Digital Migration to Serverless Microservices Architecture for a Restaurant Chain using AWS Services

Rishabh Sinha, x21171203, Cloud Platform Programming, MSc. Cloud Computing,
National College of Ireland, Dublin, Ireland, x21171203@student.ncirl.ie,

***Abstract*—The Digital migration and transformation of Eddie Rocket's restaurant chain system to an AWS cloud-based micro-service architecture by utilizing appropriate services provided by Amazon Web Services (AWS) is the focus of this report's case study. The case study presents a micro-services architecture using AWS fully managed serverless services and explains workflow by fulfilling the functional and non-functional requirements of the system in the proposed AWS microservice architecture. Five micro-services modules in the system, including Menu, Cart, Orders, Review Analysis, and Mobile Order using text-to-order functionality which is an extension of the 'call and collect' functionality of Eddie Rocket's using AWS Lex service, and, customer support using AWS Connect are included in the suggested AWS microservice architecture. The study also explains the choice of services utilized in the design by highlighting the benefits and drawbacks of those cloud-based technologies and services. The findings and interpretations of the study are presented in the report's conclusion.**

***Index Terms*—Cloud Platform Programming, AWS Lambda, Microservice Architecture, AWS AWS Pinpoint, AWS Lex, AWS DynamoDB, AWS Step Functions, AWS Event Bridge, AWS Connect, AWS SES, AWS Elastic Beanstalk.**

I. Introduction

Businesses nowadays are required to undergo digital transformation, and the restaurant sector is no different. Many restaurant chains are implementing cloud-based solutions to move their old applications to the cloud in an effort to increase productivity, scalability, and customer experience. The case of "Eddie Rocket's," a well-known restaurant chain with numerous locations in Ireland [1], and its digital migration to Amazon Web Services (AWS) are the main topics of this paper. The goal of this project is to design a micro-service architecture for the restaurant chain domain by decoupling the various modules of the "Eddie Rocket's" web application and deploying and operating them separately on AWS. By doing this, we want to lower maintenance and operating expenses while also enhancing the application's performance, availability, and resilience. This research outlines the benefits and drawbacks of this strategy, as well as how the micro-services architecture was designed and put into use using a variety of AWS services. [20] The restaurant chain industry is a competitive, fast-paced sector that demands flexibility and adaptability to shifting market conditions. In this situation, micro-service design can offer benefits like modularity, scalability, fault tolerance, and quick feature rollout. The handling of enormous volumes of orders, managing inventory and logistics, and integrating with third-party services are just a few of the special needs and issues that the restaurant industry has, which are discussed in this paper in relation to the micro-services approach. This research highlights the potential advantages and difficulties of this strategy by presenting a case study of digital migration and micro-services architecture in the restaurant industry.

II. Company Overview and System Requirements

A well-known restaurant franchise called "Eddie Rocket's" has sites all throughout Ireland and the UK [1]. The restaurant chain specializes in providing top-notch milkshakes, fries, and hamburgers in a vintage diner setting. To manage its menu, orders, and customer interactions, the restaurant business uses a sophisticated online application. The system is in charge of processing customer orders, keeping track of inventory, collecting payments, and producing reports for restaurant management.

The 'Eddie Rocket's' system must be able to do the following *functionalities*: manage menus, process orders, manage carts, analyze sentiment, and text-to-order. The restaurant employees can alter the menu items, their prices, and descriptions with menu management functionality. The system may handle a customer's order using the order processing capability, including billing the client and delivering the order to the kitchen staff for processing. Before completing the transaction, the consumer can manage the items on their order list using the cart management functionality. Customer evaluations and opinions are analyzed using the sentiment analysis functionality in order to improve the quality of service. In order to expand the present "Call and collect" capability, the text-to-order feature employing Lex bot is built, enabling customers to place orders using text messaging which we will discuss further in the architecture explanation.

The *non-functional* requirements of the system include reliability, scalability, security, and performance. In order to guarantee that consumer orders are processed accurately and promptly, the system must be dependable. Additionally, the

system needs to be scaleable to manage peak loads during busy times. To safeguard private client data and guarantee the security of payment transactions, security is essential. The system must also be effective, with quick responses and little downtime.

Overall, the 'Eddie Rocket's' system is a complicated application that needs careful planning and design to make sure it satisfies the requirements of the personnel and patrons of the restaurant chain. The system can be divided into smaller, easier-to-manage components with higher scalability, reliability, and performance by using a microservice architecture using AWS services.

III. Micro-service Architecture and Explanation

The micro-services architecture for Eddie Rocket's restaurant chain entails the autonomous deployment and use of each application component and the recurrent overhead for each new micro-service, the difficulty in optimizing server density and consumption, the necessity to manage many versions of numerous micro-services, and the requirement for client-side code to connect with multiple services are some of the difficulties that this type of architecture might face. But many of these problems can be reduced or completely avoided by leveraging AWS's fully managed serverless services to build the application functionalities on AWS micro-services. The Server-less Micro-services Pattern is particularly helpful for lowering the barrier to adding new application architecture modules and micro-services in the future. [2]

Let's take a closer look at the proposed AWS Micro-service architecture step by step in order to comprehend this architecture better.

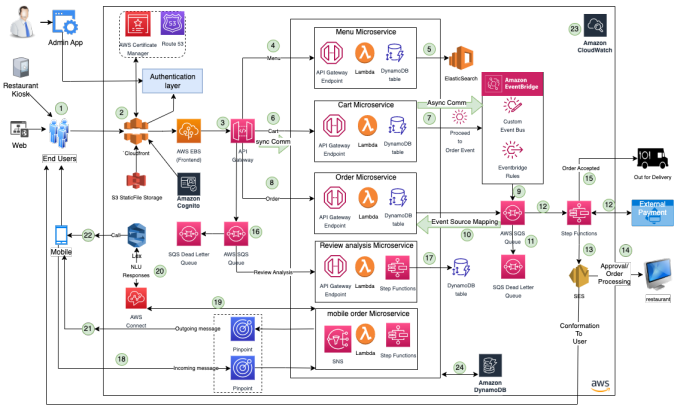

Fig. 1. Proposed Architecture for Eddie Rocket's restaurant chain system

The **first step** involves the End User accessing the Restaurant Application through various devices such as a desktop computer, mobile phone, or kiosk located at the restaurant. By using a web browser, the End User can view menu items, place orders, make payments, and provide feedback on menu items. **Step2** Users are sent to a quick and secure custom domain when they log into the admin or restaurant web application hosted on EBS. Amazon CloudFront CDN, Amazon Route 53, and AWS Certificate Manager are in charge of managing this custom domain. Both Amazon CloudFront and Amazon Simple Storage Service make the static content layer, which keeps static assets on Amazon S3, accessible. [3] Our web application may be served to consumers all over restaurant stores with minimal latency and high transfer rates by using CloudFront in front of the S3 storage cache. Additionally, there is an authentication layer that is controlled by Amazon Cognito and provides users with access to both authenticated and unauthenticated content. With the help of this layer, we can makes sure that only authorized users can access secure data and place orders, allowing for greater control over sign-up, sign-in, and access controls. **Step3** As we can easily duplicate existing APIs and use Lambda functions across numerous accounts using API Gateway. This approach will also eliminate the requirement for server utilization optimization. Additionally, to make the integration process simpler, Amazon API Gateway automatically creates client SDKs in a number of well-known programming languages. [4] AWS Lambda functions, additional AWS services, and Amazon API Gateway have all been used in this microservice architecture to create an API. The restaurant can decouple its system to the required extent of detail by using these services. This architecture includes the "Menu," "Cart," "Orders," "Review Analysis," and "Mobile Order" Microservices, among other functions that Eddie's rocket official website has. **step4** We can implement the "Menu Microservices" Module using API Gateway and AWS Lambda to power its REST APIs and CRUD endpoints. API Gateway is used to perform CRUD operations on the DynamoDB table. [5] The restaurant's web application will allow customers to search for meal items without logging in. At this point, Lambda formats the data and provides menu content to the end user from the database. **step5** The product data is updated whenever there is a change in data relating to the product and is saved in the Amazon Elasticsearch Service (Amazon ES). **step6** Customers can add, remove, or edit things in their restaurant cart by using "POST" or "PUT" calls inside the "Cart Microservices" module. Lambda and DynamoDB both enable this functionality, guaranteeing scalability and effective handling of peak load. [6] To carry out CRUD tasks, Lambda communicates with other AWS services, such as DynamoDB. After adding all the desired items to their shopping cart, the buyer can click "place order." **step7** After the "place order" button has been pressed, Amazon EventBridge is activated, allowing event-driven, asynchronous communication to take place between the "Cart Microservices" and "Ordering Microservices" modules **step8**. This enables autonomous scaling and maintenance by separating the cart and ordering capabilities. The Ordering microservices send messages to the AWS SQS queue in **step9**, where the Amazon Event Bridge triggers the "order placed" event. This makes it easier to use queues for asynchronous event processing. [8] The Ordering Lambda microservice then consumes the event in **step10** by querying the SQS queue. Event source mapping is the communication type that is being used here. The polling request is sent by the Ordering Lambda microservice, which then receives the event from the AWS queue. After processing

the event, the Ordering Lambda microservice creates an order record in its DynamoDB table.Users can enter their order data and obtain an Order-Id without having to wait for backend processing to be completed because the order submission workflow has been divided up into several parts. This strategy lessens the strain on the architecture during times of high traffic when the backend process may become congested. The 'Order-Id' that was created is then transmitted to AWS SQS. We have another Amazon SQS as a dead-letter queue (DLQ) in **step11** for messages that cannot be successfully processed as they separate unconsumed messages to figure out fails later, and is helpful for debugging software or messaging systems. [7] Additionally, the Ordering Microservice exposes RESTful APIs for querying Order tables, which starts an asynchronous process controlled by AWS Step Functions. Through an external payment API, the AWS Step Function starts the Payment process in **step12**. [9] It uses a decision-based business workflow that enables orders to be canceled in the event that a payment transaction is unsuccessful. **step13** takes advantage of AWS SES to email users informing them of a payment success or failure. Additionally, after a successful payment, **step14** comprises the AWS Step Function delivering a message to the Restaurant chefs to either accept or reject the user's order. After the restaurant approves the order, it is sent out for delivery in **step15**. The restaurant system analyzes menu items in real-time and recommends them to users based on the analysis in the "Feedback Analysis" component. The technology assesses client reviews based on their demographics, such as age and gender, to find trends across different customer groupings. The web and mobile applications of the restaurant can be used to gather this data, which can then be sent to the backend system for processing. **step16** of the architecture includes the integration of two AWS API Gateway endpoints, one of which is connected to an EBS endpoint for customer feedback and the other to a SQS queue for customer review archiving. Reviews kept in the SQS queue are processed using an AWS Lambda function, with unsuccessful messages being transferred to a "Dead Letter Queue" for review. The 'detect_sentiment' API from AWS is also called by Lambda to assess the full sentiment of the review. In **step17**, the outcome of the sentiment analysis is subsequently entered into a Dynamo Db table for further investigation. [10] The restaurant business has incorporated a "Text to Order" option to give their customers a seamless experience. The "AWS Pinpoint channel," a scalable service for inbound and outbound marketing communications, receives incoming messages from a user's mobile device in **step18**. When these SMS messages arrive, they set off an AWS Lambda function, which in **step19** sends the message to AWS Connect. [11] In **step20**, a cloud-based contact center called AWS Connect interacts with an Amazon Lex chatbot. [12] In **step24**, the Lex bot will save user profiles and order histories in DynamoDB. In **step21**, the Lex bot can respond to the user and send an SMS back to Amazon Pinpoint using an AWS Lambda function. The Lex bot can intelligently interpret the purpose of incoming text messages. The customer is given the option to choose a call if the Lex bot is unable to complete the request. In this case, control is moved to AWS Connect, where support staff can speak with the user in **step22**. For ongoing system monitoring, we have integrated AWS CloudWatch into our architecture (**step23**) [13]. Additionally, AWS provides usable tools like AWS X-Ray and AWS CloudTrail. By implementing the recommendation in the AWS Well-Architected Framework [14], we can regularly evaluate our workloads, spot potential hazards, and record advancements. We can refer to the Serverless Applications Lens guide, which offers an explanation of developing, deploying, and architecting serverless application workloads if there is a need to further decouple application components. [15]

We have *critically analyzed* different AWS services and added them to the architecture design so that they interact with each application component and the AWS services to fulfill the different functional requirements. AWS services like Amazon Elastic Beanstalk (EBS), Amazon CloudFront, Amazon Route 53, AWS Certificate Manager, Amazon Simple Storage Service (S3), Amazon Cognito, API Gateway, AWS Lambda, and Amazon Elasticsearch Service (Amazon ES), AWS Lex, AWS EventBridge, AWS Step Functions, AWS Connect, AWS Pinpoint are all used in the proposed microservice architecture for the Eddie Rocket's restaurant chain. By meeting the functional needs of both restaurant patrons and staff, this architecture seeks to assure the system's scalability, stability, security, and performance. However, like any other architecture, there are benefits and cons to employing AWS services in this case study and some of which are already discussed earlier during the architecture explanation.

AWS services like Amazon Elastic Beanstalk provide advantages including improved resource management, deployment and scalability, and automatic upgrades. However, they also offer disadvantages including limited customizability and control, less flexibility, and possible vendor lock-in. The API Gateway service is similar in that it provides benefits like easier API maintenance, interaction with other AWS services, and scalability, but it also has drawbacks like the possibility for higher costs, a lack of control over API functionality, and vendor lock-in. [16]

Other AWS services, such as Amazon Elasticsearch Service [18], AWS Lex [17], AWS EventBridge, AWS Step Functions, AWS Connect, and AWS Pinpoint, also provide a number of benefits, including streamlined management, integration with other AWS services, and scalability. However, they also have drawbacks, including the potential for higher costs, a lack of control over functionality, and vendor lock-in.

Scalability is one of the key advantages of employing AWS services in this design. When a restaurant chain faces peak loads during busy periods, the ability of the AWS services to scale up or down in response to application demand is crucial. To grow the program in accordance with demand. Utilizing AWS services also offers security. The restaurant business must make sure that customer information is protected and payments are safe. Only individuals with the proper authorization are able to access secure data and make

purchases by using AWS Cognito's authentication layer. AWS Certificate Manager is also used to safely manage custom domains. Authentication and management of users are handled using Amazon Cognito. Simplified user administration, safe authentication, and interaction with other AWS services are some of its benefits. The potential for greater prices, a lack of significant customization, and vendor lock-in are some of its drawbacks.

However, using AWS services has certain disadvantages as well. One possible problem is the price of employing these services, which, especially for small enterprises, may add up quickly. Additionally, the architecture's complexity can be daunting for developers, which could result in a longer development and maintenance cycle.

IV. Conclusions and future work

In this research, we gave a case study of the digital migration of the Eddie Rocket's restaurant chain to an Amazon Web Services (AWS) cloud-based micro-services architecture using the necessary services, particularly in the context of a restaurant chain system. The proposed AWS micro-services architecture is made up of five modules: the menu, the cart, the orders, the review analysis, and the mobile order using text-to-order functionality (an extension of Eddie Rocket's "call and collect" functionality using AWS Lex service), and the customer support using AWS Connect. In order to emphasize the advantages and disadvantages of these cloud-based technologies and services, the study highlighted the services that were selected for use in the design. The influence of AWS micro-services architecture on customer happiness, revenue, and other key performance measures for restaurant chains is one prospective route for future research. Further research may also find value in examining how other AWS services like AI and machine learning may be used to improve customer satisfaction and streamline operations.

The dependability, scalability, security, and performance requirements of the restaurant chain's system were met with the adoption of the micro-services architecture using AWS services. This was accomplished by disengaging the system's numerous modules, which raised output, scalability, and customer satisfaction while cutting maintenance and operating costs. Modularity, scalability, fault tolerance, and rapid feature rollout were also made possible by the micro-services architecture design, which is crucial in the competitive and quick-paced restaurant industry. Some of the particular requirements and problems that the restaurant business encountered were explored in this paper in regard to the micro-services strategy, including the handling of massive amounts of orders, managing inventory and logistics, and integrating with third-party services.

We now understand the benefits of using AWS fully managed serverless services for the development of micro-services, the significance of choosing the appropriate service for each system function, and the usage of APIs to connect various services. This study concludes that adopting a microservices architecture using AWS services can give restaurant chains a competitive edge by allowing them to more effectively manage the massive volume of orders, integrate with third-party services, and respond quickly to changing market conditions. Other firms intending to employ microservices architecture [19] in their digital migration strategy can learn from the advantages and disadvantages of the suggested design of Eddie Rocket's Architecture proposal.

V. References

[1] An analysis of the impact of service quality on customer satisfaction -Eddie Rocket's Available at: https://esource.dbs.ie/bitstream/handle/10788/3673/mba_rajendran_p_2019.pdf?sequence=1

[2] Engdahl, S. (2008c) Let's Architect! Serverless architecture on AWS, Amazon. Available at: https://aws.amazon.com/blogs/architecture/lets-architect-serverless-architecture-on-aws

[3] Engdahl, S. (2008a) Amazon S3 + Amazon CloudFront: A Match Made in the Cloud, Amazon. Available at: https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud

[4] Tutorial: Build an API gateway rest API with cross-account lambda proxy ... Available at: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-cross-account-lambda-integrations.html

[5] Engdahl, S. (2008b) Using Amazon API Gateway as a proxy for DynamoDB, Amazon. Available at: https://aws.amazon.com/blogs/compute/using-amazon-api-gateway-as-a-proxy-for-dynamodb

[6] Musgrave, D. (2022) Lambda function scaling, Amazon. Available at: https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

[7] Amazon SQS DEAD-letter queues - amazon simple queue service. Available at: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-dead-letter-queues.html

[8] Creating Amazon Eventbridge rules that react to events. Available at:https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-create-rule.html

[9] Engdahl, S. (2008b) Building cost-effective AWS Step Functions workflows, Amazon. Available at: https://aws.amazon.com/blogs/compute/building-cost-effective-aws-step-functions-workflows

[10] Engdahl, S. (2008c) Integrate-your-Amazon-DynamoDB-table-for-sentiment-analysis, Amazon. Available at:https://goo-gl.ink/lpMlu

[11] Engdahl, S. (2008d) Using Amazon Pinpoint to send text messages in Amazon Connect, Amazon. Available at: https://aws.amazon.com/blogs/contact-center/using-amazon-pinpoint-to-send-text-messages-in-amazon-connect

[12] Add an Amazon Lex bot to Amazon Connect - Amazon connect. Available at:https://docs.aws.amazon.com/connect/latest/adminguide/amazon-lex.html

[13] Application monitoring - amazon cloudwatch. Available at:https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Application-Monitoring-Sections.html

[14] AWS well-architected framework. Available at: https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

[15] Serverless applications lens - amazon web services, inc. Available at:https://pages.awscloud.com/rs/112-TZM-766/images/MAD-AWS-Serverless-Applications-Lens-7.pdf

[16] Why use api gateway? pros and cons: Knowledge base (2021) Dashbird. Available at:https://dashbird.io/knowledge-base/api-gateway/pros-and-cons-of-using-an-api-gateway

[17] Engdahl, S. (2008a) Moving to managed: The case for the Amazon OpenSearch Service, Amazon. Available at:https://aws.amazon.com/blogs/big-data/moving-to-managed-the-case-for-the-amazon-opensearch-service

[18] Bays, J. (1984) Cambridge walk about: Pictorial Map and Guide, Amazon. Available at:https://aws.amazon.com/about-aws/whats-new/2015/10/introducing-amazon-elasticsearch-service

[19] Restaurants, Catering, and Food Service (2012) Amazon. Available at:https://aws.amazon.com/travel-and-hospitality/restaurants

[20] Whitepapers, Implementing Microservices on AWS, Amazon. Available at:https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/microservices-on-aws.html