Microservices: Using Distributed Tracing for Monitoring & Troubleshooting

Aditya Shrivatri

MS, Software Engineering, University of Detroit Mercy, Detroit, Michigan, USA

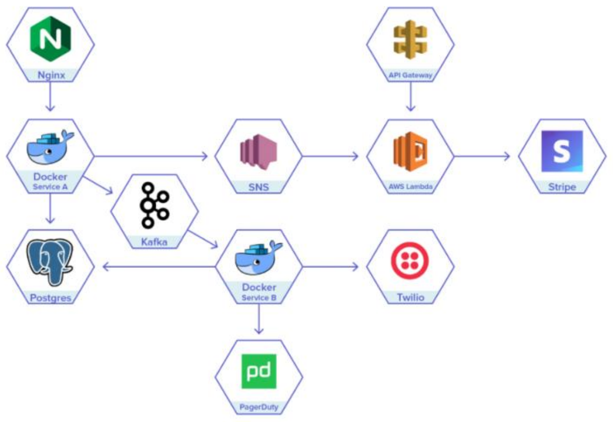
Abstract: ***Modern applications can be found everywhere today. Distributed microservices, cloud-native, managed resources, and serverless are parts of this complex whole. But how can we keep track of so many elements in our production environments? In these distributed environments, microservices communicate with each other in different ways: synchronous and asynchronous. Distributed tracing has become a crucial component of observability both for performance monitoring and troubleshooting. This paper articulates the instrumentation, distributed tracing, and modern distributed applications.***

Keywords: Microservice; Monitoring and Distributed tracing

1. Introduction

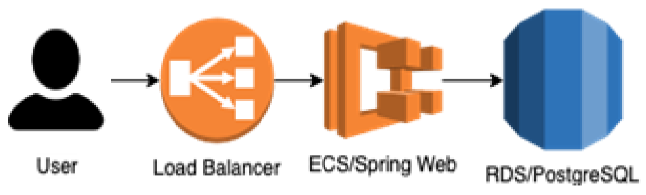
Auto scaling based on the load fluctuations is one of the biggest benefits of using micro service architecture [1] [4]. To gain the benefits of auto scaling the services have to be segregated from one another using the clear isolation techniques. Not just the services but also the databases should be isolated from one another. This has given the way to Database per service pattern as proposed by researchers like Chris Richardson and others [2][3]. With the distributed micro service architecture, it helps not in the separation of concerns but also in approaching the technical and design challenges involved in each service and database as a totally independent application. Though it is very helpful design pattern, this level of independence can cause issues in other areas. The database transaction management is one big challenge faced by the applications working on distributed microservice framework [5]. Researches likeChaitanya K Rudrabhatla [6] and others have provided solutions for solving these complex problems related to distributed transaction roll backs using the saga patterns [7]. There could be other logical issues in routing, health checks and management, including service exposition (API), inter-service communication, and infrastructure deployment
Tracing is a way of profiling and monitoring events in applications. These problems were addressed by researchers like Santos, Nuno & Ferreira, Nuno & Pereira, Manuel & Salgado, Carlos & Morais, Francisco & Melo, Mónica & Silva, Sara & Martins, Raquel & Pereira, Marco & Rodrigues, Helena & Machado, Ricardo [8]. However, even with all the research and advancements, there still are many areas which can be problematic with micro service frameworks. Isolation of failure and narrowing down the area of issue is one of the biggest concern for the developers and programming community. This problem is further aggravated by the distributed design where services can spread across multiple containers in the cloud [9]. To solve this problem, tracing mechanisms can be very helpful. With the right information, a trace can reveal the performance of critical operations. How long does a customer wait for an order to be completed? It can also help to a breakdown of our operations to our database, APIs, or other microservices.

Distributed tracing is a new form of tracing that adapted better to microservice based applications. It allows engineers to see traces from end to end, locate failures, and improve overall performance. Instead of tracking the path within a single application domain, distributed tracing follows a request from start to end.For example, a customer makes a request on our website and then we update the item suggestion list. As the request spans across multiple resources, distributed tracing takes into account the services, APIs, and resources it interacts with.



2. Automated microservices instrumentation

Exploring distributed traces might sound simple but collecting the right traces with the right context will require considerable time and efforts [10]. Let's follow an example where we got an e-commerce website that updates our database with purchases:



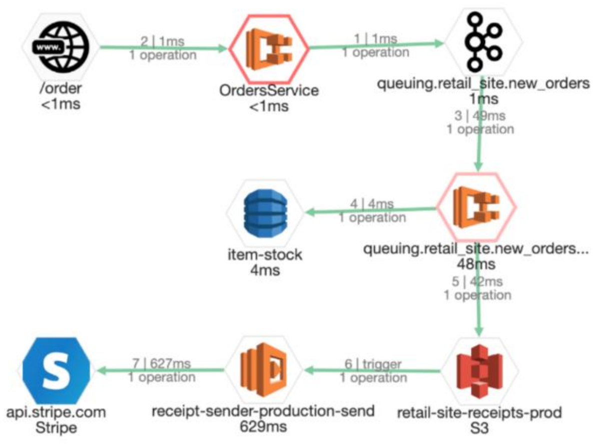
In this example, which is not distributed, to create an interesting trace, we will need to collect the following information:

a) HTTP request details:
   - URL
   - Headers
   - The ID of the user
   - Status code
b) Spring Web:
   - Matched route and function
   - Request params
   - Process duration
c) RDS database:
   - Table name
   - Operation (SELECT, INSERT, …)
   - Duration
   - Result

To capture this information we can either do it manually before and after every operation that we make in our code or automatically instrument it into common libraries.

By "automated instrumentation," we mean "hooking" into a module. For example, every time we make a GET request with "Apache HttpClient," there will be a listener. It will extract and store this information as part of the "trace."

Collecting this information manually using logging is not recommended since they are not structured well. Using a more standard way, like OpenTracing, will allow us to filter out relevant traces. We will also have the option to present them nicely in many tools. This kind of instrumentation requires heavy lifting. It involves integrating to our libraries, as well as constant maintenance to support our dynamic environments.



3. Standards and Tools

OpenTracing

Microservices standards and tools that can help us to get started with our first distributed traces. The first pioneer was OpenTracing, which is a new, open distributed tracing standard for applications and OSS packages.

Using OpenTracing, developers can collect traces into spans, and store extra context (data) to each one of them.

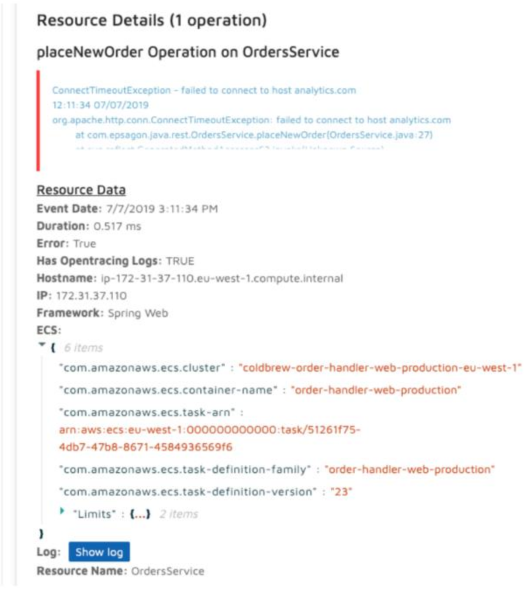
```
        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
 [Span B]      [Span C] ←←←(Span C is a `ChildOf` Span A)
     |             |
 [Span D]      +---+-------+
               |           |
           [Span E]    [Span F] >>> [Span G] >>> [Span H]
                                       ↑
                                       ↑
                                       ↑
                         (Span G `FollowsFrom` Span F)
```

Spans can have a relation – `child of` or `follows from`. These relations can help us get a better understanding of performance implications.

To trace a request across distributed microservices spans, we must implement the inject/extract mechanism to inject a unique "transaction ID." Then we would extract it on the receiving service. Note that a request can travel between microservices in HTTP requests, message queues, notifications, sockets, and more.

Managed solution

Ultimately, we might want to consider an automated distributed tracing solution. Epsagon, for example, uses automated instrumentation to provide microservices performance monitoring and visualization of requests and errors in an easier way:



Resource Details (1 operation)

placeNewOrder Operation on OrdersService

```
ConnectTimeoutException - failed to connect to host analytics.com
12:11:34 07/07/2019
org.apache.http.conn.ConnectTimeoutException: failed to connect to host analytics.com
      at com.epsagon.java.rest.OrdersService.placeNewOrder(OrdersService.java:27)
```

Resource Data
Event Date: 7/7/2019 3:11:34 PM
Duration: 0.517 ms
Error: True
Has Opentracing Logs: TRUE
Hostname: ip-172-31-37-110.eu-west-1.compute.internal
IP: 172.31.37.110
Framework: Spring Web
ECS:

```
{ 6 items
   "com.amazonaws.ecs.cluster" : "coldbrew-order-handler-web-production-eu-west-1"
   "com.amazonaws.ecs.container-name" : "order-handler-web-production"
   "com.amazonaws.ecs.task-arn" :
   arn:aws:ecs:eu-west-1:000000000000:task/51261f75-
   4db7-47b8-8671-4584936569f6
   "com.amazonaws.ecs.task-definition-family" : "order-handler-web-production"
   "com.amazonaws.ecs.task-definition-version" : "23"
   "Limits" : {...} 2 items
}
```

Log: Show log
Resource Name: OrdersService

A managed solution for distributed tracing provides the following benefits:

- Traces are being collected automatically without code changes.
- Visualizing traces and service maps with metrics and data.
- Query data and logs across all traces.

4. Conclusion

Distributed tracing is crucial for understanding complex, microservices applications. Without it, teams can be blind into their production environment when there is a performance issue or other errors.Although there are standards for implementing, collecting, and presenting distributed traces, it is not that simple to do manually. It involves a lot of effort to get up and running. Leveraging automated tools or managed solutions can cut down the level of effort and maintenance, bringing much more value to your business.

References

[1] S. Newman, Building Microservices. " O'Reilly Media, Inc.", 2015.
[2] Messina, Antonio & Rizzo, Riccardo & Storniolo, Pietro & Tripiciano, Mario & Urso, Alfonso. (2016). The Database-is-the-Service Pattern for Microservice Architectures. 9832. 223-233. 10.1007/978-3-319-43949-5_18.
[3] Chris Richardson – “microservices.io/patterns/data/database-per-service”
[4] Chaitanya K Rudrabhatla. A Systematic Study of Micro Service Architecture Evolution and their Deployment Patterns. International Journal of Computer Applications 182(29):18-24, November 2018.
[5] Elkholy, Mohamed & Elfatatry, Ahmed. (2019). Framework for Interaction Between Databases and Microservice Architecture. IT Professional. 21. 57-63. 10.1109/MITP.2018.2889268.
[6] Chaitanya K. Rudrabhatla, “Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture” International Journal of Advanced Computer Science and Applications(IJACSA), 9(8), 2018. http://dx.doi.org/10.14569/IJACSA.2018.090804
[7] Pedro Valderas, Victoria Torres, Vicente Pelechano, A microservice composition approach based on the choreography of BPMN fragments, Information and Software Technology, Volume 127,2020,106370,ISSN 0950-5849,https://doi.org/10.1016/j.infsof.2020.106370.
[8] Santos, Nuno & Ferreira, Nuno & Pereira, Manuel & Salgado, Carlos & Morais, Francisco & Melo, Mónica & Silva, Sara & Martins, Raquel & Pereira, Marco & Rodrigues, Helena & Machado, Ricardo. (2019). A logical architecture design method for microservices architectures. 145-151. 10.1145/3344948.3344991.
[9] Christian Esposito, Aniello Castiglione, Kim-Kwang Raymond Choo, "Challenges in Delivering Software in the Cloud as Microservices", Cloud Computing IEEE, vol. 3, no. 5, pp. 10-14, 2016.
[10] André Pascoal Bento, “Observing and Controlling Performance in Microservices”