Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 18 No. 10 (2024) | OPEN ACCESS

https://doi.org/10.3991/ijim.v18i10.45929

PAPER

Containerized Microservices for Mobile Applications Deployed on Cloud Systems

Jordan Jordanov, Dimitrios Simeonidis, Pavel Petrov(✉)

University of Economics – Varna, Varna, Bulgaria

petrov@ue-varna.bg

ABSTRACT

This study explores the transformative role of containerized microservices in the sphere of mobile application development, especially within public cloud ecosystems. It focuses on how technologies such as Docker and Kubernetes contribute to improving deployment, scalability, and overall management of mobile applications, with an emphasis on containerizing back-end services. We analyze their efficiency in streamlining deployment processes, focusing on how they improve the application's performance and reliability. Additionally, we examine various alternative deployment strategies, such as blue-green, rolling, and canary releases, to emphasize their effectiveness in minimizing risks and facilitating smooth transitions in dynamic cloud environments. The study takes a comprehensive approach to achieve this goal, which includes a systematic review of existing literature, a thorough examination of relevant use cases, and an assessment of open-source technologies. Our findings reveal not only the practical benefits of these strategies but also their strategic application, offering important insights for software engineers and decision-makers. This study emphasizes the significance of integrating and optimizing containerized microservices in mobile app development to achieve more efficient, scalable, and manageable application lifecycles on cloud-based platforms.

KEYWORDS

microservices, mobile application, public cloud, containerization, virtualization

1 INTRODUCTION

In the context of a more digitized environment, contemporary deployment tactics have undergone significant transformations, facilitated by technologies such as continuous integration (CI), continuous delivery (CD) [1–3], Docker, and Kubernetes. These technologies have the dual benefit of streamlining the deployment process and reducing the likelihood of integration difficulties through automated build and test methods. CI procedures facilitate the frequent merging of changes made by developers back into the main branches, thereby accelerating

Jordanov, J., Simeonidis, D., Petrov, P. (2024). Containerized Microservices for Mobile Applications Deployed on Cloud Systems. *International Journal of Interactive Mobile Technologies (iJIM)*, 18(10), pp. 48–58. https://doi.org/10.3991/ijim.v18i10.45929

Article submitted 2023-10-16. Revision uploaded 2024-01-12. Final acceptance 2024-03-01.

the development cycle. In addition, the robust characteristics of cloud resources, enhanced by worldwide data centers, provide exceptional benefits in terms of scalability and security.

Nevertheless, the academic discourse lacks comprehensive knowledge of the concerns related to the development cycle, scalability, and post-deployment maintenance of mobile apps, which is a significant gap that this study aims to address. There is a lack of academic research specifically examining the impact of containerized microservices on the performance, manageability, and scalability of mobile apps deployed on cloud platforms.

Considering the significant potential for transformation offered by cloud computing and containerization technologies such as Docker and Kubernetes, conducting comprehensive research is crucial and timely. This assertion holds special validity as enterprises increasingly shift their focus towards mobile technology, highlighting the importance of efficiently implementing and overseeing mobile apps as a critical aspect of company operations.

To address this gap, the main objective of this study is to evaluate the utilization of containerized microservices in public cloud systems, specifically in the context of mobile application development, with a particular focus on backend containerization. Our objective is to examine various deployment strategies and monitoring tools and assess their overall impact on the efficiency and effectiveness of mobile application development processes. By exploring the relationship between these technologies and mobile application development, we aim to gain a thorough understanding of how containerization can enhance the development lifecycle in cloud-based environments.

2 LITERATURE REVIEW

Cloud computing has emerged as a disruptive technological paradigm, significantly reshaping company operations and service delivery. As defined by the National Institute of Standards and Technology (NIST), cloud computing is characterized by five fundamental attributes, including on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. They form the foundation of cloud computing's operational capabilities [4]. These collective characteristics enable organizational adaptability, cost effectiveness, and scalability. The cloud may be accessible via various deployment modes, including public clouds managed and operated by external service providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. These clouds allow customers to access computational resources via the Internet, while the cloud providers retain exclusive ownership and management of all hardware, software, and supporting infrastructure. Access is primarily achieved using conventional web browsers, making it a highly advantageous option for a wide range of tasks, including temporary or exploratory ones.

The academic discussion regarding cloud computing has mainly focused on its service models, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [5], each with distinct advantages and drawbacks. Nevertheless, a noticeable deficiency exists in the existing body of scholarly work regarding the use of cloud computing for deploying mobile applications. The need for further investigation arises from the potential transformative impact of combining the fundamental attributes of cloud computing with the features

of public clouds to revolutionize the administration and deployment of mobile applications.

The microservice architecture has become a significant paradigm, especially for cloud-native applications. The architectural approach described involves decomposing an application into discrete, autonomous services [6], designed to encapsulate specific business functionalities that interact with each other through well-defined application programming interfaces (APIs). The architectural concept of technology agnosticism enables teams to flexibly select the most appropriate technology stack for each microservice.

Business teams often emphasize the urgent need for promptly incorporating novel features, creating a conducive atmosphere that fosters creativity and swift experimentation, and having the flexibility to deploy changes instantly without being constrained by predetermined deployment timelines. Attributes of microservices, such as increased adaptability, innovation, and rapid feature implementation, offer significant advantages in enhancing the performance and utility of mobile applications. The interaction between mobile apps as front-end and cloud-based microservice systems as backend often emphasizes the importance of data storage, synchronization, and security.

Nevertheless, its architectural design has specific intricacies that require scientific investigation. While the initial administration of individual services may seem simple, the complexity of the entire system increases, particularly with deployments and interactions between services. The complexity of network latency and fault management increases with service expansion. The dispersed nature of the architecture introduces additional complexities to operations such as testing and debugging. The difficulty of data integrity arises because each service maintains a data repository, necessitating methods such as eventual consistency.

Numerous potential remedies have been suggested to address these issues. Containerization and orchestrators are essential components for enabling flexible deployment and scaleability in diverse computing environments. Several deployment techniques (e.g., blue-green, rolling, and canary deployment) are designed to achieve a harmonious balance between rapid feature delivery and service reliability. Docker and Kubernetes have been explored as methods to facilitate smooth service operations. Furthermore, DevOps methodologies and the utilization of CI/CD pipelines have been widely promoted to automate and optimize deployment procedures [7].

The lack of scholarly research in the academic discussion regarding the use of containerized microservices for deploying mobile applications on cloud systems has inspired the current study. This study aims to emphasize the impact of containerization technologies on the advancement and acceptance of mobile apps in cloud environments.

3 METHODOLOGY AND DATA COLLECTION

This section aims to explain the methodological framework, data sources, and analytical processes used in the current study to enhance transparency and facilitate the replication of the research. The theoretical foundations of the study are based on a comprehensive analysis of the existing literature, including essential ideas and models related to containerization, microservices, and mobile and cloud

computing technologies. The study utilizes many benchmark programs, primarily those developed by Microsoft [8], to provide an evaluative framework. These programs demonstrate cross-platform compatibility and are built using a containerized and microservices-oriented design that is compatible with Linux, Windows, Docker containers, and cloud-based Kubernetes services.

The primary data sources for this inquiry are peer-reviewed academic papers, case studies, and open-source code repositories. Moreover, precise technical guidelines, characteristics, and limitations are derived from relevant official documents to establish the technical boundaries of the research. The data collection procedure rigorously adheres to ethical standards.

4 TECHNOLOGIES FOR FRONTEND DEVELOPMENT

Mobile app development has primarily utilized three types of platform technologies: native, hybrid, and cross-platform. Each represents a unique combination of development languages, integrated development environments (IDEs), level of access to smartphone features, and user experience. Native technologies, such as Swift for iOS and Java or Kotlin for Android, offer superior performance and provide extensive access to device capabilities. They utilize dedicated IDEs such as Xcode and Android Studio. However, they require parallel development and support for each platform. In contrast, hybrid technologies such as Apache Cordova (utilizing HTML, CSS, and JavaScript) provide a write-once, run-anywhere approach with a moderate level of device accessibility. However, they can compromise the app's performance and feel less "native." Finally, cross-platform technologies such as React Native (JavaScript and JSX) and Flutter (Dart) attempt to bridge this gap. Utilizing their own IDEs, such as Visual Studio Code or IntelliJ IDEA, these technologies enable a unified codebase that compiles native code, offering a more authentic user experience and extensive access to device features. However, cross-platform technologies may still lag behind native technologies in accessing the latest platform-specific features or handling complex graphical interfaces. Hence, carefully considering the advantages and disadvantages of each platform based on the project's specific requirements is critical to making an informed choice [9].

The emergence of cloud-based platforms, such as Expo and GitHub Codespaces, has significantly contributed to the advancement of mobile application development, achieving unparalleled levels of quality and performance. The controlled environment provided by Expo enhances the speed of development cycles through features such as real-time code compilation, a collection of pre-configured native modules, and the availability of Web, Android, and iOS emulators on demand. Consequently, the need for complex local configurations is eliminated. In a similar vein, GitHub Codespaces offers a comprehensive cloud-based IDE that facilitates real-time coding, debugging, and testing. These technologies, along with several others, bring about a significant change in the realm of mobile development by providing production-like environments at any stage of the programming process.

The effectiveness and efficiency of a mobile application are closely tied to the strength and reliability of its backend communication infrastructure. The backend system serves as a central component for data administration, computational operations execution, and client communication. Figure 1 illustrates the current trend towards cloud-centric designs, which serves as a prime example of this interconnection [9].

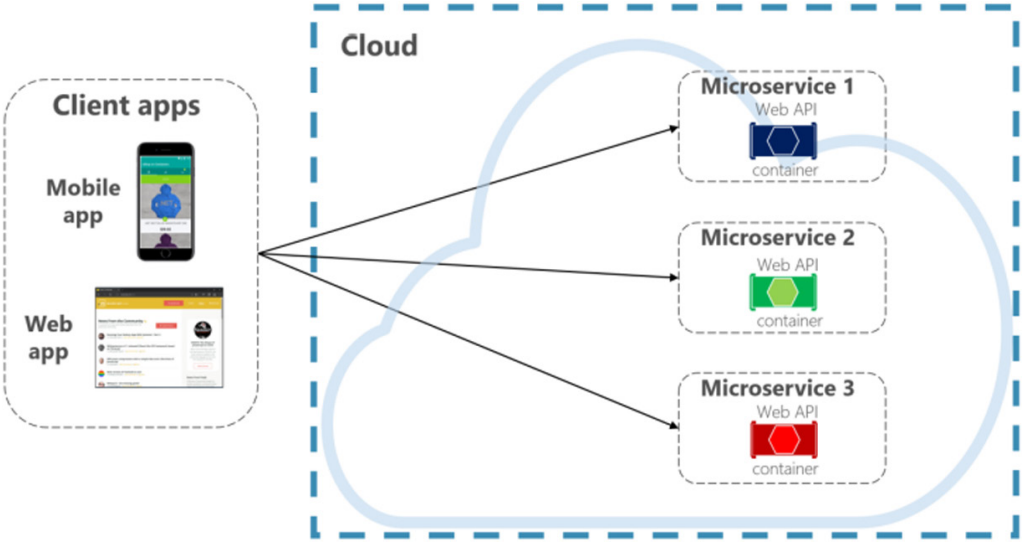

Fig. 1. Structural composition of a cloud-centric ecosystem that utilizes web and mobile application with the microservices [9]

The system architecture being discussed emphasizes client apps, which serve as the primary way users interact with digital services. The system's highest level of performance and the resulting satisfaction of the users depend on effective interactions with the backend cloud infrastructure. The design paradigm places significant emphasis on the importance of a robust backend, a concept that will be further discussed in the following sections.

5 TECHNOLOGIES FOR BACKEND CONTAINERIZATION

Containerization is an approach in software development where an application's code, dependencies, and configurations are packaged into a binary file called an image. Images are read-only "templates" stored in a registry that serves as a repository or library for images. The image is transformed into a running container instance that can be started, stopped, moved, and deleted. Containers are created for different parts of the application, such as the web service, database, and caching. Software containerization enables developers and IT professionals to automatically propagate changes across environments. Containers also isolate applications from each other within a shared operating system. Applications run on the container host. From an application perspective, instantiating an image means creating a container. Another benefit of containerization is scalability. Scaling happens quickly; new containers are created for short-term tasks. Containers offer the benefits of isolation, portability, flexibility, and control over the entire application lifecycle. Major public cloud service providers that can be utilized include Azure, Google Cloud, and Amazon Web Services.

Docker is the most widely used and established technology [10–11]. It is an open-source project that automates the deployment of applications as portable, self-contained containers that can run on-premises or in the cloud. Docker promotes and develops this technology. Its containers can run on Linux or Windows. Advantages for developers include accelerated onboarding of new programmers to the project, elimination of application conflicts, and streamlined software updating and migration processes. Figure 2 presents a comparison between a virtual machine and a Docker container [12].

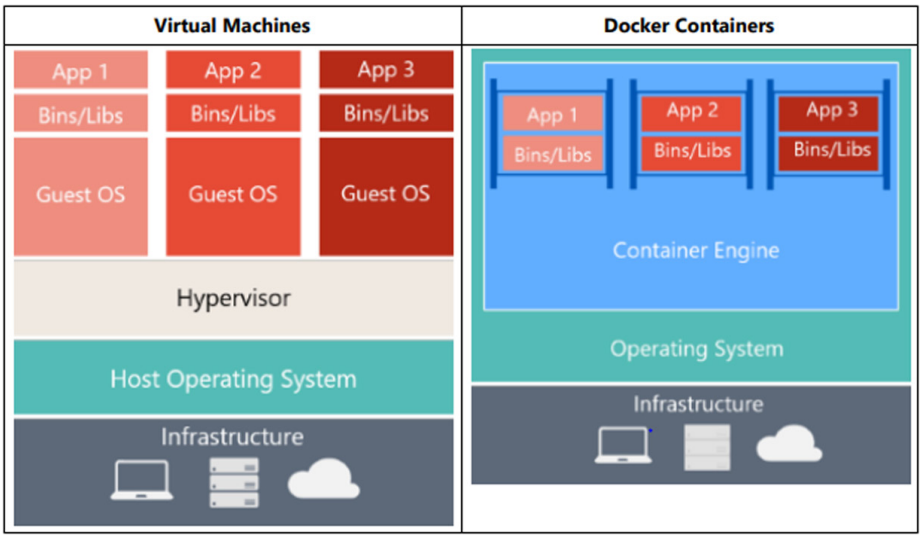

Fig. 2. Comparison between virtual machines and Docker containers [12]

Virtual machines include the application, necessary libraries, and a full operating system. In comparison, full virtualization requires one resource and a longer startup time.

Docker containers include the application and all of its dependencies. However, they share the OS kernel with other containers running as isolated processes in the user space of the host operating system. Except for Hyper-V containers, where each container runs inside a dedicated virtual machine. Even though Docker simplifies the application packaging process, a solution like Kubernetes is necessary to manage these containers, especially at scale. Kubernetes automates deploying and scheduling application containers in a cluster, provides self-healing capabilities (e.g., automatic container restart, rescheduling, and replication), and facilitates horizontal scalability [13].

Kubernetes provides high-level operations that can be executed through the microservices' code. Robots with instructions are transferred to the cloud machines, known as a "cluster," which is a collection of Linux or Windows virtual machines (node points) where the applications are deployed (but not directly). Kubernetes manages the routing and logistics of microservices (most commonly used in this architecture).

Docker and Kubernetes facilitate deployment strategies such as blue-green, rolling, and canary release deployments.

Blue-green deployment: This strategy involves two identical production environments, blue and green [14]. Only one of these environments is active at any time. Suppose the "blue" environment is active and handling traffic. If a new version of the application needs to be deployed, a "green" environment is then set up. Thus, tests can be extensively conducted in this separate environment. Once stability and performance are satisfactory, the router switches to a "green" environment, which becomes active. The "blue" environment remains inactive until the next release, enabling quich rollback if necessary.

Rolling deployment: In a rolling deployment, a new version of the application is gradually deployed to several instances at a time, instead of all at once, while the remaining instances still retain the old version [15]. This approach allows for careful deployment and maintains service availability during the deployment process. If problems occur, the deployment process can be halted, affecting only a subset of instances.

Canary release deployment: Named after the practice of sending a canary into a mine to check for hazardous gases, canary deployment involves introducing a change to a small subset of users before applying it to the entire infrastructure. The goal is to test the new release on a small portion of traffic, ensuring it works as expected, before rolling it out to a wider user base [16]. If something goes wrong, only the canary instances are affected, and we can roll back the changes without affecting all users.

The canary release deployment model may be well-suited for a cloud-based order management system. This strategy involves gradually implementing changes to a subset of users before rolling them out to the entire system. By segmenting the deployment in this way, it is possible to monitor the impact of system changes in real time, thereby reducing the risk of widespread disruption. It provides a comprehensive testing environment for new features or modifications to the order management system, enterprise resource planning, fleet management, and monitoring systems. This enables the team to identify potential issues before they impact all end users. Production test patterns are strategies used in software development to ensure that the software functions as expected in a production environment. These models can help prevent software defects, improve system resilience, and maintain quality and reliability. A/B testing is one such model that enables data-driven decision-making in the context of a cloud-based order management system by simultaneously deploying different versions of system improvements or new features to subsets of users, thereby allowing comparative performance evaluations [17–22].

All these strategies offer various ways to reduce risk and minimize downtime during deployment. The choice of strategy should be based on the specific needs and circumstances of the project.

Regarding deployment strategies, Docker can be useful in blue-green deployments. A new container can be set up with the updated version of the application, and traffic can be directed to it when it is ready. Docker also enables the easy creation and management of separate instances required for rolling and canary deployments. Kubernetes can help by managing two different sets of pods (blue and green). Service objects can be used to route traffic between the two environments. Kubernetes initially supports this strategy through rolling deployment, updating an implementation by gradually replacing old modules with new ones. This function ensures that a minimum number of packages are always available during the update, and at most, a certain number of packages are created above the desired amount. Kubernetes can gradually transition traffic to the updated version of the application and monitor performance. If the updated version works well, traffic redirection can continue until the new version can handle all requests. However, if something goes wrong, the traffic can be redirected back to the older, stable version.

6 AFTER-DEPLOYMENT MANAGEMENT OF CLOUD MICROSERVICES

Installing a service in the cloud is not the final step of the deployment. Therefore, effective system logging and monitoring are essential management procedures. Understanding the complex responsibilities they have and the wide range of tools available is essential to guaranteeing optimal system functionality. Infrastructure monitoring and application monitoring are the two main categories. Infrastructure monitoring involves estimating and controlling system resources such as CPU, memory, disk space, and network traffic. Due to their comprehensive resource monitoring capabilities and capacity to identify constraints, tools like Nagios are well-suited for this purpose. In contrast, application monitoring focuses on the

functionality and efficiency of the application within the system. It addresses aspects such as response time, error rate, and transaction tracking critical to a seamless user experience in order management systems [23–24].

Maintaining a system log supplements monitoring. It helps developers track bugs and understand the sequence of events that led to system failure. For example, the ELK (Elasticsearch, Logstash, and Kibana) stack is an open-source logging system that collects logs from various sources [25–27], stores them for quick retrieval (Elasticsearch), processes and transforms them (Logstash), and then visualizes them in a user-friendly manner (Kibana). This utilization of resources lays the groundwork for a strong, dependable, and efficient procurement management system. The ELK enables quick identification of system errors and proactive problem-solving, thus facilitating the delivery of a quality product to the end user.

Monitoring is crucial for application administration and management, particularly in the context of APIs. The purpose of monitoring serves the following objectives:

- Anticipation and preventive measures involve identifying potential problems before they escalate.
- Identifying problems as soon as they become apparent.
- Providing operational oversight to gain a thorough understanding of API efficiency and usage patterns.
- Significance of the observation.
- Customer expect APIs to deliver consistent performance and availability in today's digital age.
- Functionality protection offers dependable monitoring mechanisms to ensure that the API operates optimally and meets its service level objectives.

Some important metrics to track in an API ecosystem include the following:

- Requests per second provide insight into the current API traffic and demand.
- Monitoring the number of crashes can help identify recurring issues or vulnerabilities early.
- Latency estimates the response time of an API, providing insight into its efficiency and performance.
- Monitoring the number of active users can provide insight into the demand and popularity of the system.
- Session count provides an overview of user interaction and engagement with the API.
- Understanding the geographic distribution of users can help optimize server location and enhance the user experience.
- Monitoring CPU usage can reveal potential limitations or areas that require optimization.
- Regular monitoring of memory usage ensures that the system is not overloaded and operates efficiently.

7 LIMITATIONS AND FUTURE WORK

This study is limited by its primary focus on public cloud systems, potentially overlooking the specifics of private or hybrid cloud deployments. The rapid evolution of cloud technologies and microservices necessitates an ongoing reassessment of our findings to ensure their relevance.

In discussing front-end development technologies, it is crucial to consider security challenges. These include vulnerabilities in code libraries, risks associated with data storage and transmission, and the need for robust authentication mechanisms. The integration of front-end technologies with back-end containerization demands attention to security at both ends, ensuring a comprehensive approach to safeguarding the application.

Future research should explore the application of containerized microservices across various cloud models, including private and hybrid, to gain a more comprehensive view of deployment challenges. Investigating emerging technologies such as serverless computing and AI-driven tools within microservice architectures could provide valuable insights.

8 CONCLUSION

This paper explores the dynamics of mobile application development technologies, backend containerization, and post-deployment management. It emphasizes the importance of making informed technology choices and implementing effective monitoring and logging practices for successful software development projects.

Mobile application development encompasses various frontend platform technologies, including native, hybrid, and cross-platform approaches, each offering distinct advantages and limitations. The choice of technology depends on project requirements, and the mobile application's backend is an important part. Backend containerization, a pivotal aspect of modern software development, involves packaging an application's code, dependencies, and configurations into images that can be deployed as containers. Docker, a widely adopted open-source platform, simplifies application deployment and offers benefits such as scalability, isolation, portability, and flexibility. Container orchestration tools such as Kubernetes automate container management, enabling scalability and self-healing.

Effective post-deployment management of cloud microservices includes robust monitoring and logging. Infrastructure monitoring oversees system resources, while application monitoring evaluates functionality and efficiency. Tools such as Nagios excel at resource monitoring, while application monitoring tools focus on response time, error rates, and transaction tracking. Finally, a comprehensive logging system, such as the ELK stack, enables quick error identification and proactive issue resolution, thereby enhancing the overall product quality.

9 ACKNOWLEDGMENT

This study has been financed by NPD-331/2023 from the University of Economics-Varna Science Fund.

10 REFERENCES

[1] E. Soares, G. Sizílio, J. Santos, D. Alencar da Costa, and U. Kulesza, "The effects of continuous integration on software development: A systematic literature review," *Empirical Software Engineering*, vol. 27, no. 3, 2022. https://doi.org/10.1007/s10664-021-10114-1

[2] J. P. Lima and S. R. Vergilio, "Test case prioritization in continuous integration environments: A systematic mapping study," *Information & Software Technology*, vol. 121, p. 106268, 2020. https://doi.org/10.1016/j.infsof.2020.106268

[3] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey, "Uncovering the benefits and challenges of continuous integration practices," in *IEEE Transactions on Software Engineering*, 2022, vol. 48, no. 7, pp. 2570–2583. https://doi.org/10.1109/TSE.2021.3064953

[4] NIST, "The NIST definition of cloud computing," National Institute of Standards and Technology, U.S. Department of Commerce, no. 800–145. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[5] S. Stuckenberg and S. Beiermeister, "Software-As-A-Service development: Driving forces of process change," in *PACIS 2012 Proceedings*, 2012, vol. 122. https://aisel.aisnet.org/pacis2012/122

[6] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, and M. A. Babar, "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Information & Software Technology*, vol. 131, p. 106449, 2021. https://doi.org/10.1016/j.infsof.2020.106449

[7] C. De La Torre, "Containerized docker application lifecycle with Microsoft platform and tools," *Microsoft Corp.*, 2022. https://learn.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/

[8] R. Vettor and S. Smith, "Architecting cloud native .NET applications for Azure," *Microsoft Learn*, 2023. https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/

[9] C. De La Torre, B. Wagner, and M. Rousos, ".NET microservices. Architecture for containerized .NET applications," *Microsoft Learn*, 2023. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

[10] A. M. Potdar, D. G. Narayan, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020. https://doi.org/10.1016/j.procs.2020.04.152

[11] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in docker development: A large-scale study using stack Overflow," *arXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2008.04467.

[12] C. De La Torre, B. Wagner, and M. Rousos, ".NET Microservices: Architecture for Containerized .NET applications," *Microsoft.com*, 2022. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined

[13] T. Menouer, "KCSS: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, pp. 4267–4293, 2020. https://doi.org/10.1007/s11227-020-03427-3

[14] B. Yang, A. Sailer, and A. Mohindra, "Survey and evaluation of Blue-Green deployment techniques in cloud native environments," in *Lecture Notes in Computer Science*, 2020, vol. 12019, pp. 69–81. https://doi.org/10.1007/978-3-030-45989-5_6

[15] C. K. Rudrabhatla, "Comparison of zero downtime based deployment techniques in public cloud infrastructure," in *Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, 2020, pp. 1082–1086. https://doi.org/10.1109/I-SMAC49090.2020.9243605

[16] N. C. Mendonca, P. Jamshidi, D. Garlan, and C. Pahl, "Developing self-adaptive microservice systems: Challenges and directions," in *IEEE Software*, 2021, vol. 38, no. 2, pp. 70–79. https://doi.org/10.1109/MS.2019.2955937

[17] R. Kohavi, D. Tang, and Y. Xu, *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge: Cambridge University Press, 2020. https://doi.org/10.1017/9781108653985

[18] P. Petrov, S. Ivanov, P. Dimitrov, G. Dimitrov, and O. Bychkov, "Projects management in technology start-ups for mobile software development," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 15, no. 7, pp. 194–201, 2021. https://doi.org/10.3991/ijim.v15i07.19291

[19] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. C. Gall, "We're doing it live: A multi-method empirical study on continuous experimentation," *Information & Software Technology*, vol. 99, pp. 41–57, 2018. https://doi.org/10.1016/j.infsof.2018.02.010

[20] L. I. Todoranova, R. V. Nacheva, V. S. Sulov, and B. P. Penchev, "A model for mobile learning integration in higher education based on students' expectations," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 11, pp. 171–182, 2020. https://doi.org/10.3991/ijim.v14i11.13711

[21] T. F. Düllmann, C. Paule, and A. Van Hoorn, "Exploiting devops practices for dependable and secure continuous delivery pipelines," in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering (RCoSE '18)*, 2018, pp. 27–30. https://doi.org/10.1145/3194760.3194763

[22] R. V. Nacheva, "Standardization issues of mobile usability," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 7, pp. 149–157, 2020. https://doi.org/10.3991/ijim.v14i07.12129

[23] Y. Aleksandrova and M. Armianova, "Evaluation of cost-sensitive machine learning methods for default credit prediction," in *2022 International Conference Automatics and Informatics (ICAI)*, 2022, pp. 89–94. https://doi.org/10.1109/ICAI55857.2022.9960023

[24] J. Vasilev, R. Nikolaev, and T. Milkova, "Transport task models with variable supplier availabilities," *Logistics*, vol. 7, no. 3, p. 45, 2023. https://doi.org/10.3390/logistics7030045

[25] P. Petrov, I. Kuyumdzhiev, R. Malkawi, G. Dimitrov, and O. Bychkov, "Database administration practical aspects in providing digitalization of educational services," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 17, no. 20, pp. 274–282, 2022. https://doi.org/10.3991/ijet.v17i20.32785

[26] B. P. Rao and N. N. Rao, "HDFS logfile analysis using ElasticSearch, LogStash and Kibana," in *Integrated Intelligent Computing, Communication and Security*, 2019, vol. 771, pp. 185–191. https://doi.org/10.1007/978-981-10-8797-4_20

[27] F. Ahmed, U. Jahangir, H. Rahim, K. Ali, and D.-E.-S. Agha, "Centralized log management using ElasticSearch, Logstash and Kibana," in *2020 International Conference on Information Science and Communication Technology (ICISCT)*, Karachi, Pakistan, 2020, pp. 1–7. https://doi.org/10.1109/ICISCT49550.2020.9080053

11 AUTHORS

Jordan Jordanov is an Assistant Professor and a doctoral candidate in the Department of Informatics at the University of Economics – Varna, Bulgaria. His research interests include modern cloud services and mobile technologies (E-mail: jordanov.jordan@ue-varna.bg).

Dimitrios Simeonidis is a PhD candidate in Computer Science at the University of Economics – Varna, Bulgaria. He is currently working as an IT Engineer in Thessaloniki, Greece (E-mail: simeonidis@ue-varna.bg).

Pavel Petrov is a Professor in the Department of Informatics at the University of Economics – Varna, Bulgaria. His research interests include distributed web systems, big data, and mobile computing (E-mail: petrov@ue-varna.bg).