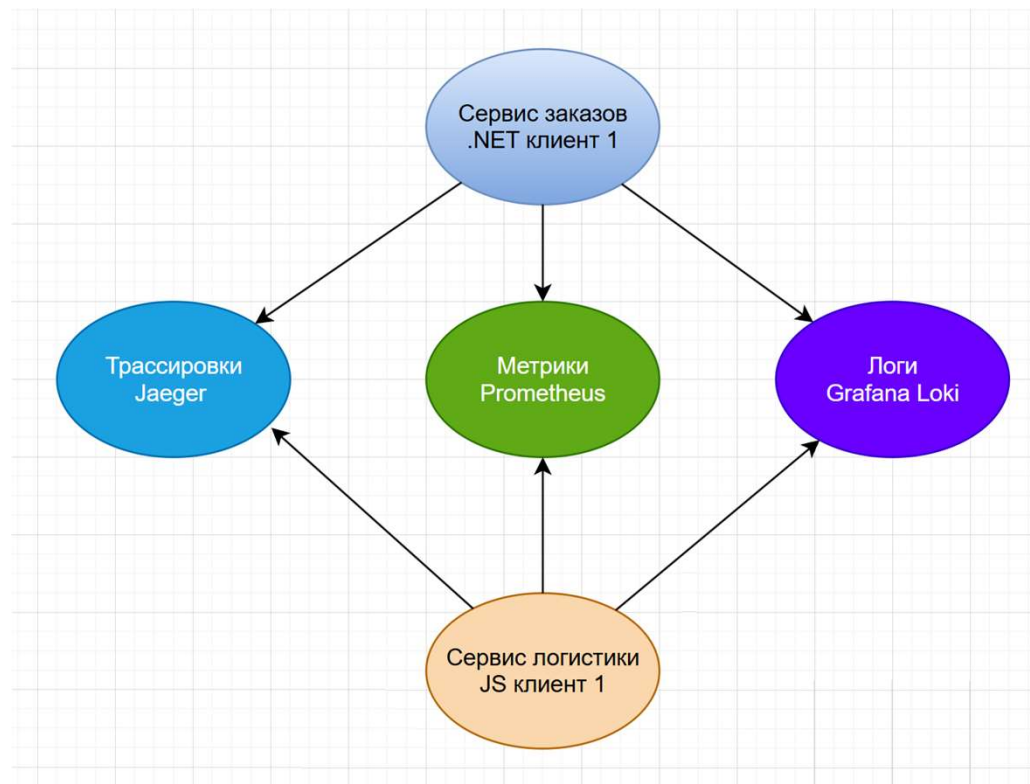
У КАЖОГО СВОИ ПРЕДПОЧТЕНИЯ



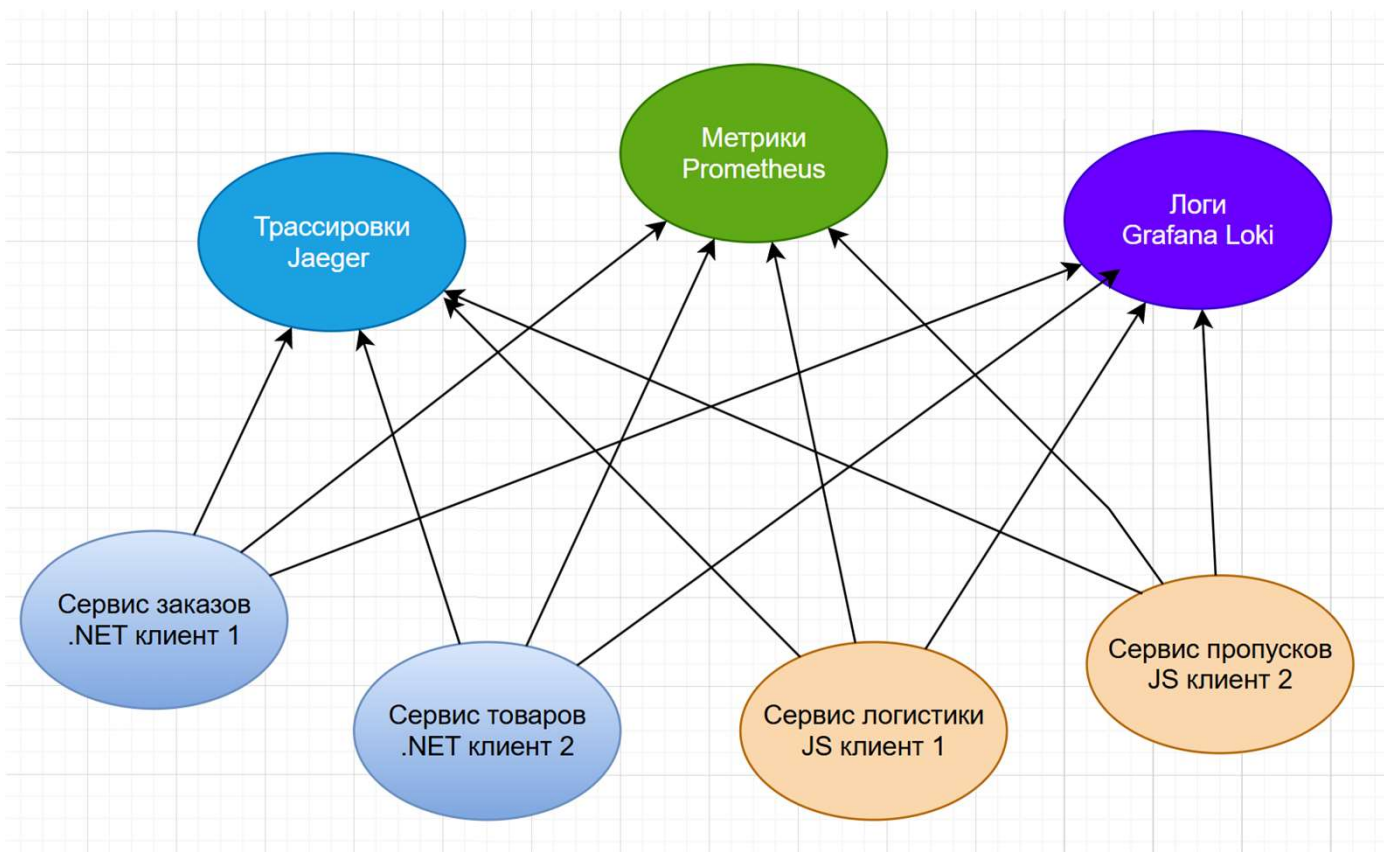
САНКЦИИ



У КАЖДОГО КЛИЕНТА СВОИ ИМЕНОВАНИЯ (UL)



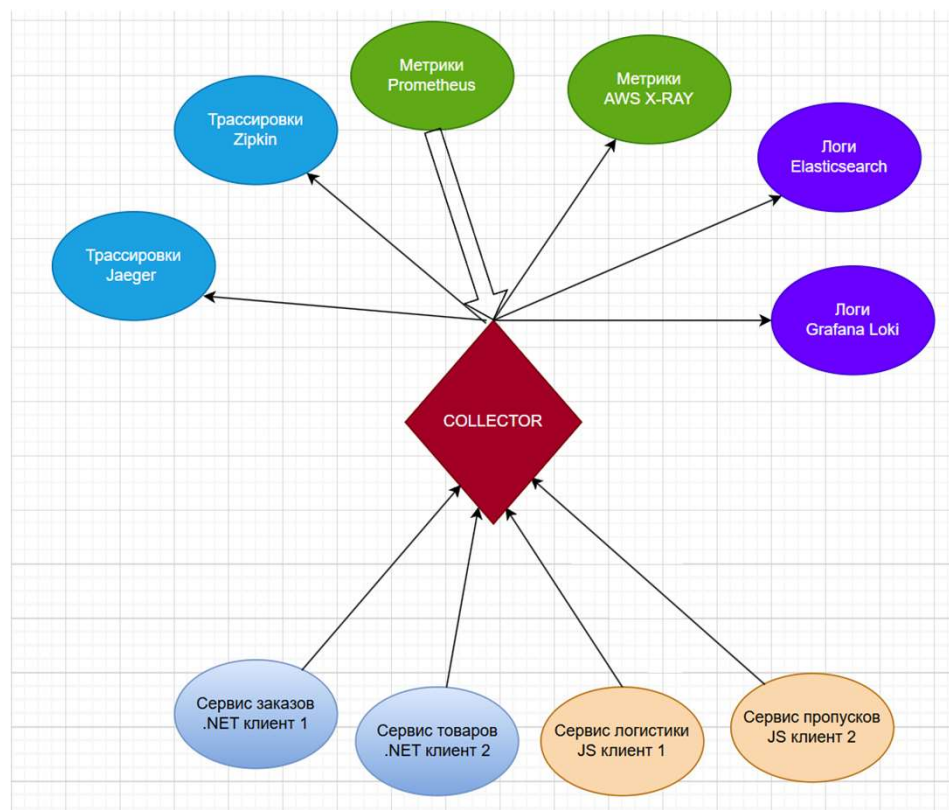
СИЛЬНАЯ СВЯЗАННОСТЬ

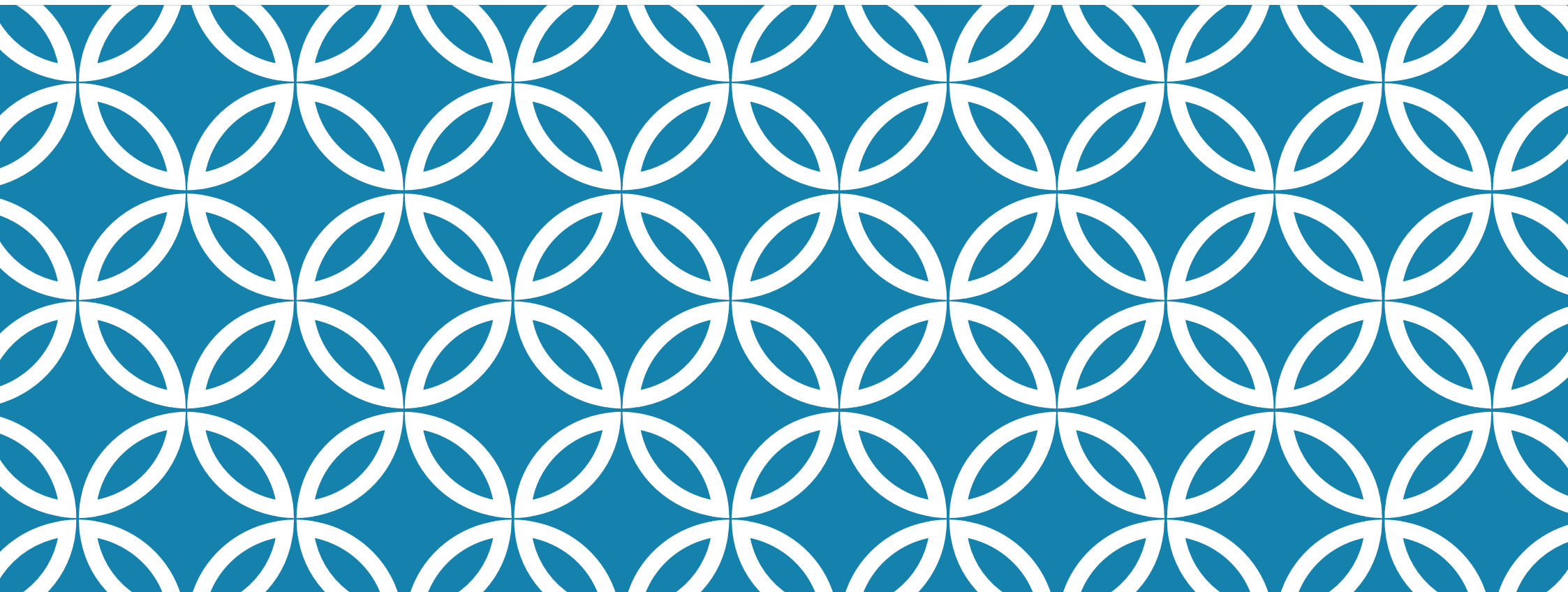


ИТОГ ПО ВНЕДРЕНИЮ

Куча инструментов,
библиотек, определений →
Стандартизация

Сильная связанность →
Прокси коллектор





OPEN TELEMETRY

Унификация телеметрии для
логов, метрик и трассировок

OPENTELEMETRY

OpenTelemetry (OTel) — это открытый стандарт для сбора, обработки и экспорта телеметрии (метрик, логов и трейсов) в распределённых системах. Это проект под эгидой Cloud Native Computing Foundation (CNCF)



ИСТОРИЯ

В 2010 году Google опубликовал статью об инфраструктуре высоко-масштабируемой системы распределенных трассировок Dapper

В 2012 год. Вдохновленные идеями Dapper Twitter выпускают свою систему под названием Zipkin

В 2014 году в мире IT появился Kubernetes. Он не является инструментом трассировки, но мир облачных технологий с этого момента точно изменился.

В 2015, через 3 года после появления Zipkin, Uber выкатил свою систему распределенных трассировок под названием Jaeger

Также в 2015 к Cloud Native Computing Foundation's присоединился проект OpenTracing

В 2017 году Google выкатил проект OpenCensus для сбора метрик

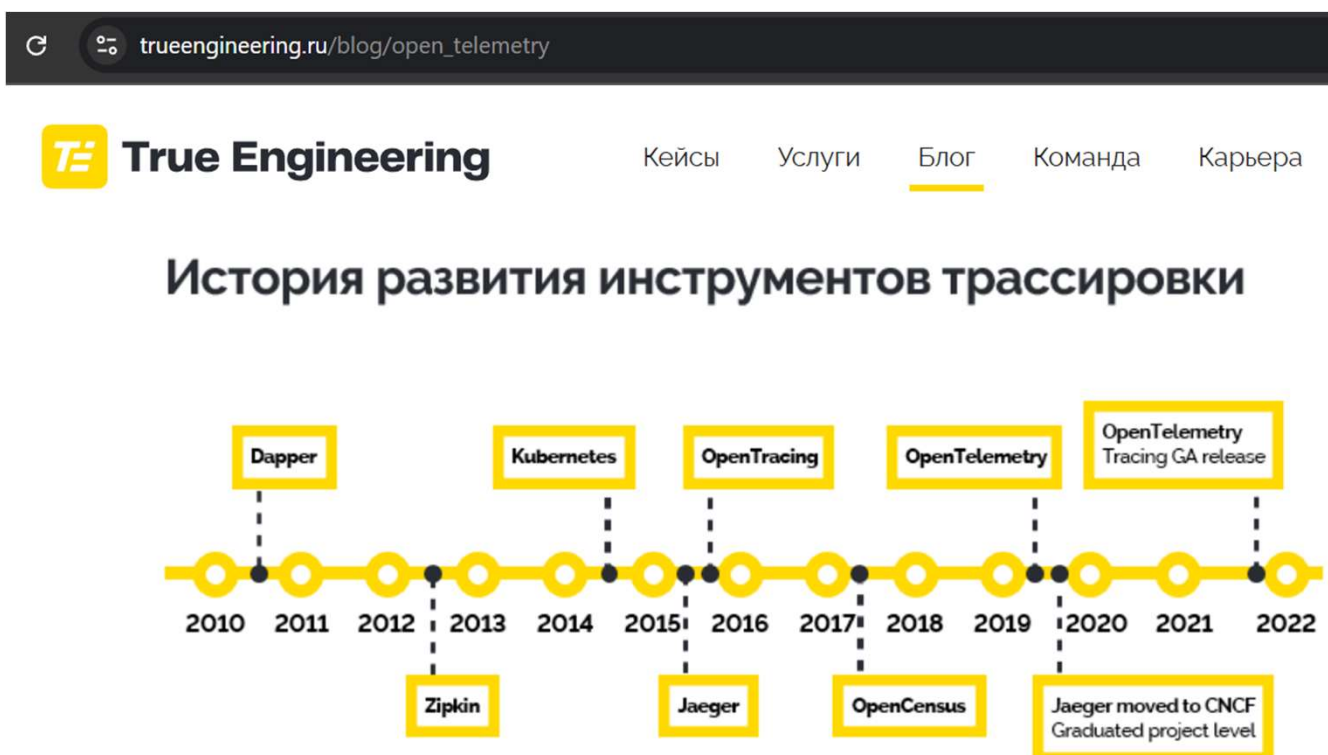
В 2019 году OpenTracing и OpenCensus решили объединиться и сконцентрировать свои силы над общим проектом под названием OpenTelemetry

К концу 2021 года была предоставлена первая общедоступная версия с поддержкой трейсинга.

2022-2023 широкий охват, присоединились почти все BigTech, период Beta версий инструментариев

2024 Апрель .NET Release 1.8.0

ИСТОРИЯ



КАК РАБОТАЕТ

1.Инструментация (Instrumentation)

- Автоматическая (для популярных фреймворков: Python, Java, Go, .NET и т. д.) или ручная.
- Пример: отслеживание HTTP-запросов в Django или Spring Boot.

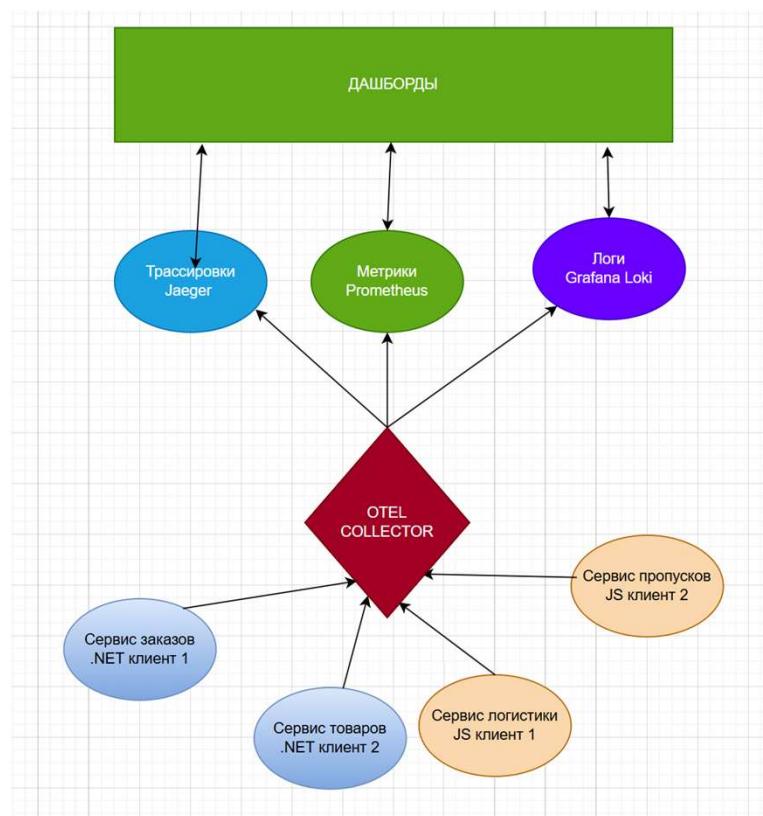
2.Сбор данных (Collector)

- **OpenTelemetry Collector** — отдельный сервис для приёма, обработки и экспорта данных.
- Формат OTLP

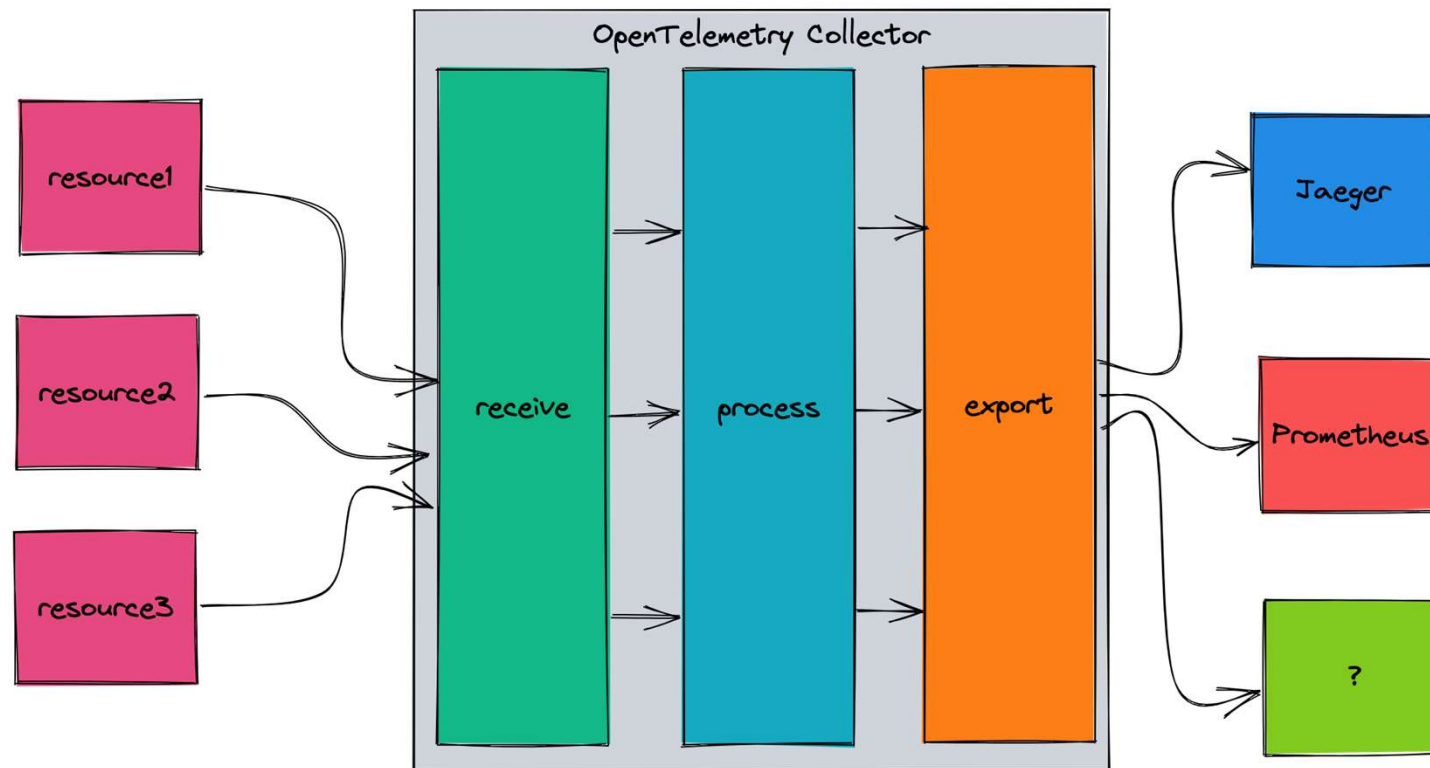
3.Экспорт (Exporters)

- **Jaeger** (трейсы),
- **Prometheus/Grafana** (метрики),
- **ELK/Loki** (логи),
- **Облачные решения** AWS, Microsoft и т. д
- **OTLP Collector**

OTEL COLLECTOR



OTEL COLLECTOR PIPELINE



OTEL CONFIG

```
1 receivers:
2   otlp:
3     protocols:
4       grpc:
5         endpoint: 0.0.0.0:4317
6       http:
7         endpoint: 0.0.0.0:4318
8
9 processors:
10  batch:
11  resource:
12    attributes:
13      - action: insert
14        key: loki.resource.labels
15        value: EnvNameTest
```

OTEL CONFIG

```
17 ∨ exporters:
18 ∨   prometheus:
19     endpoint: "0.0.0.0:8889"
20     # namespace: app
21 ∨   debug:
22     verbosity: detailed # можно detailed, normal, basic
23 ∨   otlp/jaeger:
24     endpoint: "jaeger:4317"
25 ∨     tls:
26       insecure: true
27 ∨   loki:
28     endpoint: "http://loki:3100/loki/api/v1/push"
29     # sentry:
30     # dsn: https://<aaa>@<bbb>.ingest.de.sentry.io/<ccc>
31     # environment: prod
32     # insecure_skip_verify: true
33 ∨   otlp/tempo:
34     endpoint: "tempo:4317"
35 ∨     tls:
36       insecure: true
37
```

OTEL CONFIG

```
38 extensions:
39   health_check:
40     endpoint: "0.0.0.0:13133"
41     path: "/health/status"
42     check_collector_pipeline:
43       enabled: true
44       interval: "5m"
45       exporter_failure_threshold: 5
46
47 service:
48   extensions: [health_check]
49   pipelines:
50     metrics:
51       receivers: [otlp]
52       processors: [batch]
53       exporters: [prometheus, debug]
54     traces:
55       receivers: [otlp]
56       processors: [batch]
57       exporters: [otlp/jaeger, otlp/tempo, debug]
58     logs:
59       receivers: [otlp]
60       processors: [resource]
61       exporters: [loki, debug]
```

ИНТЕГРАЦИЯ С .NET

.NET	OpenTelemetry	Комментарий
Activity	Span	Операция, производимая приложением (бизнес-логика, запросы)
ActivitySource	Tracer	Создатель Activity/Span
Tag	Attribute	Метаданные спана/операции
ActivityKind	SpanKind	Взаимоотношения между зависимыми спанами
ActivityContext	SpanContext	Контекст выполнения, который можно передать в другие спаны
ActivityLink	Link	Ссылка на другой спан

ИНТЕГРАЦИЯ С .NET

```
1 using System.Diagnostics;
2 using System.Diagnostics.Metrics;
3
4 namespace EShop;
5
6 public static class Diagnostic
7 {
8     public const string GlobalSystemName = "EShop";
9     public const string ApplicationName = "EShop.OrderService";
10    public const string InstrumentsSourceName = "EShop.OrderService";
11    public static readonly ActivitySource Source = new(InstrumentsSourceName);
12
13    private static readonly Meter Meter = new(InstrumentsSourceName);
14
15    public static readonly Counter<int> SuccessOrdersCounter =
16        Meter.CreateCounter<int>("success-orders", "count", "the number of success orders");
17
18    public static readonly Counter<int> TryOrdersCounter =
19        Meter.CreateCounter<int>("try-orders", "count", "number of attempts to order");
20 }
```

```
public async Task<OrderItemDto> ReserveProduct(int productId, int quantity)
{
    using var activity = Diagnostic.Source.StartActivity();
    activity?.AddTag("productId", productId);
    activity?.AddTag("quantity", productId);

    var requestUrl = $"{BaseUrl}?productId={productId}&quantity={quantity}";
    var response = await _client.PostAsync(requestUrl, null);
    response.EnsureSuccessStatusCode();

    return await response.Content.ReadFromJsonAsync<OrderItemDto>()
        ?? throw new NullReferenceException(nameof(ReserveProduct));
}
```

ИНТЕГРАЦИЯ С ASP.NET CORE

```
Log.Logger = new LoggerConfiguration()
    .WriteTo.Console()
    .WriteTo.OpenTelemetry()
    .CreateLogger();

builder.Services.AddOpenTelemetry()
    .ConfigureResource(x => x
        .AddService("EShop.OrderService"))
    .WithTracing(b => b
        .AddSource("EShop.OrderService")
        .AddOtlpExporter())
    .WithMetrics(x => x
        .AddMeter("EShop.OrderService")
        .AddOtlpExporter());
```


ИНТЕГРАЦИЯ С ASP.NET CORE

```
builder.Services.AddOpenTelemetry()
    .ConfigureResource(x => x
        .AddService(Diagnostic.ApplicationName)
        .AddAttributes([
            new KeyValuePair<string, object>("EnvNameTest", Diagnostic.GlobalSystemName)
        ]))
    .WithTracing(b => b
        .AddSource(Diagnostic.InstrumentsSourceName)
        .AddAspNetCoreInstrumentation(o =>
            {
                o.RecordException = true;
                o.Filter = httpContext =>
                {
                    var pathValue = httpContext.Request.Path.Value;
                    return pathValue is null || (pathValue != "/metrics" &&
                        !pathValue.StartsWith("/swagger") &&
                        !pathValue.StartsWith("/_vs") &&
                        !pathValue.StartsWith("/framework"));
                }
            });
        )
    .AddHttpClientInstrumentation()
    .AddOtlpExporter()
    .WithMetrics(x => x
        .AddMeter(Diagnostic.InstrumentsSourceName)
        .AddAspNetCoreInstrumentation() // Входящие HTTP-запросы к ASP.NET Core приложению
        .AddHttpClientInstrumentation()
        .AddRuntimeInstrumentation() // Метрики работы .NET runtime: process.runtime.dotnet.gc.collections.count
        .AddProcessInstrumentation() // Метрики процесса приложения: CPU, RAM process.cpu.usage
        .AddOtlpExporter((options, readerOptions) =>
            {
                readerOptions.PeriodicExportingMetricReaderOptions.ExportIntervalMilliseconds =
                    1000;
            }
        ));
```

ИНТЕГРАЦИЯ С NODE.JS

```
17 import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-grpc';
18
19 const traceExporter = new OTLPTraceExporter({
20   url: 'http://localhost:4317',
21 });
22
23 const metricExporter = new OTLPMetricExporter({
24   url: 'http://localhost:4317',
25 });
26
27 const logExporter = new OTLPLogExporter({
28   url: 'http://localhost:4317',
29 });
30
31 const resource = resourceFromAttributes({
32   EnvNameTest: 'EShop',
33   [SEMRESATTRS_SERVICE_NAME]: 'EShop.WarehouseService',
34   // [ SEMRESATTRS_SERVICE_NAMESPACE ]: "yourNameSpace",
35   // [ SEMRESATTRS_SERVICE_VERSION ]: "1.0",
36   // [ SEMRESATTRS_SERVICE_INSTANCE_ID ]: "my-instance-id-1",
37 });
38
39 const sdk = new NodeSDK({
40   resource,
41   traceExporter,
42   metricReader: new PeriodicExportingMetricReader({
43     exporter: metricExporter,
44     exportIntervalMillis: 1000,
45   }),
46   instrumentations: [getNodeAutoInstrumentations(), new NestInstrumentation()],
47 });
48
49 const loggerProvider = new LoggerProvider({
50   resource,
51   processors: [new BatchLogRecordProcessor(logExporter)],
52 });
53 logs.setGlobalLoggerProvider(loggerProvider);
```

СТАТИСТИКА СКАЧИВАНИЙ

Install

```
> npm i @opentelemetry/api
```

Repository

[github.com/open-telemetry/opentele...](https://github.com/open-telemetry/opentelemetry-api)

Homepage

[github.com/open-telemetry/opentele...](https://github.com/open-telemetry/opentelemetry-api)

Weekly Downloads

18 558 837




Version

1.9.0

License

Apache-2.0

 **OpenTelemetry.Api** 1.12.0

Prefix Reserved

[.NET 8.0](#) [.NET Standard 2.0](#) [.NET Framework 4.6.2](#)

[.NET CLI](#) [Package Manager](#) [PackageReference](#)

[Central Package Management](#) [Paket CLI](#) [Script & Interactive](#) [Cake](#)

> dotnet add package OpenTelemetry.Api --version 1.12.0 [Copy](#)

[README](#) [Frameworks](#) [Dependencies](#) [Used By](#)

[Versions](#) [Release Notes](#)

For highlights and announcements see: <https://github.com/open-telemetry/opentelemetry-dotnet/blob/core-1.12.0/RELEASENOTES.md>.

For detailed changes see: <https://github.com/open-telemetry/opentelemetry-dotnet/blob/core-1.12.0/src/OpenTelemetry.Api/CHANGELOG.md>.

Downloads

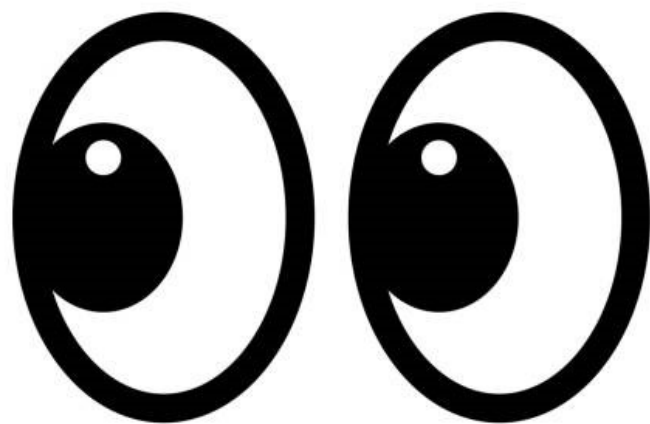
Full stats →

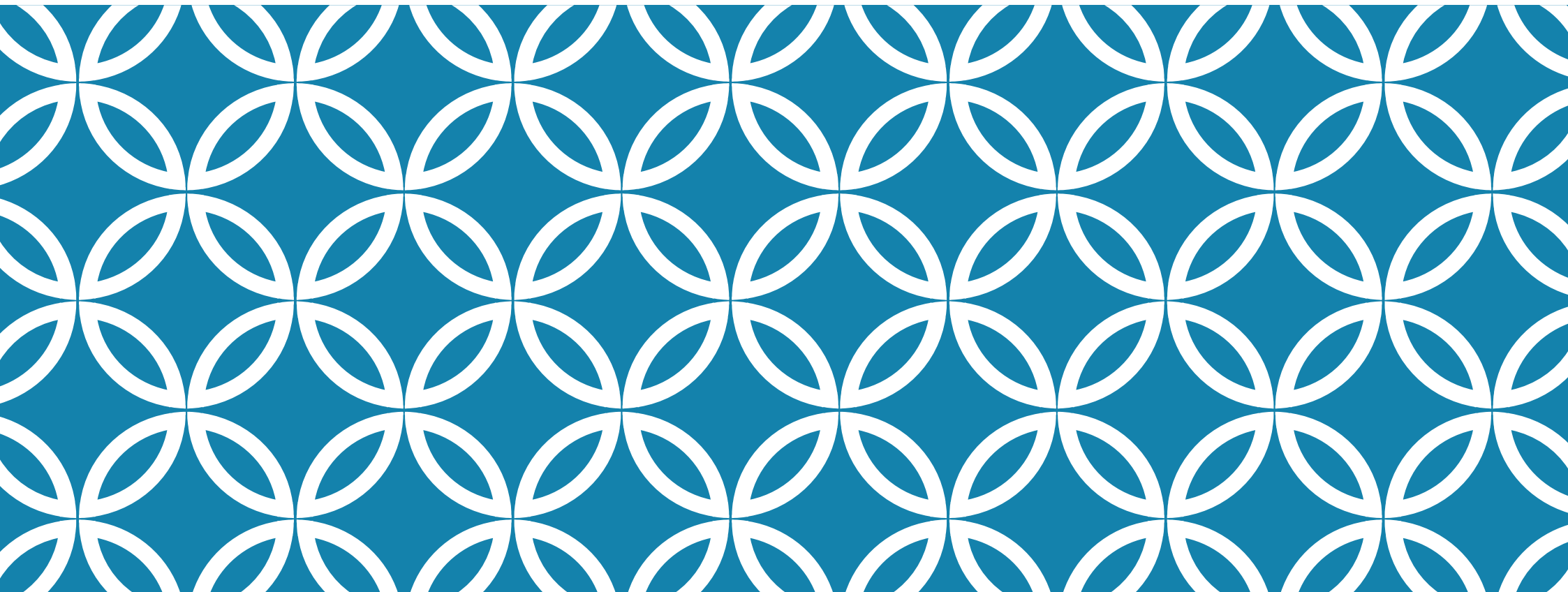
Total 319.6M

Current version 824.6K

Per day average 157.9K

ПОКАЗ ДЕМО





СПАСИБО ЗА ВНИМАНИЕ

Демо проект:
github.com/Nachyn/OpenTelemetryDemo