Using OpenTelemetry to keep track of data flow in distributed systems

Elias Leonhardsberger

BACHELOR THESIS (PROPOSAL)

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

Contents

Kurzfassung ii

Abstract iii

Exposé 1

1 Motivation . . . 1
1.1 TO BE DELETED: NOTES . . . 1
1.2 ACTUAL TEXT . . . 1
2 Problem . . . 1
2.1 TO BE DELETED: NOTES . . . 1
2.2 ACTUAL TEXT . . . 1
3 Goals . . . 2
3.1 TO BE DELETED: NOTES . . . 2
3.2 ACTUAL TEXT . . . 2
4 Methodic ODA SO IRGENDWIE . . . 2
4.1 TO BE DELETED: NOTES . . . 2
4.2 ACTUAL TEXT . . . 3

References 4

Kurzfassung

Den Überblick über verteilte Softwaresysteme zu behalten, ist schon schwierig genug, aber wenn große Datenmengen von und zu mehreren Endpunkten transportiert werden, scheitert traditionelles Logging oft daran Fragen über die Datenqualität und Programmzuverlässigkeit zu beantworten.

Traces und Metriken können dabei helfen, diese Probleme zu bewältigen, aber die Programme die diese Daten transferieren, sind oft als unabhängige Hintergrund Prozesse mit unterschiedlichen Ausführungszeiten implementiert, oft sogar auf unabhängigen Maschinen.

Glücklicherweise kann man mit OpenTelemetry Traces über mehrere Maschinen hinweg durch einen Kollektor vereinheitlichen. Dadurch bleibt aber noch das Problem des Umfangs eines Traces.

Der Umfang eines Traces ist normalerweise eine Anfrage oder Jobausführung. Das macht es schwierig den Überblick über einzelne Datensätze zu behalten. Der Fokus dieser Arbeit liegt darauf dieses Problem zu lösen.

Das Ziel ist es, eine Möglichkeit zu schaffen, Traces zu sammeln, zu vereinheitlichen und dann in neue Traces aufzuteilen, die zeigen, wie sich Datenpunkte durch das System bewegen.

Die Arbeit erstreckt sich von der Erstellung von Traces mit semantischen Ereignissen auf der Job Ebene, über die Verbindung von Traces von verschieden Jobs, die an den gleichen Daten arbeiten, dann das Aufteilen und Abtasten der Traces in je einen Trace für jeden Datenpunkt in einem OpenTelemetry Kollektor und schließlich die Generierung von Metriken aus diesen Traces, um ein Gesamtbild des Datentransfers zu erhalten.

Abstract

Keeping track of distributed software systems is hard as it is, but when moving a large amount of data from and to multiple endpoints, traditional logging often fails to answer questions about data quality and program reliability.

Traces and metrics can help to mitigate these problems, but the programs transferring the data are often implemented as independent background jobs using different schedules, often even on independent machines.

Luckily using OpenTelemetry you can unify traces across multiple machines easily with collectors, leaving only the problem of trace scope.

The scope of a trace is normally one request or job execution, which makes keeping track of specific data points hard.

The focus of this thesis is fixing this problem.

The goal is to provide a way to collect, unify and then split traces into new traces showing data points travelling through a system.

This thesis stretches from creating traces with semantic events on the job level, to connecting traces from different jobs working on the same data, then splitting and sampling these traces into multiple traces for each data point in an OpenTelemetry collector and finally generating metrics out of these traces to provide a big picture view of the data transfer.

Exposé

1 Motivation

1.1 TO BE DELETED: NOTES

- Current State of OTEL and a future of very probable mass adoption(with citations)
- OTEL not only in Microservices, but all distributed systems. Company wide observability
- Transition to the problem, mentioning services which transfer a lot of data

1.2 ACTUAL TEXT

OpenTelemetry has established itself as a groundbreaking new standard in the world of observability. It is replacing many vendor specific solutions with a single, standardized, extensible, open source and language integrated solution.

OpenTelemetry has been designed with microservices in mind, but it is not exclusive to microservices. Distributed systems of all kinds, even legacy systems, can profit of OpenTelemetry.

2 Problem

2.1 TO BE DELETED: NOTES

- Fractured company software systems with multiple sources of truth.
- Merging and transferring data across multiple systems.
- Inconsistent data
- Notifying admins of the different single systems about data quality.
- Concrete Problem - try to explain swietelsky hell

2.2 ACTUAL TEXT

A big problem in legacy systems, especially naturally grown ones, is that often no single source of truth exists. So data entries, like employee data, are often stored in multiple loosely connected sub systems. For example the information which employee has which manager could be stored in another subsystem as the information in which

team an employee is working. Issues like these are often structural and can not directly and immediately be influenced by the developers tasked with dealing with the system. Unifying the data sources and saving them in a centralized system is a tedious and long undertaking, requiring the whole companies effort to clean up past technical dept and incorrect data.

To keep the company operational while this is and to migrate the system step by step, the data from the legacy sources needs to be synced with the new system, but there are bound to be mismatches, invalid states and other errors, due to no single source of truth existing. These issues can often take a long time to fix, but they should not stop the entire syncing process. The developers and decision makers do however need to be informed when, for how long and why errors are happening, without having to click through every applications logs.

3 Goals

3.1 TO BE DELETED: NOTES

- Explain Traces and Metrics in the context of OpenTelemetry.
- Explain Collectors, their use, components and different types of Collectors.
- Write a Collector in GO and generate traces in a .NET Application to
  - Provide an overview over the data quality in the system.
  - See where errors are happening and what their source could be.
  - Have traces for every data point travelling through a system, next to the traces per job execution.
  - Metrics which are able to be grouped by source, destination, entity being synced, job or machine executing the job.
- do that a bit more and a bit better lol

3.2 ACTUAL TEXT

To help getting through this chaos, all the observability data, meaning logs, traces and metrics, should be collected to a single dashboard showing all the issues and the performance of the whole system.

This alone is easily achieved using simple zero-code instrumentation and a telemetry backend of your choice like Jaeger, Zipkin, Prometheus or in this case Microsoft Application Insights.

WOS DRAUM SCHWA IS; WOS AUSAKEMMA SOI UND WIE I DES LÖS MIT COLLECTOREN UND SO

4 Methodic ODA SO IRGENDWIE

4.1 TO BE DELETED: NOTES

- Explain Techstack(.net, golang)
- Talk about Prototype and results

4.2 ACTUAL TEXT

References