
.NET Conf China
2022
线上+线下 2022.12.03~04
开源 · 安全 · 赋能

.NET Conf China
2022

ASP.NET Core Apps Observability & OpenTelemetry

黄凯华
iHerb .NET 后端开发

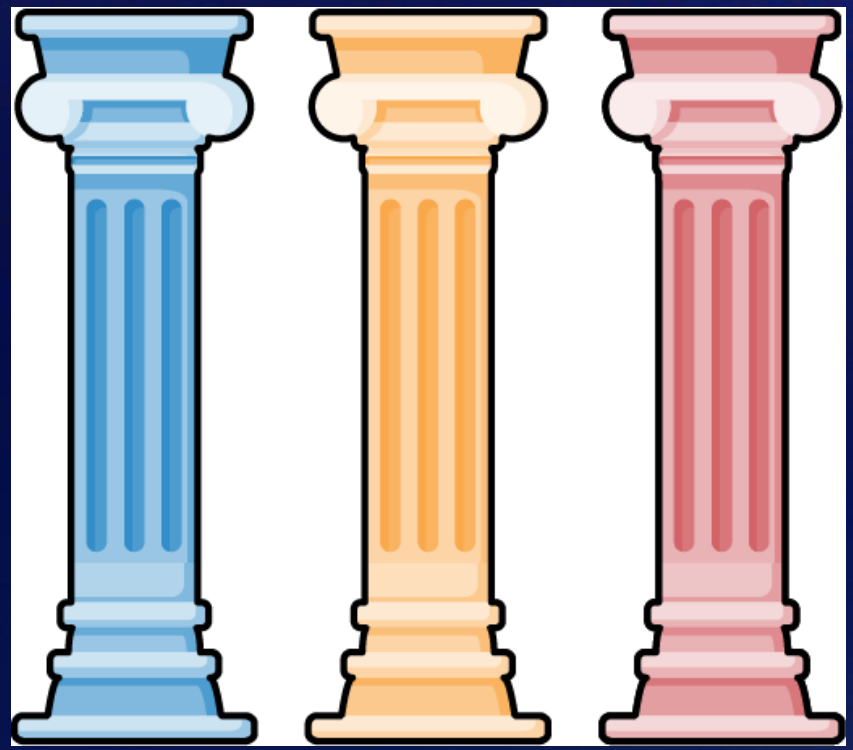
监控（Monitring）

为了保障分布式系统的正常运行，我们需要对系统有必要的监控，检查系统的可用性和进行故障排查。
我们监控的东西可以分为两大类：事务（transactions）和资源（resources）：

Logging
记录构成事务的各个事件。

Metrics
记录构成一个事务的事件的集合。

Tracing
测量操作的延迟和识别事务中的性能瓶颈，或者类似的东西。



Trace Trees & Spans

Span：可以理解成某项操作的记录，其数据结构应该足够简单，以便于能放在日志或者网络协议的报文头里；也应该足够完备，起码要含有时间戳、起止时间、Trace 的 ID、当前 Span 的 ID、父 Span 的 ID 等能够满足追踪需要的信息。

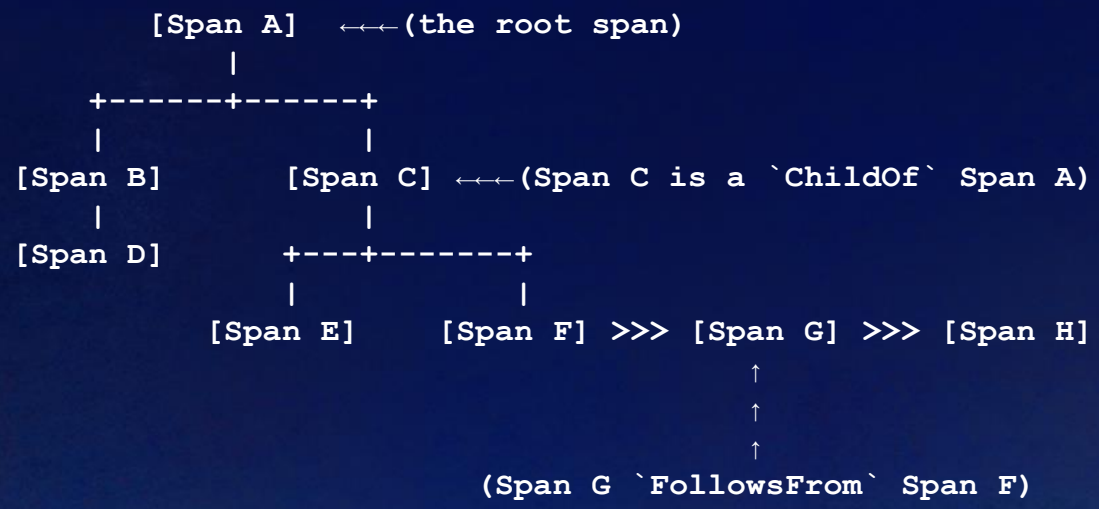
Trace：由若干个有顺序、有层级关系的 Span 所组成一颗“追踪树”（Trace Tree）。

Span:

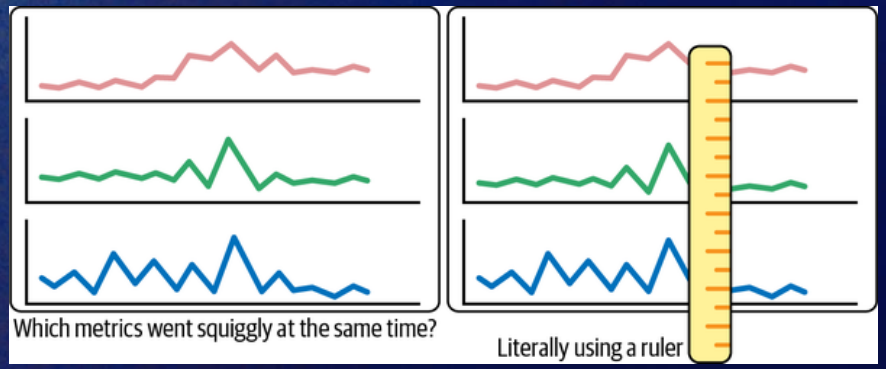
- Operation name
- Start timestamp
- Finish timestamp
- Current span ID
- parent span ID
- Trace ID
- Key:value Span Tags.

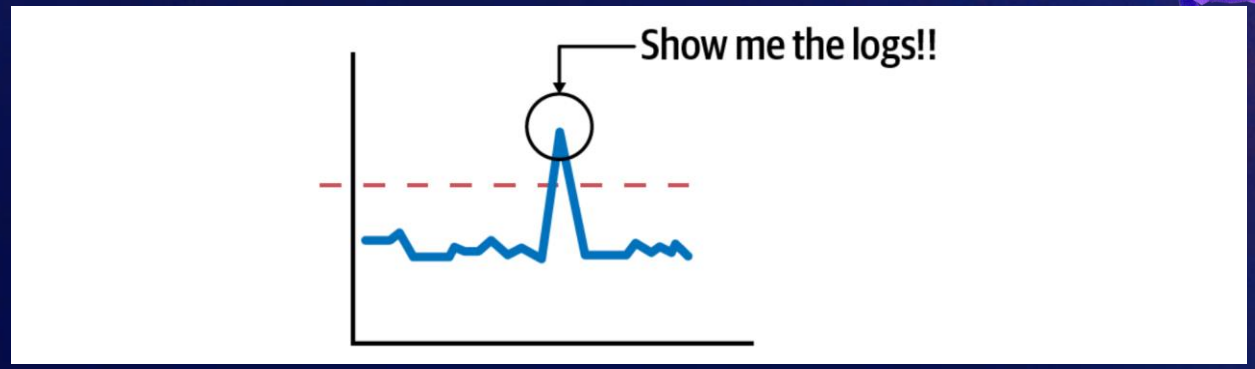
…

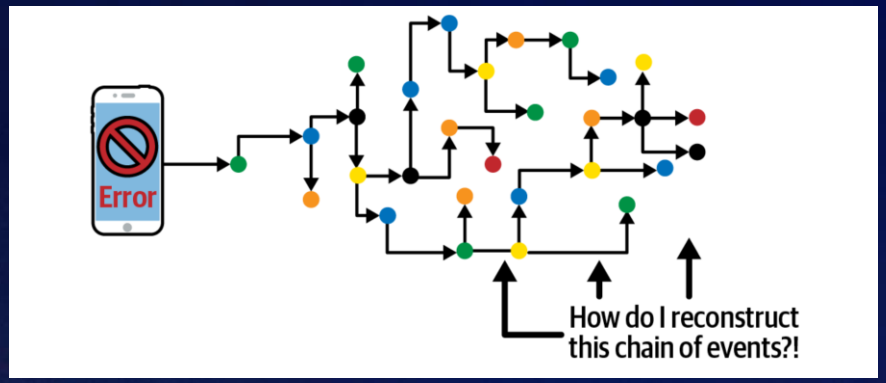
一个 trace 中 span 之间的关联关系



Observability = Logging + Metrics + Tracing ?







可观测性（Observability）

概念来自于控制理论，指系统可以由其外部输出推断其内部状态的程度。
https://zh.m.wikipedia.org/zh-cn/可觀測性

可观测性可以帮助我们了解系统

- 正在发生什么？
- 为什么会这样？

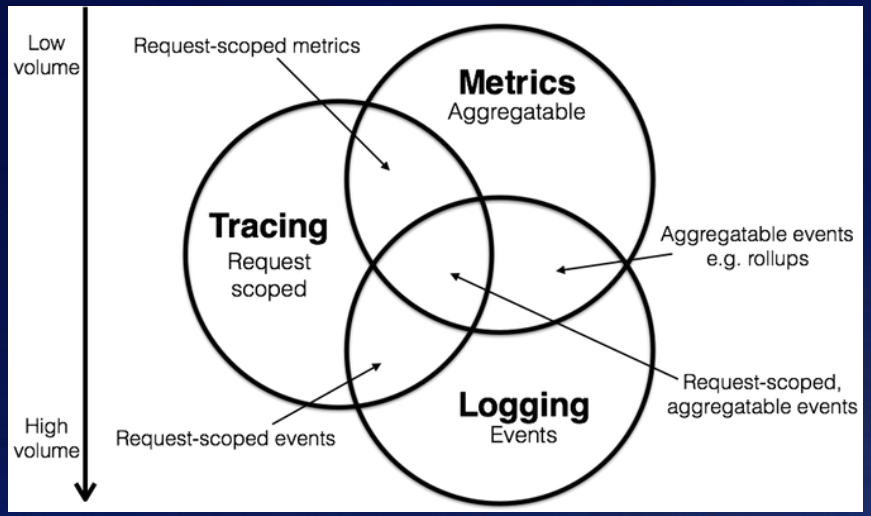
监控 VS 可观测性

Known Unknowns VS Unknown Unknowns

通过可观察性，比较**高维**和**高基数**的数据，我们能够在复杂系统架构中的发现隐藏问题。

维度（Dimensionality）：数据中键（key）的数量。

基数（Cardinality）：包含在一个集合中的唯一值的数量。低基数意味着这一列在其集合中有很多重复的值；高基数意味着该列包含很大比例的完全唯一的值。



OpenTelemetry

简称 **OTel**，包含一整套的工具以及 API 和 SDK。

- Specification
- API / SDK
- Collector

OpenTelemetry Specification

OpenTelemetry 定义了一套跨语言的规范
https://opentelemetry.io/docs/reference/specification/

- API Specification
  - Context
    - Propagators
  - Baggage
  - Tracing
  - Metrics
- SDK Specification
  - Tracing
  - Metrics
  - Resource
  - Configuration
- Data Specification
  - Semantic Conventions
  - Protocol
    - Metrics
    - Logs

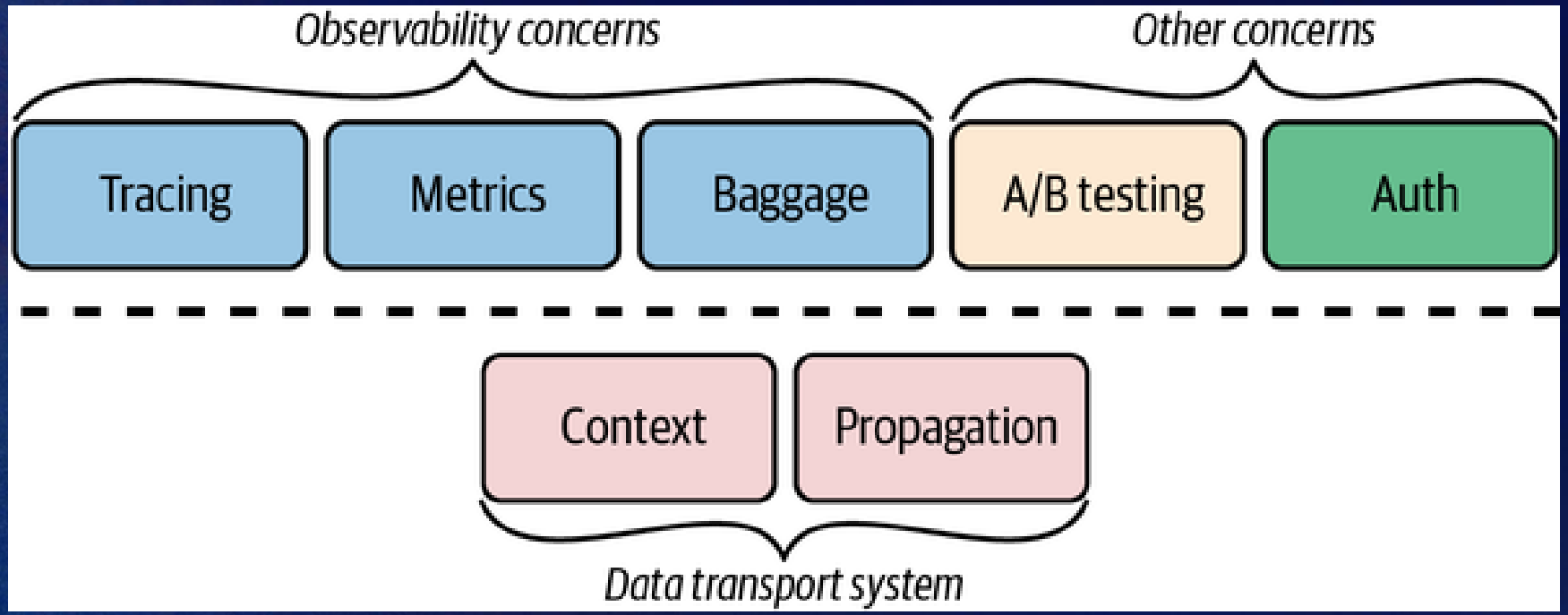
Specification: Signals

OpenTelemetry specification 为 telemetry 数据定义了不同的类型，称之为各种 Signal。

- Traces
- Metrics
- Logs
- Baggage

Specification: Context & Propagators

传播器利用 Context 为每个横切关注点注入和提取数据，例如 traces 和 Baggage。



Specification: OpenTelemetry Protocol

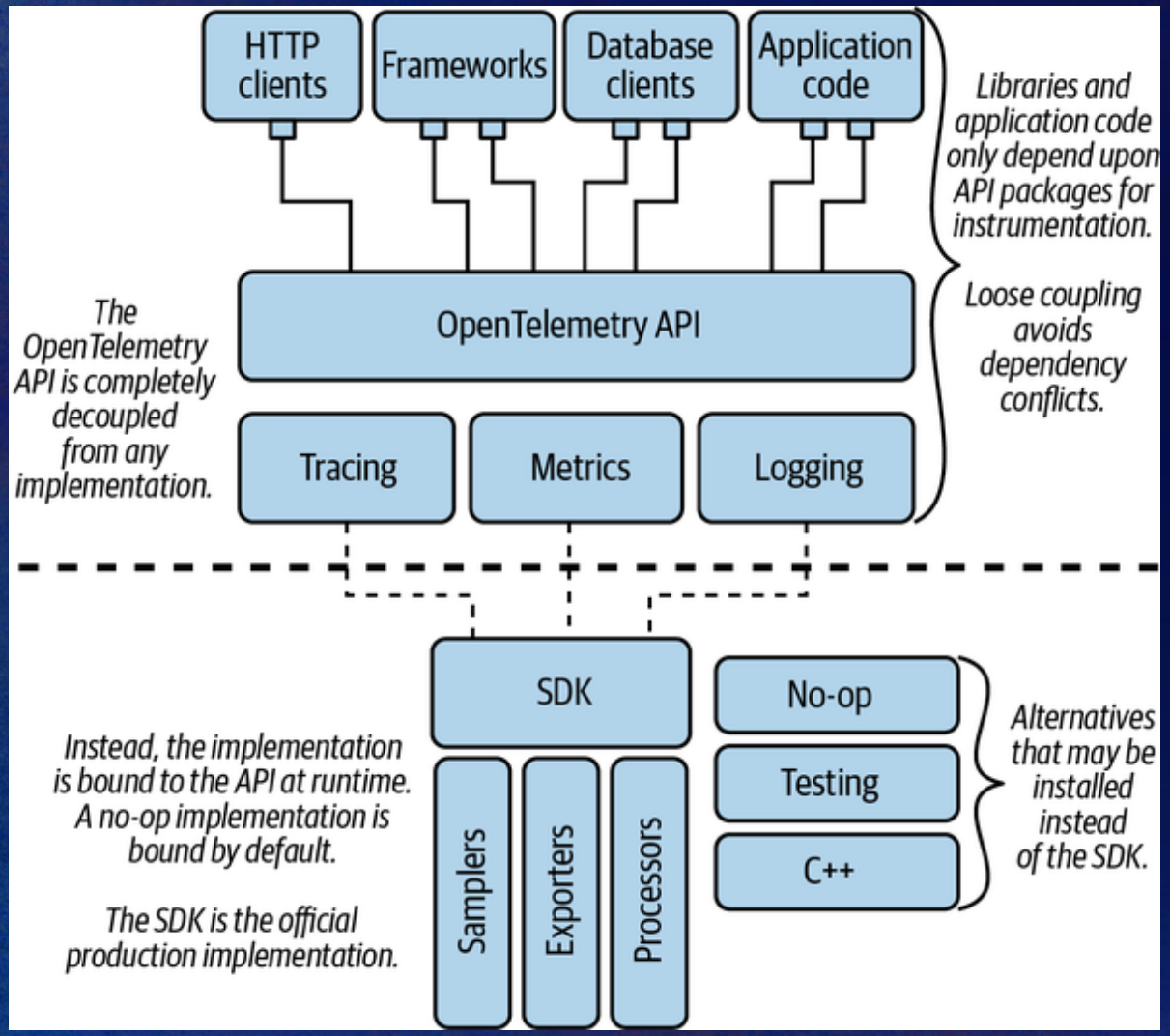
OTLP 是一套 grpc 协议，用来实现 sdk 和可观测性后端的数据交互。
https://github.com/open-telemetry/opentelemetry-proto

main / opentelemetry-proto / opentelemetry / proto / Go to file

jmacd Change the exponential histogram boundary condition (#409) ✓ 724e427 on 19 Aug History

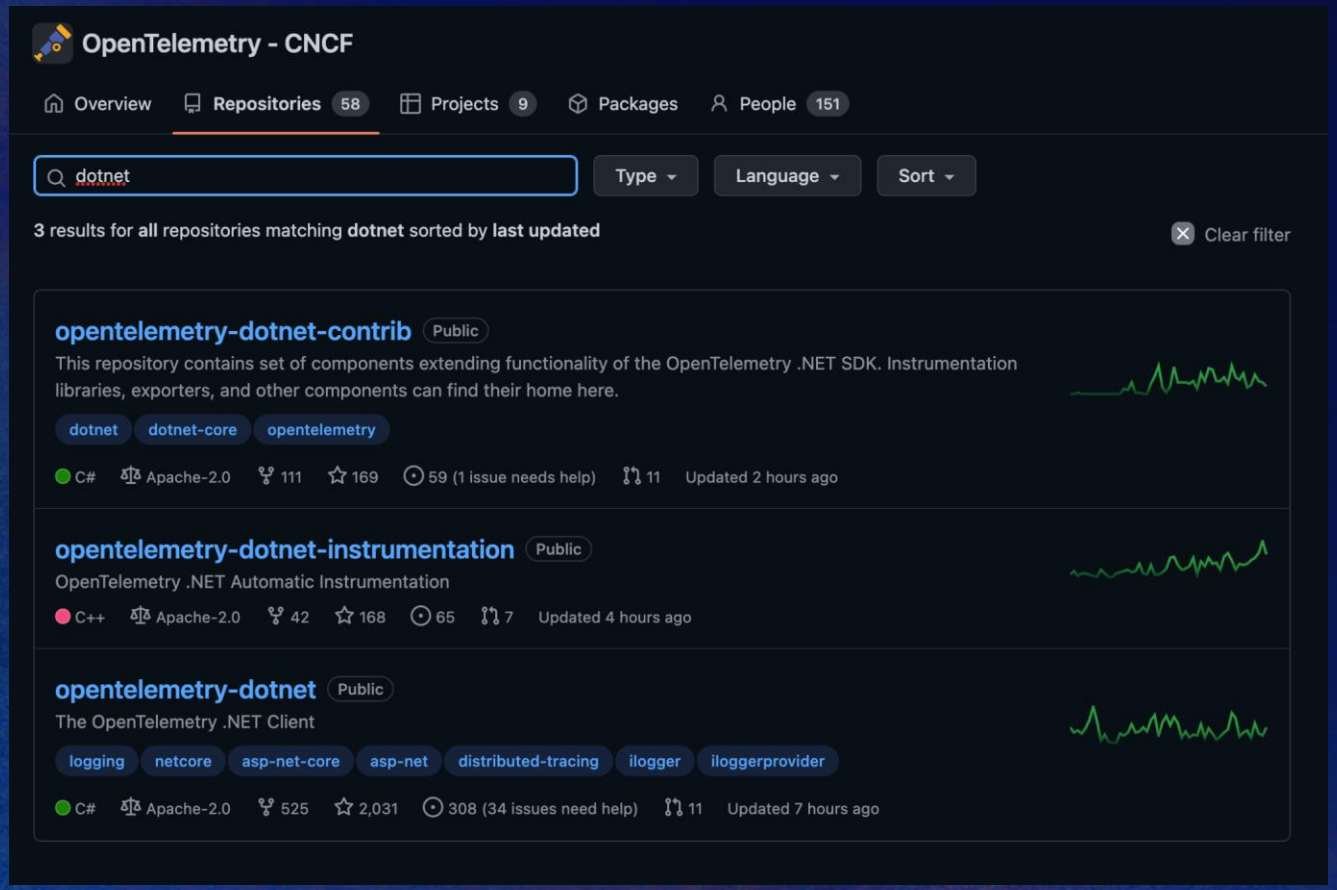
..		
collector	Make it possible to indicate partial success in an OTLP export respon...	2 months ago
common/v1	Delete deprecated InstrumentationLibrary-related fields and messages (#...	3 months ago
logs/v1	Delete deprecated InstrumentationLibrary-related fields and messages (#...	3 months ago
metrics/v1	Change the exponential histogram boundary condition (#409)	2 months ago
resource/v1	Define csharp_namespace option (#399)	4 months ago
trace/v1	fix a tiny typo and reflow (#421)	2 months ago

API / SDK: Client Architecture



SDK

- **Core**：手动给应用创建指标（Instrumentation）的 API/SDK 的实现。
- **Instrumentation**：Core 的功能加强，提供了一套自动给应用创建 Instrumentation 的类库和框架。
- **Contrib**: 可选的组件。 Instrumentation 类库, exporters, 及其他组件。



OpenTelemetry Trace API VS .NET Activity API

TracerProvider = new ActivitySource

OpenTelemetry API	Activity API
var tracer = TracerProvider.Default.GetTracer("name", "version")	var activitySource = new ActivitySource("name", "version")

Tracer = ActivitySource

OpenTelemetry API	Activity API	Comments
tracer.StartActiveSpan()	activitySource.StartActivity()	
Tracer.Current	Activity.Current	Note that the .Current is obtained from Activity, not ActivitySource.
tracer.StartSpan		No equivalent. Starting an activity always makes it as the current.

OpenTelemetry Trace API VS .NET Activity API

Span = Activity

OpenTelemetry API	**Activity API**
OpenTelemetry.Trace.TelemetrySpan	System.Diagnostics.Activity
span.Name	activity.DisplayName
span.SpanContext	activity.ActivityContext
span.ParentSpanContext	activity.Parent.ActivityContext
span.ParentSpan	activity.Parent
span.ParentSpanId	activity.ParentSpanId
span.SpanKind	activity.ActivityKind
span.SetAttribute	activity.SetTag
span.AddEvent	activity.AddEvent
span.Links	activity.Links
span.StartTime	activity.StartTimeUtc
span.EndTime	activity.StartTimeUtc + Activity.Duration
span.IsRecording	activity.IsAllDataRequested

OpenTelemetry Trace API VS .NET Activity API

SpanKind = ActivityKind

github.com/open-telemetry/opentelemetry-proto/blob/main/opentelemetry/proto/trace/v1/trace.proto

```
122
123   // SpanKind is the type of span. Can be used to specify additional relationships between spans
124   // in addition to a parent/child relationship.
125   enum SpanKind {
126     // Unspecified. Do NOT use as default.
127     // Implementations MAY assume SpanKind to be INTERNAL when receiving UNSPECIFIED.
128     SPAN_KIND_UNSPECIFIED = 0;
129
130     // Indicates that the span represents an internal operation within an application,
131     // as opposed to an operation happening at the boundaries. Default value.
132     SPAN_KIND_INTERNAL = 1;
133
134     // Indicates that the span covers server-side handling of an RPC or other
135     // remote network request.
136     SPAN_KIND_SERVER = 2;
137
138     // Indicates that the span describes a request to some remote service.
139     SPAN_KIND_CLIENT = 3;
140
141     // Indicates that the span describes a producer sending a message to a broker.
142     // Unlike CLIENT and SERVER, there is often no direct critical path latency relationship
143     // between producer and consumer spans. A PRODUCER span ends when the message was accepted
144     // by the broker while the logical processing of the message might span a much longer time.
145     SPAN_KIND_PRODUCER = 4;
146
147     // Indicates that the span describes consumer receiving a message from a broker.
148     // Like the PRODUCER kind, there is often no direct critical path latency relationship
149     // between producer and consumer spans.
150     SPAN_KIND_CONSUMER = 5;
151   }
152
```

github.com/dotnet/runtime/blob/main/src/libraries/System.Diagnostics.DiagnosticSource/src/System/Diagnostics/ActivityKind.cs

46 lines (41 sloc) | 1.85 KB Raw Blame

```
1  // Licensed to the .NET Foundation under one or more agreements.
2  // The .NET Foundation licenses this file to you under the MIT license.
3
4  namespace System.Diagnostics
5  {
6      /// <summary>
7      /// Kind describes the relationship between the Activity, its parents, and its children in a Trace.
8      ///     ------------------------------------------------------------------------------------
9      ///    ActivityKind    Synchronous  Asynchronous    Remote Incoming    Remote Outgoing
10     ///     ------------------------------------------------------------------------------------
11     ///       Internal
12     ///       Client          yes                                          yes
13     ///       Server          yes                             yes
14     ///       Producer                    yes                              maybe
15     ///       Consumer                    yes             maybe
16     ///     ------------------------------------------------------------------------------------
17     /// </summary>
18     public enum ActivityKind
19     {
20         /// <summary>
21         /// Default value.
22         /// Indicates that the Activity represents an internal operation within an application, as opposed to an operations with remote parents or children.
23         /// </summary>
24         Internal = 0,
25
26         /// <summary>
27         /// Server activity represents request incoming from external component.
28         /// </summary>
29         Server = 1,
30
31         /// <summary>
32         /// Client activity represents outgoing request to the external component.
33         /// </summary>
34         Client = 2,
35
36         /// <summary>
37         /// Producer activity represents output provided to external components.
38         /// </summary>
39         Producer = 3,
40
41         /// <summary>
42         /// Consumer activity represents output received from an external component.
43         /// </summary>
44         Consumer = 4,
45     }
46 }
```

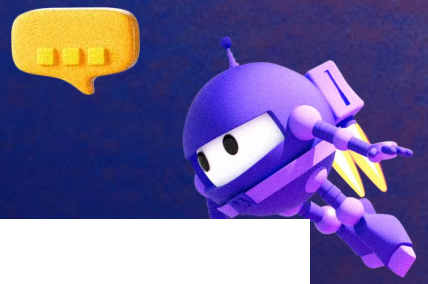
Manually Instrument: Traces

https://github.com/open-telemetry/opentelemetry-dotnet/blob/main/src/OpenTelemetry/README.md - tracing-configuration

dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Exporter.Console

```
var serviceName = "MyCompany.MyProduct.MyService";
var serviceVersion = "1.0.0";

var resourceBuilder = ResourceBuilder.CreateDefault()
   .AddService(serviceName: serviceName, serviceVersion: serviceVersion);

using var tracerProvider = Sdk.CreateTracerProviderBuilder()
   .AddSource("ActivitySource1")
   .AddSource("ActivitySource2")
   .SetResourceBuilder(resourceBuilder)
   .AddConsoleExporter()
   .Build();

var activitySource1 = new ActivitySource("ActivitySource1");
var activitySource2 = new ActivitySource("ActivitySource2");
```

Manually Instrument: Traces



```
using (var activity1 = activitySource1.StartActivity("Activity1"))
{

  activity1?.SetTag("foo", 1);
  activity1?.SetTag("bar", "Hello, World!");

  using (var activity2 = activitySource2.StartActivity("Activity2"))
  {
     activity2?.SetTag("foo", 2);
     activity2?.SetTag("bar", "Hello, OpenTelemetry!");

     Debug.Assert(activity2?.ParentId == activity1?.Id);
  }
}
```

```
Activity.TraceId:          c94fd57bf0fe9424c2965f338fc399ec
Activity.SpanId:           003ab102249a5c5f
Activity.TraceFlags:           Recorded
Activity.ParentSpanId:     53342a6af70ff1be
Activity.ActivitySourceName: ActivitySource2
Activity.DisplayName: Activity2
Activity.Kind:         Internal
Activity.StartTime:    2022-10-07T10:39:01.2177110Z
Activity.Duration:     00:00:00.0001000
Activity.Tags:
    foo: 2
    bar: Hello, OpenTelemetry!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: b276abd8-5e65-4d07-9ce2-6cb9a954a0e0


Activity.TraceId:          c94fd57bf0fe9424c2965f338fc399ec
Activity.SpanId:           53342a6af70ff1be
Activity.TraceFlags:           Recorded
Activity.ActivitySourceName: ActivitySource1
Activity.DisplayName: Activity1
Activity.Kind:         Internal
Activity.StartTime:    2022-10-07T10:39:01.2162810Z
Activity.Duration:     00:00:00.0417300
Activity.Tags:
    foo: 1
    bar: Hello, World!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: b276abd8-5e65-4d07-9ce2-6cb9a954a0e0
```

Manually Instrument: Logging

```
var serviceName = "MyCompany.MyProduct.MyService";
var serviceVersion = "1.0.0";

var resourceBuilder = ResourceBuilder.CreateDefault()
   .AddService(serviceName: serviceName, serviceVersion: serviceVersion);

using var loggerFactory = LoggerFactory.Create(
  builder => builder.AddOpenTelemetry(
      options =>
      {
          options.AddConsoleExporter();
          options.SetResourceBuilder(resourceBuilder);
      }));

var logger = loggerFactory.CreateLogger("MyLogger");
```

Manually Instrument: Logging

```
var activitySource1 = new ActivitySource("ActivitySource1");
var activitySource2 = new ActivitySource("ActivitySource2");

using (var activity1 = activitySource1.StartActivity("Activity1"))
{
  activity1?.SetTag("foo", 1);
  activity1?.SetTag("bar", "Hello, World!");

  logger.LogInformation("Hello from {ActivitySource}!", activitySource1.Name);

  using (var activity2 = activitySource2.StartActivity("Activity2"))
  {
    activity2?.SetTag("foo", 2);
    activity2?.SetTag("bar", "Hello, OpenTelemetry!");

    Debug.Assert(activity2?.ParentId == activity1?.Id);

    logger.LogInformation("Hello from {ActivitySource}!", activitySource2.Name);
  }
}
```

Manually Instrument: Logging

```
Activity.TraceId:          478d21e1307305397b0f0a6e9e6e35af
Activity.SpanId:           1be65cc75b8118d1
Activity.TraceFlags:       Recorded
Activity.ParentSpanId:     ade5623f46664cc1
Activity.ActivitySourceName: ActivitySource2
Activity.DisplayName: Activity2
Activity.Kind:        Internal
Activity.StartTime:   2022-10-07T13:07:30.3628620Z
Activity.Duration:    00:00:00.0001920
Activity.Tags:
    foo: 2
    bar: Hello, OpenTelemetry!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: 965600d9-4729-4a8c-b411-2e3a67f511db

Activity.TraceId:          478d21e1307305397b0f0a6e9e6e35af
Activity.SpanId:           ade5623f46664cc1
Activity.TraceFlags:       Recorded
Activity.ActivitySourceName: ActivitySource1
Activity.DisplayName: Activity1
Activity.Kind:        Internal
Activity.StartTime:   2022-10-07T13:07:30.3227010Z
Activity.Duration:    00:00:00.0494030
Activity.Tags:
    foo: 1
    bar: Hello, World!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: 965600d9-4729-4a8c-b411-2e3a67f511db
```

```
LogRecord.Timestamp:       2022-10-07T13:07:30.3310890Z
LogRecord.TraceId:         478d21e1307305397b0f0a6e9e6e35af
LogRecord.SpanId:          ade5623f46664cc1
LogRecord.TraceFlags:      Recorded
LogRecord.CategoryName:    MyLogger
LogRecord.LogLevel:        Information
LogRecord.State:           Hello from ActivitySource1!

Resource associated with LogRecord:
service.name: MyCompany.MyProduct.MyService
service.version: 1.0.0
service.instance.id: 965600d9-4729-4a8c-b411-2e3a67f511db

LogRecord.Timestamp:       2022-10-07T13:07:30.3629530Z
LogRecord.TraceId:         478d21e1307305397b0f0a6e9e6e35af
LogRecord.SpanId:          1be65cc75b8118d1
LogRecord.TraceFlags:      Recorded
LogRecord.CategoryName:    MyLogger
LogRecord.LogLevel:        Information
LogRecord.State:           Hello from ActivitySource2!

Resource associated with LogRecord:
service.name: MyCompany.MyProduct.MyService
service.version: 1.0.0
service.instance.id: 965600d9-4729-4a8c-b411-2e3a67f511db
```

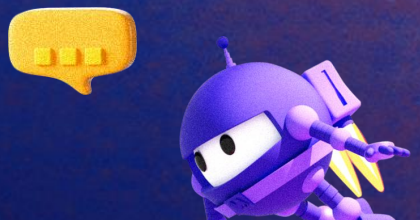
Manually Instrument: Metrics

```
var serviceName = "MyCompany.MyProduct.MyService";
var serviceVersion = "1.0.0";

var resourceBuilder = ResourceBuilder.CreateDefault()
   .AddService(serviceName: serviceName, serviceVersion: serviceVersion);

using MeterProvider meterProvider = Sdk.CreateMeterProviderBuilder()
   .AddMeter("Meter1")
   .SetResourceBuilder(resourceBuilder)
   .AddConsoleExporter()
   .Build();

var meter = new Meter(name: "Meter1", version: "1.0.0");
```

Manually Instrument: Metrics

```
using (var activity1 = activitySource1.StartActivity("Activity1"))
{
    activity1?.SetTag("foo", 1);
    activity1?.SetTag("bar", "Hello, World!");

    logger.LogInformation("Hello from {ActivitySource}!", activitySource1.Name);

    var counter = meter.CreateCounter<int>("counter");

    counter.Add(1);
    counter.Add(2);

    using (var activity2 = activitySource2.StartActivity("Activity2"))
    {
        activity2?.SetTag("foo", 2);
        activity2?.SetTag("bar", "Hello, OpenTelemetry!");

        Debug.Assert(activity2?.ParentId == activity1?.Id);

        logger.LogInformation("Hello from {ActivitySource}!", activitySource2.Name);
    }
}
```

Manually Instrument: Metrics



```
Activity.TraceId:          a1545bce5e56994facda3ee1a320a592
Activity.SpanId:           4edc67c2a5e78902
Activity.TraceFlags:           Recorded
Activity.ParentSpanId:     e701c130096a5686
Activity.ActivitySourceName: ActivitySource2
Activity.DisplayName: Activity2
Activity.Kind:         Internal
Activity.StartTime:    2022-10-07T13:35:00.1217020Z
Activity.Duration:     00:00:00.0003150
Activity.Tags:
    foo: 2
    bar: Hello, OpenTelemetry!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: a574c20a-215c-4ed9-b371-4a69dcb8a022

Activity.TraceId:          a1545bce5e56994facda3ee1a320a592
Activity.SpanId:           e701c130096a5686
Activity.TraceFlags:           Recorded
Activity.ActivitySourceName: ActivitySource1
Activity.DisplayName: Activity1
Activity.Kind:         Internal
Activity.StartTime:    2022-10-07T13:34:59.8851770Z
Activity.Duration:     00:00:00.2454450
Activity.Tags:
    foo: 1
    bar: Hello, World!
Resource associated with Activity:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: a574c20a-215c-4ed9-b371-4a69dcb8a022
```

```
Resource associated with Metric:
    service.name: MyCompany.MyProduct.MyService
    service.version: 1.0.0
    service.instance.id: a574c20a-215c-4ed9-b371-4a69dcb8a022

Export counter, Meter: Meter1/1.0.0
(2022-10-07T13:35:00.1138830Z, 2022-10-07T13:35:00.1361700Z]
LongSum
Value: 3
```

Manually Instrument: Metrics

https://github.com/open-telemetry/opentelemetry-proto/blob/main/opentelemetry/proto/metrics/v1/metrics.proto - L635

```
631  // A representation of an exemplar, which is a sample input measurement.
632  // Exemplars also hold information about the environment when the measurement
633  // was recorded, for example the span and trace ID of the active span when the
634  // exemplar was recorded.
635  message Exemplar {
636    reserved 1;
637
638    // The set of key/value pairs that were filtered out by the aggregator, but
639    // recorded alongside the original measurement. Only key/value pairs that were
640    // filtered out by the aggregator should be included
641    repeated opentelemetry.proto.common.v1.KeyValue filtered_attributes = 7;
642
643    // time_unix_nano is the exact time when this exemplar was recorded
644    //
645    // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
646    // 1970.
647    fixed64 time_unix_nano = 2;
648
649    // The value of the measurement that was recorded. An exemplar is
650    // considered invalid when one of the recognized value fields is not present
651    // inside this oneof.
652    oneof value {
653      double as_double = 3;
654      sfixed64 as_int = 6;
655    }
656
657    // (Optional) Span ID of the exemplar trace.
658    // span_id may be missing if the measurement is not recorded inside a trace
659    // or if the trace is not sampled.
660    bytes span_id = 4;
661
662    // (Optional) Trace ID of the exemplar trace.
663    // trace_id may be missing if the measurement is not recorded inside a trace
664    // or if the trace is not sampled.
665    bytes trace_id = 5;
666  }
```

Manually Instrument: Metrics

https://opentelemetry.io/docs/reference/specification/metrics/sdk/#exemplar

Exemplar

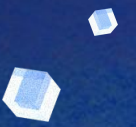
Status: Feature-freeze

Exemplars are example data points for aggregated data. They provide specific context to otherwise general aggregations. Exemplars allow correlation between aggregated metric data and the original API calls where measurements are recorded. Exemplars work for trace-metric correlation across any metric, not just those that can also be derived from `Span`s.

An Exemplar is a recorded Measurement that exposes the following pieces of information:

- The `value` of the `Measurement` that was recorded by the API call.
- The `time` the API call was made to record a `Measurement`.
- The set of Attributes associated with the `Measurement` not already included in a metric data point.
- The associated trace id and span id of the active Span within Context of the `Measurement` at API call time.

For example, if a user has configured a `View` to preserve the attributes: `X` and `Y`, but the user records a measurement as follows:



集成到 ASP.NET Core 应用

- **OpenTelemetry**
- **OpenTelemetry.Extensions.Hosting**
- **OpenTelemetry.Instrumentation.AspNetCore**

- OpenTelemetry.Instrumentation.Http
- OpenTelemetry.Exporter.Jaeger
- OpenTelemetry.Exporter.Console
- OpenTelemetry.Exporter.Prometheus.AspNetCore

集成到 ASP.NET Core 应用

```
var builder = WebApplication.CreateBuilder(args);
// ...
var resourceBuilder = ResourceBuilder.CreateDefault().AddService("FooService", "1.0.0");
builder.Services.AddOpenTelemetryTracing(tracerProviderBuilder =>
{
  tracerProviderBuilder
    .SetResourceBuilder(resourceBuilder)
    .AddSource("FooSource")
    .AddAspNetCoreInstrumentation()
    // 如果想把 baggage 传给下游服务的话 HttpClient instrumentation 是必须得
    .AddHttpClientInstrumentation()
    .AddJaegerExporter(options =>
    {
      options.Protocol = JaegerExportProtocol.HttpBinaryThrift;
      options.AgentHost = "localhost";
      options.AgentPort = 14268;
    });
});
builder.Services.AddLogging(loggingBuilder => loggingBuilder
  .AddOpenTelemetry(
    options =>
    {
      options.SetResourceBuilder(resourceBuilder);
      options.AddConsoleExporter();
    }));
```

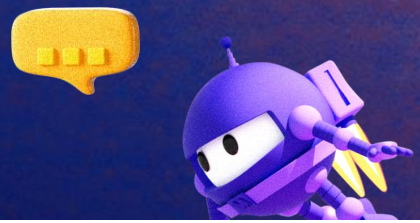
集成到 ASP.NET Core 应用

```
builder.Services.AddOpenTelemetryMetrics(metricsBuilder => metricsBuilder
    .SetResourceBuilder(resourceBuilder)
    .AddAspNetCoreInstrumentation()
    .AddMeter("FooMeter")
    .AddPrometheusExporter());

var app = builder.Build();

app.UseOpenTelemetryPrometheusScrapingEndpoint();
// ...
app.Run();
```

集成到 ASP.NET Core 应用

```
var builder = WebApplication.CreateBuilder(args);
// ...
var resourceBuilder = ResourceBuilder.CreateDefault().AddService("BarService", "1.0.0");
builder.Services.AddOpenTelemetryTracing(tracerProviderBuilder =>
{
  tracerProviderBuilder
      .SetResourceBuilder(resourceBuilder)
      .AddAspNetCoreInstrumentation()
      .AddJaegerExporter(options =>
      {
          options.Protocol = JaegerExportProtocol.HttpBinaryThrift;
          options.AgentHost = "localhost";
          options.AgentPort = 14268;
      });
});
builder.Services.AddLogging(loggingBuilder => loggingBuilder
  .AddOpenTelemetry(
      options =>
      {
          options.SetResourceBuilder(resourceBuilder);
          options.AddConsoleExporter();
      }));
var app = builder.Build();
```

集成到 ASP.NET Core 应用



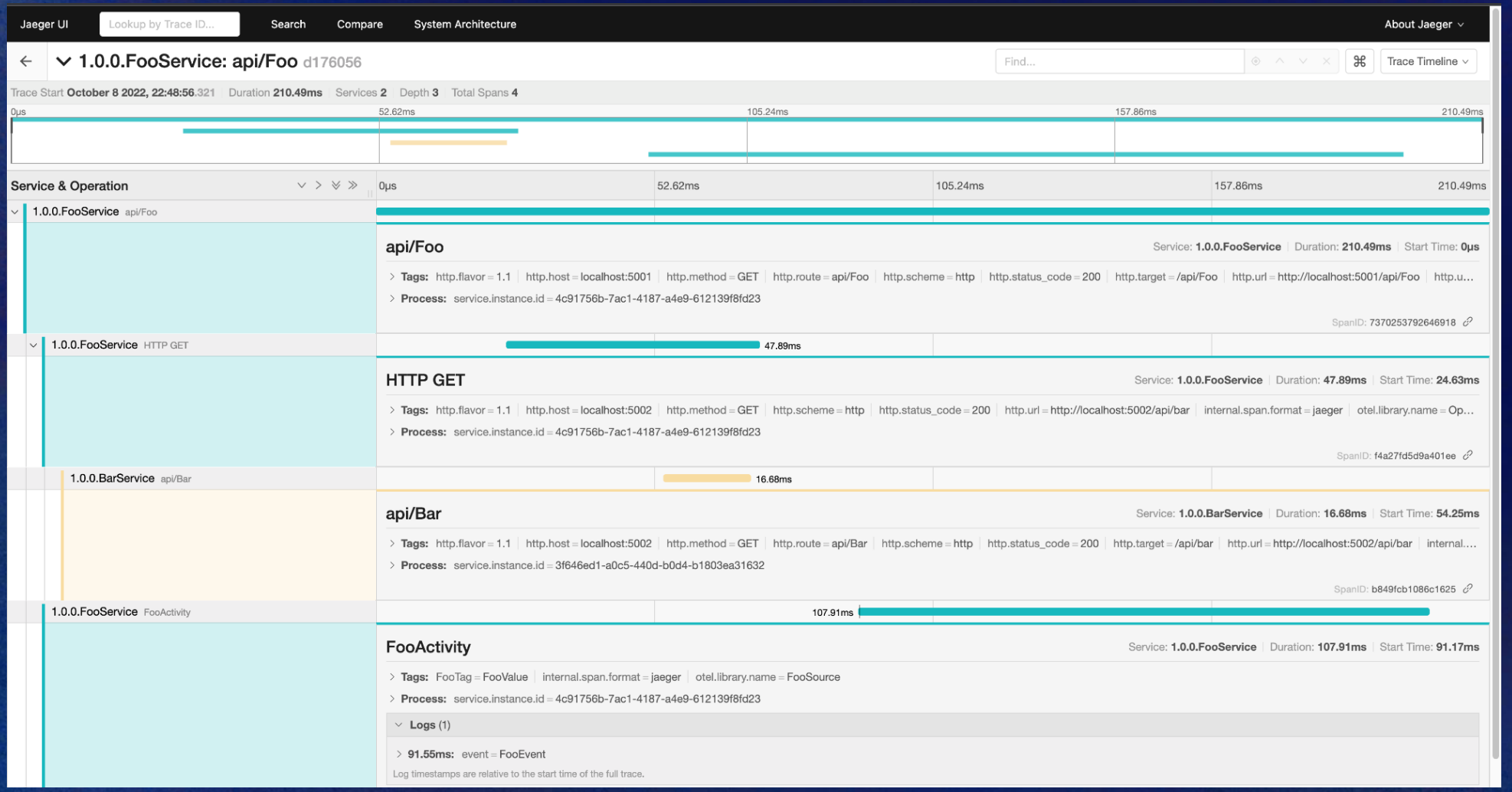
```
[Route("/api/[controller]")]
public class FooController : ControllerBase
{
  private static readonly ActivitySource FooActivitySource
    = new ActivitySource("FooSource");
  private static readonly Counter<int> FooCounter
    = new Meter("FooMeter").CreateCounter<int>("FooCounter");

  private readonly IHttpClientFactory _clientFactory;
  private readonly ILogger<FooController> _logger;

  public FooController(
    IHttpClientFactory clientFactory,
    ILogger<FooController> logger)
  {
    _clientFactory = clientFactory;
    _logger = logger;
  }
```

```
  [HttpGet]
  public async Task<IActionResult> Get()
  {
    Baggage.SetBaggage("FooBaggage1", "FooValue1");
    Baggage.SetBaggage("FooBaggage2", "FooValue2");

    var client = _clientFactory.CreateClient();
    var result = await client.GetStringAsync("http://localhost:5002/api/bar");

    using var activity = FooActivitySource.StartActivity("FooActivity");
    activity?.AddTag("FooTag", "FooValue");
    activity?.AddEvent(new ActivityEvent("FooEvent"));
    await Task.Delay(100);

    _logger.LogInformation("/api/foo called");

    FooCounter.Add(1);

    return Ok(result);
  }
}
```

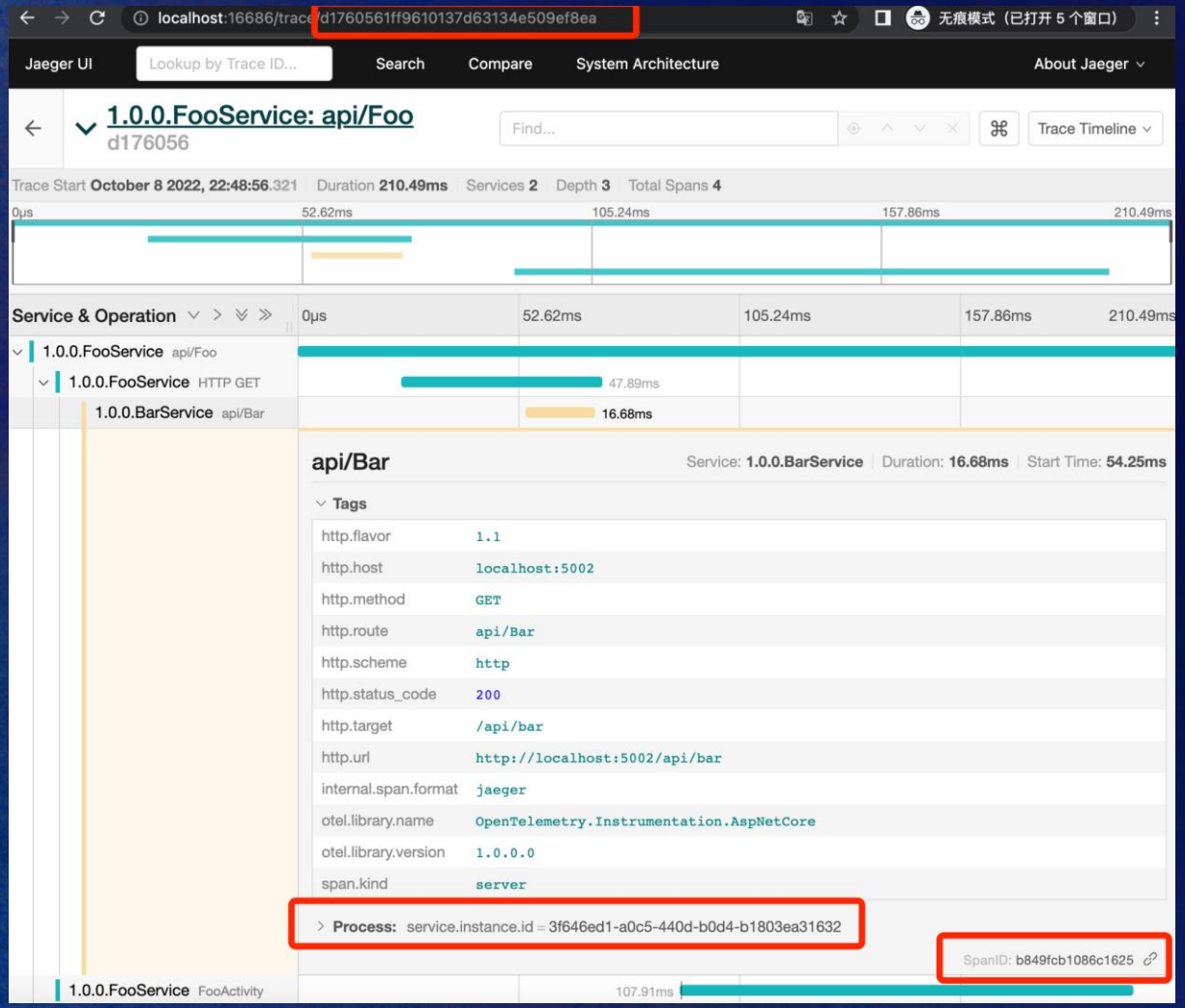
集成到 ASP.NET Core 应用

```
[Route("/api/[controller]")]
public class BarController : ControllerBase
{
    private readonly ILogger<BarController> _logger;

    public BarController(ILogger<BarController> logger)
    {
        _logger = logger;
    }

    [HttpGet]
    public string Get()
    {
        _logger.LogInformation("/api/bar called");
        var baggage1 = Baggage.GetBaggage("FooBaggage1");
        var baggage2 = Baggage.GetBaggage("FooBaggage2");
        _logger.LogInformation("baggage1 value: {baggage}, baggage2 value: {baggage2}", baggage1, baggage2);

        return "Hello from Bar";
    }
}
```

集成到 ASP.NET Core 应用



集成到 ASP.NET Core 应用



LogRecord.Timestamp: 2022-10-08T14:48:56.3911910Z
LogRecord.TraceId: d1760561ff9610137d63134e509ef8ea
LogRecord.SpanId: b849fcb1086c1625
LogRecord.TraceFlags: Recorded
LogRecord.CategoryName: WebApplication2.Controllers.BarController
LogRecord.LogLevel: Information
LogRecord.State: baggage1 value: FooValue1, baggage2 value: FooValue2

Resource associated with LogRecord:
service.name: BarService
service.namespace: 1.0.0
service.instance.id: 3f646ed1-a0c5-440d-b0d4-b1803ea31632

集成到 ASP.NET Core 应用

Expression:

```
HttpContext.Request.Headers
```

Use ⇧⌘⏎ to add to Watches

Result:

- result = HttpRequestHeaders
  - Raw View
  - Results = Expanding will force enumeration of the object
    - [0] = {KeyValuePair<string, StringValues>} [Host, localhost:5002]
    - [1] = {KeyValuePair<string, StringValues>} [baggage, FooBaggage1=FooValue1,FooBaggage2=FooValue2]
    - [2] = {KeyValuePair<string, StringValues>} [traceparent, 00-4b0231deae9608dcb308c0e200ccc42e-302b1a9ff3adb1e0-01]

集成到 ASP.NET Core 应用

localhost:5001/metrics

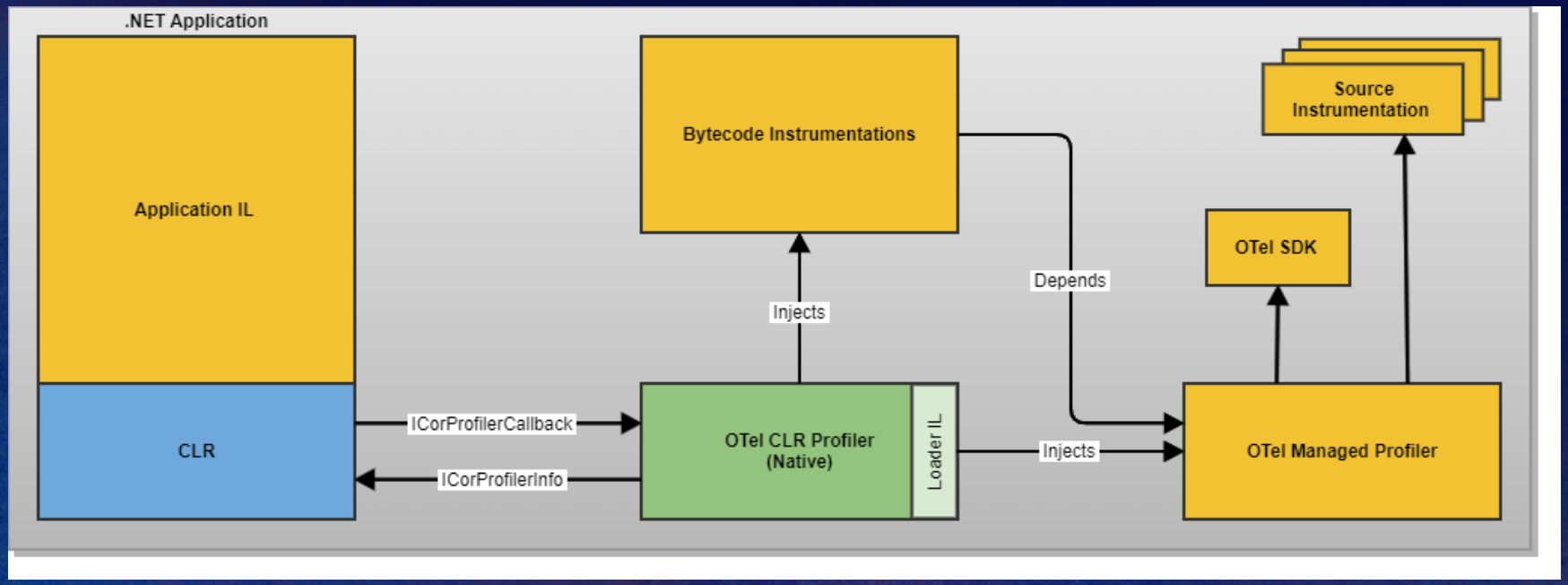
```
# TYPE http_server_duration_ms histogram
# UNIT http_server_duration_ms ms
# HELP http_server_duration_ms measures the duration of the inbound HTTP request
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="0"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="5"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="10"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="25"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="50"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="75"} 1 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="100"} 1 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="250"} 2 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="500"} 2 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="1000"} 2 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",le="+Inf"} 2 1665494157768
http_server_duration_ms_sum{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200"} 231.267 1665494157768
http_server_duration_ms_count{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200"} 2 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="0"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="5"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="10"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="25"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="50"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="75"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="100"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="250"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="500"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="1000"} 1 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo",le="+Inf"} 1 1665494157768
http_server_duration_ms_sum{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo"} 679.907 1665494157768
http_server_duration_ms_count{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="500",http_target="api/Foo"} 1 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="0"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="5"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="10"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="25"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="50"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="75"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="100"} 0 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="250"} 6 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="500"} 7 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="1000"} 7 1665494157768
http_server_duration_ms_bucket{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo",le="+Inf"} 7 1665494157768
http_server_duration_ms_sum{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo"} 1071.552 1665494157768
http_server_duration_ms_count{http_flavor="1.1",http_host="localhost:5001",http_method="GET",http_scheme="http",http_status_code="200",http_target="api/Foo"} 7 1665494157768

# TYPE FooCounter counter
FooCounter 7 1665494157768

# EOF
```

Automatic instrument

通过 clr profiling 机制实现 sdk 的无侵入式注入

https://github.com/open-telemetry/opentelemetry-dotnet-instrumentation

https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/november/net-clr-profiling-services-track-your-managed-components-to-boost-application-performance

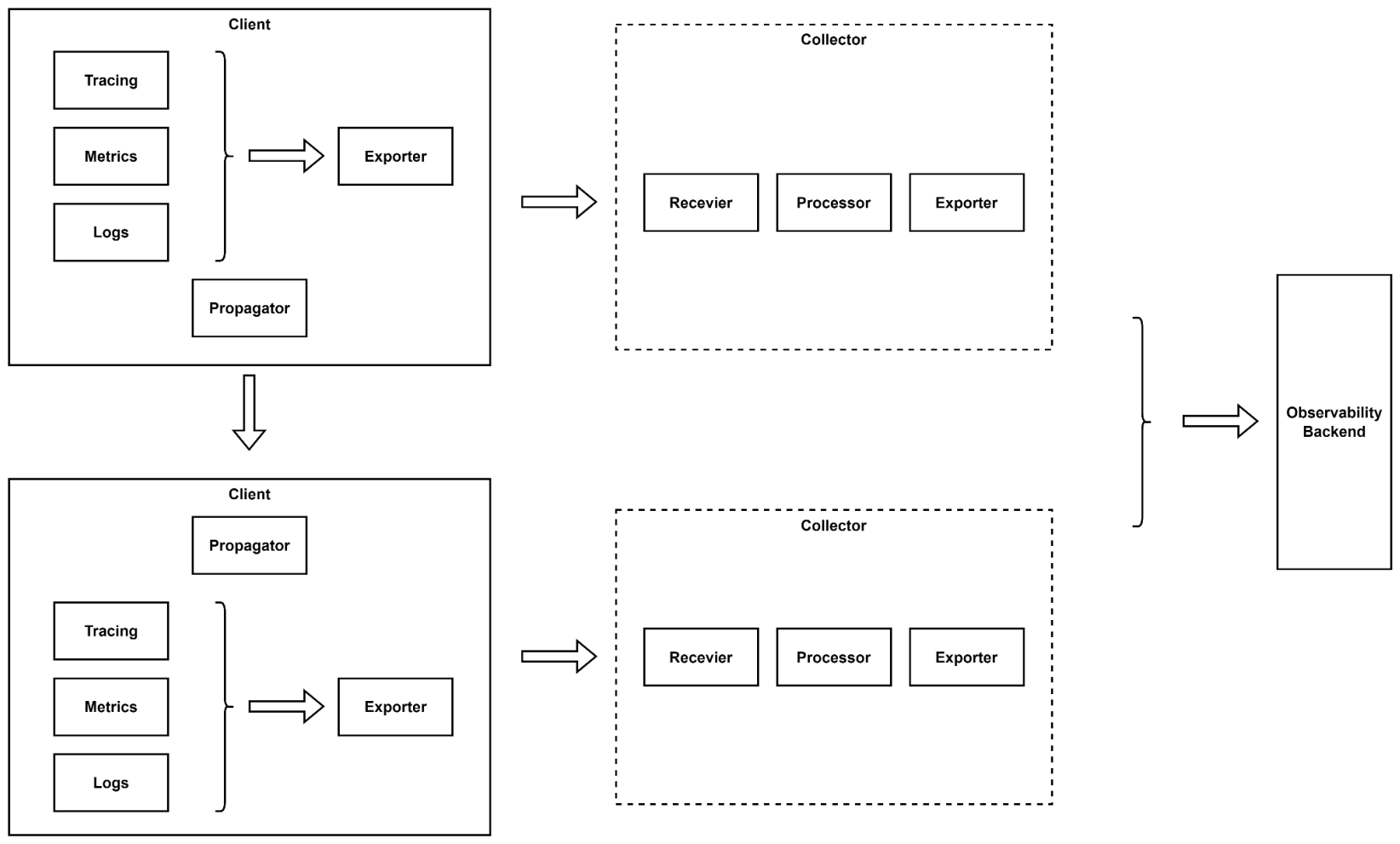


OpenTelemetry Collector

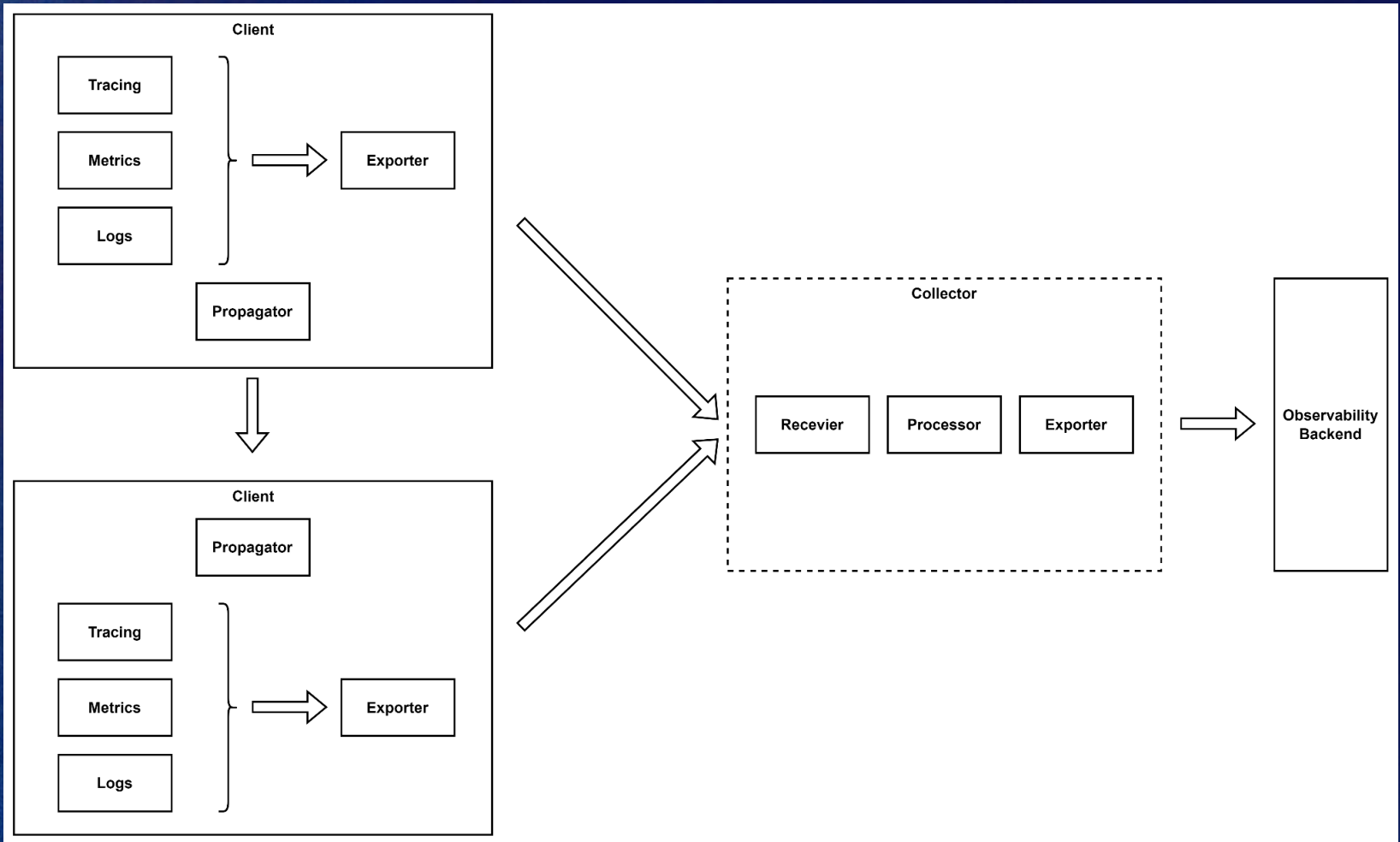
可以部署成一个 agent 或者一个独立的 gateway
https://opentelemetry.io/docs/collector/getting-started/

OTEL COLLECTOR

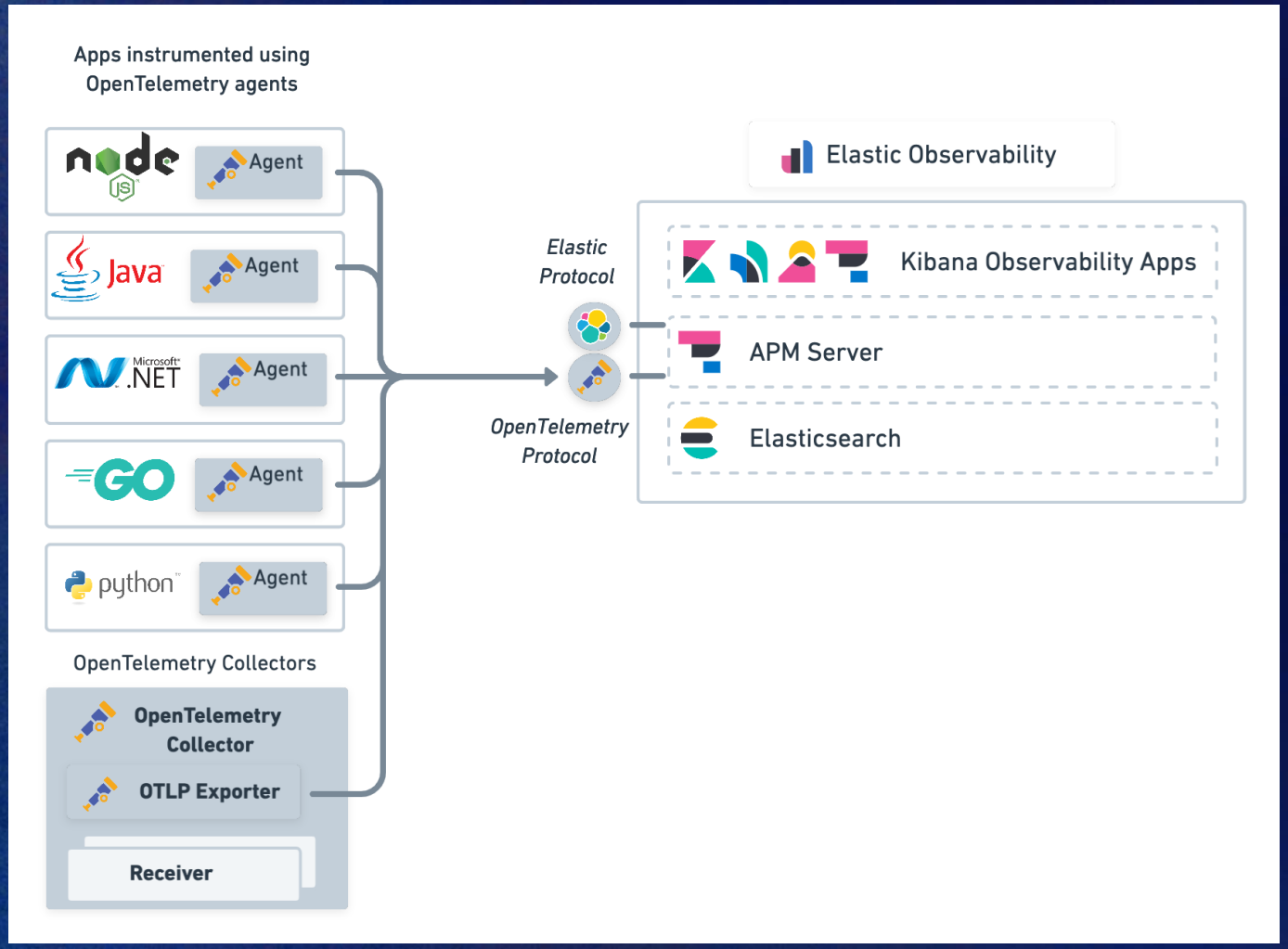
OpenTelemetry Collector: Agent



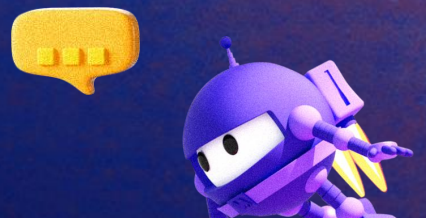
OpenTelemetry Collector: Gateway



Elastic APM


Apps instrumented using OpenTelemetry agents
node
Agent
Java
Agent
Microsoft .NET
Agent
GO
Agent
python
Agent
OpenTelemetry Collectors
OpenTelemetry Collector
OTLP Exporter
Receiver
Elastic Protocol
OpenTelemetry Protocol
Elastic Observability
Kibana Observability Apps
APM Server
Elasticsearch

集成 Elastic APM： Collector



```yaml
receivers:
 # ...
 otlp:
   protocols:
     grpc:

processors:
 # ...
 memory_limiter:
   check_interval: 1s
   limit_mib: 2000
 batch:

exporters:
 logging:
   loglevel: warn
 otlp/elastic:
   # Elastic APM server https endpoint without the "https://" prefix
   endpoint: "xxx"
   headers:
     # Elastic APM Server secret token
     Authorization: "Bearer xxx"
```

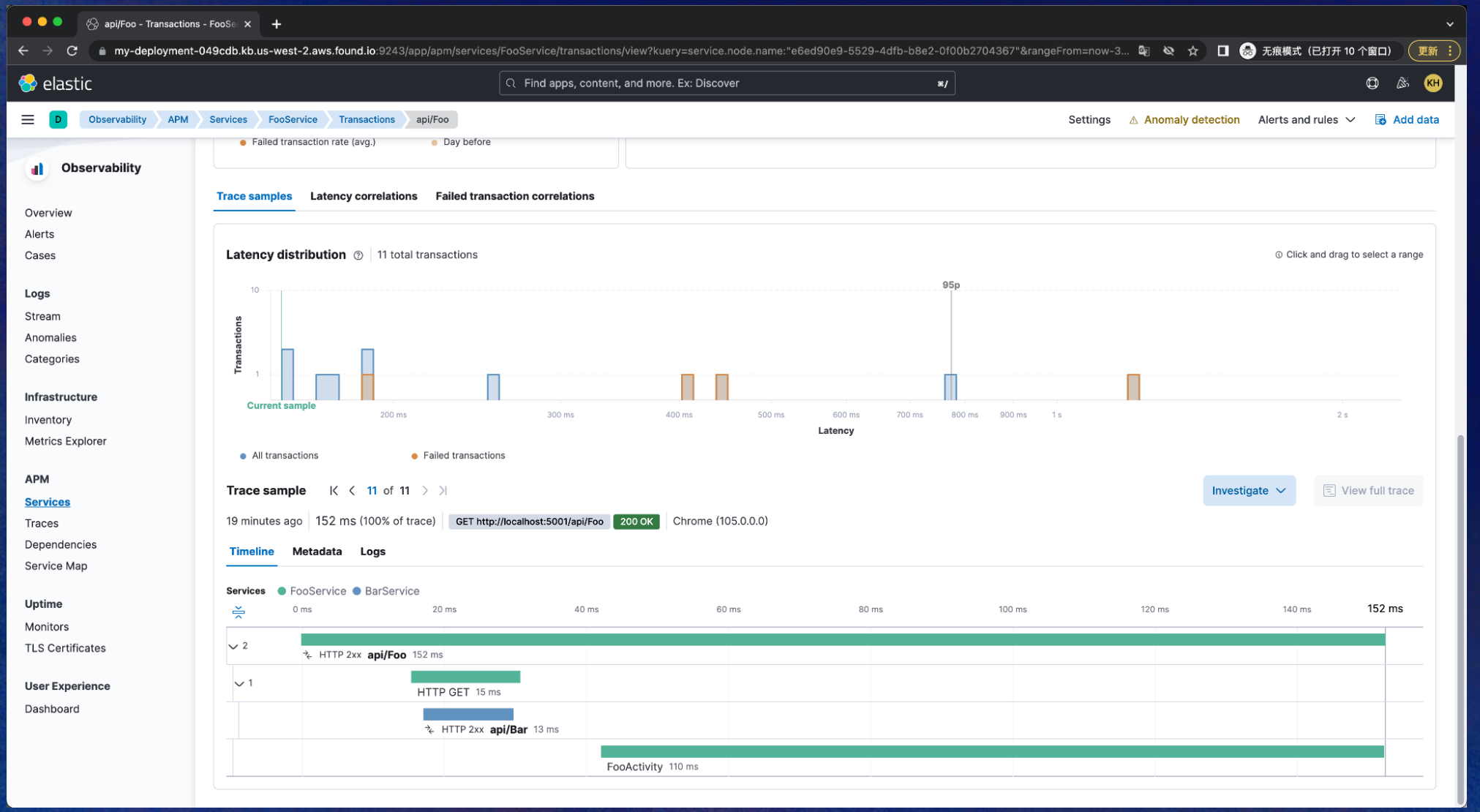
```yaml
service:
 pipelines:
   traces:
     receivers: [otlp]
     exporters: [logging, otlp/elastic]
   metrics:
     receivers: [otlp]
     exporters: [logging, otlp/elastic]
   logs:
     receivers: [otlp]
     exporters: [logging, otlp/elastic]
```

集成 Elastic APM： SDK

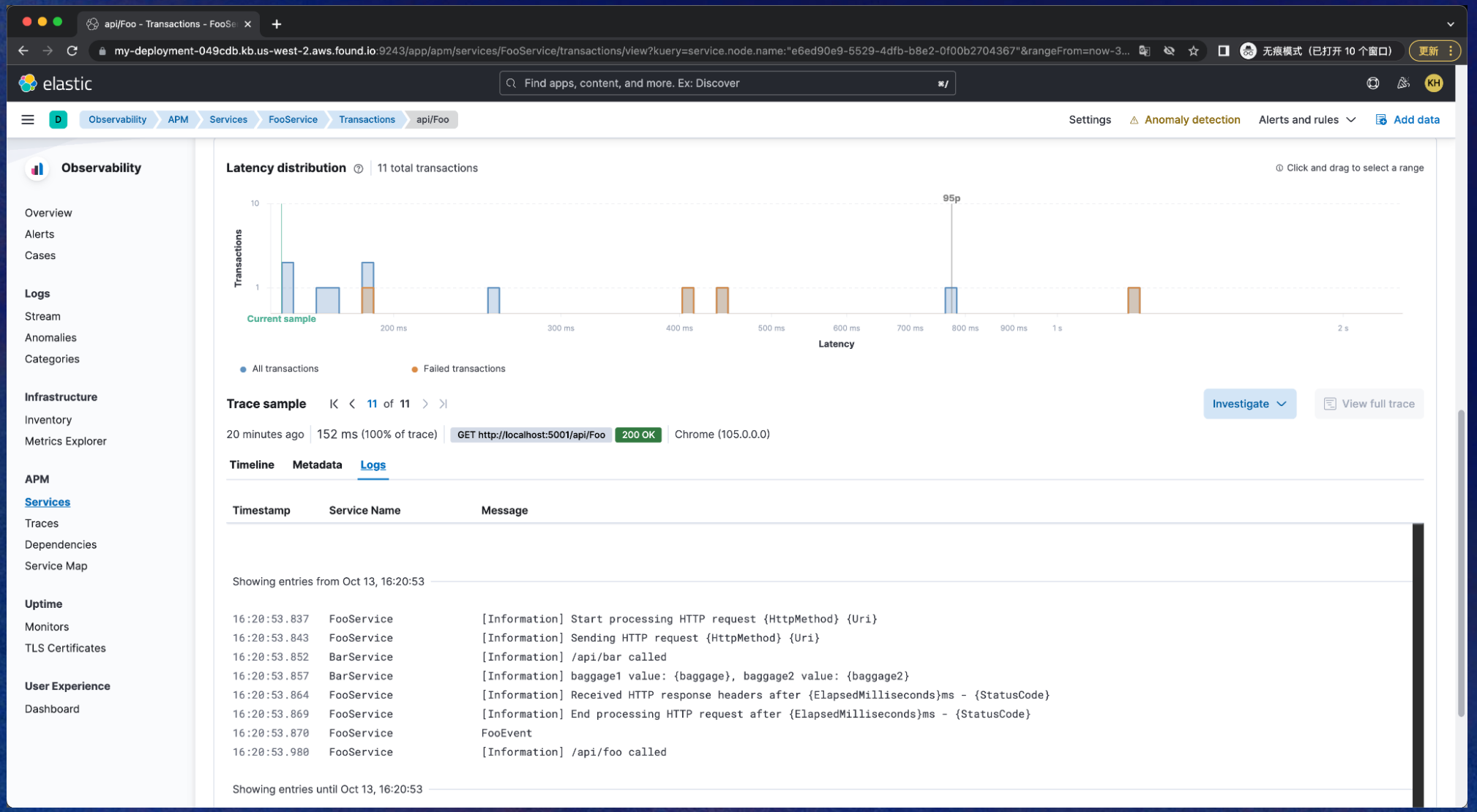
- OpenTelemetry.Exporter.OpenTelemetryProtocol
- OpenTelemetry.Exporter.OpenTelemetryProtocol.Logs

```
.AddOtlpExporter(options =>
{
    options.Protocol = OtlpExportProtocol.Grpc;
    options.Endpoint = new Uri("http://localhost:4317");
});
```

集成 Elastic APM



集成 Elastic APM



Opentelemetry Operator

https://github.com/open-telemetry/opentelemetry-operator

Continuous Integration passing | go report A+ | reference

OpenTelemetry Operator for Kubernetes

The OpenTelemetry Operator is an implementation of a Kubernetes Operator.

The operator manages:

- OpenTelemetry Collector
- auto-instrumentation of the workloads using OpenTelemetry instrumentation libraries

Documentation

- API docs

Helm Charts

You can install Opentelemetry Operator via Helm Chart from the opentelemetry-helm-charts repository. More information is available in here.

Getting started

To install the operator in an existing cluster, make sure you have `cert-manager` installed and run:

```
kubectl apply -f https://github.com/open-telemetry/opentelemetry-operator/releases/latest/download/
```

Thank you!

Let's build amazing apps with .NET 7

get.dot.net/7