Containerized Microservices for Mobile Applications Deployed on Cloud Systems

<https://doi.org/10.3991/ijxx.vx.ix.xxxx>

1st Author1(), 2nd Author1,2, 3rd Author2

1 International Association of Online Engineering, Vienna, Austria

2 Carinthia University of Applied Sciences, Villach, Austria

[example@example.org](mailto:example@example.org)

**Abstract—** The present study centers on a comprehensive examination of contemporary mobile application development paradigms, specifically emphasizing the containerization of backend services and administrative procedures inside a public cloud-based environment. Considering the increasing prevalence of cloud computing in the software business, this research aims to comprehensively analyze the benefits, constraints, and fundamental factors inherent in these emerging technologies and processes. The study utilizes a comprehensive approach to accomplish this goal, including a systematic analysis of existing literature, thoroughly examining pertinent use cases, and assessing open-source technologies. The results support the proposition that utilizing containerized microservices significantly enhances the capacity to effectively expand, oversee, and administer mobile apps hosted on cloud platforms. The findings possess significant importance within the software engineering domain, providing helpful advice on selecting optimum architectural choices and deployment methodologies.

**Keywords—**microservices, mobile application, public cloud, containerization

1. Introduction

In the context of a more digitized environment, contemporary deployment tactics have undergone significant transformations, facilitated by technologies like as continuous integration (CI), continuous delivery (CD) [1-3], Docker, and Kubernetes. These technologies have the dual benefit of streamlining the deployment process and greatly decreasing the likelihood of integration difficulties via automated build and test methods. CI procedures facilitate the frequent merging of changes made by developers back into the main branches, hence accelerating the development cycle. In addition, the robust characteristics of cloud resources, further enhanced by worldwide data centers, provide exceptional benefits in scalability and security.

Nevertheless, the academic discourse lacks full knowledge of the concerns regarding the development cycle, scalability, and post-deployment maintenance of mobile apps, a substantial gap that this study seeks 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 opportune. This assertion has special validity as enterprises progressively shift their focus towards mobile technology, elevating the significance of effectively implementing and overseeing mobile apps as a matter of utmost importance for company operations.

To fill this research gap, the primary objective of this study is to examine the latest technologies in mobile application development, with a specific emphasis on backend containerization. Thus, this study aims to analyze the merits and drawbacks of several post-deployment management strategies in cloud-based systems. Furthermore, the article intends to assess the impact of containerization technologies on the scalability, operational efficiency, and overall manageability of mobile applications deployed on cloud infrastructures.

1. 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 many deployment modes, including the public clouds managed and operated by external service providers, like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. These clouds enable customers to access computational resources through the Internet, while the cloud providers exclusively own and manage all hardware, software, and supporting infrastructure. Access is primarily attained using conventional web browsers, rendering it a very advantageous option for a diverse range of tasks, including temporary or exploratory ones.

The academic discussion pertaining to cloud computing has mostly centered on its service models, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [5], each possessing distinct advantages and drawbacks. Nevertheless, a noticeable deficiency exists in the existing body of scholarly work regarding using cloud computing 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 architectural style has become a significant paradigm, particularly cloud-native applications. The architectural approach described involves decomposing an application into discrete, autonomous services [6], designed to encapsulate certain business functionalities that interact with one another through well-defined application programming interfaces (APIs). The architectural idea of technology agnosticism allows teams to flexibly choose the most suitable technology stack for each microservice.

Business teams often emphasize the urgent need for the prompt incorporation of novel features, a conducive atmosphere fostering creativity and swift experimentation, and the flexibility to deploy changes instantly without being constrained by predetermined deployment timelines. Attributes of microservices, including heightened adaptability, ingenuity, and expeditious implementation of features, provide significant benefits in augmenting the performance and usefulness of mobile applications. The interaction between mobile apps as front-end and cloud-based microservice systems as backend often highlights the significance 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 simpler, the complexity of the total system increases, especially with deployments and interactions between services. The complexity of network latency and fault management increases with the expansion of services. The dispersed nature of the architecture introduces additional complexities to operations like testing and debugging. The difficulty of data integrity arises since each service maintains a data repository, requiring methods such as eventual consistency.

Numerous potential remedies have been suggested to tackle these issues. Containerization and orchestrators are fundamental components for facilitating flexible deployment and scalability in various computing environments. Several deployment techniques, (e.g., blue-green, rolling, and canary deployment) are designed to achieve a harmonious equilibrium between expeditious feature delivery and the reliability of services. Docker and Kubernetes have been examined as possible techniques for enabling seamless service operations. Furthermore, DevOps methods and the utilization of (CI/CD) pipelines have been widely promoted to automate and optimize deployment procedures [7].

The dearth of scholarly investigation in the academic discourse surrounding the utilization of containerized microservices for deploying mobile applications on cloud systems has prompted the current study, which seeks to highlight the influence of containerization technologies on the progress and adoption of mobile apps in cloud environments.

1. Methodology & Data Collection

This section seeks to clearly explain of the methodological framework, data sources, and analytical processes used in the current study to promote transparency and the capacity to replicate the research. The theoretical foundations of the research 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 exhibit cross-platform compatibility and are constructed using a containerized and microservices-oriented design 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 paperwork to establish the technical boundaries of the research. The data collection procedure rigorously adheres to ethical issues.

1. 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 a high level of access to device capabilities, as they use 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 (HTML, CSS, and JavaScript) offer a write-once, run-anywhere model with a moderate level of device access. 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. Using their own IDEs, such as Visual Studio Code or IntelliJ IDEA, these technologies facilitate a single codebase that compiles native code, providing a more native user experience and significant access to device features. However, cross-platform technologies may still be behind native technologies in accessing the latest platform-specific features or dealing with highly 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, reaching unparalleled levels of quality and performance. The controlled environment provided by Expo greatly enhances the speed of development cycles via 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 offering 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 the administration of data, execution of computational operations, and communication with the client. Figure 1 illustrates the current trend towards cloud-centric designs, which serves as a prime example of this interconnection [9].



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

The system architecture being discussed places emphasis on the client apps, which serve as the primary means through which users interact with digital services. The achievement of 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 need of a strong backend, a concept that will be further discussed in the following sections.

1. 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 acts 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 the different parts of the application, such as: the web service, database, and caching. Software containerization enables developers and IT professionals to automatically renew changes across environments. Containers also isolate applications from each other in 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. Big public cloud service providers that can be used include Azure, Google Cloud and AWS.

Docker is the most used and established technology [10-11], an open-source project that automates deploying 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 are the accelerated introduction of new programmers to the project, eliminating application conflicts and updating and migrating software. Fig. 2 presents a comparison between a virtual machine and a Docker container [12].

Graphical user interface

Description automatically generated

1. Comparation between virtual machines and Docker containers [12]

Virtual machines include the application, required libraries, and a complete operating system. In comparison, full virtualization requires more resources and more startup time in comparison.

Docker containers include the application and all 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 needed 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 to be performed through the microservices’ code. Robots with instructions are transferred to the cloud machines, the so-called "cluster": a set of Linux or Windows virtual machines (node points) on which the applications themselves are deployed (but not directly). Kubernetes handles the routing and logistics of microservices (most used in this architecture).

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

**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 alive and serving traffic. If it must deploy a new version of the application, a “green” environment is then deployed. Thus, tests can be extensively conducted in this separate environment. Once the stability and performance are satisfactory, the router switches to a “green” environment, which becomes active. The “blue” environment remains inactive until the next release, allowing it to quickly roll back if needed.

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

**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 well suit a cloud-based order management system. This strategy involves progressively applying changes to a subset of users before applying them to the entire system. By segmenting the deployment in this way, monitoring the impact of system changes in real time is possible, thereby reducing the risk of widespread disruption. It offers a complete test environment for new features or modifications to the order management system, enterprise resource planning, fleet management, and monitoring systems, allowing the team to identify potential problems before they affect all end users. Production test patterns are strategies used in software development to ensure 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 provide diverse ways to reduce risk and minimize downtime during deployment and can be chosen based on the specific needs and circumstances of the project.

Regarding deployment strategies, Docker can be useful in blue-green deployments, as a new container can be set up with the new version of the application, and traffic can be directed to it when it is ready. Docker also allows easy creation and management of the required separate instances for rolling and canary deployments. Kubernetes can help by managing two different sets of pods (blue and green). Service objects can be used to switch 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 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 new version of the application and monitor performance. If the new version works well, redirecting the traffic can continue until the new version handles all requests. However, if something goes wrong, the traffic can be redirected back to the older, stable version.

1. After deployment management of cloud microservices

Installing a service to a cloud is not the final step of the deployment and effective system logging and monitoring are essential management procedures. Understanding their complex duties and the vast array of tools available is essential to ensuring 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 its comprehensive resource monitoring capabilities and capacity to identify constraints, tools such as Nagios are well suited for this purpose. In contrast, application monitoring focuses on the functionality and efficiency of the application in the system. It addresses aspects such as response time, error rate, and transaction tracking that are critical to a seamless order management system user experience.

Maintaining a system log is a supplement to monitoring. It helps developers in tracking bugs and understanding the sequence of events that led to system failure. For example, the ELK (Elasticsearch, Logstash, Kibana) stack is an open-source logging system that collects logs from various sources [23-25], stores them for quick retrieval (Elasticsearch), processes and transforms them (Logstash), and then visualizes them in user-friendly manner (Kibana). This use of resources lays the foundation for a robust, reliable, and efficient procurement management system. ELK enables not only quick identification of system errors, but also proactive problem solving, thus facilitating the delivery of a quality product to the end user.

Monitoring is important for application administration and management, especially in the context of APIs. The purpose of monitoring could be mainly in the following directions:

• Anticipation and preventive measures: Identify potential problems before they escalate.

• Diagnose problems as soon as they become apparent.

• Operational Oversight: Gain a deep understanding of API efficiency and usage patterns.

• Significance of observation:

• Customer Expectations: In today's digital age, customers expect API to deliver consistent performance and availability.

• Functionality Protection: Reliable monitoring mechanisms are required to ensure that the API functions optimally and meets its service level objectives.

Some important metrics to track in an API ecosystem are:

• Requests per second: This metric provides insight into the current API traffic and demand.

• Monitoring the number of crashes can help identify recurring issues or vulnerabilities early.

• Latency: Estimating the response time of an API gives insight into its efficiency and performance.

• Number of users: Monitoring the number of active users can provide insight into the demand and popularity of the system.

• Session Count: This provides an overview of user interaction and engagement with the API.

• Understanding the geographic distribution of users can help optimize server location and improve user experience.

• Monitoring CPU usage can reveal potential limitations or areas requiring optimization.

• RAM Usage: Regular monitoring of memory usage ensures that the system is not overloaded and is running efficiently.

1. Conclusion

This paper explores the dynamics of mobile application development technologies, backend containerization, and post-deployment management, emphasizing 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 an important part of the project requirements is also the backend part of the mobile application. 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 like scalability, isolation, portability, and flexibility. Container orchestration tools like Kubernetes automate container management and facilitate scalability and self-healing.

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

1. Acknowledgment

You may mention here granted financial support or acknowledge the help you got from others during your research work. Simply delete this section if it doesn’t apply.

1. References
2. E. Soares, G. Sizílio, J. Santos, D. Alencar, and U. Kulesza, “The effects of continuous integration on software development: a systematic literature review,” *Empirical Software Engineering*, vol. 27, no. 3, Mar. 2022, doi: 10.1007/s10664-021-10114-1.
3. 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, May 2020, doi: 10.1016/j.infsof.2020.106268.
4. 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*, vol. 48, no. 7, pp. 2570-2583, 1 July 2022, doi: 10.1109/TSE.2021.3064953.
5. “The NIST Definition of Cloud Computing”. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. [Accessed: 10-Oct-2023]
6. S. Stuckenberg, and S. Beiermeister, "Software-As-A-Service Development: Driving Forces Of Process Change" (2012). PACIS 2012 Proceedings. 122. <https://aisel.aisnet.org/pacis2012/122>
7. Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information & Software Technology, 131, 106449. <https://doi.org/10.1016/j.infsof.2020.106449>
8. De La Torre, C., (2023). “Containerized Docker Application Lifecycle with Microsoft Platform and Tools”. Microsoft Learn. Retrieved from: https://learn.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/ [Accessed: 11-Sep-2023]
9. Vettor, R., & Smith, S. (2023). “Architecting cloud native .NET applications for Azure”. Microsoft Learn. Retrieved from: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/ [Accessed: 19-Aug-2023]
10. De La Torre, C., Wagner, B., & Rousos, M., (2023). “.NET microservices. architecture for containerized .NET applications”. Microsoft Learn. Retrieved from: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/ [Accessed: 1-Sep-2023]
11. 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, Jan. 2020, doi: 10.1016/j.procs.2020.04.152.
12. M. U. Haque, L. H. Iwaya, and M. A. Babar, “Challenges in Docker Development: A large-scale study using stack Overflow,” *arXiv (Cornell University)*, Oct. 2020, [Online]. Available: http://arxiv.org/abs/2008.04467
13. “What is Docker?,” Microsoft.com. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined. [Accessed: 1-Aug-2023].
14. T. Menouer, “KCSS: Kubernetes container scheduling strategy,” *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, Sep. 2020, doi: 10.1007/s11227-020-03427-3.
15. 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, pp. 69–81. doi: 10.1007/978-3-030-45989-5\_6.
16. C. K. Rudrabhatla, “Comparison of zero downtime based deployment techniques in public cloud infrastructure,” *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, Oct. 2020, doi: 10.1109/i-smac49090.2020.9243605.
17. N. C. Mendonca, P. Jamshidi, D. Garlan and C. Pahl, "Developing Self-Adaptive Microservice Systems: Challenges and Directions," in *IEEE Software*, vol. 38, no. 2, pp. 70-79, March-April 2021, doi: 10.1109/MS.2019.2955937.
18. R. Kohavi, D. Tang, and Y. Xu, Trustworthy online controlled experiments. 2020. doi: 10.1017/9781108653985.
19. 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.07, pp.194-201, 2021, doi: 10.3991/ijim.v15i07.19291
20. 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, Jul. 2018, doi: 10.1016/j.infsof.2018.02.010.
21. L. Todoranova, R. Nacheva, V. Sulov, and B. 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, Jul. 2020, doi: 10.3991/ijim.v14i11.13711.
22. 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*, pp.27-30, 2018. doi: 10.1145/3194760.3194763
23. R. Nacheva, “Standardization issues of mobile usability,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol.14, no.07, pp.149-157, May 2020, doi: 10.3991/ijim.v14i07.12129.
24. P. Petrov, I. Kuyumdzhiev, R. Malkawi, G. Dimitrov, and O. Bychkov, “Database Ad-ministration Practical aspects in providing digitalization of educational services,” *Inter-national Journal of Emerging Technologies in Learning (iJET)*, vol.17, no.20, pp.274–282, Oct. 2022, doi: 10.3991/ijet.v17i20.32785.
25. B. P. Rao and N. N. Rao, “HDFS Logfile analysis using ElasticSearch, LogStash and Kibana,” in *Integrated Intelligent Computing, Communication and Security*, pp.185-191, 2019. doi: 10.1007/978-981-10-8797-4\_20.
26. F. Ahmed, U. Jahangir, H. Rahim, K. Ali and D. Agha, "Centralized Log Management Using Elasticsearch, Logstash and Kibana," *2020 International Conference on Information Science and Communication Technology (ICISCT)*, Karachi, Pakistan, 2020, pp.1-7, doi: 10.1109/ICISCT49550.2020.9080053.
27. Authors

**1st Author ( write the name here)** is member of the.

**2nd Author ( write the name here)** is a.

**3rd Author ( write the name here)** is member of the.

Article submitted 2020-10-19. Resubmitted 2021-11-18. Final acceptance 2020-11-27. Final version published as submitted by the authors.