IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES
ISSN PRINT 2319 1775 Online 2320 7876
 UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

COMPARISION AND SURVEY OF SERVERLESS TECHNOLOGY WITH CHALLENGES AND SOLUTION

Mr. Rahul D Thaokar
Asst. Professor
MCA Department
VMV Commerce, JMT Arts And JJP Science College
Wardhman Nagar, Nagpur
thaokar.rahul1@gmail.com

Dr. Vaibhav R Bhedi
Asso. Professor
MCA Department
VMV Commerce, JMT Arts And JJP Science College
Wardhman Nagar, Nagpur
vaibhav_bhedi@rediffmail.com

ABSTRACT

Serverless computing, a paradigm shift in cloud computing, has won significant traction because of its promise of abstracting the underlying infrastructure, permitting developers to recognition totally on writing code. This paper presents a complete survey and comparative evaluation of the leading serverless technology, highlighting their structure, performance, and scalability. This paper goals to manual corporations in correctly leveraging serverless technology for their cloud-primarily based programs.

KEYWORDS

Survey, Cloud computing, Serverless computing, Serverless platforms, Serverless benefits, Serverless challenges.

INTRODUCTION

In recent years, serverless computing has emerged as a transformative paradigm in the cloud computing landscape, offering a new approach to building and deploying applications without the need for managing underlying server infrastructure. Unlike traditional cloud models, where developers must provision and maintain servers, serverless computing abstracts these concerns, allowing developers to focus exclusively on writing code. This shift has led to increased agility, scalability, and cost-efficiency in software development.[1][2][3]

Serverless computing, also known as Function as a Service (FaaS), enables developers to deploy discrete functions that automatically scale in response to demand. The serverless model is event-driven, meaning functions are triggered by specific events, such as HTTP requests or changes in data storage. This model has gained popularity due to its promise of reducing operational complexity and lowering costs by charging only for actual execution time.[3]

Despite its benefits, the adoption of serverless computing is not without challenges. Issues such as cold start latency, where functions experience delays during initialization, vendor lock-in due to platform-specific implementations, and difficulties in debugging and monitoring distributed functions, have been significant concerns for organizations. Additionally, security vulnerabilities unique to serverless environments, such as increased attack surfaces due to the granularity of functions, pose further challenges.[3][7][8]

This paper aims to provide a comprehensive survey and comparative analysis of leading serverless computing platforms, including AWS Lambda, Azure Functions, Google Cloud

Functions, and various open-source alternatives. By examining the architecture, performance, and scalability of these platforms, the paper seeks to identify their respective strengths and weaknesses.[4][5][6]

Moreover, the study will delve into the key challenges associated with serverless computing and propose potential solutions to mitigate these issues. The goal is to equip organizations and developers with the knowledge needed to effectively navigate the serverless landscape, enabling them to make informed decisions when selecting a serverless platform and implementing best practices to overcome common obstacles.[4][5][6]

The paper will first provide an overview of serverless computing, followed by a detailed comparison of the major serverless platforms. The discussion will then focus on the primary challenges faced in serverless computing, offering practical solutions and strategies. Finally, the paper will conclude with recommendations for organizations looking to adopt serverless technologies, emphasizing the importance of a well-considered approach to leveraging this powerful paradigm.

TECHNOLOGIES USED IN SERVERLESS COMPUTING

Serverless computing leverages a combination of cloud services, event-driven architectures, and containerization technologies to deliver its promise of infrastructure abstraction and dynamic scaling.

1. ***Function as a Service (FaaS)***: FaaS is the core of serverless computing, enabling developers to write individual functions that are executed in response to events. These functions are stateless and run in isolated environments.[16][17][18]
2. ***Backend as a Service (BaaS)***: BaaS provides ready-to-use backend services such as databases, authentication, and file storage, which can be accessed via APIs. These services complement FaaS by handling backend operations, allowing developers to focus on front-end and business logic.[19][20]
3. ***Containerization:*** Containers encapsulate applications and their dependencies, providing a consistent runtime environment. In serverless, containers are often used to isolate functions, ensuring that they run consistently across different environments.[21][22]
4. ***Event-Driven Architecture***: Serverless computing is inherently event-driven, meaning that functions are triggered by specific events such as HTTP requests, database updates, or messaging queues.[23][24]
5. ***API Gateway***: API Gateways are used to manage, secure, and scale APIs that serve as the front end to serverless functions. They handle tasks such as routing requests, rate limiting, and authentication.[25][26]
6. ***Monitoring and Logging Tools:*** Effective monitoring and logging are crucial in serverless environments to track function performance, diagnose issues, and optimize resource usage.[27][28]
7. ***Security and Identity Management***: Security is a critical aspect of serverless applications, involving the management of access controls, encryption, and identity.[29][30]

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

COMPARISON OF TRADITIONAL VS SERVERLESS COMPUTING

1. ***Infrastructure Management***

***Traditonal Computing*:**

In traditional cloud computing, developers and system administrators are responsible for provisioning, configuring, and maintaining servers, whether they are virtual machines or physical servers.

Provides greater control over the infrastructure, including the ability to customize configurations and manage resources directly.

***Serverless Computing*:**

Serverless computing abstracts away the underlying infrastructure management. Developers deploy functions or code snippets, while the cloud provider manages the servers and scaling automatically.

Limited control over the infrastructure, focusing instead on the functionality and business logic of the application.[1][2][3][8][9]

2. ***Scalability***

Traditional Computing:

Scaling in traditional environments often requires manual intervention to add or remove servers or adjust resource allocations. This can be complex and time-consuming.

Usually requires planning and provisioning ahead of time for anticipated load, which may lead to over-provisioning or under-provisioning.

Serverless Computing:

Serverless platforms automatically handle scaling in response to demand, scaling functions up or down based on the volume of incoming events or requests.

Highly elastic and can scale seamlessly without manual intervention, adapting to real-time workload changes[1][2][3][8][9]

3. ***Cost***

***Traditional Computing*:**

Typically involves paying for reserved instances or virtual machines on an hourly or monthly basis, regardless of actual usage.

Costs can be higher due to over-provisioning or idle resources, as payment is based on allocated capacity rather than actual usage.

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

Serverless Computing:

Costs are based on the actual execution time and resources consumed by functions, following a pay-as-you-go model.

Can be more cost-effective, as you only pay for the time your code is running, and there is no charge for idle time.[1][2][3][8][9]

4. ***Development Complexity***

Traditional Computing:

Developers must manage the full stack, including servers, databases, and networking. This can add complexity and require more extensive knowledge of infrastructure.

Typically involves setting up environments, configuring servers, and deploying applications, which can be time-consuming.

Serverless Computing

Simplifies development by allowing developers to focus on writing code rather than managing infrastructure. Functions are deployed independently and can interact with other services via APIs.

Deployment is often streamlined, involving just the function code and configuration, which can accelerate the development cycle. [1][2][3][8][9]

5. ***Performance and Latency***

Traditional Computing

Performance is generally more predictable as applications run on dedicated or reserved resources.

Can be optimized based on the specific infrastructure and configuration.

Serverless Computing:

May experience variability due to cold start latency, where functions can experience delays during initialization after being idle.

Generally acceptable for most use cases but can be impacted by cold starts and other platform-specific factors.[1][2][3][8][9]

6. ***Monitoring and Debugging***

Traditional Computing:

Typically involves configuring monitoring tools to track server and application performance. Provides detailed visibility into the infrastructure.

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

Debugging can be more straightforward as you have direct access to the server environment and logs.

Serverless Computing:

Often requires integration with cloud-native monitoring tools or third-party solutions to track function performance and usage.

Can be more challenging due to the distributed nature of serverless functions and limited access to underlying infrastructure.[1][2][3][8][9]

7. Security

Traditional Computing:

Provides control over security configurations at the server level but requires ongoing management and maintenance to address vulnerabilities.

Security practices must be implemented and managed directly by the organization.

Serverless Computing:

The cloud provider manages some aspects of security, but the granularity of functions can increase the attack surface. Security practices must be adapted to the serverless model.

Compliance responsibilities are shared between the provider and the user, with the provider handling some security aspects.[1][2][3][8][9]

LIMITATIONS AND FUTURE WORK

Warm Queues

The warm queue is a FIFO queue, which is problematic for container expiration. Imagine a function under heavy load has 10 containers allocated for execution, and then load drops such that a single container could handle all of the function's executions. Ideally, the extra 9 containers would expire after a short time, but because of the FIFO queue, so long as there are 10 executions of the function per container expiration period, all containers will remain allocated to the function. Of course, the solution is to use "warm stacks" instead of "warm queues", but Azure Storage does not currently support LIFO queues. This is perhaps the largest issue with our current implementation; however, other warm stack storage options such as a Redis cache [27] or a consistent hashing [28] implementation are promising, and may offer improved performance as well.

Asynchronous Executions

Currently the prototype only supports synchronous invocations. In other words, a request to execute a function will return the result of that function execution, it will not simply start the function and return. Asynchronous executions by themselves are simple to support, the web service can simply respond to the invocation call and then process the execution request normally. The difficulty in asynchronous execution is in guaranteeing at-least-once execution rather than best effort execution. It is important to understand that synchronous or asynchronous execution is only guaranteed once an invocation request returns with a

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

successful status code. Therefore, no further work is needed for synchronous execution requests (as in our implementation), because a successful status code is only returned once execution has completed. However, asynchronous executions respond to clients before function execution, so it is necessary to have additional logic to ensure these executions complete successfully. We believe the prototype can support this requirement by storing active executions in a set of queues and introducing a third service responsible for monitoring the status of these queue messages. Worker services would continually update message visibility delays during function execution, and the monitoring service would detect failures by looking for visible messages. Failed messages could then be re-executed. Note that this is about handling platform execution failures and not exceptions thrown by the function during execution, for which retry may also be desired.

Worker Utilization

A large area for improvement in our implementation is worker utilization. Realistic designs would require an over allocation of worker resources, with the observation that not all functions on a worker are constantly executing, or using all of their memory reservation. Utilization in a serverless context presents competing tradeoffs between execution performance and operating costs; however, the evaluation of utilization strategies is difficult without representative datasets of execution loads on serverless platforms. Future research would benefit from increased transparency from existing platforms, and from methods of synthesizing serverless computing loads.

Windows Containers

Windows containers have some limitations compared to Linux containers, largely because Linux containers were designed around Linux groups which support useful operations not available on Windows. Most notably in the context of serverless computing is the support of container resource updating and container pausing. A common pattern in serverless platform implementations is pausing containers when idle to prevent resource consumption, and then unpausing them before execution resumes [29], [30].

Another potentially useful operation is container resource updating. Because we reserve resources for containers before executions begin, it would be beneficial for cold start performance if we were able to start containers before they are assigned to a function, and then resize the container once an execution request is received. Future work can study how to support these semantics in Windows containers, perhaps by limiting or updating the resources to the function process itself rather than the container as a whole. Alternatively, the prototype could experiment with Linux containers to compare startup performances and test the viability of container resizing during cold starts.

Security

Security of serverless systems is also an open research question. Hosting arbitrary user code in containers on multitenant systems is a dangerous proposition, and care must be taken when constructing and running function containers to prevent vulnerabilities. This intersection of remote procedure calls (RPC) and container security represents a significant realworld test of general container security. Therefore, although serverless platforms are able to carefully craft

ISSN PRINT 2319 1775 Online 2320 7876
Research paper UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

the function containers and restrict function permissions arbitrarily, increasing the chances of secure execution, further study is needed to assess the attack surface within function execution environments

Performance Measures

There are significant opportunities to expand understanding of serverless platform performance by defining performance measures and tests thereof. This work focused on the overhead introduced by the platforms during single-function execution, but the quality of these measurements can be improved by better handling of network latencies and clocking considerations. Other aspects of platform performance such as latency variations between language runtimes and function code size, system-wide performance of serverless platforms, performance differences between event types, and CPU allocation scaling also warrant study.

CONCLUSION

Serverless computing offers powerful, event-driven integrations with numerous cloud services, simple programming and deployment models, and fine-grained scaling and cost management. Driven by these benefits, the growing adoption of serverless applications warrants the evaluation of serverless platform quality, and the development of new techniques to maximize the technology's potential. The performance results of our platform are encouraging and our analysis of the current implementation presents many opportunities for continued development and study. We hope to see increased interest in serverless computing by academia and increased openness by the industry leaders for the wider benefit of serverless technologies.

References:

1. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ...& Zheng, H. (2017). Serverless computing: Current trends and open problems. In Research Advances in Cloud Computing (pp. 1-20). Springer, Singapore.
2. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ...&Stoica, I. (2019). Cloud programming simplified: A Berkeley view on serverless computing. arXiv preprint arXiv:1902.03383.
3. McGrath, G., & Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance. 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 405-410). IEEE.
4. AWS Lambda. (n.d.). Amazon Web Services.https://aws.amazon.com/lambda/
5. Azure Functions. (n.d.). Microsoft Azure.
6. Google Cloud Functions. (n.d.). Google Cloud.
7. Hendrickson, S., Stancevic, M., & Farley, B. (2016). Serverless computing: One step forward, two steps back. Proceedings of the 9th Workshop on Hot Topics in Cloud Computing (HotCloud).
8. Eivy, A. (2017). Be wary of the economics of "serverless" cloud computing. IEEE Cloud Computing, 4(2), 6-12.
9. Varghese, B., & Mohan, S. (2020). Serverless computing: A survey of opportunities, challenges and applications. arXiv preprint arXiv:2006.08875.

ISSN PRINT 2319 1775 Online 2320 7876
 UGC CARE Listed (Group -I) Journal Volume 10 ,S Iss 3, 2021

10. Oakes, E., Yang, L., Zhou, W., Hou, K., Ceze, L., Ports, D. R., & Howell, J. (2018). SOCK: Rapid task provisioning with serverless-optimized containers. Proceedings of the 2018 USENIX Annual Technical Conference (pp. 57-70).
11. Sbarski, P. (2017). Serverless architectures on AWS: with examples using AWS Lambda. Manning Publications Co.
12. AWS Lambda: https://aws.amazon.com/lambda/
13. Azure Functions: https://azure.microsoft.com/services/functions/
14. Google Cloud Functions: https://cloud.google.com/functions
15. Firebase: https://firebase.google.com/
16. AWS Amplify: https://aws.amazon.com/amplify/
17. Docker: https://www.docker.com/
18. Kubernetes: https://kubernetes.io/
19. Amazon S3: https://aws.amazon.com/s3/
20. Azure Event Grid: https://azure.microsoft.com/services/event-grid/
21. Amazon API Gateway: https://aws.amazon.com/api-gateway/
22. Azure API Management: https://azure.microsoft.com/services/api-management/
23. AWS CloudWatch: https://aws.amazon.com/cloudwatch/
24. Azure Monitor: https://azure.microsoft.com/services/monitor/
25. AWS Identity and Access Management (IAM): https://aws.amazon.com/iam/
26. Azure Active Directory (AAD): https://azure.microsoft.com/services/active-directory/
27. Redis, “Redis,” Available: https://redis.io/, 2017.
28. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ser. SIGCOMM ’01. New York, NY, USA: ACM, 2001, pp. 149–160.
29. S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Serverless computation with openlambda,” in Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, ser. HotCloud’16. Berkeley, CA, USA: USENIX Association, 2016, pp. 33–39.
30. T. Wagner, “Understanding container reuse in AWS Lambda,” Available: https://aws.amazon.com/blogs/compute/container-reuse-in-lambda/, 2014.