Naumenko, Tetiana; Petrenko, Anatolii

Article

Analysis of problems of storage and processing of data in serverless technologies

Reference: Naumenko, Tetiana/Petrenko, Anatolii (2021). Analysis of problems of storage and processing of data in serverless technologies. In: Technology audit and production reserves 2 (2/58), S. 20 - 25.
http://journals.uran.ua/tarp/article/download/230174/229391.
doi:10.15587/2706-5448.2021.230174.

ISSN 2664-9969

UDC 004.624
DOI: 10.15587/2706-5448.2021.230174
Article type «Reports on Research Projects»

Tetiana Naumenko,
Anatolii Petrenko

ANALYSIS OF PROBLEMS OF STORAGE AND PROCESSING OF DATA IN SERVERLESS TECHNOLOGIES

The object of research is the problems of storing and processing data in serverless technologies. The research carried out is based on a logical approach to storage, data processing and transmission processes. The main hypothesis of the study is that when moving from a monolithic architecture to microservice architecture, and then from a microservice architecture to a serverless architecture, the process of storing and processing data requires modifications and new approaches to solving classical problems of working with data. The problem of interacting with data is an integral part of the work of almost all computer systems, as they lay the basis for the goal of creating such systems. Serverless computing has already taken root in cloud computing. Improving its work is now one of the most popular tasks in the research field. In this article, let's review one of the global problems – integrating serverless computing system with a database. As well as currently existing partial or complete solutions. Progress in this area can give impetus to the development of serverless technologies that supplant more outdated software development approaches. The result of these studies brings a certain understanding at what stage of development the above question is now. It also describes the advantages and disadvantages of the new systems. It is considered what innovations have been brought by the global giants in the development of serverless platforms, and what solutions are applied to open source platforms. This issue has not been fully resolved and requires developments and improvements, and therefore remains an excellent direction for development and new research.

Keywords: *serverless technologies, serverless platforms, data storage and processing, databases.*

Received date: 18.12.2020
Accepted date: 28.01.2021
Published date: 30.04.2021

How to cite
Naumenko, T., Petrenko, A. (2021). Analysis of problems of storage and processing of data in serverless technologies. Technology Audit and Production Reserves, 2 (2 (58)), 20–25. doi: http://doi.org/10.15587/2706-5448.2021.230174

1. Introduction

It's no secret that lately the attention of almost all public cloud providers (Amazon AWS, Microsoft Azure, Google Cloud Platform and others) concentrated on promoting serverless technologies. Based on the trend of providing convenience to software developers, such technologies are gaining popularity at an incredible speed. Software development is usually associated with the management of physical infrastructure and server maintenance. Serverless computing allows developers to forget about administrative costs and infrastructure management, and concentrate on writing code. More specifically, serverless computing allows to create and run services without installing and maintaining servers or clusters, which saves developer resources and project budget. It should be clarified that the term «serverless» does not mean the absence of a server, but simply managing servers by cloud service providers using automated systems.

Serverless computing can be viewed as event-based programming, since they use a function as the deployment unit. Event-oriented programming means that the exploitation of the resources of a cloud provider stops when software developers do not perform any functions or applications. In addition, serverless architecture provides automatic scaling, allowing users to provide services, despite the increased workload. Since the main role in this architecture is performed by the function performed by the platform, serverless computing is called «Function as a service» (FaaS). In simple words, for the execution of each request, a separate container is created, and destroyed after execution.

But is everything so smooth and ideal for a quick transition to such an attractive way to create software? Despite the fact that the technology is relatively new, the research community is actively working on studying it and ways to improve it. In addition to the noticeable advantages of serverless computation, shortcomings have already been discovered, which in some cases call into question the rationality of switching to this technology.

Advantages of serverless computing:

– Payment for the amount of resources consumed. Computing resources are paid only when they are actually used. This means that developers pay for the time when their application performs the functions of a user in response to specific events or requests.

– No server management. There is no need to allocate and maintain servers, no installation, maintenance or administration of software or runtime is required.

– Improved resource management. The cloud computing provider can exactly match the abstract demand for actual system resources. When an application does not start, the cloud provider distributes server resources among other running applications.
– Flexible scaling. Serverless architecture has the ability to scale in accordance with the workload of the application. Developers no longer need to purchase additional infrastructure to serve unanticipated growth.
– Agile development. Using serverless computing allows to focus on application code, rather than infrastructure maintenance. This benefits developers by reducing software complexity and better code optimization.
– Multi-language support. Many platforms support several programming languages, so users can choose the most convenient for them.
– Automatic provision of high availability. Serverless computing provides built-in high availability and fault tolerance. These features do not need to be specifically designed, since the services that run the application provide them by default.

Disadvantages of serverless computing:
– Cost. The fact that serverless computing can significantly reduce development costs is not always true. For example, serverless functions are currently the most attractive for CPU-related computing. While I/O related functions are less accessible than on dedicated virtual machines or containers.
– Cold start. The key advantage of serverless computing is the ability to scale to zero, which can lead to an increase in call delay due to cold start.
– Resource constraint. Resource constraints are necessary in order for the platform to cope with jumps and resist attacks. Serverless computing imposes restrictions on memory, function execution time, bandwidth and CPU usage.
– Security. There is a high risk of unsafe operation, since many users run their functions on a common platform. Thus, when downloading malicious code, there may be negative consequences for the operation of other applications.
– Monitoring and debugging. Since users do not have control over the provider's servers, there are limited opportunities to identify problems and bottlenecks.
– Availability of skills. Serverless Computing is a fairly new area, so there are often not enough developers with serverless programming skills.

In [1–3] describe well the approach to using serverless computing and the developed serverless platforms, but do not touch upon the specifics of the databases. This fact casts doubt on the transition from microservice technologies to serverless for software, an integral part of which is data storage and processing.

Therefore, it is relevant to research and analyze the work of serverless platforms with data. Thus, *the object of research* is the problems of storing and processing data in serverless technologies. And *the aim of research* is to find out and analyze what ready-made solutions are presented in serverless platforms.

2. Methods of research

The following scientific methods are used:
– analysis method when studying data on existing topics among scientific articles, forums, official sites of serverless platform providers;
– conversation method when clarifying the main features of the work of serverless databases at conferences from commercial suppliers of serverless platforms;
– synthesis method for identifying the necessary characteristics of databases that must be present in the architecture of serverless platforms;
– analogy method when highlighting the disadvantages of existing solutions when moving from a monolithic or microservice architecture to a serverless architecture.

3. Research results and discussion

3.1. Database integration in serverless computing. As the first on the market of publicly accessible servers AWS were and remain the largest one, so it is not surprising that they were the first to launch the serverless computing platform called AWS Lambda. Lambda deals with large object storage – S3, key value storage – DynamoDB, queue service – SQS, notification service – SNS, and more. It is thanks to their precedence, AWS Lambda is now considered to be the most attractive to the research community. Recently, researchers from UC Berkeley published an article [4], which described the current problems, disadvantages and consequences of these shortcomings, which they found in cloud computing, based on the AWS Lambda platform. This does not mean that other representatives of cloud computing do not have these disadvantages, since they are all similar in terms of technology, although they differ in details. Describing in detail the limitations that are present today in the FaaS proposals (which were also indicated above), the authors derived a list of the ensuing consequences. And more specifically:
– FaaS Stymies Distributed Computing;
– FaaS is a Data-Shipping Architecture;
– FaaS stymies hardware-accelerated software innovation;
– FaaS discourages Open Source service innovation [4].

The first two points are directly related to the topic of this article. It would be better to describe in more detail what is meant. The first consequence emphasizes the fact that the lack of network addressing in serverless functions leads to the fact that the two functions can work together, transferring data through slow and expensive storage. This stymies basic distributed computing. The second consequence is that serverless functions are performed on isolated virtual machines separately from the data. Also, serverless functions are short-lived and non-addressable, so their capacity to cache state internally to service repeated requests is limited. The authors of the article describe the consequence of such an approach as an architectural anti-pattern: «ships data to code» rather than «shipping code to data» [4]. From the point of view of this approach to working with data, the hierarchy of memory makes it quite a bad design decision for reasons of delay, bandwidth and cost (and this contradicts the first advantage of FaaS, which refers to cost reduction).

Relying on such a pressing and clearly distinguished problem, the research community began an active work in the search for a solution. And there is progress! For example, in [5] authors talk about some *Elastic Database Systems*. Their assumption is that the design principles underlying the «stateless» FaaS platforms point the way to system abstractions that offer transformative improvements in the flexible scalability of state-intensive applications. Concentrating on the problem of the lack of a way to

ISSN 2664-9969

address a function in FaaS, the authors propose extending FaaS by partitioning the function executing instances across an invocation key space [2].

The authors claim that: «As illustrated in Fig. 1, the serverless runtime establishes a partitioning on the space of keys, then routes function invocations for each key to the same instance consistently. Additional configuration allows users to specify whether partitions may be served by only one instance, or by many, e. g., to support replicas for read scalability. If a partition becomes overloaded, the partitioned FaaS (pFaaS) runtime terminates the associated function instance, splits its key set, and creates two or more new ones in its place. Merging key sets for scale-in works similarly» [5].

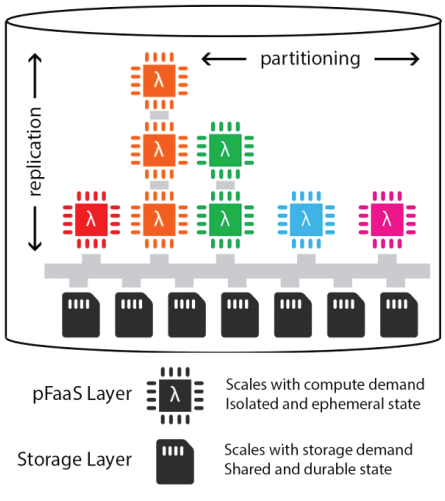

Fig. 1. An elastic database with a shared disk architecture built from partitioned FaaS (pFaaS) serverless compute and serverless storage (e. g., AWS S3 or EFS) [5]

This approach resembles the work of the virtual system of actors [6], which leads to the possible further approach Actors as a Service (AaaS) approach. How realistic it is, is not clear, since this is only a theory, not confirmed by practice yet.

3.2. Serverless Database. Restrictions on the use of databases in applications built on serverless computing technology lead to the fact that it is necessary a completely new approach to creating a database. And since the principle of building architecture has moved from monolith to microservices, and now completely to a set of functions, a similar solution should be implemented for databases, based on the same principles: lack of maintenance, payment only for execution and global distribution. This results in serverless databases.

In [7, 8] it is possible to track the transition from the usual and familiar to us databases to a new generation of serverless databases from the main suppliers of cloud technologies (such as AWS, Azure, Google Cloud Platform, etc.). The authors describe the problem of dividing the work of both SQL databases and NoSQL, relying on the same problems and limitations. Thus, taking to this line of presentation of new products, describing their advantages in the form of solving tasks, but without mentioning the disadvantages. This is due to the fact that the products are only presented and have not yet been fully studied, in order to understand whether they are absolutely fascinating to all areas in which problems and restrictions have somehow arisen.

These restrictions are why it is necessary a new database for the serverless age. This next generation of database should share the same principles as serverless computing – *global distribution, pay-per-execution pricing, zero maintenance, data replication and synchronization, sequence* [8].

The most popular solutions at the moment:
- Amazon Aurora Serverless;
- Azure Cosmos DB;
- Fauna DB.

3.2.1. Amazon Aurora Serverless. Amazon Aurora Serverless is an on-demand, auto-scaling configuration for Amazon Aurora (MySQL-compatible edition), where the database will automatically start up, shut down, and scale capacity up or down based on your application's needs. It enables to run your database in the cloud without managing any database instances. It's a simple, cost-effective option for infrequent, intermittent, or unpredictable workloads [9].

Manually managing database capacity can take up valuable time and can lead to inefficient use of database resources. With Aurora Serverless, it simply create a database endpoint, optionally specify the desired database capacity range, and connect your applications. It is necessary to pay on a per-second basis for the database capacity you use when the database is active, and migrate between standard and serverless configurations with a few clicks in the Amazon RDS Management Console. Advantages [9]:
- Simple;
- Scalable;
- Cost-effective;
- Highly available.

3.2.2. Azure Cosmos DB. A database for building blazingly fast, planet scale applications with native support for NoSQL. Azure Cosmos DB was built from the ground up with global distribution and horizontal scale at its core. It offers turnkey global distribution across any number of Azure regions by transparently scaling and replicating your data wherever your users are. Elastically scale your writes and reads all around the globe, and pay only for what you need. Azure Cosmos DB provides native support for NoSQL and OSS APIs including MongoDB, Cassandra, Gremlin and SQL, offers multiple well-defined consistency models, guarantees single-digit-millisecond read and write latencies at the 99th percentile, and guarantees 99.999 high availability with multi-homing anywhere in the world – all backed by industry-leading, comprehensive service level agreements (SLAs) [10]. Advantages:
- Turnkey Global Distribution;
- Limitless and elastic scalability of writes and reads;
- Guaranteed low latency at 99th percentile;
- Well-defined consistency choices;
- Multi-model with native support for NoSQL APIs;
- Enterprise-grade performance and security.

Fig. 2 illustrates architecture of globally distributed, scalable serverless applications using Azure Functions and Azure Cosmos DB.

3.2.3. Fauna DB. FaunaDB Cloud is a serverless managed offering of FaunaDB in the cloud that can be accessed via a function-as-a-service provider.

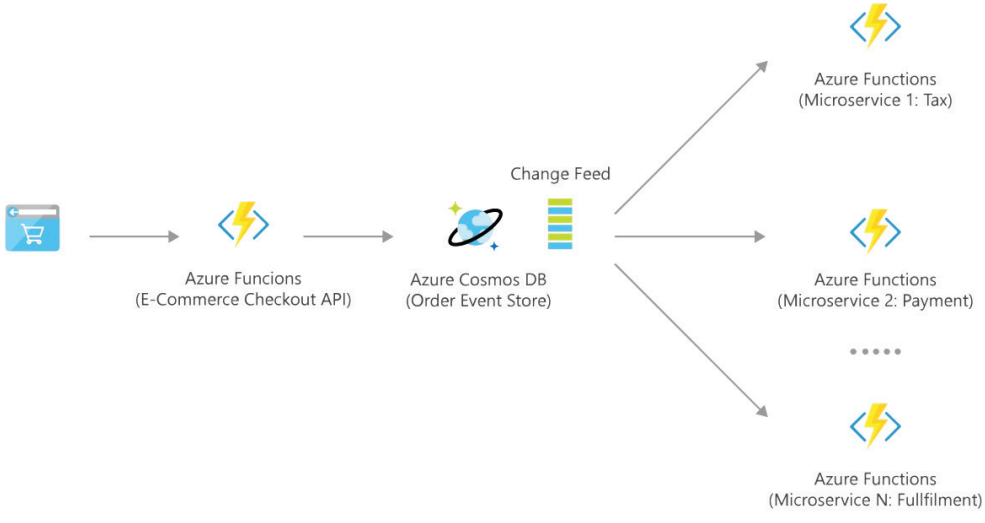

Fig. 2. Serverless apps using Cosmos DB [11]

In FaunaDB Cloud, data is replicated across multiple datacenters running on Amazon Web Services, Google Cloud Platform, and, Microsoft Azure [12]. Advantages [13]:

– Unified Multi-Model;
– 100 % Distributed ACID;
– High Security;
– Horizontal Scalability;
– Multi-Tenancy;
– Operational Simplicity;
– Temporality;
– Fault Tolerance.

At first glance, it seems that these three representatives of serverless databases provide an ideal future for the field of serverless technologies. Of course, it is not possible immediately rely on the fact that they are the final solution and can be used anywhere and in any way. But it is also impossible not to agree with the fact that their appearance – this is a big leap in the database for the improvement of their integration into the serverless space.

3.3. Database as a Service. Despite such an attractive solution, which is described in the previous section, there are still supporters of other approaches.

As a variant of the reason for their appearance, wit is possible to assume that the developers decided not to wait for decisions from the market leaders, but to try to invent their own ways to solve the problem themselves.

So in [14], the authors, hooking on the problem of connection delay to the database, paid attention to solutions from OpenFaaS using connection pools, since they support a set of established connections to the database in memory for future queries.

The authors also mentioned alternative ways to reduce latency:

1. Process the work asynchronously (using NATS Streaming to be able to fulfill requests some time after they are accepted).

2. Build a microservice (initial cost absorption in the long-running microservice, sqlrest created in OpenFaaS – Golang microservice for T-SQL).

3. Use a managed DB service (using the database as third-party software, such as Firebase Realtime; «database as a service» approach).

Solutions of this kind require minimal changes to the interfaces, but are not designed to address a global problem.

A similar approach can be seen in [15]. The author focuses on the proposal to use the «database as a service» approach on the example of the transfer of the well-known MongoDB database to MongoDB Atlas. This solution has the following advantages:

– Possibility to have automated operations;
– Role-based access controls;
– Authentication;
– Scalability;
– The clusters are geo-distributed and they come with excellent fault tolerance.

Despite the list of advantages, this approach still has a negative side. Not using the databases very often can cause the database to suffer from more significant response latency compared to a database that's actively running on a dedicated server, virtual machine, or in a container [15].

3.4. Serverless Database Platform. Another approach to databases in serverless technologies is presented in Fig. 3.

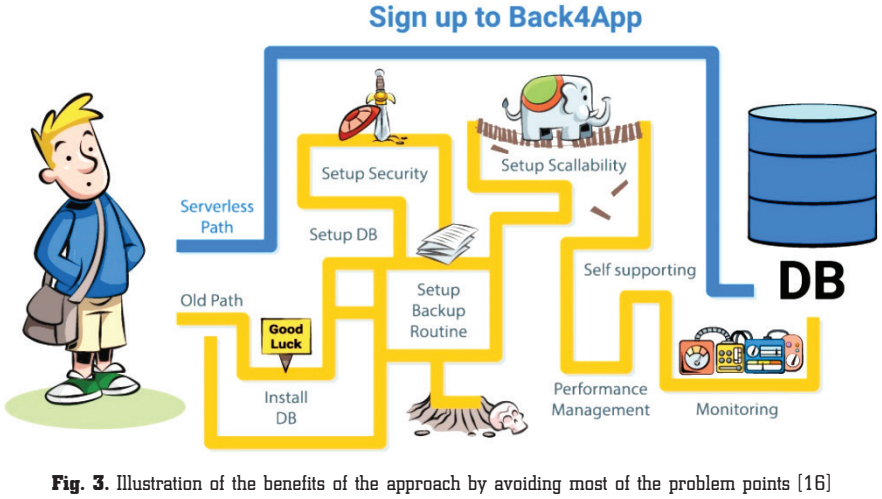

Fig. 3. Illustration of the benefits of the approach by avoiding most of the problem points [16]

ISSN 2664-9969

The authors describe a full-featured serverless platform that will help create serverless applications and serverless databases, suggesting the use of Back4App.

The advantages of such a database include the following [16]:

– No server maintenance;
– Automatic and Flexible scaling;
– Built-in availability;
– No payment for idle capacity.

3.5. Embedded Serverless Database. A special vision of the serverless database concept was proposed in [17]. The author claims that «by really serverless», it means «lambda-embedded databases» or «function-as-a-service-embedded databases», to make it generic [17].

As the positive aspects of this approach highlighted:

– Databases are scaled along with lambda scaling;
– High performance due to the lack of network hops or serialization/deserialization for data access;
– Reducing components means reducing problems with their connections.

But this solution imposes its own restrictions on its use:

– The data must be relatively infrequently updated;
– Read-only data is considered ideal;
– The entire payload (including code and whatever data is being stored) should be under 250 MB unzipped, and ideally 50 MB zipped;
– Low memory profile for lower cost;
– Queries should be difficult to cache.

Tasks that can be solved in this way exist and the author cites them as an example (Geospatial Databases, Security Screening, Spell Checking/Typo Correction, Caching The Hot Head, Graph Databases), so do not be afraid of such a number of restrictions.

3.6. Discussion. Each described approach to solving the problem of integrating a database into serverless technology has its advantages, and some of them, which have already passed the tests, have disadvantages or limitations. Returning to the initial requirements for a new generation of databases, it is possible to conclude in the form of Table 1.

According to the Table 1, leading positions were headed by serverless technologies from leading companies in the field of serverless technologies.

But do not forget that they are new and are not fully understood and tested, which means that there are no disadvantages and limitations for them yet.

It is impossible not to emphasize the fact that each of the solutions implies that the project will be fully deployed on the provided platform. And since they differ from each other in basing on different types of databases (SQL, NoSQL etc.), users will not be able to choose between them. Perhaps, it is worth thinking about standardization for such serverless databases.

Despite the fact that the new databases, due to their advantages, make it possible to solve previously discovered problems of their integration with serverless computing, the more global problems of serverless computing still remain open. If the cost of the project with the use of new databases is reduced, then such concepts as «cold start», monitoring, debugging and security become only more relevant.

4. Conclusions

Overall, the results show that the serverless computing giants are actively addressing the storage and use of data in serverless technologies. The data collected and analyzed in this article sheds light on the current state of affairs in this area, which has not been previously done. The analysis leads to the following conclusions:

– ready-made commercial products provide the ability to work with data, providing the necessary functionality from the point of view of a serverless architecture;
– private solutions allow users to independently implement processes for working with data, but do not have a wider range of capabilities, in contrast to commercial products;
– despite the leap in solving this problem, important aspects of working with data, such as cold start, security, state tracking, etc., remain unaffected.

The comparison results of the solutions presented in the article provide a basis for manipulating the choice of one or another serverless platform for solving specific problems. In contrast to the information that existed earlier, where only advantages are indicated, this article also describes disadvantages. This data will allow to calculate risks even at the stage of system design.

As recommended above, further research should focus on solving the problems that remain in the industry.

Table 1

Comparison of databases that are used in serverless architectures by their main characteristics

Main characteristics	Elastic Database Systems	Serverless Database			Database as a Service (DBaaS)		Back4App	Embedded Serverless Database
		Serverless Aurora	Azure Cosmos DB	Fauna DB	Open FaaS	Mongo DB Atlas		
Pay-per-execution	×	✓	✓	✓	×	×	×	✓
Global distribution	×	✓	✓	✓	×	✓	×	✓
Zero maintenance	×	✓	✓	✓	×	×	✓	×
Data replication and synchronization	✓	✓	✓	✓	×	×	×	×
Work with the same data of two or more functions	✓	✓	✓	✓	✓	✓	×	×

References

1. Enes, J., Expósito, R. R., Touriño, J. (2020). Real-time resource scaling platform for Big Data workloads on serverless environments. *Future Generation Computer Systems, 105,* 361–379. doi: http://doi.org/10.1016/j.future.2019.11.037
2. Giménez-Alventosa, V., Moltó, G., Caballer, M. (2019). A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Generation Computer Systems, 97,* 259–274. doi: http://doi.org/10.1016/j.future.2019.02.057
3. Yussupov, V., Soldani, J., Breitenbücher, U., Brogi, A., Leymann, F. (2021). FaaSten your decisions: A classification framework and technology review of function-as-a-Service platforms. *Journal of Systems and Software, 175,* 110906. doi: http://doi.org/10.1016/j.jss.2021.110906
4. Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., Wu, C. (2018). Serverless Computing: One Step Forward, Two Steps Back. *CIDR'19*. Available at: https://arxiv.org/pdf/1812.03651.pdf
5. Schleier-Smith, J. (2019). Serverless Foundations for Elastic Database Systems. *Conference on Innovative Data Systems Research (CIDR)*. Available at: http://cidrdb.org/cidr2019/gongshow/abstracts/cidr2019_140.pdf
6. Bernstein, P. A., Bykov, S., Geller, A., Kliot, G., Thelin, J. (2014). Orleans: Distributed virtual actors for programmability and scalability. *MSR-TR-2014–41*. Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Orleans-MSR-TR-2014-41.pdf
7. DeBrie, A. *Serverless Aurora: What it means and why it's the future of data*. Serverless Blogs. Available at: https://serverless.com/blog/serverless-aurora-future-of-data/
8. Winnicki, M. *Serverless Database Wish List – What's Missing Today*. Serverless Blogs. Available at: https://serverless.com/blog/serverless-database-wish-list/
9. *Amazon Aurora Serverless*. Available at: https://aws.amazon.com/rds/aurora/serverless
10. *Azure Cosmos DB – Globally distributed, multi-model database service*. Available at: https://azure.microsoft.com/en-us/services/cosmos-db/
11. *Serverless apps using Cosmos DB*. Available at: https://docs.microsoft.com/bs-cyrl-ba/azure/architecture/solution-ideas/articles/serverless-apps-using-cosmos-db
12. Ramel, D. (2017). *FaunaDB takes First Serverless Database to the cloud*. ADTmag. Available at: https://adtmag.com/articles/2017/03/16/faunadb-serverless-cloud.aspx
13. *FaunaDB: A fundamental shift in database technology*. Available at: https://fauna.com/faunadb
14. Ellis, A. (2018). *Serverless: Databases with OpenFaaS and Mongo*. Alex Ellis' Blog. Available at: https://blog.alexellis.io/serverless-databases-with-openfaas-and-mongo/
15. Novkovic, N. (2018). *What Is a Serverless Database? (Overview of Providers, Pros, and Cons)*. Available at: https://dzone.com/articles/what-is-a-serverless-database-overview-of-provider
16. Melo, A. *A Serverless Database Platform*. The Back4App Blog. Available at: https://blog.back4app.com/2017/12/28/serverless-database/
17. Barratt, J. (2018). *Really Serverless Databases*. Josh Barratt's Blog. Available at: https://serialized.net/2018/07/serverless_db/

Tetiana Naumenko, *Postgraduate Student, Department of System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Kyiv, Ukraine, ORCID: https://orcid.org/0000-0002-8660-597X, e-mail: tnaumenko13@gmail.com*

Anatolii Petrenko, *Doctor of Technical Sciences, Professor, Department of System Design, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Kyiv, Ukraine, ORCID: https://orcid.org/0000-0001-6712-7792, e-mail: tolja.petrenko@gmail.com*