A Review on Ensuring Robust Security Posture: BestPractices for Securing Serverless Architectures in Devops Pipelines

Veeresh Akki [1], Vehana M Naik [2], V. Bhavani Krishna [3]
Mahesh Kini [Associate Professor]
Department of Computer Science and Engineering.
Visvesvaraya Technological University.
Alvas Institute of Engineering and Technology.

[1] veereshveeru676@gmail.com

[3] vbk1606@gmail.com

Moodabidri, Dakshina Kannada, Karnataka, India

***Abstract*—Ensuring Robust Security Posture explores the convergence of DevOps and serverless architectures, emphasizing crucial security practices. It highlights the advantages of serverless, including enhanced agility and cost-efficiency. The article advocates for the implementation of the zero-trust security model, secure deployment practices, and comprehensive incident response strategies. Addressing challenges like inadequate access controls and cold start attacks, it promotes best practices such as the principle of least privilege and continuous monitoring. Real-world case studies of financial services and e-commerce applications underscore the efficacy of these security measures. This concise guide is essential for practitioners navigating the dynamic landscape of DevOps and serverless architectures, ensuring an integrated approach to security in the ever-evolving realm of software development and deployment.**

***Keywords*— Serverless Security, Zero Trust Model, Access Controls, Incident Response, Dependency Scanning, Throttling and Rate Limiting, Secure Deployment Practices.**

I. INTRODUCTION

A. *DevOps in Modern IT*

DevOps, a fusion of "Development" and "Operations," plays a pivotal role in the contemporary IT landscape, serving as a bridge between software development and operational processes. This section explores the significance of DevOps in facilitating the rapid and seamless development and deployment of high-quality software. DevOps functions as a conduit, facilitating the integration of development and operational aspects. It expedites the delivery of software by streamlining communication and collaboration between teams.

- *DevOps Practices Overview:* Continuous Integration / Continuous Delivery / Deployment (CI/CD) Continuous integration involves merging code updates into the existing codebase, facilitating swift conflict resolution. This practice enhances deployment performance. IaC utilizes code scripts to manage infrastructure, ensuring consistent and reproducible environment management. DevOps embraces Agile methodologies, employing rapid feedback loops and iterative development to adapt to evolving requirements.
- *Essential Tools in DevOps:* DevOps leverages various tools to enhance development and deployment processes. CI/CD Platforms: Travis CI, Jenkins. Monitoring and Logging Tools: ELK Stack, Prometheus. Containerization Tools: Docker, Kubernetes.

B. *Serverless Architecture*

Serverless architecture, despite its name, does not eliminate the need for servers. This section delves into the principles of serverless architecture, highlighting key components and distinguishing features.

- *Definition and Misconceptions:* Serverless architecture is a software design approach where applications are built and operated without directly managing the underlying infrastructure. Contrary to common misconceptions, servers are still integral; however, cloud providers handle related tasks.[1] [2]
- *Components of Serverless Architecture*: The Serverless applications are composed of individual functions, each designed to perform specific tasks. Serverless Platforms like Leading cloud providers, such as AWS and Microsoft Azure, offer serverless platforms for hosting and executing functions. Event Triggers like Serverless functions are triggered by specific events, such as changes in data or incoming messages.
- *Contrasting Monolithic Architecture:* This *are* subsection explores the traditional monolithic architecture, emphasizing its unified software design model. Key Components Monolithic architecture encompasses authorization, presentation, business logic, database layer, and application integration. [2] [3]
- *Benefits of Serverless Architecture over Monolithic:* Highlighting the advantages of serverless architecture in comparison to monolithic counterparts.
- *Agility and Development:* Rapid Deployment: Serverless enables quick and iterative development cycles, facilitating faster feature releases. Developer Efficiency: Focus on code and business logic, while the cloud provider manages infrastructure maintenance.
- *Cost-efficiency:* Reduced Operating Costs is Elimination of server patching, maintenance, upgrades. The Pay-per-Use Model is Payment based on resource usage, avoiding costs for idle resources.
- *Improved Monitoring and Troubleshooting:* The Comprehensive Logs Serverless systems offer detailed logs and metrics for each function. Enhanced Debugging Simplified monitoring and debugging through efficient log data.
- *Drawbacks of Serverless Architecture: A* Acknowledging challenges, such as a vendor lock-in and cold start latency, associated with serverless architecture.

II. LITERATURE REVIEW

Focuses on Service Function Chaining (SFC) and its potential to

enhance service chain provisioning (SFC leverages Network Functions Virtualization (NFV) and Software-Defined Networking (SDN). Security is a major concern for widespread SFC adoption due to increased attack surface. Comprehensive analysis of SFC architecture, design principles, and relationships with NFV and SDN. Highlighting significant enhancements achieved by adopting SFC, with deployment examples. Layer-specific threat taxonomy analysis based on SFC layering model. Evaluation of existing defensive solutions and proposed security recommendations. Aims to assist network operators in deploying cost-effective security measures based on specific requirements.[4] [5]

Addresses security considerations unique to serverless architectures. Emphasizes the importance of protecting application logic in serverless applications. Identification of common attack vectors and risks associated with misconfigurations. Discussion on the role of consumers in ensuring serverless application protection. Strategies for protecting serverless applications against vulnerabilities and attacks. Linux containers present a lightweight solution to package applications into images and instantiate them in isolated environments. Such images may include vulnerabilities that can be exploited at runtime. A vulnerability scanning service can detect these vulnerabilities by periodically scanning the containers and their images for potential threats. When a threat is detected, an event may be generated to quarantine or terminate the compromised container(s) and optionally remedy the vulnerability by rebuilding a secure image. We believe that such an event-driven process is a great fit to be implemented in a serverless architecture. In this paper we explore the design of an automated threat mitigation architecture based on OpenWhisk and Kubernetes. [5] [6] [7]

A. Zero Trust Security Model

The Zero-Trust Security model assumes that no user or device is inherently trustworthy. This approach requires all users and devices to be authenticated and authorized before accessing resources, regardless of their origin within the network. Implementing a zero-trust approach in serverless environments strengthens security by minimizing the attack surface.

B. Secure Deployment Practices

Security checks shouldn't be a one-time activity. Integrate security checks into the CI/CD pipeline to identify vulnerabilities before deployment. This can involve automated vulnerability scanning, configuration validation, and security policy enforcement. By automating these checks, you can ensure consistent security practices across all deployments[8].

C. Incident Response and Recovery

No security strategy is foolproof. Developing a comprehensive incident response plan is crucial for responding effectively to security incidents. This plan should outline procedures for finding, holding, and recovering from security breaches. Regularly conducting drills ensures team preparedness and minimizes potential damage during a real-world incident.

D. Dependency Scanning

Third-party dependencies can introduce vulnerabilities into your serverless applications. Regularly scan these dependencies for known vulnerabilities and update them promptly to support a secure codebase. Utilize automated dependency scanning tools within the CI/CD pipeline to streamline this process.

E. Throttling and Rate Limiting

Implement mechanisms to prevent DoS attacks by limiting the number of requests a user or service can make within a period. This can be achieved through throttling and rate limiting techniques. These measures safeguard serverless functions from being overwhelmed by malicious activity. [9]

F. Serverless-Specific Security Tools

Many cloud providers offer serverless-specific security tools designed to provide deeper visibility and control over serverless environments. These tools can help you monitor function execution, identify security risks, and manage access controls. Utilizing these tools alongside general security best practices can further enhance your serverless security posture. [10][11]

III. SECURITY CHALLENGES IN SERVERLESS ARCHITECTURES

A. Inadequate Access Controls

Challenges: Sometimes, it is hard to control who can use serverless things and what they can do best practices are

- Implement fine-grained access controls based on the principle of least privilege.
- Regularly review and update access policies to align with changing requirements.
- Utilize identity and access management (IAM) tools for robust access control.

B. Data Security and Encryption

Challenges: We need to make sure that important data is kept secret and safe when it is moving around the best practices are

- Implement strong encryption protocols for data in transit and at rest.
- Regularly audit data access logs and employ anomaly detection for suspicious activities.
- Utilize secure key management practices to safeguard encryption keys. [12] [13]

C. Limited Visibility and Monitoring

Challenge: It can be hard to see what's happening with serverless things and the best practices are

- Implement comprehensive monitoring solutions that provide real-time insights into serverless functions.
- Utilize logging and analytics tools to identify and troubleshoot issues promptly.
- Establish automated alerting systems for immediate response to anomalous activities.

D. Cold Start Attacks

Challenge: Sometimes, when we start things up, they can be slow and not safe and the best practices are

- Optimize code and configurations to minimize cold start times.
- Implement warming strategies, such as scheduled executions, to reduce latency.
- Regularly test and monitor cold start behaviours to address potential security risks. [14]

E. Dependency Security Risks

Challenge: Sometimes, we use things that others made, and they might not be safe and the best practices are

- Regularly scan and update third-party dependencies to address known vulnerabilities.
- Implement a secure software supply chain with thorough vetting of external components.
- Utilize automated dependency scanning tools within the CI/CD pipeline. [15] [16]

F. Stateless Execution Challenges

Challenge: Serverless things don't remember what happened before, and that can be a problem and the best practices are

- Implement secure external storage for persistent data needs.
- Leverage state management solutions to handle necessary information between function executions.
- Regularly review stateless design choices for security implications.

G. Denial-of-Service (DoS) Attacks

Challenge: Some people might try to make serverless things stop working by using them too much and the best practices are

- Implement throttling and rate-limiting mechanisms to mitigate excessive requests.
- Utilize traffic shaping techniques to differentiate legitimate from malicious traffic.
- Employ automated scaling and resource provisioning to handle sudden spikes in demand. [17] [18]

H. Secure Deployment Practices

Challenge: Putting new things into serverless can be risky if we don't do it right and the best practices are

- Integrate security checks into the CI/CD pipeline to identify vulnerabilities before deployment.
- Implement automated testing for configuration validation and adherence to security policies.
- Regularly update and patch serverless components to address emerging security threats.

IV. BEST PRACTICES FOR SECURING SERVERLESS ARCHITECTURES

A. Principle of Least Privilege (PoLP)

Best Practice: Only let people do what they need to do, and check this regularly.

- Enforce the principle of least privilege (PoLP) to ensure that users and functions have only the minimum permissions necessary.
- Regularly audit and update access policies to align with changing roles and responsibilities.
- Implement role-based access controls (RBAC) to assign permissions based on job functions.

B. Secure Configuration Management

Best Practice: Keep vital information safe and change it regularly.

- Utilize secure configuration management practices to safeguard sensitive information.
- Regularly review and update configurations to address evolving security requirements.
- Employ configuration validation tools to ensure adherence to security policies.

C. Data Encryption

Best Practice: Make sure important data is safe when it is moving and when it is stored.

- Implement robust encryption protocols for data in transit and at rest.
- Utilize secure key management practices to protect encryption keys from unauthorized access.
- Regularly audit and monitor data access logs to detect and respond to potential security incidents.

D. Continuous Monitoring and Logging

Best Practice: Use good tools to keep an eye on what is happening and fix things quickly.

- Implement comprehensive monitoring solutions to track the behaviour of serverless functions.
- Utilize logging and analytics tools to collect and analyse relevant security data.
- Establish automated alerting systems to promptly respond to security incidents.

E. Access Controls and Authentication

Best Practice: Make sure only the right people can use things and check this a lot.

- Implement strong access controls and authentication mechanisms to verify user identities.
- Utilize multi-factor authentication (MFA) for an additional layer of user verification.
- Regularly review and update access controls to align with changing security requirements.

F. Automated Security Testing

Best Practice: Use tools that check for problems in the software automatically.

- Integrate automated security testing tools into the CI/CD pipeline to identify vulnerabilities.
- Conduct regular security assessments, including static and dynamic code analysis.
- Implement automated testing for security policy adherence and compliance. [19]

G. Zero Trust Security Model

Best Practice: Always check if people are allowed to use things, even if they are inside the company.

- Embrace the zero-trust security model to verify the identity and authorization of all users.
- Implement continuous authentication and authorization mechanisms to monitor user activities.
- Regularly review and update access policies to align with the zero-trust principles.

H. Secure Deployment Practices

Best Practice: Make sure new things are checked for safety before we put them in.

- Integrate security checks into the CI/CD pipeline to identify vulnerabilities before deployment.
- Implement automated testing for configuration validation and adherence to security policies.
- Regularly update and patch serverless components to address emerging security threats. [19] [20] [21]

I. Incident Response and Recovery

Best Practice: Have a plan for when things go wrong and practice it a lot.

- Develop a comprehensive incident response plan outlining procedures for security incidents.
- Conduct regular drills and simulations to ensure team readiness and effectiveness.
- Establish communication protocols and escalation procedures for timely incident resolution.

J. Dependency Scanning

Best Practice: Check if things we use are safe and update them a lot.

- Regularly scan and update third-party dependencies to address known vulnerabilities.
- Utilize automated dependency scanning tools within the CI/CD pipeline for efficiency.
- Maintain an inventory of dependencies and track their security status continuously. [22] [23]

K. Throttling and Rate Limiting

Best Practice: Make sure only the right number of people can use things and stop dreadful things from happening.

- Implement throttling and rate-limiting mechanisms to mitigate denial-of-service (DoS) attacks.
- Utilize traffic shaping techniques to differentiate between legitimate and malicious traffic.
- Regularly review and adjust throttling parameters based on evolving usage patterns.

L. Serverless-Specific Security Tools

Best Practice: Use tools made for serverless things that help us see problems and fix them.

- Leverage serverless-specific security tools provided by cloud providers for enhanced visibility.
- Utilize tools for monitoring function execution, identifying security risks, and managing access controls.
- Stay informed about updates and advancements in serverless security tools for continuous improvement.

V. STATISTICAL ANALYSIS

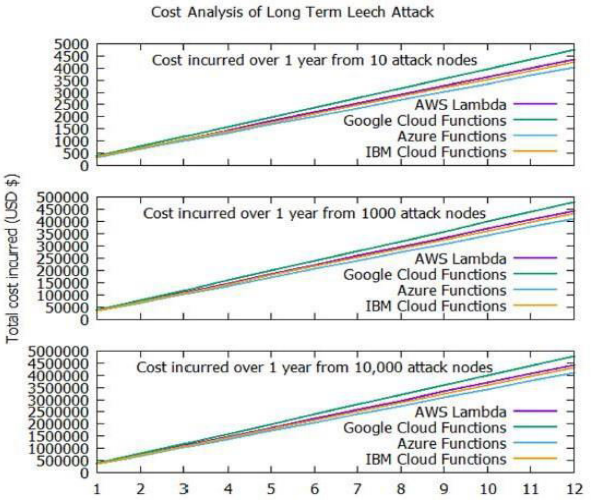

Fig. 1 Cost Incurred When an Increasing Number of Nodes Send 2000 Requests Every Hour.[31]

In fig. 1.,A cost analysis table compares the expense of running a year-long leach attack using different cloud function providers. The table shows how the cost scales based on the number of nodes involved in the attack, ranging from 10 to 10,000. Among the providers listed, AWS Lambda is the most economical, followed by Google Cloud Functions, Azure Functions, and finally IBM Cloud Functions. The cost increases significantly as the number of nodes grows. Notably, the table doesn't display the cost for AWS Lambda at higher node counts, but it's likely to be more than $5,000. [31]

TABLE I.

COLD START DURATION OF DIFFERENT PLATFORMS

Programming Language	Aws (Seconds)	Gcp (Seconds)	Azure (Seconds)
JavaScript	0.2-0.4	0.4-0.6	-
Python	0.2-0.25	0.2-0.25	0.45
Go	0.3-0.4	0.35	-
C#	-	-	0.35-0.5

TABLE II.

DEVELOPERS IN CLOUD COMPUTING

Category	Active Developers (Q1 2021)	Change from Q1 2020
Cloud Native	6.8 million	- (not specified in the information provided)
Containers	4.6 million	+0.7 million
Container Orchestration Tools & Management Platforms	4.0 million	+0.1 million
Cloud Functions or Serverless Architecture	4.0 million	+0.1 million

In Table 1. We can see that the cold start duration varies depending on the programming language and cloud platform. Generally, languages that are interpreted (like JavaScript and Python) have faster cold start times than languages that are compiled (like Java and C#). This is because interpreted languages don't need to be compiled before they can be run.

The cloud platform can also affect cold start duration. For example, Azure functions typically have longer cold start times than AWS Lambda functions. This is likely due to differences in the way that the two platforms provision resources.

In Table 2. Cloud Native - This section likely refers to the total number of cloud native developers, which is 6.8 million. Containers - This section showcases the number of active developers using containers, which is around 4.6 million in Q1 2021. There's a gradual increase from 3.9 million in Q1 2020. Container Orchestration Tools & Management Platforms - This section shows the number of active developers using container orchestration tools and management platforms. There are 4 million developers in Q1 2021, which has grown from 3.9 million in Q1 2020. Cloud Functions or Serverless Architecture - This section highlights the number of active developers using cloud functions or serverless architecture. There are 4.0 million developers in Q1 2021, which has grown from 3.9 million in Q1 2020.

VI. CONCLUSIONS

In conclusion, this review paper has explored the multifaceted landscape of securing serverless architectures within DevOps pipelines. Through a comprehensive analysis of existing literature, key security challenges in serverless environments have been identified, ranging from inadequate access controls to cold start attacks. By synthesizing best practices and strategies advocated by scholars and practitioners, this paper has outlined a robust framework for ensuring a resilient security posture in serverless deployments. The significance of implementing a zero-trust security model, secure deployment practices, and incident response strategies has been emphasized to mitigate potential risks and vulnerabilities. Additionally, the importance of continuous monitoring, dependency scanning, and throttling mechanisms in safeguarding serverless functions from malicious activities has been underscored. Real-world case studies and statistical analysis have provided empirical evidence of the effectiveness of these security measures in various scenarios, further validating their relevance and applicability. Looking ahead, the implications of this review extend beyond academic discourse to practical implementation in industry settings. As organizations increasingly adopt serverless architectures to drive innovation and agility, it becomes imperative to prioritize security measures throughout the software development lifecycle. By adhering to the principles and recommendations outlined in this paper, stakeholders can enhance their security posture and better protect their assets and data from evolving threats in the dynamic landscape of modern IT. In essence, this review serves as a comprehensive guide for practitioners, offering actionable insights and best practices for securing serverless architectures within DevOps pipelines. By integrating these recommendations into their development and deployment processes, organizations can navigate the complexities of serverless security with confidence, ensuring the integrity, availability, and confidentiality of their applications and infrastructure.

ACKNOWLEDGMENT

I am grateful to Dr. G. Srinivasan for their unwavering support and invaluable insights. His expertise and guidance have been critical in refining the research design, analysing data, and interpreting findings. His constructive feedback and

intellectual discussions have truly enriched this study.
I am deeply grateful to my mentor, Mr. Mahesh Kini, for their exceptional guidance and unwavering support throughout this research endeavour. Their expertise, insightful feedback, and continuous encouragement have been invaluable in shaping the direction and outcomes of this study. Their unwavering commitment to my academic growth and professional development has been truly inspiring.
I would like to express my heartfelt gratitude to the faculty members and academic advisors who have provided guidance, feedback, and support throughout my academic journey. Their expertise, wisdom, and dedication to teaching and mentoring have been instrumental in shaping my research skills and scholarly pursuits.

REFERENCES

[1] Montida Pattaranantakul, Chalee Vorakulpipat, Takeshi Takahashi, "Service Function Chaining security survey: Addressing security challenges and threats", Computer Networks,Volume 221,2023, https://www.sciencedirect.com/science/article/pii/S1389128622005187

[2] W. O'Meara and R. G. Lennon, "Serverless Computing Security: Protecting Application Logic," 2020 31st Irish Signals and Systems Conference (ISSC), Letterkenny, Ireland, 2020, pp. 1-5.

[3] N. Bila, P. Dettori, A. Kanso, Y. Watanabe and A. Youssef, "Leveraging the Serverless Architecture for Securing Linux Containers," 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 2017, pp. 401-404.

[4] N. Mateus-Coelho and M. Cruz-Cunha, "Serverless Service Architectures and Security Minimals," 2022 10th International Symposium on Digital Forensics and Security (ISDFS), Istanbul, Turkey, 2022, pp. 1-6.

[5] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill and R. Buyya, "iFaaSBus: A Security- and Privacy-Based Lightweight Framework for Serverless Computing Using IoT and Machine Learning," in IEEE Transactions on Industrial Informatics, vol. 18, no. 5, pp. 3522-3529, May 2022.

[6] X. Li, X. Leng and Y. Chen, "Securing Serverless Computing: Challenges, Solutions, and Opportunities," in IEEE Network, vol. 37, no. 2, pp. 166-173, March/April 2023.

[7] Ahmed Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, Antonio Filieri, "Quality-Aware DevOps Research: Where Do We Stand?", in IEEE Access ,Vol. 9, pp. 44476-44489, March 2021.

[8] Trend Micro. (Oct.24, 2019). Trend Micro Security News. "The Cloud: What it is and what it's for." Accessed on May 25, 2020, at https://www.trendmicro.com/vinfo/us/security/news/security-technology/the-cloud-what-it-is-and-what-it-s-for.

[9] Amazon Web Services. (March 22, 2019). YouTube. "Build a Serverless Startup in Just 30 Minutes!" Accessed on May 25, 2020, at https://www.youtube.com/watch?v=qBNYmYRlTpU.

[10] Mark Nunnikhoven. (Oct. 22, 2019). Trend Micro Simply Security. "The Shared Responsibility Model." Accessed on May 25, 2020, at https://blog.trendmicro.com/the-shared-responsibility-model/.

[11] AWS. (n.d.). AWS. "The Shared Responsibility Model." Accessed on May 25, 2020, at https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/the-shared-responsibility-model.html.

[12] Security Best Practices in IAM. (n.d.). AWS. "Security Best Practices in IAM." Accessed on May 25, 2020, at https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege.

[13] Chris Munns. (July 23, 2018). AWS Compute Blog. "Powering HIPAA-compliant workloads using AWS Serverless technologies." Accessed on June 8, 2020, at https://aws.amazon.com/blogs/compute/powering-hipaa-compliant-workloads-using-aws-serverlesstechnologies/.

[14] Ihor Lobastov. (March 8, 2019). DZone. "Comparing Serverless Architecture Providers: AWS, Azure, Google, IBM, and Other FaaS Vendors." Accessed on June 3, 2020, at https://dzone.com/articles/comparing-serverless-architecture-providers-aws-az.

[15] David Ramel. (May 8, 2020). Virtualization and Cloud Review. "Cloud-Native Development Survey Details Kubernetes, Serverless Data." Accessed on May 25, 2020, at https://virtualizationreview.com/articles/2020/05/08/cloud-native-dev-survey.aspx.

[16] AWS. (n.d.). AWS. "Amazon S3." Accessed on May 25, 2020, at https://aws.amazon.com/s3/.

[17] AWS. (n.d.). AWS. "Amazon Lambda." Accessed on May 25, 2020, at https://aws.amazon.com/lambda/.

[18] Amazon API Gateway. (n.d.). AWS. "Amazon API Gateway." Accessed on May 25, 2020, at https://aws.amazon.com/api-gateway/.

[19] AWS (n.d.). AWS. "IAM Roles." Accessed on Aug. 11, 2020, at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html

[20] Derek Belt. (May 15, 2018). AWS Partner Network (APN) Blog. "The 5 Pillars of the AWS Well-Architected Framework." Accessed on July 23, 2020, at https://aws.amazon.com/blogs/apn/the-5-pillars-of-the-aws-well-architected-framework/#:~:text=The%20AWS%20Well%2DArchitected%20Framework%20provides%20architectural%20best%20practices%20across,an%20existing%20or%20/

[21] Montida Pattaranantakul, Chalee Vorakulpipat, Takeshi Takahashi,Service FunctionChaining security survey: Addressing securitychallengesandthreats,ComputerNetworks,Volume221,2023,109484,ISSN13891286, https://doi.org/10.1016/j.comnet.2022.109484. https://www.sciencedirect.com/science/article/pii/S1389128622005187

[22] 2023 REPORT CLOUD SECURITY, https://resources.trendmicro.com/rs/945-CXD-062/images/2023-Cloud-Security-Report-TrendMicro-Final.pdf

[23] Global Cybersecurity Outlook 2023 INSIGHT REPORT JANUARY 2023, https://www3.weforum.org/docs/WEF_Global_Security_Outlook_Report_2023.pdf

[24] K. Djemame, M. Parker, D. Datsev, Open-source serverless architectures: An evaluation of Apache OpenWhisk, in: 2020 IEEE/ACM 13th International

[25] Conference on Utility and Cloud Computing, UCC, 2020, pp. 329–335, http://dx.doi.org/10.1109/UCC48980.2020.00052.

[26] N. Kaviani, D. Kalinin, M. Maximilien, Towards serverless as commodity: A case of knative, in: Proceedings of the 5th International Workshop on Serverless

[27] Computing, WOSC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 13–18, http://dx.doi.org/10.1145/3366623.3368135.

[28] D. Balla, M. Maliosz, C. Simon, Open source FaaS performance aspects, in: 2020 43rd International Conference on Telecommunications and Signal Processing, TSP, 2020, pp. 358–364, http://dx.doi.org/10.1109/TSP49548.2020.9163456.

[29] S.K. Mohanty, G. Premsankar, M. Di Francesco, An evaluation of open source serverless computing frameworks, in: Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, Vol. 2018-Decem, IEEE Computer Society, 2018, pp. 115–120, http://dx.doi.org/10.1109/CloudCom2018.2018.00033.

[30] P. Garcia Lopez, M. Sanchez-Artigas, G. Paris, D. Barcelona Pons, A. Ruiz Ollobarren, D. Arroyo Pinto, Comparison of FaaS orchestration systems, in: Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, 2019, pp. 109–114, http://dx.doi.org/10.1109/UCC-Companion.2018.00049, arXiv:1807.11248.

[31] Daniel Kelly, Frank G. Glavin, Enda Barrett, Denial of wallet—Defining a looming threat to serverless computing, Journal of Information Security and Applications,Volume 60,2021,102843,ISSN 2214-2126, https://doi.org/10.1016/j.jisa.2021.102843. (https://www.sciencedirect.com/science/article/pii/S221421262100079X)