computers

MDPI

Article

Availability, Scalability, and Security in the Migration from Container-Based to Cloud-Native Applications

Bruno Nascimento [1], Rui Santos [1], João Henriques [1,2,*], Marco V. Bernardo [1,3] and Filipe Caldeira [1,2]

[1] Informatics Department, Polytechnic of Viseu, 3504-510 Viseu, Portugal; estgv7494@alunos.estgv.ipv.pt (B.N.); estgv16816@alunos.estgv.ipv.pt (R.S.); mbernardo@estgv.ipv.pt (M.V.B.); caldeira@estgv.ipv.pt (F.C.)
[2] CISeD—Research Centre in Digital Services, Polytechnic of Viseu, 3504-510 Viseu, Portugal
[3] Instituto de Telecomunicações, 6201-001 Covilhã, Portugal
* Correspondence: joaohenriques@estgv.ipv.pt

Abstract: The shift from traditional monolithic architectures to container-based solutions has revolutionized application deployment by enabling consistent, isolated environments across various platforms. However, as organizations look for improved efficiency, resilience, security, and scalability, the limitations of container-based applications, such as their manual scaling, resource management challenges, potential single points of failure, and operational complexities, become apparent. These challenges, coupled with the need for sophisticated tools and expertise for monitoring and security, drive the move towards cloud-native architectures. Cloud-native approaches offer a more robust integration with cloud services, including managed databases and AI/ML services, providing enhanced agility and efficiency beyond what standalone containers can achieve. Availability, scalability, and security are the cornerstone requirements of these cloud-native applications. This work explores how containerized applications can be customized to address such requirements during their shift to cloud-native orchestrated environments. A Proof of Concept (PoC) demonstrated the technical aspects of such a move into a Kubernetes environment in Azure. The results from its evaluation highlighted the suitability of Kubernetes in addressing such a demand for availability and scalability while safeguarding security when moving containerized applications to cloud-native environments.

Keywords: availability; scalability; security; orchestration; Kubernetes

check for updates

Citation: Nascimento, B.; Santos, R.; Henriques, J.; Bernardo, M.V.; Caldeira, F. Availability, Scalability, and Security in the Migration from Container-Based to Cloud-Native Applications. *Computers* **2024**, *13*, 192. https://doi.org/10.3390/computers13080192

Academic Editor: Leandros Maglaras and Paolo Bellavista

Received: 28 June 2024
Revised: 2 August 2024
Accepted: 7 August 2024
Published: 9 August 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Many organizations have already recognized the benefits of moving from monolithic to containerized applications. But, as the limitations of container-based applications also become more apparent, they are now focused on scaling them and looking for more efficient, resilient, and flexible solutions.

This work explores how containerized applications can be customized to address modern application requirements when shifting to cloud-native orchestrated environments. With that aim, a Proof of Concept (PoC) demonstrated the technical aspects of such move into a Kubernetes environment in Azure. The results from its evaluation highlighted the suitability of Kubernetes to address such demands for availability and scalability while safeguarding security aspects when moving containerized applications to cloud-native environments. With that aim, this work starts by explaining its motivation and providing the relevant background.

The paper is organized as follows: Section 2 presents related work. Section 3 describes the methodology, the prototype, and the aspects of its deployment. Section 4 presents and evaluates the achieved results. Finally, Section 5 concludes and discusses future work.

1.1. Motivation

Application deployment methodologies have shifted from traditional monolithic architectures to container-based solutions. Containers have provided a robust mechanism for

packaging applications and their dependencies into isolated units that can run consistently across various environments. However, as organizations scale and seek more efficiency, resiliency, security, and flexible deployment strategies, the limitations of container-based applications become more apparent. This drives the motivation for adopting cloud-native approaches, which offer several advantages over traditional container-based deployments.

While container-based applications have revolutionized how software is developed, deployed, and managed, several inherent limitations motivate the transition to cloud-native architectures. The primary shortcomings of container-based applications are due to the existing limitations in scalability and resilience, as they require manual scaling and resource allocation, and they present complexity in their management and operational overhead and limited integration with cloud services while also giving rise to security concerns.

Scaling container-based applications often requires manual intervention or complex scripting, which can be error-prone and inefficient. Moreover, containers present a single point of failure, as containers running on a single node or a limited set of nodes are susceptible to failures that can disrupt the entire application. Containers require the careful management of resources such as CPU and memory to avoid contention and ensure optimal performance. While containers encapsulate dependencies, managing these across different containers and ensuring compatibility can be challenging. Effective monitoring, logging, and troubleshooting across a distributed container environment require sophisticated tools and expertise, adding to the operational overhead. Ensuring security in a containerized environment involves managing vulnerabilities in container images, securing the container runtime, and implementing network policies.

Moreover, by themselves, container-based applications do not leverage the full spectrum of cloud-native services like managed databases, AI/ML services, and serverless functions, which offer higher efficiency and agility.

1.2. Background

An architectural style called cloud-native [1–3] has emerged in contemporary application design as a programming philosophy against on-premise applications [4]. Cloud-native applications are created using the cloud computing (CC) concept. Developing and implementing applications in a cloud-native manner follow the principles of DevOps [5,6], microservices [7,8], Continuous Integration/Continuous Delivery (CI/CD) [9–11], and containers [12,13]. However, there is a growing agreement on the critical concepts and informal design patterns widely used in many successful cloud applications. Cloud-native applications typically involve using microservices [14], containers, and Service Fabrics.

The concept of observability was first introduced in Control Theory [15]. This theory states that if the current state of a system can be ascertained in a finite amount of time using only the outputs, then the system is observable. The application's Quality of Service (QoS) metrics are enforced through the measurement of its total microservice performance. To meet agreed-upon Service Level Agreement (SLA) standards, the system must appropriately externalize its state through instrumentation techniques. Monitoring such requirements will contribute to observing all the collected application performance metrics. These metrics are essential to identify system gaps or anomalies and prevent future issues [16]. Dashboards provide visually appealing capabilities by summarizing numeric data from continuous metric-processing processes and will help to analyze performance bottlenecks [17].

Such an approach allows the fast delivery of new features to adapt to changing demands regarding flexibility, agility, and scalability. These applications often operate at a global scale and scale well, with thousands of concurrent users. Their infrastructure is fluid, and failure is constant. Moreover, the applications are designed to upgrade and test without disrupting production. Finally, security is considered. Its adoption requires significant work and presents new engineering hurdles for businesses [18].

Cloud-native services leverage the scalability and resilience that are associated with cloud infrastructures. Their auto-scaling capability can automatically fit the resources

to the workload demand for optimal performance and cost-efficiency. Load balancers distribute incoming traffic across multiple instances of services, ensuring high availability and fault tolerance [19]. Distributed data storage and caching mechanisms are employed to handle large volumes of data and provide fast and efficient access [20]. Moreover, to ensure optimal performance and detect issues, cloud-native applications require monitoring and observability capabilities to extract insights about the cluster's health and performance and help troubleshoot their applications [21]. In that regard, there exist monitoring tools [22] that can collect and analyze metrics, logs, and traces.

Kubernetes is an open-source system for orchestrating containers. It handles containerized applications' deployment, monitoring, and scaling across multiple hosts. Google originally developed it and then donated it to the Cloud Native Computing Foundation (CNCF) in March 2016 [23]. Often also referred to as "k8s" or "kube", Kubernetes enables users to specify the desired state of an application through concepts like "deployments" and "services". For instance, a user can request the deployment of three cases of a web application. Kubernetes will then launch and continuously monitor the containers, ensuring the application remains desired by automatically restarting, rescheduling, and replicating the containers as needed.

However, migrating containerized applications to the cloud entails addressing the critical requirements of more availability, scalability, and security [24]. Availability is intimately related to the reliability of services. Scalability refers to accommodating additional workload or demand without negatively impacting performance. Finally, security protects data from unauthorized access while maintaining privacy when these data are at rest or in motion. As services are provided in a distributed manner, they contribute to increasing the attack surface. Due to this, they are susceptible to all sorts of threats, such as malware, Distributed Denial of Service (DDoS), and Man-in-the-Middle (MITM) attacks. For example, Internet Of Things (IoT) solutions can combine many distinct hardware, software, and network requirements, impacting cloud-based solutions regarding availability, scalability, and security [25].

First, the availability requirement aims to reduce the impact on service performance and avoid disruption. High availability will require using the cluster nodes' computing (CPU) and storage resources. Those resources are readily available in high-demand scenarios. This requirement also identifies the nodes to be replaced [26,27]. A trade-off between availability and costs should be considered since costs can rise during highly demanding periods.

Second, the infrastructure's scalability requirement aims dynamically fit the demand by leveraging the available resources in case of high-demand workloads and reducing the resources allocated to low-demand areas. This way, it avoids over-allocating resources during less demanding periods. Swapping between low and high demand must happen smoothly and safeguard the previously discussed availability [28].

The third requirement in migrating containerized to cloud-native applications is security. This aims to reduce the attack surface of the application environment. Access management capabilities are centralized and enabled for the whole solution, including the source code repository and container registry [29–31].

This work explores the improvement of containerized applications regarding their availability, scalability, and security capabilities. With that aim, it demonstrates the migration from a containerized [32] to a cloud-native application in Kubernetes [33]. It also explores how this migration can improve availability while maintaining its security.

2. Related Work

Application availability is a relevant topic discussed in the literature as it can severely impact economic and social losses. This is aggravated in the case of critical systems due to errors here threatening the lives of large portions of the population. In [34], the authors explored how to increase the availability of the solution by optimizing Kubernetes configurations. They report that it is possible to improve availability by 55%.

Cloud-based applications offer the capability of scaling horizontally (more instances) and vertically (more resources per instance). A natural evolution to modern applications typically involves processes supported by increasingly automated approaches relying on advanced tools. In [35], Machine Learning was adopted to scale applications while keeping the QoS consistent. A prototype included a first module identifying the component to be scaled, while the second was supported by reinforced learning techniques to learn the best parameters. A more agnostic approach is possible, as demonstrated in [36], where an algorithm is applied to improve the approach used based on characteristics such as the application and demand using the Kubernetes system.

In [37], an algorithm that detects the optimum level of Pod resources for horizontally scaled solutions and fits ongoing workloads is presented. In this way, it is possible to keep applications available while avoiding the use of unnecessary resources and subsequently reducing their costs.

Despite the benefits of adopting distributed capabilities, this increases the attack surface of applications, which then require holistic approaches to security. As discussed by [38], security principles should be followed to keep the environment safe, relying on the best practices, for example, by maintaining up-to-date tools, especially the latest security updates, restricted and profile-based access control, and security policies on component networks. The orchestration of containerized applications improves their resiliency despite the increased attack surface. In [39], the authors discuss this type of solution and present challenges and their exposure to different attacks. The authors of [40] surveyed the literature and gathered the critical features for security regarding cloud-native services.

The adoption of cloud-native frameworks has become increasingly prevalent as organizations seek to leverage the scalability, flexibility, and cost-efficiency offered by cloud computing. This section reviews notable cloud-native frameworks, the fundamental principles underlying their design, and significant contributions to the field.

Docker [41] is another foundational technology in the cloud-native ecosystem. It provides a platform for developing, shipping, and running container applications. Docker's containerization technology encapsulates applications and their dependencies, allowing for consistent and isolated execution environments across different development and deployment stages. This ensures that applications run reliably regardless of where they are deployed, a core tenet of cloud-native development.

Apache Mesos [42] is a cluster manager that simplifies the complexity of running applications on a shared pool of resources. Mesos abstracts CPU, memory, storage, and other computing resources, enabling fault-tolerant and elastic distributed systems. DC/OS (Data Center Operating System), which is built on Mesos, extends these capabilities by providing additional tools and services for deploying and managing containerized applications at scale.

Istio [43] is a service mesh that provides a uniform way to secure, connect, and observe microservices. As cloud-native applications often consist of numerous microservices, Istio's ability to manage service-to-service communication is crucial. It offers capabilities such as traffic management, security, and observability without requiring changes to the application code. Istio integrates seamlessly with Kubernetes, enhancing its native service management capabilities.

Cloud-native Application Bundles (CNAB) [44] is an open-source specification for packaging distributed applications with all their dependencies, configuration, and deployment logic. It provides a consistent way to manage the lifecycle of cloud-native applications, facilitating their installation, upgrading, and deletion across various environments. CNAB aims to address the complexities of deploying applications that span multiple cloud services and providers.

Serverless computing frameworks, such as AWS Lambda [45], Google Cloud Functions, and Azure Functions, represent another significant advancement in cloud-native technology. These frameworks abstract infrastructure management, allowing developers to focus solely on writing code. Serverless architectures automatically scale with

demand and charge based only on execution time, making them highly cost-effective for variable workloads.

The reviewed frameworks and technologies illustrate the breadth of innovations within the cloud-native domain. Kubernetes and Docker have established themselves as foundational container orchestration and management technologies. Apache Mesos and DC/OS offer robust resource management and distributed system solutions. Istio enhances microservice communication and security, while CNAB and serverless architectures represent forward-looking application packaging and deployment approaches. Together, these frameworks enable the development and operation of scalable, resilient, and efficient cloud-native applications, setting the stage for future advancements in cloud computing.

Compared to this related work, our work provides a more comprehensive comparison between container-based and cloud-native architectures, specifically highlighting the limitations of traditional container-based deployments and how cloud-native approaches address these challenges. While previous studies have explored these topics separately, our work integrates these aspects into one cohesive comparison, offering practical insights for organizations considering the transition.

3. Methodology

This section presents the methodology driving the move from existing container-based to cloud-native applications by following key non-functional requirements such as availability, scalability, and security, for which observability is assured via monitoring and dashboards.

Our work's novelty lies in describing the practical aspects of moving containerized applications to cloud-native orchestrated environments, specifically within the Azure Kubernetes Service (AKS), focusing on the specific requirements of availability, scalability, and security.

DevOps and DevSecOps approaches such as Continuous Integration and Continuous Deployment (CI/CD) [46] were adopted to demonstrate the technical aspects of the implementation of a cloud-native PoC from a container-based application while attending to key non-functional requirements such as availability, scalability, and security. They allowed us to define a consistent and reproducible approach at different deployment stages, such as source code development and the provision and management of infrastructure resources [47,48].

Microsoft Azure technology was adopted as the cloud provider service following our previous work on the containerized SmartCollect application [32]. Microsoft Azure provides service compatibility and quality improvement in interconnections while avoiding services' fragmentation. Listing 1 depicts the deployment command for a new AKS cluster in the Free Tier.

Listing 1. AKS cluster deployment.

```
az aks create
    --resource-group $RESOURCE_GROUP
    --name $CLUSTER_NAME
    --tier free
    --generate-ssh-keys
```

Additionally, Kubernetes has the capability of orchestrating containers and supporting automation administration tasks, and it represents a reference in the literature [49].

The first step of this methodology was to design the application, by following a microservice architecture, to be containerized and later deployed to Kubernetes's orchestrated environment. With that aim, a container registry was used to keep the container images safe.The desired state for migrating the SmartCollect application to Kubernetes was described through the use of a YAML API Deployment configuration according to Listing 2.

Figure 1 depicts the architecture of a PoC application migrated to a cloud-native environment, such as the Azure Kubernetes Service (AKS). Several users with a mobile

app interact with the application in the cluster. The cluster includes a load balancer that balances the worker's workloads. In that regard, several resources are used, including secrets, auto-scaling, repair, and upgrade, while monitoring capabilities are offered. Several dashboards are used to depict data for analysis purposes. Moreover, alerts are used to notify users. The features provided by each one of the services in the cluster are made available from the container images in the container registry. These images were previously built from CI/CD pipelines, with the source code from the repository used for building.

Listing 2. SmartCollect API deployment.

```
kind: Deployment
apiVersion: apps/v1
metadata:
spec:
  replicas: 2
  selector:
    matchLabels:
      app: api-deployment
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: api-deployment
    spec:
      containers:
        - name: api-deployment
          image: smartcollectv2.azurecr.io/api:latest
          ports:
            - containerPort: 7118
              protocol: TCP
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 250m
```

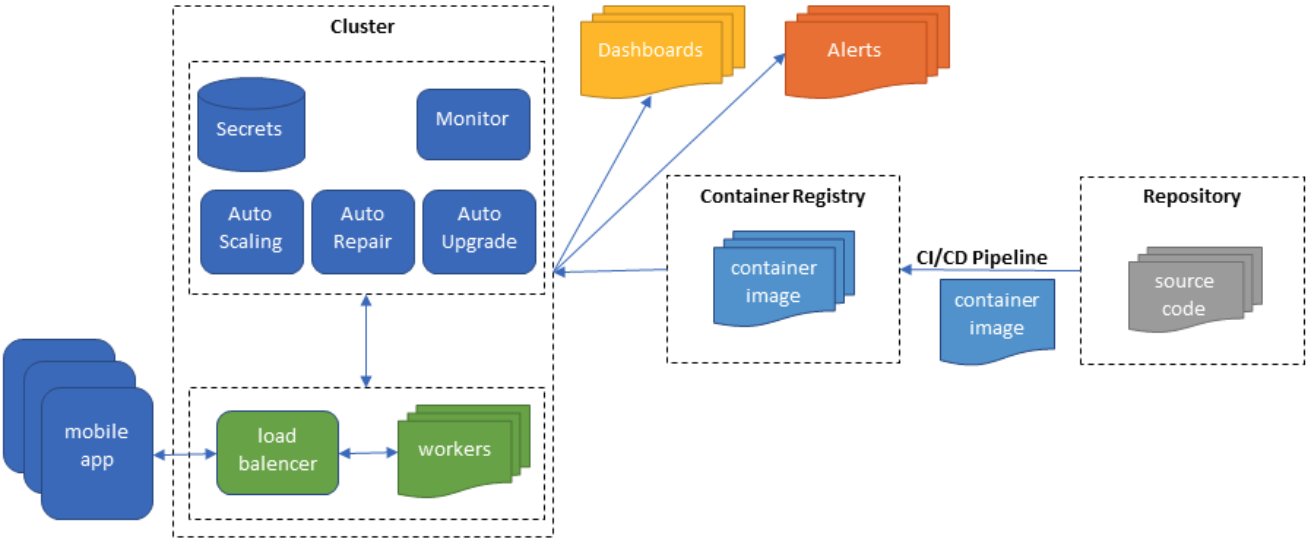

Figure 1. Cloud-native application architecture.

Such a move starts with a new combination in the main branch of the source code repository. Then, a batch script generates a new application image to deploy containers. This image is then stored in the container registry and made available for deployment. The PoC cloud-native application includes scaling applications (scalability) while providing the capability for recovering them (availability) and supporting their upgrade with new container images. Regarding security, authentication mechanisms and other sensitive data are kept in a secrets module and made available on demand. A module collects the metrics to be presented in the dashboards. The services are made available through a load balancer in charge of distributing the workloads.

Regarding the availability requirements, Kubernetes offers cluster deployment capabilities, providing access to and balancing workloads with the application's services through a load balancer.

In terms of the scalability requirement, its implementation was supported using the HorizontalPodAutoscaler (Listing 3), the use of which was dictated by the definition of the range of Pod replicas. It describes the activation conditions to be supported using metrics such as CPU, memory, and other system resources. These will dictate when their deployable units of computing (Pods) should be instantiated or deleted. The target CPU utilization was set to 50% (targetCPUUtilizationPercentage). The maximum number of replicas was set to 5 (maxReplicas), while the minimum was set to 2 (minReplicas).

Listing 3. Horizontal Pod Autoscaler Resource.

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: api-hpa
spec:
  maxReplicas: 5
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api-deployment
  targetCPUUtilizationPercentage: 50
```

Regarding security, Kubernetes' secret resources helped keep important information safe, such as usernames and other sensitive data, such as passwords, and tokens. This way avoids including them in source code or as part of the configuration files. A container registry (Azure Container Registry) is also private and only available to restricted users. Another security-related technology is the Active Directory, which manages and grants authorization access to Azure services by defining the resource authorization of groups and users. By default, public access to services and resources is denied. Also, Namespace resources helped to provide logical separation and access control.

The metrics service Azure Monitor (which has a monitoring role) collects the application's resource metrics, and it is defined according to the YAML definition in Listing 4. These metrics are continuously collected and made available for further analysis. For example, they can support operators in debugging and understanding the denoted system misbehavior. The collected metrics can also be made available to third-party services.

Listing 4. Metrics service: Azure Monitor.

```
- name: metrics-server
  image: mcr.microsoft.com/oss/kubernetes/metrics-server:v0.6.3
  command:
    - /metrics-server
    - '--kubelet-insecure-tls'
    - '--kubelet-preferred-address-types=InternalIP'
    - >-
      --tls-cipher-suites=TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256...
    - '--cert-dir=/tmp'
    - '--secure-port=4443'
  ports:
    - name: ms-https
      containerPort: 4443
      protocol: TCP
  resources:
    limits:
      CPU: 45m
      memory: 55Mi
    requests:
      cpu: 45m
      memory: 55Mi
```

The WorkBooks service (which has a dashboard role) presents metrics appealingly. It includes a wide range of templates based on the platform's experience but also allows for building reports from scratch by customizing graphs and tables.

However, regarding metrics, the Azure Alerts service also creates alerts when identifying abnormal events or behavior. This prevents catastrophic events and avoids unpleasant surprises.

Azure provided the metrics service to extract the metrics associated with a cluster. It is available in all other services and collects many indicators. Its user interface is intuitive and easy to use and helps build different graph types. It is helpful for data analyses and exploration depending on the date range.

4. Evaluation

This evaluation targeted the platform and tasks such as logging different users, querying routes, and searching in the history. The evaluation relied on a Python client producing a large number of different requests .

Figure 2 denotes the analysis of the cluster's behavior when the traffic increases, which is the result of an exponential growth in the number of requests. The graph depicts the inbound data volume (Network In) and the corresponding outbound data volume (Network Out). Such data can be summarized into two distinct types: average and max value. After a while, it is possible to see the outbound data being dropped due to the Azure platform's response, which is to optimize the network's load by leveraging its caching capabilities.

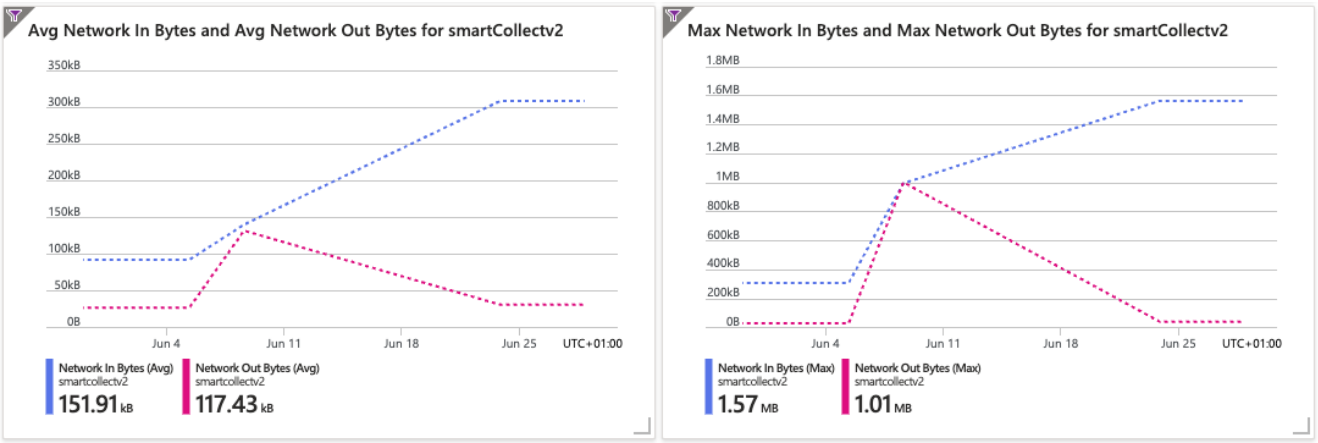

Figure 2. Network traffic.

Regarding availability, Figure 3 depicts the resources allocated to the nodes, including the CPU, memory, and disk usage. It denotes the cluster's ability to cope with a large number of requests as the result of a smooth resource allocation.

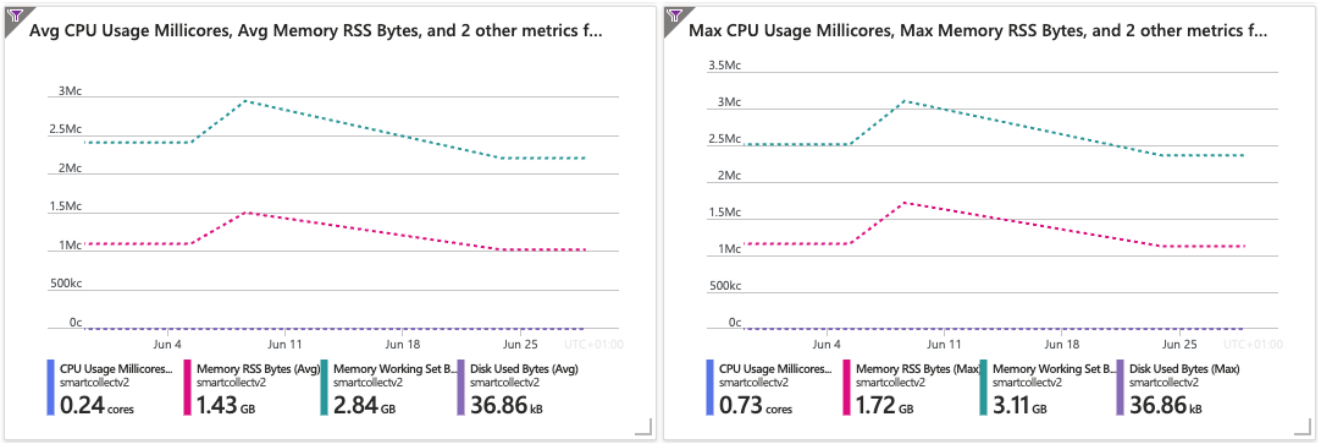

Figure 3. CPU, memory, and disk consumption.

Figure 4 helps to analyze the workload that resulted from requests for available resources. These results highlight the Azure platform's performance, while no impact was seen in terms of availability.

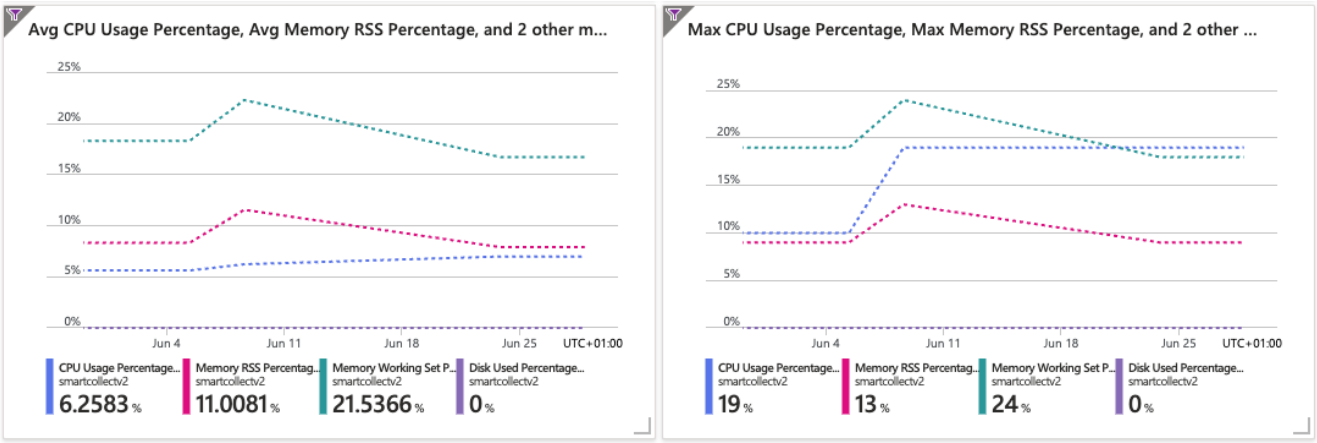

Figure 4. Average and max CPU, memory, and disk usage.

Figure 5 presents the status of all cluster nodes. The collected data include several node resource metrics, such as CPU and memory. The base node available in the Azure Free Tier plan already supports large task loads and offers good performance and robustness. Even under intense workloads, it remained available.

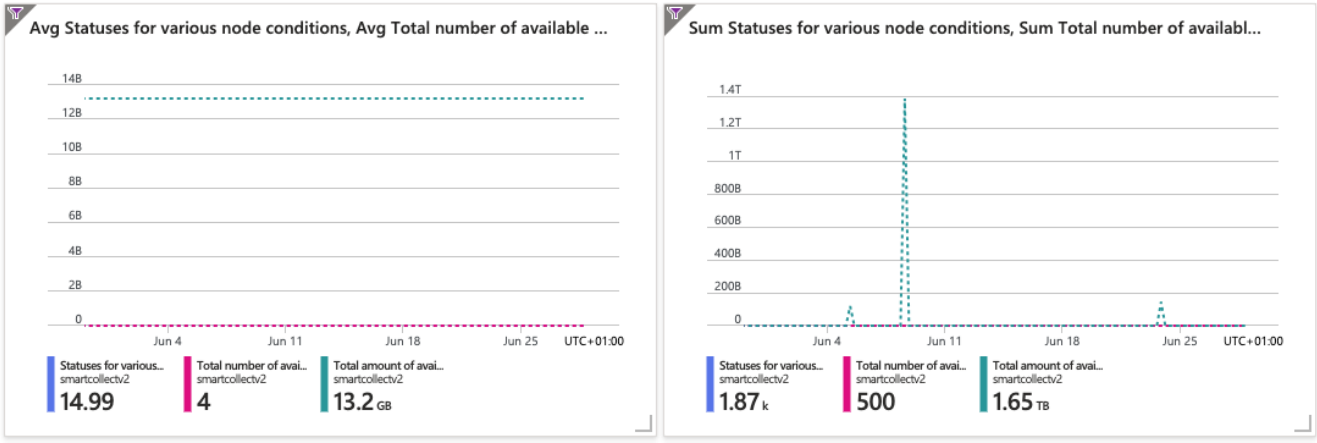

Figure 5. Average and summed node statuses.

The cluster's health status was always acceptable and ran smoothly. Figure 6 depicts its availability, where failures or errors remain undetected. The charts present the data related to the cluster health, Scale Down Cooldown, Unneeded Nodes, and Unschedulable Pods metrics.

Regarding scalability, this was evaluated by increasing the number of requests made to the system. The results denote an increase in the number of pods meeting such a demand. The HorizontalPodAutoscaler resource was demonstrated to be effective in maintaining service performance and availability. Figure 7 denotes this fact through an analysis of the change in state (from available to active). The number of Pods by phase and the number of Pods in a ready state are the reported metrics. Once again, it can be seen that these state changes are carried out progressively, avoiding sudden events that could lead to severe impacts on the system.

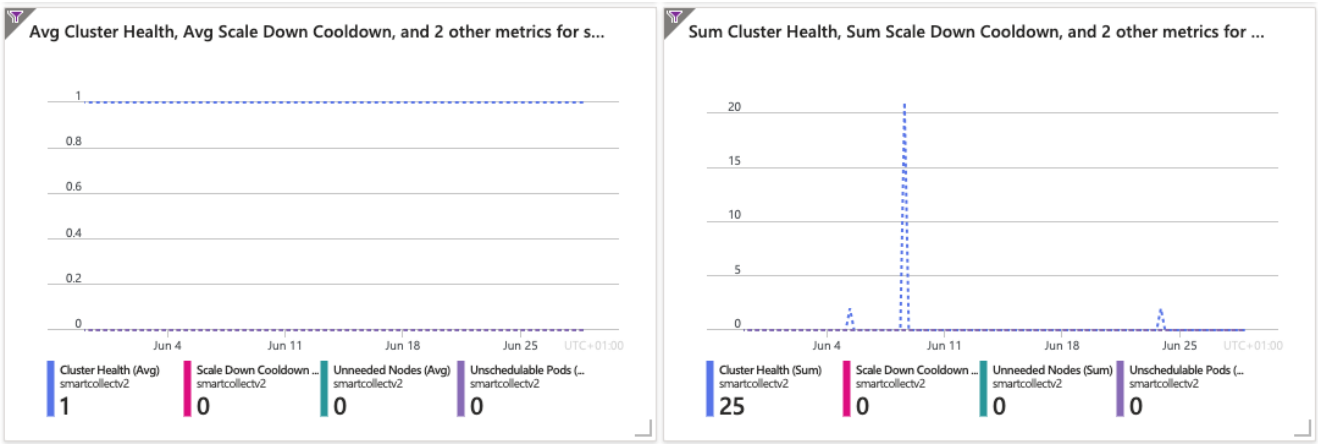

Figure 6. Average and summed cluster health.

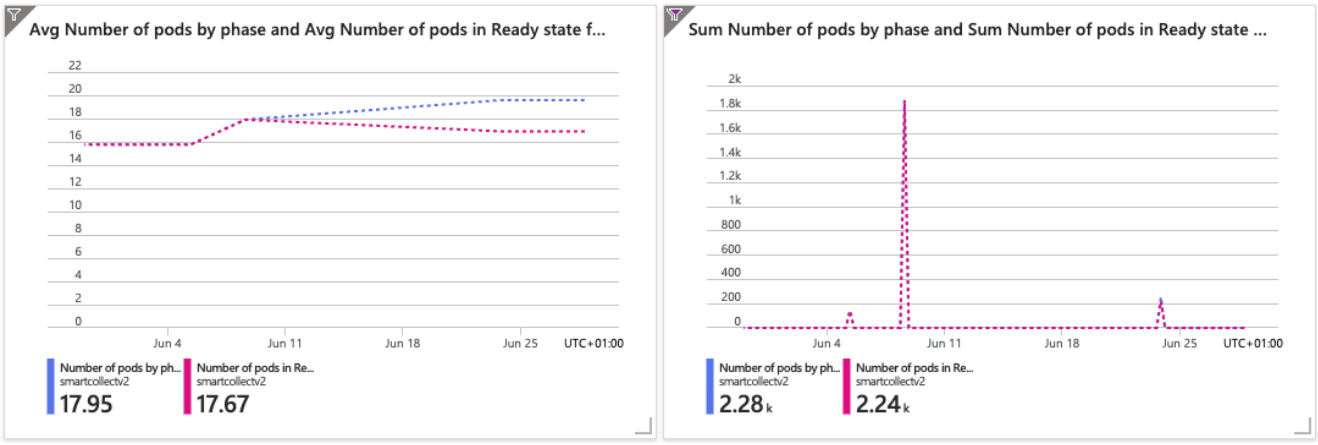

Figure 7. Number of Pods in different phases.

Regarding security, the use of secrets demonstrates its effectiveness by encrypting sensitive data instead of including them as plain text in the source code. The Azure Container Registry helps secure and maintain private application containers and is only made available to authorized users throughout the Access Control (AIM) and Azure Active Directory (AAD). However, the security domain is vast and deserves further research focused on intrusion detection.

Cloud-native applications are designed to exploit the advantages of the cloud environment, fully addressing many of the limitations associated with container-based applications, including enhanced scalability and resilience.

Regarding cloud-native platforms, they provide auto-scaling capabilities that automatically adjust resources based on demand, ensuring their high availability and performance. Moreover, they provide fault tolerance with built-in mechanisms for load balancing, replication, and failover to improve resilience and reduce downtime. In this way, it is possible to streamline service management operations and reduce costs.

Cloud-native environments offer managed services that offload the complexity of maintaining infrastructure, allowing developers to focus on building features. Unified monitoring and logging, integrated monitoring and logging solutions, streamline the operational aspects of maintaining applications.

Cloud-native applications can optimize costs by leveraging pay-as-you-go pricing models and dynamic resource allocation. Moreover, resource efficiency can be achieved with orchestration tools, which ensure efficient resource use, reduce wastage, and improve overall application performance.

Other aspects, such as security and compliance, are also considered. Regarding integrated security, cloud providers offer robust security features and compliance certifications that streamline the implementation of security best practices. Also, regular updates and patches from cloud providers ensure that the infrastructure remains secure and compliant with the latest standards. By transitioning from container-based to cloud-native applications, organizations can achieve greater agility, scalability, and resilience, allowing them to respond more quickly to market demands and reduce their operational complexity. This migration also enables the full potential of cloud services to be leveraged, driving innovation and enhancing the overall efficiency of IT operations.

5. Conclusions

This work demonstrated how container-based applications can be improved, regarding non-functional aspects such as availability, scalability, and security, when migrating container-based applications to a cloud-native environment. Kubernetes provided cluster robustness and resiliency capabilities while keeping the environment secure.

Regarding availability, Kubernetes demonstrated how resources can be managed to provide availability during continuous responses to requests. If the nodes present some issues, this can impact the available services. Therefore, a non-responding node can be quickly removed, and a new one can be created. These errors can be caused for testing purposes, behavior analysis, or other reasons, and the prompt response demonstrates Kubernetes' self-healing capabilities.

In terms of scalability, Kubernetes is a good fit for high-demand environments in terms of fitting to variations in workloads resulting from a high number of requests. Its AutoScaling component offers monitoring capabilities and follows the rules to increase the number of nodes that respond to increased workload levels. On the other hand, the number of cluster nodes can be decreased when the number of requests drops. This guarantees that the number of nodes fits the current workload, thus avoiding the under- and over-allocation of resources.

Still, organizations face increasing numbers of threats and attacks, so security is an important issue. It is a top priority, as systems are continuously available, resilient, scalable, and distributed worldwide.

Although aspects related to the implementation of the PoC and the robustness and complexity of cloud-native environments provide a complete set of features and the capability to cope with the requirements mentioned above, the related costs still lay within the Free Tier plan offered by Microsoft Azure.

In the future, application security can be improved by considering the use of service meshes and mutual TLS to encrypt and certificate the communication channels between services, for example, using Istio. Given their monitoring and dashboard capabilities, such an approach could also improve observability and security.

Moreover, in the future, we aim to compare the performance of container-based and cloud-native approaches in terms of their availability, scalability, and security.

Author Contributions: Conceptualization, B.N. and R.S.; Methodology, B.N. and R.S.; Software, B.N. and R.S.; Validation, B.N. and R.S.; Formal analysis, J.H.; Investigation, J.H.; Resources, J.H.; Writing—original draft, B.N. and R.S.; Writing—review & editing, J.H., M.V.B. and F.C.; Supervision, J.H.; Project administration, J.H.; Funding acquisition, J.H., M.V.B. and F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by National Funds through the FCT—Foundation for Science and Technology, I.P., within the scope of the project Ref UIDB/05583/2020. Furthermore, we thank the Research Centre in Digital Services (CISeD) and the Polytechnic University of Viseu for their support. This work is also supported by FCT/MCTES through national funds and, where applicable, co-funded EU funds under the project UIDB/50008/2020.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Scholl, B.; Swanson, T.; Jausovec, P. *Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
2. Davis, C. *Cloud Native Patterns: Designing Change-Tolerant Software*; Simon and Schuster: New York, NY, USA, 2019.
3. Kratzke, N.; Quint, P.C. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *J. Syst. Softw.* **2017**, *126*, 1–16. [CrossRef]
4. Gannon, D.; Barga, R.; Sundaresan, N. Cloud-Native Applications. *IEEE Cloud Comput.* **2017**, *4*, 16–21. [CrossRef]
5. Wettinger, J.; Andrikopoulos, V.; Leymann, F.; Strauch, S. Middleware-oriented deployment automation for cloud applications. *IEEE Trans. Cloud Comput.* **2016**, *6*, 1054–1066. [CrossRef]
6. Senapathi, M.; Buchan, J.; Osman, H. DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. In Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, New York, NY, USA, 28–29 June 2018; EASE '18; pp. 57–67. [CrossRef]
7. Daya, S.; Van Duy, N.; Eati, K.; Ferreira, C.M.; Glozic, D.; Gucer, V.; Gupta, M.; Joshi, S.; Lampkin, V.; Martins, M.; et al. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*; IBM Redbooks: Armonk, NY, USA, 2016.
8. Balalaie, A.; Heydarnoori, A.; Jamshidi, P. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Softw.* **2016**, *33*, 42–52. [CrossRef]
9. Humble, J.; Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*; Pearson Education: London, UK, 2010.
10. Shahin, M.; Babar, M.A.; Zhu, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access* **2017**, *5*, 3909–3943. [CrossRef]
11. Duvall, P.M.; Matyas, S.; Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk*; Pearson Education: London, UK, 2007.
12. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P. Cloud container technologies: A state-of-the-art review. *IEEE Trans. Cloud Comput.* **2017**, *7*, 677–692. [CrossRef]
13. Jain, S.M. Linux Containers and Virtualization. In *A Kernel Perspective*; Springer: Berlin/Heidelberg, Germany, 2020.
14. Alshuqayran, N.; Ali, N.; Evans, R. A systematic mapping study in microservice architecture. In Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016; pp. 44–51.
15. Kalman, R.E. On the general theory of control systems. In Proceedings of the First International Conference on Automatic Control, Moscow, Russia, 27 June–7 July 1960; pp. 481–492.
16. Quiñones-Grueiro, M.; Prieto-Moreno, A.; Verde, C.; Llanes-Santiago, O. Data-driven monitoring of multimode continuous processes: A review. *Chemom. Intell. Lab. Syst.* **2019**, *189*, 56–71. [CrossRef]
17. Verbert, K.; Ochoa, X.; De Croon, R.; Dourado, R.A.; De Laet, T. Learning analytics dashboards: The past, the present and the future. In Proceedings of the Tenth International Conference on Learning Analytics & Knowledge, Frankfurt, Germany, 25–27 March 2020; pp. 35–40.
18. Kosińska, J.; Baliś, B.; Konieczny, M.; Malawski, M.; Zieliński, S. Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art. *IEEE Access* **2023**, *11*, 73036–73052. [CrossRef]
19. Theodoropoulos, T.; Makris, A.; Violos, J.; Tserpes, K. An automated pipeline for advanced fault tolerance in edge computing infrastructures. In Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, Minneapolis, MN, USA, 1 July 2022; pp. 19–24.
20. Makris, A.; Psomakelis, E.; Theodoropoulos, T.; Tserpes, K. Towards a distributed storage framework for edge computing infrastructures. In Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, Minneapolis, MN, USA, 1 July 2022; pp. 9–14.
21. Theodoropoulos, T.; Makris, A.; Psomakelis, E.; Carlini, E.; Mordacchini, M.; Dazzi, P.; Tserpes, K. GNOSIS: Proactive Image Placement Using Graph Neural Networks & Deep Reinforcement Learning. In Proceedings of the 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2–8 July 2023; pp. 120–128.
22. Ramesh, G.; Logeshwaran, J.; Aravindarajan, V. A secured database monitoring method to improve data backup and recovery operations in cloud computing. *BOHR Int. J. Comput. Sci.* **2022**, *2*, 1–7. [CrossRef]
23. Luksa, M. *Kubernetes in Action*; Simon and Schuster: New York, NY, USA, 2017.
24. Hardikar, S.; Ahirwar, P.; Rajan, S. Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes. In Proceedings of the 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 4–6 August 2021; pp. 1996–2003. [CrossRef]
25. Tchernykh, A.; Babenko, M.; Chervyakov, N.; Miranda-López, V.; Avetisyan, A.; Drozdov, A.Y.; Rivera-Rodriguez, R.; Radchenko, G.; Du, Z. Scalable Data Storage Design for Nonstationary IoT Environment with Adaptive Security and Reliability. *IEEE Internet Things J.* **2020**, *7*, 10171–10188. [CrossRef]
26. Bourke, T. *Server Load Balancing*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2001.

27. Moniruzzaman, A.; Waliullah, M.; Rahman, M.S. A High Availability Clusters Model Combined with Load Balancing and Shared Storage Technologies for Web Servers. *arXiv* **2014**, arXiv:1411.7658.
28. Al-Said Ahmad, A.; Andras, P. Scalability analysis comparisons of cloud-based software services. *J. Cloud Comput.* **2019**, *8*, 1–17. [CrossRef]
29. Alhenaki, L.; Alwatban, A.; Alamri, B.; Alarifi, N. A survey on the security of cloud computing. In Proceedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 1–3 May 2019; pp. 1–7.
30. Joshi, M.; Budhani, S.; Tewari, N.; Prakash, S. Analytical review of data security in cloud computing. In Proceedings of the 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, UK, 28–30 April 2021; pp. 362–366.
31. Sun, P.J. Privacy protection and data security in cloud computing: A survey, challenges, and solutions. *IEEE Access* **2019**, *7*, 147420–147452. [CrossRef]
32. Nascimento, B.; Santos, R.; Henriques, J.; Abbasi, M.; Martins, P.; Bernardo, M.V.; Wanzeller, C.; Caldeira, F. A Framework to Optimize Waste Containers Collection Enabled by an ARIMA Model and IoT Data. In *International Conference on Disruptive Technologies, Tech Ethics and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 205–215.
33. Kubernetes Documentation. Available online: https://kubernetes.io/docs/ (accessed on 25 May 2024).
34. Abdollahi Vayghan, L.; Saied, M.A.; Toeroe, M.; Khendek, F. Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. In Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Guangzhou, China, 22–26 July 2019; pp. 176–185. [CrossRef]
35. Khaleq, A.A.; Ra, I. Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications. *IEEE Access* **2021**, *9*, 35464–35476. [CrossRef]
36. Abdel Khaleq, A.; Ra, I. Agnostic Approach for Microservices Autoscaling in Cloud Applications. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 5–7 December 2019; pp. 1411–1415. [CrossRef]
37. Balla, D.; Simon, C.; Maliosz, M. Adaptive scaling of Kubernetes pods. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–5. [CrossRef]
38. Islam Shamim, M.S.; Ahamed Bhuiyan, F.; Rahman, A. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. In Proceedings of the 2020 IEEE Secure Development (SecDev), Atlanta, GA, USA, 28–30 September 2020; pp. 58–64. [CrossRef]
39. Mondal, S.K.; Pan, R.; Kabir, H.D.; Tian, T.; Dai, H.N. Kubernetes in IT administration and serverless computing: An empirical study and research challenges. *J. Supercomput.* **2022**, 78, 2937–2987. [CrossRef]
40. Theodoropoulos, T.; Rosa, L.; Benzaid, C.; Gray, P.; Marin, E.; Makris, A.; Cordeiro, L.; Diego, F.; Sorokin, P.; Girolamo, M.D.; et al. Security in Cloud-Native Services: A Survey. *J. Cybersecur. Priv.* **2023**, *3*, 758–793. [CrossRef]
41. Docker Documentation. Available online: https://docs.docker.com/ (accessed on 25 May 2024).
42. Apache Mesos Documentation. Available online: http://mesos.apache.org/documentation/latest/ (accessed on 25 May 2024).
43. Istio Documentation. Available online: https://istio.io/latest/docs/ (accessed on 25 May 2024).
44. CNAB Specification. Available online: https://cnab.io/spec/ (accessed on 25 May 2024).
45. AWS Lambda Documentation. Available online: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (accessed on 25 May 2024).
46. Thatikonda, V.K. Beyond the Buzz: A Journey Through CI/CD Principles and Best Practices. *Eur. J. Theor. Appl. Sci.* **2023**, *1*, 334–340. [CrossRef] [PubMed]
47. Walls, M. *Building a DevOps Culture*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2013.
48. Khan, M.S.; Khan, A.W.; Khan, F.; Khan, M.A.; Whangbo, T.K. Critical challenges to adopt DevOps culture in software organizations: A systematic review. *IEEE Access* **2022**, *10*, 14339–14349. [CrossRef]
49. Al Jawarneh, I.M.; Bellavista, P.; Bosi, F.; Foschini, L.; Martuscelli, G.; Montanari, R.; Palopoli, A. Container orchestration engines: A thorough functional and performance comparison. In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.