Using OpenTelemetry to track data flow in distributed legacy systems

Elias Leonhardsberger

BACHELOR THESIS

submitted to the
Bachelor's Degree Program
Software Engineering

at the
University of Applied Sciences Upper Austria
in Hagenberg

2025

Supervisor: FH-Prof. DI Johann Heinzelreiter

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, July 1, 2025

Elias Leonhardsberger

Abstract

Keeping track of distributed software systems is already a complex task. When large volumes of data are transferred between multiple endpoints, traditional logging often fails to provide answers regarding data quality and system reliability.

Traces and metrics can help to mitigate these issues, but the programs responsible for data transfer are often implemented as independent background jobs with varying schedules, often even on independent machines.

Fortunately, OpenTelemetry allows for the unification of traces across multiple machines using collectors. However, the issue of trace scope remains: a trace typically represents a single request or job execution, making it difficult to track specific data points.

The focus of this thesis is solving this problem. The goal is to provide a method for collecting, unifying, and then splitting traces into new ones that illustrate how individual data points move through the system. The work includes generating traces with semantic events at the job level, linking traces from different jobs that process the same data, splitting and sampling these traces into one per data point using an OpenTelemetry collector, and finally generating metrics from these traces to provide a big picture view of data transfer.

Kurzfassung

Den Überblick über verteilte Softwaresysteme zu behalten, ist bereits eine Herausforderung. Wenn jedoch große Datenmengen zwischen mehreren Endpunkten übertragen werden, scheitert traditionelles Logging oft daran, Fragen zur Datenqualität und Zuverlässigkeit der Systeme zu beantworten.

Traces und Metriken können helfen, diese Probleme zu bewältigen. Die Programme, die diese Daten übertragen, sind jedoch häufig als unabhängige Hintergrundprozesse mit unterschiedlichen Ausführungszeiten implementiert, oft sogar auf voneinander unabhängigen Maschinen.

Glücklicherweise ermöglicht OpenTelemetry die Vereinheitlichung von Traces über mehrere Maschinen hinweg mithilfe eines Collectors. Dennoch bleibt das Problem des Trace-Scopes bestehen: Ein Trace umfasst in der Regel eine Anfrage oder die Ausführung eines Jobs, was es schwierig macht, einzelne Datensätze nachzuverfolgen.

Der Fokus dieser Arbeit liegt darauf, dieses Problem zu lösen. Ziel ist es, eine Möglichkeit zu schaffen, Traces zu sammeln, zu vereinheitlichen und anschließend in neue Traces aufzuteilen, die den Weg einzelner Datenpunkte durch das System zeigen. Die Arbeit umfasst die Erstellung von Traces mit semantischen Ereignissen auf Job-Ebene, die Verknüpfung von Traces verschiedener Jobs, die an denselben Daten arbeiten, das Aufteilen und Abtasten dieser Traces in jeweils einen Trace pro Datenpunkt in einem OpenTelemetry Collector sowie die Generierung von Metriken aus diesen Traces, um ein Gesamtbild des Datentransfers zu erhalten.

Contents

Chapter 1

Introduction

1.1 Motivation

OpenTelemetry has established itself as a groundbreaking new standard in the field of observability Young, 2021; Young and Parker, 2024, Young, 2024 S.1234. It is replacing many vendor-specific solutions with a unified, standardized, extensible, open-source, and language-integrated framework.

While OpenTelemetry was initially designed with microservices in mind, its applicability extends far beyond that Boten and Majors, 2022. Distributed systems of all kinds, including legacy systems, can benefit significantly from its capabilities Gomez Blanco, 2023.

1.2 Problem

A major challenge in legacy systems, especially those that have evolved organically over time, is the absence of a single source of truth Flanders, 2024. Data entries, such as employee information, are often stored across multiple, loosely connected subsystems. For instance, the data indicating which manager an employee reports to might reside in a different subsystem than the one storing team assignments. These structural issues are typically beyond the immediate control of developers. Centralizing and unifying these data sources is a complex and time-consuming process that requires a company-wide effort to address technical debt and correct historical inconsistencies.

To maintain operational continuity during this transition, data from legacy systems must be synchronized with the new system. However, this migration is prone to mismatches, invalid states, and other errors due to the fragmented nature of the data. While these issues may take time to resolve, they should not halt the entire migration process.

Developers and decision-makers need to stay informed about when, why, and for how long errors occur, without having to manually sift through logs from multiple applications. A centralized observability dashboard that aggregates logs, traces, and metrics can provide this visibility.

While basic telemetry collection is achievable using zero-code instrumentation and backends like Jaeger, Zipkin, Prometheus, or Microsoft Application Insights, the challenge lies in the volume and complexity of the logs. Traces, which are mostly underuti-

lized, offer a more structured and insightful alternative Young, 2021.

1.3 Goals

The primary goal of this thesis can be split into two distinct parts: (1) to generate detailed traces for individual data points during data transfer jobs, and (2) to collect and reorganize these traces to visualize the flow of data through the system.

1.3.1 Trace Creation

Each data transfer job execution should generate a trace that includes child spans for every data point processed. These spans will contain metadata such as transfer status, timestamp, data type, and a unique identifier. The parent span will include contextual information like source, destination, and a flag indicating that the trace should be processed by a custom collector.

1.3.2 Trace Collection and Regrouping

OpenTelemetry allows telemetry data to be exported either directly to a backend or via a collector. The extended collector developed in this thesis will intercept traces, extract spans based on data point identifiers, and regroup them into new traces that represent the journey of individual data points. These regrouped traces will then be sampled and exported. Since jobs may run independently and at different times, the collector must persist telemetry data to enable correlation across time.

1.4 Methodology

This thesis will involve the development of a prototype system using two main technologies:

- **.NET (C#):** Used to implement the base system responsible for executing data transfer jobs and generating OpenTelemetry traces.
- **GO (Golang):** Used to extend an OpenTelemetry collector with a custom processor that processes, filters, and reorganizes traces.

1.4.1 Base System Development

A .NET-based application will be implemented to simulate data transfer jobs. The application will be instrumented using the OpenTelemetry SDK to generate traces enriched with semantic events and metadata. Each job execution will produce a trace with child spans for each data point, including metadata such as transfer status, timestamp, data type, and a unique identifier.

1.4.2 Custom Collector Implementation

The OpenTelemetry collector will be expanded with a custom processor to reorganize traces. This collector will:

- Intercept and persist telemetry data.
- Extract spans based on data point identifiers.
- Regroup spans into new traces representing the journey of individual data points.
- Sample and export the reorganized traces to a telemetry backend.

The collector must support delayed correlation of spans, as jobs may run independently and at different times.

1.4.3 Evaluation and Metrics

Metrics will be generated from the reorganized traces to provide insights into:

- Data quality and consistency.
- Frequency and duration of errors.
- System performance across different jobs and machines.

These metrics will be visualized using a telemetry backend such as Microsoft Application Insights or Prometheus.

Chapter 2

Basics

2.1 Monitoring vs Observability

2.2 Logs

2.3 Metrics

2.4 Traces

2.5 Distributed Systems

2.5.1 Legacy Systems

2.5.2 Microservices

2.5.3 Observability in Distributed Contexts

Chapter 3

OpenTelemetry in .NET

3.1 Instrumentation

3.1.1 Auto- vs Manual Instrumentation

3.1.2 Trace Generation

Chapter 4

OpenTelemetry Collector

4.1 Architecture and Deployment

4.1.1 No Collector

4.1.2 Agent

4.1.3 Gateway

4.2 Configuration

4.3 Extending the Collector

4.3.1 Creating a custom Processor

4.3.2 Span Extraction and Regrouping

4.3.3 Metric Generation

4.3.4 Sampling

4.3.5 Error Handling

Chapter 5

Prototype

5.1 Architecture Diagram

5.2 Configuration

5.3 Example

References

Literature

Boten, A., & Majors, C. (2022, May). *Cloud-Native observability with OpenTelemetry*. Packt Publishing. (Cit. on p. 1).

Flanders, S. (2024, October). *Mastering OpenTelemetry and observability*. John Wiley & Sons. (Cit. on p. 1).

Gomez Blanco, D. (2023, March). *Practical OpenTelemetry* (1st ed.). APress. (Cit. on p. 1).

Young, T. (2021, December). *The future of observability with OpenTelemetry*. O'Reilly Media. (Cit. on pp. 1, 2).

Young, T., & Parker, A. (2024, March). *Learning OpenTelemetry*. O'Reilly Media. (Cit. on p. 1).