Cloud Computing- Serverless Computing and Function as a Service. (FaaS)

Calvin Ssendawula
Adams State University
208 Edgemont Blvd. Unit 890
Alamosa. Colorado 81101 ssendawulac@adams.edu

***Abstract*—Serverless computing has emerged as a cloud paradigm that allows developers to run code without managing servers, Function as a Service offerings like AWS Lambda. This paper provides an exploration of serverless computing focusing on real-world applications, performance comparisons with traditional architectures and across platforms, and future trends.**

We discuss how serverless architectures are employed in production from web and mobile backends to IoT data processing – and examine performance characteristics such as scalability, latency, and cost. AWS Lambda is used as a representative example of technology alongside other platforms (Azure Functions, Google Cloud Functions), illustrating commonalities and differences. We include case studies (e.g., Coca-Cola's vending machine platform and iRobot's IoT backend) that demonstrate the benefits and challenges of going serverless in practice. Finally, emerging trends and research directions are highlighted, indicating how serverless computing is evolving to address current limitations (such as cold start latency, tooling, and state management) and to broaden its applicability. The presentation is formal and informative, with a clear structure and references to relevant literature and industry sources to aid further reading.

***Keywords*— AWS Lambda, Serverless, scaling, trigger events, Function as a Service, Google Cloud functions**

I. Introduction

Cloud computing continues to evolve from low-level Infrastructure-as-a-Service (IaaS) offerings toward more managed and granular services. *Serverless computing* – often manifested through Function-as-a-Service – is the latest step in this evolution, freeing developers from operational concerns like provisioning or scaling servers. In a serverless model, developers deploy functions (small units of code) that are executed on-demand in a fully managed environment, typically triggered by events. The term "serverless" is somewhat a misnomer; servers exist but are abstracted away by the provider. This approach lets developers focus on application logic rather than infrastructure management. Function-as-a-Service (FaaS) platforms have rapidly gained traction since their introduction. Amazon launched AWS Lambda in 2014 as the first FaaS service, and other cloud providers like Google Cloud Functions and Microsoft Azure Functions followed by 2016. These services enable an *event-driven* architecture where functions automatically run in response to events (HTTP requests, file uploads, database updates, IoT sensor readings, etc.), with the cloud provider handling resource allocation and scaling transparently. Industry adoption has grown quickly; many organizations now run production workloads on serverless platforms, attracted by benefits such as automatic scaling, fine-grained billing, and faster development cycles. At the same time, researchers and practitioners have identified challenges that come with this new paradigm – for example, performance variability and tooling gaps – which we will discuss in depth.

This paper provides a comprehensive overview of serverless computing and FaaS, in the style of an IEEE technical report. We survey the architecture and design of serverless platforms, highlight real-world use cases and deployments, compare performance across different FaaS providers and against traditional server-based architectures, and discuss future trends. By analyzing both academic studies and industry case studies, we aim to give readers a balanced understanding of when and how serverless architectures are beneficial, and what trade-offs to consider.

II. Related Work & Application

In a serverless FaaS platform, the underlying architecture is built to automatically manage function execution in response to events. Understanding the design of serverless systems helps to appreciate their performance characteristics and limitations. This section describes the general architecture of FaaS, including how functions are invoked, isolated, scaled, and integrated with other services.

Function Invocation Model: In a serverless setup, developers deploy one or more *functions* along with configuration specifying what events should trigger each function. Common event sources include HTTP endpoints (via API gateways), message queues, database change streams, scheduled cron-like timers, cloud storage events (e.g., "file uploaded to bucket"), IoT device messages, and more. When an event occurs, the platform's *Event Router* or *Dispatcher* will locate the appropriate deployed function and route the event to it for processing. For example, AWS uses Amazon API Gateway to route HTTP requests to Lambda functions, and AWS S3 can trigger Lambda functions on object uploads. The key is that event handling is built-in – the cloud handles listening for events and calling the function, rather than requiring the developer to run a daemon or server to poll or receive events.

Execution Environment: Each serverless function runs within a sandboxed environment that the provider prepares on-demand. Typically, this is implemented with containers or micro-VMs under the hood. For instance, AWS Lambda historically used containers (based on Amazon Linux) to run functions, and more recently uses Firecracker micro-VMs to enhance security isolation with minimal startup overhead. When

a function is invoked and no warm instance is available, the platform will create a new container/VM for that function, load the code and runtime (e.g., Node.js, Python interpreter), execute the function with the event data, and then keep the instance alive for a short period in case subsequent events arrive

Scalability Mechanisms: From an architecture standpoint, the FaaS platform includes an *autoscaling controller* that monitors incoming event rates and manages the pool of function instances. All major providers advertise virtually unlimited scaling – in practice, they have default safety limits (for example, AWS Lambda default is 1000 concurrent executions per account per region, which can be raised on request)

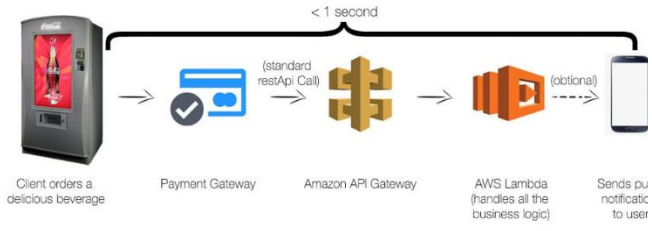
Integration with Other Services: A serverless function rarely lives in isolation – it often needs to interact with other services (databases, APIs, caches, etc.). Cloud providers offer extensive integration, essentially composing FaaS with BaaS. For example, a typical web application in AWS might use API Gateway + Lambda + DynamoDB (a NoSQL database). Notably, all these components can be serverless: API Gateway is a managed service to accept HTTP calls, Lambda is FaaS, and DynamoDB in on-demand mode can autoscale throughput. As one article notes, *"each of the AWS components in the diagram… are considered serverless"* – the API gateway, the function, and the database all scale automatically and require no customer-managed servers

Case Study 1: Coca-Cola's Serverless Vending Machine Backend

Coca-Cola is a globally recognized brand, and their vending machines serve millions of customers. Around 2016, Coca-Cola North America undertook an initiative to migrate some of their digital services to a serverless architecture. One key system is the remote monitoring and marketing system for vending machines. Traditionally, a fleet of always-on servers (Amazon EC2 instances) was used to handle machine telemetry and promotional campaigns (such as updating a machine with "buy one get one free" offers, or notifying technicians when a machine is low on stock) This system ran on 6 EC2 virtual machines with load balancers, costing about $12,864 per year to operate, including maintenance and management overhead.

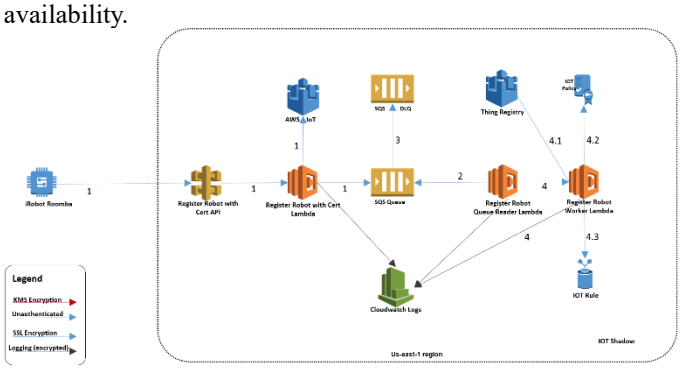
In the serverless revamp, Coca-Cola moved this functionality to AWS Lambda (FaaS) along with other AWS services like API Gateway and DynamoDB. Now, when a vending machine reports an event (e.g., a sale or a status update), it triggers an API Gateway endpoint which invokes a Lambda function that contains the business logic.

Figure 1: Architecture of Coca-Cola's serverless vending machine application. A drink purchase triggers a Payment Gateway (standard REST API call) which goes through the Amazon API Gateway to invoke an AWS Lambda function that handles all business logic. Optionally, a mobile app is notified of the transaction. The end-to-end interaction completes in under one second.



Case Study 2: iRobot's Serverless IoT Cloud Platform

iRobot, the maker of the Roomba robotic vacuum, transitioned to a serverless backend to support their internet-connected devices. When iRobot introduced Wi-Fi enabled Roombas, they needed a cloud platform that could reliably connect thousands (eventually millions) of robots, handling sporadic bursts of activity – for example, many users activating their new vacuums during a holiday season. Initially, iRobot tried a turnkey IoT cloud solution but found it lacking in scalability and control. They decided to migrate to AWS and crucially chose a serverless architecture for their IoT backend. In iRobot's architecture, each Roomba connects via AWS IoT Core (MQTT messaging broker). The IoT messages from devices trigger AWS Lambda functions that implement various backend logic – such as registering a device, sending commands to the robot (start cleaning, stop, dock, etc.), and processing telemetry data coming from the robot. The platform also exposes web APIs (for the mobile app) which are backed by Lambda functions. The use of serverless was motivated by the need for *massive scalability* and not having to worry about infrastructure – as iRobot's Cloud Robotics research scientist put it, AWS offered tools "that enable us to use a serverless architecture that saves us the headaches of learning to scale". With unpredictable surges (like a big sale where thousands of new robots come online), the automatic scaling was essential to maintain availability.



III. RESULTS

The results of Coca Cola migration were impressive. In terms of cost, the serverless solution cost approximately $4,490 per year (for around 30 million requests per month) – roughly 65% cheaper than the previous static server setup. The Coca-Cola cloud team noted that the break-even point where running their own servers would start to make sense economically was around 80 million calls per month (far above their normal traffic at the time). This means serverless provided substantial cost savings up to very high scale. Performance-wise, the system achieved an end-to-end latency of under 1 second for a vending machine transaction to be processed and confirmed. This included third-party calls (to payment APIs and push notification services). The serverless backend easily scaled during peak usage (for example, during promotions or seasonal spikes), without any manual intervention to add servers. Another benefit reported was the increased speed of development and deployment – new features or changes could be rolled out as isolated functions without affecting the whole system. An interesting consequence of the success of this project was cultural: Coca-

Cola's technology leadership was so satisfied with the outcomes that they set a strategic direction to pursue a "Serverless First" approach for new application. When architects present new ideas internally, they are expected to consider serverless implementations as the default, unless a strong case for an alternative is made. Of course, not everything can be serverless (Coke still had to keep some EC2 instances to support older machines until they are phased out), but this case study demonstrates how a large enterprise can leverage FaaS to modernize part of their infrastructure, significantly reduce costs, and improve maintainability.

By using a serverless approach with AWS Lambda and other managed services, iRobot achieved several benefits; The platform easily handled surges in device connections and user interactions. When a spike of events happened (e.g., many users scheduling cleanings on Christmas morning), AWS Lambda automatically ran the needed functions in parallel. There was no need to manually add servers or capacity; the system scaled out and back down smoothly. The serverless model kept costs low relative to the volume of activity. Importantly, iRobot avoided having to build an expensive always-on infrastructure that would be underutilized much of the time. As noted in an AWS case study, "by using a serverless architecture based on AWS IoT and AWS Lambda, iRobot is able to keep the cost of the cloud platform low, avoid the need for subscription services, and manage the solution with fewer than 10 people". Freed from infrastructure concerns, iRobot's developers could focus on features and product improvements. The serverless backend let them concentrate on code and customers rather than operations. For example, they could rapidly develop new cleaning features or integration with smart home ecosystems, deploying the necessary backend functions quickly. AWS's global regions allowed iRobot to deploy their functions in multiple regions to serve customers in over 60 countries with low latency. Features like AWS Lambda's integration with CloudFront (Lambda@Edge) were also evaluated to potentially move some processing closer to where devices are.

IV. CONCLUSION

Serverless computing and Function-as-a-Service have transformed the landscape of cloud architecture by abstracting away server management and enabling highly scalable, event-driven applications. In this paper, we presented an overview of serverless computing suitable for an undergraduate understanding, covering its fundamental concepts, architecture, practical applications, performance characteristics, and future trends. We saw that serverless platforms like AWS Lambda, Azure Functions, and Google Cloud Functions allow developers to run code in response to events with automatic scaling and pay-per-use pricing. Real-world usage of these platforms ranges from web APIs and chatbots to IoT backends and big data processing, demonstrating the versatility of the paradigm. Through case studies such as Coca-Cola's vending machine backend and iRobot's IoT cloud, we illustrated concrete benefits of serverless in production: simplified operations, rapid scalability, and cost savings, without sacrificing performance for appropriate workloads. In performance comparisons, serverless solutions proved capable of matching or exceeding traditional architectures in handling bursty loads and parallel tasks, though considerations around cold start latency and execution limits must be managed. We highlighted that AWS Lambda often sets the benchmark in FaaS performance (with optimizations to minimize cold starts and robust scaling), while other platforms are quickly catching up and offering unique features of their own. The paper also discussed challenges that come with serverless computing, including execution delays due to cold starts, difficulties in debugging and monitoring distributed functions, and potential vendor lock-in. These are active areas of improvement. The future trends analysis indicates an ongoing evolution: research and development efforts are aiming to reduce latencies, introduce more flexible pricing and hybrid deployment models, support stateful and long-running scenarios, and provide better developer tools and security for serverless applications. In particular, we expect serverless to integrate more with edge computing, enabling functions to run wherever optimal (cloud or edge) to serve end-users with minimal latency. In conclusion, serverless computing has proven to be more than a buzzword – it represents a powerful cloud computing model that is likely to persist and grow. For organizations and developers, serverless offers a path to focus on innovation and functionality while leaving the undifferentiated heavy lifting of infrastructure to cloud providers. It is not a one-size-fits-all solution, and traditional servers and containers will continue to have their place. However, as serverless platforms address their current limitations, their domain of applicability will broaden. We may envision a future where deploying applications as collections of serverless functions is as common as deploying to VMs or containers is today. In educating the next generation of engineers, understanding serverless and FaaS is therefore crucial, as it has become a key part of the modern cloud computing toolbox. With ongoing advancements, serverless computing is poised to play a central role in how we build scalable, resilient, and cost-effective systems in the cloud.

REFERENCE:

1. R, Moneer. “Serverless Showdown: Aws Lambda vs Azure Functions vs Google Cloud Functions.” *Pluralsight*, 2022, www.pluralsight.com/resources/blog/cloud/serverless-showdown-aws-lambda-vs-azure-functions-vs-google-cloud-functions#:~:text=While%20cloud%20providers%20do%20not,as%20observed%20by%20industry%20analysts.
2. Randall, James. “Test Methodology.” *James Randall*, 2020, www.jamesdrandall.com/posts/comparative_performance_of_azure_functions_and_aws_lambda/#:~:text=Image.
3. Hassan, Hassan B., et al. “Survey on Serverless Computing - Journal of Cloud Computing.” *SpringerOpen*, Springer Berlin Heidelberg, 12 July 2021, journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-021-00253-7#:~:text=,comprehend%20how%20this%20technology%20works.
4. Amazon, AWS. “IRobot Case Study.” *AWS AMAZON*, 2019, aws.amazon.com/solutions/case-studies/irobot/.
5. Rehemägi, Taavi. “Serverless Framework: The Coca-Cola Case Study.” *Dashbird*, 3 Aug. 2020, dashbird.io/blog/serverless-case-study-coca-cola/#:~:text=The%20logic%20behind%20the%20vending,Android%20Pay%20or%20Apple%20Pay.
6. A. Christidis, S. Moschoyiannis, C. -H. Hsu and R. Davies, "Enabling Serverless Deployment of Large-Scale AI Workloads," in IEEE Access, vol. 8, pp. 70150-70161, 2020, doi: 10.1109/ACCESS.2020.2985282.
7. keywords: {Libraries;Load modeling;Computer architecture;Real-time systems;Predictive models;Optimization;Intelligent transportation;predicting train delays;AWS;functions as-a-service;Lambda;NoSQL;serverless;resource-constrained;serverless codebase optimization;rail traffic big data},
8. A. Corradi and A. Sabbioni, "Serverless Computing for Society 5.0," in IT Professional, vol. 26, no. 5, pp. 79-84, Sept.-Oct. 2024, doi: 10.1109/MITP.2024.3409942.
9. keywords: {Protocols;Soft sensors;Ecosystems;Serverless computing;Rapid prototyping;Security;Reliability;Sustainable development;Standards;Resilience},

USA, 2018, pp. 125-126, doi: 10.1109/PAC.2018.00022. keywords: {Servers;Privacy;Encryption;Data privacy;Resistance;Privacy Preservation, Location Based Services, Identity-Based Encryption},