© INNOVATIVE RESEARCH THOUGHTS | Refereed | Peer Reviewed | Indexed

ISSN : 2454 – 308X | Volume : 06 , Issue : 03 | March 2020

# Server less Architectures in Cloud Computing: Evaluating Benefits and Drawbacks

**Uday Krishna Padyana**
Independent Researcher, USA.

**Hitesh Premshankar Rai**
Independent Researcher, USA.

**Pavan Ogeti**
Independent Researcher, USA.

**Narendra Sharad Fadnavis**
Independent Researcher, USA.

**Gireesh Bhaulal Patil**
Independent Researcher, USA.

DOI: https://doi.org/10.36676/irt.v10.i3.1439

**Abstract**

A highly efficient computer paradigm for optimising resource usage is emerging: cloud computing. Even with the advantages taken into account, switching to cloud technology is always dangerous from the customer's standpoint. Research on cloud computing that is currently available concentrates on technical issues including efficiency, quality, and security. On the other hand, the actualization of cloud computing is still a very new area of study. The new paradigm of server less computation, which allows developers to provide programmes as stateless functions without consideration for the infrastructure that supports them, is the outcome of recent advancements in virtualisation and software design. Therefore, the lifespan, execution, and scalability of the actual functions—which are only required to run when called upon or triggered by an event—are managed by a server-less system. We present the design of a cutting-edge, performance-focused, server-less computing platform that runs on Microsoft Azure, is built in.NET, and uses Windows container technology for function implementation. Implementation issues are thoroughly investigated, including function scalability and container discovery, longevity, and reuse. We evaluate our prototype and provide metrics for assessing the execution effectiveness of server less platforms, including IBM's Apache Open Whisk implementation, AWS Lambda, Microsoft Azure Functions, and Google Cloud Function. We test the preliminary version and find that it beats competing platforms at most concurrent levels. We also look at the implementations' scalability and instance expiration patterns. In addition, we address the shortcomings and restrictions of our present design, suggest possible solutions, and outline further investigation.

**Keywords: -** Server Less, Computing Platform, Cloud Computing, Software Architecture, Possible Solutions, Execution Environments, Open Whisk, Implementations.

## I. INTRODUCTION

Additional solutions are required to manage the network traffic generated by the increasing number of IoT devices being deployed in support of emerging technologies such as self-driving vehicles, augmented reality applications, and smart cities [1]. Edge technology has showed promise in satisfying the high Quality of Service (QoS) requirements of these kinds of applications while decreasing energy consumption [2] by expanding the data processing horizon to the edge of the network and relieving gadgets of computationally intensive tasks [1,2].

Server less computation, or Function-as-a-Service (FaaS), a popular cloud computing paradigm, has eliminated the need for always-on infrastructures with the usage of ephemeral containers. It is possible to halt, destroy, rebuild, and replace these containers with the least amount of preparation and configuration required. This event-driven services execution technique enables on-demand access to

functions (or services), which may assist with bandwidth costs, latency, availability, and data privacy concerns brought on by IoT devices [2]. Integrating server-less computation at the edge of an IoT network may reduce the computation time required for small tasks [2].

Server-less computing environments (such as Amazon Web Services Lambda, Azure Functions, IBM's Cloud Functions, [2, 3], or Cloud Functions) are currently supported by all of the main cloud service providers. Vendor lock-in results from these platforms' requirements for functionality to be developed or implemented in a particular way [2]. Without requiring any kind of vendor lock-in, a number of open-source FaaS platforms have been created to enable server less computing on private infrastructure. Recent research has looked at the effectiveness and usability of a few open source server-less computing systems, but it hasn't considered the constraints that come with working in an edge context [2, 3].

A new concept labelled server less computing divides software platforms into several independent, stateless processes. Because functions are stateless, they may grow independently and are only run in response to triggers, such as user interactions, message events, or database changes. Function-as-a-service, or FaaS, is another name for server-less computing [3, 4]. Almost all operational issues are abstracted away from developers using this strategy. Entrepreneurs just create code and publish it on an operating system without a server.

After that, the platform takes care of networking, fault-tolerant storage, and containerisation. Moreover, the platform with fewer servers controls function scaling according to actual usage. A number of applications, such as mobile computing, computational science, and data analytics at the border of networks, have shown interest in server-less computer. When using private clouds, server-less computing infrastructure is frequently managed by an outside business or the operation teams [3, 4]. Server-less computing solutions are now offered by all of the main cloud service providers, such as Google's Cloud Functions, IBM Cloud Functions, Microsoft Cloud Functions, and Amazon Web Services (AWS) Lambda. But certain platforms need features to be developed in a particular way, which leads to suppliers who are locked in [3, 4].

Server-less technological advances is a service offered in the cloud where the logic of an application is divided into functions that run in response to events. Services like Apache Open Whisk, Functions from Azure, Google Cloud Features, Iron.io Iron Functions, & Open Lambda have developed and provided server-less technology, resulting from in the footsteps of AWS Lambda [4]. These events can come from sources inside or outside of the cloud platform, but they also frequently happen within its product and service offerings, which made it simple for developers to create applications that are dispersed over several cloud services. The event-driven ideal, in which programming are defined by actions and the events that trigger them, is partially realised via server-less computing [4, 5].

The terminology used here is reminiscent of active systems with databases, and event-driven computer systems have long been suggested in the literature. In these systems, actions are handled in reactions to streams of events. These ideas are fully supported by server-less functional systems, which define actions as straightforward function abstraction and implement processing of events logic across their clouds [5]. These ideas are strongly echoed by IBM's Open Whisk platform (now Apache Open Whisk) [5, 6], where functions are explicitly defined in terms of event, trigger, and action.

As serverless computing interacts with the edge/fog infrastructures of computing, it has shown to be a great fit for Internet of Things applications. To accommodate the expected rise in Internet of Things (IoT) devices, efforts are underway to incorporate server-less computation into a "hierarchy of data centres." With the launch of Lambda Edge, a new offering from AWS, [6, 7], developers of applications may now deploy restricted Lambda services in order to execute on edge nodes. Additionally, AWS has worked on a number of server-less computing expansions, such Green Grass, which provides a common structure for programming for Lambda and Internet of Things applications [7]. With server less

computing, programmers may divide big programmes into smaller functions, enabling individual application components to grow independently [7]. However, this presents a new difficulty for the efficient management of numerous functions. Step Functions, which AWS just added [7, 8], make it simpler to organise and visualise function interactions.

Applications and web programmes may be executed on a foundation that is both flexible and efficient thanks to cloud computer. The intricacy of using electronic computing equipment increases with its capabilities. Local devices need regular hardware and software changes since they have already become less versatile [8]. Over time, programme design changes, become less flexible and more sophisticated [7, 8]. Cloud solutions that allow for resource optimisation, scalability, flexibility, and—most importantly—cost allocation according to system performance have been developed to address this problem. Cloud platforms provide software structures with flexibility and scalability, but in order to maximise manufacturing operations, limit system costs, and make efficient use of computing capacity, a thorough study is still necessary. Important factors to take into account are the platform for implementation and the appropriate architecture for this system. But even the use of cloud platform capabilities cannot always ensure that the software product will function well and achieve its intended objective [8].

According to Gartner, server less cloud computing is becoming more and more relevant,

> *"There is absolutely no question about the usefulness of [server less computers], which naturally flows to [8], micro service software designs, and is poised for rapid expansion and absorption".*

***"Today's PaaS Investments Lead to Server less Technology,"*** according to Forrester, which defines server less technology as the next wave of cloud-based abstractions. A rising variety of mobile and Internet of Things (IoT) applications are made possible by server less computing, which is quickly gaining favour among cloud providers [8]. It is imperative to maintain the fundamental performance attributes of server less platforms as their scope and acceptance expand. We want to contribute to this attempt in this study by describing the development of a new server-less technology that is performance-focused and contrasting its performance with that of current technologies.

### 1.1 Objectives of the study

- The use of server-less architecture can reduce operating costs by transitioning from server-based to pay-per-execution methods.
- Examine server-less architectures to facilitate quicker development and simpler implementation.

## II. LITERATURE REVIEW

(Kritikos, K., 2018) [9] The emerging computing paradigm called "Server Less Computing" has the potential to completely change how apps are created and implemented. This computing approach deploys small software components, or functionalities, to the cloud without management or cost to the programme developer. Moreover, there are other uses for this kind of computing, such as scientific computing and the processing of images. Owing to the aforementioned benefits, established large cloud providers like Amazon are addressing the growing acceptability of server less computing by providing server less platforms for the setup and provisioning of server less apps for computing. But similar to cloud computing, these companies want to keep their clients by providing extra services that make server less apps more valuable. Server-less frameworks have been developed recently to overcome such problem.

(McGrath, G., 2017) [10] We present the design of a brand-new, performance-focused server-less computing platform that is hosted on Microsoft Azure and is developed in.NET, using containers in Windows as the function execution platform. We go into great detail on the implementation of lifespan

and reuse, as well as function scalability and container discovery. However, we also conduct testing on our prototype, AWS Lambda, Azure Functions, Google Cloud Functions, and IBM's installations of Apache Open Whisk. We provide metrics to assess the successful performance of server less platforms. (Gadepalli, P. K., 2019) [11] With the help of server less computing structures, users may operate single-purpose or small-scale apps without worrying about managing resources, procuring servers, or scaling them up to handle increasing demand. Although server-less computing has its roots in cloud computing architecture, it may be a perfect fit for edges IoT data processing. However, to offer flexible growth at the edge and operational efficiency, traditional server-less solutions based on virtual machines and containers are too heavy-weight (large memory footprint and lengthy function invocation time). Moreover, a lot of special applications for the Internet of Things needed near-real-time responses and low-latency technological data manufacturing, which made the existing cloud-based server-less solutions useless. Web Assemble (Wasm) has been proposed as a substitute method for running server-less applications at performance close to native, with optimised invocation time and a lower memory footprints.

(Pérez, A., Moltó, G., 2018) [12] With the advent of new patterns of architecture like micro services, the increasing use of Linux containers like Docker containers, and improvements to fundamental Cloud computing features like auto-scaling, developers are now able to break down large, complicated systems into smaller, stateless services. As a result, server-less computing—which characterises programmes as a system of event-triggered processes—was developed by cloud providers. These applications are severely restricted by server-less products or services, such AWS Lambda (which may use a limited set of languages for programming or prevent the development and deployment of new libraries). In order to address these issues, this paper provides a paradigm and method for developing Server-less Container-aware Architectures (SCAR).

## III. PROPOSED DESIGN

In order to look into server less difficulties with implementation and provide a standard for comparing current systems, we developed a focused on outcomes server less computing system. The platform has a simple design and a limited feature set. It is developed in.NET and hosted on Microsoft Azure [13, 14]. The prototype uses a message layer and Azure's storage service to store data. Our configuration consists of two parts in addition to Azure Storage offerings: an employee service that manages and runs function containers, and a web service that provides access to the platform's public REST API for developers [13]. The web service uses an interaction layer consisting of many Azure storage queues to identify workers who are accessible. Azure Storage tables are used to store function metadata, whereas Azure Storage blobs are used to store function code. An overview of the platform's elements is shown in Figure 1. The reason Azure Storage was chosen is that it provides highly scalable and low-latency storage primitives through an easy-to-use API, which aligns with the objectives of this solution. These storage devices shall be referred to as queues, tables, and blobs for the sake of brevity [13], with it being understood that in the context of this study, these terms relate to the relevant Azure Storage facilities.

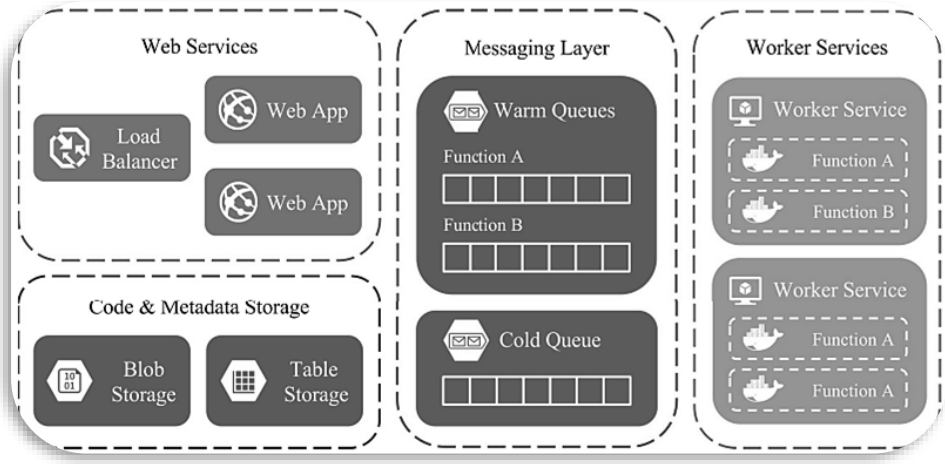


Fig. 1 An overview of the components of the prototype, showing how the web and worker services and products are organised, together with the code, metadata, and message entities stored in Azure Storage.

### A. Function Description

A function is associated with several elements on the system, such as code, operating containers as well, data information about them, or "Warm Queue" [13]. Four fields define functional metadata, which is the source of truth for functional existence:

1) **Function Identifier** - Function resources are located and identified securely using function identities, which are produced at random GUIDs provided during functional construction.
2) **Language Runtime** - The programming language of a function's code is determined by the function's language declaration [13, 14].
3) **Memory Size** - The maximum storage that a function's container can use is indicated by the function's memory size. Right now, 1 GB is the maximum function memory amount that may be used. A function's capacity for memory determines how many CPU cores are assigned to its container [13, 14].
4) **Code Blob URI** - Some of you will receive a zip package containing the code when you create a function. The URI of this code is added to the function's metadata once it has been transferred to a blobs inside the platform's account for storage [14, 15].

### B. Function Execution

Our strategy only permits manual invocations and is based on a very basic function programming paradigm. Our study focuses on the functioning of systems, for which human help is adequate execution. While event source processor or programming quality of construction are important aspects in server less services, [15], our investigation concentrates on operation of the system. Calling the 'invoke' route of the REST API from functional sources is how function are executed. The bodies of the initiating call request act as the function' inputs, while the answer's bodies involve the function's output [15].

### C. Container Allocation

Every worker can allocate space to operate containers from a pool of free memory. The name of the container, which uniquely identifies both the container and the amount of memory reservation, is produced when memory is assigned, and it is included in the URI sent in container allocations signals. Because of this, each communication in the queue may be uniquely identified and associated with a particular memory reservation inside the worker service instance [15, 16]. Carefully chosen memory has been distributed, and worker functions expect each function to consume the amount of memory allotted to it.

**D. Container Removal**

In order to be exact, there are two methods for removing a container. First, worker service instances that carry containers for a function regularly check for the existence of the warm queue, which is deleted by the web services when a function is killed. When a worker service finds a deleted function queue, it ends the operating function container and recovers its memory allocation. Second, our technology allows for the deletion and recovery of a container's memory after it has been inactive for 15 minutes [16]. If the amount of memory left over after memory reclamation above the permitted function memory size, worker services will move the newly allocated container to the cold queues.

**D. Container Image**

Docker is used by the platform to run Microsoft Nano Server containers, and the Docker Engine API is used for communication. An execution controller or a function implementation (currently Node.js v6.9.5) are intended to be included in the container image. There is no function code in the image. Upon starting the container that will be used, we attach an only accessible for reading volumes comprising code for those functions, rather than creating bespoke containers for each platform function [13, 17]. One image's development was selected for several reasons, such as the ease of use with a single image, the speed at which volumes can be attached, [17], or the fact that Windows Nano Server containers are significantly larger than those of lightweight Linux systems like Alpine Linux, which may have an impact on start-up times or storage expenses. The function's memory size determines the container's storage capacity and CPU percentage, in addition to the read-only volumes.

## IV. PERFORMANCE RESULTS

Nevertheless, we used AWS Lambda, Functions from Azure, Google Cloud Functions, and Apache Open Whisk to develop two tests in order to evaluate the system's execution speed. To conduct these evaluations, we developed a speed tool, which uses the Server Less Computing Environment to deploy a Node.js test functions to each of the apps [17, 18]. In order to provide Server less Framework support on our platform, we have developed a Server less Plugins. This tool uses a straightforward test procedure that runs and responds rapidly in order to quantify the administrative expenses offered by platforms [18]. Using the function's execution route on our platform, this function is called synchronous via HTTP events/triggers as supplied for all of the platforms.

### A. Concurrency Test

The results of the simultaneous test, which evaluates a server-less platform's capacity to call a given function at scale, are shown in Table 1 [18]. By reissuing each request as soon as the preceding call's response is received, our tool keeps making repeated calls to the test procedure.

Table 1 Results of the concurrency test, showing the average number of executions finished in a second compared to the quantity of requests for simultaneous operations made to the role. [18]

| Functions of Google Cloud | AWS Lambda | Apache Open Whisk | Azure Functions | Prototype |
|---|---|---|---|---|
| 0.219 | 2.54 | 66.21 | 21.66 | 69.59 |
| 0.418 | 2.69 | 54.69 | 14.65 | 5.99 |
| 20 | 4.51 | 41.65 | 41.52 | 45.96 |
| 1.25 | 6.98 | 21.65 | 25.1 | 54.96 |
| 0.986 | 79.9 | 98.47 | 54.21 | 15.65 |
| 36.98 | 4.98 | 54.6 | 16.52 | 52.96 |
| 2.96 | 79.59 | 69.4 | 42.51 | 44.25 |
| 1.59 | 5.99 | 63.1 | 21.66 | 98.64 |
| 2.54 | 54.96 | 59.4 | 42.51 | 41.62 |
| 6.49 | 21.58 | 41.59 | 21.63 | 59.58 |

| | | | | |
|---|---|---|---|---|
| 4.89 | 59.5 | 98.96 | 14.96 | 14.51 |
| 5.89 | 57.6 | 14.59 | 54.25 | 21.65 |
| 6.49 | 19.54 | 24.96 | 26.65 | 12.05 |
| 4.66 | 35.64 | 58.96 | 14.56 | 11.98 |
| 2.59 | 11.96 | 14.98 | 21.06 | 14.596 |

A maximum of 15 continuous calls to the test function are allowed in the test, which begins with the first initiating call and adds a new one every 10 seconds (Figure 2) [17, 18]. The tool tallies the number of responses it receives every second, and this number ought to grow as simultaneously increases. Ten attempts were made to complete the test on each platform.

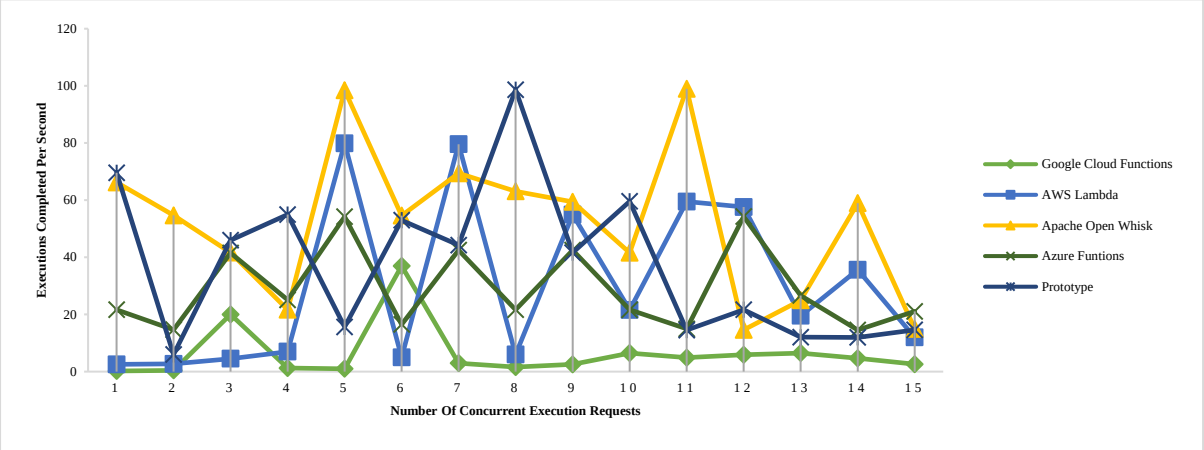


Fig. 2 Results of the concurrency test, showing the average number of executions finished in a second compared to the quantity of requests for simultaneous operations made to the function. [18]

### B. Back off Test

The back off test results are shown in Table 2, which serves to look into the cold start times and expiration behaviours of function instances on different platforms [19]. The back off test sends a single executing request, increasing in time from one to thirty minutes to the test procedure.

Table 2 Results of the Back off Test, showing the function's total execution delay and the amount of time since its last execution. [18, 19]

| **Google Cloud Functions** | **AWS Lambda** | **Apache Open Whisk** | **Azure Functions** | **Prototype** |
|---|---|---|---|---|
| 0.219 | 2.54 | 66.21 | 21.66 | 69.59 |
| 0.418 | 54.21 | 54.69 | 14.65 | 5.99 |
| 21.65 | 16.52 | 41.65 | 21.65 | 45.96 |
| 6.49 | 19.54 | 24.96 | 26.65 | 54.21 |
| 4.66 | 35.64 | 58.96 | 14.56 | 16.52 |
| 2.59 | 11.96 | 14.98 | 21.06 | 42.51 |
| 63.1 | 21.63 | 69.4 | 63.1 | 21.66 |
| 59.4 | 14.96 | 63.1 | 59.4 | 42.51 |
| 0.219 | 2.54 | 66.21 | 21.66 | 69.59 |
| 0.418 | 54.21 | 54.69 | 14.65 | 5.99 |
| 21.65 | 16.52 | 41.65 | 21.65 | 45.96 |
| 5.89 | 57.6 | 14.59 | 54.25 | 26.65 |
| 6.49 | 19.54 | 24.96 | 26.65 | 12.05 |
| 4.66 | 35.64 | 58.96 | 14.56 | 11.98 |
| 2.59 | 11.96 | 14.98 | 21.06 | 14.596 |

The function containers in the prototype are designed to expire after 15 minutes of inactivity. This characteristic is seen in Figure 3, where the completion latencies after 15 minutes show how effective our prototypes are at starting cold [20]. The function resources within Azure Functions appear to be finished after a few minutes, and the cold start timings are comparable to those of our prototype. It's important to emphasise that although though Azure Functions and the prototype we created are Windows apps, their function execution environments are very different [21, 22]. While Azure Functions run inside of Azure App Service, our prototype makes advantage of Windows container technology [22].

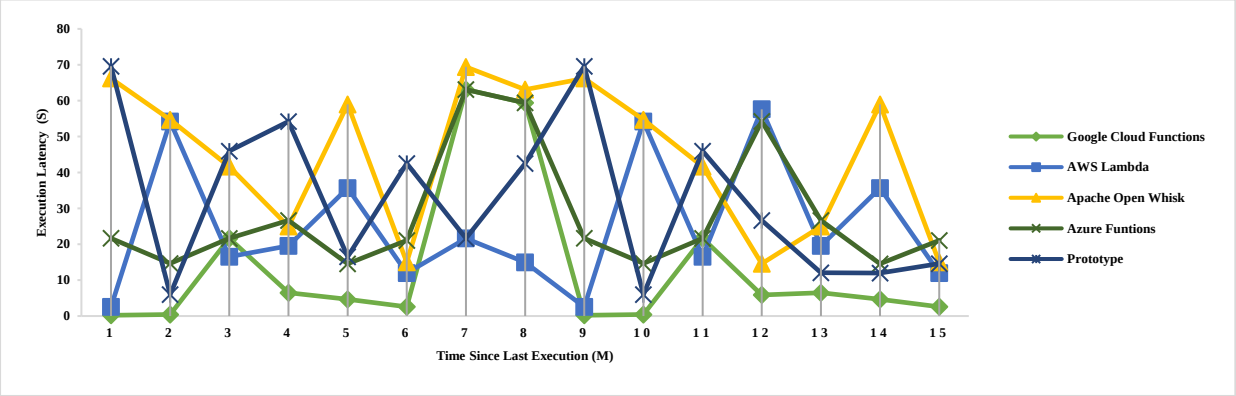


Fig. 3 Results of the back off test, showing the function's total execution delay and the amount of time since its last execution. [23]

Operational idling does not seem to harm Cloud Functions [23, 24]. As discussed below in the examination of Windows containers, unusually quick container activation or pre-allocation of containers might be contributing factors to this trend [25, 26].

## V. LIMITATION AND FUTURE

- **Warm Queues:** Because it is a FIFO queue, the warm queue causes issues with container expiry. Let's say that the function with high load has ten containers assigned to it for execution. Later, the load decreases to the point wherein only one container can manage every function execution.
- **Asynchronous Executions:** The prototype only allows for simultaneously processed invocation at this time. Put otherwise, a request that calls a function will receive the result of that call, not just the procedure's start and stop. Handling asynchronously executions is simple; the web service may simply respond to the invocation call and proceed to handle the execution of the request as usual [26, 27].
- **Worker Utilization:** One aspect of our implementation that we can greatly improve is worker utilisation. Since that not every function on a worker executes or uses up all of their brain's reserve, feasible designs would require an excessive allocation of worker resources [27, 28].
- **Windows Containers:** Compared to containers developed for Linux, containers created for Windows are limited in several ways. This is mainly because Linux container systems were developed on top of Linux c-groups, which provide useful functions that are not accessible on Windows. The ability to update storage container resource and suspend containers is the most noteworthy feature when considering server less computing. When implementing server-less platforms, it is common practice to pause containers when they are inactive in order to prevent resource usage and to uncaused them before execution starts.
- **Security:** One area of ongoing study is server-less the safety of the system. It is a dangerous idea to host arbitrarily drawn user code in function container on multitenant systems; care must be taken to prevent vulnerabilities while developing and using function containers. An essential practical test of generic container security is the junction of RPC & privacy [28, 29].

- **Performance Measures:** Creating and evaluating performance metrics offers several opportunities to further knowledge of server-less platform performances [30]. Although the platform' overhead during single-function implementation was the main focus of this work, improving network latency and timing problems management might improve the quality of these measures.

## VI. CONCLUSION

The current status of open-source server less computing framework was examined in the present research. Cloud platforms are extensively utilised for the development and integrating of software architecture. This article presents many architectures that are used to analyse the involved capabilities. Compared to other architecture examined, server designs offer total access to services and are far safer. In addition to providing dependable, event-driven access to an array of cloud services, server less computing facilitates cost control and incredibly fine-grained scalability through straightforward deployment and programming techniques. Motivated by these advantages, the increasing utilisation of server-less applications requires assessing the quality of server-less computing platforms and developing novel strategies to fully leverage the potential of the technology. The performance results of our platform are promising, and our assessment of the existing application indicates several areas for further exploration and development.

## VII. REFERENCES

[1] Baldini et al., "Serverless computing: Current trends and open problems," in Research Advances in Cloud Computing. Springer, 2017, pp. 1–20.

[2] A. Kanso and A. Youssef, "Serverless: beyond the cloud," in Proceedings of the 2nd International Workshop on Serverless Computing. ACM, 2017, pp. 6–10.

[3] A.Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," Future Generation Computer Systems, vol. 79, pp. 849–861, 2018.

[4] S. Nastic et al., "A serverless real-time data analytics platform for edge computing," IEEE Internet Computing, vol. 21, no. 4, pp. 64–71, 2017.

[5] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: extending serverless computing to the edge of the network," in Proceedings of the 10th ACM International Systems and Storage Conference. ACM, 2017, p. 28.

[6] Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in Proceedings of the 2017 Symposium on Cloud Computing. ACM, 2017, pp. 445–451.

[7] Baldini, P. Castro, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, and P. Suter, "Cloud-native, eventbased programming for mobile applications," in Proceedings of the International Conference on Mobile Software Engineering and Systems. ACM, 2016, pp. 287–288.

[8] Mahmoudi, N. and Khazaei, H. (2022). "Performance modelling of metric-based serverless computing platforms," IEEE transactions on cloud computing, pp. 1–1.

[9] Pérez, A. et al. (2018). "Serverless computing for container based architectures," Future generation's computer systems: FGCS, 83, pp. 50–59.

[10] Pavych, N. and Pavych, T. (2019). "Method for time minimization of API requests service from cyber-physical system to cloud database management system," Advances in Cyber-Physical Systems, 4 (2), pp. 125–131. DOI: 10.23939/acps2019.02.125.

[11] A. Cabrera, G. White, A. Palade, and S. Clarke, "The Right Service at the Right Place: a Service Model for Smart Cities," in 2018 IEEE per Com. IEEE, 2018.

[12] A. Cabrera, A. Palade, G. White, and S. Clarke, "Services in IoT: A Service Planning Model Based on Consumer Feedback," in International Conference on Service-Oriented Computing. Springer, 2018.

[13] White, A. Palade, and S. Clarke, "Qos Prediction for Reliable Service Composition in IoT," in ICSOC. Springer, 2017.

[14] A. Palade, C. Cabrera, G. White, and S. Clarke, "Stigmergic Service composition and Adaptation in Mobile Environments," in International Conference on Service-Oriented Computing. Springer, 2018.

[15] Palade and S. Clarke, "Stigmergy-Based QoS Optimisation for Flexible Service Composition in Mobile Communities," in 2018 IEEE World Congress on Services (SERVICES). IEEE, 2018.

[16] White, A. Palade, C. Cabrera, and S. Clarke, "IoTPredict: Collaborative QoS Prediction in IoT," in 2018 IEEE IPerCom. IEEE, 2018.

[17] A. Pinto, J. P. Dias, and H. Sereno Ferreira, "Dynamic Allocation of Serverless Functions in IoT Environments," in IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC), 2018.

[18] A. Hammond, "Lambdash: Run sh commands inside AWS Lambda environment," lambdash, 2017.

[19] Sparta, "Sparta: A Go framework for AWS Lambda micro services," 2017.

[20] M. Villamizar, O. Garcs, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and micro service architectures," in 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 2016, pp. 179–182.

[21] Kritikos, K., & Skrzypek, P. (2018, December). A review of serverless frameworks. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) (pp. 161-168). IEEE.

[22] McGrath, G., & Brenner, P. R. (2017, June). Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 405-410). IEEE.

[23] Gadepalli, P. K., Peach, G., Cherkasova, L., Aitken, R., & Parmer, G. (2019, October). Challenges and opportunities for efficient serverless computing at the edge. In 2019 38th Symposium on Reliable Distributed Systems (SRDS) (pp. 261-2615). IEEE.

[24] Pérez, A., Moltó, G., Caballer, M., & Calatrava, A. (2018). Serverless computing for container-based architectures. Future Generation Computer Systems, 83, 50-59.

[25] Wagner and A. Sood, "Economics of Resilient Cloud Services," ArXiv e-prints, Jul. 2016.

[26] A. Warzon, "AWS Lambda pricing in context: A comparison to EC2,", 2016.

[27] A. Lowery, "Emerging Technology Analysis: Serverless Computing and Function Platform as a Service," Gartner, Tech. Rep., September 2016.

[28] S. Hammond, J. R. Rymer, C. Mines, R. Heffner, D. Bartoletti, C. Tajima, and R. Birrell, "How To Capture The Benefits Of Microservice Design," Forrester Research, Tech. Rep., May 2016.

[29] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms," in 2017 IEEE CloudCom, Dec 2017.

[30] E. d. Lara, C. S. Gomes, S. Langridge, S. H. Mortazavi, and M. Roodi, "Hierarchical Serverless Computing for the Mobile Edge," in 2016 IEEE/ACM Symposium on Edge Computing (SEC), 2016.

[31] L. F. Herrera-Quintero, J. C. Vega-Alfonso, K. B. A. Banse, and E. Carrillo Zambrano, "Smart ITS Sensor for the Transportation Planning Based on IoT Approaches Using Serverless and Microservices Architecture," IEEE Intelligent Transportation Systems Magazine, 2018.

[32] J. Franz, T. Nagasuri, A. Wartman, A. V. Ventrella, and F. Esposito, “Reunifying Families after a Disaster via Serverless Computing and Raspberry Pis,” in 2018 IEEE LANMAN, 2018.

[33] C. Cicconetti, M. Conti, and A. Passarella, “An Architectural Framework for Serverless Edge Computing: Design and Emulation Tools,” in 2018 IEEE CloudCom. IEEE, 2018.

[34] A. Kuntsevich, P. Nasirifard, and H.-A. Jacobsen, “A Distributed Analysis and Benchmarking Framework for Apache OpenWhisk Serverless Platform,” in Middleware Conference. ACM, 2018.

[35] Kaur, J., Choppadandi, A., Chenchala, P. K., Nakra, V., & Pandian, P. K. G. (2019). AI Applications in Smart Cities: Experiences from Deploying ML Algorithms for Urban Planning and Resource Optimization. Tuijin Jishu/Journal of Propulsion Technology, 40(4), 50-56.

[36] Case Studies on Improving User Interaction and Satisfaction using AI-Enabled Chatbots for Customer Service . (2019). International Journal of Transcontinental Discoveries, ISSN: 3006-628X, 6(1), 29-34. https://internationaljournals.org/index.php/ijtd/article/view/98

[37] Kaur, J., Choppadandi, A., Chenchala, P. K., Nakra, V., & Pandian, P. K. G. (2019). Case Studies on Improving User Interaction and Satisfaction using AI-Enabled Chatbots for Customer Service. International Journal

[38] ofTranscontinental Discoveries, 6(1), 29-34. https://internationaljournals.org/index.php/ijtd/article/view/98

[39] Choppadandi, A., Kaur, J., Chenchala, P. K., Kanungo, S., & Pandian, P. K. K. G. (2019). AI-Driven Customer Relationship Management in PK Salon Management System. International Journal of Open Publication and Exploration, 7(2), 28-35. https://ijope.com/index.php/home/article/view/128

[40] AI-Driven Customer Relationship Management in PK Salon Management System. (2019). International Journal of Open Publication and Exploration, ISSN: 3006-2853, 7(2), 28-35. https://ijope.com/index.php/home/article/view/128

[41] Big Data Analytics using Machine Learning Techniques on Cloud Platforms. (2019). International Journal of Business Management and Visuals, ISSN: 3006-2705, 2(2), 54-58. https://ijbmv.com/index.php/home/article/view/76

[42] Shah, J., Prasad, N., Narukulla, N., Hajari, V. R., & Paripati, L. (2019). Big Data Analytics using Machine Learning Techniques on Cloud Platforms. International Journal of Business Management and Visuals, 2(2), 54-58. https://ijbmv.com/index.php/home/article/view/76

[43] Mahesula, Swetha, Itay Raphael, Rekha Raghunathan, Karan Kalsaria, Venkat Kotagiri, Anjali B. Purkar, Manjushree Anjanappa, Darshit Shah, Vidya Pericherla, Yeshwant Lal Avinash Jadhav, Jonathan A.L. Gelfond, Thomas G. Forsthuber, and William E. Haskins. "Immunoenrichment Microwave & Magnetic (IM2) Proteomics for Quantifying CD47 in the EAE Model of Multiple Sclerosis." Electrophoresis 33, no. 24 (2012): 3820-3829. https://doi.org/10.1002/elps.201200515.

[44] Big Data Analytics using Machine Learning Techniques on Cloud Platforms. (2019). International Journal of Business Management and Visuals, ISSN: 3006-2705, 2(2), 54-58. https://ijbmv.com/index.php/home/article/view/76

[45] Mahesula, S., Raphael, I., Raghunathan, R., Kalsaria, K., Kotagiri, V., Purkar, A. B., & ... (2012). Immunoenrichment microwave and magnetic proteomics for quantifying CD 47 in the experimental autoimmune encephalomyelitis model of multiple sclerosis. Electrophoresis, 33(24), 3820-3829.

[46] Mahesula, S., Raphael, I., Raghunathan, R., Kalsaria, K., Kotagiri, V., Purkar, A. B., & ... (2012). Immunoenrichment Microwave & Magnetic (IM2) Proteomics for Quantifying CD47 in the EAE Model of Multiple Sclerosis. Electrophoresis, 33(24), 3820.

[47] Raphael, I., Mahesula, S., Kalsaria, K., Kotagiri, V., Purkar, A. B., Anjanappa, M., & ... (2012). Microwave and magnetic (M2) proteomics of the experimental autoimmune encephalomyelitis animal model of multiple sclerosis. Electrophoresis, 33(24), 3810-3819.

[48] Salzler, R. R., Shah, D., Doré, A., Bauerlein, R., Miloscio, L., Latres, E., & ... (2016). Myostatin deficiency but not anti-myostatin blockade induces marked proteomic changes in mouse skeletal muscle. Proteomics, 16(14), 2019-2027.

[49] Shah, D., Anjanappa, M., Kumara, B. S., & Indiresh, K. M. (2012). Effect of post-harvest treatments and packaging on shelf life of cherry tomato cv. Marilee Cherry Red. Mysore Journal of Agricultural Sciences.

[50] Shah, D., Salzler, R., Chen, L., Olsen, O., & Olson, W. (2019). High-Throughput Discovery of Tumor-Specific HLA-Presented Peptides with Post-Translational Modifications. MSACL 2019 US.

[51] Big Data Analytics using Machine Learning Techniques on Cloud Platforms. (2019). International Journal of Business Management and Visuals, ISSN: 3006-2705, 2(2), 54-58. https://ijbmv.com/index.php/home/article/view/76

[52] Purohit, M. S. (2012). Resource management in the desert ecosystem of Nagaur district_ An ecological study of land agriculture water and human resources (Doctoral dissertation, Maharaja Ganga Singh University).

[53] Kumar, A. V., Joseph, A. K., Gokul, G. U. M. M. A. D. A. P. U., Alex, M. P., & Naveena, G. (2016). Clinical outcome of calcium, Vitamin D3 and physiotherapy in osteoporotic population in the Nilgiris district. Int J Pharm Pharm Sci, 8, 157-60