International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 08 | Aug 2022 www.irjet.net p-ISSN: 2395-0072

Continuous Health Monitoring of Micro-Service based Application

Vyomikaa Basani[1], Anitha G S[2]

[1]Student, Dept. of Electrical and Electronics Engineering, RV College of Engineering, Karnataka, India
[2]Associate Professor, Dept. of Electrical and Electronics Engineering, RV College of Engineering, Karnataka, India

---

Abstract - *Popularity of developing an application using microservices architecture is gaining more attention because of smaller and faster deployments, ease of understanding, scalability, Continuous Integration, Continuous Delivery, and improved fault isolation. Though Micro-Services bring lots of advantages, it has its own challenges. Teams can easily manage and monitor individual services, but they lose sight of the global system behavior. The objective of this paper is to develop a centralized and continuous health monitoring system for an application built with Micro-Services architecture. The proposed system is developed with the help of AWS lambda that is triggered by an API Gateway for every cron schedule. The real-time health status of each service of the application is displayed in Kibana in the form of a dashboard. This developed system keeps the developers alerted whenever a particular service stops functioning.*

***Key Words*: Microservices, Serverless, AWS lambda, API Gateway, ELK Stack;**

1.INTRODUCTION

Recent developments of the cloud landscape for applications suggests there is a shift towards microservisation. The purpose of microservices is to use autonomous units that are isolated from one another and coordinate them into a distributed infrastructure by a lightweight container technology, such as Docker. Though Micro-Services/Serverless/Containers bring lots of advantages, it has its own challenges. Teams can easily manage and monitor individual components/services, but they lose sight of the global system behaviour. Traditional forms of monitoring are not suitable for microservices because there are multiple services that make up the same functionality that was previously supported by a single application. Consider a scenario when the application fails to function normally, an issue may be reported with a transaction that is distributed across several microservices, server-less functions and teams. It is difficult to differentiate the service/component that is responsible for the issue from those that are affected by it. Monitoring the health of microservices is an important part of ensuring developers are alerted quickly to interruptions in service for mission critical applications. These health checks also provide a means to keep API warm, so it is ready to service requests as quickly as possible. Hence, it is important to establish a different, easy, and effective process of achieving the distributed tracing using log collection, log aggregation and visualization.

2. LITERATURE REVIEW

The study by C. Pahl and P. Jamshidi was conducted to discuss the microservices architecture. The details of a microservice are hidden from the other microservices.[1] The services interact with each other with well-defined APIs. This reduces the number of requests made to the application. Each microservice can be developed with different programming language, and technologies. Microservices embrace the concept of decentralization as each service can be deployed and maintained by different teams.

M. Viggiato et al., presented a review paper on the use of microservices architecture in practice. In the microservices architecture, the application is composed of many independently deployable and loosely coupled smaller services.[2] Microservices provide many advantages such as scalability, maintainability, easy and faster deployment, and no commitment to a single technology stack. With the use of microservices, developers also face challenges such as monitoring of the application, and complex interaction between the services.

According to the survey conducted by J. Ghofrani and D. Lübke, one of the main challenges of building an application with microservices architecture is that its distributed nature makes it difficult to debug the issues. Monitoring of microservices based application requires larger efforts because it requires to go through huge volumes of data. Application logging and tracing allows operators to debug the errors.[3]

V. Ivanov and K. Smolander have presented a review paper on the impact of serverless on DevOps practices. From the research, the results show that the serverless approach strongly affects various automation practices such as deployment, test execution and monitoring of the application.[4] The use of serverless reduces the infrastructure cost and provides automatic scalability. It also reduces the time on the maintenance and management of servers.

H. Andi discussed the concept of serverless cloud computing framework, its benefits and usage in IT industry.[5] The analysis suggests serverless cloud computing reduces the execution time, and cost of maintenance, also it offers high security. [6] B. Choudhary et al., have developed the serverless chat application to discuss the functioning of Amazon Web Services (AWS) lambda along

with other services. The developed model did not require any maintenance or management of servers and the number of users could be augmented as required.

M. Villamizar et al., in the paper have compared Amazon Web Services with other cloud computing services based on various factors such as cost, performance and response time.[7] The cost per million requests of the architectures implemented with the cloud computing services were compared and it was found that AWS lambda can reduce the cost per scenario up to 77.08% compared to other cloud computing services. [8]-[10] For the architecture operated by AWS lambda, the response time was found to be less comparatively. From the research, the results obtained concludes better performance can be obtained at lower costs with AWS lambda.

The paper by L. Muller et al., focuses on the implementation of a file upload stream on AWS lambda to discuss the performance indicators influencing traffic on serverless computing. From the study, it was found that the performance of the lambda function is determined by the overall latency in the Round-Trip-time and their execution time. Further it was found that the latency increased with the incorporation of other cloud services in the lambda function such as API gateway. However, a serverless architecture facilitates quick deployment, better scalability, and reduced architecture costs.[11]

A.-V. Zamfir et al., presents a review paper on the need for Elasticsearch system for systems monitoring and big data analysis.[12] The paper focuses on the current state of Elasticsearch, Logstash, and Kibana (ELK) stack and the possibility of extension of the Elasticsearch system with machine learning to automate the elastic technology. The machine learning techniques can help in the root cause analysis and with further advancements, it can also suggest the possible mitigation steps based on the past events.

P. Bavaskar et al., discussed the performance of Elastic stack in log analysis for big data processing. The purpose of tracking and analyzing the logs is to find malfunctioning of a particular system. From the analysis, it can be concluded that Elasticsearch is the most suitable for data visualization as it provides advanced search capabilities, centralized data processing and aids in picturing the logs in the form of pie-charts, graphs, dashboards, etc. [13]

Elasticsearch is based on Lucene search engine that allows users to store, search and analyze big volume of data. It is built with Representational State Transfer (REST) API. According to A. Neumann et al., in the paper, a REST service is a server-client model that allows easy API usage.[14] The paper by O. V. R. Nikita Kathare and D. V. Prabhu presents a comprehensive study of Elasticsearch. It has many features such as high scalability, index management, full text search engine, high security, and availability. Elasticsearch supports various datatypes, also optimized and aggregation querying in search indices and eventual consistency.[15] According to D. Kalyani et al., Elasticsearch makes the search process faster because of its use of inverted indices. It also has a failure recovery mechanism as its architecture is distributed in nature.[16]

M. Mitra and D. Sy documented about the ELK stack. Elastic stack provides a way to consolidate storage of logs, event monitoring and report generation. Based on the study, it is found that elastic stack is well suited for time series data as it can pull events using plugins such as Elasticsearch, log4j, Kafka, HTTP, JDBC, etc. Kibana allows users to visualize the logs, from various sources, in the form of bar graphs, pie-chart etc. It also allows the users to search for the keyword for various requirements such as root cause analysis.[17] Zhao. J et al., presents management of API gateway based on Micro-Service Architecture. The API gateway provides a means of integrating various microservices and hence simplifies the interaction between client and the application.[18], [19]

3. PROPOSED HEALTH MONITORING SYSTEM

The centralized and continuous health monitoring system is developed using the lambda function provided by the Amazon Web Services and the results are visualized in the Kibana. The services are monitored continuously with the help of cron jobs.

The implementation of the system is as shown in Fig.1. The development of the continuous health monitoring system includes the following steps:

1. Querying the status of all the components of the application by creating an AWS lambda function.
2. Creation of a CI pipeline to trigger the AWS lambda function periodically.
3. Sending the obtained health status results from the lambda function to the ELK stack in the required format.
4. Generating a report using Kibana metrics that gives information about the health status of the services.

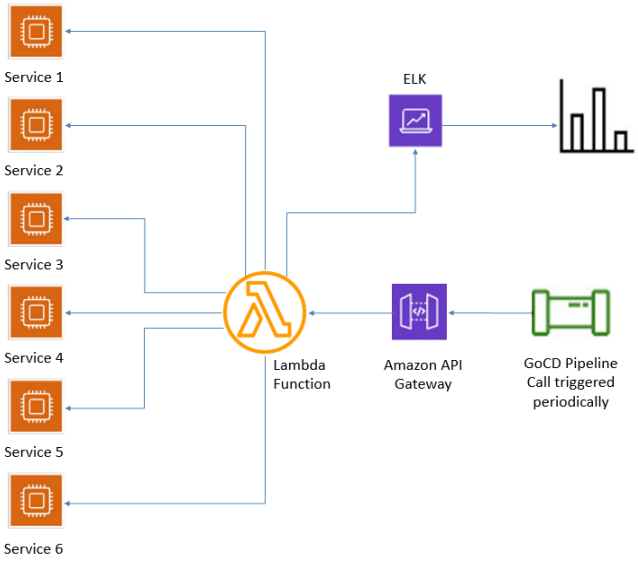

Fig -1: Project Design

3.1 Python Lambda Function

An AWS lambda function is created to get and query the health status from the services of the application with the help of the microservice endpoints. The required python libraries are imported to make HTTP requests and to send the logs to the ELK stack. An array is initialized with the microservices endpoints. These endpoints are required to make HTTP requests to check the health of a service. For each service, the Kibana logs are initialized in the form of key value pairs. There are two key value pairs: one that logs the success count and the other logs the failure count.

The function handler gets executed when the lambda is invoked. This lambda function is invoked by the API gateway which is added as a trigger. The API gateway is called by the CI pipeline and hence the lambda function is triggered periodically. A function check_endpoint is defined that checks the health of a service by taking two arguments, argument one is the service name and argument two is the microservice endpoint. If the microservice endpoint is a POST call, it makes a POST HTTP request and returns the response, similarly if the microservice endpoint is a GET call, it makes a GET HTTP request and returns the response.

Another function send_logs is defined that sends the status of the response obtained to a topic in the ELK stack in the form of key value pairs. If the service is functioning properly, the success key is appended with the value 1 and if the service is not functioning properly, then the failure key will have the value 1. Each time the handler is executed, the Kibana logs are initialized, and the health status of the services are sent to the ELK stack.

3.2 API Gateway and Cron Jobs

An API Gateway is created that allows the developers to connect non-AWS applications to AWS backend resources, such as code and servers. REST API is chosen to cache endpoint responses and to gain control over various API management capabilities such as per-client rate limiting and API keys. The API Gateway is then added as a trigger to the lambda function. A cron job is created in the EventBridge service provided by Amazon Web Services. The AWS lambda is triggered every five minutes, so the cron period is five min. The event bus selected is default and the rule type is schedule. The lambda function is then added as the target to the created cron job.

4. RESULTS

4.1 AWS Lambda Results

The lambda function is executed every five minutes. Every time the lambda gets executed; the logs are sent to the ELK stack. The output obtained in the AWS console is as shown in Fig. 2. First, the logs in the form of key value pairs are initialized. The lambda function queries the health status of each service by making HTTP requests to the microservices endpoints. Then displays the status of the response of all the services. Finally, these key value pairs that contain the information regarding the health status of the services of the application are sent to the Kibana.

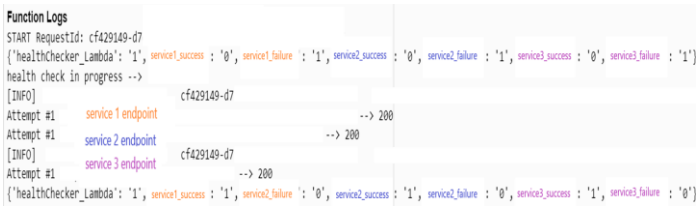

Fig -2: AWS Lambda Function Output

4.2 Kibana Dashboard

The Kibana Dashboard visualizes the health status results obtained from the AWS lambda. In the Kibana dashboard, each pie-chart shows the success rate of a service of the application over a period as shown in Fig. 3.

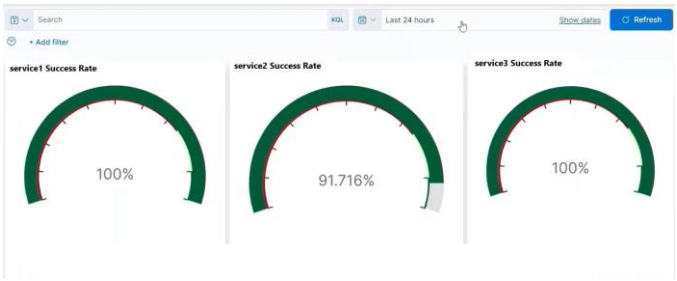

Fig -3: Success Rate of Services

The kibana dashboard also has a time series graph for each service. The time series graph for one of the services is as shown in Fig. 4. The failure count is represented by a red vertical bar and the success count is represented by a green vertical bar. It also displays the overall success rate and overall failure rate.

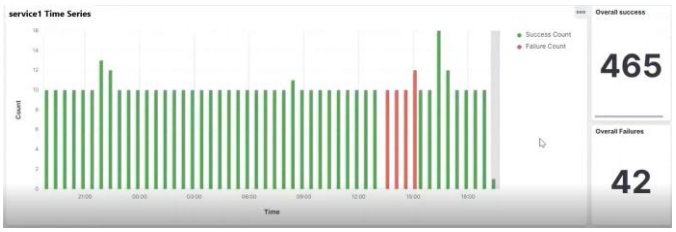

Fig -4: Time Series Graph of Service1

4. CONCLUSION

The continuous health monitoring system developed in this project provides end-to-end visibility, faster and easy debugging of customer issues and automated SLA tracking. With the developed centralized health monitoring system, the health status of each service of the application can be viewed on the Kibana dashboard in real time. It also displays the overall success rate and overall failure rate. The health status of each service can also be viewed for a particular time by applying the time filter. The continuous health monitoring system keeps the developers alerted whenever a particular component stops functioning.

The developed centralized and continuous health monitoring system can be applied to other microservices based applications. The developed continuous health monitoring system can be combined with an alerting system so that whenever a particular component stops functioning, notifications can be sent with the help of Amazon SNS. The health monitoring system can also be combined with distributed tracing to capture more granular metrics.

REFERENCES

[1] M. Viggiato, R. Terra, H. Rocha, M. Valente, and E. Figueiredo, "Microservices in practice: A survey study," Sep. 2018

[2] C. Pahl1 and P. Jamshidi, "Microservices: A Systematic Mapping Study," in *2016 International Conference on Cloud Computing and Services Science*, vol. 1, pp. 137-146, ISBN: 978-989-758-182-3.

[3] J. Ghofrani and D. Lübke, "Challenges of Microservices Architecture: A Survey on the State of the Practice," in Feb. *2018 10th Central European Workshop on Services and their Composition*, vol. 2072.

[4] V. Ivanov and K. Smolander, "Implementation of a devops pipeline for serverless applications," in 2018 Lecture notes in Computer Science Book Series, IEEE, Nov. 2018. doi: 10.1007/978-3- 030-03673-7_4.

[5] H. Andi, "Analysis of serverless computing techniques in cloud software framework," Journal of ISMAC, vol. 3, pp. 221–234, Aug. 2021. doi: 10.36548/jismac.2021.3.004.

[6] B. Choudhary, C. Pophale, A. Gutte, A. Dani, and S. Sonawani, "Case study: Use of aws lambda for building a serverless chat application," in. Jan. 2020, pp. 237–244, isbn: 978-981-15-0789-2. doi: 10.1007/978-981-15-0790-8_24.

[7] M. Villamizar, O. Garc´es, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 179–182. doi: 10.1109/CCGrid.2016.37.

[8] S. Mukherjee, "Benefits of aws in modern cloud," Mar. 2019. doi: 10.5281/zenodo.2587217.

[9] D. Rajan, "Serverless architecture - a revolution in cloud computing," Dec. 2018, pp. 88–93. doi: 10.1109/ICoAC44903.2018.8939081.

[10] C. Kotas, T. Naughton, and N. Imam, "A comparison of amazon web services and microsoft azure cloud platforms for high performance computing," in 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1–4. doi: 10.1109/ICCE.2018.8326349.

[11] L. Muller, C. Chrysoulas, N. Pitropakis, and P. Barclay, "A traffic analysis on serverless computing based on the example of a file upload stream on aws lambda," Big Data and Cognitive Computing, vol. 4, Dec. 2020. doi: 10.3390/bdcc4040038.

[12] A.-V. Zamfir, M. Carabas, C. Carabas, and N. Tapus, "Systems monitoring and big data analysis using the elasticsearch system," May 2019, pp. 188–193. doi: 10.1109/CSCS.2019.00039.

[13] P. Bavaskar, O. Kemker, A. Sinha, and M. Sabri, "A survey on: "log analysis with elk stack tool"," SSRN Electronic Journal, vol. 6, pp. 965–968, Nov. 2019.

[14] A. Neumann, N. Laranjeiro, and J. Bernardino, "An analysis of public rest web service apis," IEEE Transactions on Services Computing, vol. 14, pp. 957–970, Jul. 2021. doi: 10.1109/TSC. 2018.2847344.

[15] O. V. R. Nikita Kathare and D. V. Prabhu, "A comprehensive study of elasticsearch," International Journal of Science and Research (IJSR), vol. 10, Jun. 2021.

[16] D. Kalyani and D. Mehta, "Paper on searching and indexing using elasticsearch," International Journal of Engineering and Computer Science, 2017.

[17] M. Mitra and D. Sy, "The rise of elastic stack," Nov. 2016. doi: 10.13140/RG.2.2.17596.03203.

[18] J. Zhao, S. Jing, and L. Jiang, "Management of api gateway based on micro-service architecture," Journal of Physics: Conference Series, vol. 1087, p. 032 032, Sep. 2018. doi: 10 .1088 / 1742 - 6596/1087/3/032032.

[19] M. Tomi´c, V. Dimitrieski, M. Vje˘stica, R. Zupunski, A. Jeremi´c, and H. Kaufmann, "Towards ˇ applying api gateway to support microservice architectures for embedded systems," Mar. 2022.