International Journal of Enhanced Research in Management & Computer Applications
ISSN: 2319-7471, Vol. 11 Issue 12, December, 2022, Impact Factor: 7.751

Demystifying the Operational Complexity with Distributed Tracing (A Step Ahead in Open Telemetry)

Amit Sengupta

Senior Delivery Manager, Software Engineering, Capgemini America

ABSTRACT

Microservices provides enormous agility and flexibility to software development process. By partitioning large applications into interdependent services which communicate via explicit network communication contracts, each team encapsulates their implementation from the others. In practice, microservices are really a tradeoff for one set of organizational and technical problems for others. While the benefits of microservices amount to greater independence, clearer organizational boundaries and greater agility, such benefits do come with some distinct costs:

Loss of Traceability- As part of microservice design augmentation single end-user request is broken across multiple processes, possibly written in multiple frameworks and implementation languages which makes it much harder to track what exactly happened in the course of processing a request. Unlike a monolithic process, where we could gather the complete story of how a request was handled from a single process written in a single language, we no longer have an easy way of doing that in a microservices environment.

Increased troubleshooting spend – With the loss of traceability the act of tracking down and fixing sources of errors inside microservice architectures can be tremendously more expensive and time-consuming than its counter-parts. To add to this, in most cases failure data cannot be correlated in a clear manner inside microservices. Instead of an immediately understandable stack trace, we have to work backwards from status codes and error messages propagated across the network.

Cross-team Dependencies - Requests has to make multiple hops over the network and has to be handled by multiple processes developed by independent teams, figuring out exactly where an error has occurred and whose responsibility it is to fix, does become an exercise of frustration. The practice of debugging microservices often involves sitting developers from multiple product teams down in a conference room correlating timestamped logs from multiple services. The core of the problem really is that distributed approaches to developing software, such as microservices, really require different tools than what we used in the past when developing monoliths. We can't expect to attach debuggers to four different processes and try to step-through-debug requests in that environment and that is where **Distributed Tracing** becomes an impeccable requirement.

Keywords: - Distributed Tracing, Open Telemetry, Microservices Patterns, Customer Journey Analysis, Customer Success, Enterprise Resiliency and Reliability

Concept of Distributed Tracing: -
When a request is received in a distributed system, it can go through several microservices and infrastructure components before returning a response. Distributed tracing provides a way to visualize and track the path of requests flowing through distributed applications.

Components of Distributed Tracing: -
The components of a distributed tracing system might vary based on the implementation, but a typical system is built up with the below components:

Trace
A trace refers to a complete end-to-end path of a request or transaction as it flows through a distributed system. It represents the journey of a specific operation as it traverses various components and services in a distributed architecture.

Span
A span represents a single operation or unit of work within a distributed system. It captures the timing and metadata associated with a specific operation and provides a way to track and understand the behavior of individual components and services.

Context propagation
Context propagation refers to passing contextual information between different components or services within a distributed system. In distributed tracing, context propagation is crucial for connecting and correlating spans to construct a complete trace of a request or transaction as it flows through various services.

Instrumentation libraries
Instrumentation libraries are software components that developers integrate into their applications to collect tracing data. They are specifically designed to generate, collect, and manage trace data, which includes information about spans and the overall trace.

Instrumentation libraries can automatically capture useful information such as start time, end time, and metadata about each operation (span). They are also responsible for context propagation, ensuring that trace context is passed along with requests as they move through different services in a distributed system.

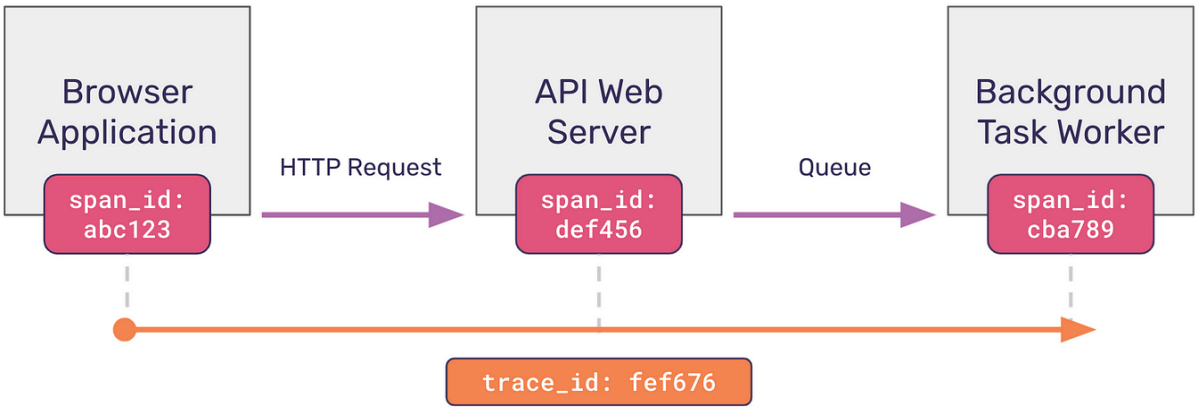
Tracing through data collectors
These are the components that receive and store trace data, usually in a distributed datastore such as Elasticsearch or Cassandra. Some of the available and well known tracing data collectors would be Jaeger, Zipkin, Open Telemetry Collector, AWS X-Ray and Google Cloud Trace.

Visualization and analysis
These are the UI Based components that provide a graphical representation of the trace data, allowing developers to visualize the flow of requests through the system and identify performance issues.

Trace Analyzer
These are tools that provide detailed analysis of trace data, allowing developers to identify bottlenecks and optimize system performance. Some of the available trace analysis tools that are widely used are Helios, Trace Compass, Amegraphs, Performance Co-Pilot (PCP), Grafana, Apache SkyWalking, Dynatrace.

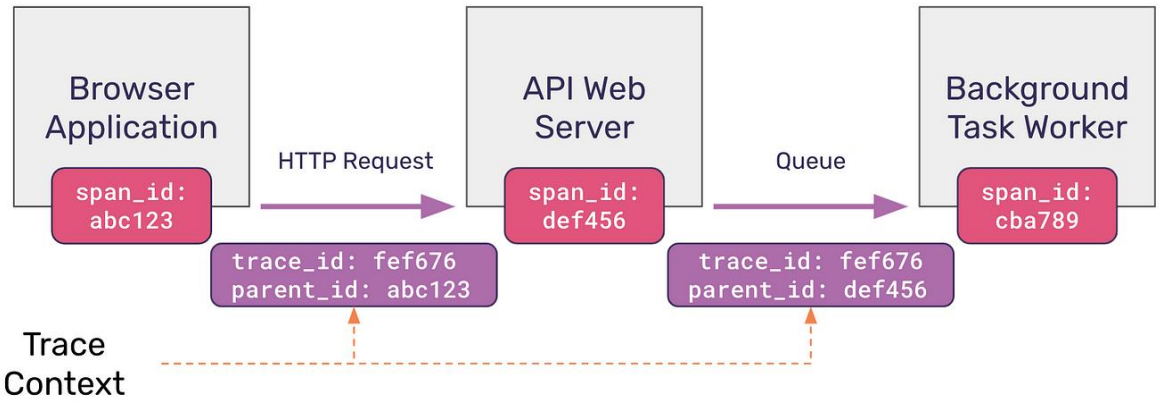
Techniques to setup distributed tracing for microservices: -
Setting up distributed tracing across application microservices landscape would typically involve integrations for the tracing components: -

A unique identifier is assigned to each incoming request
When a request enters your system, a unique identifier is assigned to it. This identifier tracks the request as it moves through the system. This is called a “Trace Id”. Bellow image shows how a trace id is assigned to a request as it flows down the downstream components.



Instrumentation libraries capture trace data

As the request moves through the system, various components and services capture trace data and add it to the request's trace context. This includes timestamps, service and endpoint names, and any relevant metadata. The below image depicts the trace context.



Trace data is propagated through the system

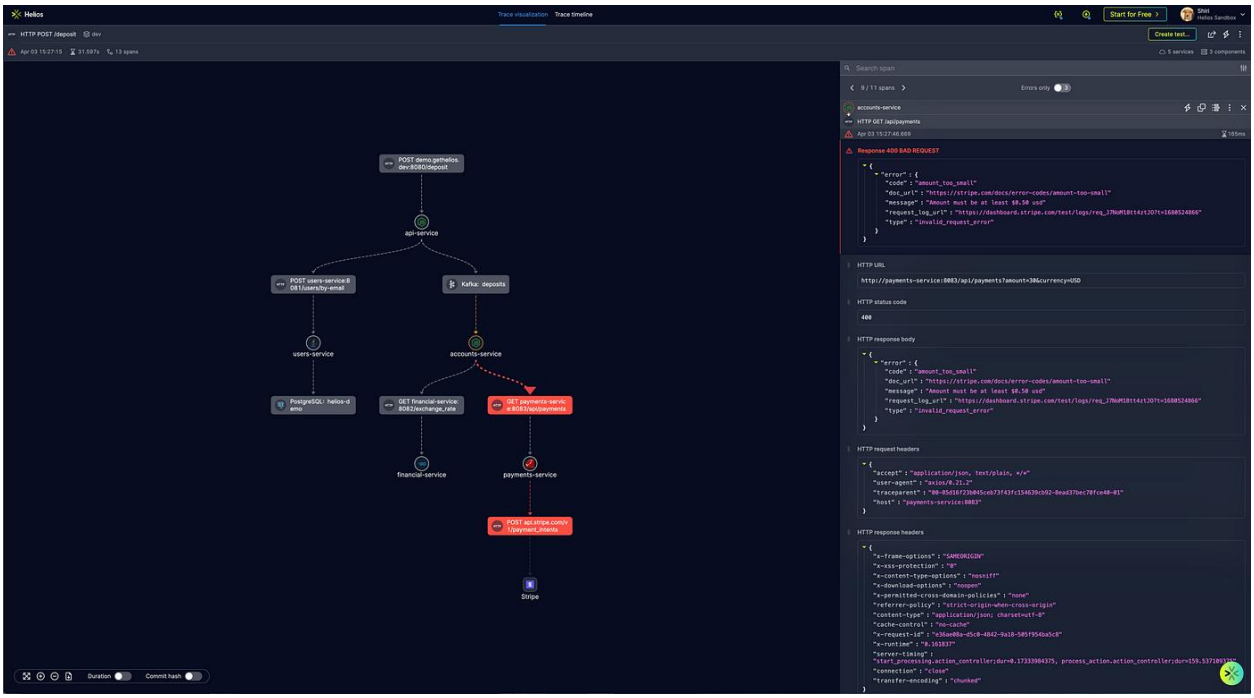
As the request moves from one component to another, the trace context is also propagated. This allows all the components involved in the request to add their trace data to the request's trace context.

Trace data is collected and stored

A tracing data collector receives the trace data from each component and service involved in the request and stores it in a distributed datastore such as Elasticsearch or Cassandra.

Trace data is visualized and analyzed

Developers can use trace visualizers and analysis tools to view the trace data and identify any performance issues or bottlenecks in the system. Below is a screen from Helios dataflow visualization.



The above distributed tracing technique provides a way to monitor and debug complex, distributed systems and help simplify and visualize customer journey within a complex microservices landscape contributing towards customer success.

Benefits of Distributed Tracing: -

The benefits of distributed tracing for software development teams are numerous.

1. Distributed tracing radically improves developer productivity and output by drastically reduce time spent debugging and troubleshooting issues with your systems. It does makes it easy to understand the behavior of distributed systems.

2. Distributed tracing works across multiple applications, programming languages, and transports. For Ex - Ruby on Rails applications can propagate traces to .NET applications over HTTP, RabbitMQ, WebSockets, or other transports and all of the relevant information can still be uploaded, decoded, and visualized by the same tracing engine such as Zipkin.

3. Distributed tracing can also help improve time and speed to market by enabling correlation between feature delivery with customer servicing performance.

4. Distributed tracing also facilitates excellent cross-team communication and cooperation. It eliminates costly data silos that could otherwise hinder developer's ability to quickly locate and fix sources of error.

REFERENCES

[1]. https://docs.openshift.com/container-platform/4.13/observability/distr_tracing/distr_tracing_arch/distr-tracing-architecture.html
[2]. Jatin Vaghela, Security Analysis and Implementation in Distributed Databases: A Review. (2019). International Journal of Transcontinental Discoveries, ISSN: 3006-628X, 6(1), 35-42. https://internationaljournals.org/index.php/ijtd/article/view/54
[3]. Sravan Kumar Pala. (2016). Credit Risk Modeling with Big Data Analytics: Regulatory Compliance and Data Analytics in Credit Risk Modeling. (2016). International Journal of Transcontinental Discoveries, ISSN: 3006-628X, 3(1), 33-39.
[4]. Anand R. Mehta, Srikarthick Vijayakumar, A Comprehensive Study on Performance engineering in nutshell, International Journal of All Research Education and Scientific Methods (IJARESM), ISSN: 2455-6211, Volume 7, Issue 7, July-2019. Available at: https://www.ijaresm.com/uploaded_files/document_file/Anand_R._Mehta_iPlu.pdf
[5]. Goswami, Maloy Jyoti. "Challenges and Solutions in Integrating AI with Multi-Cloud Architectures." International Journal of Enhanced Research in Management & Computer Applications ISSN: 2319-7471, Vol. 10 Issue 10, October, 2021.
[6]. https://www.techtarget.com/searchitoperations/tip/Considerations-when-getting-started-with-distributed-tracing
[7]. https://traefik.io/blog/the-importance-of-distributed-tracing-and-monitoring-in-a-microservice-architecture/
[8]. Sravan Kumar Pala, "Synthesis, characterization and wound healing imitation of Fe3O4 magnetic nanoparticle grafted by natural products", Texas A&M University - Kingsville ProQuest Dissertations Publishing, 2014. 1572860. Available online at: https://www.proquest.com/openview/636d984c6e4a07d16be2960caa1f30c2/1?pq-origsite=gscholar&cbl=18750
[9]. Anand R. Mehta, Srikarthick Vijayakumar, DevOps in 2020: Navigating the Modern Software Landscape, International Journal of Enhanced Research in Management & Computer Applications ISSN: 2319-7471, Vol. 9 Issue 1, January, 2020. Available at: https://www.erpublications.com/uploaded_files/download/anand-r-mehta-srikarthick-vijayakumar_THosT.pdf
[10]. https://middleware.io/blog/what-is-distributed-tracing/
[11]. Goswami, Maloy Jyoti. "Utilizing AI for Automated Vulnerability Assessment and Patch Management." EDUZONE, Volume 8, Issue 2, July-December 2019, Available online at: www.eduzonejournal.com
[12]. Bharath Kumar. (2021). Machine Learning Models for Predicting Neurological Disorders from Brain Imaging Data. Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal, 10(2), 148–153. Retrieved from https://www.eduzonejournal.com/index.php/eiprmj/article/view/565
[13]. Jatin Vaghela, A Comparative Study of NoSQL Database Performance in Big Data Analytics. (2017). International Journal of Open Publication and Exploration, ISSN: 3006-2853, 5(2), 40-45. https://ijope.com/index.php/home/article/view/110
[14]. https://guides.micronaut.io/latest/micronaut-microservices-distributed-tracing-xray-maven-java.html
[15]. Sravan Kumar Pala, "Detecting and Preventing Fraud in Banking with Data Analytics tools like SASAML, Shell Scripting and Data Integration Studio", *IJBMV*, vol. 2, no. 2, pp. 34–40, Aug. 2019. Available: https://ijbmv.com/index.php/home/article/view/61