International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/30509246/IJETCSIT-V1I4P104
Eureka Vision Publication | Volume 1, Issue 4, 30-37, 2020

Original Article

The Evolution of Cloud-Native Architectures: Exploring the Synergy between Kubernetes and Microservices

Santhosh Chitraju Gopal Varma
Software Developer, United States of America (USA).

Abstract - Together, these aspects of virtualization have greatly transformed cloud computing's approach to the development, deployment, and management of applications. Together with the appearance of cloud-native architectures, companies have started to adopt efficient, resilient, and scalable software systems. Kubernetes: A highly useful orchestration tool for managing containerized applications in microservices architecture. This paper focuses on understanding Cloud-native architectures, combining Kubernetes with microservices, and its outcomes for software development and deployment. In this context, we detail the definitions and approaches of promising methods and techniques and the strategies for implementing this domain. We also discuss the potential issues and possible research opportunities for further development of this domain. The study is hoped to provide a heuristic to users who wish to adopt Kubernetes to construct robust microservices platforms.

Keywords - Cloud-native architectures, Kubernetes, Microservices, Containerization, DevOps, Service Mesh, Orchestration, CI/CD, Scalability, Resilience.

1. Introduction

Current changes in the software business due to the digital transition depict the cruciality of cloud-native structures. [1-4] It's been a while since the ever-large monolithic applications have stepped down, and microservices have taken their place in application architectures. Container orchestration tomorrow has principally acquired one winner, the Kubernetes.

1.1. Importance of Cloud-Native Architectures

To look at the current state of architecture for cloud-native applications, it is hard to imagine that applications are developed, deployed, and run differently. With the help of containers and microservices, specifically Kubernetes, organizations may develop secure, scalable, and economical solutions. Here are six characteristics explaining why the architectures have to be cloud-native:

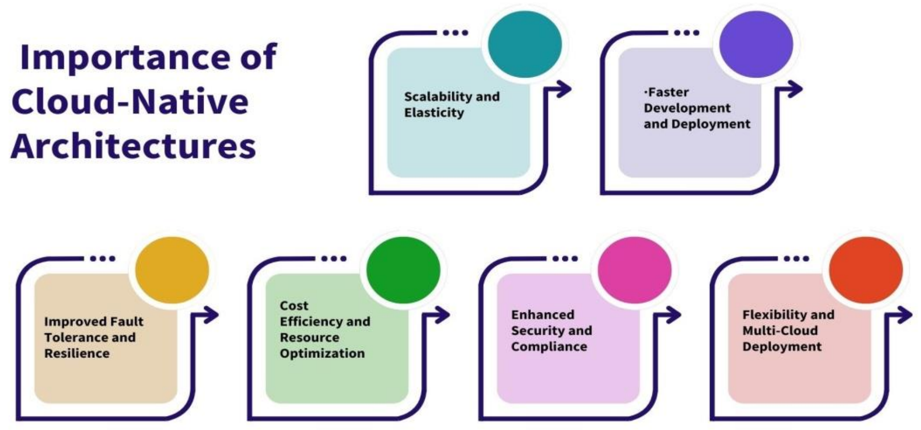
Importance of Cloud-Native Architectures
Scalability and Elasticity
·Faster Development and Deployment
Improved Fault Tolerance and Resilience
Cost Efficiency and Resource Optimization
Enhanced Security and Compliance
Flexibility and Multi-Cloud Deployment

Figure 1. Importance of Cloud-Native Architectures

- **Scalability and Elasticity:** One of the most significant advantages of cloud-native architectures is scalability. Monolith architecture faces scalability issues because, for scaling, entire systems are needed to be replicated. On the other hand, microservices-based architectures are designed to let elements be self-servicing according to the workload. Kubernetes enables the process of horizontal scaling and manages resources to avoid situations where certain applications are overloaded or unassigned due to sudden traffic overflow.
- **Faster Development and Deployment:** CI/CD, which is integration and deployment, is made easy by cloud-native architectures to support the agile development models. It also enables the distribution of updates more often than not to affect the overall system but small manageable chunks. Docker and other similar tools within the containerization and orchestration levels, such as Kubernetes, create a technical environment that accelerates the delivery cycles and shortens the time-to-market, thereby increasing innovation.

- **Improved Fault Tolerance and Resilience:** Traditional applications have a problem of singularity failure points where the failure of a single module causes the failure of the entire application. Fault tolerance is dealt with in cloud-native architectures much differently because it has self-healing capabilities. Kubernetes periodically recognizes failing pods and repurchases them while the load balancer directs traffic only to functioning instances. This helps maintain application availability under high or low workload levels.
- **Cost Efficiency and Resource Optimization:** One of the critical advantages of cloud-native architectures is that they enable low utilization of resources. Kubernetes has proper Auto-Scaling services so that services do not consume more resources than they need. Further, costs are controlled since organizations can adopt a 'pay-as-you-go' cloud model to avoid over-licensing. Lowering costs is taken a notch higher by other features like spot instances and server-less computing that make Cloud-native solutions cheaper.
- **Enhanced Security and Compliance:** The security factor regarding the new generation of data centers cannot be overemphasized. Applications found in cloud-native environments include Role-Based Access Control (RBAC), network policies, and Transport Layer Security (TLS) for their safety and security. Also, Kubernetes offers partitions of workloads with the help of namespaces and pod security policies. Cloud providers also provide ready-made compliance controls/standards such as GDPR and HIPAA to help businesses become more compliant easily.
- **Flexibility and Multi-Cloud Deployment:** Contemporary patterns are designed for multi-cloud and hybrid-cloud, which enables organizations to run workloads on AWS, Azure, Google Cloud, and on-premise. This does create a lack of association and offers a consumer the option to switch providers according to their performance, pricing, and overall accessibility. This means that Kubernetes makes it easier to implement multi-cloud orchestration, and its effective adoption optimises the disaster recovery process.

1.2. Role of Kubernetes in Microservices

One cannot underestimate the role Kubernetes serves in micro-services-based architectures, where microservices are the backbone of various organizations to build, deploy, or scale applications. As an orchestrator on the application level, Kubernetes handles the containerized applications and the microservices that make up the applications while ensuring that they are up and running across the various clouds in a distributed manner. [5,6] In traditional monolithic applications, managing and anticipating dependencies, scaling different resources, and ensuring the high availability of an application typically becomes a massive chore. In the same regard, Kubernetes solves these challenges by providing auto-scaling, self-healing, load balancing, and service discovery, which is why it is an important aspect of cloud-native environments. Another important and obvious benefit of Kubernetes is the feature of scaling microservices depending on their load. In terms of HPA, it is a Kubernetes feature that automatically scales the pods, and along with them, the running containers save resources and keep them at optimal utilized levels.

This elasticity is profound in utilising various high-traffic applications that demand an efficient workload. Further, Kubernetes has features such as automatic container restart, node replacement, and rolling updates, which ensure that microservices are protected from failure. Within the framework, networking, and service discovery are also easy and make the communication of the microservices more effective and secure. Kubernetes provides easy and automatic inter-service communication with objects as intermediaries using KubeDNS/CoreDNS. Besides that, it supports network policies and ingress controllers to provide possibilities for defining the set of rules for secure communication of microservices within an organization's environment. Another feature that Kubernetes facilitates is security. RBAC, secrets management, and Namespace isolation features restrict only some workloads to a certain entity. Moreover, with solutions like Istio, which work in a Kubernetes environment and support service mesh architectures, the system enhances the security, manageability, and traffic flow at a microservices level. Kubernetes ensures productivity in applications based on containers by creating a solid, adaptable, and programmatic environment for the efficient management of the applications. It enables organizations to develop cloud-first applications that can withstand changing business applications.

2. Literature Survey

2.1. Evolution of Cloud Computing

The evolution of cloud computing has revealed substantial changes throughout the years since moving past fundamental virtualization approaches to create advanced distributed computing models. Organizations began their operations using physical servers, resulting in suboptimal resource deployment. Virtualization introduced a technology that enabled multiple virtual machines to share one physical server, thus maximizing hardware resources. [7-10] The evolution of virtualization technologies led to multiple cloud services models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) which provide scalable virtualized computing resources and development environments and runtime support as well as fully managed over-the-internet applications. Organizations can achieve three primary benefits through these cloud service models because they reduce operational expenses and increase scalability while improving IT management efficiency.

2.2. Transition from Monolithic to Microservices Architecture

A traditional monolithic structure unites various components in dense arrangements, yet these designs become more difficult to scale and maintain because of increased application complexity. The transition to microservices architecture solves these difficulties by dividing applications into separated smaller services that operate independently. The system divides itself into multiple compact services that realize individual capabilities while using REST and gRPC APIs to exchange lightweight messages. The modular system structure brings better agility to development teams and independent control abilities to deploy and scale individual services. The microservice design enables continuous integration and deployment through fault isolation support and better resource allocation for improved resilience.

2.3. Kubernetes: An Overview

Kubernetes is the world's leading container orchestration tool, which was developed at Google and later donated to the Cloud Native Computing Foundation. It is a process of deploying, organizing, and managing applications that have been containerized across networks. Some of these features include auto-scaling, where resources are adjusted depending on the current load for an application; service discovery, since microservices communicate with each other; and load balancing, which ensures an even distribution of traffic across instances of an application. Further, Kubernetes enables the feature self-healing, where failed containers are automatically revived, and the declaration configuration, where the infrastructure is coded. These features, in turn, make Kubernetes an essential technology for controlling today's complex cloud-based applications.

2.4. Related Work

Regarding microservices, several works have been conducted to examine Kubernetes functionality. It has been discussed in relation to DevOps and its uses in reaching goals within the deployment pipeline and continuous delivery. It also establishes how Kubernetes raises the level of fault tolerance in that the system can detect and fix failures and maintain the right availability of services. Additionally, studies on Kubernetes have discussed the optimization of resources it employs such that the availability of computing resources to containers and the overall costs have been improved. Furthermore, a comparison with other types of container orchestration tools applied for testing Kubernetes, including Docker Swarm and Apache Mesos, has been made to analyze the specifics of Kubernetes' strengths concerning scalability, automation, and extensibility.

3. Methodology

3.1. Architectural Components

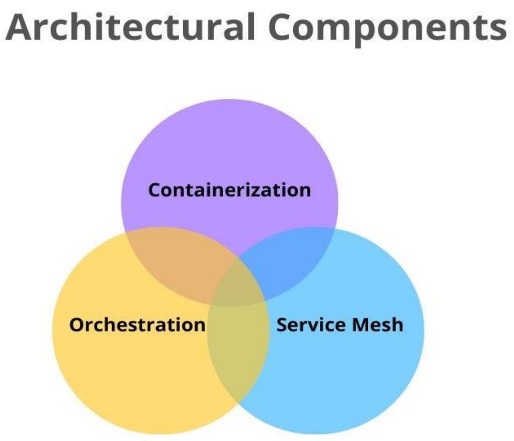

Figure 2. Architectural Components

- **Containerization:** Containers, on the other hand, are an essential technology of cloud-native systems that allows the bundling of applications and all their requirements in the smallest packaging, called a container. Docker is a major platform that supports containerization, which enables developers to ship the application and deploy it with the same specs in different settings. [11-15] However, maintaining multiple containers across different systems is complex, and to overcome this, Kubernetes helps in container orchestration. Containers make it possible for organizations to launch applications in a shorter span, optimize application resources, and be able to run applications on both cloud and physical data centers.
- **Orchestration:** Containerization is a popular method of packaging an application, and Kubernetes is widely used as an orchestration layer for managing applications based on containers. It reduces operational challenges through features like auto-scaling, self-healing, load balancing, and rolling updates. Kubernetes helps to maintain the high availability of applications through its capabilities to restart dead containers, adequately distribute the load, and dynamically run applications based on traffic volumes. It enables infrastructure to be described using code and makes CD and DevOps aspects of infrastructure more effective.
- **Service Mesh:** Microservices are a relatively new concept that has found its way into almost all applications, with most of them being based on complex architectures, and this has greatly challenged ways of communication between the different services. They are addressed by current solutions such as Istio, Linkerd, and Consul by adopting another architecture layer, independent of the application layer, to manage inter-service communication. For instance, Istio adds security and advanced visibility into the services and traffic controls, including encryption, authentication, and

microservices authorization controls. It also enables traffic routing, failure recovery, and load balancing decisions to be undertaken and implemented at runtime, all without any coding change in the application. These technologies are crucial for large-scale microservices, preventing traffic from going through the internet or other uncontrolled paths.

3.2. Implementation Strategies

- **CI/CD Pipelines:** CI CD pipelines have the features for implementing and automating how software applications are built and when they can be deployed. Jenkins, GitLab CI/CD pipelines, and GitHub Actions are all the tools that help with CI/CD. These pipelines ensure that every code change is automatically tested & deployed and minimize the part of the process done manually, hence improving the software's delivery speed. CI/CD pipelines successfully work with Docker and Kubernetes with the help of containerized applications, allowing smooth changes in the microservices environment. Additional advantages of automation, rollbacks, and blue-green deployments help reduce downtime and potential deployment problems
- **Monitoring & Logging:** It goes without question that monitoring and logging are important for managing applications' current status and troubleshooting in real-time. Prometheus is another tool that collects information from a containerized application and can diagnose the performance of an application, while Grafana gives a visual outlook of the data collected. For centralized logging, elastic search, log stash, and Kibana help organizations aggregate, analyse, and visualise logs of different microservices. These observability solutions support DevOps monitoring and measures to get insights on inconsistencies, better performance, and the ability to troubleshoot within the systems.

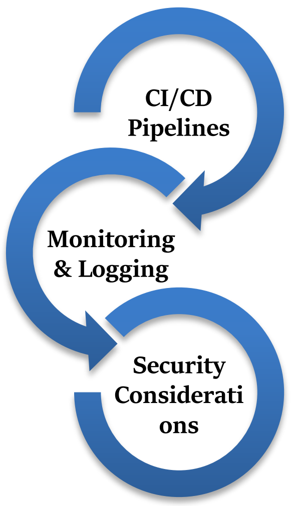

Figure 3. Implementation Strategies

- **Security Considerations:** Security is another challenge for cloud-native applications, as they should have proper access controls and strong network protection. RBAC in Kubernetes is a security concept that limits a certain subject's access to the Kubernetes cluster. Also, within a network, policies are made on how every pod or container interacts with others, which will minimize security risks in the network. Other security controls are the usage of TLS encryption for security, scanning container images for known vulnerabilities, and applying service mesh for inter-service connection security.

3.3. Comparative Analysis

- **Scalability:** Current architecture has some challenges, especially because many follow the monolithic model and traditional architectural designs. Growing in an original system is usually illustrated by adding physical servers or upgrading existing ones, which is cumbersome and costly. Kubernetes) are utilized in cloud-native approaches to support dynamic scaling. It can automatically scale out horizontally where there are more instances or up where there are more resources for a given application.
- **Deployment:** In the classical technical stacks, software deployment is generally a manual process that involves much time for configuration, installation, and updates. As a result, the development environment may not remain consistent with the testing or the production environment. CI/CD pipelines, IaC, and containerization processes are the most common in cloud-native architectures. These hits ensure faster, less susceptible deployments with remarkable efficiency and produce less downtime. Some features include blue and green deployment rolling updates and canary releases.
- **Fault Tolerance:** Monolithic, legacy applications usually have a low level of fault tolerance, and the consequence is that if one element fails, the entire system may stop. Recovery will take some time and may need an administrator's interference as well as a longer period of recovery time. On the other hand, cloud-native systems have inherent aspects of high availability and resilience of the information technology infrastructure. Kubernetes and service mesh technologies allow self-healing, automatic failover, and proper distribution of traffic for high availability of services. Also, it is a

feature that microservices architectures can accommodate isolated failures; if one service fails, the others do not take damage.

- **Cost:** Managing conventional physical IT infrastructures requires relatively high capital and operational expenditures on hardware, hardware maintenance, and human efforts. Due to the fact that traditional systems do not have methods to scale, different organizations over-allocate resources, thus increasing their expenses. This makes cloud-native architectures cost-efficient through scale-out architectures, auto scalability, and consumption-based billing. Thus, containerized applications allow for rationalizing resource consumption and avoiding unprofitable expenses to the overall infrastructure.
-

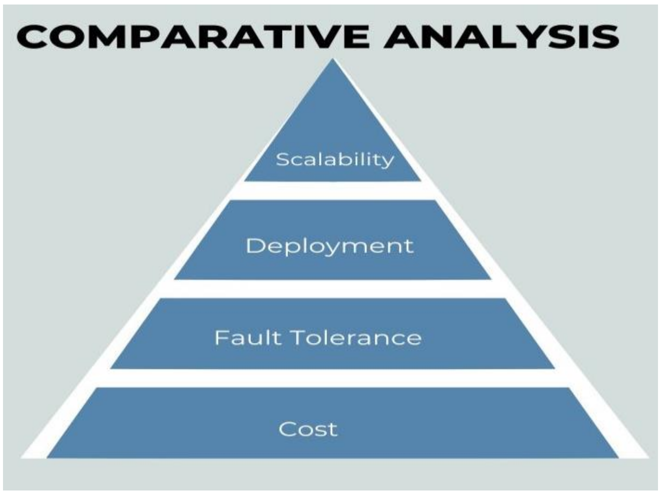

Figure 4. Comparative Analysis

4. Results and Discussion

4.1. Performance Analysis

To evaluate the advantages of microservices as a deployment model on Kubernetes, scalability and response time tests were conducted to compare them with monolithic architectures. The performance parameters that were monitored were request throughput (RPS), latency (ms), and resource usage (%).

4.1.1. Comparison of Response Times

Table 1: Comparison of Response Times

Deployment Model	Average Response Time (ms)	Peak Response Time (ms)
Monolithic	250	450
Kubernetes-Based Microservices	90	180

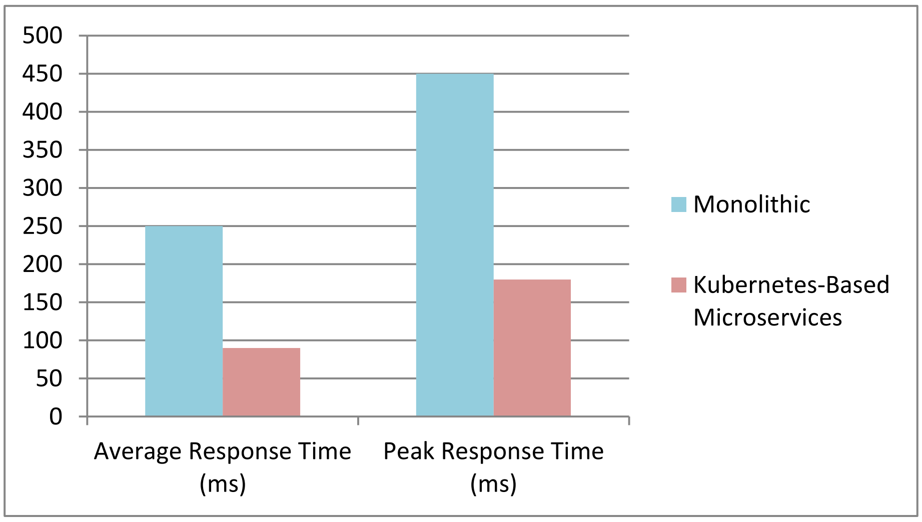

Figure 5. Graph representing Comparison of Response Times

Kubernetes provides microservices with a much faster response time than monolithic systems. Applications in a monolithic deployment environment are single bundled elements of a large application system; they function independently but are strongly interrelated, exposing a number of complexities and requirements competition that hikes the latency levels. That is why single-tier architectures exhibited an average response time of 250 ms for averagely coded pages and a maximum of 450 ms for the heavily loaded ones. The Kubernetes microservices are based on the idea and application of containers and distributed computing, allowing for parallel operations, scaling independently, or efficiently assigning resources. Due to this architecture, response times are lower and are spread out, closely averaging at 90 ms, and in the worst-case scenario, they are not going beyond 180 ms. Since app development is divided into small independent services, the components can grow and shrink flexibly, avoiding the problem of clogging and improving app performance. This significant improvement shows the capability of a microservices architecture that Kubernetes shifted to improve response time and user experience of cloud-native applications.

4.1.2. Throughput Comparison

Table 2: Throughput Comparison

Deployment Model	Requests Per Second (RPS)	CPU Utilization (%)
Monolithic	500	85
Kubernetes-Based Microservices	1200	60

Throughput in terms of Request Per Second (RPS) is an important metric to evaluate how the system can take the load of the throughput. This results in resource contention and low scalability because all applications run in a monolithic architecture where requests are processed in a single and closely integrated system. Therefore, it has low operations per second for these monolithic systems, less than 500 RPS, and percent CPU usage is at 85 percent of the time, showing improper consumption of resources and possible limits on taken capacities under the higher loads. In contrast, microservices deployed on Kubernetes are based on the containerized, distributed computing paradigm where every service is a separate entity managing a certain function independently. This architecture can be scaled horizontally, that is, to add a number of instances of the services in response to a load that is to be processed. Consequently, microservices-based applications sustain a much higher throughput with around 1200 RPS, nearly 2.5 times more than monolithic architectures. In addition,

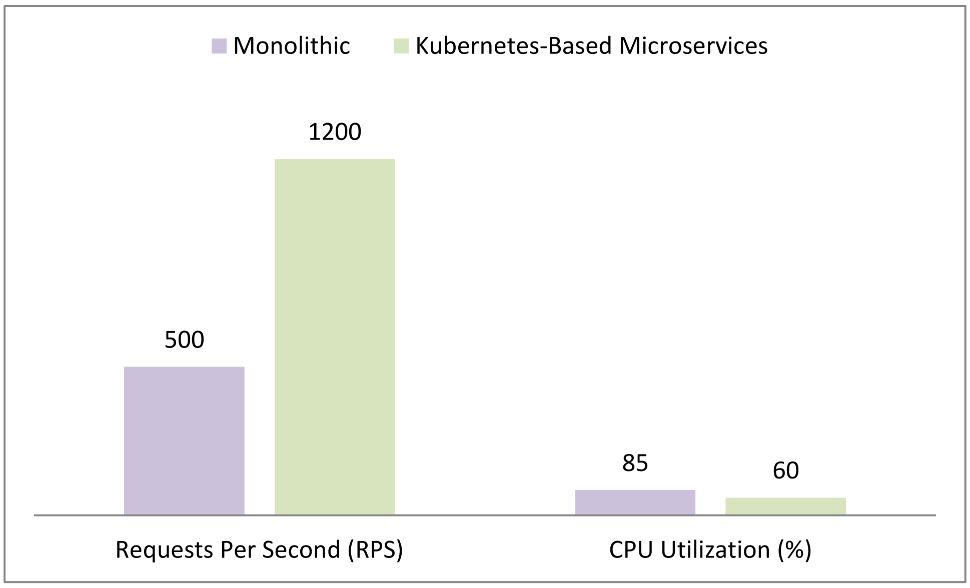

Figure 6. Graph representing Throughput Comparison

CPU usage is set at an average of 60 percent, thus facilitating better CPU resource management and minimizing the possibility of system overload while spreading workloads better. This highlights the impact of microservices architectures on Kubernetes, compliance with scalability, resource management, and increased systems performance in contemporary cloud applications.

4.2. Challenges and Solutions

As organizations incorporate Kubernetes for managing containerized applications, they face various issues dealing with the intricacies, security, and adequately harnessing costs. This paper will discuss these challenges and how they can be addressed.

- **Complexity:** Kubernetes has virtually every tool an administrator will need for container orchestration; however, on the downside, it is rather complex. Its constituent information pieces, including pods, deployments, services, and networking, must be understood in detail. Moreover, the configuration and management of Kubernetes clusters are not simple tasks, particularly when there is no prior experience in the cloud-native environment. Various third-party or paid Kubernetes services such as AWS EKS, Microsoft AKS, and Google GKE have default installation options, auto-scaling features, security updates, and cluster management to ensure more convenient, optimised, and smooth operations. It relieves the overbearing tasks of implementing such instances to a point where the teams can concentrate on creating applications instead of attending to infrastructure issues.

- **Security Concerns:** Kubernetes clusters are open platforms for managing applications and, therefore, if not well secured, are prone to some of these security threats, as outlined below. Some of the most disastrous mistakes are the exposure of services to the public internet or the granting of permissions to users. To reduce these risks, organizations should deny access to resources based on the role of users through RBAC, contain traffic by means of network policies between pods, and secure communication between services through TLS encryption. Further, those who run a service mesh such as Istio or Linkerd get improved authentication and access, allowing specific entities to exchange communications within the cluster.
- **Cost Optimization:** When a Kubernetes cluster is managed across multiple environments, it is realizable that considerable money could be spent on computing resources if they are not managed properly. Having more compute instances than needed and having workloads running when they are not operational result in part of the wastage of costs. In order to manage costs, it is possible to use horizontal pod auto-scaling, allowing for changing the number of currently running pods following the actual usage of CPU or memory. Furthermore, running non-critical applications on spot instances or preemptible VMs is also cost-effective. Kubernetes also has Locality, which contains resource quotas and limits for the services. Its purpose is to allocate resources in the best and most reasonable way and avoid affecting the administration in situations where a lot of money was spent on the service of the cluster.

4.3. Future Trends

Some trends likely to define the direction of Kubernetes as a microservices framework include the following. These innovations, including artificial intelligence for automation, incorporation of edge computing, and serverless Kubernetes, achieve functionality improvements, scalability, and cost optimization.

- **AI-Driven Kubernetes:** Currently, AI and ML are being implemented into Kubernetes to be used in intelligent scheduling of the workload, anticipating the workload, and detecting any problematic patterns. Most scaling techniques revolve around fixed parameters like CPU or memory, which was not the case with AI-driven Kubernetes, which can analyze the application's behavior at runtime and, hence, the probable demands. It also improves the performance and cost optimization by loading balance, minimize resource consumption, and reETS deployment policy. KubeFlow and the newly launched OpenAI Kubernetes-native models are used to optimize the AI workflows and make the Kubernetes environment more automated.
- **Edge Computing:** Kubernetes is evolving from cloud environments to edge computing, executing applications and services closer to end-users and at the IoT edges. This transition is due to the requirements of achieving low latencies, real-time decisions, and reduced data transfer costs. Thus, the response time of autonomous vehicles, industrial automation, 5G-based services, and other applications can be increased by launching microservices on edge nodes. K3s, the lightweight Kubernetes and MicroK8s, have been developed for resource-limited edges, which makes Kubernetes a feasible solution for edge computing architecture.
- **Serverless Kubernetes:** The development of serverless computing affects Kubernetes through deployment through KNative, AWS Fargate, and Google Cloud Run, where developers can run the containers on the cloud without bothering with the infrastructure. Unlike the conventional deployment of nodes and clusters, serverless Kubernetes provides operational overhead, while serverless Kubernetes allows for event-based scaling and inexpensive strategies. It is most suited when many applications are likely to experience fluctuating utilization levels and where resources are dynamically allocated and deallocated accordingly. When Kubernetes is employed together with serverless technologies, organisations could benefit from better cost optimisation, the absence of operational overheads, and better elasticity.

5. Conclusion

The coupling of Kubernetes with microservices has revolutionised cloud-native solutions, making it easy for organizations to scale, maintain high availability and ease operations. Due to the application approach to different services that can be deployed individually, there is the flexibility of easily developing, deploying, and scaling applications. Kubernetes is a container orchestration platform that takes this model further by automating some crucial tasks, including scaling, load balancing, and failing recovery, making it the go-to tool for modern cloud hosting. It has sped up the software development process, optimised the use of resources within the cloud based on demand, and reduced costs.

Nevertheless, for all these benefits that come with Kubernetes as well as microservices, some problems are experienced. Distributed systems management requires additional learning; thus, skills in container orchestration, networking, and security are necessary. The organisation gets challenges when implementing Kubernetes clusters, and data security is easily compromised through various issues with Kubernetes configurations, making organisations vulnerable to various ways, such as access to the wrong people, data theft, and compliance issues. However, cost continues to be an issue of concern since improperly managed Kubernetes clusters can waste many resources on cloud computing. However, recent evolutions that amended managed Kubernetes services such as Amazon EKS, Azure AKS, and Google GKE have, in a big way, minimized the difficulties of the operation while improving security measures. Kubernetes and Microservices in the Future With today's technology trends, such as AI, edge computing, and serverless computing, Kubernetes-based microservices have a great future. Kubernetes uses artificial intelligence for scalable and self-learning through the deployment of workloads, recognition of anomalies, and resource optimisation. This improves performance and minimizes utilization costs, enabling applications to self-tune according to the ever-changing demands.

Furthermore, the advances in edge computing bring the new domain of Kubernetes to IoT uses and 5G networks by providing low-latency processing. Such decentralization of computing power makes it possible for organizations to deploy services closer to the users, thus freeing the backbone of the networks to handle real-time data processing. Also, serverless Kubernetes like KNative and AWS Fargate are causing less dependency on physical infrastructure, making cloud-native deployment even cheaper.

To sum up, due to the constant development of cloud services, organizations should follow new technologies and industry standards to achieve the desired outcome. Both Kubernetes and microservices have changed cloud-native applications significantly, and continuous improvement and development in the coming years will enhance this model significantly with the involvement of new technologies like Artificial Intelligence, Security, and Edge computing. Thus, by being flexible, security-oriented, and cost-saving, businesses can maximize the Kubernetes-based architectures' potential and design long-term, scalable, reliable, and innovative architectures for the digital age.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

[2] Buyya, R., Yeo, C. S., & Venugopal, S. (2008, September). Market-oriented cloud computing: Vision, hype, and reality for delivering its services as computing utilities. In 2008 10th IEEE International Conference on high-performance computing and communications (pp. 5-13). IEEE.

[3] Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2019, July). Microservice-based architecture: Towards high-availability for stateful applications with Kubernetes. In 2019 IEEE 19th international conference on software quality, reliability, and security (QRS) (pp. 176-185). IEEE.

[4] Lewis, J., & Fowler, M. (2014, March). A Definition of this New Architectural Term.

[5] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering, 195-216.

[6] Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect's perspective. Addison-Wesley Professional.

[7] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50-57.

[8] Pahl, C., Jamshidi, P., & Zimmermann, O. (2018). Architectural principles for cloud software. ACM Transactions on Internet Technology (TOIT), 18(2), 1-23.

[9] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015, September). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. 2015 the 10th computing conference was held in Colombia (10ccc) (pp. 583-590). IEEE.

[10] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux j, 239(2), 2.

[11] Turnbull, J. (2014). The Docker Book: Containerization is the new virtualization. James Turnbull.

[12] Kang, H., Le, M., & Tao, S. (2016, April). Container and microservice-driven design for cloud infrastructure devops. In 2016 IEEE International Conference on Cloud Engineering (IC2E) (pp. 202-211). IEEE.

[13] Davis, C. (2019). Cloud Native Patterns: Designing Change-Tolerant Software. Simon and Schuster.

[14] Laszewski, T., Arora, K., Farr, E., & Zonooz, P. (2018). Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud. Packt Publishing Ltd.

[15] Gilbert, J. (2018). Cloud Native Development Patterns and Best Practices: Practical architectural patterns for building modern, distributed cloud-native systems. Packt Publishing Ltd.

[16] Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2019). Kubernetes as an availability manager for microservice applications. arXiv preprint arXiv:1901.04946.

[17] Sayfan, G. (2019). Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes. Packt Publishing Ltd.

[18] Luksa, M. (2017). Kubernetes in action. Simon and Schuster.

[19] Arundel, J., & Domingus, J. (2019). Cloud Native DevOps with Kubernetes: building, deploying, and scaling modern applications in the Cloud. O'Reilly Media.

[20] Carrasco, J., Cubo, J., & Pimentel, E. (2014, September). Towards a flexible deployment of multi-cloud applications based on TOSCA and CAMP. In European conference on service-oriented and cloud computing (pp. 278-286). Cham: Springer International Publishing.