Microservices: architecture, container, and challenges

Guozhi Liu
School of Big Data and Intelligent Engineering
Southwest Forestry University
Kunming, China
1499938459@qq.com

Bi Huang
School of Big Data and Intelligent Engineering
Southwest Forestry University
Kunming, China
125149146@qq.com

Zhihong Liang
School of Big Data and Intelligent Engineering
Southwest Forestry University
Kunming, China
zhliang@swfu.edu.cn

Minmin Qin
School of Big Data and Intelligent Engineering
Southwest Forestry University
Kunming, China
swfuqmm@qq.com

Hua Zhou
School of Big Data and Intelligent Engineering
Southwest Forestry University
Kunming, China
hzhou@swfu.edu.cn

Zhang Li*
School of Continuing Education and International Exchange
Yunnan Forestry Technological College
Kunming, China
779411989@qq.com
corresponding author

***Abstract*—Microservices are emerging as a new computing paradigm which is a suitable complementation of cloud computing. Microservices will decompose traditional monolithic applications into a set of fine-grained services, which can be independently developed, tested, and deployed. However, there are many challenges of microservices. This paper provides a comprehensive overview of microservices. More specifically, firstly, we systematically compare traditional monolithic architecture, service-oriented architecture (SOA), and microservices architecture. Secondly, we give an overview of the container technology. Finally, we outline the technical challenges of microservices, such as performance, debugging and data consistency.**

Keywords—microservices, debugging, container, performance, monolithic architecture, service-oriented architecture.

I. INTRODUCTION

Microservices are new architectural styles [1]. Microservices have the following characteristics: independent development, independent deployment, independent release, high concurrency, high availability, high cohesion, and low coupling [1]. Based on the application of traditional monolithic architecture, with the iteration of business requirements and the additional expansion of functions, the application is difficult to expand and highly coupled [2]. In recent years, the mobile Internet of Things has developed rapidly, the scale of major companies has continued to expand, and business has developed rapidly. Applications such as search engines, e-commerce websites, and chat software all require high concurrency, high availability, high scalability, high cohesion, and low coupling. The traditional monolithic architecture will not meet the requirements [3]. Microservices by refining a monolithic application into multiple fine-grained microservices, each service can be independently developed, deployed, extended, and tested. To meet the requirements of the above applications.

In recent years, microservice has become one of the latest architectural trends in the field of software engineering, and more and more cloud computing applications have begun to adopt microservices [4][5]. In foreign countries, Amazon, Netflix, The Guardian, Twitter, PayPal, SoundCloud, and others began to microservices software systems on the cloud. In China, Tencent, Baidu, Jingdong, Taobao, 360 Search, etc. have also begun to migrate software systems microservices architecture. For example, Tencent's WeChat system [6] contains more than 3,000 microservices, distributed on more than 20,000 machines [7]. Netflix's online service system uses more than 500 microservices, involving 5 billion service interactions per day [8]. Each page of Amazon involves 100-150 microservice calls [9].

With the iteration of business requirements and the additional expansion of functions, traditional monolithic applications lead to problems such as difficulty in expanding applications, high coupling, and high deployment costs [1]. The emergence of microservice technology will solve the above-mentioned problems encountered by traditional monolithic applications. This article will introduce microservices from two aspects of development style and deployment. From the aspect of architecture, we introduce the process of adopting different architectures from monolithic applications to componentization to microservices, to achieve decoupling and high expansion of server-side applications. However, after the traditional monolithic application is split into multiple microservices, decoupling, high expansion, and rapid development iteration can be achieved, but the problem that comes with it is the increase in the cost of testing and operation and maintenance deployment [1]. The emergence of container technology is undoubtedly an icing on the cake. We will introduce the use of container technology to achieve resource isolation and independent deployment of microservices from the perspective of deployment and operation. To achieve the single responsibility of microservices, service autonomy, lightweight communication, and reduce operation and maintenance deployment costs [3].

The main contributions of this work can be summarized as follows.

- We present a comprehensive review of microservices architectures.
- We give an overview of the container technology.
- We outline three technical challenges of microservices.

The rest of this paper is organized as follows. Section 2 describes the evolution of application architecture patterns

978-1-7281-8915-4/20/$31.00 ©2020 IEEE
DOI 10.1109/QRS-C51114.2020.00107

from traditional monolithic architecture, SOA to microservices architectures. Section 3 describes the container technology. Section 4 outlines the three challenges microservices faced: performance, debugging and data consistency. Section 5 summarizes this paper.

II. ARCHITECTURE

With increasing complexity and the need for highly scalable and robust applications, the traditional monolithic architecture is no longer the best choice. After a certain threshold, the monolithic architecture often hinders the performance and scalability of the application. Besides, due to the huge codebase, changes to closely coupled related processes in a monolithic architecture will greatly increase the impact of a single process failure.

To cope with these limitations of a single architecture, developers adopted the principle of single responsibility proposed by Robert C. Martin (co-author of the Agile Manifesto). The principle says: bring together those that change for the same reason, and separate those that change for different reasons.

Eventually, Service Oriented Architecture (SOA) and microservice architectures were recognized and enabled developers to build applications as a set of small, decoupled services that run in their environment and can be deployed independently.

Let us look at the evolution of application architecture patterns from traditional monolithic architecture, SOA to microservices architecture.

A. *Monolithic Architecture*

The monolithic architecture is a traditional method of software development, which has been used by large companies such as Amazon and eBay in the past. In a monolithic architecture, functions are encapsulated in an application. When a whole is small and has only a few functions, it can have its advantages, such as ease of development, testing, deployment, and expansion [10]. For the monolithic architecture, if we need to expand, we only need to copy the whole. However, as applications tend to become more complex, weaknesses appears [10]. For example, high complexity, poor reliability, limited scalability, and hindering technological innovation. As shown in figure1, when a traditional monolithic architecture is used to develop an application, the user interacts with the front-end application. The front-end application redirects the user request to the software instance hosted in the container and interacts with the database to complete all applications. Procedural responsibilities [11].

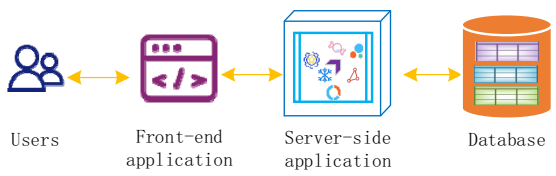

Fig. 1. Monolithic Architecture.

B. *Service-oriented architecture(SOA)*

In the 1990s, SOA was proposed as a revolutionary innovation to decouple service-side applications and improve the reuse of components [12]. As shown in figure2, the SOA architecture could be divided into multiple server application oriented function of loosely coupled services, each service can be managed in different containers, between services through an enterprise service bus to communicate, and share the same database [13].

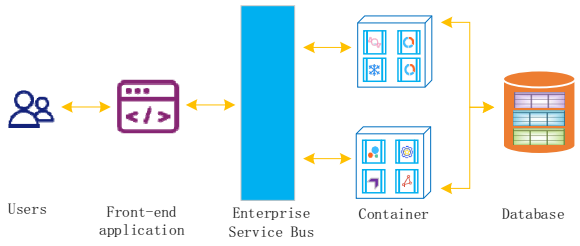

Fig. 2. Service-Oriented Architecture.

C. *Microservices Architecture*

Microservices inherited the principles and concepts of the service-oriented architecture (SOA) style, and structure a service-based application into a very small set of loosely coupled software services [10]. In order to further decouple the service side applications, the microservices architecture proposes to divide the service side applications into several loosely coupled services oriented to business responsibilities [14]. In figure3, the server application is further divided into multiple fine-grained microservices, each service to achieve a given business responsibilities, and managed to run in different containers [15]. Each container has its own private database that cannot be accessed directly by other containers [16].

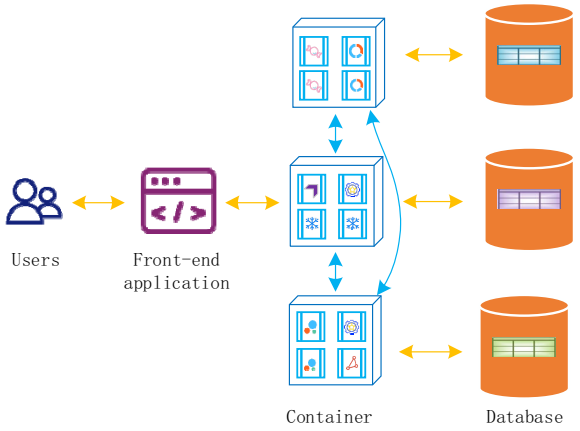

Fig. 3. Microservices Architecture.

D. *Summary*

As shown in table 1, we compare traditional monolithic architecture, Service-oriented architecture, and microservices architectures in terms of componentization, component size, elasticity, deployment, storage mechanisms, technology and scalability.

TABLE I. COMPARISON OF DIFFERENT ARCHITECTURES

	Monolithic Architecture	*Service-Oriented Architecture*	*Microservices Architecture*
Componentization	Module	Service	Microservice
Component size	Big	Coarse-grain	Fine-grained
Elasticity	A single point of failure	No single point of failure.	No single point of failure.
Deployment	Holistic creation and deployment.	Each component deployed independently.	Each service is built and deployed independently.
Storage mechanism	Shared database.	Shared database.	Private database.
Technology	The same programming language and framework.	Isomorphism	Heterogeneous
Scalability	Unable to scale on demand.	Scale on demand.	Scale on demand.

III. TECHNOLOGY OF CONTAINER

The container is a lightweight virtualization technology. The virtualization of microservices has been the key to improving the performance of cloud applications [17]. The traditional virtual machine is a method of virtualization, and the container is another emerging virtualization technology. Due to its high performance, lightweight, and higher scalability [17], it is becoming more and more popular in virtual technology. Applications developed using microservice technology contain many independent decoupling services. These services have their development frameworks, and each service performs specific tasks of independent development and deployment [15]. Because microservices are independent of each other, you can use a virtualized environment to sandbox these microservices, making the system more secure and easier to manage.

In the following, we will make a comprehensive review of container technology. In particular, firstly, we elaborated on the concepts of the container. Secondly, we introduced the container-based cluster architecture. Thirdly, we introduced the core technology of containers-virtualization. Finally, we introduced the containerized deployment of microservices.

A. Concepts of the container

The application and its run-time dependent environment are packaged and packaged into a standardized and strongly portable image [18]. The container engine provides a running environment with process isolation and limited resources to realize the decoupling of the application from the OS platform and underlying hardware. The application is packaged once and runs everywhere [19]. The Container can be deployed on a physical machine or virtual machine. Container engine and container orchestration scheduling platform can realize the life cycle management of the containable application [18].

Container technology based on namespaces and cgroups. Namespace isolation allows process groups to be separated, preventing them from seeing resources in other groups. Container technology uses different namespaces to isolate processes, network interfaces, access interprocess communication, mount points, isolate kernel, and version identifiers. The control group manages and restricts resource access to the process group by restricting enforcement, accounting, and isolation—for example, by restricting the available memory of specific containers [18].

B. Container-based Cluster architecture

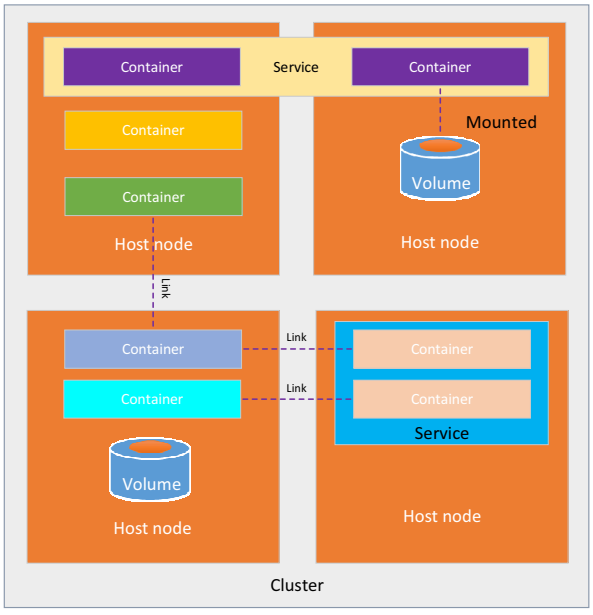

Fig. 4. Container-based cluster architecture.

Containerization simplifies the steps from a single application in a container to the ability to run containerized applications across clustered hosts [18]. The latter benefits from the container's built-in interoperability. Individual container hosts are grouped into interconnected clusters, as shown in figure4. Each cluster consists of several (host) nodes. Application services are logical groups of containers from the same image. Application services allow applications to be extended across different host nodes [20]. A volume is a mechanism for applications that require data persistence. Containers can mount these volumes for storage. Linking allows two or more containers to connect and communicate [18]. The setup and management of these container clusters require choreography support for inter-container communication, linking, and service assemblies.

As shown in figure4. Container linking allows multiple containers to be linked together and send information between them. Linked containers can transfer data about themselves through environment variables. To establish links and certain types of relationships, Docker relies on the name of the container, which must be unique, which means that links are usually limited to containers on the same host.

C. *Virtualization*

Resource virtualization includes to use an intermediate software layer above the underlying system to provide abstractions of multiple virtual resources [20]. Generally, a virtual resource is called a virtual machine (VM) and can be regarded as an independent execution context. The most common form of hypervisor-based virtualization (hosted virtualization) consists of a virtual machine monitor (VMM) that sits on top of the host operating system (OS), which provides a complete abstraction for VMs. In this case, each VM has its operating system, and its execution is completely independent of other VMs. For example, this allows multiple different operating systems to run on one host. There are various virtualization technologies. At present, the most popular is hypervisor-based virtualization, which is mainly represented by Xen, VMware, and KVM.

Xen is an open-source project developed by the Cambridge University Computer Lab [21][22]. It is a software layer that directly runs on the computer hardware to replace the operating system [22]. It can run multiple guest operating systems (Guest OS) concurrently on the computer hardware. Xen uses the ICA protocol. High performance is achieved through a technology called paravirtualization [23]. Even on some architectures that are extremely unfriendly to traditional virtualization technologies (such as x86), Xen also performs well.

Vmware uses full virtualization technology [24][25], so there is no need to modify the original operating system, and it can support different operating systems at the same time. The Guest OS task runs on the hardware and cannot sense other Guest OS [24]. Vmware's host VM mode divides the virtual software into two parts. One part is VMM, used for virtual CPU. The other part is an app that uses OS for device support and a VM driver placed inside OS as an intermediary between app and OS [25]. The VMware virtual machine is installed on the host OS, and the host OS provides a good device driver.

KVM stands for Kernel-based Virtual Machine. It is a kernel-based virtualization technology [26][27]. It is a virtualization module embedded in the system. It uses virtualization technology by optimizing the kernel. This kernel module makes Linux a hypervisor, and the virtual machine uses Linux. Managed by its scheduler [26]. The virtual machine in KVM is implemented as a regular Linux process, which is scheduled by a standard Linux scheduler; each virtual CPU of the virtual machine is implemented as a regular Linux process [28]. This allows KVM to use the existing features of the Linux kernel. However, KVM itself does not perform any hardware emulation and requires the client space program to set the address space of a guest virtual server through the /dev/kvm interface, provide it with simulated I/O, and map its video display back to the host display Screen [27][28].

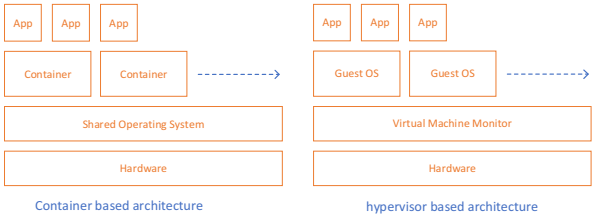

Fig. 5. Comparison of container- and hypervisor-based virtualization architectures.

The lightweight alternative to the hypervisor is container-based virtualization, also known as OS-level virtualization. This virtualization partitions the resources of the physical machine, thereby creating multiple isolated user-space instances on the same OS. Nevertheless, under these circumstances, users still have the illusion that they are working on their separate network, memory, and file system subsystems [29]. The main difference is that there is no need to convert instructions from the upper layer to the lower layer in the hypervisor. For this, a virtual driver is required. In figure5, the differences between container-based and hypervisor-based architectures are depicted [18]. It can be seen that although virtualization based on hypervisors provides an abstraction for a complete guest OS (one for each virtual machine), container-based virtualization works at the OS level, directly providing an abstraction for guest applications. The hypervisor works at the level of hardware abstraction, while the container works at the system call/ABI layer [18]. Because container-based virtualization works at the operating system level, all containers share an operating system kernel. Therefore, the container-based system is weaker than the hypervisor-based system. However, from the user's perspective, each container looks and executes exactly like an independent OS [29].

D. *Containerized deployment of microservices*

For microservices, containers are superior to virtual machines in many aspects. Containers are lightweight, easy to manage, and have fast startup times. They can fundamentally reduce the downtime of enterprise-level applications, thereby reducing the deployment of microservice-based applications [17],the workload and cost.

As shown in figure6, containerized deployment of microservices is mainly divided into four steps: microservice development, microservice container image construction,

microservice container image management, and microservice container orchestration management.

- Development of microservices: services have their development frameworks and each service independent development.
- Building a microservice container image: Package microservices into multiple container images.
- Manage microservice container images: Optimize microservices container image, for example, reduce the size of container image. Then, use DockerHub to manage container images.
- Microservice container orchestration management: Select the appropriate container management tool, To deploy and manage the microservice container through the writing of the configuration file. So that the microservice instance runs normally in the respective container and maintains good interaction. To the entire system, application runs normally and completes the microservice Containerized deployment.

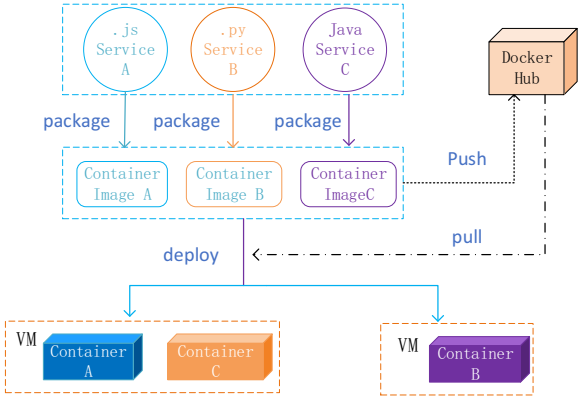

Fig. 6. Containerized deployment of microservices.

Through container technology, the deployment of applications will become more convenient. A single container does not need to host a complete application, and a whole application is refined into multiple fine-grained microservices. Use containers to perform fine-grained microservices. Independently deploy and implement given business responsibilities. Therefore, container deployment will be an ideal choice for microservice deployment.

IV. CHALLENGES

A. Performance

Performance is defined as a software quality attribute that includes software system behavior [30]. Performance is an important but often overlooked aspect of software development methods. Performance refers to the system's responsiveness: the time required to respond to a specific event, or the number of events processed within a given time interval [31].

The microservices architecture requires continuous interaction between the services that make up the application. At the same time, each service needs to use the communication interface between its services to perform transaction operations[32]. Therefore, network communication is a negative factor affecting the performance of microservices.

Lightweight communication mechanisms are used to communicate between microservices. When a service provider cannot be called due to network reasons, subsequent service consumers will have a "cascading failure", that is, an avalanche effect [33][34], as shown in figure7.

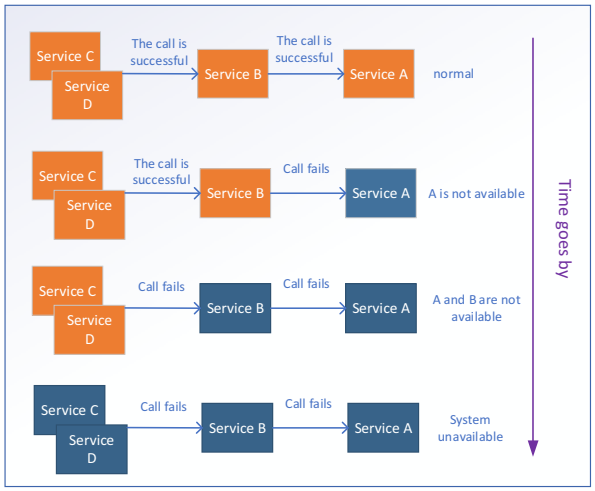

Fig. 7. Cascading effect.

The inherent performance challenges of distributed applications, complex system communication [34]. Traditional monolithic applications are called in memory, while distributed applications are calls that move between processes, possibly through the network, which brings additional latency and speed loss. Calling remote microservices in a loop increases the delay of each loop iteration and may quickly render the service call unavailable.

B. Debugging

Microservices systems are essentially concurrent distributed systems. In general, the effective way to debug concurrent distributed systems is to track and visualize the execution of the system [35]. Compared with traditional concurrent distributed systems, microservices systems are more complex and dynamic [36], as follows:

- A large number of microservices instances run on a large number of nodes (such as physical or virtual machines), and the distribution of microservices instance is also constantly changing, which brings great uncertainty to the communication between microservices [37].
- Microservices systems involve complex environmental configurations, and incorrect or inconsistent environmental configurations may cause runtime failures [38].
- A large number of complex asynchronous interactions are involved between microservices. These asynchronous interactions involve complex call chains, which can easily lead to improper collaboration, which in turn can lead to runtime failures [36].
- Because microservices instance can be created and destroyed dynamically, there is a lack of correspondence between microservices and system nodes in microservices systems [37].

These high complexity and dynamics have brought many challenges to debugging microservices systems [39][37]. Literature [36] pointed out that the current debugging of microservices systems depends largely on the developer's experience with the system and similar failure cases, and mainly relies on manual methods to check logs.

C. Data consistency

Although microservices systems bring many benefits in terms of flexibility and scalability, how to ensure the consistency of data in the system is challenging [38]. In a monolithic system, data is stored centrally in the same database; while in a microservices architecture, data is distributed in databases of different microservices, how to ensure transaction consistency across multiple databases is a critical issue that needs to be resolved [40][41].

Obviously, the traditional method of solving data consistency in a single database is not suitable for microservices systems [42]. In the context of microservices, there are currently two methods for solving data consistency:

- Cloning-based method. Each microservice creates a clone of the original database and interacts with its private database independently [43]. Each update in the private database will be notified to other cloned databases by broadcast to ensure the consistency of data between the databases. However, the disadvantage of this method is that it requires more resources to store data [40].
- Based on the private database method, each microservice has its private database [44]. For example, a document [45] proposes a slicing mode, which divides the unified central data storage into a set of horizontal slices according to the business boundary of each microservice. However, the main challenge faced by this method is how to ensure the consistency of the data when connecting the data in different databases [40].

Distributed applications cannot guarantee simultaneous transactions, and all distributed transactions must be called asynchronously [45][40]. Influenced by network delays, complicated communication modes, etc., may cause data to be inconsistent[46]. According to CAP theory, a choice must be made between usability and consistency. If you choose to provide consistency, you have to pay the price of blocking other concurrent accesses before consistency is met. This may continue for an indefinite period, especially in distributed systems that already have network delays, complex communications, and other features that can easily lead to lost connections[45]. Usability is generally a better choice, but maintaining data consistency between services and databases is a fundamental requirement [46]. Therefore, data consistency will be an important challenge for microservices.

V. CONCLUSION

The goal of this paper is to provide a comprehensive overview of the current status of widely used microservices technologies. Firstly, it describes the evolution of application architecture patterns from traditional monolithic architecture, SOA to microservices architecture. Secondly, the important technologies of microservices are explained from two aspects of container and data management. Finally, the three technical challenges faced by microservices are listed: performance, debugging, and data consistency. Besides, the needs of the organization should be considered when choosing a microservices architecture. There is usually no one development model that is the best. Instead, each design pattern performs better in different situations. The complex architecture is accompanied by a long development cycle and additional licensing fees for third-party applications. At the same time, hiring higher quality developers and testers in the team is another factor that increases the total cost. However, it should not be forgotten that these architectures can increase productivity and reduce costs because they have higher energy efficiency in the long run.

ACKNOWLEDGEMENTS

This work was supported in part by the Project of National Natural Science Foundation of China under Grant No. 61702442, 61862065, and 61662085, the Application Basic Research Project in Yunnan Province Grant No. 2018FB105, the Major Project of Science and Technology of Yunnan Province under Grant No. 202002AD080002 and No. 2019ZE005.

REFERENCES

[1] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," in IEEE Software, vol. 33, no. 3, pp. 42-52, May-June 2016, doi: 10.1109/MS.2016.64.

[2] X. Xu, X. Liu, Z. Xu, F. Dai, X. Zhang and L. Qi, "Trust-Oriented IoT Service Placement for Smart Cities in Edge Computing," in IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4084-4091, May 2020, doi: 10.1109/JIOT.2019.2959124.

[3] C. Richardson, "Microservices-Pattern: Microservice Architecture," March 2014. http://microservices.io/patterns/microservices.html.

[4] A. Krylovskiy, M. Jahn and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," 2015 3rd International Conference on Future Internet of Things and Cloud, Rome, 2015, pp. 25-30, doi: 10.1109/FiCloud.2015.55.

[5] H. Knoche and W. Hasselbring, "Using Microservices for Legacy Software Modernization," in IEEE Software, vol. 35, no. 3, pp. 44-49, May/June 2018, doi: 10.1109/MS.2018.2141035.

[6] F. Rademacher, J. Sorgalla and S. Sachweh, "Challenges of Domain-Driven Microservice Design: A Model-Driven Perspective," in IEEE Software, vol. 35, no. 3, pp. 36-43, May/June 2018, doi: 10.1109/MS.2018.2141028.

[7] F. Dai, Q. Mo, Z. Qiang, B. Huang, W. Kou and H. Yang, "A Choreography Analysis Approach for Microservice Composition in Cyber-Physical-Social Systems," in IEEE Access, vol. 8, pp. 53215-53222, 2020, doi: 10.1109/ACCESS.2020.2980891.

[8] Q. Mo, W. Song, F. Dai, L. Lin and T. Li, "Development of Collaborative Business Processes: A Correctness Enforcement Approach," in IEEE Transactions on Services Computing, doi: 10.1109/TSC.2019.2961346.

[9] X. Xu, H. Cao, Q. Geng, X. Liu, F. Dai, C. Wang, "Dynamic Resource Provisioning for Workflow Scheduling under Uncertainty in Edge Computing Environment," Concurrency and Computation-Practice & Experience, 2020, doi: 10.1002/cpe.5674.

[10] L. De Lauretis, "From Monolithic Architecture to Microservices Architecture," 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 2019, pp. 93-96, doi: 10.1109/ISSREW.2019.00050.

[11] C. Esposito, A. Castiglione and K. R. Choo, "Challenges in Delivering Software in the Cloud as Microservices," in IEEE Cloud Computing, vol. 3, no. 5, pp. 10-14, Sept.-Oct. 2016, doi: 10.1109/MCC.2016.105.

[12] V. Singh and S. K. Peddoju, "Container-based microservice architecture for cloud applications," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 847-852, doi: 10.1109/CCAA.2017.8229914.

[13] A. Koschel, I. Astrova and J. Dötterl, "Making the move to microservice architecture," 2017 International Conference on Information Society (i-Society), Dublin, 2017, pp. 74-79, doi: 10.23919/i-Society.2017.8354675.

[14] A. Sill, "The Design and Architecture of Microservices," in IEEE Cloud Computing, vol. 3, no. 5, pp. 76-80, Sept.-Oct. 2016, doi: 10.1109/MCC.2016.111.

[15] Z. Xiao, I. Wijegunaratne and X. Qiang, "Reflections on SOA and Microservices," 2016 4th International Conference on Enterprise Systems (ES), Melbourne, VIC, 2016, pp. 60-67, doi: 10.1109/ES.2016.14.

[16] A. Furda, C. Fidge, O. Zimmermann, W. Kelly and A. Barros, "Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency," in IEEE Software, vol. 35, no. 3, pp. 63-72, May/June 2018, doi: 10.1109/MS.2017.440134612.

[17] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, 2017, pp. 185-190, doi: 10.1109/ICIN.2017.7899408.

[18] C. Pahl, A. Brogi, J. Soldani and P. Jamshidi, "Cloud Container Technologies: A State-of-the-Art Review," in IEEE Transactions on Cloud Computing, vol. 7, no. 3, pp. 677-692, 1 July-Sept. 2019, doi: 10.1109/TCC.2017.2702586.

[19] C. Pahl, "Containerization and the PaaS Cloud," in IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, May-June 2015, doi: 10.1109/MCC.2015.51.

[20] M. G. Xavier, I. C. D. Oliveira, F. D. Rossi, R. D. D. Passos, K. J. Matteussi and C. A. F. D. Rose, "A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, 2015, pp. 253-260, doi: 10.1109/PDP.2015.67.

[21] W. Liangming, L. Fagui, Z. Hao and Z. Haiyan, "Trusted network connection of Inter-Domian on Xen virtual machine," 2011 International Conference on Electronics, Communications and Control (ICECC), Ningbo, 2011, pp. 4058-4061, doi: 10.1109/ICECC.2011.6066603.

[22] J. Wu and J. Li, "ERTDS: A dynamic CPU scheduler for Xen virtualization systems," 2017 International Conference on Applied System Innovation (ICASI), Sapporo, 2017, pp. 457-460, doi: 10.1109/ICASI.2017.7988453.

[23] R. Sailer et al., "Building a MAC-based security architecture for the Xen open-source hypervisor," 21st Annual Computer Security Applications Conference (ACSAC'05), Tucson, AZ, 2005, pp. 10 pp.-285, doi: 10.1109/CSAC.2005.13.

[24] I. Ahmad, J. M. Anderson, A. M. Holler, R. Kambo and V. Makhija, "An analysis of disk performance in VMware ESX server virtual machines," 2003 IEEE International Conference on Communications (Cat. No.03CH37441), Austin, TX, USA, 2003, pp. 65-76, doi: 10.1109/WWC.2003.1249058.

[25] D. T. Vojnak, B. S. Đorđević, V. V. Timčenko and S. M. Štrbac, "Performance Comparison of the type-2 hypervisor VirtualBox and VMWare Workstation," 2019 27th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2019, pp. 1-4, doi: 10.1109/TELFOR48224.2019.8971213.

[26] W. Bai and W. Li, "A Novel VSFTP-Based KVM Virtualization Cloud Deployment Scheme," 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Shanghai, 2018, pp. 211-217, doi: 10.1109/CSCloud/EdgeCom.2018.00045.

[27] S. Zhang, L. Wang and X. Han, "A KVM Virtual Machine Memory Forensics Method Based on VMCS," 2014 Tenth International Conference on Computational Intelligence and Security, Kunming, 2014, pp. 657-661, doi: 10.1109/CIS.2014.72.

[28] C. Guo, T. Li, Z. Gong and H. Han, "A virtual vulnerability validation platform based on KVM," 2015 IEEE 5th International Conference on Electronics Information and Emergency Communication, Beijing, 2015, pp. 228-231, doi: 10.1109/ICEIEC.2015.7284527.

[29] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange and C. A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, 2013, pp. 233-240, doi: 10.1109/PDP.2013.41.

[30] S. Prasad and S. B. Avinash, "Application of polyglot persistence to enhance performance of the energy data management systems," 2014 International Conference on Advances in Electronics Computers and Communications, Bangalore, 2014, pp. 1-6, doi: 10.1109/ICAECC.2014.7002444.

[31] S. Nadkarni, A. Kadakia and K. Shrivastava, "Providing Scalability to Data Layer Using a Novel Polyglot Persistence Approach," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697383.

[32] F. Dai, H. Chen, Z. Qiang, Z. Liang, B. Huang, L. "Wang, Automatic Analysis of Complex Interactions in Microservice Systems," Complexity, 2020, doi: 10.1155/2020/2128793.

[33] K. Takeda et al., "Franz-Keldysh and avalanche effects in a germanium waveguide photodiode," 10th International Conference on Group IV Photonics, Seoul, 2013, pp. 138-139, doi: 10.1109/Group4.2013.6644409.

[34] M. M. Hayat, B. E. A. Saleh and M. C. Teich, "Effect of dead space on gain and noise of double-carrier-multiplication avalanche photodiodes," in IEEE Transactions on Electron Devices, vol. 39, no. 3, pp. 546-552, March 1992, doi: 10.1109/16.123476.

[35] F. Dai, Q. Mo, T. Li, B. Huang, Y. Yang, Y. Zhao. "Refactoring Business Process Models with Process Fragments Substitution". Wireless Networks, 2020, doi: 10.1007/s11276-020-02367-3.

[36] X. Zhou et al., "Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2018.2887384.

[37] X. Zhou et al., "Delta Debugging Microservice Systems," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 2018, pp. 802-807, doi: 10.1145/3238147.3240730.

[38] L. Safina, M. Mazzara, F. Montesi and V. Rivera, "Data-Driven Workflows for Microservices: Genericity in Jolie," 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, 2016, pp. 430-437, doi: 10.1109/AINA.2016.95.

[39] C. Esposito, A. Castiglione and K. R. Choo, "Challenges in Delivering Software in the Cloud as Microservices," in IEEE Cloud Computing, vol. 3, no. 5, pp. 10-14, Sept.-Oct. 2016, doi: 10.1109/MCC.2016.105.

[40] M. E. Kholy and A. E. Fatatry, "Framework for Interaction Between Databases and Microservice Architecture," in IT Professional, vol. 21, no. 5, pp. 57-63, 1 Sept.-Oct. 2019, doi: 10.1109/MITP.2018.2889268.

[41] R. M. Munaf, J. Ahmed, F. Khakwani and T. Rana, "Microservices Architecture: Challenges and Proposed Conceptual Design," 2019 International Conference on Communication Technologies (ComTech), Rawalpindi, Pakistan, 2019, pp. 82-87, doi: 10.1109/COMTECH.2019.8737831.

[42] M. Villamizar et al. Cost comparison of running web applications in the cloud using monolothic, microservice, and AWs Lambada architecture [C]// Proceedings of the Service Oriented Computing and Applications, 2017, vol. 11, pp. 233- 247, doi: 10.1007/s11761-017-0208-y.

[43] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino and A. Di Salle, "MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems," 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, 2017, pp. 298-302, doi: 10.1109/ICSAW.2017.9.

[44] N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, 2016, pp. 44-51, doi: 10.1109/SOCA.2016.15.

[45] C. H. Costa, J. Filho, F. Oliveira. "Sharding by Hash partitioning. A database scalability pattern to achieve evenly sharded database clusters," Proceedings of the International Conference on Enterprise Information Systems, 2015, doi: 10.5220/0005376203130320.

[46] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan and W. Dou, "Dynamic Resource Provisioning With Fault Tolerance for Data-Intensive Meteorological Workflows in Cloud," in IEEE Transactions on Industrial Informatics, vol. 16, no. 9, pp. 6172-6181, Sept. 2020, doi: 10.1109/TII.2019.2959258.