CONSTANT MONITORING OF AN APPLICATION BASED ON MICRO-SERVICES

Dr. Purushottam Rohidas Patil[1], Dr. Vivek N. Waghmare[2], Mr. Mahesh Vijayrao Korde[3]

Article History: Received: 12.12.2022 **Revised:** 29.01.2023 **Accepted:** 15.03.2023

Abstract

Because of smaller and faster deployments, simplicity, scalability, continuous integration, continuous delivery, and increased fault isolation, the usage of microservices architecture in application development is gaining popularity. Micro-Services have many benefits, but they also present certain difficulties. Teams can easily manage and keep an eye on each individual service, but often lose track of the behaviour of the entire system. The goal of this study is to create a centralised, continuous health monitoring system for a micro-services-based application. The suggested system is created using Amazon Lambda, which is launched via an API Gateway in response to each cron schedule. With Kibana, a dashboard displaying the current health condition of each service in the application is visible. This created system keeps the developers informed if a certain service fails.

Keywords: API Gateway, Microservices, Serverless, AWS Lambda

[1]Associate Professor, Department of Computer Science, School of Engineering and Technology, Sandip University, Nashik, Maharashtra 422213
[2]Associate Professor, Department of Computer Engineering, Sandip Institute of Technology & Research Centre, Nashik, Maharashtra 422213
[3]Assistant Professor, Department of Computer Engineering, Sandip Institute of Engineering and Management, Nashik, Maharashtra 422213

Email: [1]purushottam.patil@sandipuniversity.edu.in, [2]vivek.waghmare@sitrc.org, [3]mahesh.korade@siem.org.in

DOI: 10.31838/ecb/2023.12.s3.118

1. Introduction

Current changes in the cloud application ecosystem indicate a move towards mercerization. Using independent components that are separated from one another and coordinating them into a distributed architecture using a lightweight container technology, such as Docker, is the goal of microservices. Although Micro-Services, Serverless, and Containers have many benefits, they also have their own set of difficulties. Teams can easily manage and keep an eye on individual services and components, but often lose track of the behavior of the entire system. Because there are several services that make up the same functionality that was previously covered by a single application, traditional approaches of monitoring are not appropriate for microservices. Think of a situation when the application stops functioning as it should, and a problem is reported with a transaction that is split up among several microservices, serverless processes, and teams. It is challenging to distinguish between persons who are impacted by the problem and the service or component that is to blame. In order to make sure that developers are rapidly informed of service outages for mission-critical systems, it is crucial to monitor the health of microservices. These health checks offer a way to keep the API warm so that it is prepared to respond to queries as soon as feasible. As a result, it's crucial to build a unique, simple, and efficient method of performing distributed tracing through log collecting, log aggregation, and log visualization.

2. Literature Review

The purpose of the study by C. Pahl and P. Jamshidi was to talk about the architecture of microservices. A microservice keeps its specifics a secret from other microservices.(Ghofrani & Lübke, 2018; Viggiato et al., 2018) The services communicate with one another via clear APIs. As a result, fewer queries are made of the programme. Several programming languages and technologies might be used to construct each microservice. Due to the fact that each service may be deployed and managed by a distinct team, microservices embrace the idea of decentralisation.
A review study on the use of the microservices architecture in practise was provided by M. Viggiato et al. The application is divided into several smaller, loosely connected, independently deployable services under the microservices design.(Pahl & Jamshidi, 2016) Microservices have a number of benefits, including scalability, maintainability, quick and simple deployment, and no lock-in to a particular technology stack. Developers must deal with issues like monitoring the application and intricate service interactions while using microservices.
According to a poll done by J. Ghofrani and D. Lübke, one of the biggest problems with developing an application using a microservices architecture is that it is hard to debug problems because of how scattered the design is. Because monitoring microservices-based applications entails sifting through enormous amounts of data, it takes more work. With application logging and tracing, operators may troubleshoot issues.(Ghofrani & Lübke, 2018)
In their review article, V. Ivanov and K. Smolander discuss how serverless computing has affected DevOps procedures. The study's findings indicate that the serverless method has a significant impact on a number of automated techniques, including application deployment, test execution, and monitoring.(Ivanov & Smolander, 2018) Using serverless technology lowers infrastructure costs and offers built-in scalability. Also, it cuts down on the time needed for server management and maintenance.
H. Andi spoke on the idea of a serverless cloud computing architecture, as well as its advantages and applications in the IT sector.(Andi, 2021) According to the investigation, serverless cloud computing provides good security while reducing execution time and maintenance costs.(Choudhary et al., 2020) B. Choudhary et al. created the serverless chat application to talk about how other services, such as Amazon Web Services' (AWS) lambda, operate. The created concept could be expanded in terms of users as needed and did not require any server upkeep or management.
In the article, M. Villamizar et al. compared Amazon Web Services to other cloud computing services based on a number of criteria, including price, performance, and response time.(Villamizar et al., 2016) After comparing the cost per million requests of the cloud computing service-implemented designs, it was discovered that Amazon Lambda may lower costs per scenario by up to 77.08% when compared to other cloud computing services.(Villamizar et al., 2016, 2017) It was discovered that the response time was lower in comparison with the architecture run using Amazon lambda. The study's findings suggest that with Amazon lambda, higher performance may be attained at reduced prices.
The implementation of a file upload stream on AWS Lambda is the main topic of the work by L. Muller et al., which also discusses the performance metrics affecting traffic on serverless computing. The study discovered that the lambda function's execution time and overall round-trip latency are what affect how well it performs. Furthermore, it was discovered that adding other cloud services to the lambda function, such an API gateway, increased latency. A serverless architecture, on the other hand, enables faster deployment, higher

scalability, and lower architectural costs.(Mukherjee, 2019)

A review study on the necessity of the Elasticsearch system for system monitoring and large data analysis is presented by (Rajan, 2018) The present status of the Elasticsearch, Logstash, and Kibana (ELK) stack and potential machine learning extensions to the Elasticsearch system to automate elastic technology are the main topics of this study. Machine learning techniques can aid in root cause investigation and, with further development, they may even be able to recommend potential mitigation measures based on prior incidents.

The effectiveness of Elastic Stack in log analysis for large data processing was explored by P. Bavaskar et al. Finding system malfunctions is the goal of monitoring and analysing the logs. Based on the investigation, it can be said that Elasticsearch is most suited for data visualization since it offers centralised data processing, powerful search capabilities, and helps visualise logs using pie charts, graphs, dashboards, etc. (Van et al., 2015)

Users may store, search, and analyse large volumes of data with Elasticsearch, which is based on the Lucene search engine. Representational State Transfer (REST) API was used in its construction. A. Neumann et al. claim in their article that a REST service is a server-client approach that makes it simple to use an API.(Van et al., 2015) The research of Elasticsearch by O. V. R. Nikita Kathare and D. V. Prabhu is thorough. It offers various characteristics, including full text search engine, index management, high availability, and scalability. Elasticsearch enables a wide range of datatypes in addition to querying that is optimised and aggregated in search indices and eventual consistency.(Muller et al., 2020) Users may store, search, and analyse large volumes of data with Elasticsearch, which is based on the Lucene search engine. Representational State Transfer (REST) API was used in its construction. A. Neumann et al. claim in their article that a REST service is a server-client approach that makes it simple to use an API.(Muller et al., 2020) The research of Elasticsearch by O. V. R. Nikita Kathare and D. V. Prabhu is thorough. It offers various characteristics, including full text search engine, index management, high availability, and scalability. Elasticsearch enables a wide range of datatypes in addition to querying that is optimised and aggregated in search indices and eventual consistency.

Users may store, search, and analyse large volumes of data with Elasticsearch, which is based on the Lucene search engine. Representational State Transfer (REST) API was used in its construction. A. Neumann et al. claim in their article that a REST service is a server-client approach that makes it simple to use an API.(Zamfir et al., 2019) The research of Elasticsearch by O. V. R. Nikita Kathare and D. V. Prabhu is thorough. It offers various characteristics, including full text search engine, index management, high availability, and scalability. Elasticsearch enables a wide range of datatypes in addition to querying that is optimised and aggregated in search indices and eventual consistency.

3. Suggested approach for tracking health

The lambda function offered by Amazon Web Services was used to construct the centralized and continuous health monitoring system, and Kibana was used to display the data. Cron jobs are used to continually monitor the services.

The system is implemented as seen in Fig. 1. The following stages are involved in creating the continuous health monitoring system:

1. By building an Amazon lambda function, you may check the status of every component of the programmed.
2. Building a CI pipeline to launch the Amazon lambda function on a regular basis.
3. Delivering to the ELK stack the results of the lambda function's health status query in the appropriate manner.
4. Producing a report using Kibana metrics that details the services' state of health.

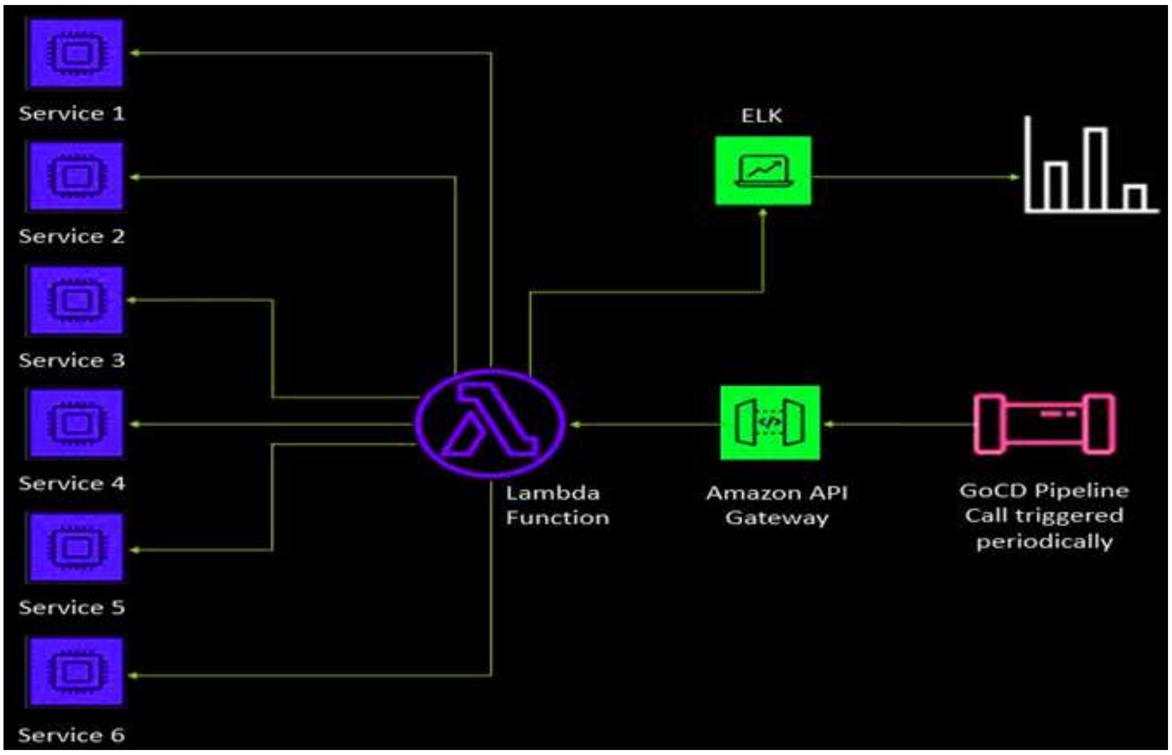


Figure 1: Project Planning

3.1 Lambda Function in Python

With the aid of the microservice endpoints, an Amazon lambda function is developed to obtain and query the health status from the application's services. The necessary Python libraries are imported in order to transmit the logs to the ELK stack and make HTTP queries. The endpoints for the microservices are used to initialise an array. These endpoints must send HTTP queries in order to verify the status of a service. The Kibana logs are initialised as key value pairs for each service. One key value pair logs the success count, while the other logs the failure count. There are two key value pairs.

When the lambda is called, the function handler is also called. The API gateway, which is inserted as a trigger, calls this lambda function. The CI pipeline calls the API gateway, which causes the lambda function to be regularly activated. The method check endpoint is defined, and it takes two arguments: the name of the service and the endpoint for the microservice. This function verifies the health of the service. When a GET call is made to the microservice endpoint, a GET HTTP request is made and the response is returned. Similarly, when a POST call is made to the microservice endpoint, a POST HTTP request is made and the response is returned.

Another function, send logs, is defined to send key-value pairs containing the status of the answer received to a topic in the ELK stack. The value 1 is attached to the success key if the service is operating correctly, and the value 1 is appended to the failure key if the service is not operating correctly. The Kibana logs are initialised each time the handler is run, and the ELK stack is informed of the services' current state of health.

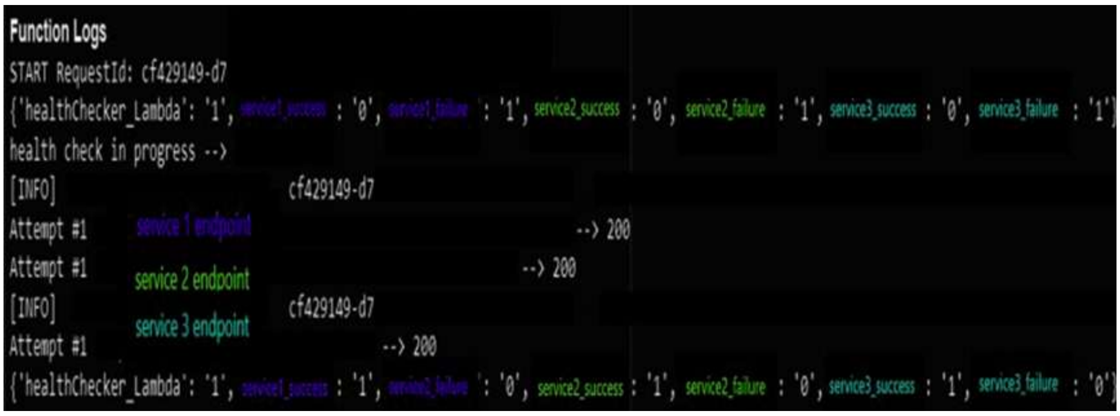
3.2.1 Cron Jobs and API Gateway

Developers may link non-AWS apps to Amazon backend resources, such as code and servers, by building an API Gateway. The REST API is used in order to regulate different API management features, such as per-client rate limitation and API keys, and to cache endpoint answers. The lambda function is then given the API Gateway as a trigger. The Event Bridge service offered by Amazon Web Services creates a cron task. The cron period is five minutes since the Amazon lambda is called every five minutes. The rule type is schedule, and the event bus is set to default. The target of the newly formed cron job is then changed to include the lambda function.

4. Results and Discussion

4.1 Outcomes from AWS Lambda

Every five minutes, the lambda function is carried out. The logs are transmitted to the ELK stack each time the lambda is invoked. The results displayed in the AWS interface are as seen in Fig. 2. The logs are initialised first in the form of key-value pairs. By sending HTTP requests to the microservices endpoints, the lambda function checks the state of each service's health. The status of each service's response is then shown. Ultimately, these key value pairs that include the data pertaining to the functionality of the application's services are transmitted to Kibana.



```
Function Logs
START RequestId: cf429149-d7
{'healthChecker_Lambda': '1', service1_success : '0', service1_failure : '1', service2_success : '0', service2_failure : '1', service3_success : '0', service3_failure : '1'}
health check in progress -->
[INFO]                    cf429149-d7
Attempt #1      service 1 endpoint                                  --> 200
Attempt #1      service 2 endpoint                              --> 200
[INFO]                    cf429149-d7
Attempt #1      service 3 endpoint        --> 200
{'healthChecker_Lambda': '1', service1_success : '1', service1_failure : '0', service2_success : '1', service2_failure : '0', service3_success : '1', service3_failure : '0'}
```

Figure 2: Amazon Lambda Function Result

4.2. Kibana Dashboard

The Amazon Lambda health status results are seen through the Kibana Dashboard. As seen in Fig. 3, each pie-chart in the Kibana dashboard displays the success rate of a service provided by the application over time.

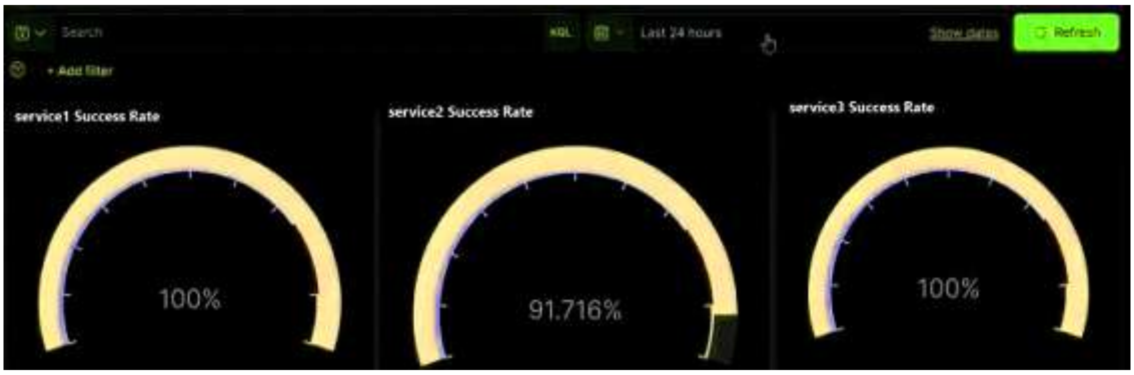


Figure 3: Services' Success Rate

For each service, there is a time series graph on the Kibana dashboard. Figure 4 depicts the time series graph for one of the services. A red vertical bar represents the number of failures, and a green vertical bar the number of successes. Moreover, the total success and failure rates are shown.

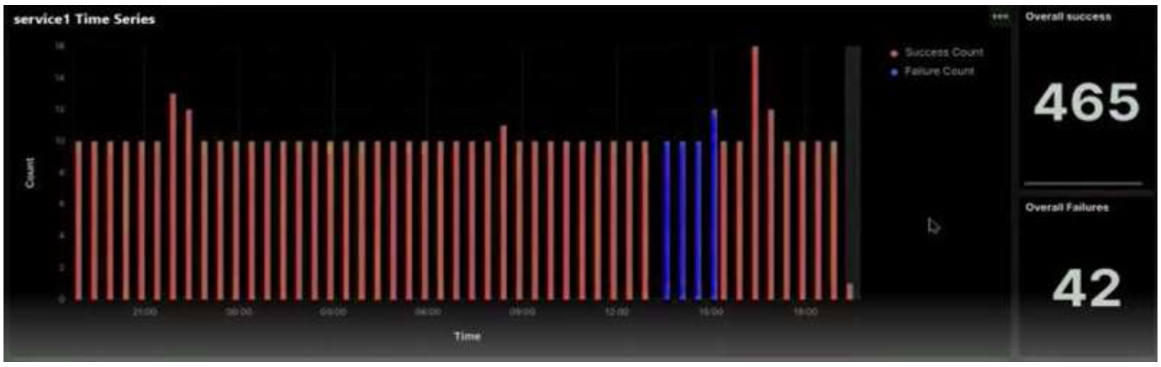


Figure 4: Service Time Series Graph

5. Conclusion

This project's continuous health monitoring solution offers end-to-end visibility, quick and simple customer issue troubleshooting, and automatic SLA tracking. The health condition of each service of the application can be seen in real time on the Kibana dashboard thanks to the established centralised health monitoring system. Moreover, the total success and failure rates are shown. By using the time filter, it is also possible to see the health condition of each service for a certain period of time. When a certain component stops working, the developers are informed via the system's continuous health monitoring.

Other microservices-based apps can use the created centralised and continuous health monitoring system. When an individual component stops working, alerts may be issued via Amazon SNS thanks to the created continuous health monitoring system that can be integrated with an alerting system. To collect more precise measurements, the health monitoring system can potentially be integrated with distributed tracing.

6. Reference

Andi, H. K. (2021). Analysis of serverless computing techniques in cloud software framework. Journal of IoT in Social, Mobile, Analytics, and Cloud, 3(3), 221–234.

Choudhary, B., Pophale, C., Gutte, A., Dani, A., & Sonawani, S. S. (2020). Case Study: use of AWS lambda for building a serverless chat application. Proceeding of International Conference on Computational Science and Applications, 237–244.

Ghofrani, J., & Lübke, D. (2018). Challenges of Microservices Architecture: A Survey on the State of the Practice. ZEUS, 2018, 1–8.

Ivanov, V., & Smolander, K. (2018). Implementation of a DevOps pipeline for serverless applications. International Conference on Product-Focused Software Process Improvement, 48–64.

Mukherjee, S. (2019). Benefits of AWS in modern cloud. ArXiv Preprint ArXiv:1903.03219.

Muller, L., Chrysoulas, C., Pitropakis, N., & Barclay, P. J. (2020). A traffic analysis on serverless computing based on the example of a file upload stream on aws lambda. Big Data and Cognitive Computing, 4(4), 38.

Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. CLOSER (1), 137–146.

Rajan, R. A. P. (2018). Serverless architecture-a revolution in cloud computing. 2018 Tenth International Conference on Advanced Computing (ICoAC), 88–93.

Van, L. P., De Praeter, J., Van Wallendael, G., De Cock, J., & Van de Walle, R. (2015). Machine learning for arbitrary downsizing of pre-encoded video in HEVC. 2015 IEEE International Conference on Consumer Electronics (ICCE), 406–407.

Viggiato, M., Terra, R., Rocha, H., Valente, M. T., & Figueiredo, E. (2018). Microservices in practice: A survey study. ArXiv Preprint ArXiv:1808.04836.

Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., & Zambrano, A. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 179–182.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., & Zambrano, A. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. Service Oriented Computing and Applications, 11(2), 233–247.

Zamfir, V.-A., Carabas, M., Carabas, C., & Tapus, N. (2019). Systems monitoring and big data analysis using the elasticsearch system. 2019 22nd International Conference on Control Systems and Computer Science (CSCS), 188–193.