OpenTelemetry .Net Instrumentation

前言

本篇文章為研究如何在.net中使用OpenTelemetry進行Instrumentation，會探討Auto + manually 以及全manually的內容

本文撰寫時的OTel版本為: `1.25`

OpenTelemetry on .net

在.net framework中，已提供logging， metrics以及activity APIs 實作OTel的標準，代表OTel不用再額外提供APIs，只要使用原生的APIs即可。.Net OTel以下列的方式實作了OTel的標準

- Microsoft.Extensions.Logging.ILogger (Logging)
- System.Diagnostics.Metrics.Meter (Metrics)
- System.Diagnostics.ActivitySource and System.Diagnostics.Activity (Tracing)

`OpenTelemetry in .NET is implemented as a series of NuGet packages that form a couple of categories:`

- Core API
- Instrumentation - these packages collect instrumentation from the runtime and common libraries.
- Exporters - these interface with APM systems such as Prometheus, Jaeger, and OTLP.

Package Name	Description
OpenTelemetry	Main library that provides the core OTEL functionality
OpenTelemetry.Instrumentation.AspNetCore	Instrumentation for ASP.NET Core and Kestrel
OpenTelemetry.Instrumentation.GrpcNetClient	Instrumentation for gRPC Client for tracking outbound gRPC calls
OpenTelemetry.Instrumentation.Http	Instrumentation for `HttpClient` and `HttpWebRequest` to track outbound HTTP calls
OpenTelemetry.Instrumentation.SqlClient	Instrumentation for `SqlClient` used to trace database operations
OpenTelemetry.Exporter.Console	Exporter for the console, commonly used to diagnose what telemetry is being exported
OpenTelemetry.Exporter.OpenTelemetryProtocol	Exporter using the OTLP protocol
OpenTelemetry.Exporter.Prometheus.AspNetCore	Exporter for Prometheus implemented using an ASP.NET Core endpoint
OpenTelemetry.Exporter.Zipkin	Exporter for Zipkin tracing

Register Customer Signals to Auto Instrumentation

第一種方式為在使用Automatic的情況下塞入自定義的Instrumentation，這種方式可以在僅修改少量原始碼的情況下，將自定義的Instrumentation塞入到Auto Instrumentation中。

接續先前`rollingDice`的範例，首先在專案中加入`System.Diagnostics.DiagnosticSource`的Nuget Package

```
<PackageReference Include="System.Diagnostics.DiagnosticSource" Version="7.0.2" />
```

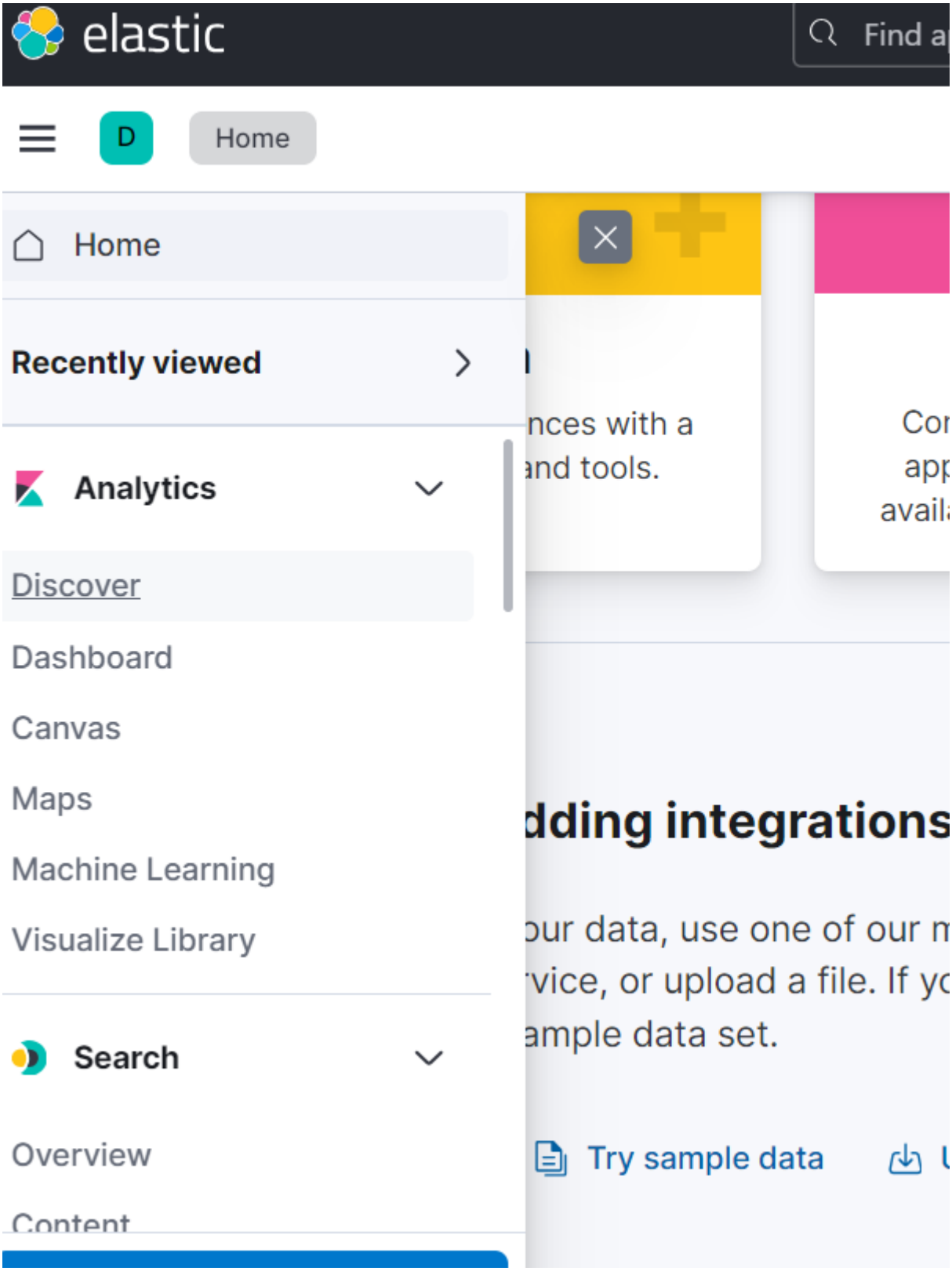
接著新增一個名為`source`的`ActivitySource`，使用的名稱為`Sample.DistributedTracing`

```
ActivitySource source = new ActivitySource("Sample.DistributedTracing", "1.1.0");
```

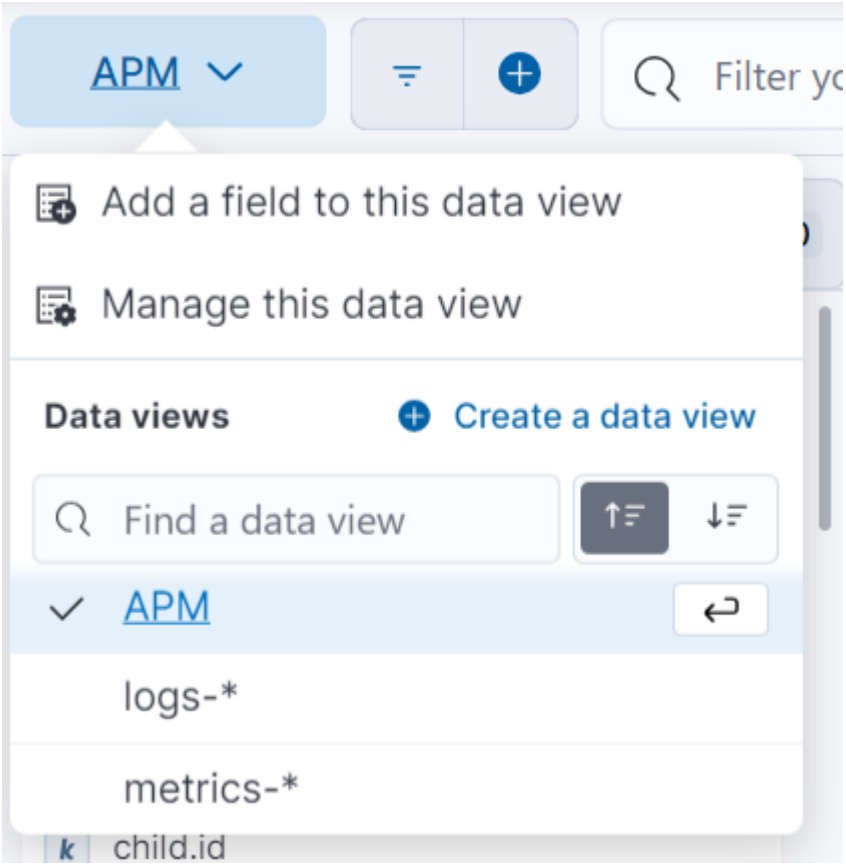
接著幫此Activity加入`key`為`foo`的tag，`value為bar1`

```
using (var activity = source.StartActivity("Main"))
{
    activity?.SetTag("foo", "bar1"); //加入此trace要使用的log
}
```

接著就是開始建置後使用ELK觀察結果，首先到左方的menu選擇discover



接著左方的filter選擇APM



接著在右方可看到剛剛傳送上來的`traces`，點選其中一個並查看其內容。

可以看到剛剛設的tag已經被傳送上來了。

以上就成功埋入相關的tag到automatic instrumentation中了。

接著為嘗試加入`metrics`

首先一樣在專案中加入`System.Diagnostics.DiagnosticSource`的Nuget Package

接著在code裡加入以下的程式碼

```
    var meter = new Meter("Sample.Service", "1.0");
    var successCounter = meter.CreateCounter<long>("srv.successes.bing",
description: "Number of successful responses");
    successCounter.Add(1, new KeyValuePair<string, object?>("tagName",
"tagValue"));
```

命名一個名為`Sample.Service`的`Meter`，並且建立一個名為`srv.successes.bing`的`counter`，並且加入一個`tag`為`tagName`, `value`為`tagValue`的`counter`。

接著就是開始建置後使用ELK觀察結果，首先一樣到左方的`menu`選擇`discover`，並且左方的filter選擇`APM` 接著在右方的filter輸入`data_stream.type:"metrics"` ，可看到下方出現相關的record，點選其中一條即可看到剛剛傳送上來的`srv.successes.bing`，並且可以看到剛剛設的tag已經被傳送上來了。

ps. metrics預設為每分鐘一筆，所以要等一分鐘才會出現。traces則是每個operation的當下產生並送出。

常用的Trace Instrumentation

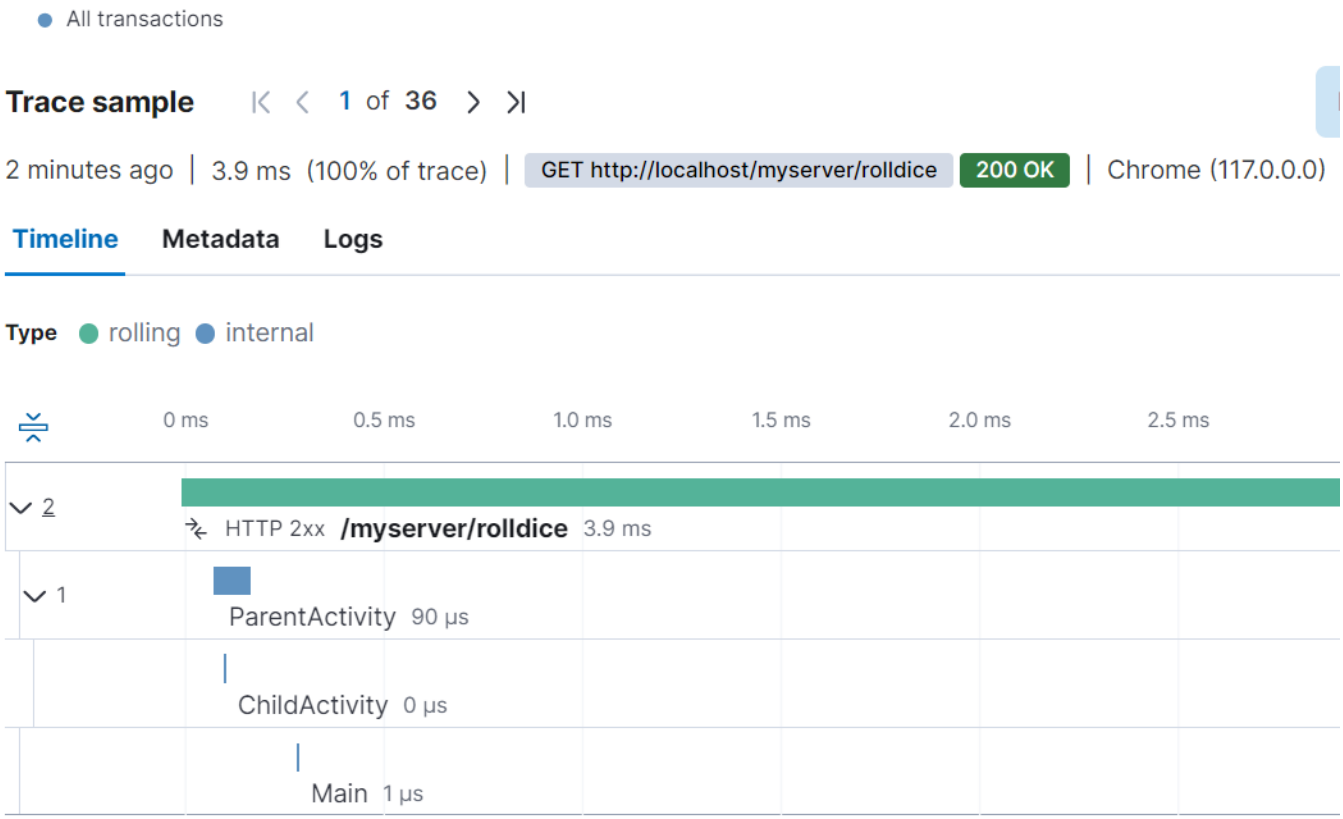
Nest Activities

指的是多個Activities並且有父子(Parent-Child)關係，這種情況下，會有一個root activity，並且會有一個parent activity，parent activity會有一個或多個child activity，child activity也可以有child activity，如此一來就會形成一個activity tree。


```
{
  "_index": ".ds-traces-apm-default-2023.10.06-000001",
  "_id": "NSAAJ4sBh0qJozGPC6sB",
  "_version": 1,
  "_score": null,
  "fields": {
    "span.name": [
      "ChildActivity"
    ],
    "service.language.name": [
      "dotnet"
    ],
    "labels.telemetry_auto_version": [
      "1.0.2"
    ],
    "trace.id": [
      "627ca227ecc96876d51575f50c8a5f39"
    ],
    "span.duration.us": [
```

```
      1
    ],//下略
}
```

```
{
  "_index": ".ds-traces-apm-default-2023.10.06-000001",
  "_id": "NiAAJ4sBh0qJozGPC6sB",
  "_version": 1,
  "_score": null,
  "fields": {
    "span.name": [
      "ParentActivity"
    ],
    "service.language.name": [
      "dotnet"
    ],
    "labels.telemetry_auto_version": [
      "1.0.2"
    ],
    "trace.id": [
      "627ca227ecc96876d51575f50c8a5f39"
    ],//下略
  },
  "sort": [
    1697166392771
  ]
}
```



Todo

1. manually instrumentation # D
2. manually register to Auto # D
3. HTTP
4. SQL
5. library instrumentation lib list
6. metrics
7. trace
8. event/span/link
9. filter
10. sampling
11. processor

resource

淺談OpenTelemetry

Microsoft learning

https://code-maze.com/opentelemetry-in-dotnet/

https://training.onedoggo.com/tech-sharing/uncle-joe-teach-es-elastc-observability

https://opentelemetry.io/docs/specs/otel/metrics/data-model/#exemplars

https://marcus116.blogspot.com/2022/01/opentelemetry-in-asp-net.html.html