Leaders.Tec.Br

Volume 1, Number 13

Strategies for collecting Metrics and Logs in WebAPI using Csharp with OpenTelemetry

Nagib Sabbag Filho
FIAP (Faculty of Informatics and Administration Paulista) Avenida Paulista, 1106 - 7º andar - Bela Vista, São Paulo, Brazil.
e-mail: profnagib.filho@fiap.com.br
PermaLink: https://leaders.tec.br/article/eefcca
set 30 2024

Abstract:
The article provides an introduction to OpenTelemetry, an essential tool for the observability of distributed systems. With the increasing complexity of applications, OpenTelemetry offers a unified standard for collecting metrics, logs, and tracing. The text details the initial setup in C# applications, including the collection of metrics and logs, integration with monitoring backends like Jaeger, and best practices for instrumentation.

Key words:
Strategies, Metrics Collection, Logs, C# Systems, OpenTelemetry, Monitoring, Observability, Instrumentation, Telemetry, Performance Analysis, Error Diagnosis, Integration, API, Structured Data, APM (Application Performance Management), Tracing, Events, Execution Context, Data Export, Visualization, Monitoring Tools, Continuous Improvement, Agile Development.

Introduction to OpenTelemetry

OpenTelemetry is a collection of tools, APIs, and SDKs that enable observability of distributed systems. With the increasing complexity of modern applications, the need for effective monitoring has become crucial. OpenTelemetry provides a unified standard for collecting metrics and logs, allowing developers and SRE engineers to better understand the behavior of their applications. This tool is an open-source project aimed at providing a pathway for developers to implement observability in their applications consistently and efficiently.

The concept of observability involves not only data collection but also the ability to understand that data and make informed decisions to improve the performance and reliability of applications. In this sense, OpenTelemetry stands out as a solution that integrates tracing, metrics, and logs, facilitating debugging and problem analysis in complex systems.

Initial Setup of OpenTelemetry in C# Applications

To start using OpenTelemetry in a C# project, you need to install the appropriate NuGet packages. Below is an example of how to configure OpenTelemetry in an ASP.NET Core application:

```
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Extensions.Hosting
dotnet add package OpenTelemetry.Instrumentation.AspNetCore
```

After installing the packages, you can configure OpenTelemetry in your ConfigureServices method:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddOpenTelemetry()
        .WithTracing(builder =>
        {
            builder
                .AddAspNetCoreInstrumentation()
                .AddHttpClientInstrumentation()
                .AddConsoleExporter();
        });
}
```

With this, you enable tracing collection for your applications, allowing OpenTelemetry to collect data from received and sent HTTP requests. This basic configuration is a great starting point for adding more instrumentation as needed.

Collecting Metrics with OpenTelemetry

Collecting metrics is an essential part of application monitoring. OpenTelemetry provides support for metrics in a simple and effective way. Here is an example of how to collect count metrics in a C# application:

```
using OpenTelemetry.Metrics;

public void ConfigureServices(IServiceCollection services)
{
    services.AddOpenTelemetryMetrics(builder =>
    {
        builder.AddAspNetCoreInstrumentation();
        builder.AddMeter("MyApplication");
        builder.AddConsoleExporter();
    });
}

private static readonly Counter&lt;long&gt; myCounter =
MeterProvider.Default.GetMeter("MyApplication").CreateCounter&lt;long&gt;("my_counter")
;

public void SomeMethod()
{
    myCounter.Add(1);
}
```

In the example above, you are creating a counter that counts how many times a specific method is called. This is useful for understanding the volume of calls and aiding in performance analysis.

Implementing Logs with OpenTelemetry

Implementing logs in OpenTelemetry is equally important. You can collect logs using the OpenTelemetry.Logs library. Here’s how to set up log collection:

```
dotnet add package OpenTelemetry.Logs
```

Then, configure log collection in your ConfigureServices method:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddLogging(builder =>
    {
        builder.AddOpenTelemetry();
    });
}
```

With this, all logs generated in the application will be collected and can be sent to the configured monitoring backends. Log collection is crucial for understanding what is happening in the application, especially in cases of failures or unexpected behaviors.

Integration with Monitoring Backend

After collecting metrics and logs, the next step is to integrate this information with a monitoring backend. OpenTelemetry supports various options, such as Jaeger, Prometheus, and Zipkin. Below is an example of integration with Jaeger:

```
dotnet add package OpenTelemetry.Exporter.Jaeger
```

In your configuration method, add the Jaeger exporter:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddOpenTelemetry()
        .WithTracing(builder =>
        {
            builder
                .AddAspNetCoreInstrumentation()
                .AddJaegerExporter(options =>
                {
                    options.AgentHost = "localhost";
                    options.AgentPort = 6831;
                });
        });
}
```

Jaeger is a popular tool for distributed tracing and allows you to visualize the calls made between services in a microservices architecture. Integrating OpenTelemetry with Jaeger makes it easier to analyze performance and identify bottlenecks in service communications.

Strategies for Creating Custom Instrumentations

Sometimes, ready-made metrics and logs are not enough. For this, OpenTelemetry allows for the creation of custom instrumentations. Here’s an example of how to create a custom metric:

```
private static readonly Histogram&lt;double&gt; myHistogram =
MeterProvider.Default.GetMeter("MyApplication").CreateHistogram&lt;double&gt;("my_histo
gram");

public void MeasureExecutionTime(Action action)
{
    var startTime = DateTime.UtcNow;
    action();
    var executionTime = (DateTime.UtcNow - startTime).TotalMilliseconds;
    myHistogram.Record(executionTime);
}
```

In the example above, you create a histogram that measures the execution time of a specific action. This metric can be extremely useful for identifying which parts of your application are taking the longest to execute.

Monitoring Distributed Applications

In microservices environments, monitoring becomes even more challenging. OpenTelemetry facilitates the collection of data from multiple distributed applications. By using tracing identifiers, you can correlate events across services. Here’s an example of how to use tracing in HTTP calls:

```
using System.Net.Http;

public async Task CallAnotherServiceAsync()
{
    using var httpClient = new HttpClient();
    var response = await httpClient.GetAsync("http://another-service/api/data");
    response.EnsureSuccessStatusCode();
}
```

This snippet of code demonstrates how to make calls to other services while maintaining the tracing context. This allows you to see the journey of a request across different services in your architecture, helping to identify where problems might be occurring.

Best Practices for Collecting Metrics and Logs

To ensure the effectiveness of metrics and logs collection, consider the following best practices:

Define relevant metrics for the business, focusing on what really matters to stakeholders.
Avoid collecting excessive data that might cause overload and hinder analysis.
Implement alerts based on anomalies in metrics, allowing for a quick response to issues.
Use tags and attributes to enrich the collected data, making it more informative and useful for analysis.
Document your instrumentations and the logic behind the metrics so that other team members can understand and contribute.
Conduct periodic reviews of the collected metrics to ensure they remain relevant and useful.

Final Considerations

OpenTelemetry is a powerful tool for collecting metrics and logs in C# applications. By following the practices and strategies discussed in this article, you can implement a robust observability solution that will not only help in detecting issues but also in the continuous improvement of your applications. Observability is not just about collecting data, but about how that data is used to guide the evolution and maintenance of your applications.

Moreover, by adopting OpenTelemetry, you are aligning with best practices in modern development, where observability is an integral part of the software lifecycle. This not only improves the quality of your applications but also provides a better experience for end-users.

References

OPEN TELEMETRY. OpenTelemetry. Available at: https://opentelemetry.io/. Accessed on: September 29, 2024.
MICROSOFT. OpenTelemetry for .NET. Available at: https://github.com/open-telemetry/opentelemetry-dotnet. Accessed on: September 29, 2024.
JAeger. Jaeger, a Distributed Tracing System. Available at: https://www.jaegertracing.io/. Accessed on: September 29, 2024.

Nagib is a University Professor and Tech Manager. He has a track record of achievements in technical and agile certifications, including MCSD, MCSA, and PSM1. He holds a postgraduate degree in IT Management from SENAC and an MBA in Software

Volume 1, Number 13

Technology from USP. Nagib has completed extension programs at MIT and the University of Chicago. Other achievements include authorship of a peer-reviewed article on chatbots presented at the University of Barcelona.