PalArch's Journal of Archaeology of Egypt / Egyptology

ARCHITECTURE OF MONITOR API TO HANDLE INTEGRATION DATA PROBLEM IN DATABASE-MICROSERVICE

Esa Fauzi[1]*, Yani Iriani[2]

[1,2]Department of Informatics Engineering, Faculty of Engineering, Universitas Widyatama, Bandung, Indonesia

[*1]esa.fauzi@widyatama.ac.id

Esa Fauzi, Yani Iriani. Architecture of Monitor Api To Handle Integration Data Problem In Database-Microservice-- Palarch's Journal Of Archaeology Of Egypt/Egyptology 17(10), 1422-1428. ISSN 1567-214x

Keywords:microservice, database-per-service, monitor.

ABSTRACT

Microservice is one of the architectural trends in software development. Microservices divide normally large applications into smaller, more independent parts. One of these small parts can be a function that communicates with the database (database-per-service). However, in microservice architecture, sometimes not only one database is used. In one transaction of the microservice architecture sometimes requires inputting into multiple databases. This can be a problem if in one of the input processes there is an error in one of the services, causing loss of data integration between databases. Based on this problem we proposed a system to monitor the transaction process. The monitor that we have developed will check the data that is entered into the database at each microservice service. If there is an error, the monitor will roll back the failed transaction.

Keywords: microservice, database-per-service, monitor.

INTRODUCTION

Microservice is one of the trends in software architecture design which is an approach with the concept of modularization. Each module is divided into small sections as an independent system. So that each module in the microservice has a light coupling level [1]. Applications that developed with this architecture have advantages including being easy to develop, deploy, maintain and scale compared to a single application.[2].But from all of these advantages, this microservice architecture has disadvantages including complex deployments and autonomy[3]. This autonomy is essentially a positive aspect but achieving this with data integration is a big challenge.

In a microservice architecture, the database is often made into a service or it can be called database-per-service [4]. By making it like this, the transaction process into the database becomes more independent because it is not affected by other things. However, if there are multiple databases-per-service, data integration may be a problem.

In a single request to a system with a microservice architecture, there are usually many transactions in many databases. However, if an error occurs in the process, another service usually has to be canceled. Based on this problem, a monitor is proposed to supervise the transaction process so that if an error occurs, the data integration can be maintained.

RELATED WORKS

If an error occurs in a request to the microservice application, it is necessary to know which service link is fault. Therefore, monitoring has an important role in microservices. Monitoring in a microservice can be run in multiple levels: hardware level, network level, system level, microservice application level, and service access level [2]. In our own research, monitoring is carried out at the application level because it is specifically aimed at monitoring data integration. However, there are also several microservice monitoring at other levels, including:
Benjamin Mayer conducted research to build microservice monitors [5]. The microservice monitor is built in the form of a dashboard based on stakeholder needs by retrieving metrics system data in the form of CPU, memory consumption, workload, and error rate.
Marcello Cinque developed the MetroFunnel monitoring application to monitor request-response messages between microservices [6]. The steps taken in MetroFunnel are sniffing the request-response REST message. The advantage is that there is no need to change the microservice application itself.
Shang-Pin Na developed a version-based monitoring microservice [7]. The proposed scheme is based on versioning, monitoring and visualization (VMAMV), as well as metrics with statistical process control (SPC) methods.

ANALYSIS AND DESIGN

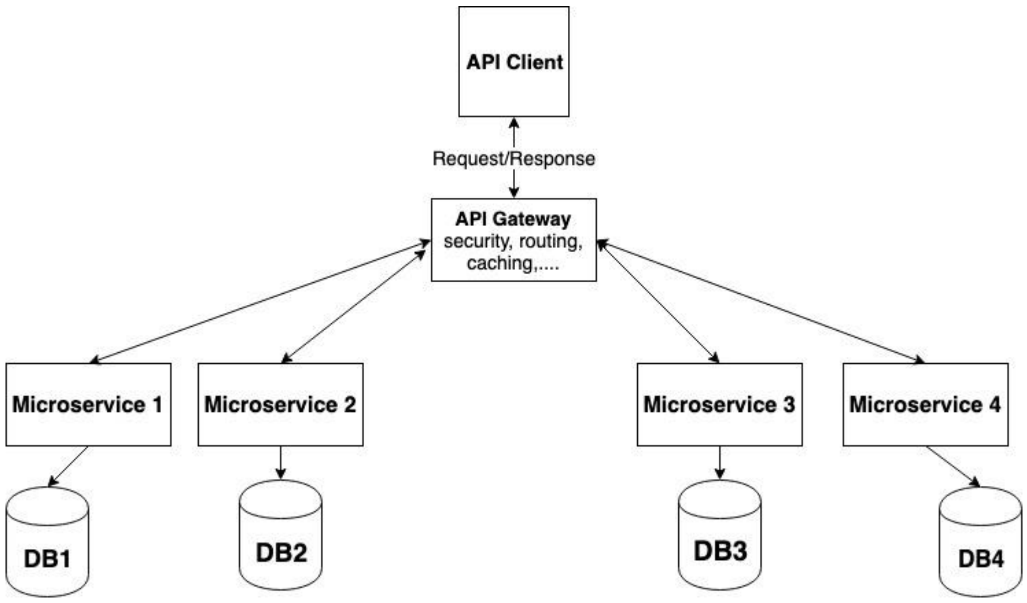

Fig. 1.Microservice with Database-per-Service

The monitor that is built is intended to handle data mismatch issues in the database-as-service API. In a request, there are usually many transactions into the database at the same time. If one of the transactions has an error, the other transaction must also be corrected or canceled. The monitor built into our development will be formed as a service / API. We initially had 2 approaches for designing this API monitor:

Busy-Waiting Approach

The first approach is to use the busy-waiting method[8]which makes the monitor check data changes stored on each database-as-service. This monitor will check every incoming data content along with the time the data is inputted. If there is a change in data in the same time frame, it can be said that it is a new transaction. However, if in the new transaction there is a mismatch between each database-per-service, it means that the monitor will consider the transaction to be a failure. If it is considered a failed transaction, the monitor will roll back the data.

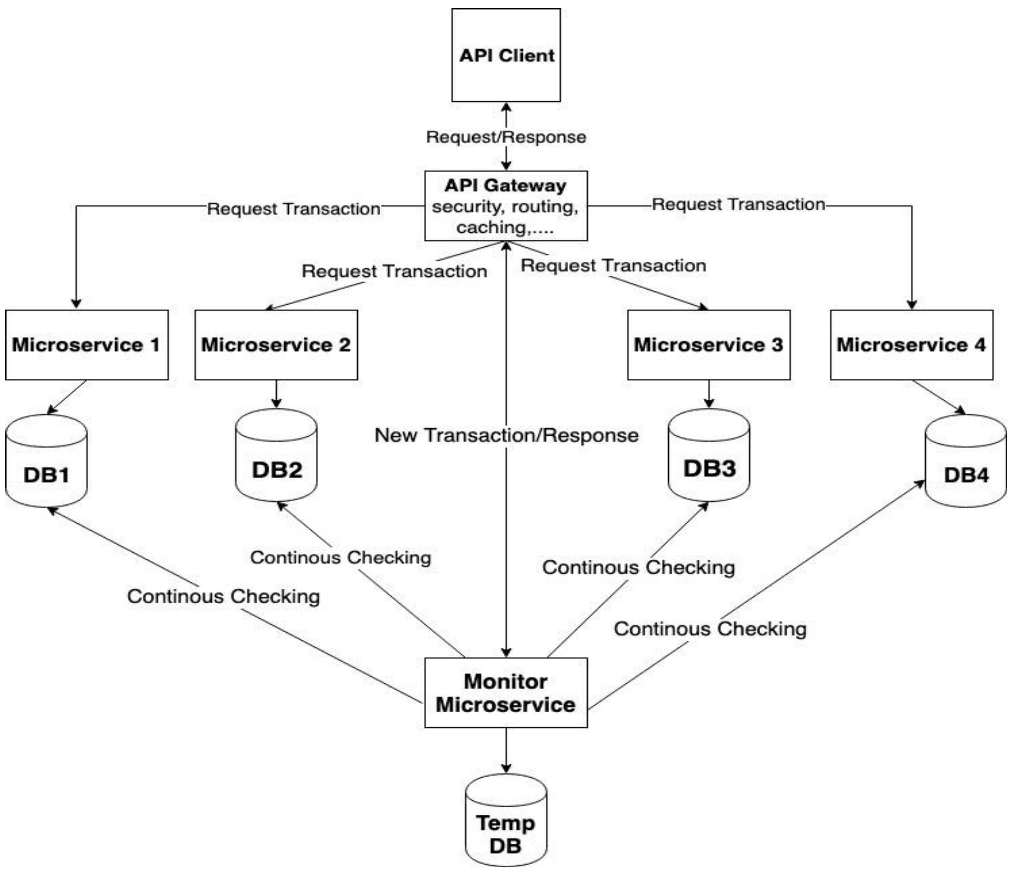

Fig. 2. Monitor with busy-waiting approach

After we analyze this approach has advantages and disadvantages, including:

Advantages:
The monitor becomes more independent and is not affected by other APIs because it only pays attention to the database in the microservice architecture
Good for handling data transactions in the form of new entries

Disadvantages:
Difficult to detect and search transactions for data changes or deletions.

We did not implement this approach because it has disadvantages as mentioned.

Sleep & Wake Up Approach
This approach uses the concept of sleep and wake up; sleep, wake up, idle in the process [9]. When there is no transaction process, the API monitor is idle. However, when there is a transaction process, the monitor will be awakened to check the data transactions that occur.
This approach requires reports from other database APIs to find out whether the transaction processing to the database has been carried out or not. So in the process, all APIs before and after making transactions into the database are required to report to the API monitor. If all database APIs report a successful transaction, it means that the entire transaction to the database is considered

successful. However, if it fails, the monitor will roll back the incoming transactions so that data integration can be maintained.

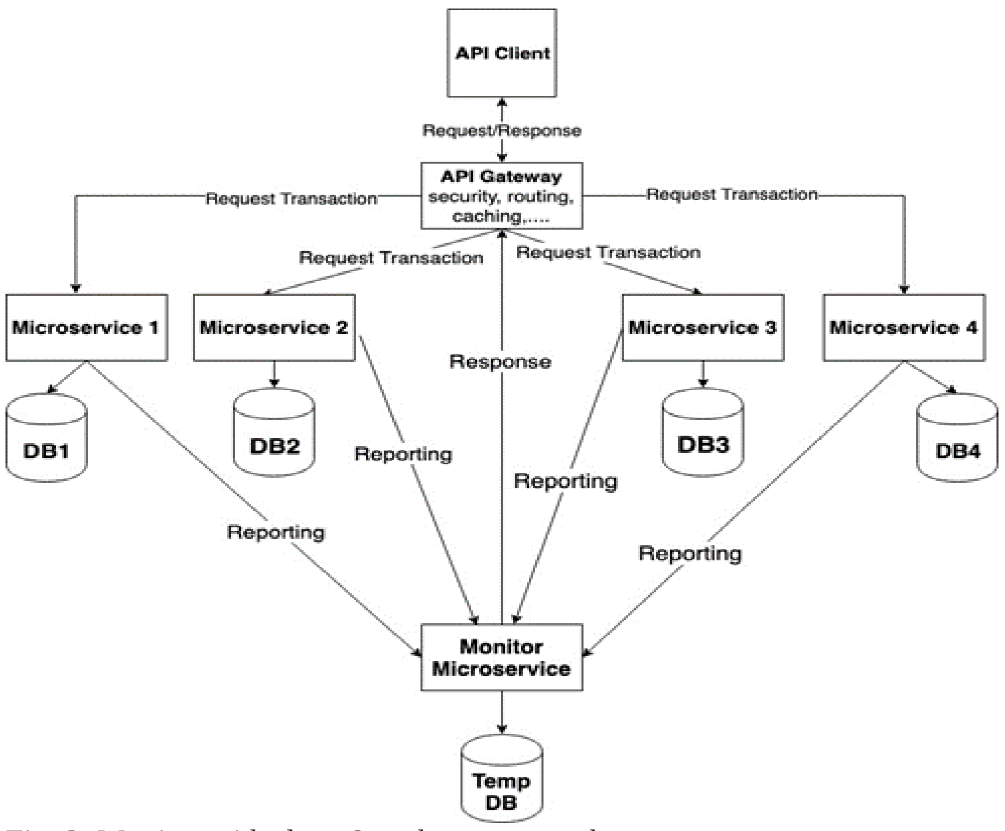

Fig. 3. Monitor with sleep & wakeup approach

From the results of our analysis this approach has several advantages & disadvantages:

Advantages:
• Can detect changes in the form of input, change, or deletion

Disadvantages:
• It takes longer because there must be a transaction process report from all database APIs

We apply this approach because even though it is a little longer, it can maintain data integration.

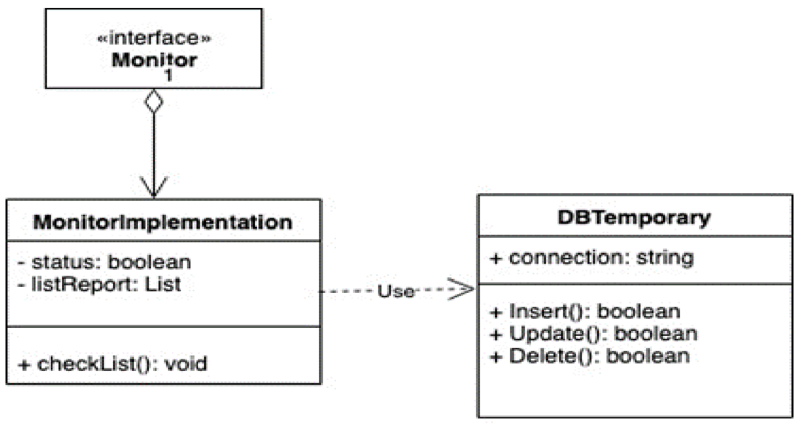

Fig. 4. Class Diagram Monitor API

In development, we made this monitor in the form of an API. There are 2 main classes and 1 interface. The main class of this monitor is used to handle checking the list of microservice transactions to the database, giving responses to the API gateway, and receiving reports from the microservice. In addition, there is a support class for recording transaction lists into a temporary database.

CONCLUSION

Monitoring is one way to maintain data integrity in microservice applications. Monitoring can also be used to control data transaction errors in the database (for example rolling back) such as in the application we developed. The microservice monitor that we have developed can maintain the integration of data in a transaction to multiple databases. But it has a disadvantages because each microservice has to report to the monitor so it takes a little more time. Therefore, for further development we plan to change the reading pattern of the monitor data by sniffing the request / response messages. From this message, the monitor will check the data in the database within a certain period of time if there is an error response and will make data corrections or rollbacks of failed transactions.

REFERENCES

[1] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” IEEE Softw., vol. 35, no. 3, pp. 24–35, 2018.
[2] Y. Jiang, N. Zhang, and Z. Ren, “Research on intelligent monitoring scheme for microservice application systems,” Int. Conf. Intell. Transp. Big Data Smart City, pp. 791–794, 2020.
[3] K. Gos and W. Zabierowski, “The Comparison of Microservice and Monolithic Architecture,” IEEE 16th Int. Conf. Perspect. Technol. Methods MEMS Des., pp. 150–153, 2020.
[4] M. S. S. Kumar and D. S. Mallikarjuna, “Database-per-Service for E-Learning System with Microservice Architecture,” Int. Conf. Smart Technol. Smart Nation, pp. 352–355, 2017.
[5] B. Mayer and R. Weinreich, “A dashboard for microservice monitoring and management,” IEEE Int. Conf. Softw. Archit. Work., pp. 66–69, 2017.

[6] M. Cinque, R. Della Corte, R. Iorio, and A. Pecchia, “An Exploratory Study on Zeroconf Monitoring of Microservices Systems,” 14th Eur. Dependable Comput. Conf., pp. 112–115, 2018.
[7] S. P. Ma, I. H. Liu, C. Y. Chen, J. T. Lin, and N. L. Hsueh, “Version-Based Microservice Analysis, Monitoring, and Visualization,” Asia-Pacific Softw. Eng. Conf., pp. 165–172, 2019.
[8] R. Höttger, B. Igel, and O. Spinczyk, “On reducing busy waiting in autosar via task-release-delta-based runnable reordering,” Des. Autom. Test Eur., pp. 1510–1515, 2017.
[9] J. Haimour and O. Abu-Sharkh, “Energy efficient sleep/wake-up techniques for IOT: A survey,” IEEE Jordan Int. Jt. Conf. Electr. Eng. Inf. Technol., pp. 459–464, 2019.