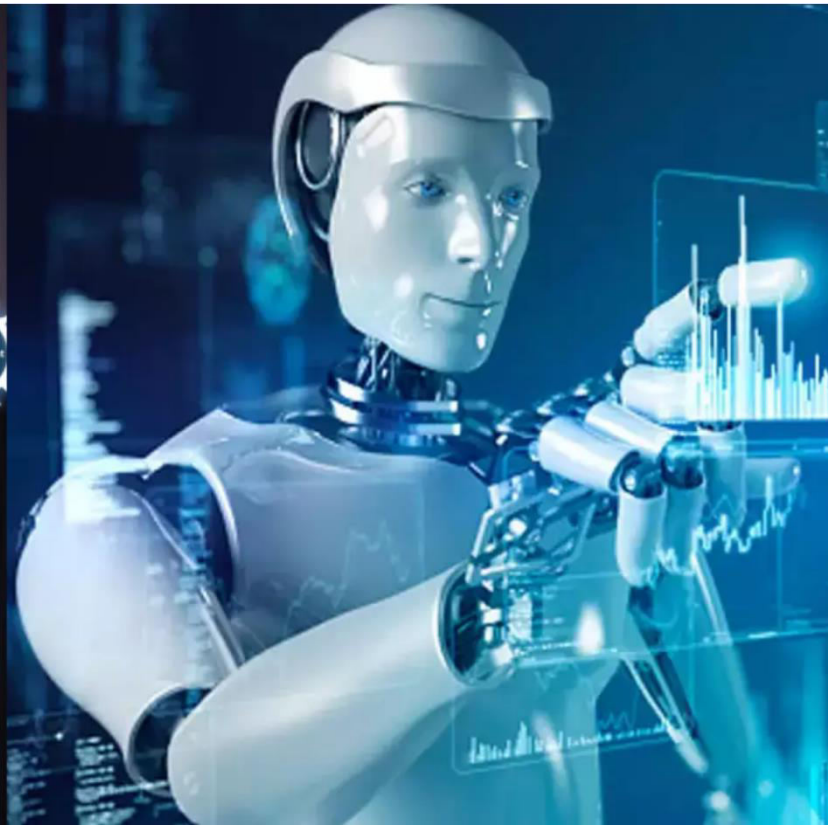
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

# International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

*(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)*

**Impact Factor: 8.699**

**Volume 14, Issue 5, May 2025**

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

# Serverless Computing: Current Trends and Open Problems

**Varsha Ambaldhage, GS Udaya Kiran Babu**

M.Tech Student, Dept. of CSE, Bheema Institute of Technology & Science, Adoni, India

Professor of Dept. CSE, Bheema Institute of Technology & Science, Adoni, India

**ABSTRACT:** Serverless computing has emerged as a new compelling paradigm for the deployment of applications and services. It represents an evolution of cloud pro gramming models, abstractions, and platforms, and is a testament to the maturity and wide adoption of cloud technologies. In this chapter, we survey existing server less platforms from industry, academia, and open source projects, identify key char acteristics and use cases, and describe technical challenges and open problems

## I. INTRODUCTION

Serverless Computing (or simply serverless) is emerging as a new and compelling paradigm for the deployment of cloud applications, largely due to the recent shift of enterprise application architectures to containers and microservices [22]. Figure 1 below shows the increasing popularity of the "serverless" search term over the last f ive years as reported by Google Trends. This is an indication of the increasing attention that serverless computing has garnered in industry tradeshows, meetups, blogs, and the development community. By contrast, the attention in the academic community has been limited. From the perspective of an Infrastructure-as-a-Service (IaaS) customer, this paradigm shift presents both an opportunity and a risk. On the one hand, it pro vides developers with a simplified programming model for creating cloud applica tions that abstracts away most, if not all, operational concerns; it lowers the cost of deploying cloud code by charging for execution time rather than resource allo cation; and it is a platform for rapidly deploying small pieces of cloud-native code Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, Philippe Sute

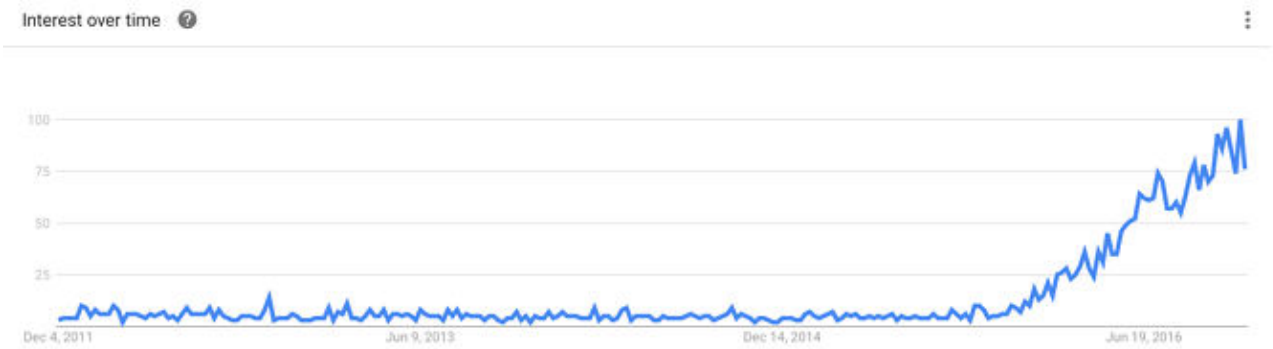



**Fig. 1** Popularity of the term "serverless" as reported by Google Trends

that responds to events, for instance, to coordinate microservice compositions that would otherwise run on the client or on dedicated middleware. On the other hand, deploying such applications in a serverless platform is challenging and requires relinquishing to the platform design decisions that concern, among other things, quality-of-service (QoS) monitoring, scaling, and fault-tolerance properties. From the perspective of a cloud provider, serverless computing provides an addi tional opportunity to control the entire development stack, reduce operational costs by efficient optimization and management of cloud resources, offer a platform that encourages the use of additional services in their

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

ecosystem, and lower the effort required to author and manage cloud-scale applications. Serverless computing is a term coined by industry to describe a programming model and architecture where small code snippets are executed in the cloud without any control over the resources on which the code runs. It is by no means an indi cation that there are no servers, simply that the developer should leave most opera tional concerns such as resource provisioning, monitoring, maintenance, scalability, and fault-tolerance to the cloud provider. The astute reader may ask how this differs from the Platform-as-a-Service (PaaS) model, which also abstracts away the management of servers. A serverless model provides a "stripped down" programming model based on stateless functions. Un like PaaS, developers can write arbitrary code and are not limited to using a pre packaged application. The version of serverless that explicitly uses functions as the deployment unit is also called Function-as-a-Service (FaaS). Serverless platforms promise new capabilities that make writing scalable mi croservices easier and cost effective, positioning themselves as the next step in the evolution of cloud computing architectures. Most of the prominent cloud computing providers including Amazon [1], IBM [16], Microsoft [23], and Google [14] have recently released serverless computing capabilities. There are also several open source efforts including the OpenLambda project [24]. Serverless computing is in its infancy and the research community has produced only a few citable publications at this time. OpenLambda [24] proposes a reference architecture for serverless platforms and describes challenges in this space (see Sec tion 3.1.3) and we have previously published two of our use-cases [6, 29] (see Section 5.1). There are also several books for practitioners that target developers interested in building applications using serverless platforms [11, 28]

### 1.1Defining Serverless

Succinctly defining the term serverless can be difficult as the definition will overlap with other terms such as PaaSandSoftware-as-a-Service (SaaS). One way to explain serverless is to consider the varying levels of developer control over the cloud in frastructure, as illustrated in Figure 2. The Infrastructure-as-a-Service (IaaS) model is where the developer has the most control over both the application code and oper ating infrastructure in the cloud. Here, the developer is responsible for provisioning the hardware or virtual machines, and can customize every aspect of how an appli cation gets deployed and executed. On the opposite extreme are the PaaS and SaaS models, where the developer is unaware of any infrastructure, and consequently no longer has control over the infrastructure. Instead, the developer has access to pre packaged components or full applications. The developer is allowed to host code here, though that code may be tightly coupled to the platform. For this chapter, we will focus on the space in the middle of Figure 2. Here, the developer has control over the code they deploy into the Cloud, though that code has to be written in the form of stateless functions. (The reason for this will be explained in Section 3.) The developer does not worry about the operational aspects of deployment and maintenance of that code and expects it to be fault-tolerant and auto-scaling. In particular, the code may be scaled to zero where no servers are actually running when the user's function code is not used, and there is no cost to the user. This is in contrast to PaaS solutions where the user is often charged even during idle periods.

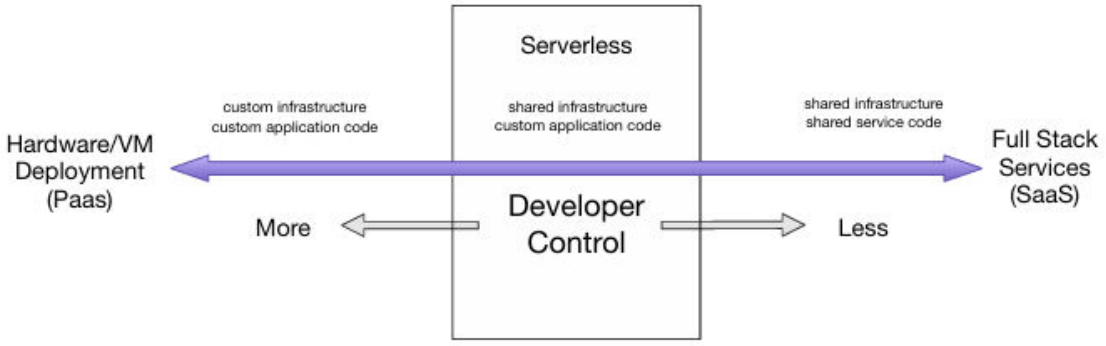



**Fig. 2 Developer control and Serverless computing**

There are numerous serverless platforms the fall into the above definition. In this chapter, we present the architecture and other relevant features of serverless com puting, such as the programming model. We also identify the types of

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

application workloads that are suitable to run on serverless computing platforms. We then con clude with open research problems, and future research challenges. Many of these challenges are a pressing need in industry and could benefit from contributions from academia

## II. EVOLUTION

Serverless computing was popularized by Amazon in the re:Invent 2014 session "Getting Started with AWS Lambda" [4]. Other vendors followed in 2016 with the introduction of Google Cloud Functions [14], Microsoft Azure Functions [23] and IBM OpenWhisk [16]. However, the serverless approach to computing is not com pletely new. It has emerged following recent advancements and adoption of virtual machine (VM) and then container technologies. Each step up the abstraction lay ers led to more lightweight units of computation in terms of resource consumption, cost, and speed of development and deployment. Among existing approaches, Mobile Backend as-a-Service (MBaaS) bears a close resemblance to serverless computing. Some of those services even provided "cloud functions", that is, the ability to run some code server-side on behalf of a mobile app without the need to manage the servers. An example of such a service is Facebook's Parse Cloud Code [27]. Such code, however, was typically limited to mobile use cases. Software-as-a-Service (SaaS) may support the server-side execution of user pro vided functions but they are executing in the context of an application and hence limited to the application domain. Some SaaS vendors allow the integration of arbi trary code hosted somewhere else and invoked via an API call. For example, this is approach is used by the Google Apps Marketplace in Google Apps for Work [26]

## III. ARCHITECTURE

There are a lot of misconceptions surrounding serverless starting with the name. Servers are still needed, but developers need not concern themselves with managing those servers. Decisions such as the number of servers and their capacity are taken care of by the serverless platform, with server capacity automatically provisioned as needed by the workload. This provides an abstraction where computation (in the form of a stateless function) is disconnected from where it is going to run. The core capability of a serverless platform is that of an event processing system, as depicted in Figure 3. The service must manage a set of user defined functions, take an event sent over HTTP or received from an event source, determine which function(s) to which to dispatch the event, find an existing instance of the function or create a new instance, send the event to the function instance, wait for a response, gather execution logs, make the response available to the user, and stop the function when it is no longer needed. The challenge is to implement such functionality while considering metrics such as cost, scalability, and fault tolerance. The platform must quickly and efficiently start a function and process its input. The platform also needs to queue events, and based on the state of the queues and arrival rate of events, schedule the execution of functions, and manage stopping and deallocating resources for idle function in stances. In addition, the platform needs to carefully consider how to scale and man age failures in a cloud environment

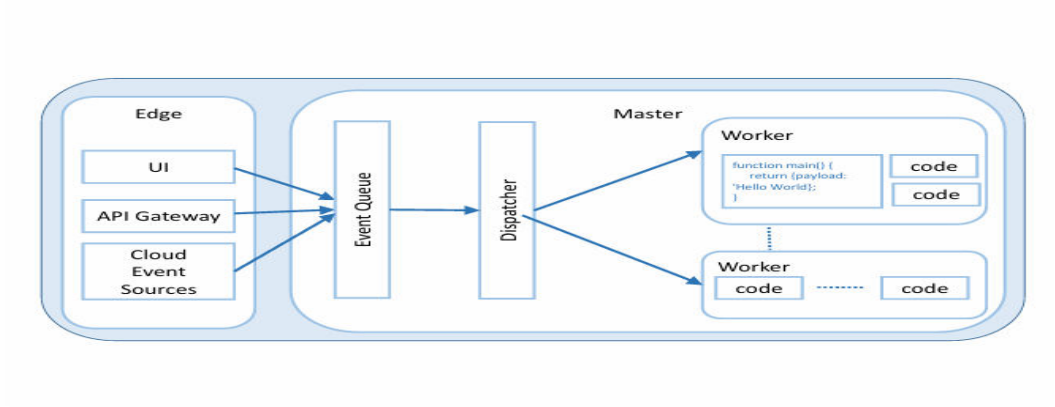



**Fig. 3** Serverless platform architecture

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

## IV. PROGRAMMING MODEL

Serverless functions have limited expressiveness as they are built to scale. Their composition may be also limited and tailored to support cloud elasticity. To maxi mize scaling, serverless functions do not maintain state between executions. Instead, the developer can write code in the function to retrieve and update any needed state. The function is also able to access a context object that represents the environment in which the function is running (such as a security context). For example, a function written in JavaScript could take the input, as a JSON object, as the first parameter, and context as the second:

```
function main(params, context) {
return {payload: 'Hello, ' + params.name + ' from ' + params.place};
}
```

## V. USE CASES AND WORKLOADS

Serverless computing has been utilized to support a wider range of applications. From a functionality perspective, serverless and more traditional architectures may be used interchangeably. The determination of when to use serverless will likely be influenced by other non-functional requirements such as the amount of control over operations required, cost, as well as application workload characteristics. From a cost perspective, the benefits of a serverless architecture are most appar ent for bursty, compute intensive workloads. Bursty workloads fare well because the developer offloads the elasticity of the function to the platform, and just as im portant, the function can scale to zero, so there is no cost to the consumer when the system is idle. Compute intensive workloads are appropriate since in most platforms today, the price of a function invocation is proportional to the running time of the function. Hence, I/O bound functions are paying for compute resources that they are not fully taking advantage of. In this case, a multi-tenant server application that multiplexes requests may be cheaper to operate. From a programming model perspective, the stateless nature of serverless func tions lends themselves to application structure similar to those found in functional reactive programming [5]. This includes applications that exhibit event-driven and f low-like processing patterns

## VI. CHALLENGES AND OPEN PROBLEMS

We will list challenges starting with those that are already known based on our experience of using serverless services and then describe open problems

### 6.1 System level challenges

Here is a list of challenges at the systems level. Cost: Cost is a fundamental challenge. This includes minimizing the resource usage of a serverless function, both when it is executing and when idle. Another aspect is the pricing model, including how it compares to other cloud comput ing approaches. For example, serverless functions are currently most economical for CPU-bound computations, whereas I/O bound functions may be cheaper on dedicated VMs or containers. Cold start: A key differentiator of serverless is the ability to scale to zero, or not charging customers for idle time. Scaling to zero, however, leads to the problem of cold starts, and paying the penalty of getting serverless code ready to run. Techniques to minimize the cold start problem while still scaling to zero are critical. Resource limits: Resource limits are needed to ensure that the platform can han dle load spikes, and manage attacks. Enforceable resource limits on a serverless function include memory, execution time, bandwidth, and CPU usage. In addi tional, there are aggregate resource limits that can be applied across a number of functions or across the entire platform Security: Strong isolation of functions is critical since functions from many users are running on a shared platform. Scaling: The platform must ensure the scalability and elasticity of users' func tions. This includes proactively provisioning resources in response to load, and in anticipation of future load. This is a more challenging problem in serverless because these predictions and provisioning decisions must be made with little or no application-level knowledge. For example, the system can use request queue lengths as an indication of the load, but is blind to the nature of these requests. Hybrid cloud: As serverless is gaining popularity there may be more than one serverless platform and multiple serverless services that need to work together. It is unlikely one platform will have all functionality and work for all use cases. Legacy systems: It should be easy to access older cloud and non-cloud systems from serverless code running in serverless platforms.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

**6.2 Programming model and DevOps challenges**

Tools: Traditional tools that assumed access to servers to be able to monitor and debug applications aren't applicable in serverless architectures, and new ap proaches are needed. Deployment: Developers should be able to use declarative approaches to control what is deployed and tools to support it. Monitoring and debugging: As developers no longer have servers that they can access, serverless services and tools need to focus on developer productivity. As serverless functions are running for shorter amounts of time there will be many orders of magnitude more of them running making it harder to identify problems and bottlenecks. When the functions finish the only trace of their execution is what serverless platforms monitoring recorded. IDEs: Higher level developer capabilities, such as refactoring functions (e.g., splitting and merging functions), reverting to an older version, etc. will be needed and should be fully integrated with serverless platforms. Composability: This includes being able to call one function from another, cre ating functions that call and coordinate a number of other functions, and higher level constructs such as parallel executions and graphs. Tools will be needed to facilitate creation of compositions and their maintenance. Long running: Currently serverless functions are often limited in their execu tion time. There are scenarios that require long running (if intermittent) logic. Programming models and tools may decompose long running tasks into smaller units and provide necessary context to track them as one long running unit of work. State: Real applications often require state, and it's not clear how to manage state in stateless serverless functions- programing models, tools, libraries etc. will need to provide necessary levels of abstraction Concurrency: Express concurrency semantics, such as atomicity (function exe cutions need to be serialized), etc. Recovery semantics: Includes exactly once, at most once, and at least once se mantics. Code granularity: Currently, serverless platforms encapsulate code at the granu larity of functions. It's an open question whether coarser or finer grained modules would be useful

**6.3 Open Research Problems**

Now we will describe a number of open problems. We frame them as questions to emphasize that they are largely unexplored research areas. What are the boundaries of serverless? A fundamental question about server less computing is of boundaries: is it restricted to FaaS or is broader in scope? How does it relate to other models such as SaaS and MBaaS?

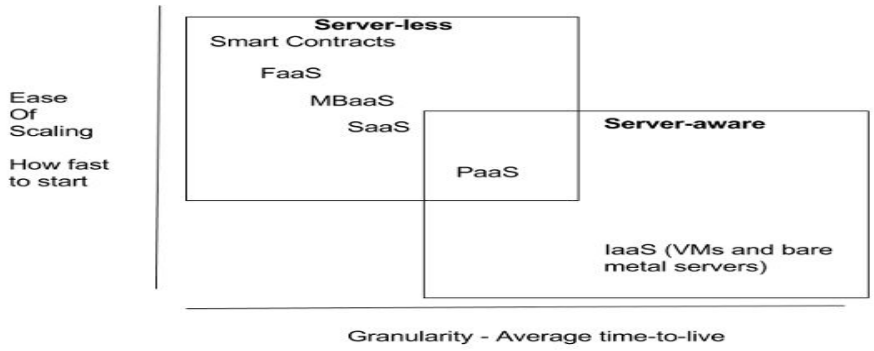



Fig. 9 The figure is showing relation between time-to-live (x-axis) and ease-of-scaling (y-axis). Server-aware compute (bare metal, VMs, IaaS) has long time to live and take longer to scale (time to provision new resources); server-less compute (FaaS, MBaaS, PaaS, SaaS) is optimized to work on multiple servers and hide server details. Asserverless is gaining popularity the boundaries between different types of "as a-Service" may be disappearing (see Figure 9). One could imagine that developers not only write code but also declare how they want the code to run- as FaaS or MBaaS or PaaS- and can change as needs change.

## VII. CONCLUSION

In this chapter we explored the genesis and history of serverless computing in detail. It is an evolution of the trend towards higher levels of abstractions in cloud program ming models, and currently exemplified by the Function-as-a-Service (FaaS) model where developers write small stateless code snippets and allow the platform to man age the complexities of scalably executing the function in a fault-tolerant manner. This seemingly restrictive model nevertheless lends itself well to a number of common distributed application patterns, including compute-intensive

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

event pro cessing pipelines. Most of the large cloud computing vendors have released their own serverless platforms, and there is a tremendous amount of investment and at tention around this space in industry. Unfortunately, there has not been a corresponding degree of interest in the re search community. We feel strongly that there are a wide variety of technically chal lenging and intellectually deep problems in this space, ranging from infrastructure issues such as optimizations to the cold start problem to the design of a composable programming model. There are even philosophical questions such as the fundamen tal nature of state in a distributed application. Many of the open problems identified in this chapter are real problems faced by practitioners of serverless computing to day and solutions have the potential for significant impact. We leave the reader with some ostensibly simple questions that we hope will help stimulate their interest in this area. Why is serverless computing important? Will it change the economy of computing? As developers take advantage of smaller granularities of computational units and pay only for what is actually used will that change how developers think about building solutions? In what ways will serverless extend the original intent of cloud computing of making hardware fungible and shifting cost of computing from capital to operational expenses? The serverless paradigm may eventually lead to new kinds of programming mod els, languages, and platform architectures and that is certainly an exciting area for the research community to participate in and contribute to.

## REFERENCES

1. Awslambda. URLhttps://aws.amazon.com/lambda/. Online;accessed December 1, 2016
2. S3 Simple Storage Service. URL https://aws.amazon.com/s3/. Online; accessed December 1, 2016
3. Building Serverless Apps with Webtask.io. URL https://auth0.com/blog/ building-serverless-apps-with-webtask/. Online; accessed December 1, 2016
4. Awsre:invent 2014 — (mbl202) new launch: Getting started with aws lambda. URL https: //www.youtube.com/watch?v=UFj27laTWQA. Online; accessed December 1, 2016
5. Bainomugisha, E., Carreton, A.L., Cutsem, T.v., Mostinckx, S., Meuter, W.d.: A survey on re active programming. ACM Comput. Surv. 45(4), 52:1–52:34 (2013). DOI 10.1145/2501654. 2501666. URL http://doi.acm.org/10.1145/2501654.2501666
6. Baldini, I., Castro, P., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P.: Cloud-native, event-based programming for mobile applications. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, pp. 287–288. ACM, New York, NY, USA (2016). DOI 10.1145/2897073.2897713. URL http://doi.acm.org/10.1145/2897073.2897713
7. Bienko, C.D., Greenstein, M., Holt, S.E., Phillips, R.T.: IBM Cloudant: Database as a Service Advanced Topics. IBM Redbooks (2015)
8. Learn Chaincode. URL https://github.com/IBM-Blockchain/learn chaincode. Online; accessed 1 December 2016
9. chalice: Python serverless microframework for aws. URL https://github.com/ awslabs/chalice. Online; accessed December 1, 2016
10. Ethereum. http://ethdocs.org/en/latest/introduction/what-is ethereum.html. URL http://ethdocs.org/en/latest/introduction/ what-is-ethereum.html. Online; accessed December 1, 2016
11. Fernandez, O.: Serverless: Patterns of Modern Application Design Using Microservices (Amazon Web Services Edition). in preparation (2016). URL https://leanpub.com/ serverless
12. OpenFog Consortium. URL http://www.openfogconsortium.org/. Online; ac cessed December 1, 2016 13. Galactic Fog Gestalt Framework. URL http://www.galacticfog.com/. Online; ac cessed December 1, 2016
14. Cloudfunctions. URL https://cloud.google.com/functions/. Online;accessed December 1, 2016
15. Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Serverless computation with openlambda. In: 8th USENIX Work shop on Hot Topics in Cloud Computing, HotCloud 2016, Denver, CO, USA, June 20 21, 2016. (2016). URL https://www.usenix.org/conference/hotcloud16/ workshop-program/presentation/hendrickson
16. Openwhisk. URLhttps://github.com/openwhisk/openwhisk. Online;accessed December 1, 2016
17. Cloud Foundry and Iron.io Deliver Serverless. URL https://www.iron.io/cloud foundry-and-ironio-deliver-serverless/. Online; accessed December 1, 2016
18. Introducing LambdasupportonIron.io. URLhttps://www.iron.io/introducing aws-lambda-support/. Online; accessed December 1, 2016
19. Sharable, Open Source Workers for Scalable Processing. URL https://www.iron.io/ sharable-open-source-workers-for/. Online; accessed December 1, 2016
20. Jira. URL https://www.atlassian.com/software/jira. Online; accessed De cember 5, 2016

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 5, May 2025

|DOI: 10.15680/IJIRSET.2025.1405252|

21. LeverOS. https://github.com/leveros/leveros. URL https://github. com/leveros/leveros. Online; accessed December 5, 2016
22. NGINXAnnouncesResults of 2016 Future of Application Development and Delivery Survey. URL https://www.nginx.com/press/nginx-announces-results-of 2016-future-of-application-development-and-delivery-survey/. Online; accessed December 5, 2016
23. Azurefunctions. URLhttps://functions.azure.com/. Online;accessedDecember 1, 2016
24. Openlambda. URL https://open-lambda.org/. Online; accessed December 1, 2016
25. Jira. URL https://www.openstack.org. Online; accessed December 5, 2016
26. Google Apps Marketplace. URL https://developers.google.com/apps marketplace/. Online; accessed December 1, 2016
27. Parse Cloud Code Getting Started. URL https://parseplatform.github.io/ docs/cloudcode/guide/. Online; accessed December 1, 2016
28. Sbarski, P., Kroonenburg, S.: Serverless Architectures on AWS With examples using AWS Lambda. in preparation (2016). URL https://www.manning.com/books/ serverless-architectures-on-aws
29. Yan, M., Castro, P., Cheng, P., Ishakian, V.: Building a chatbot with serverless computing. In: First International Workshop on Mashups of Things, MOTA '16 (colocated with Middleware) (2016)
30. Zappa: Serverless python web services. URL https://github.com/Miserlou/ Zappa. Online; accessed December 1, 2016

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

# INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

## IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details