Serverless Architectures and Function-As-A-Service (Faas): Scalability, Cost Efficiency, And Security Challenges

Bhanuprakash Madupati

MNIT, MN, April 2023

Abstract:
Serverless computing and Function-as-a-Service (FaaS) have become the new "new hotness" in cloud computing. Since FaaS completely abstracts away the infrastructure, developers are left with having to write only their app logic — a very cost-effective and scalable solution for modern workloads. This paper looks at the rise of serverless architectures through these key platforms. We get answers on the benefits of the scalability and elasticity models and cost implications compared with traditional server-based infrastructures. The case study demonstrates how the pay-per-execution model can prevent resource waste and some problems developers might face in implementing this way of working, such as cold-start latency and informal ecosystems. We further present three case studies on serverless machine learning workloads, which show the practical benefits of serverless computing.

Keywords: Serverless computing, Function-as-a-Service (FaaS), cloud computing, scalability, cost efficiency, AWS Lambda, Azure Functions, Google Cloud Functions, and machine learning.

1. Introduction

Cloud computing is one of the biggest game-changers in how businesses and developers handle applications. Serverless computing — Serverless computing (Function-as-a-Service model) is one of the latest evolutions in this field, where users can run codewithout provisioning or managing servers. This model abstracts away infrastructure management and provides a billing structure on a pay-by-the-hour basis, focusing on just the workloads. It is a very attractive programming model for modern applications as it allows you to be flexible and scalable and offer low costs — The heavy lifting of resource management is done by cloud providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) [1]

FaaS enables developers to split an application into small, independent functions that respond to events. These are stateless functions and execute in ephemeral runtimes (only while warm), which aids in optimally cutting down costs by using resources when needed only. This model is excellent for event-driven applications like handling HTTP requests, Uploading files and Real-time data processing [2]. This model has been adopted by all major cloud platforms, such as AWS Lambda, Azure Functions, and Google Cloud Functions, making it possible for users to realize these benefits irrespective of the technology ecosystem they operate [3].

While there is a lot to love about serverless architectures, several challenges need to be addressed. A cold start latency is one of the primary concerns that delays setting up the function when it is executed for the first time. Moreover, where it eliminates the need for infrastructure management, serverless computing introduces challenges in security, like function isolation and resource contention by tenants in multi-tenant environments [4]. In this paper, we work to solve both of these challenges while also exploring the overall scalability and cost-efficiency of serverless computing — including some interesting real-world applications coming into play as it grows more prevalent with machine learning workloads.

2. Serverless Architectures and FaaS Overview

Serverless Architectures are the most significant change in developing and deploying cloud applications. Serverless computing is an event-driven cloud-based model where developers can focus on writing code without worrying about managing application infrastructure, including scalability or availability. This is done by Function-as-a-Service (FaaS), where small, individual functions are executed in response to specific events like an HTTP request or a change of something in your data storage[1]

2.1 Key Features of Serverless and FaaS

How does serverless computing differ from traditional cloud infrastructure models?

1. Functions in FaaS are stateless, so the invoker does not maintain any information from call to call. They can scale out because each instance processes functions independently, allowing for many instances to be spun up to handle concurrent chores [2].
2. Event-driven Model—Operations are executed when something happens, e.g., an HTTP request, a file upload, or a database change. Serverless is, in fact, an event-driven approach, like real-time data processing and IoT [3]. It enables the applications to respond to user actions (or system changes) rather than being invoked as a result of some HTTP (s) requests.
3. Auto-Scaling: One of the most powerful features is auto-scaling. The resources claimed by the cloud customer are dynamically allocated by the cloud provider based on the current requirements. As more requests come in, extra function instances are automatically produced, and when fewer requests arrive, resources scale back down and offer the most resource utilization [4].
4. Pay-per-Execution Pricing—Rather than paying for reserved compute resources (as is the case with traditional cloud models, regardless of whether you use them), FaaS offers a pay-per-execution model. This means users are only billed for the specific time their functions execute, which effectively reduces costs, particularly in workloads with burst demand [5].

Table 1: Comparison of Major FaaS Platforms (AWS Lambda, Azure Functions, Google Cloud Functions). This table will summarize the key features of the three major platforms.

Feature	AWS Lambda	Azure Functions	Google Cloud Functions
Event-Driven Architecture	Supports a wide range of AWS services	Integrated with Azure Event Grid and other services	Supports events from Google Cloud services
Languages Supported	Node.js, Python, Java, Go, Ruby, etc.	C#, JavaScript, Python, Java	Node.js, Python, Go, Java
Auto-Scaling	Automatically scales based on demand	Scales with load	Scales automatically based on events
Pricing Model	Pay-per-execution	Pay-per-execution with tiered pricing	Pay-per-execution
Integration	Deep integration with AWS ecosystem	Tight integration with Azure services	Integration with Google Cloud services, especially ML and big data tools

2.2 FaaS Platforms: AWS Lambda, Azure Functions, and Google Cloud Functions

Almost all the major cloud service providers provide their FaaS or function-as-a-service platform utilizing serverless architecture principles, which are being used extensively.

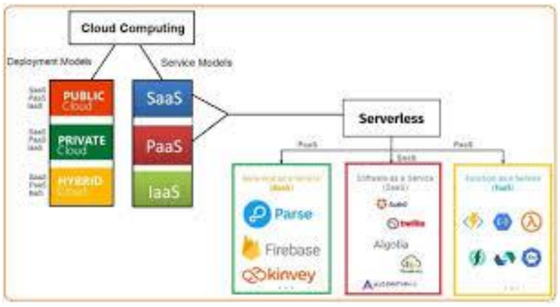
1. **AWS Lambda:** One of the first and most popular FaaS offerings, AWS Lambda lets you run your functions without provisioning (or paying for) servers. It works nicely with other AWS services like S3 and DynamoDB, enabling developers to create event-driven applications [3].

2. **Azure Functions:** Similar to AWS Lambda, this is Microsoft Azure's serverless platform that puts function into an event-driven architecture. Developers can simply write their code and deploy it in multiple supported languages. This is especially true for enterprise users, as Azure Functions plays nicely in the Azure ecosystem [6].
3. **Google Cloud Functions:** Google's implementation of FaaS supports auto-scaling and event-driven function execution, ideal for applications using Google Cloud's big data and machine learning tools [6].

2.3 Comparison to Traditional Architectures

Developers in traditional cloud architectures must deal with virtual machines (VMs) or containers for both the underlying infrastructure and scaling requirements. This frequently leads to wasteful use of resources because servers have to be planned live on peak workload, even if they sit at minimal utilization most of the time. On the other hand, serverless computing removes all of this from you — you get a very elastic infrastructure that auto-scales with only actual usage [4]. This not only cuts down on the operational complexity of managing those resources but also saves costs, especially for applications with workloads that are very irregular or drop down to zero [5].
The serverless architecture has become the default choice for new cloud-native applications, offering high productivity, extreme scalability and cost-effective resource consumption characteristics that were hard to achieve with traditional infrastructure models [2].

Figure 1: Diagram comparing serverless (FaaS) and traditional cloud (VMs/Containers) architecture. This will provide as a visual representation of the transition from server-managed models to function-managed (FaaS) systems, which prioritize the deployment of autonomous, event-driven functions.



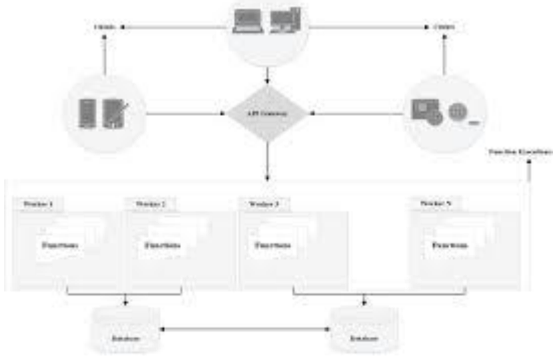
3. Scalability in Serverless Computing

One of the most interesting benefits of Serverless Computing and Function-as-a-Service(FaaS) is automatic scaling based on workload fluctuations. Developers must manually provision and manage resources in traditional cloud computing models when traffic increases. As serverless platforms take care of all this complex infrastructure, you do not need to do anything. You can even allow it to scale the instances depending on the demand for your project. This section will examine the two main factors that make serverless architectures scalable and their related obstacles.

3.1 Automatic Scaling

Serverless platforms like AWS Lambda, Azure Functions and Google Cloud Functions automatically provide the scaling of function instances according to event triggers. Both the former, horizontal scaling (i.e., increasing the number of instances) and vertical scaling (i.e., increasing resources allocated to each instance), and the latter capabilities guarantee that the system can cope with bursts in traffic effectively [1].
With usage, in a traditional cloud model, over-provisioning is resource reservation; resources are reserved for high load times excessively, like during peak season and more on cost even when the traffic is low. This problem is solved by serverless models, in which resources are dynamically provisioned and released based on demand, improving resource utilization and reducing costs[3].

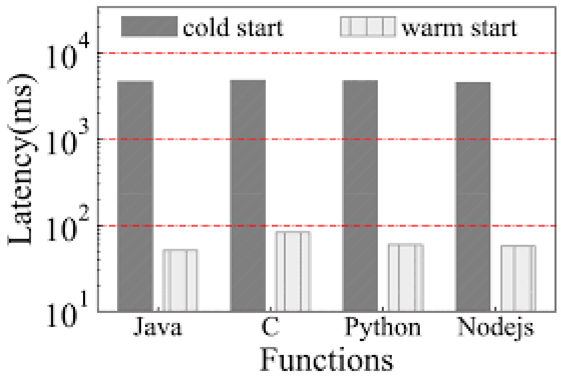
Figure 2: A serverless computing example of auto-scaling. The number of active instances increases as the event load increases in this picture, which illustrates how serverless instances dynamically scale in response to traffic surges.



3.2 Cold-Start Latency

Although serverless functions are capable of scaling infinitely, there is also a cold start delay as we need to wait for the function instance to initialize on the first run. This means that when a function is called after being idle, it takes time for the cloud provider to start up the runtime before the function can respond. Real-time respond: This cold-start latency may be important for real-time use-cases, specifically the use-cases when the functions are invoked sporadically [4].

Figure 3: Performance during warm-start versus cold-start delay. This graph shows the response time difference between serverless function executions that are cold-started and those that are warm-started (i.e., consecutive invocations). It also shows that cold-started functions have a larger delay than warm-started functions.



3.3 **Challenges in Scaling Complex Workloads**

Serverless platforms are great at scaling lightweight, stateless functions. However, challenges emerge when working with heavier and resource-hungry workloads. The stateless nature and ephemeral execution environments of serverless might not be conducive to applications which need long-running processes, state or high inter-function communication, such as machine learning model training [5]. This results in the possibility of performance bottlenecks, as data exchange between function instances or external services can add latency.

It cannot be expected that serverless will apply to all data. For example, giant deep neural network training suffers from performance degradation due to massive amounts of coarsely-grained data fetches between large-scale serverless instances. Serverless would be hindering because, while effective at parallelizing tasks, essential data exchange between functions to accomplish the task winds up increasing latency [6]

Table 2: Scalability Aspects of Serverless vs. Traditional Cloud Architectures
The scalability, resource management, and cost implications of serverless architectures and standard cloud systems are contrasted in this table.

Feature	Traditional Cloud (VMs/Containers)	Serverless (FaaS)
Scaling Mechanism	Manual or automated scaling (pre-configured)	Automatic scaling based on event-driven triggers
Resource Provisioning	Requires pre-allocation of resources	Resources are provisioned dynamically as needed
Cold-Start Latency	N/A (persistent servers)	Cold-start delays can affect initial function response times
Cost Efficiency	Higher cost due to reserved capacity	Pay-per-execution model ensures lower cost for variable workloads

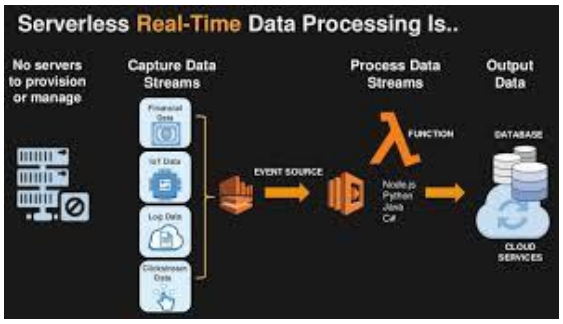
3.4 Use Cases and Real-World Applications

One of the primary use cases for serverless computing is applications with unpredictable workloads that might need to quickly scale up and down based on demand. Use cases include:

Serverless Functions: For processing data streams such as real-time log analysis, IoT data processing, monitoring [2]. Auto-scaling allows serverless platforms to easily handle this influx of data volume that can quickly scale up to accommodate the load and just as swiftly scale back down when traffic subsides.

Web Applications: The function of serverless computing is very good in applications which have variable traffic, for instance e-commerce sites such as the levels of traffic are only high and low such as sales events or significant holidays. Auto-scaling guarantees that the hardware can manage high peaks of user activity without having to be adjusted manually [4].

Figure 4: Scalability in real-time data processing is seen This figure shows how serverless functions automatically adapt based on the volume of incoming events and dynamically scale to manage growing data streams.



4. Cost Efficiency of Serverless Architectures

Cost-Effective — Serverless computing and Function-as-a-Service (FaaS) have been considered one of the most compelling reasons for adoption. Cloud computing has been tiered by higher transitional models benefitting resources over the provision of hem increases and operational costs. On the other hand, as serverless architectures are pay-per-execution type models where you pay for only the time your code runs, this model works great for any unpredictable demand or workloads. In this chapter, I will highlight the pros of cost-efficiency in serverless computing and also some caveats and best practices for modernizing costs.

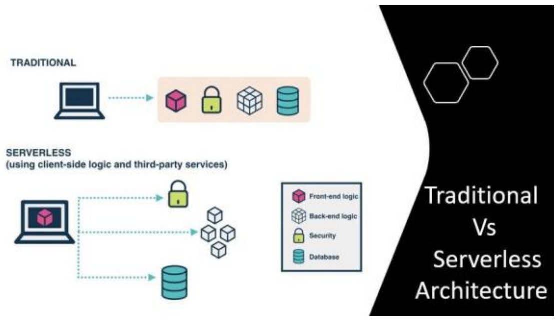
4.1 Pay-Per-Execution Model

As an alternative to container-based continuous scaling, FYTE allows for cost-containment of temporary computational resources compared to a traditional cloud model in which users have paid more according to fixed resource allotments, irrespective of their use. Here, the user has been charged for three major purposes in serverless:

1. Invocations: you have to pay for each time you invoke a function users won an event.
2. The Execution Time charges are directly proportional to the time (in milliseconds) it takes to run this function.
3. Proportional to Memory Allocation: The price is proportional to the memory allocated for the function [6].

For example, if you use AWS Lambda, they charge based on the number of requests and execution duration. This model offers a compelling price model for serverless computing use cases with random spikes, where the server would be idle for a long time [3].

Figure 5: Traditional cloud (VM-based) vs Serverless (FaaS) Cost Comparison



4.2 Cost Optimization Strategies

Gains from using serverless technology naturally reduce costs, but there are several strategies to cut costs more effectively when opting for the pay-per-execution model in your architecture. These include:

1. **Function Fusion:** Developers can create a larger function by combining the many smaller tasks that require communication with each other. This will allow developers to reduce overhead costs associated with invoking too many functions. This method is particularly handy for multiple related functions and services that operate together in the same workflow [6].
2. **Memory and resource optimization:** One critical factor in reducing costs is choosing the best memory size for your functions. Too much memory increases costs, and too little might slow your API. One should strike a balance by monitoring memory usage and tuning function settings [7].
3. **Cold-Start Optimization:** Cold-starts introduce a cost when function invocations run for the first time as they perform resource initialization. For the problem with cold starts, one can utilize provisioned concurrency, in which a cloud provider always maintains an instance of functions running but needs to pay extra cost [4].

Table 3: Cost Components of Serverless vs. Traditional Cloud Computing

Feature	Traditional Cloud (VMs)	Serverless (FaaS)
Billing Model	Pay for reserved instances (up-time)	Pay-per-execution (actual usage)
Idle Resources Cost	High	None
Resource Allocation	Predefined (based on peak usage)	Dynamic (allocated on demand)
Cold-Start Costs	N/A	Higher initial cost due to latency
Optimization Techniques	Manual scaling	Function fusion, memory optimization

4.3 Cost Efficiency Challenges

Although serverless computing has many benefits, it is not free of drawbacks related to cost optimization. Here are just some of the few things people worry about:

1. Denial-of-Wallet attacks / Over-Invocation: Misconfiguration or malicious activity may cause a function to be invoked excessively, leading to unexpectedly high costs. And while continuous invocations due to external triggers without realizing it can also happen [4].
2. Serverless architectures are designed for short-lived tasks, not long-running ones. If functions exceed the maximum allowable execution time (15 minutes in the case of AWS Lambda) and are often restarted or re-invoked [3], considerable expenses can be accumulated. Traditional cloud models are also going to be more cost-effective with long-running tasks.
3. Provisioned Concurrency Costs — A solution to cold-start latency is provisioned concurrency, where function instances are kept warm and ready to handle requests. While it improves performance, it increases costs because resources are pre-allocated, contrasting the original pure pay-per-use model [7].

4.4 Use Cases Benefiting from Cost Efficiency

Serverless is very affordable for workloads that are either used sporadically or with no way to accurately determine user behaviour. Examples of its use could be:

1. IoT Apps: Though Serverless is best suited for IoT where sensible servers may have constant data to process only a few minutes of the day and rest unlikely [2]
2. Batch Processing: Serverless architectures are a good fit for batch-processing workloads that can be queued up and processed in parallel without needing dedicated servers [5].
3. Lastly, serverless platforms are also effective for microservices due to the fine-grained scaling and billing at the function level, which reduces overall cost [6].

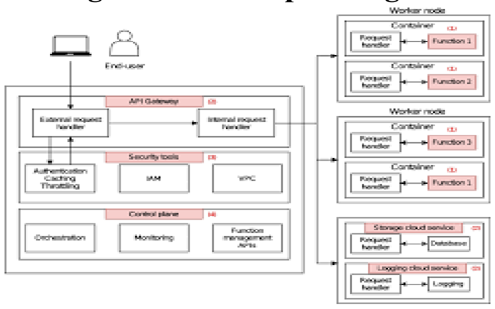
5. Serverless Architectures — Security Considerations

Serverless computing has many benefits, such as cost and scalability, but it also introduces unique security challenges. New attack surfaces: Serverless architectures run in response to external events and triggers from various third-party cloud providers, which creates new potential points of attack that were never an issue with traditional servers. This section will detail the security issues related to serverless computing and provide guidelines for mitigating those risks.

5.1 Isolation of Functions/ Multi-tenancy

In a serverless environment, multiple functions from different users or applications can run on the same underlying infrastructure. The problem is with function isolation, where a vulnerability in one function might lead to an attack on another running on the same hardware. While most serverless platforms deploy functions in containers or similar lightweight runtime environments, strong isolation should still be enforced to stop cross-function attacks [4].

Figure 6: Serverless functions' security structure is shown. The relationship between containers, serverless functions, and the underlying infrastructure is shown in this figure. It illustrates the security precautions taken by cloud providers to safeguard function boundaries as well as how isolation is preserved amongst functions operating on the same infrastructure.

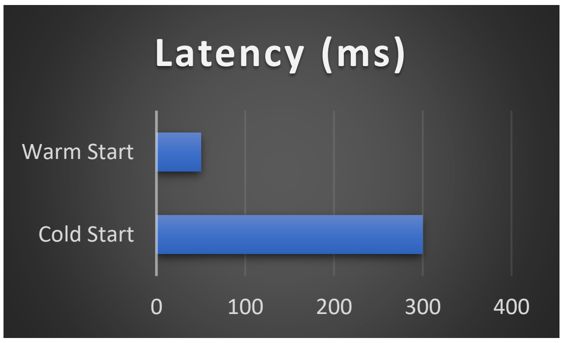


5.2 Cold start vulnerabilities

As we mentioned above, cold starts are situations where a function is started from "cold" after a period of inactivity, triggering the provider to initialize the execution environment. This results in a window of opportunity for these functions to be more susceptible to attack as the runtime environment is being established. Attackers can exploit this initialization phase by exploiting misconfigurations or delays in security enforcement mechanisms like authentication, identity management, and access control [3].

To address cold-start latency issues, some cloud providers have introduced the concept of provisioned concurrency, in which instances are replenished and kept warm to accept requests. This lowers the risk of cold-start attacks but at a price and is not always adequate for all use cases [6].

Figure 8: How security vulnerabilities are affected by cold starts. The security vulnerability window for warm-started versus cold-started functions is compared in this graph, demonstrating how improper security during the startup process might expose functions to possible threats.



5.3 Denial-of-Woney (DoW) Attacks

One danger specific to serverless architectures is a Denial-of-Wallet (DoW) attack. Here, malefic actors take advantage of the pay-per-execution model by repeatedly making calls to functions, ultimately leading to such practices being a financial loss for the victim. This may lead to a sudden surge in the number of function invocations, ultimately leading to rising costs for the user as serverless platforms scale with demand automatically [5].

With DoW attacks, mitigation strategies can include implementing rate limiting on function invocations and usage alerts to monitor for strange patterns of function executions. Further, API Gateway services could also be set to throttle incoming requests, which would add another level of protection against over-invoking [4].

Table 4: Security Challenges in Serverless vs. Traditional Cloud

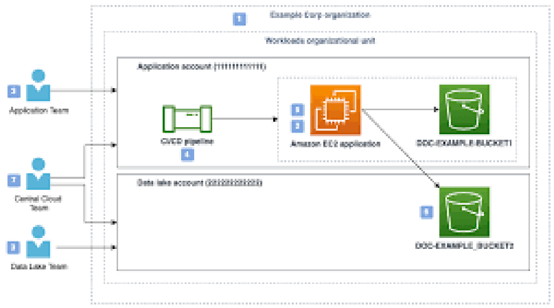
Security Concern	Traditional Cloud (VMs/Containers)	Serverless (FaaS)
Function Isolation	VM/container isolation is well-defined	Requires more lightweight isolation mechanisms such as containers
Cold-Start Vulnerabilities	N/A (persistent environments)	Cold-start delays expose potential security gaps during initialization [4]
Denial-of-Service (DoS)	Resource-based DoS attacks	Denial-of-Wallet (DoW) attacks based on pay-per-execution model [5]
Access Control	Managed by the user at the server level	Often delegated to the cloud provider, but misconfigurations can lead to vulnerabilities [3]

5.4 Identity and Access Management (IAM)

Serverless architecture is mainly used in environments where different functions make calls to other services running on another cloud service, such as databases, storage, or APIs. We must also ensure that each function

is given only as much privilege as is necessary for the function to be helpful. Improperly set up permissions allow for data breaches and privilege escalation attacks.
Cloud providers utilize this by offering Identity and Access Management (IAM) services, which give us fine-grained control over what resources a function can access. This allows developers to ensure that the only permissions required for those functions to execute are the ones granted. For example, recurring audits of IAM policies are advised to avoid unintentional over-pe

Figure 9: Example of IAM policies in serverless architectures. This diagram shows how IAM policies are applied to control access to cloud resources by serverless functions. It demonstrates how roles and permissions are assigned to functions, restricting access to sensitive data and services.



5.5 Third-Party Dependencies and Vulnerabilities

Many serverless applications will have third-party libraries/dependencies, which could also mean another entry point. Like any other system, a seemingly innocent script update can result in a potential vulnerability. The problem with this approach is that it leaves you open to vulnerabilities in these dependencies, which attackers can use to take over functions or data. Regularly updating dependencies and using vulnerability scanning tools for patches can help to reduce this risk.
Specific PaaS platforms have integrated the functionality of automatically analyzing and fixing vulnerable dependencies to add another level of protection. Minimize the use of wrought in critical functions. Developers should also consider minimizing the overuse of external libraries, especially for vital functions, to reduce the attack surface [4].

6. Future Work

While serverless computing has transformed cloud-native application development, it is not the end of the story. Many potentials remain explored and researched for better usage and performance improvements.

1. **Cold-Start Latency Optimization:** Serverless architectures are known to suffer from cold-start issues, which make them inappropriate for time-sensitive applications… However, there is a potential scope of research here that could extend the same optimization phase discussed in a previous section to include function initialization times. A cool-down warm-up solution like this (e.g., an A/B feature shunt) could diminish the effect of cold starts [3].
2. **Certain machine:** learning workloads can massively benefit from serverless computing, but it is by no means viable for frequently retraining models, and various other use cases remain almost impossible to achieve under the model without significant operational overhead. This type of work could be further extended to hybrid models or serverless infrastructure specialized in distributed machine learning workloads [7].
3. **More Secure Security Mechanisms:** As serverless architectures only start to grow, more secure security mechanisms against the risk of multi-tenancy, denial-of-wallet attacks, and dependencies must be stable. Around what we have reached so far, some of the future steps will be to construct security models which are designed to fit the specifics of serverless platforms [4]

4. **Cost Optimization of Complex Workflows:** In the future, we would like to evaluate different cost optimization strategies for various application frameworks (workflow patterns) where an application has stateful functionalities and long-living tasks or responsibilities are conditioned across functions. Advanced function fusion techniques and more cost-efficient memory allocation models could improve cost efficiency [6].
5. Serverless is already heavily used for real-time processing, IoT, and microservices; however, an interesting direction could lie in exploring its usage scenarios in other areas—edge computing or large-scale scientific simulations are examples of such work. While the use cases served by these areas also require scalability and real-time processing, their optimization strategies may differ from ours [5].

7. Conclusion

1. Serverless computing offers a fully managed, serverless and cost-effective alternative to traditional cloud architectural patterns for event-driven or ephemeral workloads. The auto-scaling capabilities of serverless platforms like AWS Lambda, Azure Functions, and Google Cloud Functions reduce idle resources, keeping the costs down and dynamically scaling the applications in response to demand, thus improving resource utilization [1], [2].
2. With pay-per-execution pricing it makes serverless a reasonbale option for workloads with random traffic (or that can run on-demand). Optimization techniques like function fusion and memory tuning can reduce costs further, but developers must avoid over-invocation issues (e.g., Denial-of-Wallet attacks [6]).
3. Security remains a key challenge in serverless architectures, with threats such as cold-start vulnerabilities, function isolation issues and third-party dependency risks. These risks can only be mitigated through secure IAM policies and by incorporating security best practices.
4. While offering many benefits, serverless computing may not be the best fit for intricate and long-running tasks. However, stateful and persistent inter-function communication applications might not see the desired benefits of an inherently stateless and ephemeral serverless environment. Serverless implementations require hybrid models and additional methodological adjustments to grow the utility of this platform towards more complex scenarios.
5. For the future of serverless computing, follow-up research has to focus on reducing cold-start latency, improving security, and optimizing costs. Serverless architectures are still evolving and rebuilding the world around them, and they are expected to become booming in areas like real-time data processing, machine learning, etc.

References

1. Barrak, F. Petrillo, and F. Jaafar, "Serverless on Machine Learning: A Systematic Mapping Study," *IEEE Access*, vol. 10, pp. 99337–99352, 2022, doi: https://doi.org/10.1109/access.2022.3206366.
2. D. Taibi, N. El Ioini, C. Pahl, and J. Raphael, "Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Literature Review," *Urn.fi*, 2020, doi: https://doi.org/9789897584244. Available: https://urn.fi/URN:NBN:fi:tuni-202008286730..
3. M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural Implications of Function-as-a-Service Computing," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, Oct. 2019, doi: https://doi.org/10.1145/3352460.3358296.
4. E. Marin, D. Perino, and R. Di Pietro, "Serverless Computing: A Security Perspective," *Journal of Cloud Computing*, vol. 11, no. 1, Oct. 2022, doi: https://doi.org/10.1186/s13677-022-00347-w.
5. Y. Li, Y. Lin, Y. Wang, K. Ye, and C.-Z. Xu, "Serverless Computing: State-of-the-Art, Challenges and Opportunities," *IEEE Transactions on Services Computing*, pp. 1–1, 2022, doi: https://doi.org/10.1109/tsc.2022.3166553.

6. T. Elgamal, A. Sandur, K. Nahrstedt, and G. Agha, “Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement,” *Proceedings of the 2018 IEEE International Conference on Edge Computing (SEC)*, Dec. 2018, doi: https://doi.org/10.1109/sec.2018.00029.
7. L. Feng, P. Kudva, D. Da Silva, and J. Hu, “Exploring Serverless Computing for Neural Network Training,” *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Jul. 2018, doi: https://doi.org/10.1109/cloud.2018.00049.