DATACIDERS

PASO

DATACIDERS
PASO

Observability

Herzlich Willkommen!

07.10.2025
in Hohenzell

DATACIDERS
PASO

Agenda

- Was versteht man unter Observability?
- Logging
- OpenTelemetry (Traces & Metrics)
- AppInsights
- Aspire

Definition

Observability

- Fähigkeit, den inneren Zustand eines Systems aus seiner externen Telemetrie abzuleiten
- Ziel
  - Schnellere Ursachenanalyse (MTTR↓)
  - Zuverlässigkeit & SLO-Einhaltung
  - Produkt-Insights & Kapazitätsplanung
- Telemetrie-Arten
  - Logs (strukturierte Ereignisse)
  - Metriken (Zeitreihen)
  - Traces (Spans über Dienste hinweg)
  - Events, Profiling, Heartbeats/Health

Kein Console-Log

DATACIDERS
PASO

- Console-Log vermeiden
  - Keine Severity
  - Keine Filterung
- Laufzeiten nicht über Logging sicherstellen
  - Cross-Cutting-Concern

❌ Don't

```
0 Verweise
public sealed class WeatherForecastService(
    WeatherDbContext dbContext
)
{
    0 Verweise
    public async Task<int> GetTemperatureForCity(string city)
    {
        Console.WriteLine($"Getting temperature for city {city}");

        Stopwatch sw = new();
        sw.Start();

        var data = (await dbContext.Weather
            .Where(x => string.Equals(x.City, city, StringComparison.OrdinalIgnoreCase))
            .FirstOrDefaultAsync()) ?? throw new EntityNotFoundException();

        sw.Stop();

        Console.WriteLine($"Getting temperature took {sw.ElapsedMilliseconds}ms");

        return data.Temperature;
    }
}
```

Keine Struktur

DATACIDERS
PASO

❌ Don‘t

```
1 Verweis
public sealed class WeatherForecastService(
    WeatherDbContext dbContext,
    ILogger<WeatherForecastService> logger
)
{
    private const string TAG = "[WeatherForecastService]";

    0 Verweise
    public async Task<int> GetTemperatureForCity(string city)
    {
        logger.LogInformation($"{TAG} ({DateTime.UtcNow}): Getting temperature for city {city}");
```

- Log soll nur die Information und nicht die Struktur beinhalten
- Struktur soll sich im Nachhinein ändern können, ohne dafür Source Code anpassen zu müssen

Log-Levels


```
// log levels
logger.LogTrace("Getting temperature for city {City}", city);
logger.LogDebug("Getting temperature for city {City}", city);
logger.LogInformation("Getting temperature for city {City}", city);
logger.LogWarning("Getting temperature for city {City}", city);
logger.LogError("Getting temperature for city {City}", city);
logger.LogCritical("Getting temperature for city {City}", city);
```

- Trace & Debug ➔ können und sollen in sehr hoch-frequenten Bereichen genutzt werden
- Information ➔ soll eher sporadisch verwendet werden und den generellen Ablauf darstellen
- Warning ➔ für unerwartete Situtationen die aber noch funktionieren
- Error ➔ für fehlgeschlagene Operationen
- Critical ➔ z.B. für Infrastructure Ausfälle

Good to know

DATACIDERS
PASO

```
logger.LogInformation($"Getting temperature for city {city}");

// vs.

logger.LogInformation("Getting temperature for city {City}", city);
```

```
if(logger.IsEnabled(LogLevel.Debug))
{
    var count = await dbContext.Weather.CountAsync();
    logger.LogDebug("Looking up temperature in {Count}", count);
}
```

```
public partial class TestingController(
    ILogger<TestingController> logger
): ControllerBase
{
    [LoggerMessage(0, LogLevel.Information, "This is testing logging for value={Value}")]
    3 Verweise
    partial void LogTesting(string value);
```

Serilog

- Serilog stellt Implementierung für ILogger<> bereit
- In unterschiedliche Nugets aufgeteilt, um die Log-Events in unterschiedliche Sinks zu schicken z.B.:
  - Serilog.Sinks.Console (in Serilog.AspNetCore enthalten)
  - Serilog.Sings.File (in Serilog.AspNetCore enthalten)
  - Serilog.Sinks.Grafana.Loki
  - Serilog.Sinks.OpenTelemetry ⭐
- Log-Events können an mehrere Sinks geschickt werden
- Configuration im Code oder über appsettings

DATACIDERS
PASO

Serilog - DEMO

DATACIDERS
PASO

Daten sammeln

- OpenTelemetry definiert ein herstellerneutrales Format für Traces, Metrics und Logs
- Wird von vielen Libraries unterstützt z.B.
  - Microsoft selbst für Request-Response
  - Datenbanken: SQL Server, Postgres* …
  - Kann einfach selbst erweitert werden
- Sammelt Daten im Hintergrund, welche später über Servicegrenzen ausgewertet werden können

OpenTelemetry

Einbinden

```
List<KeyValuePair<string, object>> attributes = [
    new("service.name", builder.Environment.ApplicationName),
    new("service.version", Assembly.GetEntryAssembly()?.GetName().Version?.ToString() ?? "unknown"),
];

builder.Services.AddOpenTelemetry()
    .WithTracing(traces => traces
        .AddAspNetCoreInstrumentation()
        .AddHttpClientInstrumentation()
        .AddNpgsql()
        .AddCustomEvent()
        .AddOtlpExporter())
    .WithMetrics(metrics => metrics
        .AddAspNetCoreInstrumentation()
        .AddHttpClientInstrumentation()
        .AddRuntimeInstrumentation()
        .AddOtlpExporter())
    .ConfigureResource(x => x.AddAttributes(attributes));
```

* Würd auch für Logging in gleicher Form funktionieren ➔ falls man Serilog nicht verwenden möchte

Funktionsweise

DATACIDERS
PASO

- Span wird verwendet, um einen begrenzten Zeitabschnitt zu definieren. Enthält dabei:
  - SpanId & ParentSpanId
  - StartZeitpunkt & Dauer
  - Typ: server, client, producer, consumer, internal
  - Attribute ➔ können beliebig vergeben werden
  - Events ➔ Logging-Einträge innerhalb des Spans
- Mehrere Spans in Parent-Child-Beziehung ergeben einen Trace
- Kommuniziert wieder über gRPC oder HTTP

DATACIDERS
PASO

Traces - DEMO

Metrics

DATACIDERS
PASO

- Standardisiert Messwerte (Zähler, Dauer, Größe, Auslastung…)
  - Counter
  - Histogram
  - Observable Gauges/Counters ➔ ziehen sich den Wert on demand
- Daten können später über Portal ausgewertet werden
- Werte können auch in Dashboards angezeigt werden
  - z.B. bei Welchen Interfaces kamen wie viele Fehler (DataHub - Steyr)

DATACIDERS
PASO

Metrics - DEMO

Überblick

DATACIDERS
PASO

- Azure Resource für Traces & Metrics
- Mittlerweile über OpenTelemetry anbindbar
- Sehr intuitiv zu bedienen
- Bietet einige gutes Features
  - Performance ➔ zeigt die durchschnittlich langsamsten Requests an
  - Transaktion-Search ➔ erlaubt das Filtern auf eine spezielle Transaktion
  - Wasserfall-Ansicht ➔ Traces können sauber in einer Wasserfall-Ansicht angezeigt werden

DATACIDERS
PASO

AppInsights - DEMO

Systeme

- Logging
  - Serilog (Framework) ➔ immer verwenden, Sink Auswahl nach belieben
  - Seq ➔ sehr einfach aufzusetzen & verwenden. Store&UI im gleichen Container
  - Loki&Grafana ➔ gut geeignet, wenn man bereits Grafana Dashboard hat oder eines plant
- Traces
  - AppInsights ➔ super einfach einzubinden, viele nützliche Funktionen.
  - Jaeger ➔ übersichtliche Darstellung, würd ich verwenden, wenn AppInsights aus irgendwelchen Gründen nicht verwendet werden kann.
- Metrics
  - AppInsights
  - Prometheus ➔ verwenden wenn AppInsights nicht verwendet werden kann
- ELK Stack (Elastic Search, Logstash, kibana) ➔ Konkurrenz zu Grafana/Prometheus/Loki

Aspire

Dashboard

DATACIDERS
PASO

- Container-Orchestrierung für lokale Entwicklung
  - Aktuell nicht in Production zu verwenden
- Vereinfacht das Aufsetzen der Entwicklungsumgebung
  - Onboarding von neuen Kollegen geht superschnell
- Bietet UI für Traces & Metrics während der Entwicklung
  - Für lokal dev daher kein anderes System nowendig
  - Für Production muss nur OTEL-Endpunkt über Umgebungsvariable geändert werden

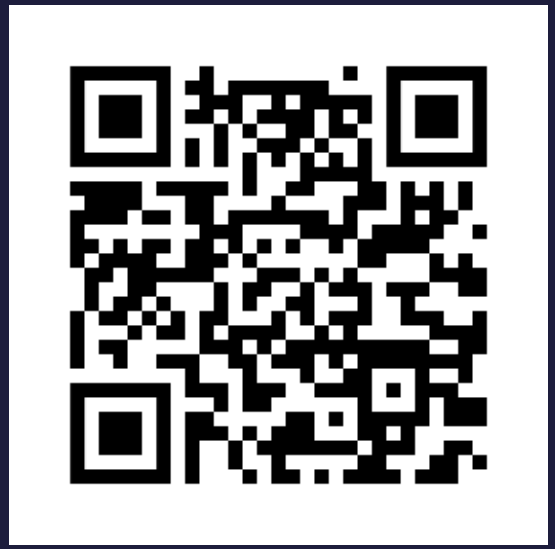
DATACIDERS
PASO

Aspire - DEMO

Recap

- Serilog verwenden ➔ direkt über OpenTelemetry
- Immer strukturierte Logs verwenden und keine String-Interpolation
- OpenTelemetry soll immer aktiviert werden
  - Zielsystem (z.B. AppInsights) kann später über Environment-Variable festgelegt werden
- Zeitmessungen über eigenen Span abwickeln und nicht manuell im Code
- Aspire ist geil 🚀

DATACIDERS
PASO



Fragen?

https://github.com/schmidi165/Observability