BASTA!
by entwickler.de

Distributed Tracing with OpenTelemetry for ASP.NET Core

Marc Müller
Principal Consultant

marc.mueller@4tecture.ch
@muellermarc
www.4tecture.ch

4tecture©
empower your software solutions

About me:

Marc Müller
Principal Consultant
@muellermarc

Microsoft Most Valuable Professional

4tecture©
empower your software solutions

Our Products:

Multi-Tenant OpenID Connect Identity Provider

ProAuth

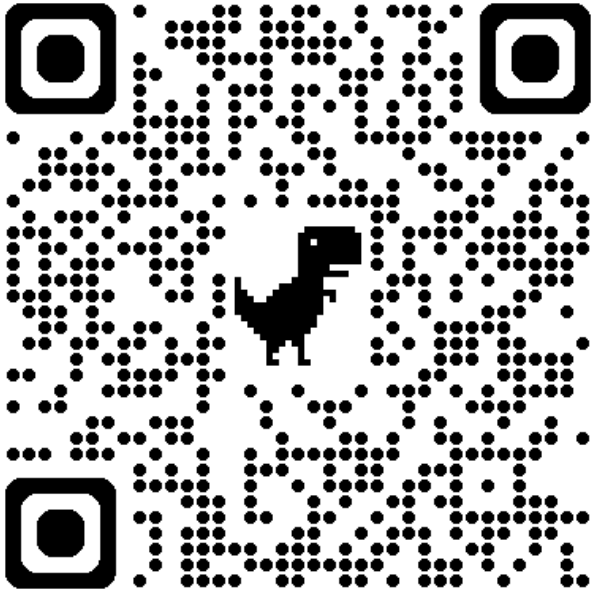
www.proauth.net

Enterprise Application Framework for .NET

ReaFx

www.reafx.net

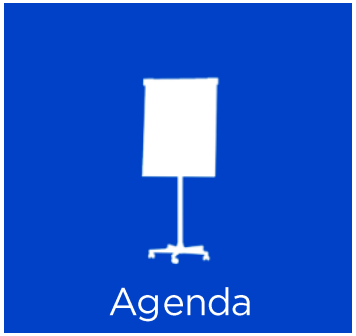
Slide Download



https://www.4tecture.ch/events/basta25-otel

Agenda

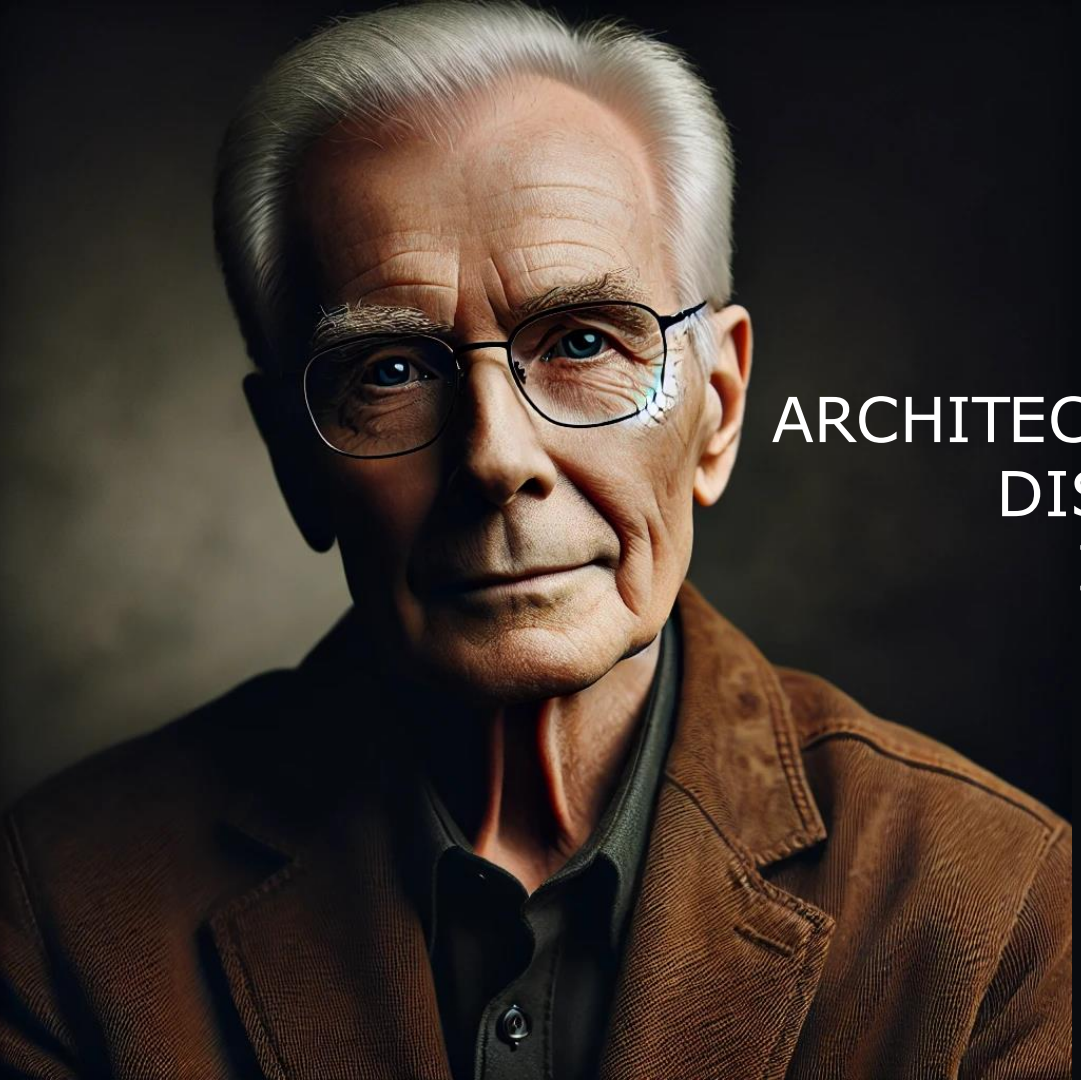
- Why OTEL
- Signals
- Hands-on instrumentation
- Collectors patterns
- Live-Demo
- Q&A



Chapter 1/8

Intro

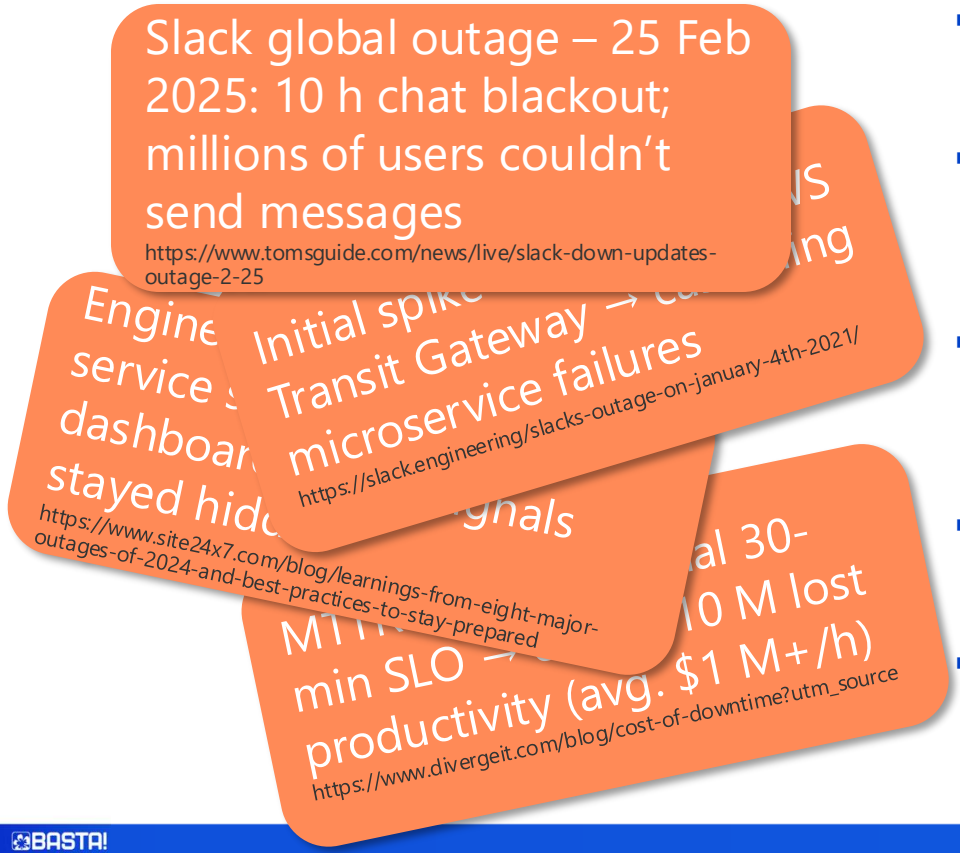
4tecture
empower your software solutions



ARCHITECTING AND IMPLEMENTING DISTRIBUTED APPLICATIONS IS NOT STRESSFUL AT ALL

John, 25 years old

Why observability matters



- 82 % of orgs still need >1 h to resolve incidents
- Biggest barrier is skills—48 % cite "knowledge gap about observability tooling"
- Teams that wired traces + metrics + logs (GitHub push pipeline) cut incident time by 65 %
- AI-driven correlation across all signals is 2025's top trend
- OpenTelemetry gives a vendor-neutral path to those wins—already stable since .NET 8

Observability ≠ monitoring.
It's about **answering *why*** your system fails *before* customers notice.

Observability

METRICS

LOGS

TRACES


DEMO
Demo App

OpenTelemetry

- Protocol
- SDKs

- Current Signals
  - Traces
  - Metrics
  - Logs

Traces and Spans

Trace

- Something which is being done
- Structure
  - Child spans
  - Baggage
  - Tags and attributes

Span

- Structured blob of data
- Items
  - SpanId (Unique Id)
  - TraceId (Correlation Id)
  - Duration
  - Timestamp
  - ParentSpanId (CausalityId)
- .NET Ergonomics
  - System.Diagnostics.Activity
  - Activity Source

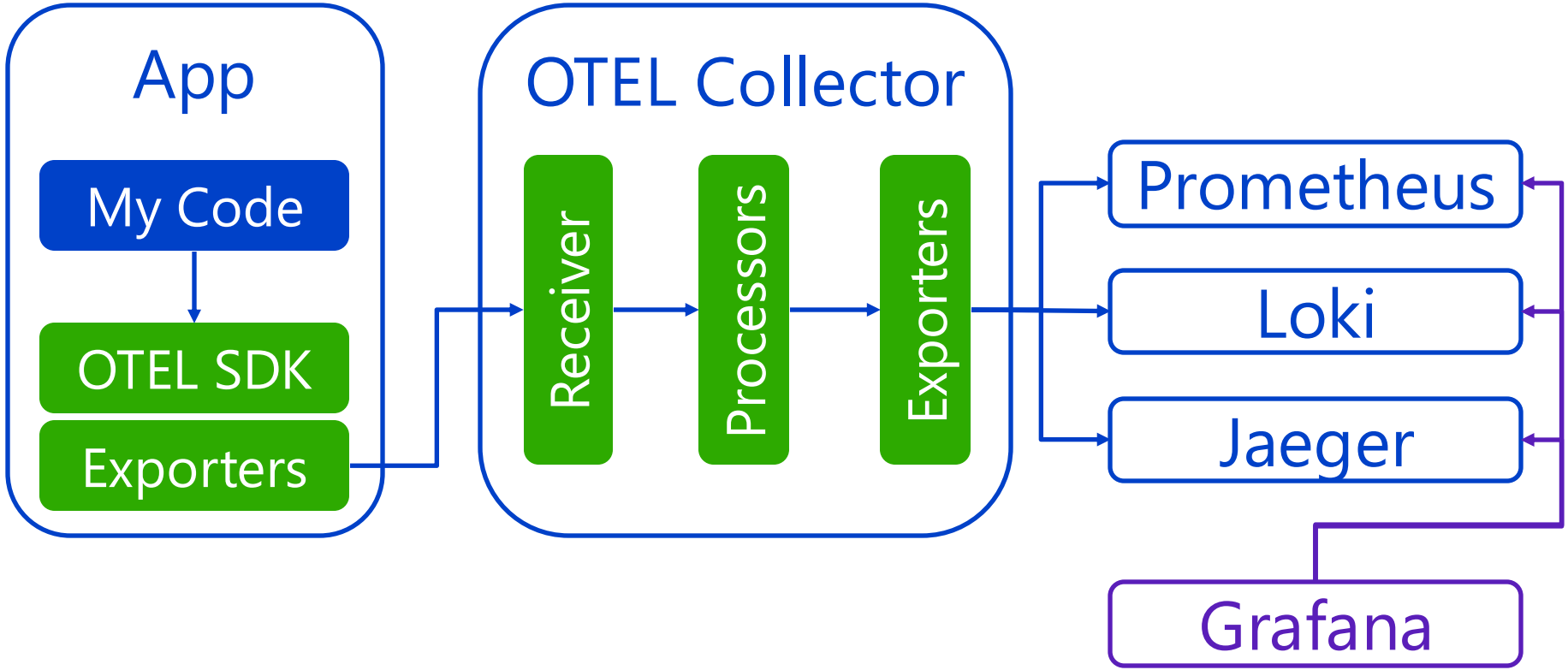
Logs

- Point in time action
- Where context doesn’t exist
- Useful for:
  - Access Logs
  - Audit Logs
- Not useful for:
  - Application debugging
- .NET Ergonomics
  - Integrates directly with Microsoft.Extensions.Logging.ILogger

Metrics

- Numeric measurements over time
- OTEL Metric Instruments
  - Counters (monotonic, e.g. request count)
  - Histograms (value distributions, e.g. latency)
  - Gauges / Observers (instantaneous values, e.g. CPU usage)
- .NET Ergonomics
  - Built-in Meter (System.Diagnostics)
  - Auto-Instrumentation – many libraries emit metrics out-of-the-box

Overall Architecture




Instrumentation
4tecture
empower your software solutions

Setup

```
using OpenTelemetry.Logs;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;

// Ideally, you will want this name to come from a config file, constants file, etc.
var serviceName = "dice-server";
var serviceVersion = "1.0.0";

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddOpenTelemetry()
    .ConfigureResource(resource => resource.AddService(
        serviceName: serviceName,
        serviceVersion: serviceVersion))
    .WithTracing(tracing => tracing
        .AddSource(serviceName)
        .AddAspNetCoreInstrumentation()
        .AddConsoleExporter())
    .WithMetrics(metrics => metrics
        .AddMeter(serviceName)
        .AddConsoleExporter());

builder.Logging.AddOpenTelemetry(options => options
    .SetResourceBuilder(ResourceBuilder.CreateDefault().AddService(
        serviceName: serviceName,
        serviceVersion: serviceVersion))
    .AddConsoleExporter());

builder.Services.AddControllers();

var app = builder.Build();

app.MapControllers();

app.Run();
```


Siehe https://opentelemetry.io/docs/languages/dotnet/instrumentation/

Setup

- ResourceBuilder
  - Unique service name per resource
- Resource Level Attributes
  - Applies to every Span, Metric, and Log
  - Applied asynchronously
  - Run at startup
  - Examples: pod name, service version, environment, etc.

```
var resourceBuilder =
    ResourceBuilder
        .CreateDefault()
        .AddService(serviceName: serviceName, serviceVersion: serviceVersion)
        .AddAttributes(new Dictionary<string, object>
        {
            ["environment.name"] = "production",
            ["team.name"] = "backend"
        });
```


Source: https://opentelemetry.io/docs/languages/dotnet/instrumentation/

Example 1/4

```
public WeatherForecastController(ILogger<WeatherForecastController> logger, InstrumentationSource instrumentationSource)
{
    this.logger = logger ?? throw new ArgumentNullException(nameof(logger));

    ArgumentNullException.ThrowIfNull(instrumentationSource);
    this.activitySource = instrumentationSource.ActivitySource;
    this.freezingDaysCounter = instrumentationSource.FreezingDaysCounter;
}
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/examples/AspNetCore/Controllers/WeatherForecastController.cs

Example 2/4

```
[HttpGet]
public IEnumerable<WeatherForecast> Get()
{
    using var scope = this.logger.BeginIdScope(Guid.NewGuid().ToString("N"));

    // Making a http call here to serve as an example of
    // how dependency calls will be captured and treated
    // automatically as child of incoming request.
    var res = HttpClient.GetStringAsync(new Uri("http://google.com")).Result;

    // Optional: Manually create an activity. This will become a child of
    // the activity created from the instrumentation library for AspNetCore.
    // Manually created activities are useful when there is a desire to track
    // a specific subset of the request. In this example one could imagine
    // that calculating the forecast is an expensive operation and therefore
    // something to be distinguished from the overall request.
    // Note: Tags can be added to the current activity without the need for
    // a manual activity using Activity.Current?.SetTag()
    using var activity = this.activitySource.StartActivity("calculate forecast");

    var forecast = Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = RandomNumberGenerator.GetInt32(-20, 55),
        Summary = Summaries[RandomNumberGenerator.GetInt32(Summaries.Length)],
    })
    .ToArray();

    // Optional: Count the freezing days
    this.freezingDaysCounter.Add(forecast.Count(f => f.TemperatureC < 0));

    this.logger.WeatherForecastGenerated(LogLevel.Information, forecast.Length, forecast);

    return forecast;
}
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/examples/AspNetCore/Controllers/WeatherForecastController.cs

Example 3/4

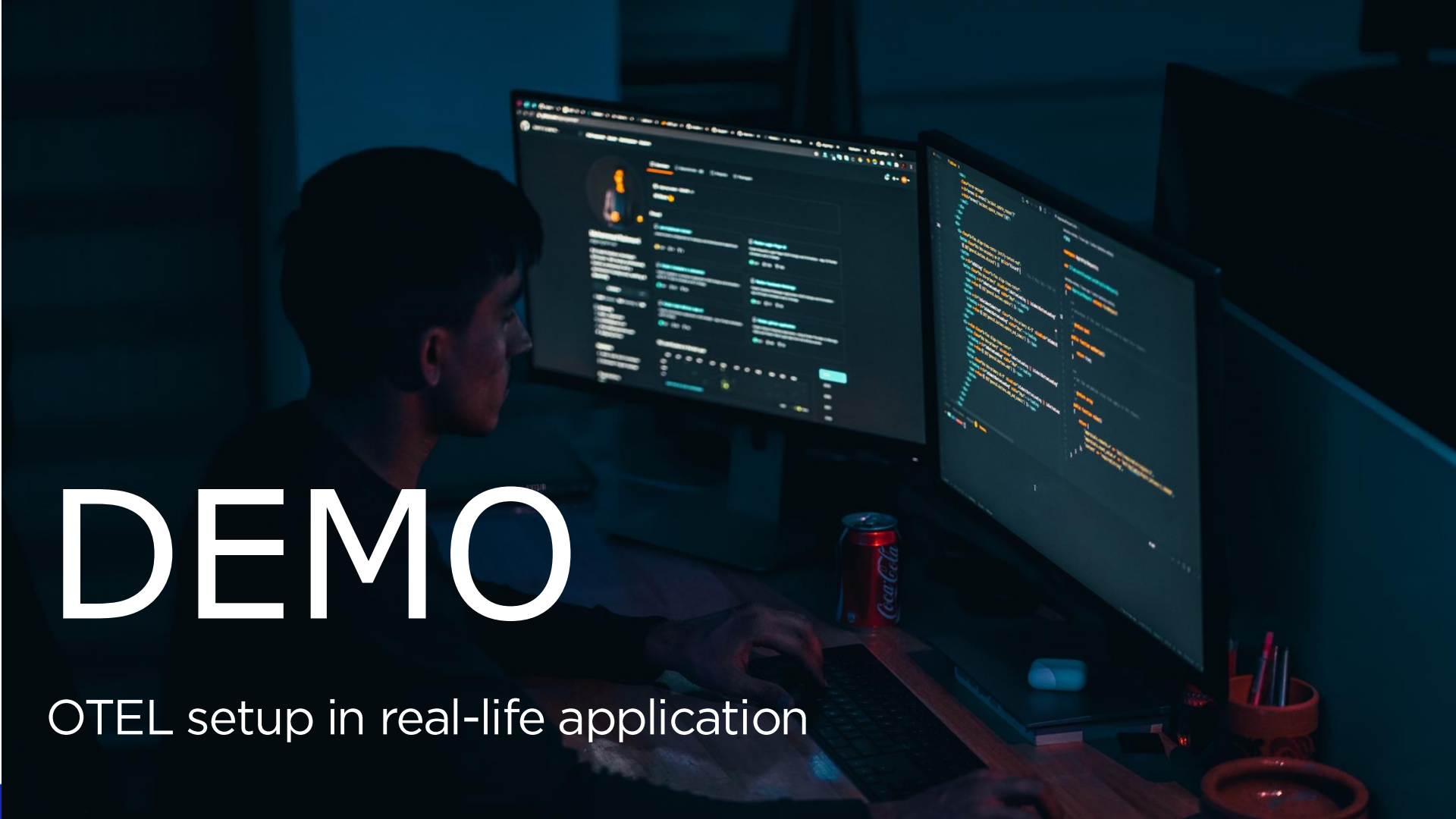
```
1   // Copyright The OpenTelemetry Authors
2   // SPDX-License-Identifier: Apache-2.0
3
4   namespace Examples.AspNetCore;
5
6   using System.Diagnostics;
7   using System.Diagnostics.Metrics;
8
9   /// <summary>
10  /// It is recommended to use a custom type to hold references for
11  /// ActivitySource and Instruments. This avoids possible type collisions
12  /// with other components in the DI container.
13  /// </summary>
14  public sealed class InstrumentationSource : IDisposable
15  {
16      internal const string ActivitySourceName = "Examples.AspNetCore";
17      internal const string MeterName = "Examples.AspNetCore";
18      private readonly Meter meter;
19
20      public InstrumentationSource()
21      {
22          string? version = typeof(InstrumentationSource).Assembly.GetName().Version?.ToString();
23          this.ActivitySource = new ActivitySource(ActivitySourceName, version);
24          this.meter = new Meter(MeterName, version);
25          this.FreezingDaysCounter = this.meter.CreateCounter<long>("weather.days.freezing", description: "The number of days where the temperature is below freezing");
26      }
27
28      public ActivitySource ActivitySource { get; }
29
30      public Counter<long> FreezingDaysCounter { get; }
31
32      public void Dispose()
33      {
34          this.ActivitySource.Dispose();
35          this.meter.Dispose();
36      }
37  }
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/examples/AspNetCore/Controllers/WeatherForecastController.cs

Example 4/4

```
1   // Copyright The OpenTelemetry Authors
2   // SPDX-License-Identifier: Apache-2.0
3
4   namespace Examples.AspNetCore.Controllers;
5
6   internal static partial class WeatherForecastControllerLog
7   {
8       private static readonly Func<ILogger, string, IDisposable?> Scope = LoggerMessage.DefineScope<string>("{Id}");
9
10      public static IDisposable? BeginIdScope(this ILogger logger, string id) => Scope(logger, id);
11
12      [LoggerMessage(EventId = 1, Message = "WeatherForecasts generated {Count}: {Forecasts}")]
13      public static partial void WeatherForecastGenerated(this ILogger logger, LogLevel logLevel, int count, WeatherForecast[] forecasts)
14  }
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/examples/AspNetCore/Controllers/WeatherForecastController.cs


DEMO
OTEL setup in real-life application


Best Practices
4tecture
empower your software solutions

Coding Best Practices

- Use constants for span names / tag names
- Extension methods for activity for repetitive tasks (i.e. start activity with tags)
- StartActivity vs Activity.Current

Adding context

- Add context like a product id as tag
- Helper methods can use
  Activity.Current?.SetTag(“productid”, productid)

Span Event

- Point in time action without duration
- Like a structured log with span/trace

```
myActivity?.AddEvent(new("Init"));
...
myActivity?.AddEvent(new("End"));
```

```
var eventTags = new ActivityTagsCollection
{
    { "operation", "calculate-pi" },
    { "result", 3.14159 }
};

activity?.AddEvent(new("End Computation", DateTimeOffset.Now, eventTags));
```


Source: https://opentelemetry.io/docs/languages/dotnet/instrumentation/#add-events-to-activities

Span Link

- Casual Link
- Not a dependency
- Transitions Trace Context
- Alternative to parent/child relationship

```
var links = new List<ActivityLink>
{
    new ActivityLink(activityContext1),
    new ActivityLink(activityContext2),
    new ActivityLink(activityContext3)
};

var activity = MyActivitySource.StartActivity(
    ActivityKind.Internal,
    name: "activity-with-links",
    links: links);
```

Activity Status

- Indicates if completed successfully

```
private int rollOnce()
{
    using (var childActivity = activitySource.StartActivity("rollOnce"))
    {
        int result;

        try
        {
            result = Random.Shared.Next(min, max + 1);
            childActivity?.SetTag("dicelib.rolled", result);
        }
        catch (Exception ex)
        {
            childActivity?.SetStatus(ActivityStatusCode.Error, "Something bad happened!");
            childActivity?.AddException(ex);
            throw;
        }

        return result;
    }
}
```


Source: https://opentelemetry.io/docs/languages/dotnet/instrumentation

Processors

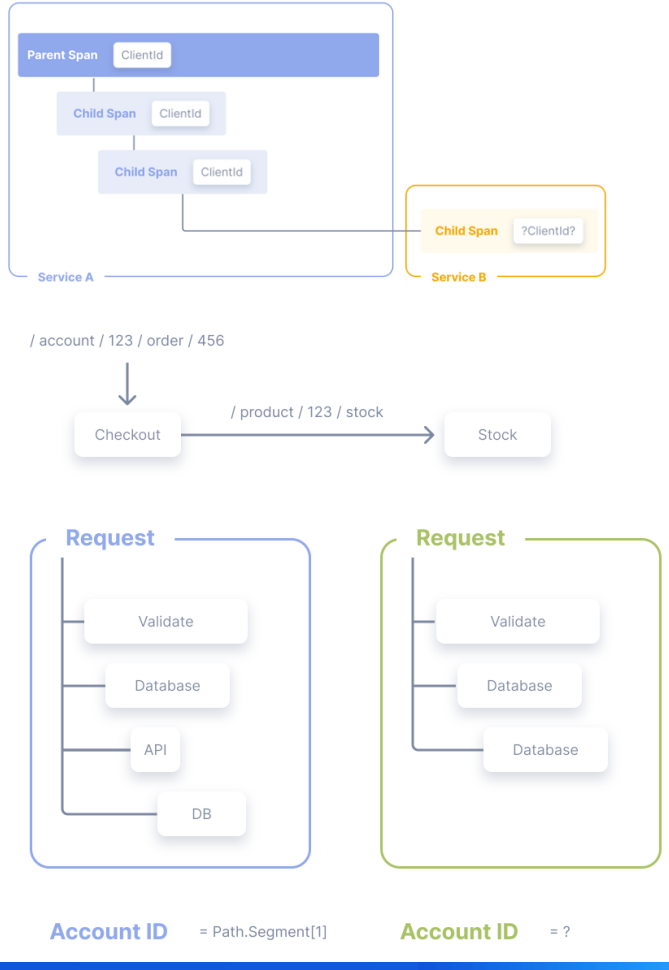
- Middleware for tracing pipelines
- Runs on creation / dispose of the activity / log
  - Performance critical
  - Do not call external services, otherwise dispose of activity is waiting.
- Adds additional context
  - Examples: Add tenant id as tag, provide more context information from the session, ...

```
1   // Copyright The OpenTelemetry Authors
2   // SPDX-License-Identifier: Apache-2.0
3
4   using System.Diagnostics;
5   using OpenTelemetry;
6
7   internal class MyEnrichingProcessor : BaseProcessor<Activity>
8   {
9       public override void OnEnd(Activity activity)
10      {
11          // Enrich activity with additional tags.
12          activity.SetTag("myCustomTag", "myCustomTagValue");
13
14          // Enriching from Baggage.
15          // The below snippet adds every Baggage item.
16          foreach (var baggage in Baggage.GetBaggage())
17          {
18              activity.SetTag(baggage.Key, baggage.Value);
19          }
20
21          // The below snippet adds specific Baggage item.
22          var deviceTypeFromBaggage = Baggage.GetBaggage("device.type");
23          if (deviceTypeFromBaggage != null)
24          {
25              activity.SetTag("device.type", deviceTypeFromBaggage);
26          }
27      }
28  }
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/24a13ab91c9c152d03fd0871bbb94e8f6ef08698/docs/trace/extending-the-sdk/MyEnrichingProcessor.cs

Baggage

- Additional context between services
- W3C Trace Compliant
- Dangerous
  - If we add baggage, every external call will include baggage
  - Data could be leaked to 3rd parties
  - Use it wisely
- Not the same as attributes
  - It is a separate key-value store and is unassociated with attributes on spans, metrics, or logs without explicitly adding them.



Source: https://opentelemetry.io/docs/concepts/signals/baggage/

Propagation

- Propagation across process boundaris
- HTTP and gRPC will do it out-of-the-box
- Not only HTTP
- Trace context and Baggage

```
35      public string SendMessage()
36      {
37          try
38          {
39              // Start an activity with a name following the semantic convention of the OpenTelemetry messaging specification.
40              // https://github.com/open-telemetry/semantic-conventions/blob/main/docs/messaging/messaging-spans.md#span-name
41              var activityName = $"{RabbitMqHelper.TestQueueName} send";
42
43              using var activity = ActivitySource.StartActivity(activityName, ActivityKind.Producer);
44              var props = this.channel.CreateBasicProperties();
45
46              // Depending on Sampling (and whether a listener is registered or not), the
47              // activity above may not be created.
48              // If it is created, then propagate its context.
49              // If it is not created, the propagate the Current context,
50              // if any.
51              ActivityContext contextToInject = default;
52              if (activity != null)
53              {
54                  contextToInject = activity.Context;
55              }
56              else if (Activity.Current != null)
57              {
58                  contextToInject = Activity.Current.Context;
59              }
60
61              // Inject the ActivityContext into the message headers to propagate trace context to the receiving service.
62              Propagator.Inject(new PropagationContext(contextToInject, Baggage.Current), props, this.InjectTraceContextIntoBasicProperties);
63
64              // The OpenTelemetry messaging specification defines a number of attributes. These attributes are added here.
65              RabbitMqHelper.AddMessagingTags(activity);
66              var body = $"Published message: DateTime.Now = {DateTime.Now}.";
67
68              this.channel.BasicPublish(
69                  exchange: RabbitMqHelper.DefaultExchangeName,
70                  routingKey: RabbitMqHelper.TestQueueName,
71                  basicProperties: props,
72                  body: Encoding.UTF8.GetBytes(body));
73
74              this.logger.MessageSent(body);
75
76              return body;
77          }
78          catch (Exception ex)
79          {
80              this.logger.MessagePublishingFailed(ex);
81              throw;
82          }
83      }
```

Source: https://github.com/open-telemetry/opentelemetry-dotnet/blob/24a13ab91c9c152d03fd0871bbb94e8f6ef08698/examples/MicroserviceExample/Utils/Messaging/MessageSender.cs#L16

Sampling Strategies

Strategy	Decision point	Typical use	Collector block
Always-on	SDK (head)	Dev & low-traffic	sampler: always_on
Probabilistic (e.g., 10 %)	SDK (head)	Prod baseline	sampler: traceidratio 0.1
Tail rule-based	Collector	High traffic, error focus	tail_sampling { policies: [...] }
Dynamic (rate-limiting)	Collector	Cost cap	ratelimit processor


DEMO
Status, Tags, Baggage


OTEL Collector
4tecture
empower your software solutions

OTEL Collector

- Dedicated service running in the cluster
- Centralized configuration
- Centralized egress
- Filtering and redaction
- Enrichment (i.e Pod information)

OTEL Collector

Source: https://opentelemetry.io/docs/collector/


DEMO
OTEL Collector


Conclusion
4tecture
empower your software solutions

Conclusion

- ## Standardize Your Telemetry
  - Adopt OpenTelemetry signals (traces, metrics, logs) with semantic conventions to ensure consistency and vendor flexibility.
- ## Leverage .NET Ergonomics
  - Use ActivitySource/Activity for spans, ILogger for logs, and Meter for metrics to minimize dependencies and simplify instrumentation.
- ## Balance Context & Performance
  - Apply resource-level attributes for static context, use processors judiciously, and be cautious with baggage to avoid overhead or data leakage.
- ## Embrace the Collector
  - Centralize configuration, enrichment, filtering, and sampling in the OpenTelemetry Collector for scalable, secure observability.


Q & A
4tecture
empower your software solutions

Thank you for your attention!

If you have any questions do not hesitate to contact us:

4tecture GmbH
Industriestrasse 25
CH-8604 Volketswil

Marc Müller
Principal Consultant

+41 44 508 37 00
info@4tecture.ch
www.4tecture.ch

4tecture
empower your software solutions

ProAuth

ReaFx


4tecture©
empower your software solutions