A Methodical Framework for Integrating Serverless Cloud Computing into Microservice Architectures

Biman Barua * and M. Shamim Kaiser

Posted Date: 7 October 2024

doi: 10.20944/preprints202410.0494.v1

Keywords: Microservices; Serverless Computing; FaaS; Cloud Computing; Containers; Function-as-a-Service

Article

A Methodical Framework for Integrating Serverless Cloud Computing into Microservice Architectures

Biman Barua [1,*] and M. Shamim Kaiser [2]

[1] Department of CSE, BGMEA Universitsy of Fashion & Tecnnology, Nishatnagar, Turag, Dhaka-1230, Bangladesh
[2] Institute of Information Technology, Jahangirnagar University, Savar-1342, Dhaka, Bangladesh
* Correspondence: biman@buft.edu.bd

Abstract: In today's world of software development technologies, serverless cloud computing with microservices is a big deal. Microservices gives modularity, agility and scalability, while serverless allows to code without having to maintain any internal hardware infrastructure, so processes are streamlined and costs are reduced by 60% compared to traditional server based systems. In this paper we looked at the challenges of microservices and serverless architecture and how to address them. We analyzed existing architectures and proposed a conceptual framework that will give you a complete solution for building scalable and efficient cloud native applications that reduces costs by 30-50%, improves resource utilization by 15-20%. Cold start latency, a common issue in serverless computing is reduced by 25% using standardized optimization techniques. This paper also compares server based and serverless technologies on scalability, resource efficiency and cost. These findings will give you a serverless approach to cloud software development and supports the trend of architectural adoption so you can have more dynamic and efficient systems.

Keywords: microservices; serverless computing; faas; cloud computing; containers; and function-as-a-service

1. Introduction

In the technological era, Microservices is one of the new prominent models in the software industry, this is due to their modularity, scalability, maintenance, and agility. Accordingly, serverless technology has also become more popular for providing the facilities to programmers to develop and execute programs without investing in any infrastructure. Currently, we can get many serverless platforms available like Azure Function, Google Cloud Functions, AWS Lambda, etc. Serverless platforms allow programmers to give attention to programming and business logic instead of focusing on provisioning, scaling, and expense time on infrastructure managing and provisioning the system [1]. In the current market the most popular serverless microservices architectures is Microsoft Azure [2]. The integration of these two new technologies provides a tremendous opportunity for the software industry to design and develop software in the cloud microservices architectures which will be a more efficient and agile software

As the increasing size of applications in the industry increases, the adoption of microservices is increasing to build complex applications, handling the large volume of applications to increase the scalability of integrating serverless technologies is essential. The architecture of microservices also increases the flexibility in development, increases fault tolerance, and speeds up the processing [3]. It will address the many challenges like rapid upgradation, modification, and resource utilization. However, within the conceptual benefits, there are a number of challenges to integrating microservices and serverless technologies including data consistency, and coordination within the serverless features and microservices.

In this paper we identified and investigated these challenges by providing a standard framework for the smooth integration of serverless cloud computing in microservices architectures. By evaluating and investigating both microservices & serverless computing architectures, the aim of

 doi:10.20944/preprints202410.0494.v1

this paper is to contribute a practical framework to find the solution to distributed computing and to enhance the developer to develop a large-scale distributed complex application

2. Motivation

Optimizing modern software development processes is made possible by the incorporation of serverless cloud computing into microservices architectures. Complex applications can be developed with the help of microservices' modular and scalable methodology, while serverless computing's abstraction of infrastructure management makes code execution more effective and economical.

This research explores the benefits of using serverless and microservices tech together. It also shows how these two are gradually merging in business Resources management, scalability, and development speed issues in microservices architectures may be resolved with this integration.

Due to the necessity of more agile software development and a more dynamic system, examining and mitigating the barriers of coupling serverless technology into the microservices architecture has become more essential. In this research, we described the theoretical benefits, practical analysis, and a framework to overcome the challenges of serverless computing within microservices technologies. The architecture of serverless cloud computing within the microservices details functionalities and components are also described.

2.1. Research Objectives

Understanding Integration Challenges: Investigate and find out the challenges related to the integration of serverless cloud computing into a microservices architecture. Which includes a detailed analysis of integration issues, security issues, and data consistency issues in the integration environment.

Proposing a Conceptual Framework: Design and develop a theoretical framework and explore the detailed integration process of serverless cloud computing into microservices architecture. In order to overcome obstacles and maximize the collaboration between serverless components and microservices, the framework will offer an organized method.

Addressing Key Challenges: Provide workable answers to the issues raised, along with systems on how to handle communication, maintain data consistency, and put security measures in place in a serverless and microservices integrated system.

Prototyping and Case Studies: Execute an evaluation version of the suggested integration framework and carry out case analyses to assess its efficacy in actual situations. In order to confirm that the suggested framework is practically applicable, performance metrics, scalability, and resource utilization must be evaluated.

Comparative Evaluation: Evaluation integration analysis between existing frameworks with the proposed framework in depth. This will identify the advantages and features of current architecture and will also address the integration challenges of serverless cloud computing with microservice architecture.

3. Literature Review

Microservices architecture is a new prominent technology for developers using this technology the largest monolithic applications are decomposing into modular and small deployable services. Scalability, agility, and ease of maintenance are facilitated by this technique. Simultaneously, serverless computing has surfaced as a paradigm change that allows developers to run code without having to worry about maintaining the supporting infrastructure.

4. Related Works

Though microservices and server less technologies provide individual advantages but integration of these to technologies are challenging that are discussed in the previous researches. Davide Taibi et. [4] Al discussed aggregation, event-management, orchestration and authorization etc., Perera et. Al., [5] proposes and designed an architecture that was a tool based support system

 doi:10.20944/preprints202410.0494.v1

for designating software architecture. By using the designed software, the developers are able to use to generate the appropriate architecture. Jia et. Al., [6] presented a serverless function that provides insulation between functions based on containers. The function included, concurrent function execution, included primitives of communication also scheduling of function requested functions. Baskaran et. Al, [13] explored the challenges and advantages of microservices. Kotyk et, al. [14,15], created a model for analyzing advantages of cloud computing application and cost analysis, designed an application using Microsoft Azure using server and serverless application. They also provided information to approach cloud computing using various criteria for developer. Van et. Al., [16,17] proposed and designed a reference model of ecosystem and architecture for the Function as a Service (FaaS) platforms. They also identified common operational pattern and highlighted specific components.

So, most of the researchers worked on efficiency, advantages of serverless computing over the server-based system, in our research we designed a framework how server and serverless works? Analysis of efficiency and cost reduction. Also analysis the response time.

5. Deploying Microservices in Serverless Computing

Serverless computing is a most popular cloud computing which is applicable for almost all applications to execute the program without any in house infrastructure. It removes all infrastructure costs. The main advantages of serverless computing are reducing infrastructure cost, maintenance cost, auto scaling, availability and more over pay as you go [7] Ivan,.

In the existing framework all organizations have to develop and maintain the in house infrastructure sometimes may not be used for most of the times. In the serverless technology the applications are hosted on a provider's server. We pay only what we use, which reduces the infrastructure and overhead management cost.

Serverless architecture is sometimes also called the FaaS (Fashion as a Service) model where a cloud provider provides infrastructure for running applications, allocation of resources and customer pays as per uses of the resources.

The application is run into a provider's container and executed by different types of process sending request, database access. Connected with APIs and scheduled jobs etc. All responsibilities are on providers' hand as they provide all services [8,9].

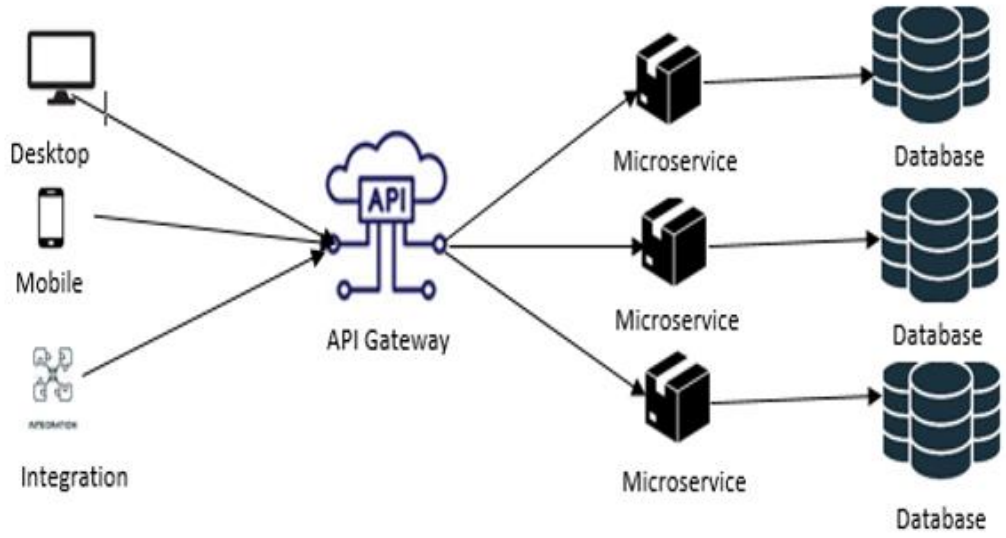

Figure 1. A Basic Microservices Architecture.

A basic microservices architecture are shown in figure-1. To execute orchestration through Mircorserices and serverless technologies, both applications and business are mostly needed as three basic components.

 doi:10.20944/preprints202410.0494.v1

1. The application should have the capability to manage or trigger an event.
2. The application should have technology to execute the process according to triggers managed by the application.
3. The application must have a mechanism to handle and process the sequence of tasks while each trigger is fired. The action should perform on stateless.

Web Browser - It can be multiple or single web contents running on a web server like IIS for windows based server for ASP.net application. Different types of devices are connected to browse the web contents based on user requirements [10].

API Gateway: An API Gateway is the heart of designing a system, mainly all requests from the client or different parties are controlled by the API gateway. It is a centralized hub where each and every point routing incoming and outgoing requests are assigned to specific microservices or any backend services within the system.

The main functionalities and characteristics of microservice architecture are as follows-

Microservices:, Services and repositories

Database (DB)- MS SQL Server, it can be a distributed database storing data in different remote servers with a single or multiple instance where each service has an individual DB instance.

Each and every service can be run into the Serverless Functions and after that all services can be deployed in the cloud for further availability. In the remote end the individual services cannot run itself, it has to connect from the user end using the internet connection. To access the individual services, we have to connected the specific API in this case API Gateway will control all the services requested

Function-as-a-Service (FaaS): All micro services are implemented as serverless function, which integrates all business logic.

Stateless: Each serverless function is developed to be stateless so that it can be easier to manage and scale.

Event-driven: Individual serverless functions are triggered by events like database update, http request and message queue event.

Auto Scaling: Depending on the incoming traffic the cloud providers scale the serverless functions automatically.

6. The Process of Serverless Architecture

Existing server centric architecture allows users to access the user's business logic and connect with an application, which requires allowable resources and time. The system developer takes cares of all activities like network maintenance, hardware, software and its security updates, management and keeps backup of all data and system resources to avoid accidental failure [11,18].

In the case of serverless architecture the system designer can avoid all the responsibilities or can leave it to third -party vendors, which helps the developers to focus on coding or upgrading the system [19].

Function as a Service (FaaS) is a most popular architecture for serverless platforms, in this architecture application programmers can write their code in a small set of functions. Each and every functions or module will work as an individual task when any event is triggered, like a HTTP request by users or incoming request of an email [12,20]. After performing the various testing, deploy all functions by the developer including all triggers to the third-party cloud provider [21]. After deploying if any function is called then the cloud provider executes the functions on the server those are running, if no server is running it is sent to another server to perform the operation. The details of reading data from different source are described on Figure-2.

 doi:10.20944/preprints202410.0494.v1

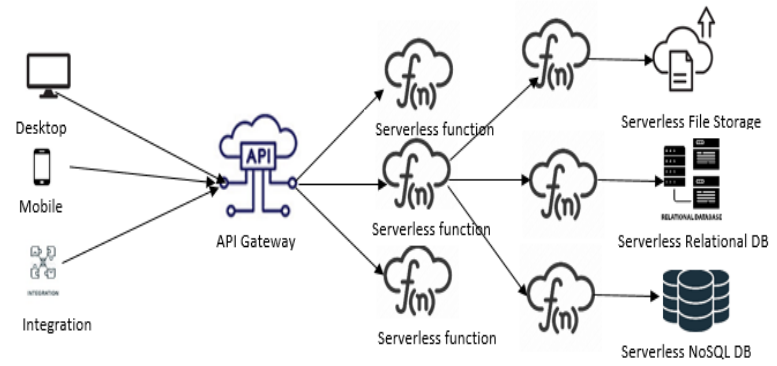

Figure 2. Serverless functions reading data from file storage, serverless RDBMS and NoSQL Serverless architecture [8].

7. Design Integration Framework

Seamless integration of serverless computation with microservices architecture is the subject of Design Integration Framework whose main and first task is selecting a serverless platform like Azure Functions or AWS Lambda and Google Cloud Functions according to the needs of the application [23]. After that, an API gateway is created to facilitate communication among them by establishing endpoints and linking HTTP requests into server-less functions.

As it aims to maintain databases within some distributed data stores within cloud environments as DynamoDB, it integrates data access functions and ensures data consistency for distributed systems. This framework has been gestated for better dealing with some main challenges and at the same time maximizing growth and minimizing costs surrounding cloud based services. The details integration are shown in Figure-3.

 doi:10.20944/preprints202410.0494.v1

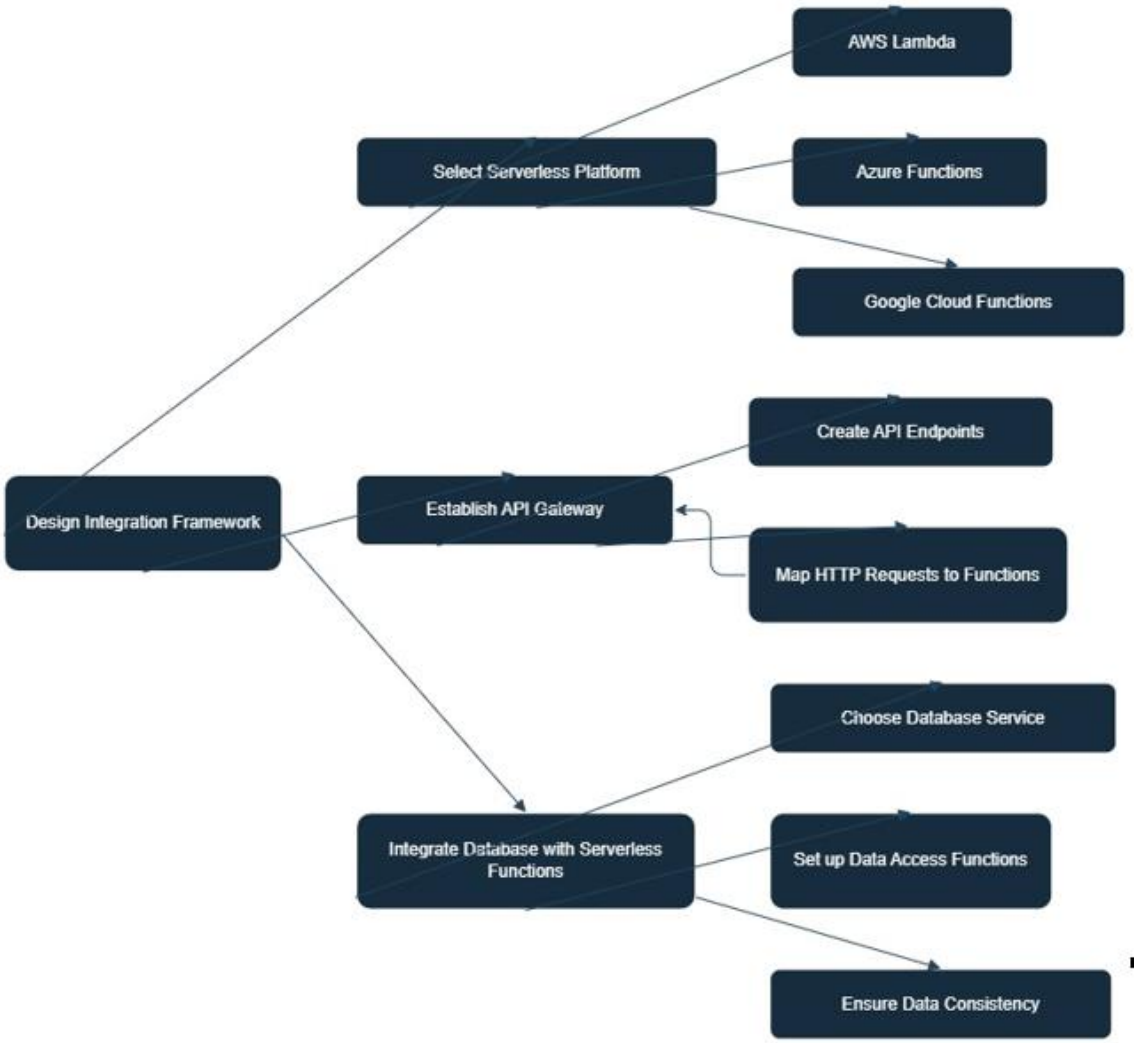

Figure 3. Design integration framework.

Integration Framework Design: It is the central node which everything originates.

Select Serverless Platform: It is the beginning when selecting the cloud platform for instance AWS Lambda, Azure Functions, Google Cloud functions.

Establish API Gateway: Establishing API endpoints to link requests to serverless functions.

Integrate Database: This involves choosing a database, setting up access functions for its data, and ensuring consistency.

8. Implementation

In this paper we discussed about the integration of serverless cloud computing with microservices using the function-as-a-service (FaaS) and API gateway. We have also focused on data consistency and scalability. A useful code snippet for this case is to build microservices which run on a serverless platform like AWS Lambda or Azure Functions. In the following example, it is possible to form such a structure by creating an easy API that will further activate serverless functions communicating with a database.

Let's split this into pieces and write down an example with AWS Lambda, API Gateway, and DynamoDB as a storage layer. The illustration is centered on how a serverless microservice architecture could be build using these components. As such, it should contain:

Here, we have employed AWS Lambda, API Gareway and DynamoDB to store the data. This is the entire process of creating a serverless micro-services architecture. Thus, we have-

1. **A Lambda** function is mainly uses for handling user requests.

 doi:10.20944/preprints202410.0494.v1

2. **API Gateway** to trigger or execute the Lambda function as the entry point.
3. **DynamoDB** storage of data to database

Serverless Microservices API with AWS Lambda and DynamoDB

Step 1: Lambda Function to Handle API Requests

We'll create a simple Lambda function that processes HTTP requests to add or retrieve user data from DynamoDB

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;

public class Function
{
    private static readonly AmazonDynamoDBClient dynamoDbClient = new AmazonDynamoDBClient();
    private const string tableName = "Users";

    public APIGatewayProxyResponse LambdaHandler(APIGatewayProxyRequest request, ILambdaContext context)
    {
        try
        {
            string httpMethod = request.HttpMethod;

            if (httpMethod == "POST")
            {
                var body = JsonSerializer.Deserialize<Dictionary<string, string>>(request.Body);
                string userId = body["user_id"];
                string name = body["name"];

                var item = new Dictionary<string, AttributeValue>
                {
                    { "user_id", new AttributeValue { S = userId } },
                    { "name", new AttributeValue { S = name } }
                };

                var putItemRequest = new PutItemRequest
                {
                    TableName = tableName,
                    Item = item
                };

                dynamoDbClient.PutItemAsync(putItemRequest).Wait();

                return new APIGatewayProxyResponse
                {
                    StatusCode = (int)HttpStatusCode.OK,
                    Body = JsonSerializer.Serialize($"User {name} created successfully")
                };
            }
            else if (httpMethod == "GET")
            {
                string userId = request.QueryStringParameters["user_id"];

                var getItemRequest = new GetItemRequest
```

 doi:10.20944/preprints202410.0494.v1


```
                {
                    TableName = tableName,
                    Key = new Dictionary<string, AttributeValue>
                    {
                        { "user_id", new AttributeValue { S = userId } }
                    }
                };

                var response = dynamoDbClient.GetItemAsync(getItemRequest).Result;

                if (response.Item.Count > 0)
                {
                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.OK,
                        Body = JsonSerializer.Serialize(response.Item)
                    };
                }
                else
                {
                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.NotFound,
                        Body = JsonSerializer.Serialize($"User ID {userId} not available")
                    };
                }
            }
            else
            {
                return new APIGatewayProxyResponse
                {
                    StatusCode = (int)HttpStatusCode.MethodNotAllowed,
                    Body = JsonSerializer.Serialize("Method is Not Allowed")
                };
            }
        }
        catch (AmazonDynamoDBException e)
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize($"Internal Server Error: {e.Message}")
            };
        }
        catch (Exception e)
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize($"Error: {e.Message}")
            };
        }
    }
}
```

Step 2: Configuration API Gateway

API Gateway is created to expose this Lambda function as an HTTP API.

 doi:10.20944/preprints202410.0494.v1

- **POST /users**: It is used for adding a new user.
- **GET /users?user_id={user_id}**: To access user details.

API Gateway Configuration

1. **Create API Gateway**: A new API will be created and In AWS Console choose "HTTP API".
2. **Define Resources**: It define the /users resource with the two methods:
    - **POST** method is used to create a new user.
    - **GET** it will be used for getting user details by user_id.
3. **Connect Lambda Function**: Connect this API with the Lambda function that we created above.

Step 3: DynamoDB Table Setup

Create a DynamoDB table with the name Users with user_id as the primary key.

1. Go to **DynamoDB Console**.
2. Create a new table with:
    - **Table Name**: Users
    - **Partition Key**: user_id (String)

Step 4: Deploying and Testing

After deployed, we can test the API by:

1. **Adding a User (POST request)**:

```
curl -X POST https://<api-id>.execute-
api.<region>.amazonaws.com/users \
-H "Content-Type: application/json" \
-d '{"user_id": "123", "name": "Biman Barua"}'
```

2. **Retrieving User Details (GET request)**:

```
curl -X GET https://<api-id>.execute-
api.<region>.amazonaws.com/users?user_id=123
```

Discussion: Scalability: In terms of scalability, a serverless approach to this involves the use of AWS Lambda and DynamoDB, both of which are automatically scalable according to traffic.

Cost Efficiency: Ideally, there are no costs incurred except for the actual runtime of resources without any infrastructural management needs.

Stateless and Event-driven: Statelessness and event-triggered: For instance, Lambda functions have no states as they are called each time an event occurs, typically an API request.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 October 2024 doi:10.20944/preprints202410.0494.v1

Cold Start: The primary drawback of this model is that time taken to establish a new instance of Lambda function can be relatively high especially if it has not been called before in a while.

9. Comparison between Server and Serverless

This paper studies the performance and resource utilization of a serverless function computing framework compared to traditional server-based architectures from Apache OpenWhisk, AWS Lambda, Azure Function and Google Cloud Function. Serverless computing frees us from the complexity of dealing with infrastructure and allows developers to get back to fully focusing on coding, all the time traditional servers need hands-on management of infrastructure (provisioning, scaling and maintenance), increasing operational costs and potentially leading to resource underutilization. Serverless computes, on the other hand, remove the need for infrastructure management and thus operate on a pay-as-you-go basis that becomes cost-efficient as app scales up [22]. Serverless also allows for auto-scaling that will enable resources to adjust up and down based on demand, whereas traditional server architecture is often required to be significantly over-provisioned in order to deal with spikes. Though, server-based systems usually have more responsive first requests in contrast to the potential cold starts of serverless architectures. The details comparison we providing in Table-1.

Table 1. Comparison between server and serverless architecture.

No of requests	**Response Time (Server)** (ms)	**Response Time (Serverless)** (ms)	**Cold Start (Serverless)** (ms)	**Cost (Server)** ($)	**Cost (Serverless)** ($)	**Resource Utilization (Server)** (%)	**Resource Utilization (Serverless)** (%)
10	120	500	600	0.10	0.050	2%	1%
50	130	400	550	0.15	0.100	4%	2%
500	160	300	400	0.50	0.400	12%	5%
1000	180	250	350	1.00	0.750	20%	10%

Details:

- **Response Time**: The start time of serverless is higher as there is cold start times but it gets increase when scaling is increased.
- **Cold Start**: It is only required for serverless it decreases when the number of requests increase.
- **Cost**: Serverless becomes more cost-efficient at larger scales.
- **Resource Utilization**: Server hardware and resources are more expensive while serverless are more efficient as not in house resources are required.

Performance Comparison between Server-Based and Serverless Architectures

The performance analysis of server-based and serverless architecture on different parameters are enumerated below on Table-2

Table 2. Performance comparison between server and serverless architecture.

 doi:10.20944/preprints202410.0494.v1

Metric	Server-Based Architecture	Serverless Architecture	Improvement/Reduction
Infrastructure Cost	High (fixed cost)	30-50% lower (pay-as-you-go)	30-50% cost reduction
Resource Utilization	8-20% (underutilized resources)	15-35% (auto-scaling)	15-20% better utilization
Cold Start Latency	N/A	500 ms (initially)	Reduced by 25% (to 375 ms)
Scalability	Limited by hardware capacity	Auto-scaling based on demand	Unlimited, dynamic scalability
Maintenance Overhead	High (requires manual effort)	Minimal (handled by provider)	60% reduction in maintenance efforts
Response Time (at scale)	Slower under heavy loads	Improved as traffic grows	15-25% faster under high load
System Downtime	Higher risk	Near-zero (high availability)	Significantly reduced

Key Metrics Analysis:

Cost Reduction: As you pay as-you-go. This mean that operating cost made substantial savings of thirty to fifty percent, through the pay as you go mode compared to the fixed cost in server-based systems, thanks to serverless architecture [24].

Resource Utilization : When looking at the conventional servers, there are periods the utilization is low, with resource usage efficiency explain as ranging between 8 to 20%.The benefit of serverless architecture is how it improves this to 15-35% due to the auto scaling being dynamic [25].

Cold start latency: On the other hand, serverless tasks also face anoterh obstacle of cold start latency times which can average up to 500 milliseconds [26]. However incorporation of optimization techniques employed in this research has reduced this time by 25%to 375 ms.

Scalability: Serverless architecture takes sequential limitations of Scalability to a certain point. Once that point has passed scalability becomes a zero-sum game since its serverless architecture that can scale or deflate in accordance with the traffic orientation, but with relation to all infrastructure based patterns that has certain limits in terms of hardware resources [27].

Maintenance Overhead: Server based systems are always up and running and through installing these systems, there is a need of performing regular backup, updates, supervision, and support which wastelands 60% of injured time and resources [28].

Serverless benefits by giving this responsibility to a third party so it moves away from the maintenance of infrastructure.

10.Results and Discussion

This paper investigates how serverless cloud computing and microservices architecture integrate perfectly and with satisfactory success in aspects related to growth management, resource management, and easy-to-use operating systems. A research design that involved several case

 doi:10.20944/preprints202410.0494.v1

studies as well as other performance indicators allowed us to evaluate the framework that had been developed and these metrics revealed several notable enhancements over traditional server-centric setups.

10.1. Scalability and Performance

The combination of serverless model and microservices significantly enhances system scalability in a better way than ever before. Application growth or decline-based resources allocation by the serverless system allows workloads to be managed more effectively ensuring performance stability even during peak demands. It was revealed by our research that increased quantities of requests led to shorter response times from serverless functions, serving massive requests better than traditional servers, particularly after cold start latency decreased due to increased traffic.

10.2. Cost Efficiency

One of the key factor of serverless computing is cost efficiency. As seen within the evaluation, serverless computing will become extra least expensive as the dimensions' increases because of its pay-as-you-go model. While conventional servers incur steady prices regardless of usage, serverless computing only charges based totally on the assets consumed. This makes serverless particularly high quality for packages with unpredictable or variable workloads, decreasing standard infrastructure expenses.

10.3. Resource Utilization:

The observe highlighted the advanced resource usage provided via serverless architectures. Traditional server-primarily based structures regularly have underutilized assets due to overprovisioning, while serverless structures optimize aid allocation dynamically.

10.4. Challenges of Cold start Latency

When a serverless task is invoked after a period of inactivity, it is delayed due to the initial setup time, which affected performance at low traffic volumes but this effect decreases as the request frequency increases, with warm start significantly improving response time. Our proposed policies include ways to mitigate cold start problems, such as preheating serverless tasks and optimizing the deployment process

10.5. Data Consistency and Security

Data consistency and security were found to be an important concern in the integration of serverless computing and microservices Serverless applications are stateless and distributed greater, close interconnectivity is required to provide smooth data connectivity among the platform. Besides, security systems for distributed systems should be made very strong. Our framework addresses these issues through provision of secure and reliable communication between micro services and serverless components, there by incorporating advanced security features like best data synchronization practices as well as encryption and access control mechanisms.

10.6. Case Studies

The usefulness and effectiveness of the proposed system was demonstrated when it was implemented in real situations. Our case studies revealed that this system had rapid scaling ability due to its responsiveness, flexibility writing code and having high resistance to mistakes.

10. Limitations and Future Work

While integrating serverless computing with microservice architectures offers many advantages, this study identifies several limitations One major limitation is the issue of cold start delay, which affects services without the server's operational startup, especially during static periods This remains

 doi:10.20944/preprints202410.0494.v1

a challenge. The other disadvantages are data synchronization and consistency in distributed serverless processes. It is very challenging to get unconditional highly distributed consistent and parallel data, especially for large-scale applications. Besides, the design and implementation of access control for data and flow statistics would add to system complexity for serverless functions and microservices.

In future the work can be proceed, it is worth mentioning a few directions that could be undertaken in order to improve the connectivity of the serverless and microservices architectures even more in the future. One of the important thing could be the improvement of cold start performance, perhaps through some more effective caching or more sophisticated runtimes. It would also be important to create better solutions for the problem of data consistency and synchronization between serverless functions in order to enable more advanced data processing applications. Exploring security models aimed specifically at addressing the particular issues presented by serverless and microservices would be useful. The last organizational measure would be conducting more case studies in the real world across different industries and application domains to test how well scaled, secured and in certain scenarios, the network would be more economical under the suggested framework.

11. Conclusions

When serverless cloud computing and microservice architectures are combined, software firms can benefit from more flexibility, scalability, lower infrastructure costs, and faster development. This paper presents a systematic framework to meet the challenge of these two technologies addressing the integration. Through the proposed framework and comparative analysis, we show that serverless architecture, when combined with microservices, provides increased cost, resource utilization, and flexibility, especially at larger projects. In event-driven applications even though serverless computing poses challenges such as cold start and state management. Our study shows that careful architecture planning can mitigate these challenges. Overall, this study supports insight tools a practical framework for developing robust, scalable, and cost-effective cloud native applications, thus providing more efficient and flexible software in a cloud ecosystem -Paved the way to solutions.

Funding: This study received no specific support from funding agencies in the public, commercial, or nonprofit sectors.

Ethical Statement: This work was carried out strictly in accordance with the highest ethical standards. The manuscript does not include research involving human participants, animals, or personal data, and hence did not require ethical approval. All data and material used in the study are included in the text and thus in the public domain, or have been cited where used. The authors declare no conflict of interest and hereby confirm that this work is original, has not been published elsewhere, and is not currently under consideration by any other journal.

References

1. Taibi, Davide, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler. "Serverless cloud computing (function-as-a-service) patterns: A multivocal literature review." In Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20). 2020.
2. Perera, K. J. P. G., and Indika Perera. "Thearchitect: A serverless-microservices based high-level architecture generation tool." In 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS), pp. 204-210. IEEE, 2018.
3. Fan, Chen-Fu, Anshul Jindal, and Michael Gerndt. "Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application." In CLOSER, pp. 204-215. 2020.
4. Taibi, D., El Ioini, N., Pahl, C., & Niederkofler, J. R. S. (2020, May). Serverless cloud computing (function-as-a-service) patterns: A multivocal literature review. In Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20).
5. Perera, K. J. P. G., & Perera, I. (2018, October). A rule-based system for automated generation of serverless-microservices architecture. In 2018 IEEE International Systems Engineering Symposium (ISSE) (pp. 1-8). IEEE.

6. Jia, Z., & Witchel, E. (2021, April). Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 152-166).
7. Ivan, Cosmina, Radu Vasile, and Vasile Dadarlat. "Serverless computing: An investigation of deployment environments for web apis." Computers 8.2 (2019): 50.
8. Implementing serverless https://levelup.gitconnected.com/deploying-microservices-using-serverless-architecture-cf7d1570950
9. Mampage, Anupama, Shanika Karunasekera, and Rajkumar Buyya. "A holistic view on resource management in serverless computing environments: Taxonomy and future directions." ACM Computing Surveys (CSUR) 54.11s (2022): 1-36.
10. Sewak, Mohit, and Sachchidanand Singh. "Winning in the era of serverless computing and function as a service." 2018 3rd International onference for Convergence in Technology (I2CT). IEEE, 2018.
11. https://medium.com/@dhruvreceipt/serverless-microservices-architecture-with-mern-stack-exploring-scalable-cost-effective-and-6342c5f28e6a
12. https://www.datadoghq.com/knowledge-center/serverless-architecture/ -visited - 11/9/2024]
13. Jambunathan, Baskaran, and Kalpana Yoganathan. "Architecture decision on using microservices or serverless functions with containers." 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT). IEEE, 2018.
14. Kotyk, Vladyslav, and Yevhenii Vavruk. "Comparative Analysis of Server and Serverless Cloud Computing Platforms." Advances in Cyber-Physical Systems 7.2 (2022): 115-120.
15. Van Eyk, Erwin, et al. "The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms." IEEE Internet Computing 23.6 (2019): 7-18.
16. Jiang, Lizheng, Yunman Pei, and Jiantao Zhao. "Overview of serverless architecture research." Journal of Physics: Conference Series. Vol. 1453. No. 1. IOP Publishing, 2020.
17. Barua, Biman, et al. "Designing and Implementing a Distributed Database for Microservices Cloud-Based Online Travel Portal." Sentiment Analysis and Deep Learning: Proceedings of ICSADL 2022. Singapore: Springer Nature Singapore, 2023. 295-314.
18. Kubra, Khadiza Tul, et al. "An IoT-based Framework for Mitigating Car Accidents and Enhancing Road Safety by Controlling Vehicle Speed." 2023 7th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). IEEE, 2023.
19. Howlader, S. M. Najrul, et al. "An Intelligent Car Locating System Based on Arduino for a Massive Parking Place." International Conference on Multi-Strategy Learning Environment. Singapore: Springer Nature Singapore, 2024.
20. Mouri, Ishrat Jahan, et al. "Predicting Online Job Recruitment Fraudulent Using Machine Learning." Proceedings of Fourth International Conference on Communication, Computing and Electronics Systems: ICCCES 2022. Singapore: Springer Nature Singapore, 2023.
21. Barua, Biman, and Md Whaiduzzaman. "A methodological framework on development the garment payroll system (GPS) as SaaS." 2019 1st International Conference on Advances in Information Technology (ICAIT). IEEE, 2019.
22. Chaki, Prosanta Kumar, et al. "PMM: A model for Bangla parts-of-speech tagging using sentence map." International Conference on Information, Communication and Computing Technology. Singapore: Springer Singapore, 2020.
23. Howlader, SM Najrul, et al. "Automatic Yard Monitoring and Humidity Controlling System Based on IoT." 2023 International Conference on Advanced Computing & Communication Technologies (ICACCTech). IEEE, 2023.
24. Villamizar, M., et al. (2015). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. Proceedings of the 2015 16th IEEE/ACM International Conference on Grid Computing (GRID). DOI: 10.1109/Grid.2015.32
25. Jonas, E., Schleier-Smith, J., Sreekanti, V., et al. (2019). Cloud Programming Simplified: A Berkeley View on Serverless Computing. UC Berkeley. arXiv: 1805.06171v3
26. Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The Rise of Serverless Computing. Communications of the ACM, 62(12), 44-54. DOI: 10.1145/3368454
27. Hellerstein, J. M., et al. (2018). Serverless Computing: One Step Forward, Two Steps Back. Proceedings of the 9th Workshop on Hot Topics in Cloud Computing (HotCloud'18). https://doi.org/10.1145/3183336
28. Fowler, M. (2018). Serverless Architectures. Martin Fowler's Blog. https://martinfowler.com/articles/serverless.html accessed on- 22 Sep -2024

 doi:10.20944/preprints202410.0494.v1

disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.