International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *05 November 2025* ***Revised:*** *30 November 2025* ***Accepted:*** *13 December 2025*

Towards a Unified Framework for Serverless Microservices in Cloud-Native Environments

Hasini Koka[1]

Full-stack Developer, Cleveland, Ohio, USA

Hasiniraokoka@gmail.com

Gopinath Karunanithi[2]

Software Developer, Edison, NJ, USA

gopinathtts@gmail.com

Rajendran Renganathan[3]

Senior Manager, Frisco, Texas, USA

rajendran_13@hotmail.com

Chittaranjan Pradhan[4]

Independent Researcher, East Brunswick, NJ, USA

cpradhan01@gmail.com

Amar Singh[5]

Platform Engineering Tech Lead, Jersey City, NJ, USA

amar.d.singh121@gmail.com

Abstract:

With cloud-native computing gaining popularity, people are using microservices and serverless more, since both help create applications that can grow, be divided into parts, and remain steady. Nevertheless, these approaches also bring some limitations—microservices make managing systems hard while serverless functions have to cope with delays, no store of state, and invisible errors. Therefore, this paper suggests a common architecture that combines serverless and microservices under Kubernetes, Knative, and Istio in a cloud setting. Because of this approach, any workload can be smoothly designed, adjusted for any demand, and regularly monitored on any type of cloud environment. This study describes a system made up of Serverless-Microservices Tier, an Abstraction Layer for control, and a Cloud-Native Platform Layer for handling management and execution. It is confirmed in real-world settings, mainly in e-commerce, healthcare, and IoT, that it is flexible and can run smoothly. If hybrid deployment is compared to using only one of the models, the unified use of control planes offers more advantages. It also describes future developments, such as using AI for orchestration, including edge computing, and operating at the declarative level, so the proposed approach stands out as an up-to-date method for the next-gen of cloud-native systems.

Keywords: Cloud, Serverless Microservices, Cloud Computing, Kubernetes, Cloud-Native Environment

1. Introduction

Over the past few years, several important changes came to software architecture thanks to cloud-native technologies, serverless computing, and microservices. All these changes reflect a big break from old systems, making modern app development more scalable, robust, and flexible. Because developers use serverless computing, they do not need to handle the infrastructure, which helps speed up the creation and release of applications and saves effort in operation (Baldini et al., 2017). At the same time, microservices divide applications into separate services, making their use more flexible and making it possible to develop, test, and deploy each module on its own (Dragoni et al., 2017). Remote application execution and cloud-native methods

make sure the combination brings exceptional scalability and enhanced efficiency.

Even when used individually, serverless and microservices bring certain problems when combined in cloud-native environments. Some of these are the difficulty in arranging services, handling state, extra time needed to process requests, and security risks associated with performing tasks right down to the small unit. Furthermore, due to the lack of common strategies to bring together serverless and microservice-based architectures, deployments tend to be different for each company, lock people into using one provider, and make the systems more complicated (Jonas et al., 2019). Most current solutions concentrate on one approach at a time, missing the chance to use their features together.

It suggests a framework that smoothly joins serverless computing and microservices in one cloud-native arrangement. The suggested framework is made up of Serverless Microservices, an Abstraction Layer, and the Cloud-Native Environment. Such an approach gathers all the significant aspects like service discovery, resource control, and function arranging; it also supports smooth integration between different cloud providers. In particular, the abstraction layer is the main coordinator, keeping the application logic and the services offered by the cloud apart.

It is meant to allow for changing workloads, make operations simpler, improve resource use, and maintain the flexibility and resistance to failure found in cloud solutions. The cloud technology Kubernetes, API gateways, and AWS Lambda or Azure Functions make the framework capable of working in different hybrid and multi-cloud settings (Mohan et al., 2020).

All in all, this research sets out to join serverless with microservices using a well-built framework that encompasses various vendors. Certainly, it helps cloud computing by combining existing cloud-native ways of working with fast and automated methods, promoting quick processing and straightforward administration.

2. Literature Review

The rise of cloud-native ideas has made a big difference in the development and deployment of software applications. Microservices and serverless computing are important to this transformation since they help developers create flexible applications. It is possible to divide applications into separate, deployable units through microservices architectures, and they then communicate using REST or gRPC (Newman, 2015). As a result, the model makes updating, increasing scale, and rolling out individual services simpler. Still, there are technical challenges involved in managing microservices, for example, service coordination, messages transferring between them, and handling each service's state (Pahl & Jamshidi, 2016).

Alternatively, developers don't have to worry about complexity in serverless computing since they only need to think about the actual code with no infrastructure management required. AWS Lambda, Azure Functions, and Google Cloud Functions adjust their resources on their own and take payment only for what is used (McGrath & Brenner, 2017). Even though serverless solutions are easy to use, they also require all data to be stored externally and limit execution time. For this reason, RPC is mostly useful in applications that run for a short time or that work independently (Hellerstein et al., 2018).

Various experts have tried to resolve the issues that microservices and serverless services deal with on their own by combining them in a hybrid fashion. For example, according to Spillner et al.'s (2019) approach, serverless functions and microservices are controlled with service meshes and container runtimes. In the same way, Jonas et al. (2019) stress how using abstraction layers improves the unity and consistency between programming and runtime components in distributed systems. Using these solutions produces good outcomes, yet they do not always work well when companies integrate multi-cloud and hybrid systems.

Research by Toffetti et al. (2015) introduces advanced orchestration platforms like Kubernetes and Docker Swarm for managing microservices at scale. However, this method requires that the system is always running, which means it fails to meet the high standards of serverless computing in event-based situations. On the other hand, Knative makes Kubernetes serverless by providing autoscaling, eventing, and a service-level interface that help mix microservices and serverless concepts (Anwar et al., 2020). Even so, some problems with adoption continue due to how difficult the processes are, as standards are missing, and running these operations demands a lot.

Although these developments are positive, still, a unified way to handle both serverless functions and microservices under the same control needs to be found. The current research is trying to solve this issue by introducing a framework that smoothly supports the use of both microservice-oriented and legacy-based systems.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *05 November 2025* ***Revised:*** *30 November 2025* ***Accepted:*** *13 December 2025*

This is particularly important these days, when applications have to ensure both flexibility and performance, as well as keep useful modules simple, and their expenses traceable.

3. Methodology

The structure of the unified framework is made to be both modular and extendable in order to follow cloud-native ideas is shown in figure 1. The developers should first have a standardized place to run and deploy their functions and services. Being based on Kubernetes, Knative is the main runtime for this framework and allows developers to run stateless functions that are triggered by HTTP or event calls. It manages the intricacies of setting up the service, and supports autoscaling, coordination of many tasks, and processing of CloudEvents.

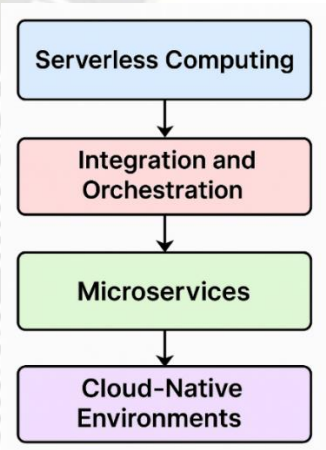

Fig. 1. Proposed Methodology

To manage communication and service discovery well, this framework makes use of Istio-based service mesh. Therefore, developers can establish how communication should work, introduce security rules, and observe communication among services without entering the application code. Istio makes it possible to use both REST and gRPC so the framework can handle all sorts of messaging from synchronous to asynchronous communication. Flexibility matters a lot for hybrid workloads since different applications require different levels of performance and speed.

It is very hard to observe distributed systems, especially serverless environments where things are always changing rapidly. It achieves this by having Prometheus for tracking metrics, Grafana to see the results, and Jaeger for following services within a distributed trace. They help see exactly how functions are called, how long requests take, and how many resources are used. The system has all logs and metrics in one location so that difficulties can be identified and corrected quickly.

Security duties and setting policies are managed both by capabilities integrated into Kubernetes and by the service mesh. With RBAC, access to resources is given only to those who have the proper authorization. All communication between services is protected with mTLS, and OAuth makes it possible to add external identity providers. Because of these mechanisms, security is always the same for serverless services and microservices.

In the end, GitOps and CI/CD pipelines ensure that lifecycle management is set up the same way each time. Both ArgoCD and Tekton are now utilized to enable automatic deployment and rollback. It provides a good experience for developers and also keeps the process of deploying software dependable and constant. Through these five main modules, the approach presents a full plan for combining serverless microservices.

4. Proposed Architecture

In a unified architecture, event-driven functions work in addition to regular containerized microservices as shown in figure 2. At this step, API Gateway or Kubernetes Ingress controller is used to control the flow of external traffic. Both functions and services can be safely accessed flexibly because requests are guided by their URL paths or by the events they are linked to. It also has capabilities that limit attackers' potential actions and check the authenticity of incoming requests.

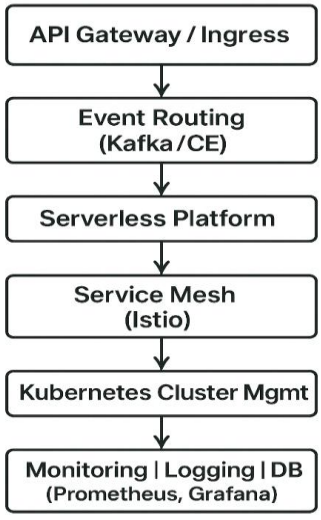

Fig. 2. Proposed architecture for Unified Framework for Serverless Microservices in Cloud-Native Environments

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *05 November 2025* ***Revised:*** *30 November 2025* ***Accepted:*** *13 December 2025*

Information from events is sent through Kafka or Knative Eventing, and these tools direct it to the appropriate functions. This way, event producers do not depend on consumers and can use several protocols such as HTTP, gRPC, and CloudEvents. The layer is also responsible for transforming and filtering messages to accommodate processing data in many different formats instantly.

Two types make up the core compute layer: Knative takes care of the serverless plane, while Kubernetes controls the other containerized microservices plane. Scaling Knative is easy when there is extra HTTP traffic, but Kubernetes takes care of persisting information for microservices using a database or state. Both of them are placed inside the same cluster to allow sharing of resources and ensure consistency in how they run.

The Istio service mesh layer guarantees that all communication is secure and functional between different parts. It puts a sidecar proxy in every pod automatically, allowing functions like divide traffic, test new and existing options, avoid failure, and observe performance. This adds useful services during program execution, all without altering the application logic, which makes it acceptable for polyglot microservices.

The last layer in the infrastructure is for monitoring and logging, where Prometheus, Jaeger, Fluentd, and Grafana are used for metric scraping, tracing, log collection, and visualization. Data is redacted and stored with the aid of either cloud-based databases or volume plugins. All of these parts create a reliable, sizable, and secure setup for launching microservices in the cloud.

5. Benefits and Challenges

The unified framework that is proposed brings a number of useful benefits. First thing first, it is important that the system is portable by using standard and container-compatible tools, allowing the deployment to basic cloud providers or Kubernetes clusters. It removes the problem of vendor lock-in and allows for more options in managing infrastructure. Also, because Knative and Kubernetes are used together, deployment, monitoring, and scaling are always consistent.

Another valuable thing about recycling is how cost-efficient it is. When not being used, serverless functions in Knative are scaled to nothing, which greatly lessens costs on wasted resources. Actually, classic microservices usually need a minimal amount of resources, even when traffic is low. With the hybrid model, companies can schedule less critical services to serverless runtimes and put important or complicated tasks in containers so they can strike a good balance.

The framework improves how developers work by freeing them from infrastructure troubles and easier processes for rolling out applications. After using CI/CD tools and GitOps, developers can easily deploy codes without much risk and repeat the process as often as required. Moreover, the use of single tools to observe the whole system helps lower the time it takes to fix issues in production.

Still, there are some issues that the framework encounters. The biggest drawback is that cold starts cause the latency at the first time a serverless function is run, because the container must be started first. Although caching and having warmed pools can assist, these types of applications are still impacted by this factor. There is an additional difficulty in fixing bugs in distributed systems, especially when some services last for a moment or work asynchronously.

An additional challenge is that getting used to Kubernetes, Istio, or Knative may be tough for teams that are new to them. In environments that are multi-tenant, there is a possibility of drift in the configuration and misconfigurations that threaten security. Thus, although the framework promises long-term success, properly adopting it requires good governance and experts in DevOps.

6. Comparative Analysis

It is necessary to look at traditional serverless, traditional microservices, and the proposed hybrid model and identify their strengths and potential problems in scalability, the process of deploying, observing their operations, and the budget needed. Comprehensive microservices are very configurational and cover lots of long-lasting tasks, but they usually have several operation issues. Being developers, we are responsible for launching containers, managing their lives, and handling discovery of services without any tools. Even though they are powerful, they need experienced DevOps users.

Table 1: Comparative Analysis of Traditional Microservices vs Pure Serverless vs Unified Framework

Feature	Traditional Microservices	Pure Serverless	Unified Framework

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *05 November 2025* ***Revised:*** *30 November 2025* ***Accepted:*** *13 December 2025*

Deployment Model	Container-based	FaaS	Hybrid (FaaS + K8s)
State Management	Externalized DBs	Stateless	Optional with sidecar
Communication	REST/gRPC	Event-driven	Mixed (Event + Sync)
Observability Tools	Prometheus, ELK	Limited	Unified (Grafana + Jaeger)
Scalability	Manual or auto via K8s	Auto-scaling	Enhanced with K8s HPA
DevOps Overhead	High	Low	Moderate

However, serverless functions make things simpler for developers by offering a quick development process and automatic scaling whenever needed. Even so, these services have restrictions—everything must happen outside of the function, they experience slow starts, and lack proper support for running tasks over a long period. It is also hard to keep an eye on and solve problems in applications because the infrastructure is managed remotely by the vendor. The approach is good for event-driven demands, yet by itself it does not meet all application requirements.

The single framework compiles the benefits of both ways of thinking. On the same platform, developers are allowed to create stateful services using containers and stateless ones using serverless functions. Consequently, one can handle latency-sensitive methods by microservices, and keep other tasks and regular jobs in functions. Teams can use deployments that suit their specific workloads instead of facing the restrictions of the architecture.

Moreover, the hybrid way lets you have consistent monitoring and tight security because Istio and Kubernetes manage everything centrally. Contrary to public serverless services, the one framework makes it possible for teams to follow, log, and oversee every service in a similar manner. Because of this transparency, it becomes simpler and quicker to solve problems, more accurate to plan capacity, and to govern the way services function.

When talking about cost, serverless computing helps you save on occasional tasks, whereas frequent or regular tasks are handled by containers that remain constant. The hybrid model lowers costs while delivering good results by using suitable resources only when needed. Hence, the unified framework becomes a useful method for teams to manage their agility, style of working, and performance.

7. Real-World Use Cases

It is especially useful to use this unified framework in e-commerce platforms. Usually, such systems must handle several jobs at the same time, like tracking stock, processing orders, providing personalized suggestions, and sending delayed notifications. With stateless functions, you can manage sending emails as well as acting on webhook signals, and microservices are useful for tasks such as store cart routing, user identification, and displaying products. It allows the workloads to communicate smoothly and change their resources, supporting a fast response and optimizing expenses.

Healthcare applications are also very important. Doctors and experts need to process medical information quickly, protect it safely, and look for useful results. Using serverless functions in the hybrid method, it is possible to process IoT medical device data immediately and in real-time, while the EMR and authentication parts will use microservices that stay constant. With service mesh security, health IT teams are sure that all modules communicate under HIPAA regulations. At the same time, observing tools help them detect any issues in performance or security immediately.

Fraud detection and risk analysis, which must happen quickly, profit from using this kind of architecture. So, you can use functions for processing real-time transactions by means of a Kafka event queue, and backend systems like account management and ledgers can be constructed as microservices. Thanks to the framework, it is possible to keep critical applications separate from experimental ones, which enhances safety during deployment in a domain with many rules.

Transcoding and distributing videos on media and content delivery platforms need elastic computing power. Such demanding applications can run through serverless functions that automatically adjust, whereas user comments and content lists are best managed with tiny microservices. Thanks to the hybrid model, platforms can roll out updates fast without losing track of

providing good service to users and keeping content fresh.

Finally, both smart cities and IoT ecosystems use the unified framework to process plenty of events generated by devices and maintain analysis services over time. Serverless functions read data coming from sensors, and microservices take care of user dashboards, alerting, and setting up devices. Thanks to this structure, the platform can operate well both now and in the future, while being governed centrally and protected against possible faults.

8. Future Trends and Research Directions

One of the best recent trends is that multi-runtime platforms now allow all kinds of computing models, including containers, functions, and Web Assembly, to work under a single control system. KEDA (Kubernetes Event-Driven Autoscaling) and Dapr (Distributed Application Runtime) are gaining greater significance in cloud users' efforts. Thanks to their abstraction layers, including event-driven patterns in microservices-based applications is made much easier.

Orchestration powered by AI/ML is increasingly becoming a valuable direction to follow. Automatic estimation of resources, finding errors, and improving performance are all improved by using machine learning. Various studies are underway to build workload schedulers that check past data and modify infrastructure on the spot. Because of this, cold starts can be eliminated, networks will work faster, and resources are optimized.

Edge computing is developing quickly and shares many points in common with serverless microservices. When businesses edge out their computing to the outermost level, the requirement for light, portable, and fast-event-triggered software rises. Frameworks used in cybersecurity should allow for using cloud, edge, and on-site environments together. Such a system will have to manage new issues in consistency, speed, and handling shared data across multiple machines.

Security and compliance will always be important topics for companies. Increased complexity in putting together different remote systems raises the chances of misconfigurations and breaches in security. Going forward, frameworks are expected to feature zero-trust security, automated auditing, and precise management of policies. They are set to play a key role in the most important applications used in healthcare, finance, and government sectors.

Lastly, nowadays, many platforms are becoming more focused on developers. Improved future frameworks will make things easier for developers by providing simple and useful interfaces. Platforms might turn into systems where developers give the application's purpose and let the platform control the execution. This will bring developers to focus on designing for a good user experience rather than building only infrastructure.

9. Conclusion

The joining of serverless computing and microservices works as a chance to rebuild cloud-native application architecture by making use of what each offers. This research's system makes it easier to tackle the trade-offs that appear when only looking at one method. The adoption of Kubernetes, Knative, and Istio in the framework ensures consistent ways of deploying, observing, scaling applications, and at the same time, enables developers to work freely and provides operations teams with greater reliability. Such an abstraction layer is necessary because it connects both areas, lets cloud-native platforms be more easily adopted, and makes applications more efficient.

In practice, the framework proves to be useful for handling many different types of tasks. Using this strategy, organizations are able to build systems that are secure, flexible, and highly automated. In the future, including intelligent orchestration, features native to the edge, and policies for management will benefit the framework more. In the end, the findings provide a flexible and workable basis for enterprises and developers who want to update their software delivery in line with cloud-native computing.

References

[1]. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Trivedi, R. (2017). *Serverless computing: Current trends and open problems. Research Advances in Cloud Computing*, 1–20. Springer.

[2]. Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2017). *Microservices: How to make your application scale. The Art of Software Architecture*, 95–104.

[3]. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., ... & Stoica, I. (2019). Cloud Programming

Simplified: A Berkeley View on Serverless Computing. arXiv preprint arXiv:1902.03383.

[4]. Mohan, N., Krintz, C., & Wolski, R. (2020). Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. *Proceedings of the 2020 USENIX Annual Technical Conference*.

[5]. Anwar, A., Dürschmid, H., Richter, J., & Schulte, S. (2020). Serverless workflow orchestration for microservices in Kubernetes. *2020 IEEE International Conference on Cloud Engineering (IC2E)*, 41–48.

[6]. Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., & Wu, C. (2018). Serverless computing: One step forward, two steps back. *CIDR*.

[7]. McGrath, G., & Brenner, P. (2017). Serverless computing: Design, implementation, and performance. *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 405–410.

[8]. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

[9]. Pahl, C., & Jamshidi, P. (2016). Microservices: A Systematic Mapping Study. *CLOSER*, 137–146.

[10]. Spillner, J., Mateos, C., & Monge, D. A. (2019). FAASCAP: A Function-as-a-Service Composition Approach. *Future Generation Computer Systems*, 96, 702–710.

[11]. Toffetti, G., Brunner, S., Dudouet, F. X., & Edmonds, A. (2015). An architecture for self-managing microservices. *2015 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 328–333.

[12]. Harve, Bindu Mohan, et al. "The Cloud-Native Revolution: Microservices in a Cloud-Driven World." *2024 International Conference on Intelligent Cybernetics Technology & Applications (ICICyTA)*. IEEE, 2024.

[13]. Weerasinghe, Sidath, and Indika Perera. "Optimized Strategy in Cloud-Native Environment for Inter-Service Communication in Microservices." *International Journal of Online & Biomedical Engineering* 20.1 (2024).

[14]. Shethiya, Aditya S. "Next-Gen Cloud Optimization: Unifying Serverless, Microservices, and Edge Paradigms for Performance and Scalability." Academia Nexus Journal 2.3 (2023).

[15]. Oyeniran, Oyekunle Claudius, et al. "Microservices architecture in cloud-native applications: Design patterns and scalability." *International Journal of Advanced Research and Interdisciplinary Scientific Endeavours* 1.2 (2024): 92-106.

[16]. Surendranath, Neha, Sai Annamaiah Basava Raju, and Ranita Ganguly. "Migrating Traditional Applications to Cloud-Native DevOps: A Framework for Successful Cloud Migration and Modernization." *2025 13th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2025.

[17]. Ambroszkiewicz, Stanislaw, Waldemar Bartyna, and Stanislaw Bylka. "Functionals in the Clouds: An abstract architecture of serverless Cloud-Native Apps." *arXiv preprint arXiv:2105.10362* (2021).

[18]. Gil, Guilherme, Daniel Corujo, and Paulo Pedreiras. "Cloud native computing for industry 4.0: Challenges and opportunities." *2021 26th IEEE international conference on emerging technologies and factory automation (ETFA)*. IEEE, 2021.

[19]. Tabbassum, Ayisha, et al. "Developing Cloud-Native Autonomous Systems for Real-Time Edge Analytics." *2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*. IEEE, 2024.

[20]. Bajaj, Deepali, et al. "Partial migration for re-architecting a cloud native monolithic application into microservices and faas." International conference on information, communication and computing technology. Singapore: Springer Singapore, 2020.