Serverless Computing in Cloud Environments: Architectural Patterns, Performance Optimization Strategies, and Deployment Best Practices

*By **Vishal Shahane**,*

Software Engineer, Amazon Web Services, Seattle, WA, United States

Orcid ID - https://orcid.org/0009-0004-4993-5488

Abstract

Serverless computing has emerged as a transformative paradigm in cloud environments, offering unparalleled scalability, cost-efficiency, and developer productivity. This research paper provides a comprehensive examination of serverless computing, focusing on architectural patterns, performance optimization strategies, and deployment best practices.

The paper begins by elucidating the fundamental concepts and principles of serverless computing, highlighting its event-driven, ephemeral nature, and its abstraction of underlying infrastructure management. Architectural patterns commonly observed in serverless applications, such as function-as-a-service (FaaS), event sourcing, and microservices, are explored in detail. These patterns enable developers to build scalable, resilient, and loosely coupled systems that can adapt to dynamic workloads and changing requirements.

Next, the paper delves into performance optimization strategies tailored for serverless environments. Techniques for minimizing cold start latency, optimizing function execution time, and managing resource allocation are discussed. Additionally, strategies for orchestrating complex workflows, caching data, and leveraging serverless-specific services for storage, messaging, and event processing are examined. These strategies aim to maximize performance and efficiency while minimizing operational overhead.

Deployment best practices for serverless applications are also elucidated, encompassing considerations such as deployment automation, continuous integration and delivery (CI/CD), environment configuration, and security. The paper emphasizes the importance of automation and infrastructure as code (IaC) principles in streamlining the deployment process and ensuring consistency across environments. Security best practices, including access control, data encryption, and compliance management, are integral components of serverless deployment strategies.

To validate the effectiveness of architectural patterns, performance optimization strategies, and deployment best practices, the paper presents case studies and real-world examples from diverse industry sectors. These case studies demonstrate how organizations have successfully leveraged serverless computing to achieve scalability, cost savings, and faster time-to-market for their applications. Lessons learned and best practices derived from these case studies inform the development of guidelines and recommendations for future serverless deployments.

In conclusion, serverless computing represents a paradigm shift in cloud computing, offering unprecedented flexibility and efficiency for developing and deploying applications. By understanding architectural patterns, implementing performance optimization strategies, and following deployment best practices, organizations can harness the full potential of serverless computing to drive innovation and accelerate digital transformation.

Keywords: serverless computing, cloud environments, architectural patterns, performance optimization, deployment best practices, function-as-a-service (FaaS), microservices, scalability, cost-efficiency

1. Introduction to Serverless Computing

As with every new technology proposing computing, serverless is on everyone's mouth. The term serverless itself creates a bit of confusion. It does not mean that servers are not used. Actually, servers are still used but the complexity to manage them is hidden from the developer. Therefore, serverless should be better called "backendless" as only the backend work is facilitated. Managing the business logic is simpler as the functions are uploaded to the cloud provider, and the events do their work being executed. In front of these advantages, there are some drawbacks and some of them are really important. Currently, a serverless function is constrained as it cannot run for long execution times or occupy large amounts of memory. Its scope is also limited, being unable to share the database connections and caches. One key aspect of serverless computing is its ability to automatically scale up and down, depending on the workload, providing cost-effectiveness and flexibility. Since it is pay-per-use, it allows for precise allocation of resources and eliminates idle time. Additionally, serverless offers seamless integration with other cloud services, enhancing the efficiency of the entire ecosystem. However, it is important to note that the security of serverless applications cannot be taken for granted. Security protocols need to be tightened and constantly monitored to prevent data breaches and unauthorized access. Furthermore, the lack of standardization across serverless platforms can pose challenges for developers when attempting to migrate code and functions. Serverless computing is

undoubtedly revolutionizing the way applications are built and deployed, but it comes with its own set of complexities that require careful consideration.

Despite this, serverless should be considered a breakthrough, and its constraints are being removed one by one. How are we going to show them be in a container, scale from zero to thousands of instances in a minute, occupy large amounts of memory, and execute for a long time until days? As the technology evolves, the industry moves fast to adopt it. As a result, it is no longer only small startups that are using serverless. Large companies are using it to replace the most tedious manual tasks, executing machine learning models and image processing, creating an event-driven application, and even serving part of the user request through the use of serverless APIs. Only by knowing its capabilities and constraints, serverless can be properly employed. That is the goal of this chapter.

1.1. Definition and Key Concepts

"Serverless computing" represents the most most recent addition of cloud computational models, allowing for more fine-grained implementation of on-demand computing services. Architecturally, serverless computing extends the capabilities of Platform as a Service (PaaS) cloud services. In a PaaS cloud, the subscriber is provided with a set of programming environments for developing web-scale cloud applications. Under the PaaS model, the application does not worry about the underlying cloud infrastructure, including operating systems, network setups, or storage systems. Instead, the application is exposed to the required middleware components needed to develop and run the cloud application. In a PaaS cloud, the cloud subscriber needed only to provide the application code and any required configurations, with the cloud being responsible for the deployment and scaling of the application according to the received load. However, applications developed under the PaaS model still need to be hosted on active servers where the provided PaaS runtime environments are executed. This is where serverless computing comes in. It allows developers to build and run applications and services without thinking about servers at all. With serverless computing, developers can focus on writing code and delivering value, without managing the underlying infrastructure. This is achieved by using cloud functions, which are event-driven, serverless compute solutions that automatically execute small, self-contained pieces of code in response to events. Serverless computing can provide benefits such as reduced operational costs, increased developer productivity, and enhanced scalability and flexibility. By embracing serverless computing, organizations can streamline their development processes and further optimize their cloud resources for maximum efficiency.

On the other hand, serverless computing, under the Function as a Service (FaaS) cloud computing model, modifies the former PaaS model to allow for more fine-grained cloud services. In serverless computing, the cloud subscriber no longer needs to define workloads using cloud-provided containers. Instead, the cloud subscriber's code is broken down into small chunks of service logic, known as

"functions". Each function execution is short-lived and is typically triggered by well-defined events, such as HTTP requests, database transactions, or message queue injections. Cloud providers host and execute the functions on behalf of the cloud subscribers, enabling the latter to truly only worry about the code running inside the function. Since its inception, the serverless computing model has seen a large adoption in both research and industry, with large commercial cloud providers exposing serverless offerings such as AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions.

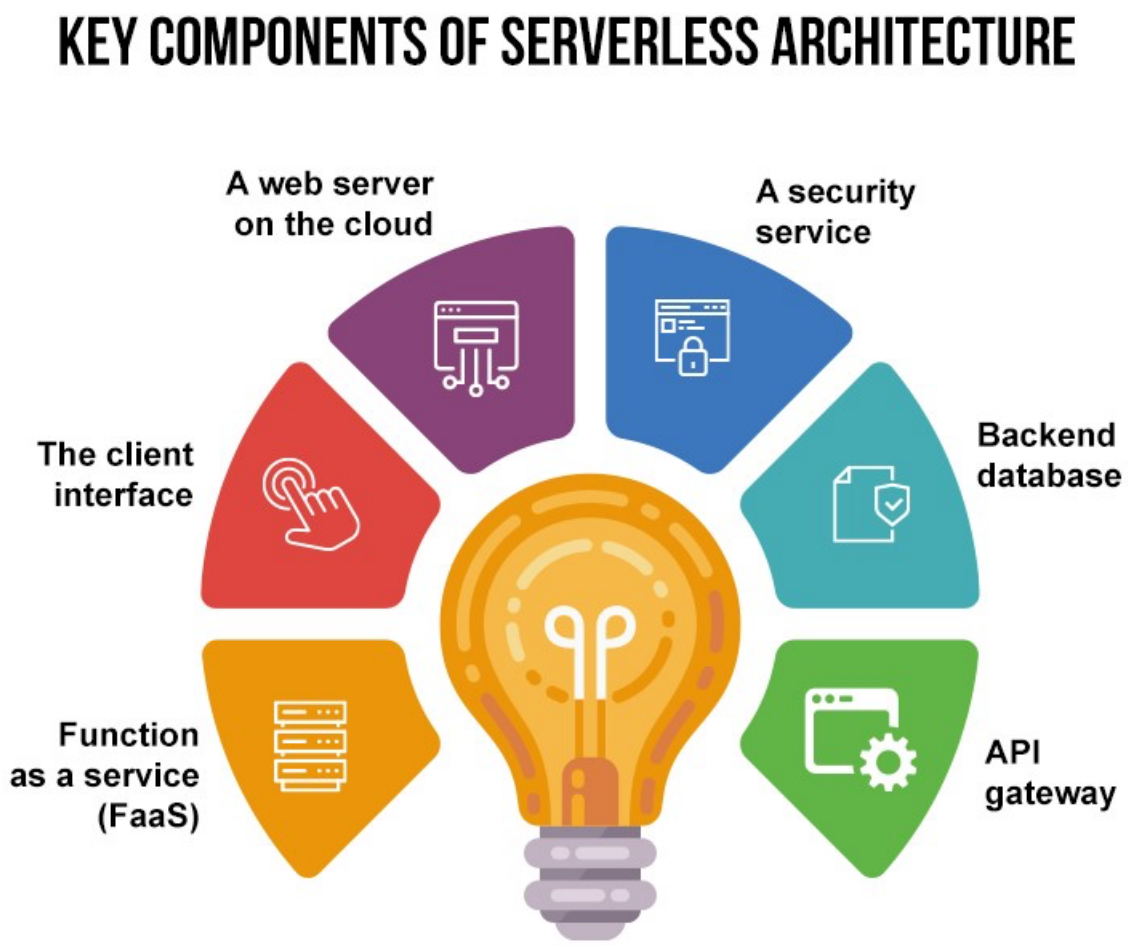
1.2. Historical Development

Providing services without maintaining physical or virtual server infrastructure has been an aspiration for many software vendors. With the developing Internet infrastructure and increased bandwidth on links, software delivery over the Internet became more feasible. Application Service Providers (ASPs) started to deliver software over the Internet to a smaller customer base. With developments in virtualization technologies adopted as enablers, Cloud Computing, as we know in its modern definition, started to flourish. However, in Cloud Computing, customers are still responsible for managing the applications and services deployed. They are required to provision, utilize, and pay for Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) components. This can be prohibitive, especially for small companies and startups that have small and often unpredictable workloads. At the same time, the benefits of cloud computing cannot be overlooked. The potential of saving time and costs by not having to manage physical servers is enormous. Furthermore, the scalability and flexibility of cloud infrastructure make it an attractive option for companies seeking to expand their digital operations without investing in physical hardware. This has paved the way for the rise of Managed Service Providers (MSPs), who offer scalable and reliable cloud-based solutions to businesses of all sizes. By partnering with an MSP, companies can take advantage of the benefits of cloud computing without the burden of managing their own infrastructure, freeing up valuable resources to focus on their core business activities.

Serverless Computing (SC) represents an emerging paradigm in which developers no longer have to consider physical or even virtual servers as part of their development and operations work. SC draws its name from the idea that cloud providers will execute functions or scripts on behalf of developers. Despite the name, SC does not imply that software functions are short-lived or are stateless. They can, and often do, execute for a longer duration and can introduce stateful processing. However, usually, individual functions execute for a short duration and perform a limited amount of work. Over the last few years, SC has gained momentum and has become a popular choice for developing event-driven and microservices-based new-generation applications.

2. The Foundational Elements of Cloud-Based Serverless Computing Solutions

Cloud computing has transitioned from the use of virtualized data-center-based infrastructure to models that feature elastic, on-demand allocation of higher-level, granular pieces of computing resources. Serverless computing is a more recent extension of this paradigm, offering event-driven execution of code snippets in the form of functions, with associated short-lived or ephemeral-state containers. Serverless computing introduces further abstraction, shifting the focus from resource-level provisioning to coding at logic and algorithm level, allowing fine-grained, parallel, and massively scalable solutions. This chapter introduces foundational aspects, best practices, and deployment strategies for building effective serverless solutions using cloud-based serverless computing frameworks such as AWS Lambda, Azure Functions, Google Cloud Functions, and IBM Cloud Functions.

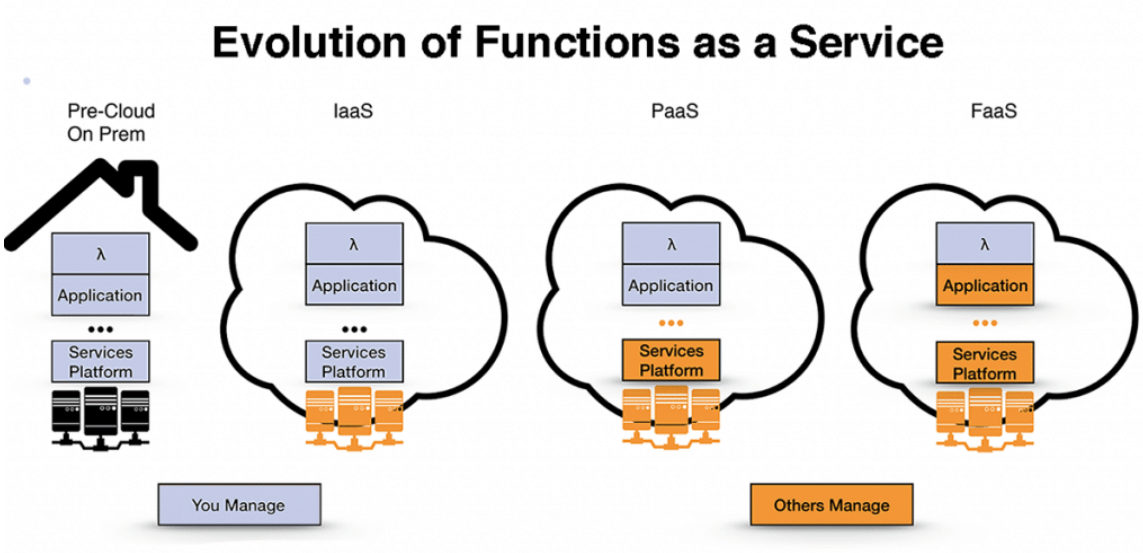


The combination of cloud computing with serverless computing frameworks offered by major cloud platform providers leads to a powerful model of computing. Cloud-based serverless computing allows

developers to focus on the core logic and algorithms of their code, letting the cloud platform take care of the runtime execution, scaling and parallelism. New architectural patterns are enabled, and new performance and cost optimization strategies need to be considered. In this chapter, we will examine the foundational elements of cloud-based serverless computing solutions and discuss a few relevant architectural patterns. We will also introduce performance optimization strategies for cloud-based serverless computing solutions and review important deployment best practices. Throughout the chapter, we will use examples from the AWS Lambda and Azure Functions platforms to illustrate the various concepts.

2.1. Function as a Service (FaaS)

Serverless computing is a computing model where the cloud provider is responsible for executing a piece of code by dynamically allocating and managing the resources required. The user only pays for the actual amount of resources consumed to execute the code. There are two major serverless computing variations: Function as a Service (FaaS) and Backend as a Service (BaaS). In the FaaS model, the user provides the piece of code that is executed in response to an event (such as an HTTP request or a database update). Execution results are returned to the invoker. Typically, function code is relatively short-lived and has a clearly defined purpose. It is generally accepted that FaaS offerings bring the true economics of serverless computing as they model compute resource usage at a fine granular level. Serverless computing can be a game-changer for organizations looking to optimize their IT infrastructure and reduce operational costs. By leveraging FaaS and BaaS, businesses can focus on writing code and implementing functionality without having to worry about managing the underlying infrastructure. This can lead to increased agility and faster time to market for software applications. Additionally, the pay-per-use model of serverless computing allows organizations to scale their applications based on actual demand, rather than over-provisioning resources. This can result in significant cost savings for businesses, especially those with fluctuating or unpredictable workloads.



Furthermore, serverless computing can enable the development of event-driven architectures, where code is triggered in response to specific events or triggers, such as user actions, file uploads, or sensor readings. This can be particularly beneficial for applications that require real-time processing and quick response times. With serverless computing, organizations can build highly responsive and scalable applications that can adapt to changing workloads and user demands. Overall, serverless computing offers a flexible and cost-effective approach to application development and deployment, making it an attractive option for modern businesses.

In the FaaS model, the function code is provided by the user and is typically a very short piece of logic. It is, therefore, easy to overlook some non-functional concerns and write inefficient code. It is very important to address performance optimization of the code to provide a better user experience, reduce cost, and lessen the environmental impact. Recently, it has been proposed that a refinement of serverless/FaaS is to allow the user to write more complex applications by combining user-provided functions with Customizable stateless services Logic (C-sLo). Such services would impose a longer code load time than the current stateless model but are more expressive than FaaS functions and more reactive than current stateless service implementations.

3. Design Patterns and Use Cases

The design of serverless applications is very different from traditional cloud-hosted or in-house developed enterprise applications. Serverless applications consist of custom code running within ephemeral computing environments. The code is deployed in response to events, execution of a specific function, and execution of a piece of custom logic. The deployment and execution of the code are fully managed by the cloud service providers (CSPs). With the serverless approach, developers do not need to think about servers, virtual machines, containers, operating systems, or even how their code will be

executed. As a result, the administration overhead is significantly reduced. Serverless computing holds code execution as the main building block that executes in response to data and state, without standing, pre-configured, idle resources. Providing the execution of the code does not take too long or require too many resources, serverless computing results in extreme cost reductions. Serverless computing enables developers to focus entirely on writing efficient code without having to worry about infrastructure management, thereby increasing productivity and reducing operational costs. Additionally, the pay-as-you-go model of serverless architecture ensures that developers only pay for the resources they use, eliminating the need to provision and pay for excess capacity. Serverless applications can automatically scale to accommodate changing workloads, ensuring optimal performance and cost-efficiency. The flexibility and scalability of serverless computing make it an attractive option for modern application development, enabling rapid innovation and time-to-market, while also reducing operational complexity and maintenance efforts.

Serverless cloud computing is becoming increasingly popular and commercial cloud providers have started offering Function-as-a-Service (FaaS) platforms to attract and support this new breed of cloud applications. Despite the differences with traditional Platform-as-a-Service (PaaS) cloud applications, those being fully event-driven with short-lived, stateless functions, and having no deployed components to form persistent service endpoints, there lacks a clear set of design patterns to guide serverless application development. In this chapter, we pave the way and establish six architectural patterns for serverless computing with the inclusion of typical industrial use cases and open-source projects that apply these patterns. We illustrate the functioning of these patterns using an example serverless application based on one of the identified commodity use cases, from which we draw recommendations and best practices that developers should adhere to in order to avoid common performance and cost pitfalls of serverless cloud computing.

3.1. Event-Driven Architectures

The second category of SEDA generalizes the previous one and combines serverless with all kinds of event-driven architectures, including the less loosely coupled event notification associations that are typical for hierarchical publish-subscribe middleware products, as well as more widely decoupled enterprise service eventing associations that are typical for commercial messaging products supporting the enterprise service bus pattern. The benefits expected from the combination of serverless and event-driven models become even more pronounced when existing message-driven service implementation elements become deployed using the serverless model of adjustable execution backend. As these benefits are connected to the usual drivers for applying a serverless model, mainly operational and engineering cost reduction through minimizing the presence of custom code that is not exercised most of the time, the benefits become independent of the used event-driven architectural pattern. The combination of serverless and event-driven architectures presents exciting opportunities for

organizations looking to streamline their operations and reduce costs while leveraging modern technological advancements in cloud computing and enterprise-level systems. These benefits have the potential to revolutionize how businesses approach and execute their technological strategies, making it possible to achieve seamless integration and efficient handling of event-driven processes across the board. By allowing for the deployment of message-driven service implementation elements using serverless models, organizations can optimize their use of resources and enhance their overall performance, further validating the significance of this approach in the context of contemporary enterprise and operational frameworks.

The first category of SEDA combines serverless programming models and event-driven architecture. Such a combination is rather natural, as usually, the event-handler code is executed sporadically, reacting to the occurrence of some business event or activity. Upon successful event handling, flow of control usually moves to another component, possibly triggering the execution of another event-driven handler. This kind of event-driven flow of control does not have any direct mapping to standard business processes, and therefore usually finds its natural modeling and implementation in a programming model, such as serverless, where the execution of the code is transparently managed by the platform. The benefits of serverless event-driven programming include reduction of custom code that is not exercised most of the time, easier operations by minimizing resources that are active and have to be monitored for idle time, as well as better elastic scalability of systems that grow in serviced event throughput.

4. Performance Optimization Strategies

I talk a little more about some performance optimization strategies that can be put in place to make sure that when you have a serverless application with a large number of functions that you're still getting the best possible performance out of the deployed applications. One of the key strategies that can be employed is to optimize function execution start times. So, if you have a small function and you put a lot of effort into making sure that it starts as quickly as possible. And one way you can do that is by reducing the function body size and initializing only essential dependencies at runtime and defer the initialization of other dependencies. So, for instance, if you have a function that uses pretty large libraries, you can use something like lazy loading where the library will only be loaded when it is required. In addition to optimizing function execution start times, it is also important to consider the strategy of scaling the application by breaking it down into smaller, more manageable components. This approach can help to distribute the load more evenly across the serverless architecture, ultimately leading to improved performance and scalability. By breaking the application into smaller components and deploying them separately, you can ensure that each component can be independently scaled and managed, allowing for more flexibility and efficiency in the use of resources. Another important

performance optimization strategy for serverless applications is caching. By implementing caching mechanisms, you can store frequently accessed data in memory or in a separate data store, reducing the need to repeatedly retrieve the same data from external sources. This can significantly improve the response time of your serverless applications and reduce the overall latency experienced by users. Implementing caching strategies such as content delivery networks or in-memory caching can greatly enhance the performance of your serverless applications, particularly for applications that require frequent access to the same data or assets. Furthermore, it is essential to continually monitor and analyze the performance of serverless applications to identify areas for improvement. By collecting and analyzing performance metrics such as latency, error rates, and resource utilization, you can gain valuable insights into the behavior of your applications and identify potential bottlenecks or areas of inefficiency. Utilizing monitoring and logging tools can provide real-time visibility into the performance of your serverless applications, allowing you to proactively identify and address any performance issues before they impact the user experience. Overall, by implementing these performance optimization strategies, serverless applications can achieve enhanced performance, scalability, and reliability, ultimately providing a better experience for users and maximizing the efficiency of resource utilization.

Additionally, you can also use parameterized initialization. So, instead of initializing the entire library or an entire module at the start of but you can just initialize a part of it and then the rest can be initialized based on receiving parameters during runtime and other techniques like function code splitting and using aggregated servers can also help in reducing function start times. And then once you've done that, one other optimization that can be put in place is utilizing well-connected function pathways. So, make sure you're doing pathing for streams functions that need to be executed in a sequence across the same path. And also use event batching. This is very useful when functions are invoked by event data arriving in bursts. So, instead of invoking the function one at a time, you can just batch all the events and then invoke the function and process events in a batch.

4.1. Cold Start Mitigation Techniques

In this particular section, we are going to display the fundamental CS problem and CS initiation time and then provide a detailed explanation of three different categories of CSTs. First and foremost, let us concentrate on the identification of the problem and the evaluation of the magnitude of the problem in AWS. We will utilize AWS Lambda, its linked request-response service model, and the typical web service via Lambda architecture to elaborate and exemplify CS-related concerns for the remainder of this section, as these concerns are frequently cited as CS-problem-incurred cloud issues. Because of the numerous issues instigated by the CS problem, a considerable amount of unfavorable public opinions regarding serverless computing have emerged. Given that public perception plays a crucial role in the success of a service, it is imperative that the CS problem is dealt with in order to restore the public's

confidence in serverless computing. Failing to do so could result in a decline in customer base for the serverless platform and a detrimental impact on the entire cloud platform that supports serverless computing. To solve the CS problem, it is essential to understand the different aspects of the issue. This includes identifying the root causes of the problem, assessing its impact on AWS and other cloud platforms, and developing effective strategies to address and mitigate the problem. Additionally, it is crucial to educate and inform the public about the measures being taken to resolve the CS problem and reassure them of the reliability and effectiveness of serverless computing. By taking proactive steps to tackle the CS problem, the cloud platform can regain trust and confidence from its users, and ensure the continued growth and success of serverless computing in the future.

CSTs play a pivotal role in minimizing the perceived penalty time of a CS and the time to completion of any component malfunctioning related corrections for initiated functions running in a serverless environment. Minimizing perceived penalty time at CS initiation is important for successful mainstream adoption of serverless computing since the current penalty time is clearly visible in the rapid scaling down of the function and subsequent invocation delays. CSTs are redundancy and orchestration layers added to the basic serverless provisioning and function run initiation architecture. Redundancy increases the probability of finding a warm copy of the required function somewhere in the cloud. If a warm copy is found, then orchestration becomes a time-critical task to divert the invoking traffic to the warm function. If no warm copy is located, then the time-critical task is to quickly start up the required function within a known, short, predictable initiation delay time.

5. Security and Compliance Considerations

In this chapter, we identify several important security and compliance issues that may influence the desirability of serverless systems in relationship to similar cloud-hosted systems. The serverless system is made of managed services and event-driven functions, which are hosted in execution containers. The code is meant to execute custom logic using one or more of the available cloud service kits. The code performs a specific task or group of tasks and is also known as a function. The code connects to cloud services via the service kits. Cloud services, generally, offer web API endpoints. Event sources are a type of external system. They initiate function execution and pass input data into the function. The event source can be any of the various available sources. The use of serverless computing promises a more secure and easily scalable environment for the execution of code as the cloud provider is tasked with managing infrastructure security. On the other hand, as serverless does not eliminate the operations of a DevOps team, necessary security measures must be considered when adopting the serverless technology. This means that ensuring secure configurations, strong access controls, and employing proper encryption and key management are essential for protecting data, applications, and workloads in a serverless environment. Additionally, monitoring and auditing the functions and

services is crucial in detecting and responding to security incidents and unauthorized access. Without proper security measures in place, serverless systems can be vulnerable to various threats including unauthorized access, data breaches, and malicious activities. Therefore, it is crucial for organizations to adapt and implement comprehensive security strategies to mitigate and manage these risks effectively. By doing so, organizations can confidently and securely leverage the benefits of serverless computing while minimizing potential security threats and compliance issues.

In cloud environments, application components are managed by the cloud provider, decreasing the management overhead in terms of security updates, configuration, and credential management concerning the developers. Serverless is being defined as a new cloud computing execution model allowing developers to build and run applications and services without caring about infrastructure and its scaling. Security of the serverless system relies significantly on both the cloud provider and the application developer. It allows serverless functions to relax and focus on its core functionality. Most of the events initiated by the function are external to the system. The flow of security context in a serverless function is from the event input to the external libraries and system: "from input to system call". The security and compliance challenges in serverless computing are still relatively unexplored. It is crucial to understand the security and compliance challenges in organizations migrating to serverless computing.

5.1. Isolation and Multi-Tenancy

Cloud service providers have taken precautions by configuring their servers to handle workloads from different customers in order to mitigate the risk of cyber attacks originating from other customers' workloads. However, there is a need for a tenant-aware cloud, where customers using the cloud infrastructure for their applications are given allocated virtual machines on the same physical machine. This allows closely collaborating applications to experience reduced communication delays and faster data transfer. Additionally, customers can benefit from increased resource utilization by migrating some of their in-house datacenter applications to the cloud. In today's interconnected and fast-paced world, the demand for cloud services continues to grow. As cloud computing becomes more prevalent, the need for secure and efficient solutions is essential. By implementing a tenant-aware cloud, service providers can ensure that customers' applications are hosted on allocated virtual machines on the same physical server. This approach not only enhances performance by reducing communication delays but also facilitates faster data transfer between closely collaborating applications. Moreover, customers stand to benefit from improved resource utilization by migrating certain in-house datacenter applications to the cloud, further optimizing their operations and efficiency.

Serverless computing environments provide convenient, pay-as-you-go development and deployment possibilities for applications. They also help cloud providers to use their resources in a very cost-

effective way. However, to enable the low-cost benefit, most serverless platforms introduce function execution environments which execute a developer-defined function code for a very short period, and the environments are torn down without leaving any state when the code execution completes. This introduces a statelessness problem in serverless platforms. The problem is that developers often want to use serverless platforms for building data-intensive and stateful applications, among which the data and function code can be co-located to achieve low-latency access, and the application can be modeled as a set of services to benefit from the low-cost serverless architecture. Many real-world applications can be described using the model, such as chat bots, image processing services, IoT backend data collection and analysis services, etc.

6. Development and Deployment Best Practices

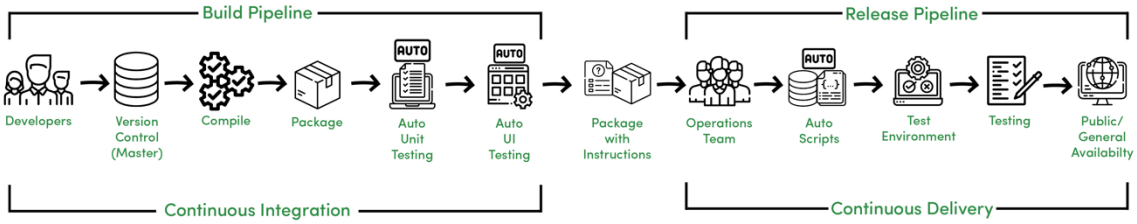
As the developer does not have to take care about the required infrastructure, one could assume that coding serverless applications is easy and can be done by every software developer without the need for further specialization. This might be true for simple applications, but for larger, more complex applications, this is not the case. It can lead to the so-called function sprawl. To prevent function sprawl, a developer should keep an eye on the number of Lambda functions deployed, and monitor the execution environment in order to include related code in the same function. As a distributed function deployment imposes a higher request overhead and increases the risk of cold starts, developers should take care to find the right balance between function simplicity and function co-location. For instance, considering the potential impact of increased latency or error rates when handling multiple API requests, it is important to ensure that the co-located functions are harmoniously balanced to optimize performance. This involves analyzing the dependencies and interactions of different functions and ensuring they are co-located based on their shared resources or functionality. By doing so, developers can minimize the overhead and latency associated with executing multiple distributed functions. Additionally, as applications scale, it becomes crucial to effectively manage the deployment of functions, ensuring that they remain cohesive and optimized. This includes continuously evaluating the functions, understanding their resource needs, and modifying their deployment strategy to achieve the most efficient arrangement. Considering the potential challenges that come with function sprawl, developers should prioritize the organization and optimization of their serverless architecture to maintain performance and scalability.

Furthermore, the deployment of serverless applications should be done in an automatic, continuous, and efficient way. As the development of a serverless application consists of the implementation of individual Lambda functions along with their configuration, the existence of Terraform or CloudFormation configuration files, including a pipeline description file for CodePipeline, is mandatory. Because of the different nature of serverless applications compared to traditionally

deployed applications, serverless pipelines benefit from being optimized. The entire pipeline execution should be fast, as fast feedback to developers increases the overall development velocity. Additionally, the pipeline should be cost-aware, which implies including a validation stage to ensure that a change does not result in a costly misconfiguration.

6.1. Scaling Up Continuous Integration and Continuous Deployment (CI/CD) Strategy

Modern complex software projects usually rely on multiple teams of developers to create code artifacts that integrate. Often enterprise-level software involves a large number of code artifacts in a single release. This could involve multiple microservices. When considering a serverless deployment of such a release, the number of deployed functions could increase dramatically in a short period of time. Even with short-lived cloud function scaling, this could lead to a denial of service cloud resources attack on the cloud provider if not handled carefully. Regular and predictable releases prepare the ground for continuous delivery. Actual deployment to pre-production and then production is further guaranteed by continuous deployment. All these steps - starting with code integration and ending with production deployment - are automated by the CI/CD pipeline.



Modern complex software projects usually rely on multiple teams of developers to create code artifacts that integrate. Often enterprise-level software involves a large number of code artifacts in a single release. This could involve multiple microservices. When considering a serverless deployment of such a release, the number of deployed functions could increase dramatically in a short period of time. Even with short-lived cloud function scaling, this could lead to a denial of service cloud resources attack on the cloud provider if not handled carefully. Regular and predictable releases prepare the ground for continuous delivery. Actual deployment to pre-production and then production is further guaranteed by continuous deployment. All these steps - starting with code integration and ending with production deployment - are automated by the CI/CD pipeline.

Advantages of the serverless model with its auto-scaling inherent feature are cheapness of operation, close-to-zero cost when having no traffic-ingress, and shorter lifecycle of development for MVPs and other short-duration projects. With such advantages, it is possible to find more clients for development teams even within the enterprise and its shared IT budget. There are, however, additional challenges

in the serverless model that could spoil the serverless benefits. These challenges include performance problems that stem from cold start events of the cloud functions on one hand, and total overspending with popular traffic-ingress due to lack of cloud function scaling control on the other hand. The open issue of the short-lived cloud function is the event loss when stopping a function in the middle of its execution. This could lead to an inconsistency in the data handled by the cloud function.

7. Economic and Business Implications

The selection of a cloud platform to implement the serverless model comes with significant cost considerations. There are regional differences in cloud pricing which impact the cost of running code and obtaining results. These differences also play a role in the selection of computing services within the extensive global data center network of the cloud provider. Additionally, there are pricing variations at the service level for comparable services offered by different providers. The increasing popularity of serverless computing has implications for businesses and the economy at various levels. Independent Software Vendors (ISVs) can benefit from reduced development and operations costs, allowing them to focus on delivering products more quickly and potentially experiencing margin erosion. Cloud platform providers are required to adapt their pricing models to promote more predictable usage of computing resources. On the other hand, enterprises may or may not reap the benefits of the more detailed usage-cost model of serverless computing. Expanding on the points listed above, it is important for businesses to carefully consider the long-term financial implications of choosing a specific cloud platform for serverless computing. The cost variations between regions, as well as the differences in pricing models offered by different providers, can have a significant impact on the overall expenses incurred. It is essential for businesses to analyze the potential cost savings and operational efficiencies that serverless computing can offer, especially for ISVs who heavily rely on software development and delivery. However, enterprises must also evaluate whether the detailed usage-cost model of serverless computing aligns with their specific business needs and operational requirements. Ultimately, the decision regarding the selection of a cloud platform for serverless computing should be based on a comprehensive analysis of the financial, operational, and strategic implications for the business.

Serverless computing abstracts event-driven computing logic at an extreme level, such that cloud users only need to write a piece of code and upload it to a cloud provider in order to have the code executed in response to a variety of events. The execution, scaling, and deployment of code, along with incurring any associated infrastructure costs, are all managed transparently by the cloud provider. All the major cloud providers currently offer a serverless computing platform. In this chapter, we describe a number of architectural patterns of serverless computing, addressing areas such as workflow coordination, data

processing, and long-running applications. For each of these patterns, we delve into code examples using the AWS Lambda and Step Functions services.

7.1. Cost Analysis and Optimization

Serverless computing in cloud environments: architectural patterns, performance optimization strategies, and deployment best practices. Whether it is worthwhile for such small businesses to rewrite their application code to take full advantage of serverless features becomes a question. It is because serverless systems, in their current form, while providing enhancement over microservice architectures deployed on IaaS clouds, are still more costly when the function execution time is significant. Commercial cloud providers offer various enterprise discount programs and provide deep commitments to large businesses to mitigate cost concerns. As the cost of cloud services for small businesses is relatively high, the research community has just begun to develop approaches that consider costs in function-specific deployment decisions. Small businesses are advised to choose the parameters provided by FRED as they can significantly affect function response time and overall cost of the serverless application. Serverless computing in cloud environments: architectural patterns, performance optimization strategies, and deployment best practices. Whether it is worthwhile for such small businesses to rewrite their application code to take full advantage of serverless features becomes a question. It is because serverless systems, in their current form, while providing enhancement over microservice architectures deployed on IaaS clouds, are still more costly when the function execution time is significant. Commercial cloud providers offer various enterprise discount programs and provide deep commitments to large businesses to mitigate cost concerns. As the cost of cloud services for small businesses is relatively high, the research community has just begun to develop approaches that consider costs in function-specific deployment decisions. Small businesses are advised to choose the parameters provided by FRED as they can significantly affect function response time and overall cost of the serverless application.

8. Case Studies and Real-World Applications

There has been a significant amount of serverless tools, frameworks, and platforms developed by both industrial companies and the open-source community. In this chapter, we will provide a thorough overview and review of the popular and representative serverless tools. Specifically, we will describe the serverless tool in terms of its construction, capabilities, and unique features, review the serverless computing platform enabled by the tool, list the typical serverless functions used to address various user requirements, and summarize the reported user experience and comments of the tool. We will also compare and evaluate popular serverless tools through a standard evaluation template with a designated set of criteria. Our analysis will focus on the practical application of these tools and their impact on the development of serverless solutions in the context of modern cloud computing

environments. Furthermore, we will explore the advantages and limitations of each tool, providing actionable insights for developers, architects, and decision-makers seeking to leverage serverless technology in their respective projects. Through this comprehensive examination, we aim to offer a comprehensive resource for understanding, utilizing, and maximizing the potential of serverless tools in contemporary IT environments.

In this chapter, we will report the major use of serverless computing from real-world applications, use cases, and case studies. The serverless computing systems of those use cases and case studies are developed by employing popular serverless tools, performing serverless functions with different optimization strategies, tailoring system deployment with best practices, handling various management issues with effective methods, and evaluating system performance using different approaches. We will provide a detailed description, analysis, and summary of each use case and case study, with the associated serverless computing techniques outlined explicitly. We will also identify the common building blocks, patterns, and features in those serverless computing systems, as well as the recurrent challenges, pitfalls, and lessons when developing, deploying, and managing serverless applications.

8.1. Industry Use Cases

SAP – In-memory ERP platform: SAP has introduced a serverless computing platform called SAP Cloud Platform, Enterprise Functions which operates on a pay-as-you-go pricing model. It is a completely serverless and event-driven platform. SAP's serverless platform offers enterprise functions as independent building blocks. The different types of enterprise functions offered by SAP include intelligent function services, process function services, service mesh, event mesh, and service composition. Each of these enterprise functions performs a specific business or technical function and can be used in a combination as per a specific use case. SAP's serverless platform uses Kyma as the underlying architecture which extends native Kubernetes with backend JavaScript lambda functions (BEB / Serverless Functions) consumed from a custom build enterprise-grade event-driven lightweight overlay API Gateway developed using the Open Source Kyma API Rule Engine Project. The serverless functions powered by Kyma can be developed using the Kyma Serverless Functions project that is a JavaScript function executed on the BEB runtime engine. It also supports the event-driven Run & Rule service – which will execute custom coded JavaScript functions on an opinionated base utilizing the Kyma Event RULES Engine.

SAP uses Kyma to extend its capabilities by building custom microservice extensions connected with BEB / Serverless Functions and their event-driven lightweight overlay API Gateway, to enrich the serverless backend of an OData exposed standard Fiori UI project by consumable enterprise functions by using Kyma Service Catalog to secure and configure standard and enterprise functions mapping

HTTP and OData protocols to backend interfaces on the Kyma API Gateway, and to maintain the Event Mesh infrastructure used to connect the built extensions ensuring enterprise messaging middleware capabilities and serverless eventing services. The Serverless Functions by the Kyma project are aimed at JavaScript and TypeScript applications that are hosted on enterprise-grade event-driven lightweight JavaScript runtimes like the BEB and the e-stereotype. This runtime executes business logic that is independent of the SAP BTP. The serverless functions can be developed using the SAP Cloud SDK, a modular set of libraries along with tools, guidelines, and documentation. The SDK includes libraries related to various concerns like connectivity to SAP BTP services, easy consumption of events, and provisioning services related to leaning upon the Kyma stack for the project such as Service binding and injection of configuration handled using the Kyma environment variables.

BOSCH – IoT and Serverless Computing Platform: Bosch IoT Insights is a serverless computing platform from Bosch.IO which allows users to uncover and predict business inefficiencies. It is a pay-as-you-go solution that gets insights from IoT data-in-motion and data-at-rest and operates without artificial backend time-windows. The Bosch IoT Suite services are built on top of a cloud-native software stack and expose functionality via well-defined RESTful APIs. Data may arrive from diverse sources. Bosch.IO leverages enterprise-grade connectors and exposes data for complex event processing in a BEB or as a microservice using Kyma. Data may also be enriched and normalized using the Bosch IoT Message-Refinery service, which embeds customized rules as BEB functions. The solution's Insight Services, running as BEB functions, empower domain experts to extract value from their data, while Bosch engineers ensure the right data and technology insights triggering hassle-free business decisions. The IoT Insights from Bosch is designed and implemented to connect, enrich, process, and analyze data, and thus help you to get insights from it. Certain cloud-based functions are specifically developed and optimized for a defined purpose and use case. They help to reveal new product and business use-cases, address domain-specific problems, and scale your connected business.

9. Conclusion and Future Directions

The cloud is an ever-changing environment, instantiated by virtualization and supported by IT industry's innovation. Having concept-proven serverless firmly locked in cloud-provider roadmaps gives us confidence. Public FaaP services have experienced a very fast growth phase, and the fast pace of cloud operational enhancements has, in a short time, addressed many of the reported drawbacks of serverless. Given the cloud's shared-data and shared-infrastructure nature, inevitable performance variability will be assumed in best-effort platforms, but research and development will undoubtedly improve resource contention mediation and effective QoS mechanisms. The continuous evolution of the cloud landscape is a testament to the relentless pursuit of technological advancement and innovation in the IT industry. This dynamic environment, driven by virtualization, has established

itself as a key enabler for various industries, providing scalable and efficient solutions to meet the ever-changing demands of the digital era. As organizations continue to leverage the cloud for their operations, the integration of serverless computing has emerged as a pivotal component, offering enhanced flexibility and cost-effective solutions for modern application development and deployment. The rapid expansion of public FaaP services underscores the growing demand for agile and scalable cloud solutions, while the swift evolution of cloud operational enhancements has effectively addressed previous limitations associated with serverless architecture. Moreover, as the cloud's shared-data and shared-infrastructure model inevitably introduces performance variability, ongoing research and development efforts are poised to enhance resource contention mediation and optimize quality of service mechanisms, further solidifying the cloud as a robust and reliable platform for diverse computing needs.

We have tackled serverless computing from the perspective of cloud environments – the natural habitat for serverless platforms. To avoid tedious repetition, let us summarize the considerations that should be taken into account when designing serverless applications as well as the challenges and opportunities. Many serverless applications exhibit concurrent invocations of a single function. Resource usage has to be carefully optimized to avoid overspending and to provide effortful developers with generous performance levels. Current performance optimization strategies are influenced by platform-specific execution-container characteristics and limitations, but will likely converge to uniform approaches as execution containers evolve and the visibility of their resource allocation improves. Serverless has democratized event-driven application design, and wider adoption should lead to best practice patterns for the composition and long-term coexistence of heterogeneous serverless applications.

10. References

1. B. Hellerstein et al., "Serverless Computing: One Step Forward, Two Steps Back," *IEEE Internet Computing*, vol. 21, no. 5, pp. 64-69, Sep.-Oct. 2017.
2. A. Baldini et al., "Serverless Computing: Current Trends and Open Problems," in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, FL, USA, 2017, pp. 232-237.
3. G. Adzic and R. Chatley, "Serverless Computing: Economic and Architectural Impact," in *Proc. 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, Paderborn, Germany, 2017, pp. 884-889.
4. S. Hendrickson et al., "Serverless Computation with OpenLambda," in *Proc. 8th USENIX Conference on Hot Topics in Cloud Computing (HotCloud '16)*, Denver, CO, USA, 2016, pp. 33-39.

5. I. Baldini et al., "The Serverless Trilemma: Function Composition for Serverless Computing," in *Proc. 10th ACM International Systems and Storage Conference (SYSTOR '17)*, Haifa, Israel, 2017, pp. 1-15.
6. P. McGrath et al., "Serverless Computing: Design, Implementation, and Performance," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1365-1378, Oct.-Dec. 2021.
7. E. Jonas et al., "Cloud Programming Simplified: A Berkeley View on Serverless Computing," *arXiv preprint arXiv:1902.03383*, 2019.
8. P. Wolski et al., "Cost Efficiency of Serverless Computing: A Comparative Study," in *Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Nicosia, Cyprus, 2018, pp. 115-122.
9. L. Wang et al., "Peeking Behind the Curtains of Serverless Platforms," in *Proc. USENIX Annual Technical Conference (ATC '18)*, Boston, MA, USA, 2018, pp. 133-146.
10. A. Akkus et al., "SAND: Towards High-Performance Serverless Computing," in *Proc. USENIX Annual Technical Conference (ATC '18)*, Boston, MA, USA, 2018, pp. 923-935.
11. C. Fehling et al., "Patterns for Cloud Computing: Architecting Solutions with AWS," in *Proc. IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC)*, Ulm, Germany, 2014, pp. 100-109.
12. M. Weerawarana et al., "A Platform Architecture for Serverless Video Processing," in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, San Francisco, CA, USA, 2018, pp. 140-150.
13. E. Van Eyk et al., "Serverless is More: From PaaS to Present Cloud Computing," *IEEE Internet Computing*, vol. 22, no. 5, pp. 8-17, Sep.-Oct. 2018.
14. A. Eivy, "Be Wary of the Economics of 'Serverless' Cloud Computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6-12, Mar.-Apr. 2017.
15. G. C. Fox et al., "Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research," *arXiv preprint arXiv:1708.08028*, 2017.
16. A. Rajan et al., "Implementing Microservices with Serverless Computing: A Case Study on Scalability and Cost Efficiency," in *Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Nicosia, Cyprus, 2018, pp. 165-172.
17. T. F. J. Pasquier et al., "Data Provenance to Audit Compliance with Privacy Policy in the Internet of Things," *Personal and Ubiquitous Computing*, vol. 22, no. 2, pp. 333-344, Apr. 2018.
18. K. Figiela et al., "Performance Evaluation of Heterogeneous Cloud Functions," in *Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Hong Kong, China, 2017, pp. 103-110.
19. I. Sbarski et al., "Serverless Architectures on AWS: With Examples Using AWS Lambda," *O'Reilly Media*, 2017.

20. D. Bermbach et al., "Using Application Knowledge to Reduce Noisy Neighbor Impact in Cloud Data Stores," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 1-14, Sep. 2018.