2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

Containerized Event-Driven Microservice Architecture

Siti Zulaikha Mohd Zuki**, Radziah Mohamad, Nor Azizah Saadon

Faculty of Computing, Universiti Teknologi Malaysia, Johor, Malaysia.
*Corresponding Author.
PARS2023: Postgraduate Annual Research Seminars 2023.

Received 28/09/2023, Revised 10/02/2024, Accepted 12/02/2024, Published 25/02/2024

Abstract

Microservice architecture offers many advantages, especially for business applications, due to its flexibility, expandability, and loosely coupled structure for ease of maintenance. However, there are several disadvantages that stem from the features of microservices, such as the fact that microservices are independent in nature can hinder meaningful communication and make data synchronization more challenging. This paper addresses the issues by proposing a containerized microservices in an asynchronous event-driven architecture. This architecture encloses microservices in containers and implements an event manager to keep track of all the events in an event log to reduce errors in the application. Experiment results show a decline in response time compared to two other benchmark architectures, as well as a lessening in error rate.

Keywords: Container environment, Error handling, Event-driven architecture, Event manager, Microservice

Introduction

The popularity of microservice architecture rises in 2014 after Netflix shared the success story of its system migration from monolithic architecture to microservice architecture in 2009[1]. Other top global companies, such as Amazon and Uber, joining the system migration further prove the competency of microservice architecture in handling larger applications. Moreover, the lightweight nature of a microservice architecture, its self-management ability, and its scalability show it to be an effective technique to manage complex large-scale systems[2].

Microservice architecture breaks down the business capabilities of an application into smaller components known as microservices with a loosely coupled structure for a more rapid, reliable, and flexible delivery of tasks[3,4]. In addition, microservices in an application are independent of each other, allowing the development, management, and maintenance processes to be done separately by different teams[5–7]. This improves productivity as development processes can be done in parallel by the teams, and maintenance is more effective with different teams focusing on specific microservices instead of an entire application.

However, microservice architecture is not without challenges and issues. Some examples of commonly addressed issues in microservice architecture are data consistency, communication between microservices, and runtime error handling[5,8,9]. In cloud computing, communication issues can be

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

addressed with load balancing[10] or managing service level agreement[11], but there are limitations in adding functionalities to microservice application as to not overwhelm its complexity.

To minimize dependencies, each microservice gets its own individual setup, such as a database and logical functions. This makes retrieving interrelated data more challenging, especially when microservice update its individual database only at the end of a query-response process[12]. The relationships between microservices are typically in a topological graph where microservices are connected in a chain of processes and some chains share similar microservices[5,13]. Furthermore, this also indicates that answering a client's queries spans multiple microservices.

To address the challenges and issues above, this paper implements containerized microservices in an asynchronous event-driven architecture. The key ideas of this paper are as follows:

- Implementation of an event manager as an event and message broker with an event log to keep track of every event that occurred in the application and record the relationships between microservices in chains and the possible events that are related to the microservices.
- The containerization of the microservices further separates individual databases from each other. With the implementation of the event log, the event manager can easily differentiate one database from the other and perform data updates more efficiently.
- By keeping track of the event and the microservices that are responsible for it, the event manager needs to notify only the associated microservices instead of blasting the message to every microservice in the application.

The remainder of this paper is organized as follows: the related work describes the different ways similar works address the issues, the methodology illustrates the proposed architecture design and finally, the result and discussion examine the results from the experiments.

Related Work

Event-Driven Architecture for Microservices

Event-driven architecture is a system where microservices exchange information or data with each other through a publish-and-listen event. It absorbs information into a message broker, usually known as an event bus, and then broadcasts it to the listening microservices[14]. Fig. 1 illustrates the publish-and-listen event in an event-driven architecture in a microservice application.

Singh et al.[15] introduce a message streaming data driven microservice system using an event-driven architecture. It is a direct messaging system with a control to monitor the message queue to improve latency. However, it does not address the handling of network failures. Surantha et al.[16] develop an intelligent sleep monitoring system using microservices and event-driven architecture. The system acquires sleeping data through sensor wearables and records the data in a database. The implementation of a message broker increases throughput and decreases response time. Nevertheless, there is no event log to trace the events in the event of a network failure.

Containerization of Microservices

Containers are a semi-isolated environment that keeps its host separated from other entities in the application but can still communicate with them. Containerization is not a new concept, though it thrives after it starts getting implemented in Service-Oriented Architecture (SOA) which then subsequently becomes synonymous with microservice architectures[17].

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

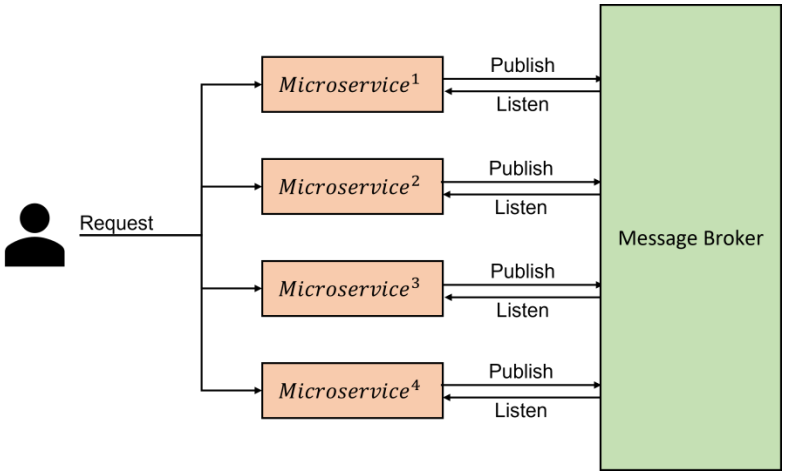

Figure 1. Publish-and-listen event in an event-driven architecture in microservice application.

Containers are pertinent to effective load balancing, efficient autoscaling, and scheduling[5,7]. An example of work involving containers is by Matani et al.[18], which involves the allocation of resources in a grid environment by using replication techniques to reduce cost and execution time. Another work is by Zhou et al.[19] where the scheduling algorithm that comes with containerization smooths the real-time workflow between tasks and processes. Finally, Yu et al.[6] use the redundancy strategy in a container environment to further optimize the dependability of workflow. These works show that using containers to enclose microservices helps smooth the workflow and manage the scheduling of an asynchronous microservice architecture.

Methodology

To address the issues with data consistency, communication between microservices, and runtime error handling, this paper implements containerized microservices in an asynchronous event-driven architecture. Fig. 2 illustrates the proposed architecture. The microservices are enclosed in separate, individual containers, where each container publishes and listens to the message broker hosted by an event manager. The event manager also keeps track of every exchange by keeping a record in an event log. Kubernetes with Docker application is used to set up the container environment and orchestration system, while Apache Kafka is used as a means of communication between the microservices and the event manager.

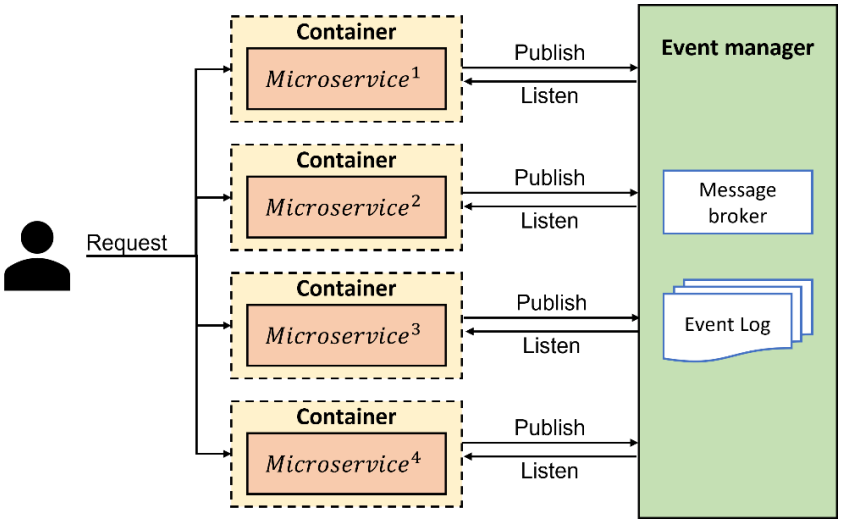

Figure 2. Containerized microservices in an asynchronous event-driven architecture.

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

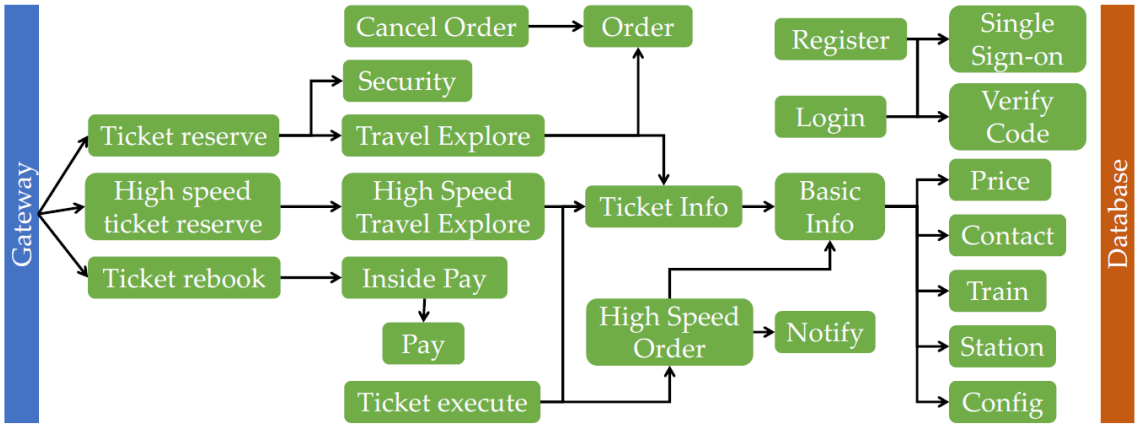

Figure 3. The TrainTicket[20] system.

The proposed architecture is tested on a TrainTicket[20] system which is illustrated in Fig. 3. Table 1 illustrates an excerpt of an event log based on the TrainTicket system. The first event 00001 is a publish event where the user requests a Ticket rebook. The request is a success, so there is no need for a retry. In the second event 00002, the event is a listen event where the microservice should receive the request and return a response to the user. The event status is dropped as there is an error in the network, so the system will attempt a retry. The log keeps track of the events to allow retries of dropped events in case of transient errors and prevent any event from being missed by the system.

Table 1. Example of event log based on TrainTicket System.

Timestamp	EventID	EventType	MS_ID	Status	Retry
20.06.23 11.11	00001	Publish	Ticket_rebook	Success	False
20.06.23 11.12	00002	Listen	Ticket_rebook	Dropped	True

Even though the proposed architecture tracks and allows retries for dropped requests due to transient errors, there is a limitation to the number of retries that can be attempted. This is to avoid overloaded requests, which can lead to a denial of service (DOS).

Results and Discussion

The proposed architecture is tested using a set of randomly pre-generated 500 request samples, and the result is compared with two other architectures: an event-driven architecture by Rahmatulloh et al.[14] and a baseline API-driven architecture, using the same set of request samples. To determine the efficiency of the architecture with regard to the user's request and the response received, the response time is recorded. Fig. 4 and Table 2 show the result of the proposed architecture compared to the other two architectures.

Table 2. The response time of the proposed architecture compared to the event-driven architecture and API-driven architecture.

Total Request	Proposed Architecture (ms)	Event-driven Architecture (ms)	API-driven Architecture (ms)
100	1889	2096	2247
200	2111	2905	3852
300	2642	3277	3895
400	3519	4433	5215
500	4820	5805	6859

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

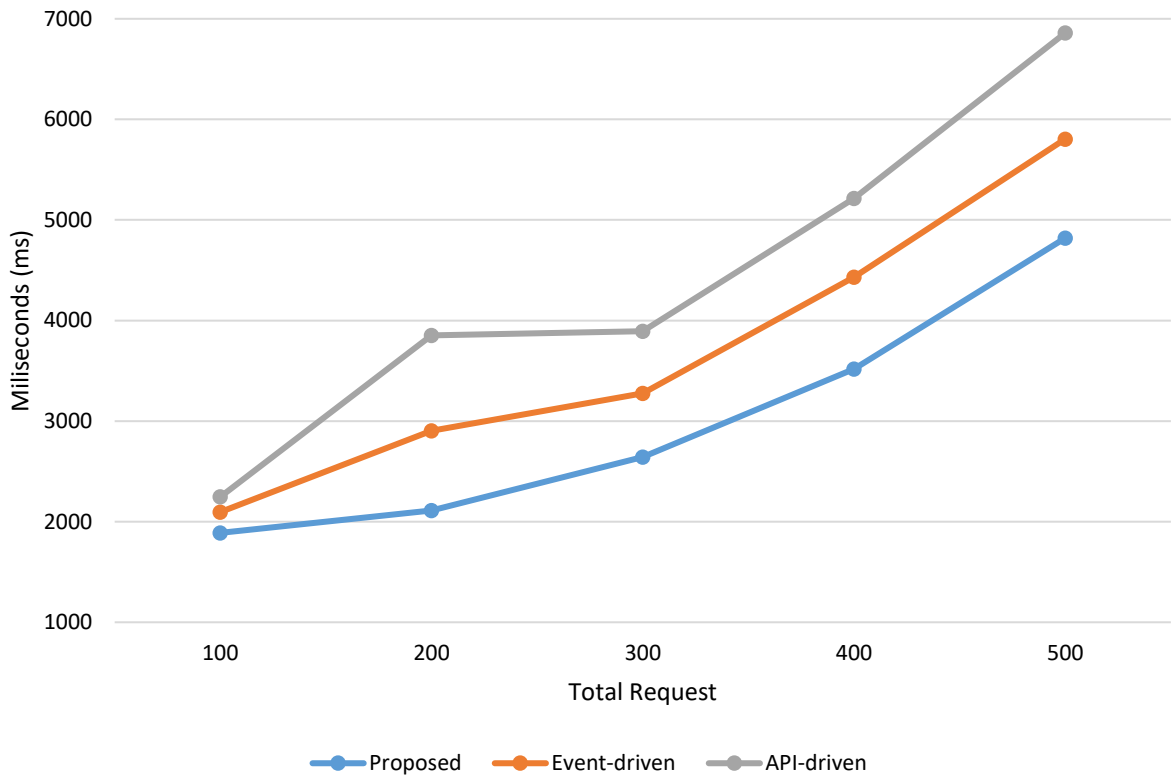

Figure 4. The response time of the proposed architecture compared to the event-driven architecture and API-driven architecture.

As can be seen in Fig. 4, the proposed architecture manages to record the lowest response time in all request increments compared to the other two benchmark architectures. The API-driven architecture uses APIs as a means of communication between microservices, whereas the event-driven architecture does not use any event logs to keep track of the events. This shows the efficiency of the event log at keeping track of the publish-and-listen events and successfully responding to the request at a low response time.

Next, Fig. 5 and Table 3 show the error rate for the total requests tested for all three architectures at increments of 100 requests each. As can be seen in the figure, the proposed architecture shows a huge difference in error rate compared to the benchmark architectures. This is due to records of event status and retry status for every event registered by the event manager. The event log ensures that no event is amiss, even in the event of a network failure.

Table 3. The error rate of the proposed architecture compared to the event-driven architecture and API-driven architecture.

Total Request	Proposed Architecture (%)	Event-driven Architecture (%)	API-driven Architecture (%)
100	0	0	0
200	4.6	7	11
300	5.1	10.33	11.3
400	4.9	10.5	9.25
500	5	4	11.2

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

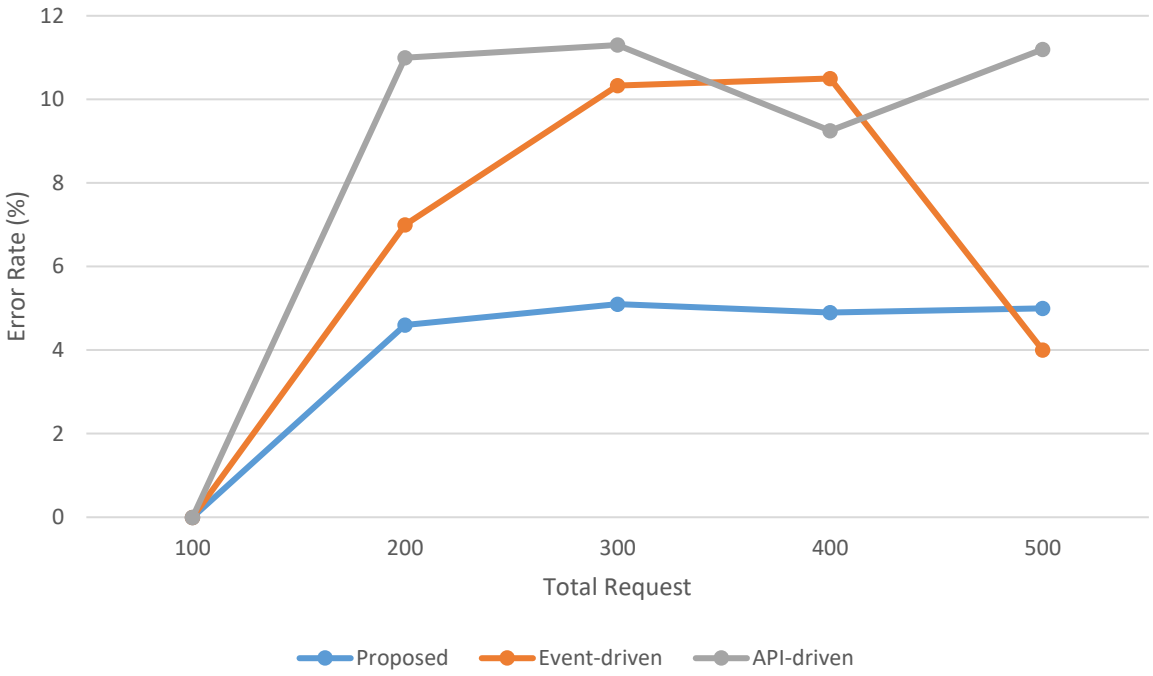

Figure 5. The error rate of the proposed architecture compared to the event-driven and API-driven architecture.

Conclusion

Based on the results obtained from the experiments, the proposed containerized microservices in an asynchronous event-driven architecture show promising results in error handling and meaningful communication between microservices. The event log proves to be efficient in tracking events and the microservices related to them, subsequently lowering the response time, and improving the user's experience. However, the event log can be further improved to also monitor the data exchange between microservices, while the event manager can facilitate individual database updates in real-time.

Acknowledgment

This research has been supported by the Ministry of Higher Education (MOHE) through the Fundamental Research Grant Scheme (FRGS/1/2021/ICT01/UTM/02/1).

Authors' Declaration

- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are ours. Furthermore, any Figures and images, that are not ours, have been included with the necessary permission for re-publication, which is attached to the manuscript.
- Ethical Clearance: The project was approved by the local ethical committee in Universiti Teknologi Malaysia.

Authors' Contribution Statement

The authors S.Z.M.Z, R.M and N.A.S contributed to the design and implementation of the research. Author S.Z.M.Z conducted the experiments, the analysis of the results and the manuscript writing. R.M and N.A.S verified the experiment result,

proofread, and approved the final version of the manuscript.

References

1. Blinowski G, Ojdowska A, Przybylek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access. 2022 Feb;10:20357–74. https://doi.org/10.1109/ACCESS.2022.3152803
2. Zhang S, Zhang M, Ni L, Liu P. A Multi-Level Self-Adaptation Approach For Microservice Systems. ICCCBDA. 2019;498–502. https://doi.org/10.1109/ICCCBDA.2019.8725647
3. He H, Su L, Ye K. GraphGRU: A Graph Neural Network Model for Resource Prediction in Microservice Cluster. ICPADS. 2023;499–506. https://doi.org/10.1109/ICPADS56603.2022.00071
4. Liu H, Zhang W, Zhang X, Cao Z, Tian R. Context-Aware and QoS Prediction-based Cross-Domain Microservice Instance Discovery. ICSESS . 2022;30–4. https://doi.org/10.1109/ICSESS54813.2022.9930241
5. Wan F, Wu X, Zhang Q. Chain-Oriented Load Balancing in Microservice System. WCCCT. 2020;10–4. https://doi.org/10.1109/WCCCT49810.2020.9169996
6. Yu X, Wu W, Wang Y. Dependable Workflow Scheduling for Microservice QoS Based on Deep Q-Network. ICWS. 2022;240–5. https://doi.org/10.1109/ICWS55610.2022.00045
7. Hossen MR, Islam MA, Ahmed K. Practical Efficient Microservice Autoscaling with QoS Assurance. HPDC. 2022;240–52. https://doi.org/10.1145/3502181.3531460
8. Gan Y, Liang M, Dev S, Lo D, Delimitrou C. Sage: Practical and scalable ML-driven performance debugging in microservices. ASPLOS. 2021;135–51. https://doi.org/10.1145/3445814.3446700
9. Chen J, Liu F, Jiang J, Zhong G, Xu D, Tan Z, et al. TraceGra: A trace-based anomaly detection for microservice using graph deep learning. Comput Commun. Elsevier B.V. 2023 Apr 15;204:109–17. https://doi.org/10.1016/j.comcom.2023.03.028
10. Abed MM, Younis MF. Developing load balancing for IoT - Cloud computing based on advanced firefly and weighted round robin algorithms. Baghdad Sci J. 2019;16(1):130–9. https://doi.org/10.21123/bsj.2019.16.1.0130
11. Kumar S, Kumar N. Conceptual service level agreement mechanism to minimize the SLA violation with SLA negotiation process in cloud computing environment. Baghdad Sci J. 2021 Jun 1;18:1020–9. https://doi.org/10.21123/bsj.2021.18.2(Suppl.).1020
12. Vohra N, Kerthyayana Manuaba IB. Implementation of REST API vs GraphQL in Microservice Architecture. ICIMTech . 2022;45–50. https://doi.org/10.1109/ICIMTech55957.2022.9915098
13. Lan Y, Fang L, Zhang M, Su J, Yang Z, Li H. Service dependency mining method based on service call chain analysis. ICSS. 2021;84–9. https://doi.org/10.1109/ICSS53362.2021.00021
14. Rahmatulloh A, Nugraha F, Gunawan R, Darmawan I. Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems. ICADEIS. 2022. https://doi.org/10.1109/ICADEIS56544.2022.10037390
15. Singh A, Singh V, Aggarwal A, Aggarwal S. Event Driven Architecture for Message Streaming data driven Microservices systems residing in distributed version control system.ICISTSD. 2022;308–12. https://doi.org/10.1109/ICISTSD55159.2022.10010390
16. Surantha N, Utomo OK, Lionel EM, Gozali ID, Isa SM. Intelligent Sleep Monitoring System Based on Microservices and Event-Driven Architecture. IEEE Access. 2022;10:42055–66. https://doi.org10.1109/ACCESS.2022.3167637
17. Mulahuwaish A, Korbel S, Qolomany B. Improving datacenter utilization through containerized service-based architecture. J Cloud Comput; 2022 Dec 1;11(1). https://doi.org/10.1186/s13677-022-00319-0
18. Matani A, Naji HR, Motallebi H. A Fault-Tolerant Workflow Scheduling Algorithm for Grid with Near-Optimal Redundancy. J Grid Comput. 2020 Sep 1;18(3):377–94. https://doi.org/10.1007/s10723-020-09522-2
19. Zhou J, Sun J, Zhang M, Ma Y. Dependable Scheduling for Real-Time Workflows on Cyber-Physical Cloud Systems. IEEE Trans Industr Inform. 2021 Nov 1;17(11):7820–9. https://doi.org/10.1109/TII.2020.3011506
20. Madi T, Esteves-Verissimo P. A Fault and Intrusion Tolerance Framework for Containerized Environments: A Specification-Based Error Detection Approach. SRMC. 2022;1–8. https://doi.org/10.1109/SRMC57347.2022.00005

2024, 21(2 Special Issue): 0584-0591
https://doi.org/10.21123/bsj.2024.9454
P-ISSN: 2078-8665 - E-ISSN: 2411-7986
Baghdad Science Journal

بنية الخدمات الصغيرة المعتمدة على الأحداث

ستي زليخة محمد زوكي، راضية محمد، نور عزيزة سعدون

كلية الحاسبات، الجامعة التكنولوجية الماليزية، جوهور، ماليزيا.

الخلاصة

توفر بنية الخدمات الصغيرة العديد من المزايا، خاصة لتطبيقات الأعمال، نظرًا لمرونتها وقابليتها للتوسيع وبنيتها المترابطة بشكل غير محكم لسهولة الصيانة. ومع ذلك، هناك العديد من العيوب التي تنبع من ميزات الخدمات الصغيرة، مثل حقيقة أن الخدمات الصغيرة مستقلة بطبيعتها يمكن أن تعيق التواصل الهادف وتجعل مزامنة البيانات أكثر صعوبة. تتناول هذه الورقة المشكلات من خلال اقتراح خدمات مصغرة مضمنة في حاوية في بنية غير متزامنة تعتمد على الأحداث. تحتوي هذه البنية على خدمات صغيرة في حاويات وتقوم بتنفيذ مدير الأحداث لتتبع جميع الأحداث في سجل الأحداث لتقليل الأخطاء في التطبيق. تظهر نتائج التجربة انخفاضًا في وقت الاستجابة مقارنةً بمبنيين معياريين آخرين، بالإضافة إلى انخفاض في معدل الخطأ.

الكلمات المفتاحية: بيئة الحاوية، معالجة الأخطاء، البنية المبنية على الأحداث، مدير الأحداث، الخدمات الصغيرة.