IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 11, November 2024

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.524

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

Kubernetes-Native API Gateways: Leveraging Ingress Controllers and Service Mesh for Scalable Microservices Communication

Anusha Kondam

VP/Lead Software Engineer, JP Morgan Chase & Co, Texas, USA

ABSTRACT: Kubernetes, the open-source container orchestration platform, has become the de facto standard for deploying and managing highly scalable microservices. As these microservices communicate, a high-performance and reliable API gateway is needed to ensure smooth and efficient communication. Traditional API gateways can cause performance bottlenecks and increased complexity, hindering the scalability and flexibility of microservices. To address this issue, a new approach has emerged, leveraging the built-in capabilities of Kubernetes to enable Kubernetes-native API gateways. This approach relies on two key components: Ingress controllers and Service Mesh. Ingress controllers are responsible for routing and load-balancing traffic to the appropriate microservices. Meanwhile, Service Mesh provides a dedicated communication layer between microservices, allowing for secure and reliable communication. The use of Kubernetes-native API gateways offers several benefits. Firstly, it eliminates the need for an external API gateway, reducing complexity and single points of failure. Additionally, it enables seamless integration with Kubernetes and DevOps processes, improving efficiency and agility.

KEYWORDS: Scalability, Microservices, Reliable Communication, Reliability, Reducing Complexity

I. INTRODUCTION

Kubernetes-native API gateways are a type of software that enables efficient communication between microservices in a Kubernetes environment. They leverage Kubernetes's Ingress Controller and Service Mesh components to provide a scalable and secure way for microservices to interact[1]. The Ingress Controller is responsible for routing external traffic into the cluster, while the Service Mesh manages the internal communication between microservices. By using these components, Kubernetes-native API gateways can effectively handle the traffic between microservices, regardless of their location within the cluster. One of the main benefits of using Kubernetes-native API gateways is their scalability[2]. As the number of microservices in a cluster grows, the Ingress Controller and Service Mesh can handle the increased traffic without any performance loss. This is achieved through load balancing and intelligent routing, ensuring each microservice receives the appropriate resources. Additionally, these API gateways provide a secure way for microservices to communicate with each other. They use authentication and authorization mechanisms to ensure only authorized microservices can access the API[3]. They also offer data encryption in transit, protecting against potential security threats. From a technical perspective, Kubernetes-native API gateways are implemented as a layer on top of the Ingress Controller and Service Mesh, often using custom controllers and operators. This allows for easier management and configuration of the API gateway and integration with other Kubernetes components[4]. In summary, Kubernetes-native API gateways are essential for building scalable and secure microservices architectures in a Kubernetes environment. By leveraging the Ingress Controller and Service Mesh, they provide an efficient and robust solution for communication between microservices, enabling organizations to build and manage complex applications easily. Kubernetes-native API gateways are rapidly gaining popularity as they provide a scalable and secure way for microservices to communicate[5]. These gateways are designed specifically for the Kubernetes platform, an open-source container orchestration system. However, several technical issues arise when using Kubernetes-native API gateways, which must be addressed to ensure the smooth functioning of microservices. One of the most significant challenges is the proper configuration and management of Ingress controllers[6]. Ingress controllers are used to manage inbound traffic to a Kubernetes cluster. However, as microservices are added or removed, the configuration of the Ingress controller must be updated continuously. With proper management, this can quickly become a smooth and easy task. Additionally, Ingress controllers may require custom configurations for specific use cases, leading to further complexity[7]. Another major issue is integrating service mesh technology with the API gateway. The main contribution of the research has the following:

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

- Simplifying Microservices Connectivity: Kubernetes-native API Gateways leverage Ingress Controllers and Service Mesh to simplify the communication between microservices within a Kubernetes cluster.
- Improving Scalability and Flexibility: Using Ingress Controllers and Service Mesh in Kubernetes-native API Gateways allows for greater scalability and flexibility.
- Enhancing Security and Observability: Kubernetes-native API Gateways provide built-in security features, such as authentication, authorization, and encryption, to ensure secure communication between microservices.

The remaining part of the research has the following chapters. Chapter 2 describes the recent works related to the research. Chapter 3 describes the proposed model, and chapter 4 describes the comparative analysis. Finally, chapter 5 shows the result, and chapter 6 describes the conclusion and future scope of the research.

II. RELATED WORDS

Li, Y., et al.[11] have discussed Serverless computing, a cloud computing paradigm in which the responsibility of managing and provisioning servers is shifted from the developer to the cloud provider. Konstantoudakis, K.,et,al.[12] have discussed a real-time adaptive streaming FaaS (Function as a Service) service, which is a cloud-based platform designed to provide small-session-oriented immersive media content in real time. Lertpongrujikorn, P., et,al.[13] have discussed that serverless clouds enable object abstraction by providing a platform where users do not have to worry about the underlying infrastructure. Ferreira, J. et al.[14] have discussed The Reactive Microservices-An Experiment, a doctoral dissertation that explores the concept of reactive microservices. Félix, R. C. et al.[15] have discussed the methodology for recommending microservices architectures. This involves assessing the current system, identifying potential microservices, determining data and communication requirements, defining service interfaces and boundaries, testing and evaluating. Cattermole, A. D.et,al.[16] have discussed Runtime adaptation in functional stream processing systems, which involves modifying the system's behaviour during execution to adjust to changing processing requirements. Pallewatta, S., et,al.[17] have discussed the framework, which involves the automated placement of microservices-based IoT applications in federated fog environments. Manco, M. et al.[18] have discussed Cloud Native Security in Service Mesh, a doctoral dissertation that explores the challenges and potential solutions for ensuring secure interactions between services in a cloud environment. Buscema, G. et al.[19] have discussed Security orchestration in Kubernetes with Verefoo. This allows for centralized management of security policies by configuring and coordinating security measures for containers and applications running on Kubernetes clusters. Neves, S. et al.[20] have discussed Mesh microservices on Kubernetes, a way to manage and orchestrate all the microservices within a Kubernetes cluster.

III. PROPOSED MODEL

The proposed model for Kubernetes-native API Gateways involves leveraging both Ingress Controllers and Service Mesh for scalable communication between microservices. Ingress Controllers act as a gateway between external traffic and the microservices running within Kubernetes clusters.

$$IZ(C) = IZ_i + (C - C_b)\frac{IZ_v - IZ_i}{C_v - C_c} \tag{1}$$

$$V_{G2G} = \max(VQ_{d1},....,VQ_{dY}) - \min(VQ_{f1},...,VQ_{FM}) \tag{2}$$

They provide advanced routing and load balancing capabilities, route configuration, and SSL termination. On the other hand, Service Mesh is responsible for managing communications between microservices within a Kubernetes cluster.

$$LO_{ObjStorage} = (F * CQ)c \tag{3}$$

$$LO_{Floki} = (V * FIC + 2 * F * D * QIC)_N \tag{4}$$

It ensures secure, reliable, and scalable communication through its data and control plane components. Data plane components handle the actual data transfer between microservices, while the control plane manages service discovery, traffic routing, and security policy enforcement.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

$$vWaves = \left\lceil \frac{mMissTasks}{QR \times Gh_{tDFOq}} \right\rceil \quad (5)$$

$$CVB(JVL) = 5.77 \times JVL + 1.89 \quad (6)$$

The second policy is mostly used for debugging purposes, and hence we mainly focus on the first policy.

$$N_{b,y} = (L_{b,y} + 120) S_{b,y} \quad (7)$$

$$b_y^+ = \arg\max_b \{N_b, y\} \quad (8)$$

$$q_v = \sum_{b=1}^{m} z_b \times pf \quad (9)$$

Ingress Controllers and Service Mesh provide a holistic approach to API gateway deployment in Kubernetes environments. Ingress Controllers handle external traffic and routing, while Service Mesh manages communication within the cluster.

$$p_{QTL}(h) = z\phi(h) + i \quad (10)$$

$$\min_{z,i} \frac{1}{2} \| z \|^2 + D\xi \quad (11)$$

$$D_I(t) = \sum_{q \neq t \neq v \in T, q \neq V} \frac{\sigma_{qv}(t)}{\sigma_{qv}} \quad (12)$$

Overall, this model helps simplify the management and scalability of API Gateways and microservices within Kubernetes environments.

3. 1. Construction

Kubernetes-native API Gateways are a crucial component for managing and securing microservices communication. They act as a traffic controller, directing incoming requests to the appropriate microservices based on different criteria.

$$D_b = \frac{1}{\sum_a c_{ab}} \quad (13)$$

$$TotalValue = Z1 * t1 + Z2 + ... Z_m * t_m \quad (14)$$

Gateway:

A class called the Gateway brings a modern computing system to life: it connects many of these pieces and provides a means for moving data across components.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

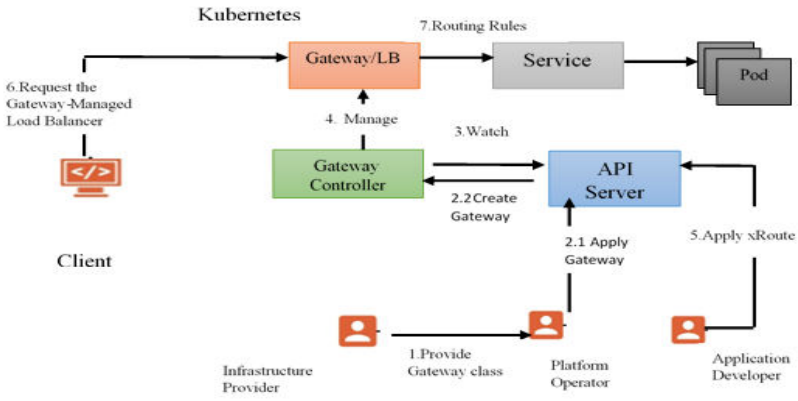

Fig 1 Construction diagram of the proposed model

Application developer:
The Application developer must know about the Gateway's protocols and capabilities. Finally, all these operations benefit the Provider Gateway class.

Service Meshes provide advanced network and communication capabilities for microservices, including service discovery, load balancing, and end-to-end encryption. This can greatly simplify managing microservices communication, especially in a large and complex system.

$$V_{G2G} = \max(VQ_{d1},...,VQ_{DY}) - \min(VQ_{f1},...VQ_{fM}) \quad (15)$$

This allows for better performance and security and easier maintenance and scalability as the system grows. Additionally, using these native Kubernetes features can help reduce the complexity and overhead of managing a separate API Gateway solution, making it a cost-effective option for organizations utilizing Kubernetes for their microservices architecture.

3. 2. Operating principle
This operating principle allows for seamless communication between microservices and external clients or services and scalable and efficient traffic routing within the cluster. It also provides additional features such as load balancing, traffic encryption, and security policies to enhance the overall performance and security of the microservices architecture.

$$k = \frac{2^h * 10}{60 * 24 * 365} \quad (16)$$

The key components of Kubernetes-native API gateways are the Ingress Controller and the Service Mesh. The Ingress Controller acts as a traffic router between external clients and the microservices within the cluster.

Key cloak:
Kong Konnect provides layer-7 load balancing, with Keycloak managing user authentication and authorization. Utilize paths to the corresponding microservices. Fig 2 Shows that the Operating Principle of the proposed model.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

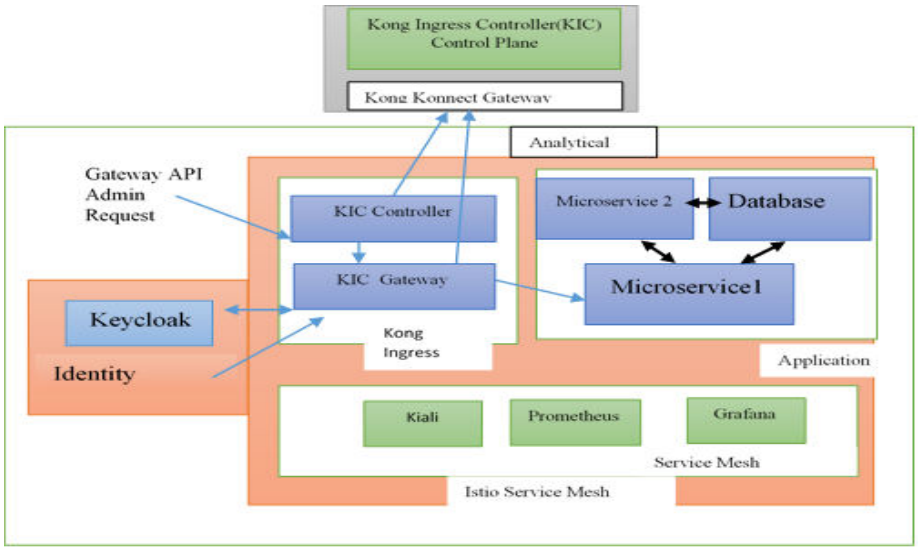

Fig 2 Operating Principle of the proposed model

Kong Konnect Database:
Kong Konnect needs a database called KongKonnect Database, which keeps details about API configuration, user profiles, and access policies.

This allows for dynamic and flexible management of the network traffic. On the other hand, the Service Mesh is responsible for the communication within the cluster. It utilizes sidecar proxies, deployed along with each microservice, to handle the network traffic between services. These proxies provide advanced features such as automatic service discovery, routing, load balancing, and security policies. This distributed architecture allows efficient and resilient communication between microservices without relying on a centralized gateway. Together, these components provide a scalable and efficient solution for microservices communication within a Kubernetes cluster. By leveraging the native capabilities of Kubernetes, API gateways can seamlessly integrate with other features such as auto-scaling, service discovery, and container orchestrations. This leads to improved microservices architecture performance, reliability, and maintainability.

IV. RESULT AND DISCUSSION

Table.1: Comprehensive Analysis

4. 1. Scalability: The API gateway should be able to handle a high volume of traffic and automatically scale up or down to meet demand. Fig 3 Shows the Computation of Scalability.

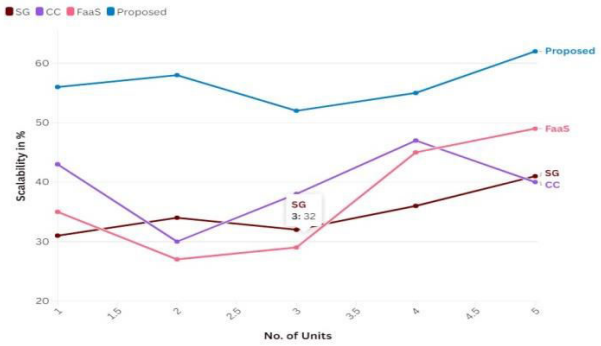

Fig 3 Computation of Scalability

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

This is crucial for microservices architectures, which often involve hundreds or thousands of services communicating with each other.

Table.2 Comprehensive Analysis

4. 2. Performance and Latency: The gateway should have low latency and high throughput to ensure efficient communication between microservices. Fig 4 Shows the Computation of Performance and Latency.

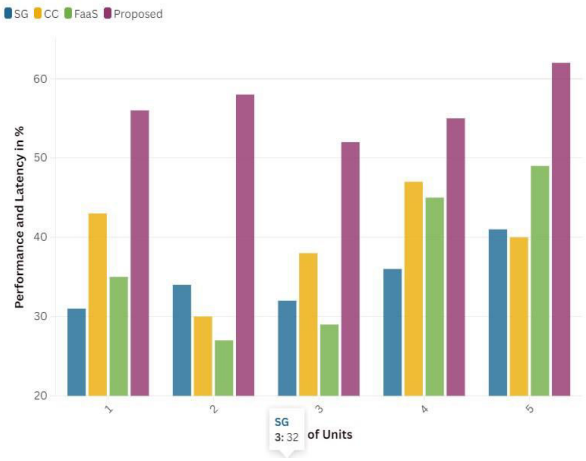

Fig 4 Computation of Performance and Latency

This is especially important for real-time applications that require low response times.

Table.3 Comprehensive Analysis

4. 3. Reliability and Availability: The API gateway should be reliable and highly available, with built-in load balancing and failover capabilities. Fig 5 Shows the Computation of Reliability and Availability.

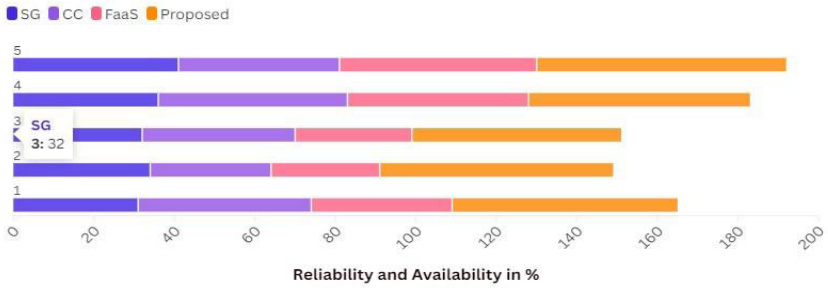

Fig 5 Computation of Reliability and Availability

This ensures the microservices are always accessible, even during failures or high traffic.

Table.4 Comprehensive Analysis

4. 4. Security: The gateway should have robust security features such as authentication, authorization, and encryption to protect against potential security threats. Fig 6 Shows the Computation of Reliability and Availability.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

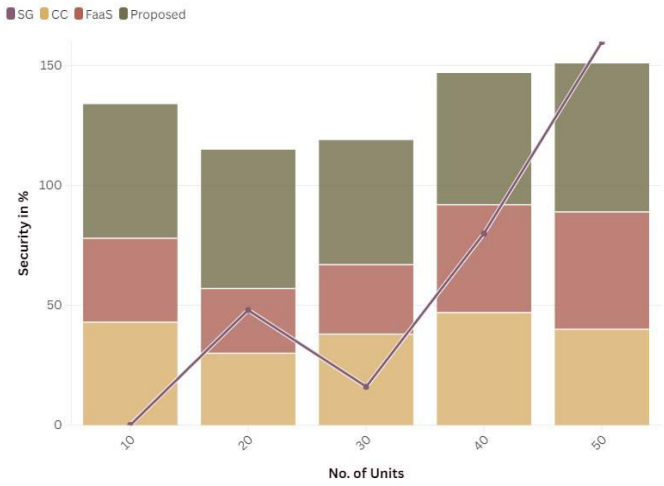

Fig 6 Computation of Reliability and Availability

It should also be able to enforce access control policies to restrict access to specific services or endpoints.

V. CONCLUSION

In the microservices world, it is all about how services communicate. Kubernetes is one of the most versatile platforms for hosting and managing microservices, so there are many ways to handle service-to-service communication. In such a landscape, Kubernetes-native API gateways are essential in that they create an abstracted and normalized way for microservices to communicate with one another inside of the limits of any given Kubernetes cluster. An API gateway serves as a bridge between the clients and the backend services — it provides an external surface area through which client applications can access one or more microservices. By contrast, Kubernetes Ingress is an API object that manages external access to the services hosted inside a cluster. The Ingress Controller routes incoming requests based on rules defined in the ingress resource to its respective service. Kubernetes native API gateways offer a valid response that leverages the combined functionality of both APIS gateway…and APIs in the case of Ingress controllers. Using Ingress controllers allows API gateways to do that without reliance on manually defining routing rules per service. It ensures the requests are routed efficiently and consistently, even when services come up or go down frequently. In addition, Kubernetes-native API gateways are also integrated with service mesh technologies such as Istio or Linkerd. Service meshes offer capabilities such as service discovery, load balancing, and traffic shaping to manage and monitor communications between services in the cluster. Kubernetes-native API gateways integrate these service mesh technologies to secure, reliably route, and scale microservices communication. Kubernetes-native API gateways offer a straightforward solution to traffic control and security for microservices in Kubernetes clusters. Through the infrastructure of Ingress controllers and connecting to service meshes, they provide a clean way to navigate complex microservices. With Kubernetes growing mainstream, k8s-native API gateways will prove quintessential in building resilient microservices architectures.

REFERENCES

1. Kuikka, S. (2024). Kubernetes Networking: Comparative Insights into API Gateways and Service Mesh Implementations.
2. Moshfeghifar, A. (2022). Active Disaster Recovery Strategy for Applications Deployed Across Multiple Kubernetes Clusters, Using Service Mesh and Serverless Workloads (Master's thesis).
3. Hinterholzer, S., Merz, I., Hintemann, R., Beucker, S., Pestarino, C., & Rimassa, G. (2022). D7. 1MARKET ANALYSIS.
4. Malhotra, A., Elsayed, A., Torres, R., & Venkatraman, S. (2024). Evaluate Canary Deployment Techniques using Kubernetes, Istio and Liquibase for Cloud Native Enterprise Applications to Achieve Zero Downtime for Continuous Deployments. IEEE Access.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 11, November 2024

|DOI: 10.15680/IJIRSET.2024.1311003|

5. Patwary, M., Ramchandran, P., Tibrewala, S., Lala, T. K., Kautz, F., Coronado, E., ... & Bhandaru, M. (2023, November). Edge Services. In 2023 IEEE Future Networks World Forum (FNWF) (pp. 1-68). IEEE.
6. Benedetti, P., Gattobigio, L., Steenhaut, K., Femminella, M., Reali, G., & Braeken, A. (2023). Open-source serverless for edge computing: a tutorial. Serverless Computing: Principles and Paradigms, 121-147.
7. Lucani, A. (2023). Liquid Computing on Multiclustered Hybrid Environment for Data Protection and Compliance (Doctoral dissertation, Politecnico di Torino).
8. Usai, C. (2021). Delivering Resilient Virtualized Services in Smart Grid Environments (Doctoral dissertation, Politecnico di Torino).
9. Bouizem, Y. (2022). Fault tolerance in FaaS environments (Doctoral dissertation, Université Rennes 1).
10. Köhler, A. (2022). Evaluation of MLOps Tools for Kubernetes: A Rudimentary Comparison Between Open Source Kubeflow, Pachyderm and Polyaxon.
11. Li, Y., Lin, Y., Wang, Y., Ye, K., & Xu, C. (2022). Serverless computing: state-of-the-art, challenges and opportunities. IEEE Transactions on Services Computing, 16(2), 1522-1539.
12. Konstantoudakis, K., Breitgand, D., Doumanoglou, A., Zioulis, N., Weit, A., Christaki, K., ... & Daras, P. (2022). Serverless streaming for emerging media: towards 5G network-driven cost optimization: A real-time adaptive streaming FaaS service for small-session-oriented immersive media. Multimedia Tools and Applications, 1-40.
13. Lertpongrujikorn, P., & Salehi, M. A. (2023, July). Object as a service (oaas): Enabling object abstraction in serverless clouds. In 2023 IEEE 16th International Conference on Cloud Computing (CLOUD) (pp. 238-248). IEEE.
14. Ferreira, J. P. R. D. C. (2022). Reactive Microservices-An Experiment (Doctoral dissertation).
15. Félix, r. c. (2023). methodology for recommendation of microservices architectures.
16. Cattermole, A. D. D. (2022). Run-time adaptation of a functional stream processing system (Doctoral dissertation, Newcastle University).
17. Pallewatta, S., Kostakos, V., & Buyya, R. (2024). Microfog: a framework for scalable placement of microservices-based iot applications in federated fog environments. Journal of Systems and Software, 209, 111910.
18. Manco, M. (2022). Cloud Native Security in Service Mesh (Doctoral dissertation, Politecnico di Torino).
19. Buscema, G. (2021). Security orchestration in Kubernetes with Verefoo (Doctoral dissertation, Politecnico di Torino).
20. Neves, S. S. (2024). MESH MICROSERVICES ON KUBERNETES

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details