w .NET

Jak to w zasadzie działa?

Intro

github.com/idle-code/OpenTelemetryDemo

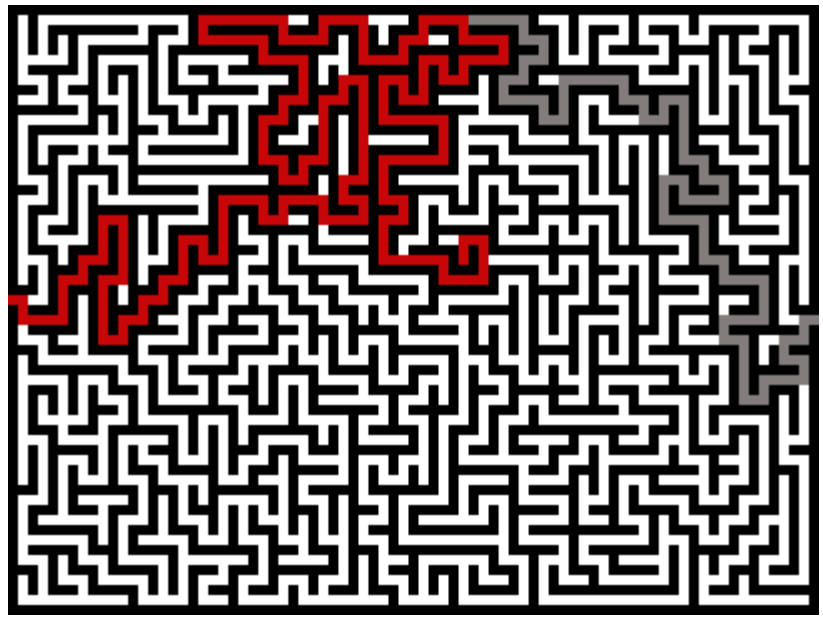
1. OpenTelemetry w .NET
2. Czym jest obserwowalność?
3. Czym jest telemetria?
4. Dlaczego OpenTelemetry?
5. Składniki OpenTelemetry
6. OpenTelemetry Pipeline
7. Projekt demo - TheButton™
8. opentelemetry-dotnet SDK
9. Sygnały
    1. Logs
    2. Traces
    3. Metrics
10. Propagacja kontekstu: Email
11. Processor: Filtrowanie
12. Baggage
13. Processor: Propagacja bagażu

Czym jest obserwowalność?

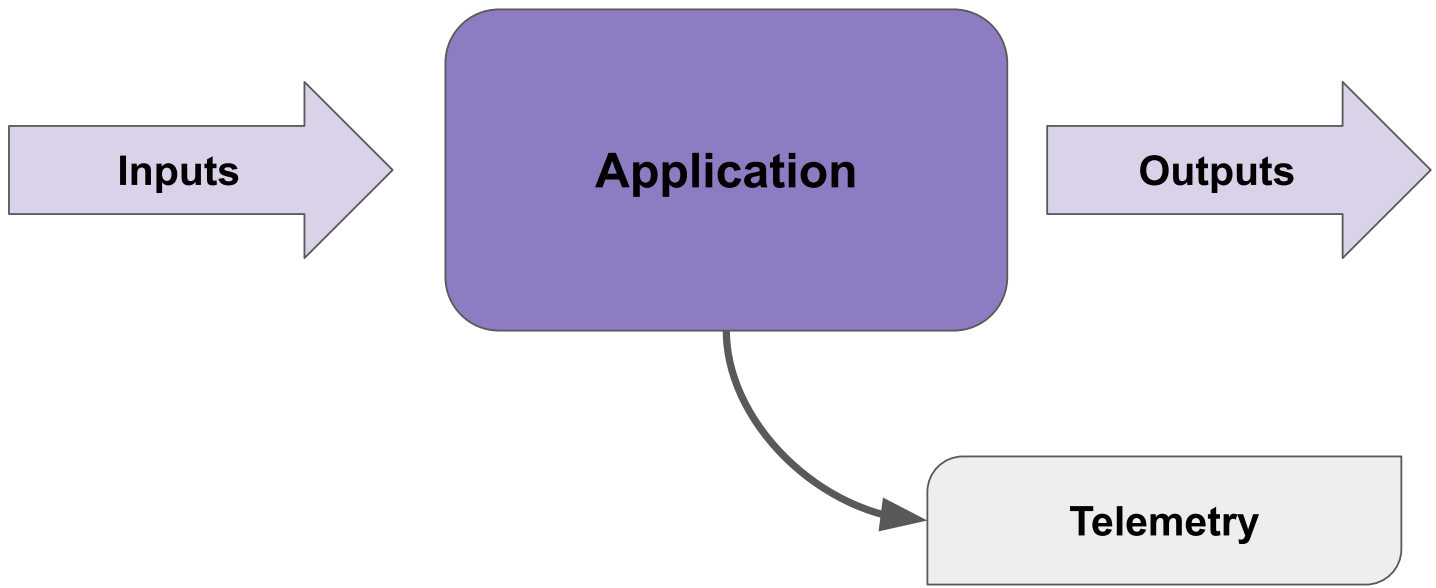
Obserwowalność jest cechą systemu określającą zdolność do zrozumienia jego wewnętrznego stanu (i zachowania) na podstawie danych przez niego generowanych.

By ją osiągnąć, aplikacje muszą być odpowiednio **instrumentowane**, aby generować **telemetrię**.

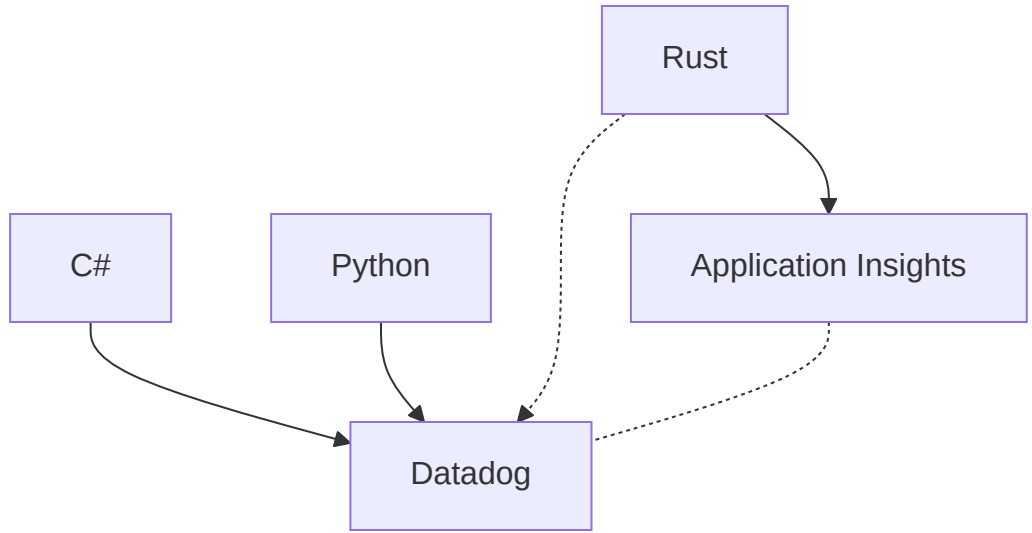
> Dobrą instrumentację mamy, gdy nie potrzebujemy dodawać nowej, aby rozwiązywać problemy pojawiające się w aplikacji



Czym jest telemetria?


Inputs
Application
Outputs
Telemetry

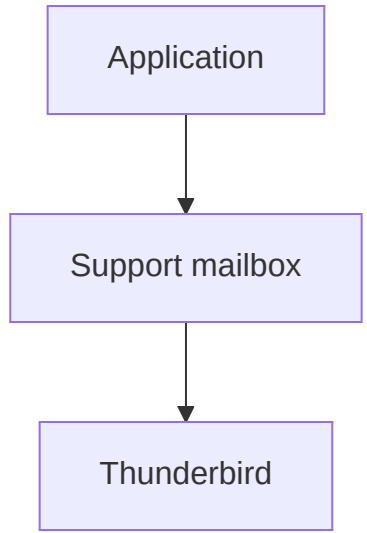
Dlaczego OpenTelemetry?



Bonus: Redneck APM

```
AppDomain.CurrentDomain.UnhandledException += new UnhandledExceptionEventHandler(SendExceptionDetails);

static void SendExceptionDetails(object sender, UnhandledExceptionEventArgs args)
{
    Exception exception = (Exception)args.ExceptionObject;
    MailMessage error_message = new MailMessage();
    PrepareErrorMessageBody(error_message, exception);

    error_message.From = new MailAddress("support@xxx.eu");
    string error_recipients = ConfigurationManager.AppSettings.Get("ErrorReportRecipients");
    foreach (string recipient in error_recipients.Split(';'))
        error_message.To.Add(new MailAddress(recipient));

    SmtpClient smtp_client = new SmtpClient
    {
        Credentials = new System.Net.NetworkCredential("support@xxx.eu", "<the password>"),
        Host = "mail.xxx.eu",
        Port = 25
    };

    smtp_client.Send(error_message);
}
```

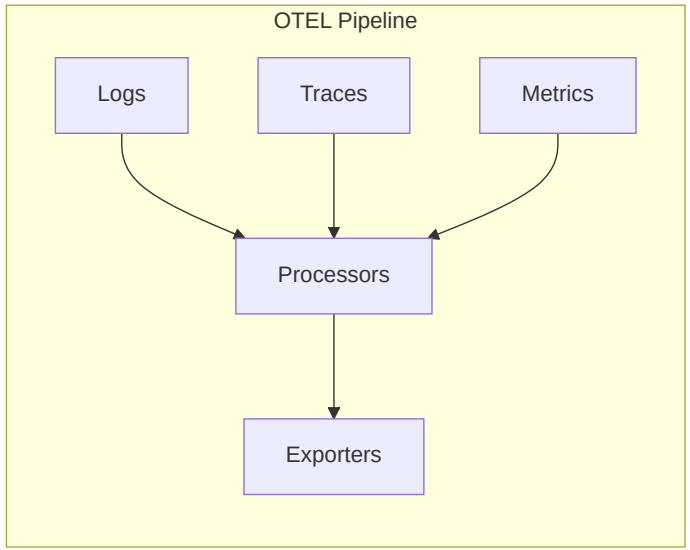
Bonus: Redneck APM



Składniki OpenTelemetry

- Konwencje nazewnicze
- Wskazówki odnośnie API
- Ekosystem dla poszczególnych języków/środowisk programowania
  - SDK
  - Biblioteki obsługujące najczęściej występujące scenariusze
  - Systemy instrumentacji automatycznej
- Protokół wymiany danych OTLP
- OpenTelemetry Collector - proxy do zbierania/przetwarzania i przekazywania sygnałów dalej

OpenTelemetry Pipeline



Projekt demo - TheButton™

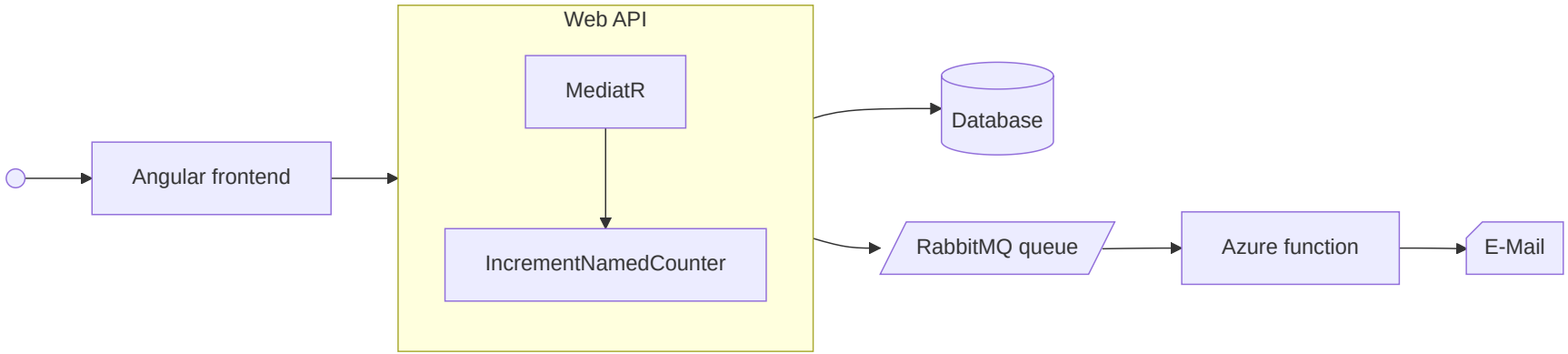
p.z.idlecode@gmail.com
do mnie

Congratulations!

You've reached 60 on test counter, click the link below to grab additional points:
http://localhost:8081/confirm?token=4d4c0c9185504795918163458 92b9351&_traceparent=00-3f710633e014b9c90b70271af31719a3-b3edfe7ffaf4fd18-01&_baggage=request.CounterId%3Dtest,request.Delta%3D1

Odpowiedz | Przekaż dalej

Projekt demo



opentelemetry-dotnet SDK

Oficjalna dystybucja SDK OpenTelemetry dla platformy .NET

github.com/open-telemetry/opentelemetry-dotnet

learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-with-otel

```
OpenTelemetry.Extensions.Hosting

OpenTelemetry.Instrumentation.EntityFrameworkCore
OpenTelemetry.Instrumentation.GrpcNetClient
OpenTelemetry.Instrumentation.Runtime
OpenTelemetry.Instrumentation.SqlClient
RabbitMQ.Client.OpenTelemetry

OpenTelemetry.Exporter.Console
Azure.Monitor.OpenTelemetry.AspNetCore
```

Sygnały

Logs

Krótkie wiadomości tekstowe generowane w czasie uruchamiania kodu

- Informacje o stanie aplikacji
- Decyzja podjęta przez aplikację
- Ostrzeżenia (błędy walidacji)
- Błędy (wyjątki)

```
1  public static void Main(string[] args)
2  {
3      _logger.LogDebug("Got args: {Args}", args);
4      if (args.Length < 2)
5      {
6          _logger.LogError("Not enough arguments provided");
7          return;
8      }
9
10     var a = int.Parse(args[0]);
11     _logger.LogInformation("A = {A}", a);
12     var b = int.Parse(args[1]);
13     _logger.LogInformation("B = {B}", b);
14
15     var sum = a + b;
16     Console.WriteLine("The sum is: {0}", sum);
17 }
```

```
public async Task<NamedCounter> Handle(IncrementNamedCounter request, CancellationToken cancellationToken)
{
    using var _ = _logger.PushProperty("CounterId", request.CounterId);

    var counter = await _dbContext.NamedCounters.SingleOrDefaultAsync(counter => counter.Id == request.CounterId, can
    if (counter is null)
    {
        _logger.LogInformation("No existing counter found - creating a new one");
        counter = new NamedCounter { Id = request.CounterId };
        _dbContext.NamedCounters.Add(counter);
    }

    _logger.LogInformation("Incrementing {CounterId} counter by {Delta}", request.CounterId, request.Delta);
    var oldValue = counter.Value;
    counter.Value += request.Delta;
    _counterMetrics.CounterIncrement(counter.Id, request.Delta);
```

```
public static IDisposable? PushProperty(this ILogger logger, string propertyName, object propertyValue)
{
    return logger.BeginScope(new Dictionary<string, object?>
    {
        { propertyName, propertyValue }
    });
}
```

.NET Logging

- `WithLogging()` : opentelemetry-dotnet SDK rejestruje `OpenTelemetryLoggerProvider`

```
var otel = builder.Services.AddOpenTelemetry();
otel
    .ConfigureResource(resource => resource
        .AddService("WebAPI"))
    .WithLogging(logging => logging
        .AddConsoleExporter())
    .WithTracing(tracing => tracing
        .AddHttpClientInstrumentation()
        .AddAspNetCoreInstrumentation()
        .AddGrpcClientInstrumentation()
        .AddEntityFrameworkCoreInstrumentation(ef => ef.SetDbStatementForText = true)
        .AddRabbitMQInstrumentation()
        .AddSource("WebAPI.*")
        .AddProcessor<BaggageEnrichingProcessor>()
        .AddConsoleExporter())
    .WithMetrics(metrics => metrics
        .AddHttpClientInstrumentation()
        .AddAspNetCoreInstrumentation()
        .AddSqlClientInstrumentation()
        .AddRuntimeInstrumentation()
        .AddMeter("WebAPI.*")
        .AddConsoleExporter());
```

.NET Logging

- `WithLogging()`: opentelemetry-dotnet SDK rejestruje `OpenTelemetryLoggerProvider`
- `ILoggerFactory` tworzy instancję `ILogger` która jest wstrzykiwana do klasy `IncrementNamedCounterHandler`

```
_logger.LogInformation("Incrementing {CounterId} counter by {Delta}", request.CounterId, request.Delta);
```

```
{
  "LogLevel": "Information",
  "Category": "WebAPI.Handlers.IncrementNamedCounterHandler",
  "TraceId": "d0194a116c84d1d36214436295d7fa55",
  "SpanId": "d251c5795cded1e9",
  "Attributes": { "CounterId": "rabarbar", "Delta": 1 }
}
```

- Log jest przetwarzany przez **procesory**
  - Tu następuje **grupowanie** (batching) rekordów
  - Na tym etapie log może zostać **odfiltrowany** (np. ze względu na swój poziom)
- Log jest jest wysyłany do **eksportera** (np. OTLP, Console)

LogRecord

Body	Treść (message)
Attributes	Dodatkowe atrybuty (tagi)
SeverityText / SeverityNumber	Log level
Timestamp / ObservedTimestamp	Czas kiedy log został wyemitowany
TraceId SpanId TraceFlags	Trace context
InstrumentationScope	Instrumentacja generująca log
Resource	Serwis/środowisko generujący log

LogRecord

Body	`Incrementing {CounterId} counter by {Delta}`
Attributes	`CounterId: rabarbar` `Delta: 1`
SeverityText / SeverityNumber	`Info`
Timestamp / ObservedTimestamp	`2025-04-12T14:57:26.6768257Z`
TraceId	`d0194a116c84d1d36214436295d7fa55`
SpanId	`32434d330c6950e9`
TraceFlags	`Recorded`
InstrumentationScope	
Resource	`service.name: WebAPI`

Traces

Operacje wykonywane przez aplikację

- Zapytanie HTTP
- Obsługa zdarzenia
- Interakcja użytkownika

Spans

Operacje wykonywane przez aplikację

- Zapytanie HTTP
- Obsługa zdarzenia
- Interakcja użytkownika
- Zapytanie do bazy danych
- Wywołanie serwisu wewnętrznego
- MediatR handler

.NET Tracing

- `WithTracing()` : opentelemetry-dotnet SDK rejestruje `ActivityListener` do wskazanych `ActivitySource` s

```
var otel = builder.Services.AddOpenTelemetry();
otel
    .ConfigureResource(resource => resource
        .AddService("WebAPI"))
    .WithLogging(logging => logging
        .AddConsoleExporter())
    .WithTracing(tracing => tracing
        .AddHttpClientInstrumentation()
        .AddAspNetCoreInstrumentation()
        .AddGrpcClientInstrumentation()
        .AddEntityFrameworkCoreInstrumentation(ef => ef.SetDbStatementForText = true)
        .AddRabbitMQInstrumentation()
        .AddSource("WebAPI.*")
        .AddProcessor<BaggageEnrichingProcessor>()
        .AddConsoleExporter())
    .WithMetrics(metrics => metrics
        .AddHttpClientInstrumentation()
        .AddAspNetCoreInstrumentation()
        .AddSqlClientInstrumentation()
        .AddRuntimeInstrumentation()
```

.NET Tracing

- `WithTracing()`: opentelemetry-dotnet SDK rejestruje `ActivityListener` do wskazanych `ActivitySource`s
- W aplikacji tworzone jest `ActivitySource`:

```
private static readonly ActivitySource ActivitySource
        = new(typeof(LoggingPipelineBehavior<TRequest, TResponse>).FullName!);
```

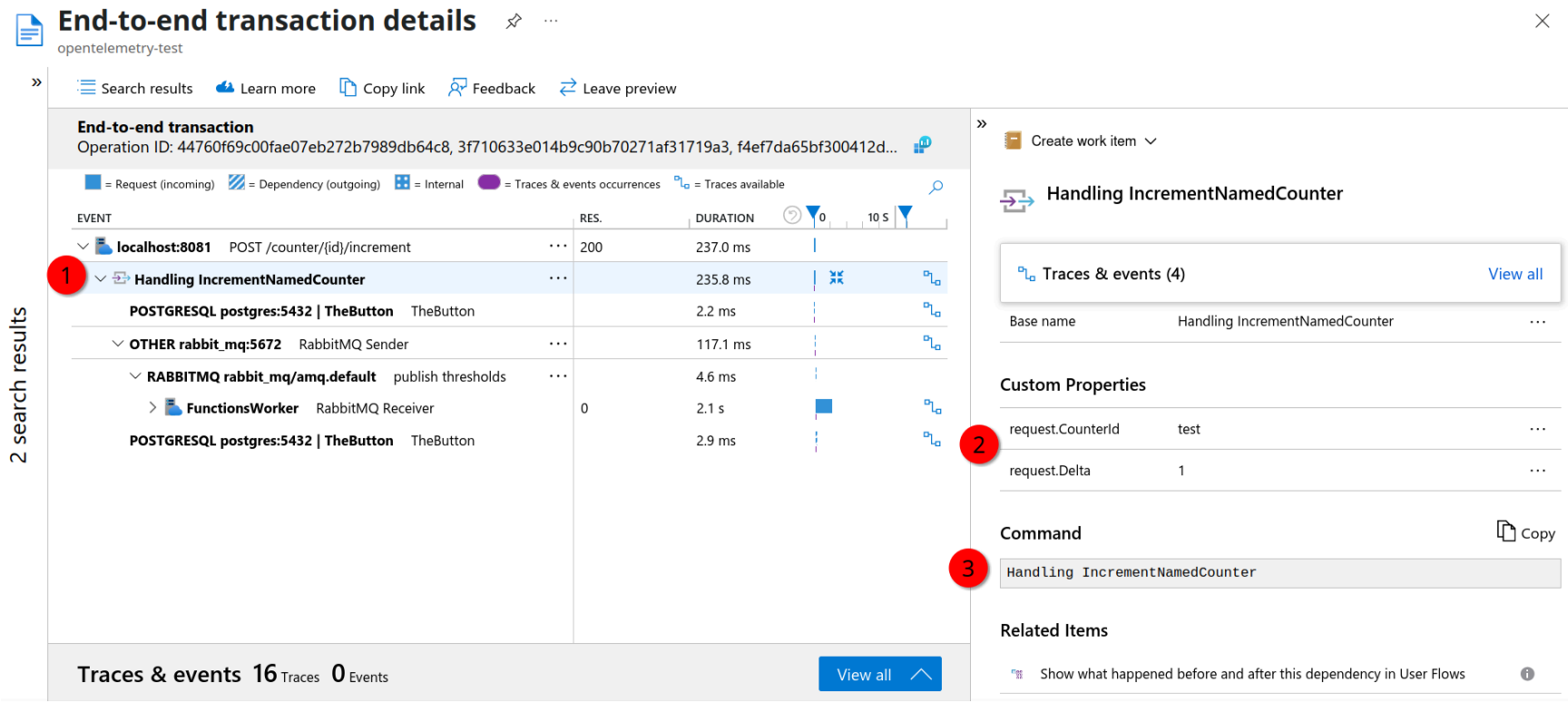
- Jeśli ktoś nasłuchuje na dane `ActivitySource`, wszyscy subskrybenci są informowani o rozpoczęciu/zakończeniu nowego `Activity`:

```
using var activity = ActivitySource.StartActivity($"Handling {requestTypeName}", ActivityKind.Internal);
```

- Span jest przetwarzany przez **procesory**
  - Tu następuje **grupowanie** (batching)
  - Na tym etapie span może zostać **odfiltrowany**
- Span jest jest wysyłany do **eksportera** (np: OTLP, Console)

```
public class LoggingPipelineBehavior<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse>
    where TRequest : notnull
```

```
1   private static readonly ActivitySource ActivitySource
2       = new(typeof(LoggingPipelineBehavior<TRequest, TResponse>).FullName!);
3
4   public async Task<TResponse> Handle(
5       TRequest request,
6       RequestHandlerDelegate<TResponse> next,
7       CancellationToken cancellationToken)
8   {
9       var requestTypeName = typeof(TRequest).Name;
10      using var activity = ActivitySource.StartActivity($"Handling {requestTypeName}", ActivityKind.Internal);
11
12      var requestTags = ToKeyValuePairs(request);
13      foreach (var (key, value) in requestTags)
14      {
15          activity?.SetTag(key, value);
16      }
17
18      return await next();
19  }
```


End-to-end transaction details
opentelemetry-test
Search results
Learn more
Copy link
Feedback
Leave preview
2 search results
End-to-end transaction
Operation ID: 44760f69c00fae07eb272b7989db64c8, 3f710633e014b9c90b70271af31719a3, f4ef7da65bf300412d...
= Request (incoming)
= Dependency (outgoing)
= Internal
= Traces & events occurrences
= Traces available
EVENT
RES.
DURATION
0
10 S
localhost:8081 POST /counter/{id}/increment
200
237.0 ms
1
Handling IncrementNamedCounter
235.8 ms
POSTGRESQL postgres:5432 | TheButton TheButton
2.2 ms
OTHER rabbit_mq:5672 RabbitMQ Sender
117.1 ms
RABBITMQ rabbit_mq/amq.default publish thresholds
4.6 ms
FunctionsWorker RabbitMQ Receiver
0
2.1 s
POSTGRESQL postgres:5432 | TheButton TheButton
2.9 ms
Traces & events 16 Traces 0 Events
View all
Create work item
Handling IncrementNamedCounter
Traces & events (4)
View all
Base name
Handling IncrementNamedCounter
Custom Properties
2
request.CounterId
test
request.Delta
1
Command
Copy
3
Handling IncrementNamedCounter
Related Items
Show what happened before and after this dependency in User Flows

Span

Name	Nazwa operacji
Attributes	Dodatkowe atrybuty (tagi)
Start and duration	Okno czasowe wykonywania operacji
Span status	Status operacji
Span kind	Rodzaj (kategoria) operacji
Parent span ID	ID rodzica
TraceId SpanId TraceFlags	Trace context

Span

Name	`Handling IncrementNamedCounter`
Attributes	`request.CounterId: rabarbar` `request.Delta: 1`
Start and duration	`2025-04-12T16:08:16.4385066Z` `00:00:00.4275037`
Span status	`Unset`
Span kind	`Internal`
Parent span ID	`0907e33699fbd0c7`
TraceId	`230909447214bb90523f50023553b274`
SpanId	`30dc7cfd9ccf4f0a`
TraceFlags	`Recorded`

Metrics

Liczniki reprezentujące aktualny stan systemu

- Liczba obsłużonych zapytań
- Aktualne użycie pamięci
- Opóźnienia obsługiwanych zapytań

.NET Metrics

- `WithMetrics()` : opentelemetry-dotnet SDK subskrybuje się do wskazanych mierników via `.AddMetrics()`
- Przy starcie tworzone są instrumenty takie jak `Counter` :

```
var meter = meterFactory.Create("WebAPI.counter");
_counterIncrements = meter.CreateCounter<int>("WebAPI.counter.increments");
```

- Podczas uruchomienia, kod rejestruje zmiany w mertykach:

```
_counterMetrics.CounterIncrement(counter.Id, request.Delta);
```

```
public void CounterIncrement(string counterName, int delta)
{
    _counterIncrements.Add(delta, new[] { new KeyValuePair<string, object?>("counter_name", counterName) });
}
```

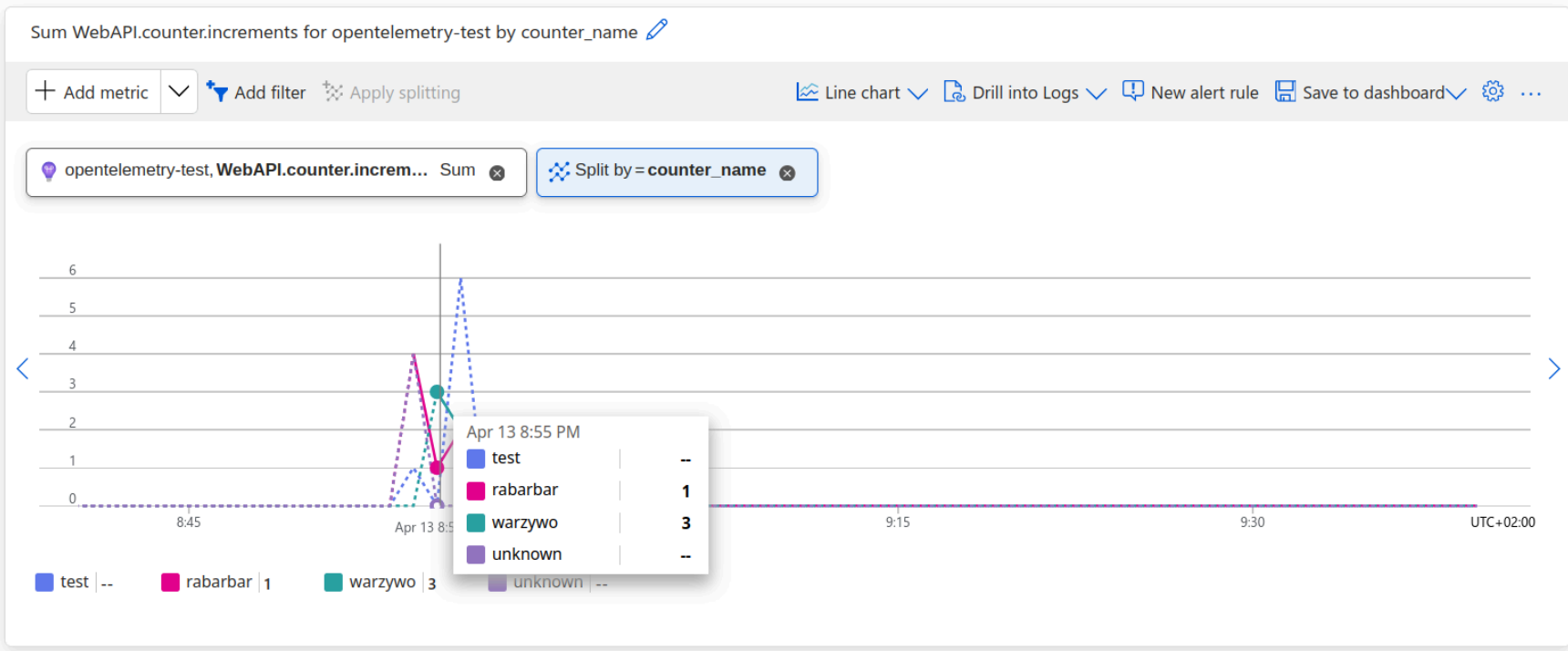
- **Procesory** są odpowiedzialne za agregację i filtrowanie metryk
- Metryki są zbierane co 10 sekund, a potem **eksportowane**
  - `PrometheusExporter` używa metody pull - via `/metrics` endpoint

Metric

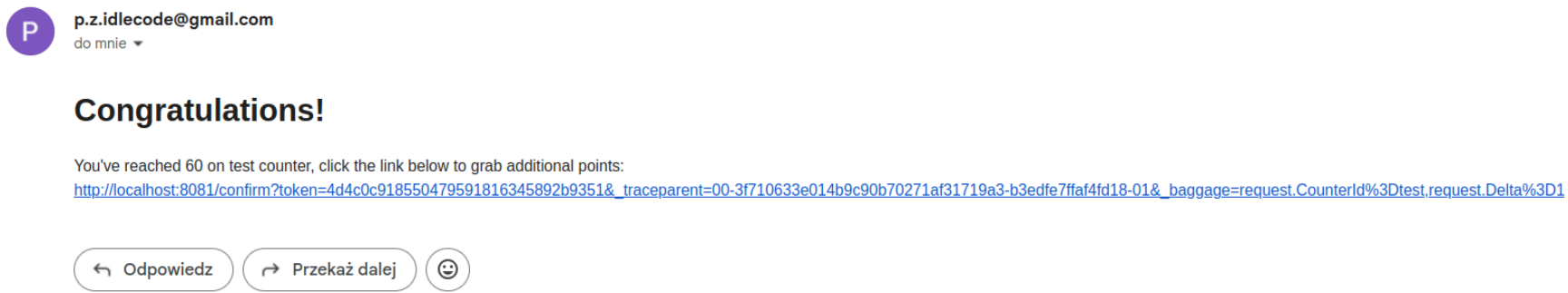
Metric name	Nazwa metryki/instrumentu
Attributes	Wymiary
Value	Wartość
Unit of measurement	Jednostka wartości
Instrumentation scope	Nazwa miernika
Resource	Serwis/środowisko generujący metrykę

Metric

Metric name	`WebAPI.counter.increments`
Attributes	`counter_name: rabarbar`
Value	12
Unit of measurement	
Instrumentation scope	`WebAPI.counter`
Resource	`telemetry.distro.name: Azure.Monitor.OpenTelemetry.AspNetCore`


Sum WebAPI.counter.increments for opentelemetry-test by counter_name
Add metric
Add filter
Apply splitting
Line chart
Drill into Logs
New alert rule
Save to dashboard
opentelemetry-test, WebAPI.counter.increm... Sum
Split by = counter_name
6
5
4
3
2
1
0
8:45
Apr 13 8:55 PM
test --
rabarbar 1
warzywo 3
unknown --
9:15
9:30
UTC+02:00
test --
rabarbar 1
warzywo 3
unknown --

Propagacja kontekstu: Email



Propagacja kontekstu: Email

```
http://localhost:8081/confirm
    ?token=4d4c0c9185504795918163458 92b9351
    &_traceparent=00-3f710633e014b9c90b70271af31719a3-b3edfe7ffaf4fd18-01
    &_baggage=request.CounterId=test,request.Delta=1
```

00-3f710633e014b9c90b70271af31719a3-b3edfe7ffaf4fd18-01

W3C traceparent - https://www.w3.org/TR/trace-context/

Version	Trace ID	Parent (span) ID	Flags
00	3f710633e014b9c90b70271af31719a3	b3edfe7ffaf4fd18	01

Propagacja kontekstu: Email - wysyłanie

```
1   private async ValueTask SendNotification(ThresholdReachedMessage message, CancellationToken cancellationToken)
2   {
3       _logger.LogInformation("Received {@Message}", message);
4       var token = UrlEncoder.Default.Encode(message.BonusToken);
5       var tokenUrl = $"http://localhost:8081/confirm?token={token}";
6       tokenUrl = EnrichWithTraceContext(tokenUrl);
7       await SendEmail(
8           toAddress: "p.z.idlecode@gmail.com",
9           subject: "Threshold reached!",
10          body: $"""
11                <h1>Congratulations!</h1>
12                <div>
13                You've reached {message.Threshold} on {message.CounterId} counter,
14                click the link below to grab additional points:<br/>
15                <a href="{tokenUrl}">{tokenUrl}</a>
16                </div>
17                """,
18          cancellationToken: cancellationToken);
19  }
```

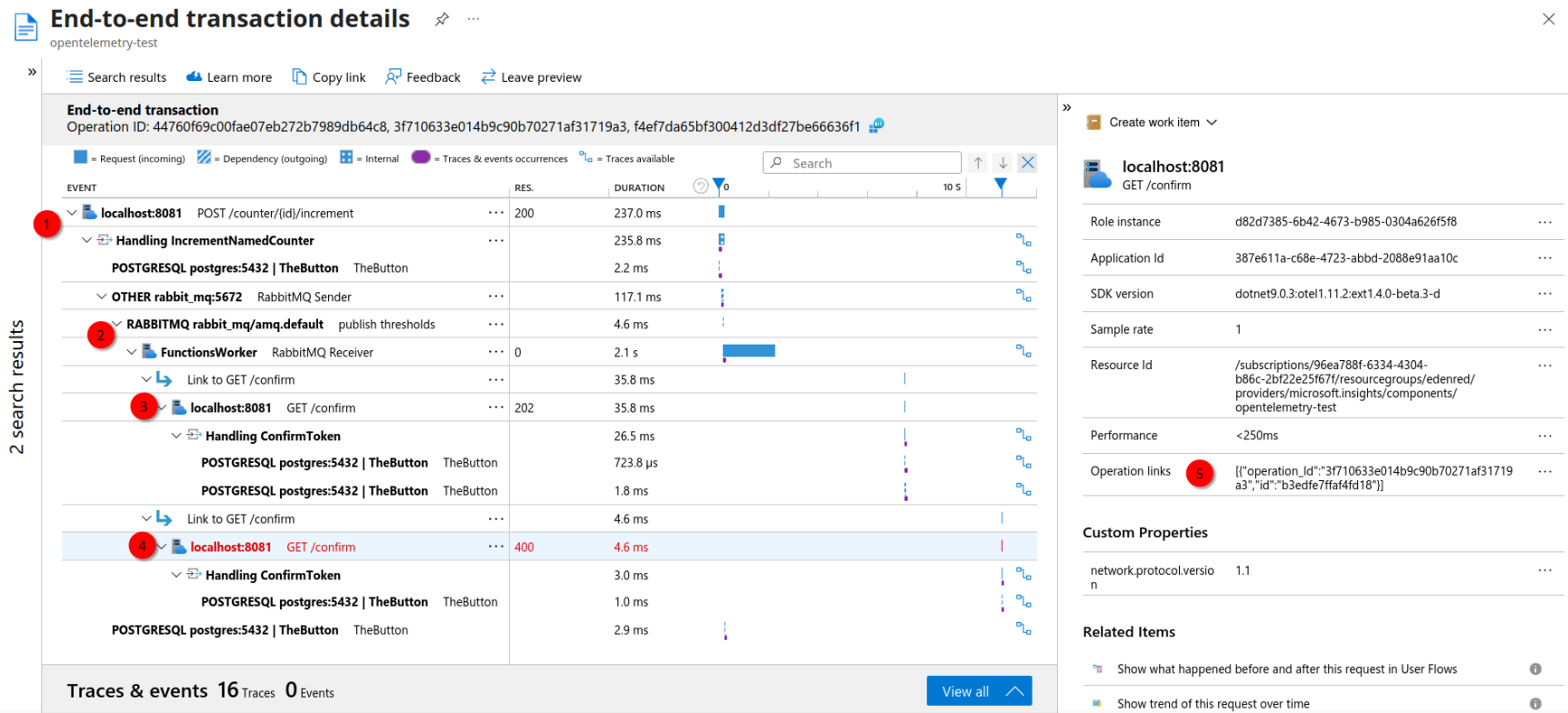
Propagacja kontekstu: Email - wysyłanie

```
private static readonly TextMapPropagator Propagator = Propagators.DefaultTextMapPropagator;
```

```
1   private string EnrichWithTraceContext(string url)
2   {
3       var activity = Activity.Current;
4       if (activity is null)
5       {
6           return url;
7       }
8
9       var uriBuilder = new UriBuilder(url);
10      Propagator.Inject(
11          new PropagationContext(activity.Context, Baggage.Current),
12          uriBuilder,
13          InjectContextTagsIntoQueryParams);
14      return uriBuilder.ToString();
15  }
16
17  private void InjectContextTagsIntoQueryParams(UriBuilder url, string key, string value)
18  {
19      url.Query += $"&_{key}={UrlEncoder.Default.Encode(value)}";
20  }
```

Propagacja kontekstu: Email - odbieranie

```
1   public class ContextFromQueryMiddleware : IMiddleware
2   {
3       private static readonly TextMapPropagator Propagator = Propagators.DefaultTextMapPropagator;
4
5       public async Task InvokeAsync(HttpContext context, RequestDelegate next)
6       {
7           ExtractActivityContextFromUrl(context);
8           await next(context);
9       }
10
11      private void ExtractActivityContextFromUrl(HttpContext context)
12      {
13          var activity = Activity.Current;
14          if (activity is null || !context.Request.QueryString.HasValue)
15          {
16              return;
17          }
18
19          var queryParams = HttpUtility.ParseQueryString(context.Request.QueryString.Value);
20          var parentContext = Propagator.Extract(default, queryParams, ExtractContextTagsFromQueryParams);
21          Baggage.Current = parentContext.Baggage;
22
```


End-to-end transaction details
opentelemetry-test
Search results
Learn more
Copy link
Feedback
Leave preview
2 search results
End-to-end transaction
Operation ID: 44760f69c00fae07eb272b7989db64c8, 3f710633e014b9c90b70271af31719a3, f4ef7da65bf300412d3df27be66636f1
= Request (incoming)
= Dependency (outgoing)
= Internal
= Traces & events occurrences
= Traces available
Search
EVENT
RES.
DURATION
0
10 s
1
localhost:8081 POST /counter/{id}/increment 200 237.0 ms
Handling IncrementNamedCounter 235.8 ms
POSTGRESQL postgres:5432 | TheButton TheButton 2.2 ms
OTHER rabbit_mq:5672 RabbitMQ Sender 117.1 ms
2
RABBITMQ rabbit_mq/amq.default publish thresholds 4.6 ms
FunctionsWorker RabbitMQ Receiver 0 2.1 s
Link to GET /confirm 35.8 ms
3
localhost:8081 GET /confirm 202 35.8 ms
Handling ConfirmToken 26.5 ms
POSTGRESQL postgres:5432 | TheButton TheButton 723.8 µs
POSTGRESQL postgres:5432 | TheButton TheButton 1.8 ms
Link to GET /confirm 4.6 ms
4
localhost:8081 GET /confirm 400 4.6 ms
Handling ConfirmToken 3.0 ms
POSTGRESQL postgres:5432 | TheButton TheButton 1.0 ms
POSTGRESQL postgres:5432 | TheButton TheButton 2.9 ms
Traces & events 16 Traces 0 Events
View all
Create work item
localhost:8081
GET /confirm
Role instance d82d7385-6b42-4673-b985-0304a626f5f8
Application Id 387e611a-c68e-4723-abbd-2088e91aa10c
SDK version dotnet9.0.3:otel1.11.2:ext1.4.0-beta.3-d
Sample rate 1
Resource Id /subscriptions/96ea788f-6334-4304-b86c-2bf22e25f67f/resourcegroups/edenred/providers/microsoft.insights/components/opentelemetry-test
Performance <250ms
Operation links 5 [{"operation_Id":"3f710633e014b9c90b70271af31719a3","id":"b3edfe7ffaf4fd18"}]
Custom Properties
network.protocol.version 1.1
Related Items
Show what happened before and after this request in User Flows
Show trend of this request over time

Processor: Filtrowanie

```
1   public class QueryFilteringProcessor : BaseProcessor<Activity>
2   {
3       public override void OnEnd(Activity data)
4       {
5           var commandText = data.GetTagItem("db.statement");
6           if (commandText is not null)
7           {
8               var query = (string)commandText;
9               if (query.StartsWith("SELECT") && data.Duration < TimeSpan.FromMilliseconds(30))
10              {
11                  data.ActivityTraceFlags &= ~ActivityTraceFlags.Recorded;
12              }
13          }
14      }
15  }
```

Baggage

Globalne metadane operacji

- ID użytkownika
- Tenant ID

Processor: Propagacja bagażu

```
1   public class BaggageEnrichingProcessor : BaseProcessor<Activity>
2   {
3       public override void OnStart(Activity data)
4       {
5           foreach (var (tagKey, tagValue) in Baggage.Current)
6           {
7               data.SetTag(tagKey, tagValue);
8           }
9       }
10  }
```