iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** VII **Month of publication:** July 2021

DOI: https://doi.org/10.22214/ijraset.2021.36551

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VII July 2021- Available at www.ijraset.com

Serverless: Next Generation of Cloud Computing

Md Amir Ali[1], Md Qamar Hashmi[2], Ankur Kumar Pathak[3], Md Faizan[4], Deepak Kanojia[5]
[1, 2, 3, 4]*B.Tech Student,* [5]*Assistant Professor, Greater Noida Institute of Technology, Greater Noida*

Abstract: ***Serverless computing has emerged as a promising new paradigm for application and service deployment. The process of delivering code into production can be made easier with serverless computing. Serverless code can be utilized alongside standard deployment strategies such as microservices and monoliths. Alternatively, programs can be developed to be completely serverless, meaning they don't require any supplied servers.***
A serverless computing service concept is also called Function as a Service (FaaS), which allows developers to deploy specific functions to the cloud. FaaS has proven to be effective for a variety of computational activities, although using it for web apps can be difficult due to excessive response times on occasion. Nevertheless, serverless computing for web applications is gaining popularity. It is a testament to the maturity and widespread usage of cloud technologies, as it symbolizes the development of cloud programming paradigms, abstractions, and platforms. In this research, we investigated existing serverless platforms from industry, academia, and open-source initiatives, identifying key criteria.
Index Terms: Serverless; Cloud Computing; Function-as-a-Service; Cold Start; Microservices

I. INTRODUCTION

Serverless Computing is coming into the picture as a very new and interesting model for the deployment of cloud applications, majorly due to the recent shift of enterprise application architectures to containers and microservices. *Figure 1* below shows the increasing interest of people towards “serverless” search terms over the last five years as reported by Google Trends. This is proof of the successive attention that serverless computing has gathered in industry trade shows, meetups, blogs, and the development community. But on the other hand, the flow in the academic community has been at a limit.

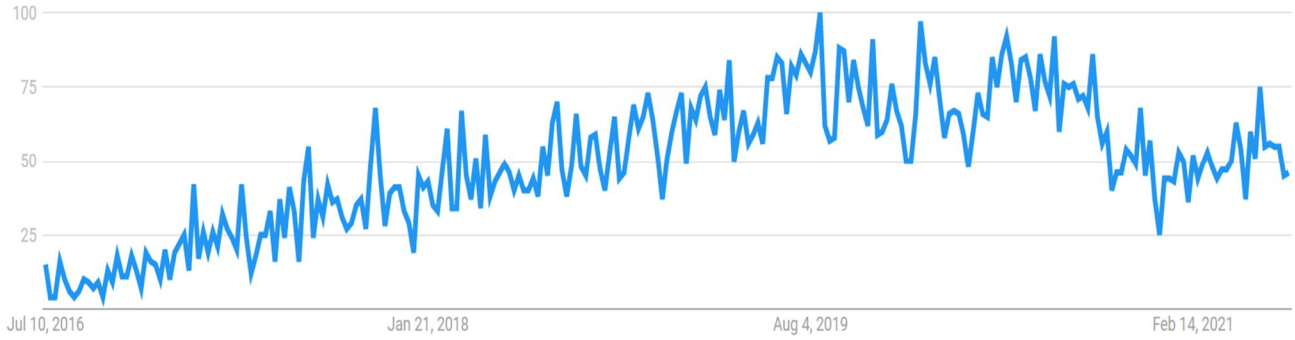


Figure 1: Serverless Google Trends

From the view of an Infrastructure-as-a-Service (IaaS) customer, this revolution presents both an opportunity and a risk. On the one side, it gives developers a simple programming model for producing cloud applications that extract away most, if not all, but operational concerns; it minimizes the cost of deploying cloud code by charging for execution time rather than resource allocation; and it is a platform for swiftly deploying small fragments of cloud-native code that responds to events, for instance, to synchronize microservice compositions that would otherwise run on the client or devoted middleware. On the other hand, deploying such applications in a serverless model is challenging and requires renouncing the platform design decisions that concern, among other things, quality-of-service (QoS) monitoring, scaling, and fault-tolerance properties.

II. FEATURES

To describe serverless architecture, it must be contrasted with more standard software architectural approaches. The vocabulary and definitions of the technology that this thesis addresses are covered in this chapter. It will also serve as a quick summary of recent research on the subject of serverless computing.

A. *Traditional*

1) *Monolith:* In this context, the word "monolithic architecture" relates to Martin Fowler's definition of it as "the traditional approach to software architecture." *Figure 2* shows the architecture of a monolithic web application.

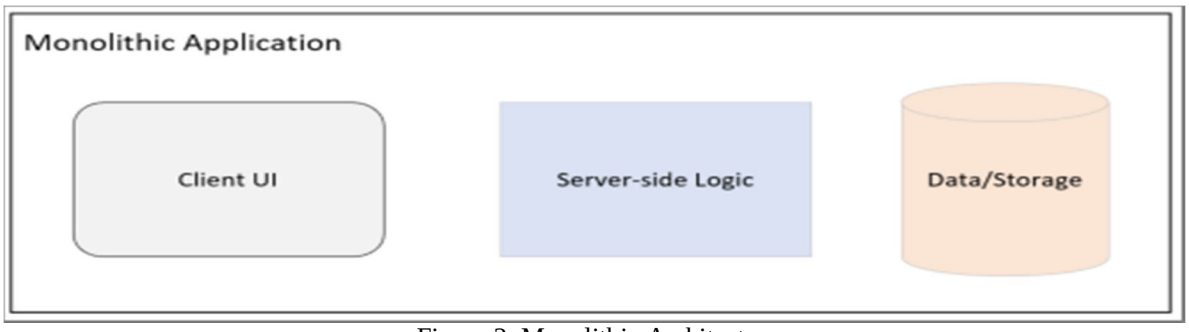


Figure 2: Monolithic Architecture

It consists of a browser-based user interface, a database for storing persistent data, and a server-side application for processing requests from the frontend application and retrieving data from the database. The server-side programme is a single executable that handles all server-side logic and is potentially big. This is a "monolith," according to Fowler's definition. The monolithic approach to application development offers numerous advantages. Developer tools, such as integrated development environments (IDEs), can be focused and configured to build a single application that is simple to deploy and scale. The disadvantages of the monolithic architecture become increasingly evident as the programme grows in size.

Assume that the server-side web application is a monolith that comprises and provides a set of services S= {S1, S2, S3,}. For example, in a web store, a service Sn could be an authentication service, a search service, and so on. Additional services are added over time, and new developers are assigned to the project, increasing the project's complexity. Changes and issue fixes become more difficult and time-consuming, which slows down progress. IDEs can also be slowed down by a huge codebase. Furthermore, developing and testing the system could take a long period, slowing down progress even more. Another thing that may become a concern with monolith systems is scaling. If the programme receives a lot of traffic, it may need to scale up to more instances to fulfil the demand. If traffic to services S is unevenly distributed and just a few services are used, the entire programme must be expanded, not just the in-demand services, which is wasteful.

B. *Modern*

1) *Microservices:* "Microservices" is a response to the monolith's intrinsic flaws, as previously mentioned. Microservices architecture is a type of software architecture in which an application is organised as a collection of loosely linked, independently deployable microservices. A microservice is a small programme that serves a specific purpose and can be scaled, tested, and deployed independently of the larger system. *Figure 3* illustrates a microservice architecture.

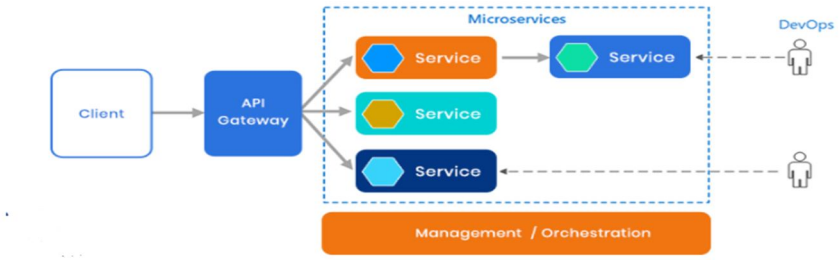


Figure 3: Microservices Architecture

The monolith server-side application is broken into a number of microservices {S =S1, S2, S3,} in the case of the web shop. Each microservice S is responsible for a small portion of the services and provides a subset of them. Instead of being submitted to a single server-side application, requests from the browser are redirected to the relevant microservice via an API gateway in this example. Instead of being submitted to a single server-side application, requests from the browser are redirected to the relevant microservice via an API gateway in this example.

This gateway is used to access the microservice application. Because microservices may be constructed by independent teams using the technology stack and programming language most suited to that service, this architecture allows for greater flexibility. Another advantage of the microservice architecture is that services can scale independently. Enterprises like LinkedIn and SoundCloud adopt this strategy of constructing loosely connected services instead of a monolith, which can offer more practical ways for companies to develop and maintain applications with massive code bases. While the microservices method can address many of the problems associated with monolithic architecture, it is not a panacea. The extra difficulty of maintaining, managing deployment, and growing multiple services in a cloud environment comes at the cost of microservice architecture. Each microservice requires its own infrastructure, environment, and configuration, rather than maintaining the infrastructure of a single monolith.

1) *Serverless:* Serverless services, despite their name, still run-on servers; however, all server and infrastructure maintenance are handled by a third party. In the context of this thesis, the word serverless refers to what is also known as Function-as-a-Service (FaaS), in which functions are the deployment unit, i.e., individual functions rather than whole applications are deployed on the cloud. Amazon's AWS Lambda, Google's Cloud Functions, and Microsoft's Azure Functions are just a few of the cloud providers that are now delivering FaaS on their platforms. In terms of development control, FaaS would fall between Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) in the cloud service classification. *Figure 4* illustrates a Serverless architecture.

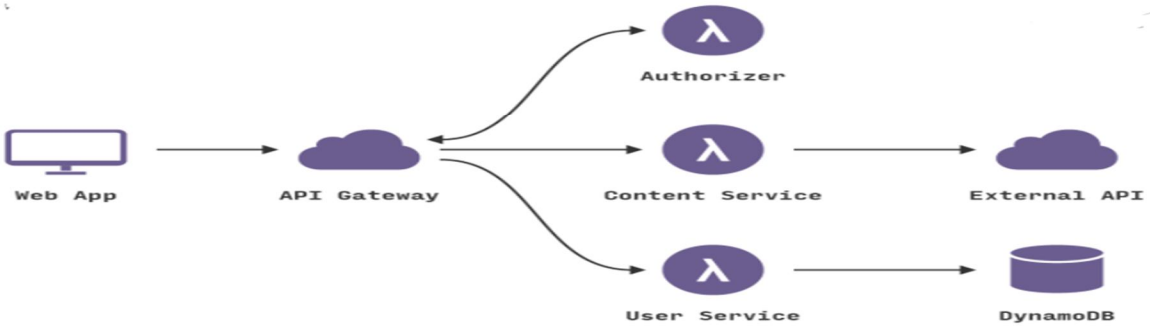


Figure 4: Serverless Architecture

PaaS enables cloud-based server provisioning and application deployment on virtual machines. The developer has more control over the infrastructure and the code that is deployed in PaaS. SaaS, such as Gmail, offers consumers with entire software while the service provider retains full control of the infrastructure and source code. FaaS is located between these *Figure 5*.

Figure 5: Type of Cloud Services

In FaaS, the developer has no control over the infrastructure, which is shared by all platform users, but does have control over the code deployed in the form of stateless functions. Scalability and cost are two more significant differences between FaaS and PaaS. Idle time is frequently paid in PaaS, while in FaaS, functions can be scaled down to zero and fired up only when needed. When a FaaS function is initiated by a trigger, such as a database change or an HTTP request, instances are automatically produced. Serverless functions are essentially stateless in order for them to scale. Variables saved in memory cannot be guaranteed to remain over many invocations of the function, necessitating statelessness or state storage outside of the FaaS function instance.

Serverless functions are essentially stateless in order for them to scale. Variables saved in memory cannot be guaranteed to remain over many invocations of the function, necessitating statelessness or state storage outside of the FaaS function instance.

III. MERITS, DEMERITS & USE CASES OF PROPOSED SYSTEM

A. *Merits*

The most common benefits of using serverless computing are listed below: -

1) *No running compute costs:* While using public serverless services, the most common benefit is that there is no extra cost when the application is simply idle. This is because with FaaS all the resources are being generated on-demand and with the BaaS all the resources are generally shared and are not provided to individual customers. For example, there would be no cost for all the virtual machines when there is no code being executed. There is no cost when the database is idle and waiting for connections. Running costs only come from stateful storage costs.
2) *No extra scaling cost:* As there is no running compute costs that also means that there is no additional cost while the application is being scaled. This is obvious that there would be additional compute cost, but there is no cost for creating an application to scale from hundreds to thousands of users, this means that users can easily scale up from zero without any extra cost and effort. As an example, FaaS services like Google Cloud and AWS Lambda functions bill the user per 100ms of computing time.
3) *Minimum operational costs:* Serverless computing can easily reduce operational costs as there is no demand to provision, update, or manage server infrastructure. Also, PaaS services provide the same benefits.
4) *Smooth scaling:* It is the duty of cloud service providers to scale the service automatically and transparently to the user. An application on serverless architecture can be scaled smoothly to thousands of users in a blink.
5) *Systematic use of resources:* As serverless resources are short-lived, the use of computing resources is much more logical. It is seen in the studies that a huge amount of computing resources is not used at all. With the serverless architecture cloud providers can easily share their resources between different customers and there is no need to reserve a particular resource for a particular customer because the resources are allocated on their demand. For example, to serve 100 HTTP requests in three minutes the instances would be created on this demand, but three minutes later, no instances would be in running and the computing resources or the power could be used somewhere else.

But in the traditional client architecture, the allocated server instance should be in running and some other server instances could be in waiting. After three minutes also these resources are still in the running and the computing power would be going to waste. Hence serverless computing can lead us to a 'greener' computing environment where the computing power would not be wasted on the idle clients.

B. *Demerits*

Serverless computing has its advantages but on the other hand, it has some drawbacks also. Some of them are listed here which might be improved later with the advancements of the tools, standards, and practices, some of the most common drawbacks are listed below: -

1) *Latency in start-up:* When a FaaS function has to be invoked, it might happen that it is being invoked in an already running resource or it might be like that a new resource has to be created to process the invocation. When the containers are already allocated the start-up is called "warm start" and when the container has to be allocated before execution it is called the "cold start". Cold starts take more time in comparison to the warm start because both the execution environment and the application code have to be initialized which can make a delay from milliseconds to seconds. For all the background tasks this delay in time might not matter but for the time treated time-critical applications like web application backend add a delay of a few seconds in time might create a critical issue. Cold start usually depends on the traffic patterns. Applications with consistent traffic patterns, for example, ten requests per second might not face a cold start but the traffic pattern where the pattern is periodic or fluctuates too much, cold starts can be seen.

2) *Execution and state duration:* FaaS architecture presents some constraints to the application code and execution process. As the execution environment can be distracted at any time just after the function invocation, hence the function should be stateless. There should not be any state information saved between the invocations. State information is to be saved in an external database. Cloud service providers already set a time duration for the single function execution. If the function is not executed in the given time slot, it is permanently terminated.
3) *Platform lock-in:* Due to the differences between FaaS platforms and the platforms provided by the cloud service providers, a platform lock-in situation is created where it is very difficult to migrate to another platform or the vendor without a crucial amount of work.
4) *Lack of documentation standardization and best practices:* Being a new concept, serverless computing does not have proper standardization, documentation, best practices, and tools. With the lack of experience in serverless computing, it might lead to some big hidden issues like structuring the code etc. But the situation has been improved with the help of ongoing research on serverless computing.

C. Use Cases

As serverless computing has its advantages and disadvantages as well. It is suitable for only certain types of workloads.

1) *Some common good fit workloads are listed here: -*

- Stateless or temporary workloads which do not get affected by long start-up times.
- Parallel and easy to break into small independent units of work.
- Odd traffic patterns and uncertain scaling requirements.
- Workloads that demand reduced time to market.

2) *On the other hand, serverless computing might not suit the below workload types: -*

- Time taking tasks with high CPU usage.
- Workloads that need to be performed without any delays

IV. CONCLUSION

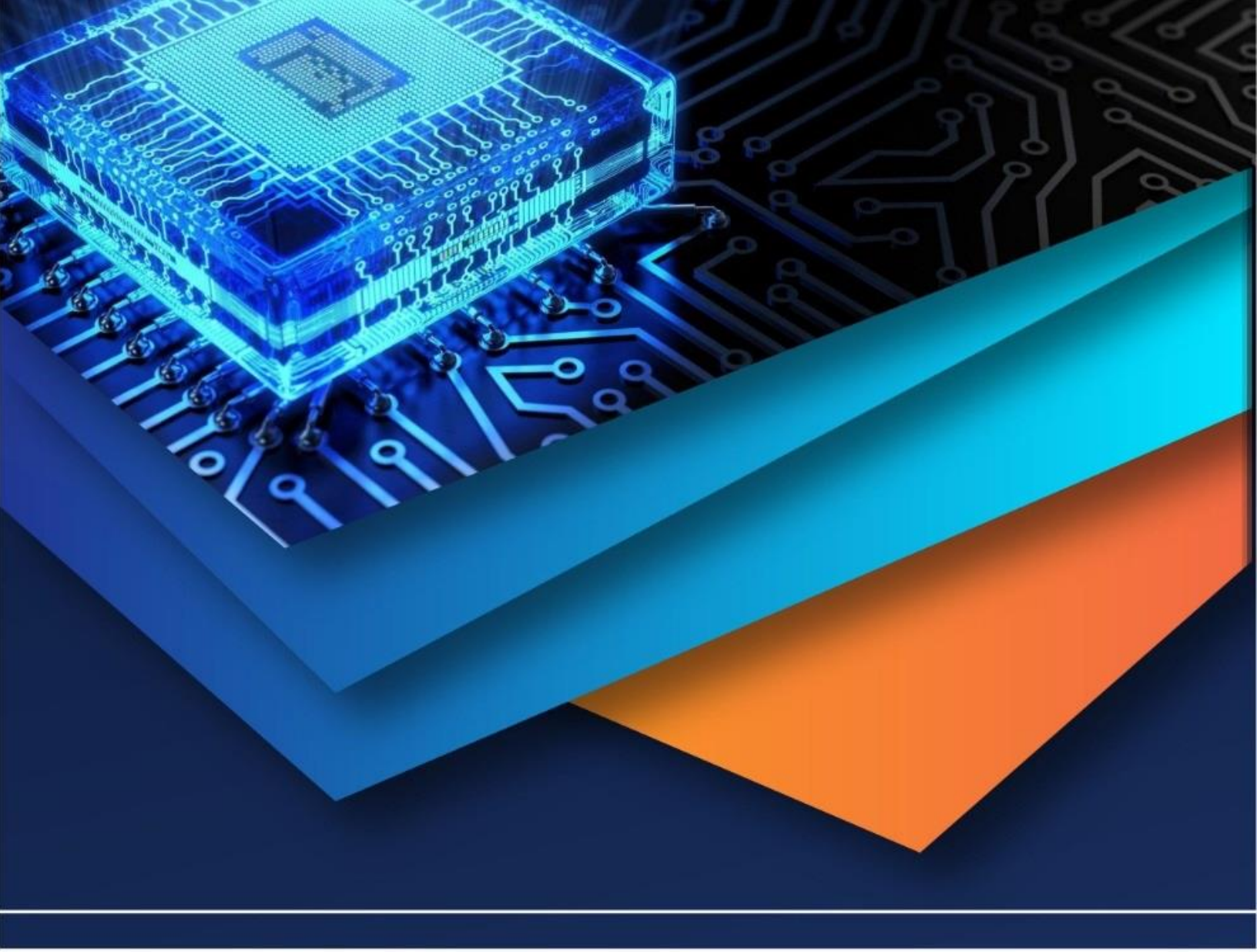
In this research paper, we explore the origin and history of serverless computing in detail. This is an associate degree evolution of the trend towards higher levels of abstraction in cloud programming models and is currently exemplified by the Function-as-a-Service (FaaS) model. Develops small stateless code snippets and allows the platform to manage the complexities of executing functions scalable in a fault-tolerant manner. This seemingly restrictive model nonetheless lends itself well to many Common distributed application patterns, including computation-intensive event processing pipelines. Most of the major cloud computing vendors have serverless platforms of their own, and there's a huge amount of investment and attention in the industry around this space. Unfortunately, there has not been a similar degree of interest in the research community. We strongly feel that this sector has a wide variety of problems ranging from infrastructure to technically challenging and intellectually deep issues such as adaptation to cold start problems for the design of a composable programming model. There are also philosophical questions such as the fundamental nature of the state in a distributed application. Several open problems identified This chapter addresses the real problems faced by practitioners of serverless computing today and the solutions have the potential to have a significant impact.

We leave the reader with some apparently simple questions we hope Help stimulate their interest in this field. Why is serverless computing important? Will it change the economy of computing? As developers take advantage of small methods, how do developers go about building solutions from the granularity of the computational units and pay only for what is actually used? how to be serverless Expanding the core objective of cloud computing to make hardware replaceable and moving the cost of computing from the capital to operating expense?

The serverless paradigm may eventually lead to new types of programming models, languages and platform architectures and this is certainly an exciting field.

REFERENCES

[1] AWS Lambda. URL https://aws.amazon.com/lambda/. Online; accessed 25th May, 2021
[2] Cloud functions. URL https://cloud.google.com/functions/. Online; accessed 19th June, 2021
[3] Cloudflare Serverless. URL https://www.cloudflare.com/en-in/learning/serverless/what-is-serverless/. Online; accessed 30th June, 2021
[4] Sourabh Sharma; Rajesh RV; David Gonzalez; Safari, an O'Reilly Media Company. “Microservices: Building Scalable Software”. Published, 2017.
[5] Sam Newman. “Monolith to Microservices Evolutionary Patterns to Transform”. Published, October 2019.
[6] Le Quoc Hung. “Cloud Computing Advantages and Disadvantages, Issues and its Application”. Published, 2019.
[7] Barrie Sosinsky. “Cloud Computing Bible” Published, 2011.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY