Cloud Based Serverless Web Application*

1st Abhinav Mishra
Honors Bachelor of Science Major in Computer Science
Lakehead University
Thunder Bay, Canada

***Abstract*—The purpose of this project is to develop a self-scalable, serverless cloud web application. Apart from fulfilling the use case of the web application, cloud technologies can be easily justified for any type of web development for the number of benefits it brings when it comes to infrastructure, costs, and performance. Consequently, I've opted to employ AWS for constructing the entire infrastructure, utilizing Infrastructure as Code (IaC), establishing a comprehensive development workflow with Continuous Integration/Continuous Deployment (CI/CD) and version control. This is complemented by a microservices-oriented design, incorporating NodeJS, ReactJS, and Python [1].**

***Index Terms*—Amazon API Gateway, Amazon Dynamo DB, Amazon S3, AWS, AWS Lambda, Cloud Computing, Cloud Storage, S3.**

I. Introduction

Cloud computing enables access to computing resources, such as servers and storage, over the Internet. It offers Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) models. This revolutionary technology has transformed how businesses manage their infrastructures and deliver services to users, offering greater agility and scalability.

In today's world server computing is commonly practiced for organizations, but due to server computing, it costs expensive for organizations to work on servers where they must buy servers which costs them a lot in the sense of expenses for the organization. As in many organizations, many applications are required to run once or twice a month like the salary calculation of an employee in an organization so there is the requirement of his Name, bank account no., company identity no., etc. so that the salary of the person can be calculated as per the required factors for the salary calculation such as hours worked, leave taken in the month, etc. So for such small applications, organizations have to buy servers which are very costly because they are physical devices either organization have to buy solely or use cloud services such as Amazon Web Services(AWS), Microsoft Azure, and Google Cloud Platform(GCP)[2] to buy servers which costs them highly but we have a solution to these as we are using serverless computing which is cost-effective and is more reliable than server computing in the modern world of digital management of the data.

We have understood about server less computing as the name suggests that serverless means it is not based on servers. It is managed by the cloud platform automatically. We do not have to take a look at the servers.

While we are performing our task for the organization we are working in. It is done by the service provider automatically we have to just look for our code in the Lambda function and take care of Application programming interface (API) gateway we are using for our organization So server less infrastructure does not require any servers in our presence. Developers can focus on developing code that serves the customer. They can focus on core products and business logic. Serverless applications don't require you to manage any different servers. We can focus on core products and business logic. Instead focusing on operating system access control, OS patching, scaling the servers, etc.

II. APPROACH AND METHODOLOGY

The approach for this project is to use the latest web development frameworks and the latest cloud technologies offered by AWS. As one of the leading cloud providers, Amazon Web Services offers one of the best solutions in the field.

A. Front-end development

For the front-end development, I will be using ReactJS a modern free and open-source front-end JavaScript library that allows the developer to build user interfaces based on components. Its capability for re-rendering the DOM elements only when they have changed their state makes it very efficient. Semantic UI [3] will be used for adding styling to each component. It is a CSS framework with already in-built classes to style components.

Finally, to have backward-compatible JS code, I will be using Babel [4], and to compress the JS and CSS code, webpack [5].

B. Back-end development

Regarding the back end, the code lives on AWS Lambdas. AWS Lambdas are serverless, event-driven compute services that run code without provisioning or managing servers. Once one Lambda gets triggered it will spawn the code, run it and then shut down. Although Lamba's that are recently executed stay in a 'warm' state, where the code is still loaded in case there are more consequents executions, if that is not the case it will shut down and repeat the same process once it is needed again.

My set of AWS Lambdas contains NodeJS, a JavaScript runtime environment for back-end development. Because Lambdas are small functions that have a brief lifetime, NodeJS is light enough for the Lambda to start up, execute the code and die in a short period, as well as for the Python application to handle the reports.

C. Infrastructure of the application

The application infrastructure is in AWS. Several services are used to achieve the most serverless application possible, and to have a fully scalable architecture. For the user's authentication, AWS Cognito [6] is used. This service offers user sign-up, sign-in and access control. As an entry point for the backend APIs, AWS API Gateway is used. This allowed me to create REST APIs and connect them to Lambda functions. It was also integrated with AWS Cognito to manage the permissions of each request.

For the database, I will make use of AWS DynamoDB, a NoSQL database prepared for high concurrency and connections with in-memory cache and excellent performance. The front end of the application needed to be located somewhere; in this case, instead of using an EC2 instance which would be running 24/7 even when no users are connected to it, I used an S3 bucket with CloudFront.

AWS S3 buckets can behave as web servers for static websites. But, because the back end is located behind an-API Gateway, I could run the static client-side rendering ReactJS on a simple S3 bucket which accommodates as many requests as it receives. If none, the cost of the S3 bucket would be 0. But ReactJS is not a typical static website; everything is managed by the framework itself using one single html file as an entry point. This can be a problem with S3 because it will try to map any route /whatever to the some S3 file. To solve this problem and make it compatible with ReactJS, AWS CloudFront was used in front of the S3 bucket to redirect all traffic to the index.html.

Finally, to be able to deploy all the previously mentioned architecture components I will use 2 more services from AWS. I will use AWS Code Pipeline for automating the deployment. I created hooks in the repositories of each service so whenever a new merger is performed to master branch, it will trigger the code pipeline. This process creates the application, runs tests or any other actions needed and finally pushes it to AWS CloudFormation, the latter will take care of the deployment to the different AWS components [7].

III. PROJECT DESIGN, SYSTEM MODEL

As can be seen in Figure 1, The proposed architecture outlines a streamlined process leveraging a suite of AWS services to ensure seamless functionality. At the forefront of this setup is the retrieval of static content from a designated web page hosted within an Amazon S3 Bucket, accessible to the client through a specific URL. This initial interaction establishes a user-friendly interface for accessing content.

Ensuring the integrity and security of user interactions lies at the core of the system, facilitated by Amazon Cognito.

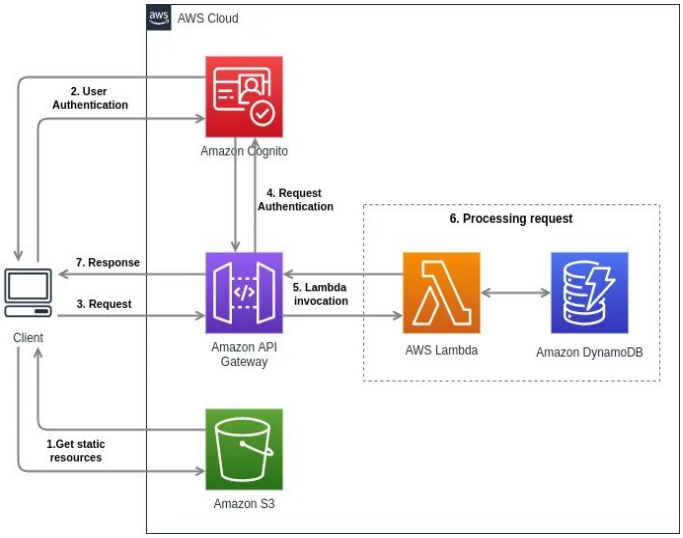

Fig. 1. System-Model

This robust authentication service not only manages user sign-up but also handles the intricacies of user login, providing a seamless and secure experience for clients. By delegating authentication responsibilities to Amazon Cognito, the system ensures adherence to best practices in user authentication and authorization, thus safeguarding sensitive user data.

For scenarios requiring dynamic content retrieval, the system seamlessly transitions to the Amazon API Gateway. Acting as a gateway for incoming requests, this service ensures that each interaction is properly authenticated through Amazon Cognito before proceeding. Once authenticated, the Amazon API Gateway efficiently directs the request to the appropriate AWS Lambda function associated with the specific endpoint.

The AWS Lambda function serves as the powerhouse of the system, executing intricate business logic tailored to the needs of the application. Upon invocation, this function orchestrates a series of operations ranging from data retrieval and manipulation to complex computations. In our scenario, the Lambda function seamlessly interacts with Amazon DynamoDB, a fully managed NoSQL database service, to store and retrieve data. This integration empowers the system to handle diverse data operations with agility and efficiency.

Furthermore, the AWS Lambda function assumes the responsibility of preparing a tailored response to each incoming request. This response, meticulously crafted based on the executed business logic, encapsulates the relevant data and insights, ensuring that clients receive timely and accurate information in response to their queries.

IV. LITERATURE REVIEW

This project has two very distinct goals. On one side, the creation of a web application tool that can be used by students and researchers to fulfil their necessities to gather data from participants. And on the other, explore the advantages of cloud development, building a cloud infrastructure with auto-scale capabilities and serverless.

Cold starts, and the delay in initializing serverless functions, have been a concern in real-time applications. Researchers have proposed strategies to mitigate cold starts, such as optimizing function packages and leveraging provisioned concurrency.

Earlier theses work either focus on the AWS or the Azure platform specifically. The thesis Multi-level FaaS Application Deployment Optimization [8] analyzes the performance when implementing a self-created framework for deployment to AWS. It discusses FaaS in general and how it is affected by deployment through a framework. Both the theses Architectural Implications of Serverless and Function- as-a-Service and Evaluation of "Serverless" Application Programming Model [9] analyzes the impact of cold starts on serverless applications and the conclusion is that it does affect the response time. The second thesis has a section discussing AWS deployment through frameworks. The Serverless Framework is mentioned and compared to AWS specific frameworks such as AWS Sam. It concludes that the Serverless Framework is one of the most mature technologies for AWS deployments. Another thesis, Serverless Development Trends in Open Source: a Mixed-Rearch Study [10] discusses current serverless trends. The thesis has analyzed serverless open-source projects on GitHub to gain insight in use cases, the complexity of the project and architectural patterns. It concludes that the primary programming languages for building serverless applications are JavaScript and Python and that most of the analyzed projects use AWS or Azure to deploy serverless applications.

No theses have been found discussing the Serverless Framework from a broader perspective, that is by deploying serverless applications to more than one provider.

My goal is to only deploy to a single provider, AWS has multiple frameworks of its own, such as Firebase and Amplify, to manage the AWS infrastructure in a serverless application. It can help a developer focus their skills on a specific provider offering. In the case of Amplify the framework can help develop both the back-end and front-end part of the serverless application.

The literature review incorporates seminal works by Roberts and Chapin, providing foundational insights into the serverless space and pinpointing areas requiring refinement, such as vendor lock- in and state management. Lynn et al.'s examination scrutinizes Function as a Service (FaaS) offerings, challenging presumed benefits based on specific use cases. Additional perspectives from Adzic and Chatley focus on production applications successfully transitioning to serverless, emphasizing the cost- saving aspects. Eivy's study compares costs between running applications on virtual machines and serverless, advocating for thorough testing before migration. For microservices adoption, presenting solutions to challenges and emphasizing alternatives like microservices and containers that complement serverless solutions. This comprehensive literature review forms the foundation for a nuanced understanding of the multifaceted landscape of serverless computing, addressing its benefits, challenges, and potential alternatives.

ACKNOWLEDGMENT

I extend my deepest appreciation to my university professor Dr. Dr. Md Moniruzzaman, for his invaluable insights, expert guidance, and unwavering support throughout the entire research process. His mentorship has been instrumental in shaping my research endeavors and refining my scholarly pursuits. This research is a collective effort, and you have played a significant role in its realization. Thank you for being an integral part of my academic journey and contributing to the success of this research review paper.

REFERENCES

[1] McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," 2017 IEEE 37th International Conference on Distributed Com puting Systems Workshops (ICDCSW), Atlanta, GA, 2017, pp. 405-410.

[2] Kotas, T. Naughton and N. Imam, "A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high-performance computing," 2018 International Conference on Consumer Electronics, Las Vegas, NV, 2018, pp. 1-4.

[3] "Getting Started — Semantic UI," semantic-ui.com. https://semantic-ui.com/introduction/getting-started.html.

[4] "What is Babel? · Babel," babeljs.io. https://babeljs.io/docs/

[5] "Concepts," webpack. https://webpack.js.org/concepts/.

[6] Narula, A. Jain, and Prachi, "Cloud Computing Security: Amazon Web Service," 2015 Fifth International Conference on Advanced Computing , Communication Technologies, Haryana, 2015, pp. 501-505.

[7] Kritikos and P. Skrzypek, "A Review of Serverless Frameworks," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, Zurich, 2018, pp. 161-168.

[8] Lynn, P. Rosati, A. Lejeune and V. Emeakaroha, "A Preliminary Review of En terprise Serverless Cloud Computing (Function-as-a-Service) Platforms," 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 2017, pp. 162-169.

[9] Swedha and T. Dubey, "Analysis of Web Authentication Methods Using AmazonWeb Services," 2018 9th International Conference on Computing, Communicationand Networking Technologies (ICCCNT), CBangalore, 2018, pp. 1-6.

[10] I. Pavlov, S. Ali, and T. Mahmud, "Serverless Development Trends in Open Source: a Mixed-Research Study," gupea.ub.gu.se, Nov. 2019, Accessed: Mar. 02, 2024. [Online]. Available: https://gupea.ub.gu.se/handle/2077/62544

[11] Garca Lpez, M. Snchez-Artigas, G. Pars, D. Barcelona Pons, . Ruiz Ollobarren and D. Arroyo Pinto, "Comparison of FaaS Orchestration Systems," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 148-153.

[12] Al-Ali et al., "Making Serverless Computing More Serverless," 2018 IEEE 11thInternational Conference on Cloud Computing, San Francisco, CA, 2018, pp. 456-459.

TABLE I
STRENGTHS AND LIMITATIONS OF SERVERLESS VERSUS TRADITIONAL SERVER-ORIENTED SOLUTIONS

SNo	**Serverless**	**Values**
Function vs application	Easier to deploy changes without affecting the whole application	No time limit on execution of code
Scalability	Scales automatically	Scaling takes time and requires adding more servers
Cost	Pay for what you use	Usually pay for more than the actual usage to have additional resources.
Managed Services	Already developed services can quickly be added.	Not available
Developer velocity	No network configuration needed	- Managing network configurations requires time and knowledge.
Cold start	Running a function can require additional time	- Servers always running, cold start is not an issue
Security and permissions	-Need to rely on the cloud providers solutions.	-Security is more up to the developer's knowledge
Debugging	May be difficult to reproduce errors	Could be less complex