UDC 004.774:519.246.8
DOI: 10.31891/CSIT-2021-5-8

TATJANA SELIVORSTOVA, SERGEY KLISHCH,
SERHII KYRYCHENKO, ANTON GUDA, KATERYNA OSTROVSKAYA
National Metallurgical Academy of Ukraine

ANALYSIS OF MONOLITHIC AND MICROSERVICE ARCHITECTURES FEATURES AND METRICS

In this paper the information technologies stack is presented. Thesetechnologies are used during network architecture deployment. The analysis of technological advantages and drawbacks under investigation for monolithic and network architectures will be useful during of cyber security analysis in telecom networks. The analysis of the main numeric characteristics was carried out with the aid of Kubectl. The results of a series of numerical experiments on the evaluation of the response speed to requests and the fault tolerance are presented. The characteristics of the of monolithic and microservice-based architectures scalability are under investigation. For the time series sets, which characterize the network server load, the value of the Hurst exponent was calculated.

The research main goal is the monolithic and microservice architecture main characteristics analysis, time series data from the network server accruing, and their statistical analysis.

The methodology of Kubernetes clusters deploying using Minikube, Kubectl, Docker has been used. Application deploy on AWS ECS virtual machine with monolithic architecture and on the Kubernetes cluster (AWS EKS) were conducted.

The investigation results gives us the confirmation, that the microservices architecture would be more fault tolerance and flexible in comparison with the monolithic architecture. Time series fractal analysis on the server equipment load showed the presence of long-term dependency, so that we can treat the traffic implementation as a self-similar process.

The scientific novelty of the article lies in the application of fractal analysis to real time series: use of the kernel in user space, kernel latency, RAM usage, caching of RAM collected over 6 months with a step of 10 seconds, establishing a long-term dependence of time series data.

The practical significance of the research is methodology creation of the monolithic and microservice architectures deployment and exploitation, as well as the use of time series fractal analysis for the network equipment load exploration.

Key words: monolith architecture, microservice architecture, replay delay, scalability, fault tolerance, monitoring, time series, Hirst's exponent.

ТЕТЯНА СЕЛІВЬОРСТОВА, СЕРГІЙ КЛІЩ,
СЕРГІЙ КИРИЧЕНКО, АНТОН ГУДА, КАТЕРИНА ОСТРОВСЬКА
Національна металургійна академія України

АНАЛІЗ ОСОБЛИВОСТЕЙ ТА МЕТРИК МОНОЛІТНОЇ І МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

В роботі представлений стек інформаційних технологій, які засовуються при розгортанні мережевої архітектури. Представлений аналіз технологічних переваг та недоліків монолітної та мережевої архітектури може бути корисним при розгляді питань кібер безпеки в телекомунікаційних мережах. Проведений аналіз основних кількісних характеристик засобами Kubectl. Наведені результати серії обчислювальних експериментів по оцінці швидкості відповіді на запити, відмовостійкості, розглянуті питання масштабування монолітної та мікросервісної мережевої архітектури. Для наборів часових рядів, що характеригують завантаження мережевого серверу, обчислені значення показника Харста.

Метою роботи є проведення дослідження основиних характеристик монолітної та мікросервісної архітектури, збір данних часових рядів з сервера та їхній статистичний аналіз.

Використано методику розгортання Kubernetes кластеру засобами Minikube, Kubectl, Docker. Проведено розгортання додатку на віртуальній машині AWS EC2 – монолітна архітектура та Kubernetes кластеру на AWS EKS – мікросервісна архітектура.

Результати отримані в роботі підтвердили, що архітектура мікросервісу має набагато більшу відмовостійкість та гнучкість у порівнянні з монолітною архітектурою. Фрактальний аналіз часових рядів навантаження серверного обладнання показав наявність довгострокової залежності, що дає змогу представити реалізацію трафіку як самоподібний процес.

Наукова новизна роботи полягає у застосуванні фрактального анілізу до реальних часових рядів: використання ядра в просторі користувача, часу очікування ядра, використання оперативної пам'яті, кешування оперативної пам'яті, зібраних на протізі 6 місяців з кроком 10 секунд, встановленні довгострокової залежності даних часових рядів.

Практична значимість роботи полягає у викладенні методології розгортання та експлуатації монолітної та мікросервісної архітектур, застосуванні фрактального аналізу часових рядів для аналізу мережевого навантаження.

Ключові слова: архітектура, монолітна, мікросервісна, затримка, надійність, масштабування, місткість, спритність, моніторинг, часовий ряд, показник Херста.

Introduction

The rapid growth in the number of complex and extensive web-applications has led to the introduction of the latest communication technologies within the project [1, 2]. The monolithic architecture [3, 4] is logically replaced by microservices [5, 6]. In this regard, the task of determining the technical [7, 8] and technological [9, 10] features of this technology [11, 12] becomes relevant.

Related works

Conceptual differences between monolithic and microservice implementations

It is known that the monolithic architecture involves the process of developing the application as a whole. Any changes, even the smallest ones, require significant restructuring and deployment of the entire application [13, 14]. Over time, it becomes increasingly difficult to maintain a clear and understandable modular structure, as changes in the logic of one module tend to affect the code of other modules [15]. Unlike a monolithic architecture, a microservice architecture is an approach to building a server application [16] as a set of almost unrelated services (Figure 1). Services are developed and deployed separately and independently of each other [17]. A separate process is required to run each service. Communication between processes is implemented using HTTP / HTTPS, WebSockets or some other protocols (Figure 2).

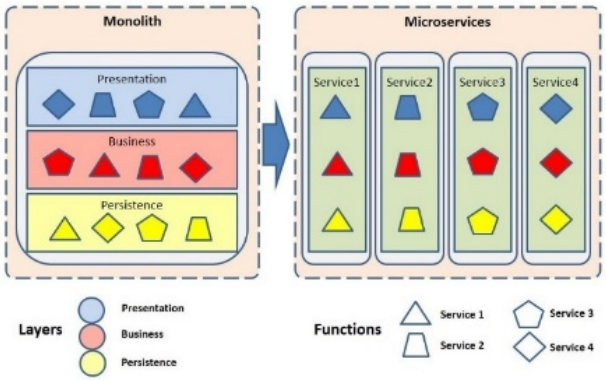

Fig. 1. The typical transition from a monolith to microservices [18]

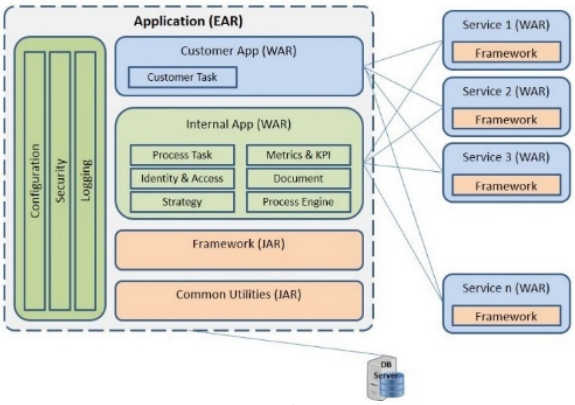

Fig. 2. Microservices landscape for the refactored monolith [18]

The benefits of microservice architecture [19, 20] become apparent when the structure of monolith [21, 22] becomes so complex that it affects both development and maintenance, as well as runtime performance [23, 24]. Microservice architecture does not benefit smaller applications as the overheads due to the distributed architecture both in development and operation outweighs the benefits [25, 26].

Research of the main characteristics of application implementation on microservices and monolith

Creation of test monolithic and microservice architecture

For comparison, we deployed the Wordpress application in an AWS EKS cluster (microservice architecture) and on the virtual machine AWS EC2 (monolithic architecture). We used the computing power of the cloud provider Amazon Web Services and the same type of virtual machines for AWS EKS and AWS EC2. The AWS EKS nodes were created on the basis of m5.large virtual machines. These machines have 2 CPUs and 8 GB of RAM. We also used an Application Load Balancer. It is important to note, that pods with MariaDB and Wordpress itself were deployed on different nods.

Speed of response to a request

Let's check the speed of response to requests for microservice and monolithic architecture [30, 27]. To check, we will use the application "httping", which can determine the speed of responses to http requests. The command "httping -i 1 -c 50" means that we will send 50 http requests with an interval of 1 second.

Run the command for the microservice application:

```
httping -i 1 -c 50 http://a179086a04dc94aa7b0fc6a61e80987e-923853456.
us-east-1.elb.amazonaws.com/wp-admin/
```

The average response time to a request in a microservice implementation is 24 ms. And at a consecutive call of commands the speed of the answer will depend on length of turn.

Let's now run the same command for the monolithic application.

```
httping -i 1 -c 50 http://54.93.175.7/
```

The average response time to a request in a monolith implementation is 1.12 ms.

Thus, the response time to the request is determined by the formula:

$$L(n) = n \times F\,, \qquad (1)$$

where n is the length of the queue, F is the delay in processing one request. For microverses $F = 24\ ms$, for monolith $F = 1.12\ ms$.

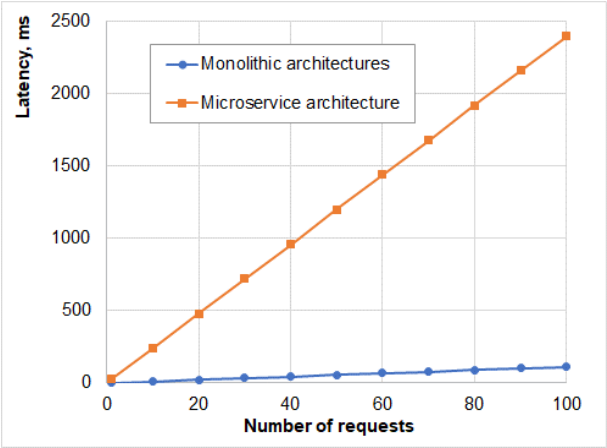

Fig. 3. Response time to the request

Thus, we see that the call delay in the microservice architecture is on average 24 ms, which is almost twenty times the delay of the monolithic architecture. It should be noted that the delay increases significantly as the queue increases.

The monolith does not have a network delay because all calls are local. Microservices cannot stay ahead of physics when it comes to the speed of response to requests.

Reliability of network requests

The following calculations are used to assess the reliability of network requests. Suppose a service accesses another service over a microservice cluster network with 99.9% reliability. This means that out of 1000 calls, one will fail due to network problems.

$$R(n) = P^n, \tag{2}$$

where n is the length of the call chains, P is the reliability of the call.

Now, if this service calls another service, we get 99.8% fault tolerance. For a chain of calls with a depth of up to 10 calls, we achieve 99% reliability – which means that 1 in 100 calls fails (Figure 5). Because all calls in the monolith are local, there is no chance of a network failure.

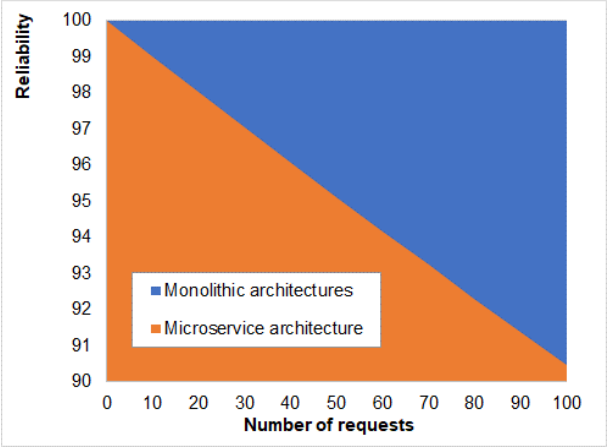

Fig. 4. Reliability of network requests

Because network problems are an expected phenomenon, microservices offer some solutions. For example, the open source Spring Cloud application offers transparent load balancing and troubleshooting for Java.

In a microservice cluster, services can fail. In this case, the cluster manager simply launches a similar service with the same characteristics. This makes the microservice architecture extremely stable.

Scaling

There are ways to scale the monolith. You can run multiple instances and configure routing. Or you can run multiple threads. For microservice architecture, this is also true. But you can scale microservices with less resources. This means that for the money spent on resources, microservices provide more bandwidth. More accurate scaling is also possible. If the monolith uses all the resources, the way to handle more connections is to output a second instance. If one microservice uses all the resources, only that service will need more copies. Because microservices are less

resource-intensive, it saves resources. Because scaling is simple and accurate, it means that only the required amount of resources is used.

For example, suppose a monolith runs on an m5.16xlarge instance of Amazon Web Service. The m5.16xlarge instance offers a whopping 64 processors and 262 GB of RAM. At the moment, the cost is also enormous – 2211.84 US dollars per month for a round-the-clock virtual machine. For the same money you can buy 9 instances of c5.2xlarge with 8 virtual processors and 16 GB of RAM, each of which works around the clock and without days off. But most workloads do not require full use of resources around the clock. Instead, the use of resources reaches a maximum at certain hours, and at other times the load becomes less.

$$Pr(H) = A \cdot H_{day} \cdot D, \quad (3)$$

where D is the number of days, H_{day} is the number of operating hours per day, $A = 3\$$ is the cost of 1 hour of the 9 instances.

Below is Figure 6, which shows how much these 9 smaller copies cost if they run only for a certain period of time instead of 24/7 (Figure 5).

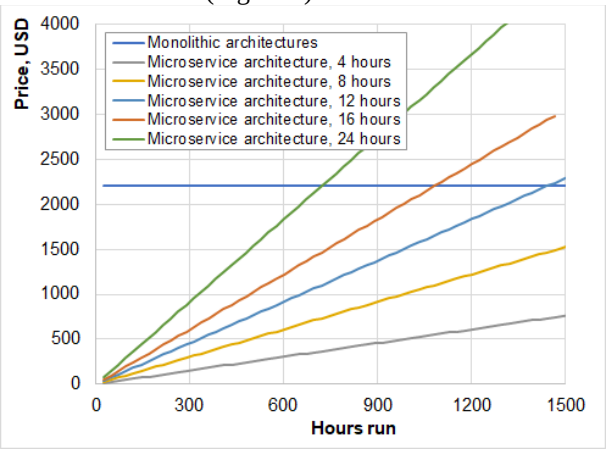

Fig. 5. Price scaling

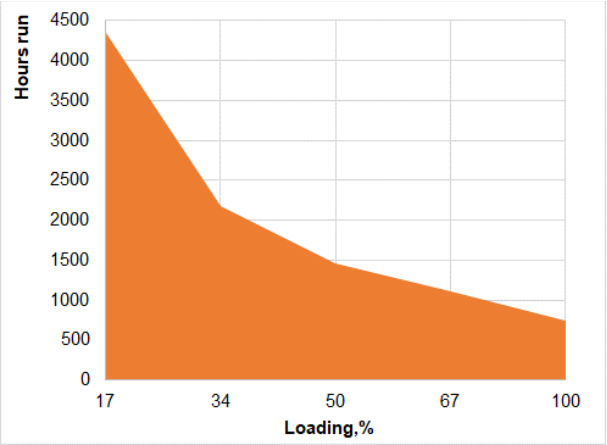

Fig. 6. Time for which the cost of operating microservice architecture becomes more expensive than operating monolithic

Because the allocated resources are cheaper than one, which is charged hourly, there is a time when a single copy becomes cheaper. In this example, the transition is (Figure 6):

– 744 hours, if the instances work 100% of the time;
– 1104 hours, if the instances work 67% of the time;
– 1464 hours, if the instances work 50% of the time;
– 2184 hours, if the instances work 34% of the time;
– 4344 hours, if the instances work 17% of the time.

So, over time, using microservices can become more expensive than a monolith. However, microservices allow us to use resources more efficiently and scale more accurately.

The results of comparing the characteristics of the implementation of the application deployment on microservices and monolith

We researched the configuration of the Kubernetes orchestration platform for deploying an application with a microservice architecture. The result of our research is a list of some advantages and disadvantages of both microservice and monolithic architecture (Table 1).

Table 1

Comparative characteristics of architectures

Characteristic	Microservice architecture	Monolithic architecture
Reliability	More fault tolerant. Microservices can be easily distributed across multiple nodes. If one node fails, the others can take the load. Microservices are based on the assumption that the network is unreliable.	Simpler, but less fault tolerant. Monoliths have fewer moving parts. Monolithic architecture is suitable for undistributed loads.
Scaling	Easier and more accurate scaling. Only the components required for load handling are scaled.	Scaling is more complex, slower, and less accurate because almost all components are scaled.
Capacity	Responds to requests more slowly because it has network nodes and an additional load of images and orchestration.	Responds faster to requests because the monolithic architecture does not have networked nodes, images, and orchestration.
Agility	More agile. Code updates can be deployed and installed very quickly because microservices are loosely interconnected.	Less agile. Updating the code takes a lot of time and checking before deployment.

Monitoring of network infrastructure

The information technology stack for monitoring network equipment includes Grafana, Prometheus, Loki. Data from services is collected in Prometheus Server and visualized using Grafana. Using Loki allows you to organize and get online access to logs.

The architecture of the project is structured in such a way that a processor load of more than 75 percent is considered unacceptable if the project is working reliably. The metric "Memory load,%" has a threshold value of 80, an exaggeration of which can become critical for the system's performance.

For reliable system operation:

1) planning of cluster large capacities at the stage of project deployment;

2) limiting the number of clients;

3) adaptive increase in cluster capacities.

The first option - excellent, however, rather expensive - involves large material investments at the start of the project.

Limiting the number of clients usually goes against a business strategy.

The adaptive increase in processor power requires flexible management of the available processor resources, which is possible manually (the human factor is a big obstacle to this approach) or by means of Prometheus / Grafana.

On the Grafana side, the "Alert" mechanism is implemented, which launches deployment-patch.yaml, which automatically increases the processor capacity by connecting an additional Node to the micro-environment.

Real time series research of server state

Consider a set of time series of server state in real time. Data were collected over a six-month period in 10 second increments. Table 2 shows the calculations of the Hurst exponent for the time series of server equipment when the window size is varied [31].

The R / S -analysis of changes in the Hurst exponent, presented in Table 2, showed the following characteristic features of these implementations: long-term dependence. The Hurst parameter is significantly greater than 0.5 in all cases, corresponds to a process which is trending (persistent).

Table 2

Hurst parameter of real time series

File name	Window size							Note
	10	50	100	150	200	250	300	
cpu-user.xml	0.79	0.90	0.96	1.03	1.06	1.05	1.07	Using kernel in user space
cpu-wait.xml	0.75	0.85	0.91	0.97	0.94	1.04	1.06	Waiting for kernel
memory-used.xml	0.95	0.96	0.96	0.94	0.91	0.85	0.79	RAM memory usage
memory-cached.xml	0.97	0.93	0.90	0.93	0.88	0.89	0.87	RAM memory caching

Conclusion

We investigated the configuration of the microservice and monolithic application architecture. We concluded that the microservice architecture has much greater fault tolerance and flexibility compared to the monolithic architecture. Applications deployed in a microservice architecture are much easier to scale and update than applications deployed in a monolithic architecture. Application developers with a microservice architecture communicate much more efficiently in smaller teams compared to the large teams required to develop applications with a monolithic architecture.

However, the monolithic application deployment architecture is simpler and great for deploying an application with unallocated workloads. Monolithic architecture responds much faster to requests and has fewer moving parts. Therefore, it may be easier to administer and maintain applications with monolithic architecture.

Time series of server equipment load have been investigated by the methods of fractal analysis. The presence of a long-term dependence is shown, which makes it possible to represent the implementation of traffic as self-similar process.

References

1. Calderon-Gomez, H., Mendoza-Pitti, L., Vargas-Lombardo, M., Gomez-Pulido, J. M., Castillo-Sequera, J. L., Sanz-Moreno, J., & Sencion, G. (2020). Telemonitoring system for infectious disease prediction in elderly people based on a novel microservice architecture. IEEE Access, 8, 118340-118354. doi:10.1109/ACCESS.2020.3005638

2. Meng, L., Sun, Y., & Zhang, S. (2020). Capestor: A service mesh-based capacity estimation framework for cloud applications doi:10.1007/978-3-030-59635-4_17 Retrieved from www.scopus.com

3. Singh, V., & Peddoju, S. K. (2017). Container-based microservice architecture for cloud applications. Paper presented at the Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017, , 2017-January 847-852. doi:10.1109/CCAA.2017.8229914 Retrieved from www.scopus.com

4. Feng, Z., Xu, Y., Xue, X., & Chen, S. (2020). Review on the development of microservice architecture. [微服务技术发展的现状与展望] Jisuanji Yanjiu Yu Fazhan/Computer Research and Development, 57(5), 1103-1122. doi:10.7544/issn1000-1239.2020.20190460

5. Velepucha, V., Flores, P., & Torres, J. (2019). MOMMIV: Model for the decomposition of a monolithic architecture towards a microservices architecture under the principle of information hiding. [MOMMIV: Modelo para descomposición de una arquitectura monolítica

hacia una arquitectura de microservicios bajo el principio de ocultación de información] RISTI - Revista Iberica De Sistemas e Tecnologias De Informacao, (E17), 1000-1009. Retrieved from www.scopus.com

6. Salah, T., Zemerly, M. J., Yeun, C. Y., Al-Qutayri, M., & Al-Hammadi, Y. (2017). The evolution of distributed systems towards microservices architecture. Paper presented at the 2016 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016, 318-325. doi:10.1109/ICITST.2016.7856721 Retrieved from www.scopus.com

7. Deploy to Kubernetes. Docker Documentation. Retrieved 1 November 2020, from https://docs.docker.com/get-started/kube-deploy/.

8. Wan, F., Wu, X., & Zhang, Q. (2020). Chain-oriented load balancing in microservice system. Paper presented at the 2020 World Conference on Computing and Communication Technologies, WCCCT 2020, 10-14. doi:10.1109/WCCCT49810.2020.9169996 Retrieved from www.scopus.com

9. Jayasinghe, M., Chathurangani, J., Kuruppu, G., Tennage, P., & Perera, S. (2020). An analysis of throughput and latency behaviours under microservice decomposition doi:10.1007/978-3-030-50578-3_5 Retrieved from www.scopus.com

10. Xu, Y., & Shang, Y. (2019). Dynamic priority based weighted scheduling algorithm in microservice system. Paper presented at the IOP Conference Series: Materials Science and Engineering, , 490(4) doi:10.1088/1757-899X/490/4/042048 Retrieved from www.scopus.com

11. Lee, G. M., Um, T. -., & Choi, J. K. (2018). AI as a microservice (AIMS) over 5G networks. Paper presented at the 10th ITU Academic Conference Kaleidoscope: Machine Learning for a 5G Future, ITU K 2018, doi:10.23919/ITU-WT.2018.8597704 Retrieved from www.scopus.com

12. Gos, K., & Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. Paper presented at the 2020 IEEE 16th International Conference on the Perspective Technologies and Methods in MEMS Design, MEMSTECH 2020 - Proceedings, 150-153. doi:10.1109/MEMSTECH49584.2020.9109514 Retrieved from www.scopus.com

13. Hasselbring, W., & Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. Paper presented at the Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, 243-246. doi:10.1109/ICSAW.2017.11 Retrieved from www.scopus.com

14. Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture doi:10.1007/978-3-319-74433-9_3 Retrieved from www.scopus.com

15. Rademacher, F., Sachweh, S., & Zündorf, A. (2020). A modeling method for systematic architecture reconstruction of microservice-based software systems doi:10.1007/978-3-030-49418-6_21 Retrieved from www.scopus.com

16. Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. Paper presented at the Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017, 524-531. doi:10.1109/ICWS.2017.61 Retrieved from www.scopus.com

17. Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., . . . Lang, M. (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. doi:10.1007/s11761-017-0208-y

18. Eski, S., & Buzluca, F. (2018). An automatic extraction approach - transition to microservices architecture from monolithic application. Paper presented at the ACM International Conference Proceeding Series, , Part F147763 doi:10.1145/3234152.3234195 Retrieved from www.scopus.com

19. Asrowardi, I., Putra, S. D., & Subyantoro, E. (2020). Designing microservice architectures for scalability and reliability in e-commerce. Paper presented at the Journal of Physics: Conference Series, , 1450(1) doi:10.1088/1742-6596/1450/1/012077 Retrieved from www.scopus.com

20. Jindal, A., Podolskiy, V., & Gerndt, M. (2019). Performance modeling for cloud microservice applications. Paper presented at the ICPE 2019 - Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, 25-32. doi:10.1145/3297663.3310309 Retrieved from www.scopus.com

21. WalidShaari/Kubernetes-Certified-Administrator. GitHub. Retrieved 1 September 2020, from https://github.com/walidshaari/Kubernetes-Certified-Administrator.

22. Kainz, A., Smith, T., & Fiskeaux, C. (2020). Microservices vs. Monoliths: An Operational Comparison – The New Stack. The New Stack. Retrieved 1 September 2020, from https://thenewstack.io/microservices-vs-monoliths-an-operational-comparison/.

23. Phain, C., & Limpiyakom, Y. (2018). Scaling network traffic logger with microservice architecture. Paper presented at the 2018 International Conference on System Science and Engineering, ICSSE 2018, doi:10.1109/ICSSE.2018.8520153 Retrieved from www.scopus.com

24. Kubernetes Certified Administration. Retrieved 2 November 2020, from https://github.com/walidshaari/Kubernetes-Certified-Administrator.

25. John, S. (2020). A Transition From Monolith to Microservices – DZone Microservices. dzone.com. Retrieved 1 November 2020, from https://dzone.com/articles/a-transition-from-monolith-to-microservices.

26. Hou, X., Liu, J., Li, C., & Guo, M. (2019). Unleashing the scalability potential of power-constrained data center in the microservice era. Paper presented at the ACM International Conference Proceeding Series, doi:10.1145/3337821.3337857 Retrieved from www.scopus.com

27. Shimoda, A., & Sunada, T. (2018). Priority order determination method for extracting services stepwise from monolithic system. Paper presented at the Proceedings - 2018 7th International Congress on Advanced Applied Informatics, IIAI-AAI 2018, 805-810. doi:10.1109/IIAI-AAI.2018.00165 Retrieved from www.scopus.com

28. Li, F., & Gelbke, L. (2018). Microservice architecture in industrial sofware delivery on edge devices. Paper presented at the ACM International Conference Proceeding Series, , Part F147763 doi:10.1145/3234152.3234196 Retrieved from www.scopus.com

29. Deployments | Kubernetes. kubernetes.io. (2021). Retrieved 10 January 2021, from https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

30. Engel, T., Langermeier, M., Bauer, B., & Hofmann, A. (2018). Evaluation of microservice architectures: A metric and tool-based approach doi:10.1007/978-3-319-92901-9_8 Retrieved from www.scopus.com

31. V. Bulakh, L. Kirichenko and T. Radivilova, "Time Series Classification Based on Fractal Properties," *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, Lviv, 2018, pp. 198-201, doi: 10.1109/DSMP.2018.8478532.

Tatjana Selivyorstova **Тетяна Селівьорстова**	candidate of technical science, assistant professor, Department of information technology and systems, National Metallurgical Academy of Ukraine. e-mail: tatyanamikhaylovskaya@gmail.com orcid.org/0000-0002-2470-6986, Scopus Author ID: 56996195600, ResearcherID: AAE-4202-2020	кандидат технічних наук, доцент, доцент кафедри інформаційних технологій та систем, Національна металургійна академія України.

	https://scholar.google.com.ua/citations?hl=ru&user=vY3wlUsAAAAJ&view_op=list_works	
Sergey Klishch **Сергій Кліщ**	postgraduate, Department of information technology and systems, National Metallurgical Academy of Ukraine. e-mail: sergey.klishch@gmail.com orcid.org/0000-0002-7799-1004	аспірант кафедри інформаційних технологій та систем, Національна металургійна академія України.
Serhii Kyrychenko **Сергій Кириченко**	master, Department of information technology and systems, National Metallurgical Academy of Ukraine. e-mail: skir79@gmail.com orcid.org/0000-0001-9700-4136	магістр кафедри інформаційних технологій та систем, Національна металургійна академія України.
Anton Guda **Антон Гуда**	doctor of engineering, associate professor, Department of information technology and systems, National Metallurgical Academy of Ukraine. e-mail: atu.guda@gmail.com orcid.org/0000-0003-1139-1580, Scopus Author ID: 57189391377, ResearcherID: AAA-1681-2019 https://scholar.google.com.ua/citations?hl=ru&user=HBeO-N0AAAAJ	доктор технічних наук, доцент, професор кафедри інформаційних технологій та систем, Національна металургійна академія України.
Kateryna Ostrovska **Катерина Островська**	candidate of technical science, assistant professor, Department of information technology and systems, National Metallurgical Academy of Ukraine. e-mail: kuostrovskaya@gmail.com orcid.org/0000-0002-9375-4121, Scopus Author ID: 57220776655	кандидат технічних наук, доцент, доцент кафедри інформаційних технологій та систем, Національна металургійна академія України.