Mini Review
Volume 16:4, 2023

Journal of Computer Science & Systems Biology

ISSN: 0974-7230 Open Access

Serverless Computing in the Cloud: Architectural Patterns, Performance Optimization and Use Cases

Jorge Volpert*
Department of Business Information Systems, University of São Paulo, Butantã, São Paulo, Brazil

Abstract

Serverless computing has emerged as a paradigm-shifting technology in cloud computing, promising scalability, cost-efficiency, and reduced operational overhead. This research article explores the architectural patterns, performance optimization techniques, and real-world use cases of serverless computing in the cloud. We delve into the core concepts of serverless computing, its advantages, challenges, and practical implementations. Through a detailed analysis of architectural patterns and optimization strategies, we provide insights into how organizations can harness the full potential of serverless computing for their applications. Additionally, we present case studies illustrating the diverse range of applications and industries where serverless computing has made a significant impact.

Keywords: Serverless computing • Microservices • Cold start mitigation

Introduction

Serverless computing, often referred to as Function as a Service, is a cloud computing model that abstracts server management and infrastructure concerns from developers, enabling them to focus solely on writing code in the form of functions. This model offers several advantages, including automatic scaling, reduced operational overhead, and cost-efficiency. This article explores the foundational concepts, architectural patterns, performance optimization strategies, and practical use cases of serverless computing in the cloud [1-3]. Serverless computing supports various architectural patterns, enabling developers to design applications that are highly scalable and fault-tolerant.

In this pattern, functions are triggered by events such as HTTP requests, message queues, database changes, or scheduled tasks. This event-driven approach allows for efficient resource utilization and automatic scaling in response to workload fluctuations. Serverless facilitates the implementation of microservices by breaking down complex applications into smaller, independently deployable functions. Each function can serve a specific purpose, and communication between them is typically managed via APIs or event triggers. Serverless functions are inherently stateless, meaning they do not maintain any persistent state. This simplifies scaling and ensures that functions can be executed in parallel without contention for shared resources. To make the most of serverless computing, performance optimization is crucial.

Literature Review

Cold starts occur when a serverless function is initialized, resulting in higher response times for the first request. To mitigate this, developers can use techniques like provisioned concurrency, warm-up functions, or optimizing code and dependencies. Efficiently managing resources, such as memory allocation and function execution time, can impact cost and performance. Careful consideration of these parameters is essential to strike the right balance. Leveraging function composition techniques like chaining or orchestration can help reduce the overhead of invoking multiple functions separately, thus improving performance.

Serverless is well-suited for web applications, particularly for handling API requests, user authentication, and data processing. Companies like Airbnb and Netflix have adopted serverless for parts of their infrastructure. Serverless can process and analyze streaming data from IoT devices, social media, and sensors, making it valuable for real-time analytics and monitoring. Automation tasks, such as continuous integration/continuous deployment pipelines and infrastructure provisioning, benefit from serverless for their event-driven nature. Scalable and cost-effective serverless solutions are used for inventory management, order processing, and recommendation engines in e-commerce platforms [4,5].

Discussion

Healthcare applications leverage serverless for processing patient data, managing appointments, and ensuring compliance with healthcare regulations. While serverless computing offers numerous benefits, it also presents challenges related to vendor lock-in, limited execution time, and increased complexity in debugging and monitoring. Event Sourcing is a software architectural pattern for capturing and storing all changes to an application's state as a sequence of immutable events. These events represent the facts about how the application's state has evolved over time. Instead of storing the current state of an object or entity, as is typical in a traditional relational database, Event Sourcing stores a historical log of events that can be replayed to reconstruct the current state.

Events are immutable records that represent state changes in an application. These events typically contain data about what changed, when it changed, and any relevant metadata. Events are appended to an event log as they occur. The event log is a central component of Event Sourcing. It's a sequential record of all events in the system. Each event is appended to the end of the log, and events are never updated or deleted. This log serves as the source of truth for reconstructing the application's state. An aggregate is a domain-driven design pattern that represents a cluster of related data and the logic to manage that data. Aggregates are responsible for handling commands, validating them, and producing events in response. Commands are requests to change the state of an aggregate. When a command is received, the aggregate processes it, potentially generates one or more events, and emits those events to the event log. Commands are mutable and can be rejected if they violate business rules [6].

Event handlers are responsible for updating the read models or projections of the application. They listen for events in the event log and update the query-side models accordingly. These models are used to retrieve current state efficiently. When a user interacts with the application, such as creating an order or updating

***Address for Correspondence:** Jorge Volpert, Department of Business Information Systems, University of São Paulo, Butantã, São Paulo, Brazil, E-mail: martinbartolini3@gmail.com*

Received: 01 July, 2023, Manuscript No. jcsb-23-113759; **Editor Assigned:** 03 July, 2023, Pre QC No. P-113759; **Reviewed:** 17 July, 2023, QC No. Q-113759; **Revised:** 22 July, 2023, Manuscript No. R-113759; **Published:** 31 July, 2023, DOI:10.37421/0974-7230.2023.16.480

a profile, the application generates events that represent these actions. These events are appended to the event log. To retrieve the current state of an entity or object, you replay the events from the event log. By starting with an initial state and applying each event in sequence, you can reconstruct the current state of the entity. For querying and displaying data, you maintain one or more projections that are updated in real-time or through batch processes as events are recorded. These projections provide efficient ways to access the current state without replaying all events each time. It provides a complete history of how the application's state has changed, making it valuable for auditing and compliance purposes. Developers can analyze past states of the application by replaying events up to a specific point in time. It enables scalability by allowing different components to subscribe to specific events and react to them independently. Since events are immutable, it's easier to recover from failures and rebuild state.

However, Event Sourcing is not without challenges. It can introduce complexity, especially in distributed systems, and may require careful consideration of event versioning, schema evolution, and event storage. Event Sourcing is a powerful architectural pattern that can be valuable in scenarios where a complete history of state changes is required, such as financial systems, e-commerce platforms, and systems with complex business rules. It's a way to capture the full story of how an application's state evolves over time.

Conclusion

Serverless computing in the cloud has emerged as a transformative technology with its architectural patterns, performance optimization strategies, and diverse use cases. By abstracting infrastructure management, it enables organizations to focus on building scalable and cost-efficient applications. However, understanding the nuances of serverless, optimizing performance, and addressing challenges are essential for successful adoption. As serverless continues to evolve, it will likely play an increasingly central role in the future of cloud computing.

Acknowledgement

None.

Conflict of Interest

Authors declare no conflict of interest.

References

1. Chen, Xiaokai, Hao Lei, Rui Xiong and Weixiang Shen, et al. "A novel approach to reconstruct open circuit voltage for state of charge estimation of lithium ion batteries in electric vehicles." *Appl Energy* 255 (2019): 113758.
2. Luo, Xuan, Longyun Kang, Chusheng Lu and Jinqing Linghu, et al. "An enhanced multicell-to-multicell battery equalizer based on bipolar-resonant LC converter." *Electronics* 10 (2021): 293.
3. Beloglazov, Anton, Jemal Abawajy and Rajkumar Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing." *Future Gener Comput Syst* 28 (2012): 755-768.
4. Mrabet, Hichem, Sana Belguith, Adeeb Alhomoud and Abderrazak Jemai. "A survey of IoT security based on a layered architecture of sensing and data analysis." *Sensors* 20 (2020): 3625.
5. Grošek, Otokar, Viliam Hromada and Peter Horák. "A cipher based on prefix codes." *Sensors* 21 (2021): 6236.
6. Košťál, Kristián, Pavol Helebrandt, Matej Belluš and Michal Ries, et al. "Management and monitoring of IoT devices using blockchain." *Sensors* 19 (2019): 856.

How to cite this article: Volpert, Jorge. "Serverless Computing in the Cloud: Architectural Patterns, Performance Optimization and Use Cases." *J Comput Sci Syst Biol* 16 (2023): 480.