How to Cite:
Thakur, A., & Chandak, M. B. (2022). A review on opentelemetry and HTTP implementation. *International Journal of Health Sciences*, *6*(S2), 15013–15023. https://doi.org/10.53730/ijhs.v6nS2.8972

A review on opentelemetry and HTTP implementation

Aadi Thakur
Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management Nagpur, India

M. B. Chandak
Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management Nagpur, India

***Abstract*---**Cloud-native is a strategy for developing and running applications that take advantage of the benefits of the cloud computing model. Companies building and operating applications by using a cloud-native architecture bring fresh ideas to the market faster and address customer demands more quickly. In the face of failures and extremely unpredictable demand, today's cloud-native application ecosystem is becoming highly dispersed, diversified, and complicated, making it impossible to forecast its behaviour. Observability is growing rapidly as a critical skill for monitoring and managing cloud-native applications in order to assure client satisfaction. Cloud-native Computing Foundation (CNCF) has introduced OpenTelemetry as its incubating project, which is an observability framework for cloud-native software. OpenTelemetry, which is an open-source framework for distributed tracing, has emerged as the industry's leading standard for distributed tracing. This paper focuses on providing a systematic literature review of various services provided by OpenTelemetry.

***Keywords*---**observability, open telemetry, distributed systems, cloud-native technology, telemetry data.

Introduction

Data collection is inexpensive, but not having it when you need it can be costly. Amazon was down for 20 minutes in March 2016, resulting in a $3.75 million revenue loss. In January 2017, Delta Airlines experienced a system outage, which led to the cancellation of more than 170 flights and an $8.5 million loss. In both cases, if we had the correct amount of data, we could have predicted or recovered from such incidents as soon as they occurred by finding the fundamental cause.

We can make improved decisions if we have more data [10]. A distributed system is the solution of a problem that has been divided into a number of parts, also known as subproblems; these subproblems are solved individually, typically by separate computers/processors. The outcomes of these solved subproblems are correctly reassembled to form the final alternative to the initial larger problem.

Distributed and multilingual architectures are the norm in cloud-native technology. Distributed architectures introduce a series of operational challenges, including how to quickly resolve availability and performance issues. These complexities have resulted in the emergence of observability. Popular web, consumer software systems, and services (for example Amazon, Google, Netflix, and Twitter) are microservices that are designed and run on common infrastructures. In today's world, users expect more interactive enrichment as well as an evolving user experience across a wide range of client devices [11]. Developers and organizations want to be able to update their services on a regular basis. As a result, dependence on monolithic applications is no longer practical.

Microservices are a type of distributed architecture in which large and complex application programs are made up of one or more multiple programs. It is proposed as a solution to the problems associated with traditional monolithic applications. Flexibility, scalability, decentralized management, and independence are key characteristics of microservice architecture. As there are numerous benefits to implementing microservices, it is now being used by many tech companies, including Amazon, Netflix, and Twitter. Because microservices can now solve many business challenges and are still growing, it is reasonable to believe that microservices are the future of distributed systems.

Observability is the capability to measure a system's present state based on data it produces, such as logs, metrics, and traces. The goal of observability is to figure out what's going on across all of these environments and systems so that you can identify and resolve these issues to keep your systems efficient and reliable [1]. The concept of observability, borrowed from control theory, telemetry acquired from the system at run time improves visibility into analyzing the complicated behaviour of software. Observability extends past conventional black-box tracking to give more background information about a system's accuracy and execution. Its purpose is to shorten the time it takes to gain insight into what's happening in the network and why it's happening.

Observability is the process of instrumenting systems in order to collect valuable information that tells you when and why systems behave in a certain way. When a system fails, you should first examine the telemetry data to determine why the error occurred. The following telemetry data, known as the pillars of observability, are critical to achieving observability in distributed systems: Logs, Metrics, and Traces [2].

LOGS

A log is a timestamped text record with metadata that can be structured (recommended) or unstructured. While logs are a stand-alone data source, they can also, be linked to spans. A log is any data that is not part of a distributed

trace or metric in OpenTelemetry. Events, for example, are a subset of logs. Logs are widely used to determine the root cause of an issue and typically include data about who changed what and the outcome of the change.

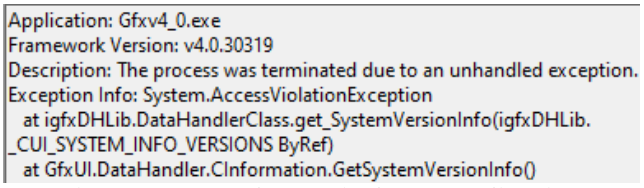

Figure 1: Logs from windows application

Failure	Definition
System Failure	A software or hardware failure is the most common cause of a system failure. A system failure always tends to result in the loss of the primary memory's contents, while the secondary storage is safe. In case of a system failure, the processor does not complete the execution and the system may reboot.
Communication medium Failure	The malfunction of the communication link is the most common cause of a communication medium failure. A possible scenario is a website within one network attempting to communicate with some other operational website inside the network but being unable to do so.
Secondary Storage Failure	A secondary storage failure occurs when the data that resides on the storage medium is inaccessible. Secondary storage failure can occur for a variety of reasons. Some of the most common reasons are as follows:Parity error, Node crash and Medium dirt.
Method Failure	A method failure frequently brings the distributed system to a halt. Furthermore, it causes the system to execute incorrectly or fail to execute at all. During a method failure, a system could go into a deadlock state or commit protection violations.

Table 1. Metrics

A metric is an evaluation of a service that is recorded at runtime. Methodically, the instant one of these measured data is recorded is known as a metric event, which includes not only the assessment itself but also the time it was captured and any related metadata. Metrics have been applied in the following ways: (i) to create alerts whenever an unexpected system state occurs, and (ii) for quantitative and real-time reporting queries. Small concerns on the most recent statistics are used to generate alerts. Fig. 2 gives the metrics as we can see in our windows system for the application running.

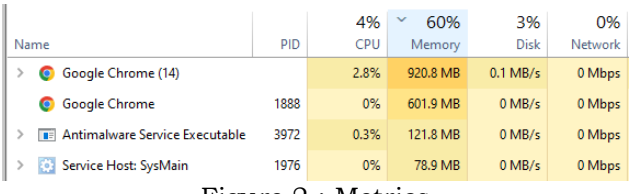

Figure 2 : Metrics

Traces

Traces follow the progress of a single request, known as a trace, as it is managed by the services that represent an application. A user or an application may initiate the request. Tracing that navigates process, network, and security borders is known as distributed tracing. A span is a sequence of tasks in a trace and therefore a trace is also a tree of many spans. Spans are entities that reflect the job completed by specific services or components as a request tends to flow through a framework. Fig. 3 demonstrates the HTTP traces when collected using tracing techniques.

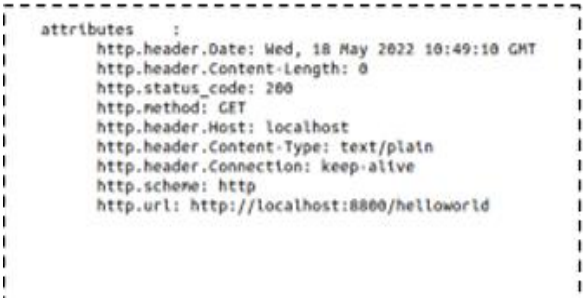

Figure 3 : Http traces

To strengthen observability products, telemetry data is required. Telemetry data has typically been provided by either open-source projects or commercial vendors. A lack of clear standard rules has led to a lack of data portability and it places the burden on the user to maintain the instrumentation. Fig. 4 explains the pillars of observability using logs, metrics and traces [7].

OpenTelemetry (also known as OTel) is an open-source observability framework that comprises a set of tools, APIs, and SDKs. IT teams can use OTel to instrument, generate, collect, and export telemetry data for analysis and understanding of software performance and behaviour. OpenTelemetry plays a significant role in providing a standard format for collecting and transmitting observability data. OTel, a Cloud Native Computing Foundation (CNCF) incubating project, aims to provide fully integrated sets of vendor-independent libraries and APIs — mainly for data collection and data transfer. OpenTelemetry is a merger of OpenCensus and OpenTracing, both of which were previously hosted by CNCF itself [6]. Many vendors, like Dynatrace, Lightstep, etc, have joined the project since its inception to help make a rich collection of data easier and more useable.

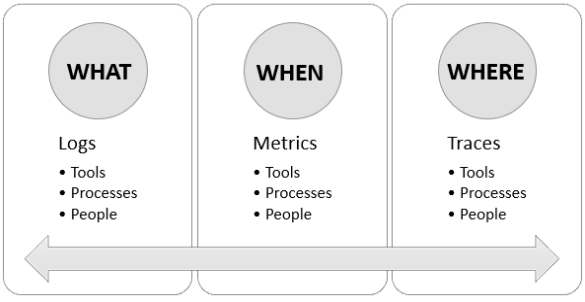

Figure 4: The three pillars of observability

The OpenTelemetry project addresses these issues by providing a single, vendor-independent solution. The project has widespread industry support and adoption, with cloud providers, distributors, and end users all on board [6]. This paper presents a literature review of OpenTelemetry and its related issues.The main objective of this paper is to review the different papers and literature available on the internet to see what are the different areas where OpenTelemetry was implemented and how it has changed the way observability frameworks work.

The remaining work in this paper is as follows. First, we discuss the research method that was used, the sources, and the criteria. Then we move on to discuss how the different papers were sorted. The main part of the paper is to see the contribution of each paper or literature towards understanding OpenTelemetry.

Research Method

The systematic review helped to perform a thorough balanced literature review, summing up the technical value of its findings. The main objective of a methodical literature review is to introduce a concise valuation concerning the research topic we choose as a result of the presentation of a dependable, accurate, and reviewed approach. The review begins with a summary of the processes that will guide the research's progress, such as research issues and techniques. According to the custom, the mandatory steps are as follows

The key issues that the paper will attempt to answer:

1. Approach that was used to search for the other authors' papers like search keywords and sources to find the papers, documents, and seminars.
2. Insertion and elimination criteria for the searched paper got over the internet.
3. Approach for information withdrawal and following procedure of obtained information.

Objective

One of the really important aspects that must be incorporated into the microservices design is observability. There are numerous observability frameworks for keeping the system in good health. Splunk, Dynatrace, and Lightstep are a few examples.

In today's scenario, everyone has their own observability pieces, big cloud services have a lot of tools that allow them to look at the data, for example, there is Splunk where the logs are structured where we can see the fancy dashboard, and their performance. But here everyone is on their own in terms of looking at the data and seeing what went wrong but as a system, we don't have a standard way to collect that data.

The main aim of OpenTelemetry is not to have a platform for observability, but instead to provide us with a conventional substrate for collecting and transmitting operational data that can be used in open-source or commercial monitoring and observational platforms. In this context, the objective of this research is to deal with the question: What is the reason that OpenTelemetry was introduced when we already had OpenCensus and OpenTracing for traces, metrics, and logs? OpenTelemetry is a reasonably followed platform for distributed tracing, it nevertheless has room for enhancements and we need to find out what are those areas.

Research Questions

1. When we already had OpenCensus and OpenTracing for observability, what was the need for OpenTelemetry?
2. In terms of observability, is there something OpenTelemetry does not provide us with?

Search approaches

After the study questions had been determined, the search approaches and the search string were demarcated. The scope of the research and the digital archives consulted were also defined. Investigation Keywords: We identify keywords based on research matters. Synonyms for the keywords did not have to be taken into account, as the names were predefined on the basis of the terms defined by the development creators. "Observability" and "OpenTelemetry" were identified as search terms.

Sources

The criteria for selecting the sources of the studies were searched on the web mechanism; Search mechanisms should enable personalized examinations by title and summary; the complete articles must be accessible for transfer via the digital library; Significance and applicability of the sources. With the search approaches defined, the following digital collections as sources: ACM Digital Library. IEEEXplore ScienceDirect, Google Scholar relevance of the sources.

Study assortment criteria

In order to exclude irrelevant studies, research work outside the scope, the subsequent elimination criteria were applied:

1. Articles that do not contain a description of the OTel framework or an application that uses observability.

2. Article without reference to a field of computer science.
3. Articles are written in a language other than English
4. Training article
5. Articles that do not (or partially discuss) OpenTelemetry or Observability

Results of the Methodical Literature Review

The conclusion of the literature review is presented in this section. The findings are focused on pre-defined replies to questions that were identified as the final outcome of our methodical literature analysis. Each part includes details that answers these questions about the study's goal. The outcomes of this review are shown in certain tables.

RQ 1. When we already had OpenCensus and OpenTracing for observability, what was the need for OpenTelemetry??

OpenCensus is a collection of libraries for different languages that enable users to gather application metrics and distributed traces and then send them in real-time to a backend of your preference. Developers and administrators can use this data to assess the safety of the application and troubleshoot issues [15]. OpenTracing is a CNCF open-source project that offers vendor-neutral APIs and instrumentation for distributed tracing. It is a vendor-independent API that allows developers to easily incorporate tracing into their codebase. It is open because it is not owned by a single company. In fact, many tracing tooling vendors are supporting OpenTracing as a standardized way to instrument distributed tracing [16].

Logs, Metrics, and Traces are the pillars of observability but individually they don't provide the value observability is supposed to deliver. To better understand this, a good analogy is to compare tyres, gasoline, and motor oil by saying that they all contribute to what we call f1 tracing. Technically, this is not incorrect, but they all work together to achieve the goal [4]. Similarly, OpenTelemetry replaced previous technologies such as OpenTracing and OpenCensus. Individually, those technologies used to cover a portion of observability, such as OpenTracing for distributed tracing and OpenCensus, will be more focused on metrics, but as we have seen before, they do not provide the value that we are attempting to achieve with observability. OpenTelemetry incorporates all of the learning experiences and even some pieces of code that have been built up over the years, and it has now grown into a new project that is supported and managed by CNCF and is free and open source.

RQ 2. 2. In terms of observability, is there something OpenTelemetry does not provide us with?

Microservices, or independent services that perform a specific core function, are used in modern cloud-native applications. A particular requirement in an use can bring up a large number of micro-services that interact with one another. In fact, it is becoming increasingly difficult to track and separate a problem, such as a

service slowdown. This is why OpenTelemetry, which is based on distributed tracing, is beneficial. It enables the observation and comprehension of a complete series of actions in a complicated interface among microservices. When the problem is network-related rather than application-related, however, OTel appears to be ineffective [7].

Assume that an HTTPS request is being monitored. Based on application traces, OpenTelemetry provides useful information at the application level. Consider the case where an unusually long execution time is noticed. It is impossible to pinpoint why this occurs. Worse, if the issue isn't with the programme, but with the link or alternate hops (for example, due to high traffic), one may question why and how the request is taking so long when everything on the application side appears to be fine [12]. A better explanation would be to demonstrate how the call continues to progress through the network hop by hop in order to identify (potential) bottlenecks [11].

To address such a problem, [Towards Cross-Layer Telemetry] introduced Cross-Layer Telemetry (Clt), which combines device, flow, packet, and application telemetry. Clt combines OpenTelemetry and network telemetry from In-Situ Operations, Administration, and Maintenance (IOAM). Fig 5 gives the architecture of Cross-Layer Telemetry. The authors have chosen Jaeger as the tracing tool for this implementation, although CLT is generic enough to accommodate any other tracing tool. The matching is accomplished by extending the IOAM headers. Lastly, a telemetry driver is responsible for retrieving data from traces and transmitting it to a collector for subsequent analysis [12].

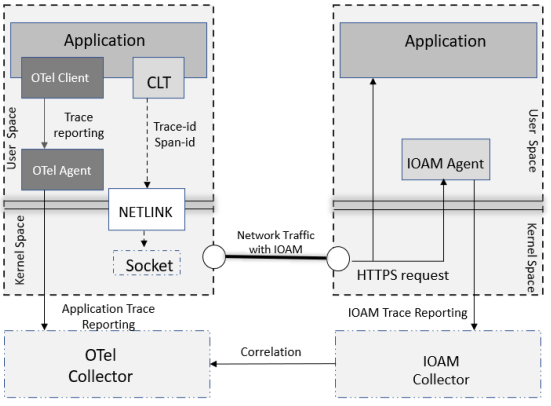

Figure 5: CLT Architecture

Implementation

OpenTelemetry is a set of tools for instrumenting, creating, gathering, and transferring telemetry data in order to manage applications and infrastructure. Metrics, logs, and traces are all part of this. Using diverse programming interfaces is encouraged by the microservices strategy. Many firms still utilize C++ to build high-performance systems instead of Java, Go or Node.js.

We have deployed an application made up of two microservices that interact through HTTP. To start, we used C++ to create two HTTP services, server and client. Both make the ping endpoints available. In addition, when server's endpoint is called, it automatically sends a ping to client .

To enable tracing, first we installed the OpenTelemetry C++ client (github).We cloned its repository into our project's common folder. Then we generate CMakeLists.txt to include both the OpenTelemetry C++ client and the OpenTelemetry C++ library.

Our cmake files contain the curl command, the executables- Client and Server and the ostream to output the logs to the console.The output from the build can be seen the figures.

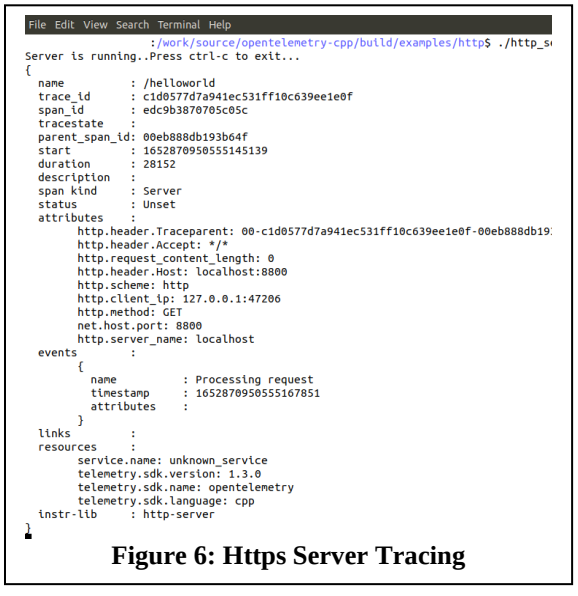

Figure 6: Https Server Tracing

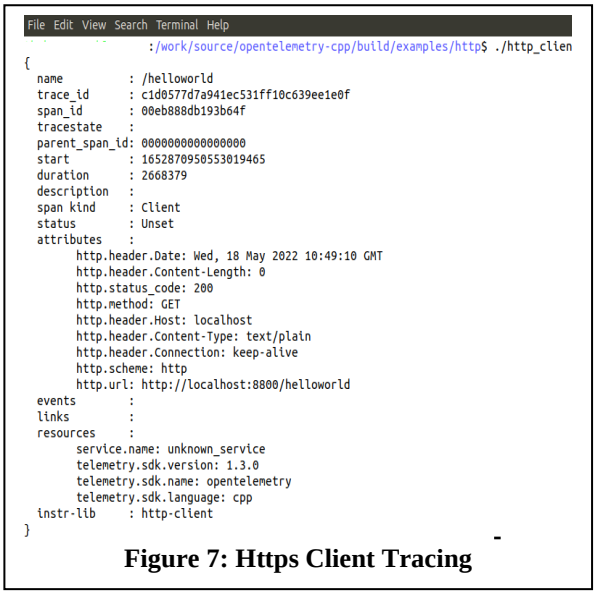

Figure 7: Https Client Tracing

Conclusion and Discussion

Failure is never predicted, and pinpointing the specific reason of post-deployed sophisticated programme issues is extremely challenging. Even for the most experienced teams it is difficult to anticipate all possible circumstances that could cause their programmes to crash or expose sensitive data. As a result, the capacity to recognise issues in real time and respond fast is critical. That's where observability and monitoring play a role, and architects who work carefully on these two tasks will enjoy the profits of a more resilient software design. The purpose of OpenTelemetry is not to have an observability platform, but instead to provide a standard platform for collecting and transmitting operational data that may be utilised in monitoring and observatory systems, both open source and commercial. We have successfully implemented tracing with the help of OpenTelemetry API. The traces gave the information about the spans, parent spans, trace-ids etc. As discussed earlier, the traces we received with

OpenTelemetry are application based and nothing on network information. Hence we cannot rely only on OpenTelemetry for analytics.

References

[1] Karumuri, Suman & Solleza, Franco & Zdonik, Stan & Tatbul, Nesime. (2021). *Towards Observability Data Management at Scale*. ACM SIGMOD Record. 49. 18-23. 10.1145/3456859.3456863.

[2] T. Salah, M. Jamal Zemerly, Chan Yeob Yeun, M. Al-Qutayri and Y. Al-Hammadi, "*The evolution of distributed systems towards microservices architecture*," 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), 2016, pp. 318-325, doi: 10.1109/ICITST.2016.7856721.

[3] Kasun Indrasiri and Prabath Siriwardena. 2018. *Microservices for the Enterprise: Designing, Developing, and Deploying (1st. ed.)*. Apress, USA.

[4] Reichelt, D.G., Kühne, S. and Hasselbring, W., 2021. *Overhead Comparison of OpenTelemetry, inspectIT and Kieker*.

[5] Ellis, A., 2022. *Emplacing New Tracing: Adding OpenTelemetry to Envoy* (Doctoral dissertation, Tufts University).

[6] Fong-Jones, L. and Parker, A., 2020. *Observing and Understanding Distributed Systems with OpenTelemetry*.

[7] Castanheira, L., Benson, T.A. and Schaeffer-Filho, A., 2020, December. *The Case for More Flexible Distributed Tracing*. In Proceedings of the Student Workshop (pp. 27-28).

[8] F. Boldrin, C. Taddia and G. Mazzini, "*Distributed Computing Through Web Browser*," 2007 IEEE 66th Vehicular Technology Conference, 2007, pp. 2020-2024, doi: 10.1109/VETECF.2007.424.

[9] N. Marie-Magdelaine, T. Ahmed and G. Astruc-Amato, "*Demonstration of an Observability Framework for Cloud Native Microservices*," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 722-724.

[10] Chumpolsathien, Nakhun. (2019). *Microservices: the Future of Distributed System*. 10.13140/RG.2.2.10322.61128/1.

[11] Gatev, R., 2021. *Introducing Distributed Application Runtime (Dapr)*. Apress.

[12] Justin Iurman, Frank Brockners, and Benoit Donnet. 2021. *Towards cross-layer telemetry*. In Proceedings of the Applied Networking Research Workshop (ANRW '21). Association for Computing Machinery, New York, NY, USA, 15–21. https://doi.org/10.1145/3472305.3472313

[13] H. Zhao and J. Bi. 2013. *Characterizing and Analysis of the Flattening Internet Topology*. In Proc. International Symposium on Computers and Communications (ISCC).

[14] OpenTelemetry. Effective Observability Requires High-Quality Telemetry. See https://opentelemetry.io/

[15] OpenCensus. Introduction, Tracing . See https://opencensus.io/introduction/

[16] Opentracing. Extended tracing with open-tracing. See https://www.dynatrace.com/support/help/extend-dynatrace/extend-tracing/opentracing/

[17] Lozano-Nieto, A., 2003. *Telemetry*. In Electrical Measurement, Signal Processing, and Displays (pp. 27-1). CRC Press.

[18] Rinartha, K., & Suryasa, W. (2017). Comparative study for better result on query suggestion of article searching with MySQL pattern matching and Jaccard similarity. In *2017 5th International Conference on Cyber and IT Service Management (CITSM)* (pp. 1-4). IEEE.

[19] Rinartha, K., Suryasa, W., & Kartika, L. G. S. (2018). Comparative Analysis of String Similarity on Dynamic Query Suggestions. In *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)* (pp. 399-404). IEEE.