IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Innovative Approaches to Serverless Computing: A Novel Architecture Perspective

[1]Salu George Thandekkattu, [2]Kalaiarasi Mani, [3]N.Anusha
[1]Professor, [2]Assistant Professor, [3]HOD
[1]American University of Nigeria, Yola, Nigeria, [2] GRIET, Hyderabad, Telangana
[3]Department of CSE(AI&ML)
Vidya Jyothi Institute of Technology
Hyderabad, Telangana

***Abstract*:** This paper presents the architectural framework for a Serverless computing service made especially for edge clouds. The proliferation of edge computing has introduced new challenges and opportunities for deploying applications closer to the data source, thereby reducing latency and improving user experience. However, traditional serverless computing platforms designed for centralized clouds may not be optimized for edge environments. This work proposes a novel architecture and features to address the unique requirements of edge cloud deployments.
Keywords: Edge Computing, Cloud Computing, Serverless Computing Architecture, Function-as-a-service, cost optimization.

I. Introduction

Processing and storing data closer to the users and applications is the concept underlying edge computing. Ultimately, it lowers latency and provides protection against internet disruptions. Edge computing designs are expected to enable breakthroughs such as Connected Smart homes, vehicle automation, Surgeries carried with the help of Robots, Gaming etc. Cloud computing moves from the data center to the network thanks to mobile edge computing, sometimes referred to as multi-access edge computing [6].

An alternative architecture to cloud computing is provided by edge computing for applications that require high availability and fast speed. This is so that programs that only use the cloud to store and process data are not immune to the inherent unreliability of internet connectivity, but rather grow dependent on it. The application as a whole lags or malfunctions when the internet slows down or stops working. Edge computing increases application performance and availability by placing data close to its point of production and consumption, hence reducing internet needs. Geo-distributed, real-time latency-sensitive applications with high computational demands are being met by the new computing paradigm known as "Fog Computing." Fog computing serves as a bridge between sensors and the Internet of Things, bringing cloud computing, storage, and networking capabilities [2][5][19].

The motivation behind the design of a serverless computing for edge clouds, is that the applications can perform better and provide a better user experience when they leverage serverless cloud edge functions. The platforms, ideas, and crucial pointers for putting edge functions into practice are listed below [3].

Initially, identify whether low-latency processing, real-time data analysis, or geographic distribution are actually required for the application. This will make it easier to decide if edge functions are the best option for the undertaking.

In this time frame, it's critical that developers understand the principles of edge computing and how it varies from conventional server-based architectures or serverless features.

Due to the many benefits that serverless architectures provide, an increasing number of businesses are implementing a serverless-first approach. Sector leaders enthusiastically welcome these advantages, which include reduced costs, scalability, and enhanced production. Developers can fully utilize these state-of-the-art technologies to spur creativity and project success by investing in knowledge and skills.

Edge functions are a cutting-edge technology that enables programmers to run code right at the edge of the network. They are also known as serverless computing or Functions-as-a-Service (FaaS). To put it briefly, these features allow certain tasks to be completed in closer proximity to the end users, which leads to quicker response times and better performance all around [4][5][7].

Although edge computing is not a novel idea, it is now more widely available than it has ever been thanks to developments in networking and cloud infrastructure. Developers can now easily deploy their code across geographic boundaries by utilizing contemporary tools like globally distributed data centers and content delivery networks (CDNs).

To put it simply, edge functions offer a scalable and adaptable way to run application logic without having to worry about maintaining the supporting infrastructure. As a result, developers are free to concentrate on writing code and can take advantage of improved user experience, increased security, decreased latency, and cost optimization [1].

Whereas, Serverless alters both the technical and conceptual aspects of how we develop applications; these modifications will facilitate the adoption of Edge Computing. That means we can create services and apps using serverless architecture without worrying about the underlying servers. Although the name isn't perfect, it encompasses a wide range of services and is fundamentally an architectural movement aimed at reducing Total Cost of Ownership (TCO) and increasing agility.

Serverless is, to put it simply, an abstraction. We've abstracted away the details of the underlying machine to enable a model where pure application code is transmitted to the cloud provider (like AWS) and executed in response to various events, using FaaS.

The most well-known serverless service is Lambda, also known as the "poster child" for serverless computing. It is an AWS Function as a Service (FaaS). The variety of triggers available for Lambda functions has allowed elegant event-driven architectures to flourish, and lambda has fundamentally changed the way we can develop applications [8][10].

II. BACKGROUND

Consider web apps as an illustration. At first, a large portion of the logic was server-side. A framework and templating engine would take care of inserting the necessary data from a database into the dynamic portions of a webpage as soon as it was loaded.

After rendering, the HTML page would be sent to the client and shown there. SPA frameworks like React and Vue emerged as a result of the shift toward ever-more-complex user experiences, and a significant amount of business logic was shifted to the client side. We soon noticed that this was having a negative impact on SEO and the user experience, as time went on. But Server-Side Rendering, or SSR, was "reinvented" by frameworks like React, which led to a return to same architecture again.

Our approach to developing entire applications has undergone a comparable cycle. When more powerful PCs became available, computing shifted from mainframes to the client side. Currently, there is a return to central cloud providers.

Therefore it is evident that we're going to go through another cycle, this time to the Edge. Edge environments consist of heterogeneous and resource-constrained devices. Research is needed to develop efficient resource allocation and management techniques that optimize the utilization of limited edge resources. This includes dynamic resource provisioning, workload placement algorithms, and load balancing mechanisms tailored for edge environments.

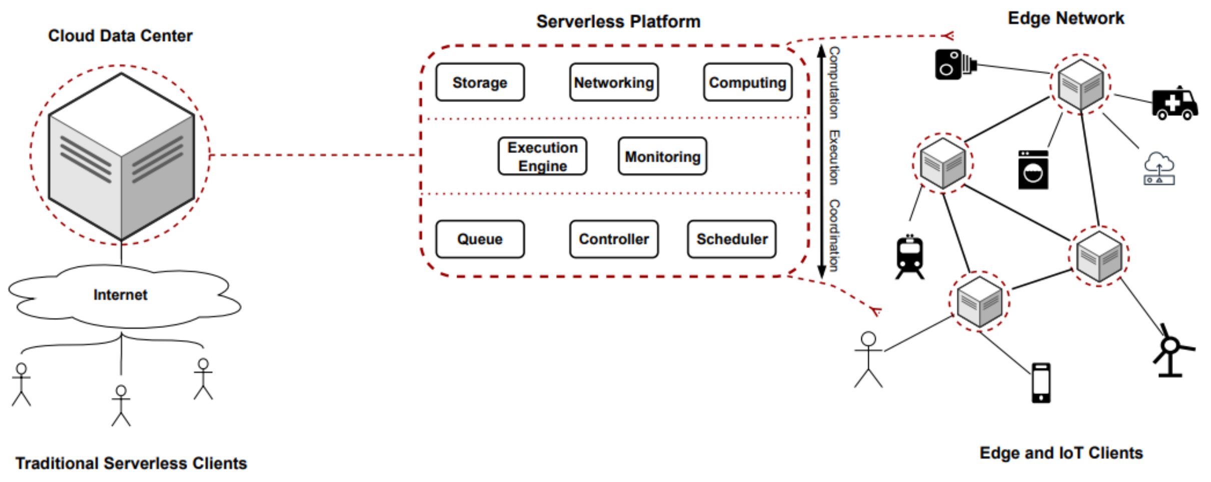
Edge computing emphasizes low-latency and real-time processing. Research is required to optimize function execution, reduce function startup time, and minimize resource usage. Techniques such as function packaging, code optimization, and caching strategies specific to edge environments can significantly enhance the performance of serverless functions [10][11][12].

Edge environments enable innovative applications in various domains such as IoT, autonomous systems, smart cities, and augmented reality. Further research is needed to explore how serverless computing can effectively support these edge-enabled applications, considering the unique requirements and challenges they present [16].

Edge environments introduce new security and privacy challenges. Research is essential to develop robust security mechanisms for secure function execution, data storage, and communication. Techniques such as secure enclaves, encryption, access control, and privacy-preserving algorithms need to be explored and refined for edge serverless platforms [20].

We can fully utilize serverless computing in edge contexts and enable low-latency, scalable, and secure application execution at the edge by making additional research and development investments. This research-driven strategy will accelerate the adoption of edge computing across multiple disciplines and open the door for creative edge-enabled applications.

Figure 1: Serverless Edge Computing



2.1 Architecture Design

Propose an architecture that combines the principles of serverless computing with the distributed nature of edge clouds.

2.2 Creating a novel architecture for serverless edge computing.

Today, edge computing services are provided by numerous large cloud service providers. For a range of use cases, AWS, for instance, has launched an extensive suite of services that enable edge computing. They essentially allow data centers to be set up locally in particular cities, on premises, and/or within 5G networks, extending their cloud infrastructure to the edge.

These kinds of services from AWS and other cloud service providers give edge computing projects additional choices, flexibility, and ease of use. In turn, by utilizing on-demand infrastructure, these services enable your business to get up and running quickly. They also help it evolve smoothly by upholding a consistent, repeatable environment.

On the Edge of Data Processing. Using an edge-ready database is essential for success with edge computing architectures.

The concept of edge functions, lightweight compute units that can be executed closer to the data source.

For edge functions to be implemented successfully, platform selection is essential. Take into account elements like community support, pricing model, language support, ease of use, and integration with current services or infrastructure.

Functions in cloud environments that are overseen by outside vendors such as Google Cloud Functions and AWS Lambda [9].

2.3 Edge operations in cloud serverless computing

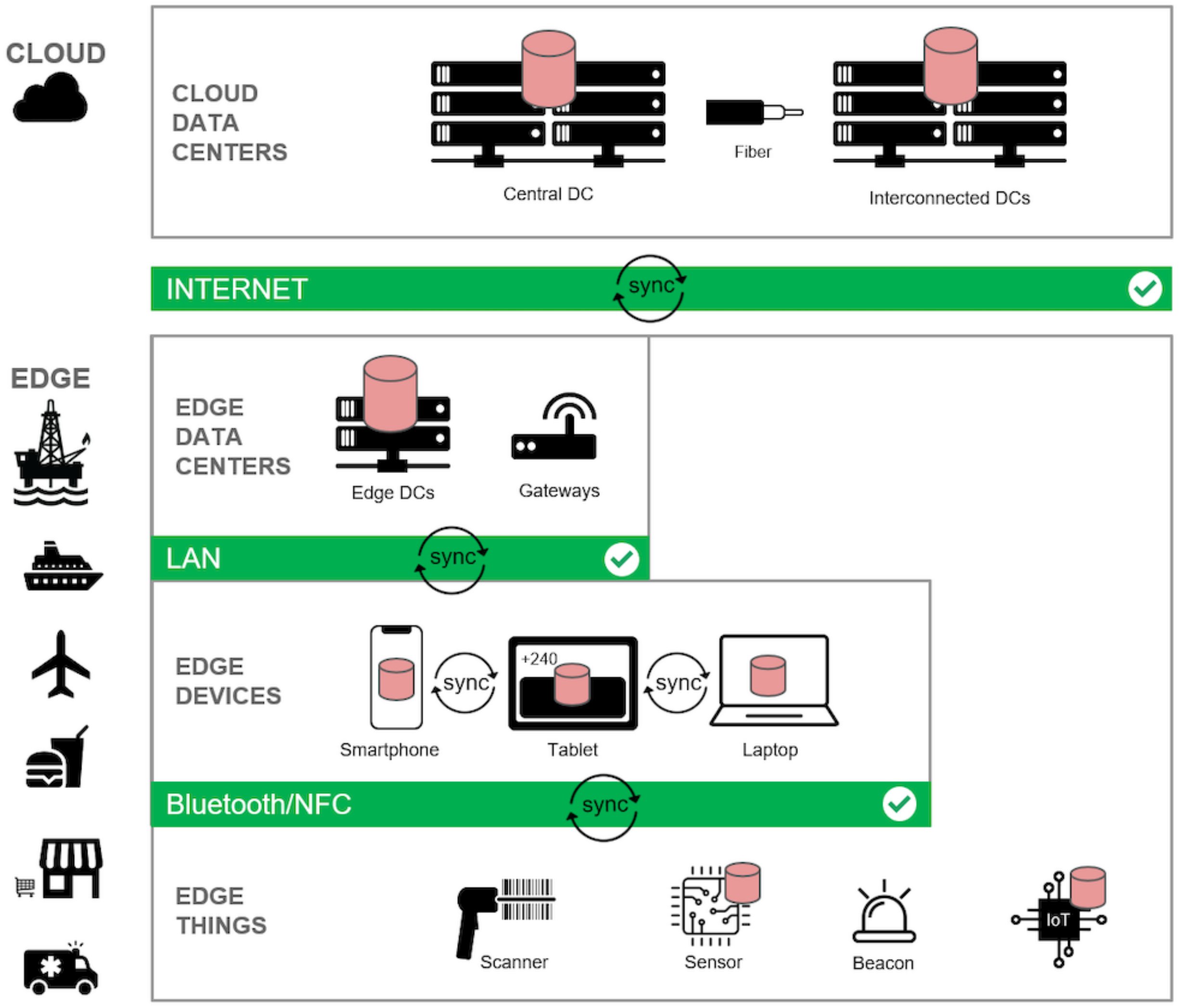
Flexibility and scalability: Serverless functions are appropriate for applications with variable workloads that do not necessarily require ultra-low latency or geographical distribution because they automatically scale with demand.

Infrastructure management: It should be kept to a minimum. Serverless functions provide a streamlined development process by having third-party providers handle resource allocation and maintenance, freeing developers to concentrate on writing code instead of worrying about managing the underlying infrastructure.

Cost Optimization: Optimizing costs is a top concern. Serverless functions that use a pay-as-you-go model only bill for the resources used while they are running, doing away with the need for idle servers or pre-allocated infrastructure.

Developing microservices and APIs: Serverless functions are ideal for developing microservices, APIs, event-driven architectures, and background tasks (like resizing images) where ultra-low latency and geographic distribution are not as important considerations.

In the realm of cloud computing, serverless computing has made a name for itself as an affordable, flexible, low-footprint option for transient applications. Its suitability for edge computing powered by Internet of Things applications is still up for debate. We thoroughly examined the benefits of pushing serverless technology to the limit as well as any potential roadblocks to such achievements in this paper.



To highlight the locations of data processing and storage, red database icons are added to this version of the edge computing architecture. This is the place where we process the serverless computing functionalities, since we can't interrupt the basic principle of edge computing. This architecture will ensure the basis of edge computing advantages and combine with serverless computing. The latency speeds up response times and enhances performance by significantly reducing latency caused by users' close proximity. Maybe not as effective at reducing latency as edge functions, but it does offer resource management and auto-scaling [13].

Users and devices can always access data quickly with an edge computing architecture, even in the case of internet outages or latency. And the key to making it all happen is your database.

Comparison table: the performance, latency, and resource utilization with existing serverless platforms in centralized clouds.

III. REAL WORLD EDGE COMPUTING

Content optimization and personalization: They can dynamically modify content according to device type, location, and user preferences, guaranteeing a customized experience for every visitor. This covers changing the size of images, providing content in multiple languages, and even tailoring product recommendations on e-commerce sites.

Gaming and video streaming: By lowering latency and buffering problems, Edge features greatly enhance these activities. These services ensure seamless playback and immersive experiences by processing requests closer to the user, allowing for the delivery of high-quality content with minimal delays.

Security and authentication: By carrying out authentication checks and screening out malicious traffic at the network's edge before it reaches the application's core infrastructure, edge functions enhance security measures.

Microservices and serverless APIs: Using edge functions, developers can create serverless APIs and Microservices that automatically scale with demand while preserving low-latency responses. This method ensures effective resource utilization while streamlining development processes[14][15].

A/B testing: Businesses can also perform experiments and A/B testing on their websites with Edge functions, all without affecting the overall performance. This implies that by handling user interactions at the edge, they can quickly acquire insights to maximize their online presence [17].

The use cases that edge functions can have are far more than what these examples show. Anticipate even more cutting-edge applications that take advantage of edge computing's potential to provide outstanding performance, scalability, and user experiences for a variety of industries as this technology develops [16][20].

3.1 Objective

The clear picture can be obtained from Taxonomy of identified opportunities and open issues for Serverless Edge Computing. This work concentrates on of the open issues, i.e. Cost Optimization. According to cloud-domain estimates [14], the main factor driving Serverless popularity is its cost-effectiveness (41%) as opposed to its performance (23%) and its straightforward pay-per-use business model make it appealing. But (1) high performance is needed for mission-critical IoT use cases, like healthcare, in order to anticipate major incidents before they really happen. (2) Because edge computing frameworks are designed with a specific purpose in mind, they are likely to use open-source and proprietary solutions that are not financially motivated. Additionally, this might lessen the importance of cloud offerings in the market going forward, doing away with the financial concerns associated with public cloud offerings. For example, a private farm with local resources and Serverless enabled won't pay itself. Rather, their deployment concerns will likely move from cloud cost-efficiency to edge high performance.

3.2 Achieving cost optimization in serverless computing involves several strategies, as below

The study follows the below approaches to help optimize costs in a serverless environment, as per the objective taken here.

a. Understand the resource requirements of your serverless functions and allocate the appropriate amount of memory and compute power. Avoid over-provisioning resources, as this can lead to unnecessary costs. Regularly analyze and adjust resource allocation based on actual usage patterns and performance requirements, which means right-sizing resources.

b. In the next step, we take, Function Runtime Duration. Optimize the execution time of your functions by analyzing and fine-tuning the code. Minimize unnecessary operations, avoid redundant computations, and optimize data processing to reduce the overall runtime duration. Shorter execution times result in lower costs.

c. Next is how to Auto-Scale? Leverage auto-scaling capabilities provided by serverless platforms. Configure scaling parameters based on workload patterns and anticipated demand. Auto-scaling ensures that resources are dynamically allocated to match the workload, preventing over-provisioning during periods of low usage and avoiding performance bottlenecks during high-demand periods.

d. Request Batching: When possible, aggregate multiple requests into a single function invocation. By processing requests in batches, you can reduce the number of function invocations, which helps optimize costs. However, consider the trade-off between reduced invocations and the potential impact on latency and response times.

e. Provisioned Concurrency: Some serverless platforms offer the option to provision concurrency, which keeps a certain number of function instances "warm" and ready to handle requests instantly. This reduces the latency associated with cold starts but comes at an additional cost. Evaluate the benefits of reduced latency against the added expense based on your specific use case.

f. Resource Cleanup: Ensure that your functions release any acquired resources promptly. Avoid unnecessary or excessive resource usage to prevent incurring additional costs. For example, close database connections, release file handles, and free memory as soon as they are no longer needed.

g. Monitoring and Optimization: Monitor the performance, usage, and costs of your serverless functions using built-in monitoring tools or third-party solutions. Regularly analyze the collected data to identify bottlenecks, inefficiencies, or potential areas for optimization. Optimize function configurations, memory allocation, and resource utilization based on these insights.

h. Serverless Framework Selection: Evaluate different serverless platforms and frameworks to find the one that aligns with your cost optimization goals. Compare pricing models, compute pricing, storage costs, and any additional fees. Consider factors such as vendor discounts, free tiers, and available cost management tools.

i. Stateless Design: Embrace a stateless design approach for your serverless functions. Avoid storing and managing persistent state within the function instances, as it can lead to increased costs and complexity. Leverage external storage services, such as databases or object storage, for persistent data storage.

j. Caching: Implement caching mechanisms to reduce the need for repetitive computations or data retrieval. Utilize caching at various levels, such as function level, API gateway level, or content delivery networks (CDNs). Caching can significantly improve performance and reduce the costs associated with repeated computations or data retrieval.

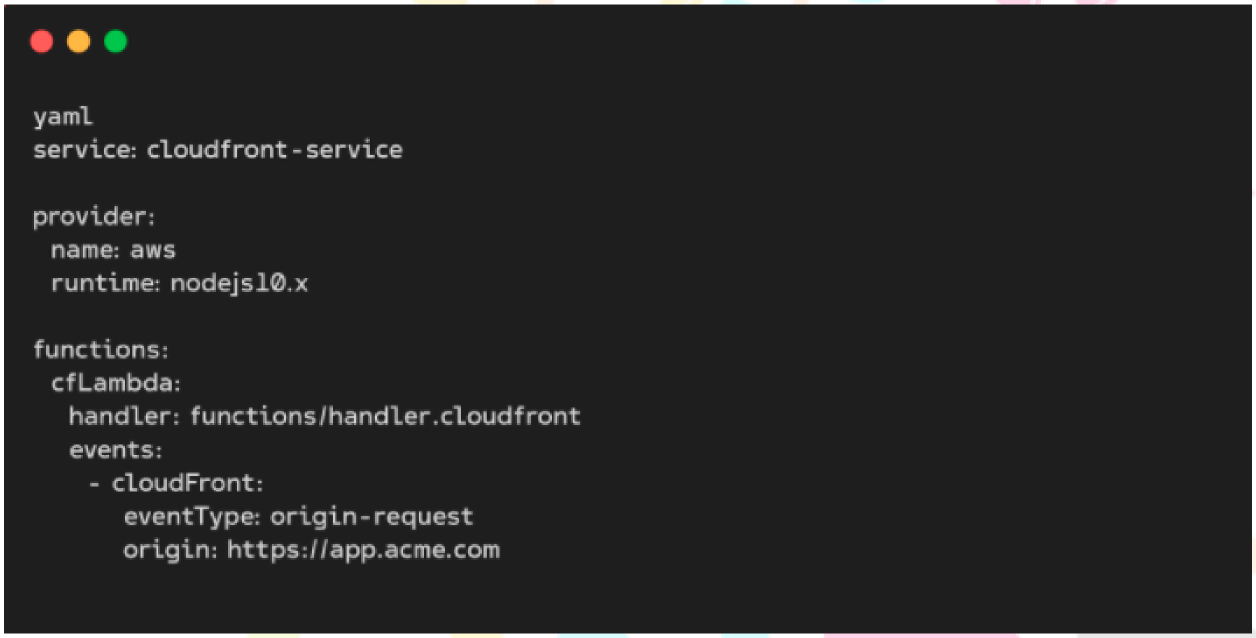
k. Cost Analysis and Optimization: Regularly review and analyze your serverless costs. Identify cost drivers, inefficient functions, or areas where optimization efforts can be applied. Explore cost optimization features provided by your serverless platform, such as cost allocation tags, budget alerts, and reserved instances.

It is observed that by adopting these strategies and continuously monitoring and optimizing your serverless environment, you can achieve cost savings while maintaining the scalability and flexibility benefits of serverless computing [18][21].

IV. THE EDGE ALREADY OFFERS SERVERLESS SERVICES WITH LAMBDA@EDGE

l. In reality, Lambda@Edge is not a part of AWS's "Serverless" team. Indeed, it's a CloudFront (CDN) offering. By operating in the CDN layer, Lambda@Edge enables clients to run application code closer to users. It's a lot like "traditional" Lambda in that we only pay while the code is executing and we don't have to worry about maintaining the infrastructure.

m. However, Lambda@Edge is frequently used by developers for security audits. Also context-specific data modification, location routing, and SSR of React applications closer to the users, the traditional image transformation use-case, and even some simple A/B testing.

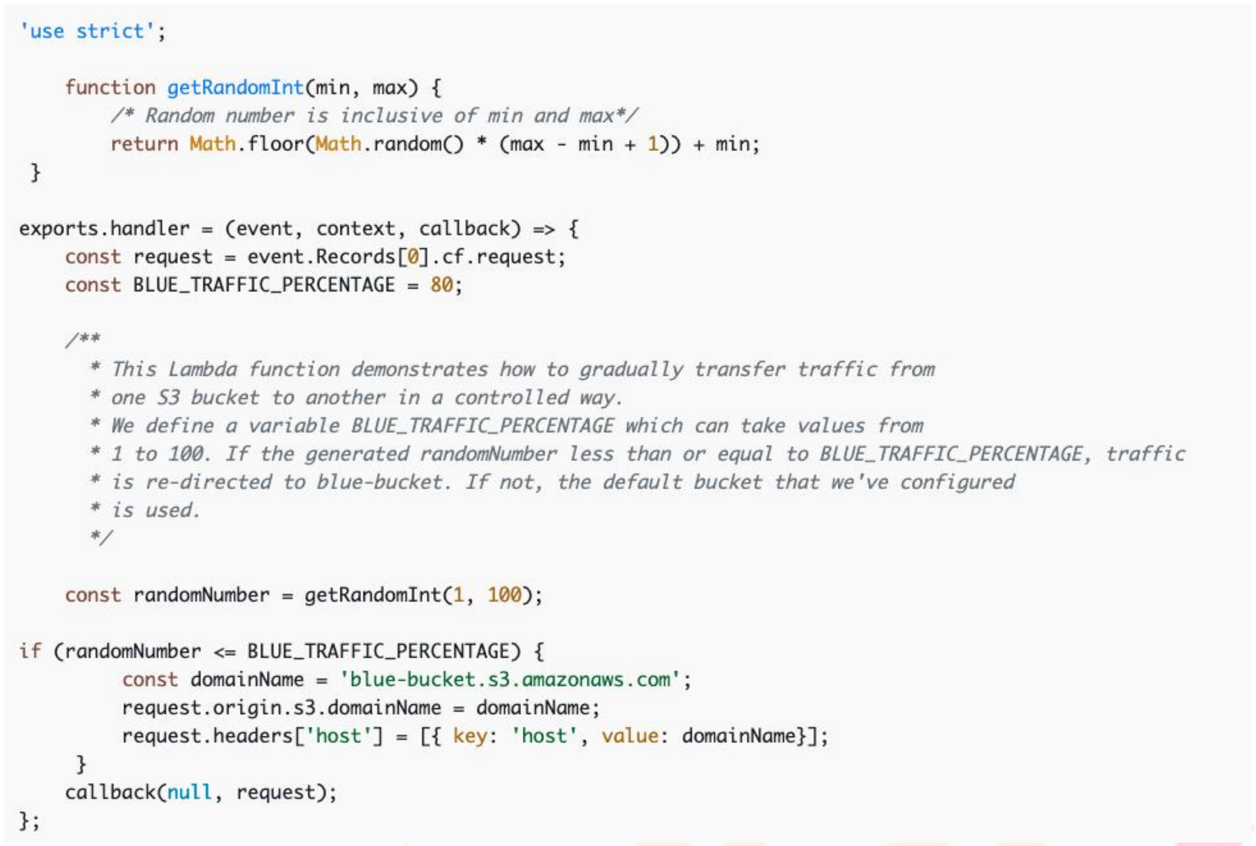


```
yaml
service: cloudfront-service

provider:
  name: aws
  runtime: nodejs10.x

functions:
  cfLambda:
    handler: functions/handler.cloudfront
    events:
      - cloudFront:
          eventType: origin-request
          origin: https://app.acme.com
```

We can specify our function, indicate that it is triggered by a CloudFront event, and then execute a single serverless deploy command using the Serverless Framework. The code itself does not require manual bootstrapping or virtualization, and it can be written in Python or JavaScript. For example, with a few lines of code, we could progressively move traffic from one S3 bucket to another.

The process for developers is the same as when creating Lambda serverless apps. The only things that are different are that more restrictions are placed on the execution time, CPU, and memory, and it must be deployed to US-East-1 beforehand. Additionally, deployment and rollback times are prolonged because of distribution time, and only Python or JavaScript are supported (no other languages or Custom Runtime options) [8].

An illustration of how the edge's execution environment and the serverless execution model work well together.



```
'use strict';

    function getRandomInt(min, max) {
        /* Random number is inclusive of min and max*/
        return Math.floor(Math.random() * (max - min + 1)) + min;
 }

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const BLUE_TRAFFIC_PERCENTAGE = 80;

    /**
      * This Lambda function demonstrates how to gradually transfer traffic from
      * one S3 bucket to another in a controlled way.
      * We define a variable BLUE_TRAFFIC_PERCENTAGE which can take values from
      * 1 to 100. If the generated randomNumber less than or equal to BLUE_TRAFFIC_PERCENTAGE, traffic
      * is re-directed to blue-bucket. If not, the default bucket that we've configured
      * is used.
      */

    const randomNumber = getRandomInt(1, 100);

if (randomNumber <= BLUE_TRAFFIC_PERCENTAGE) {
         const domainName = 'blue-bucket.s3.amazonaws.com';
         request.origin.s3.domainName = domainName;
         request.headers['host'] = [{ key: 'host', value: domainName}];
     }
    callback(null, request);
};
```

V. EDGE WILL BE MADE POSSIBLE BY SERVERLESS

It is evident that serverless technologies are already being utilized by Edge computing services. This trend will continue, but it pales in comparison to the conceptual evolution that Serverless has sparked.

The paradigm shift that Serverless sparked in developers was a pay-per-use, distributed, stateless mindset. The adoption of Edge compute will also be made possible by this paradigm change. In addition, a lot of the tools and techniques we have been working on for Serverless will serve as the foundation for Edge Compute. The Edge Computing Revolution requires the same changes that serverless alters in the way we develop applications [22][23][24].

VI. CONCLUSION

Computation, execution, coordination are the key contributions of the proposed design for a serverless computing service in edge clouds. These features of Serverless Computing are combined on the database layers of edge computing, keeping in mind the base principle of edge computing, so that the latency scalability and optimization are achieved as certain.

Potential future directions, such as scalability, interoperability with existing cloud platforms, and support for edge-enabled applications.

VII. Acknowledgment

References

[1] Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30-39.
[2] Khan, Z., Anwar, M., Madani, S. A., & Dou, W. (2020). Fog computing: Recent advances, research challenges, and future directions. Journal of Ambient Intelligence and Humanized Computing, 11(10), 4301-4326.
[3] Srikumar, V., Dwarkadas, S., Sabella, D., Ghedini, C., & Ghafoor, A. (2020). Edge computing: Vision and challenges. IEEE Cloud Computing, 7(2), 52-62.
[4] Bhatti, U., Bakken, D. E., & Magne, P. (2019). A systematic review on edge computing: Vision, trends and challenges. Future Generation Computer Systems, 97, 219-237.

[5] HaddadPajouh, H., Pedersen, J. M., & Mogensen, P. (2019). Bringing cloud to the edge: A review on fog computing architecture and opportunities. Future Generation Computer Systems, 92, 255-265.
[6] Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 19(4), 2322-2358.
[7] OpenFaaS: Serverless Functions Made Simple. (n.d.). Retrieved from https://www.openfaas.com/
[8] AWS Lambda: Serverless Compute. (n.d.). Retrieved from https://aws.amazon.com/lambda/
[9] Google Cloud Functions. (n.d.). Retrieved from https://cloud.google.com/functions
[10] Azeez, S., Elkhatib, Y., & Race, N. (2020). Function-as-a-Service: Research Landscape and Challenges. In 2020 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP) (pp. 19-26). IEEE.
[11] Please note that the references provided are generic and may not directly correspond to the specific content of the design proposal outlined earlier. It's recommended to conduct a comprehensive literature search using academic databases, such as IEEE Xplore, ACM Digital Library, or Google Scholar, to find more recent and relevant research papers on serverless computing in edge clouds.
[12] OpenWolf: A Serverless Workflow Engine for Native Cloud-Edge Continuum 2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)
[13] A Serverless Computing Fabric for Edge & Cloud 2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)
[14] TAROT: Spatio-Temporal Function Placement for Serverless Smart City Applications 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)
[15] Mobility-Aware Serverless Function Adaptations Across the Edge-Cloud Continuum 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)
[16] Serverless Vehicular Edge Computing for the Internet of Vehicles IEEE Internet Computing
[17] Securing Serverless Workflows on the Cloud Edge Continuum 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)
[18] COGNIT: Challenges and Vision for a Serverless and Multi-Provider Cognitive Cloud-Edge Continuum 2023 IEEE International Conference on Edge Computing and Communications (EDGE)
[19] eScience Serverless Data Storage Services in the Edge-Fog-Cloud Continuum 2023 IEEE 19th International Conference on e-Science (e-Science)
[20] EdgeOrcher: Predictive Function Orchestration for Serverless-Based Edge Native Applications 2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)
[21] Behavior Tree-based Workflow Modeling and Scheduling for Serverless Edge Computing 2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)
[22] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi et al. "Serverless Edge Computing: Vision and Challenges", 2021 Australasian Computer Science Week Multiconference, 2021.
[23] Gul Agha, Dipayan Mukherjee, Atul Sandur. "Performance, Energy and Parallelism: Using Near Data Processing in Utility and Cloud Computing", 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), 2022.
[24] Philipp Raith, Thomas Rausch, Alireza Furutanpey, Schahram Dustdar. "A trace- driven simulation framework for serverless edge computing platforms" , Software: Practice and Experience, 2023.

International Research Journal
IJNRD
Research Through Innovation