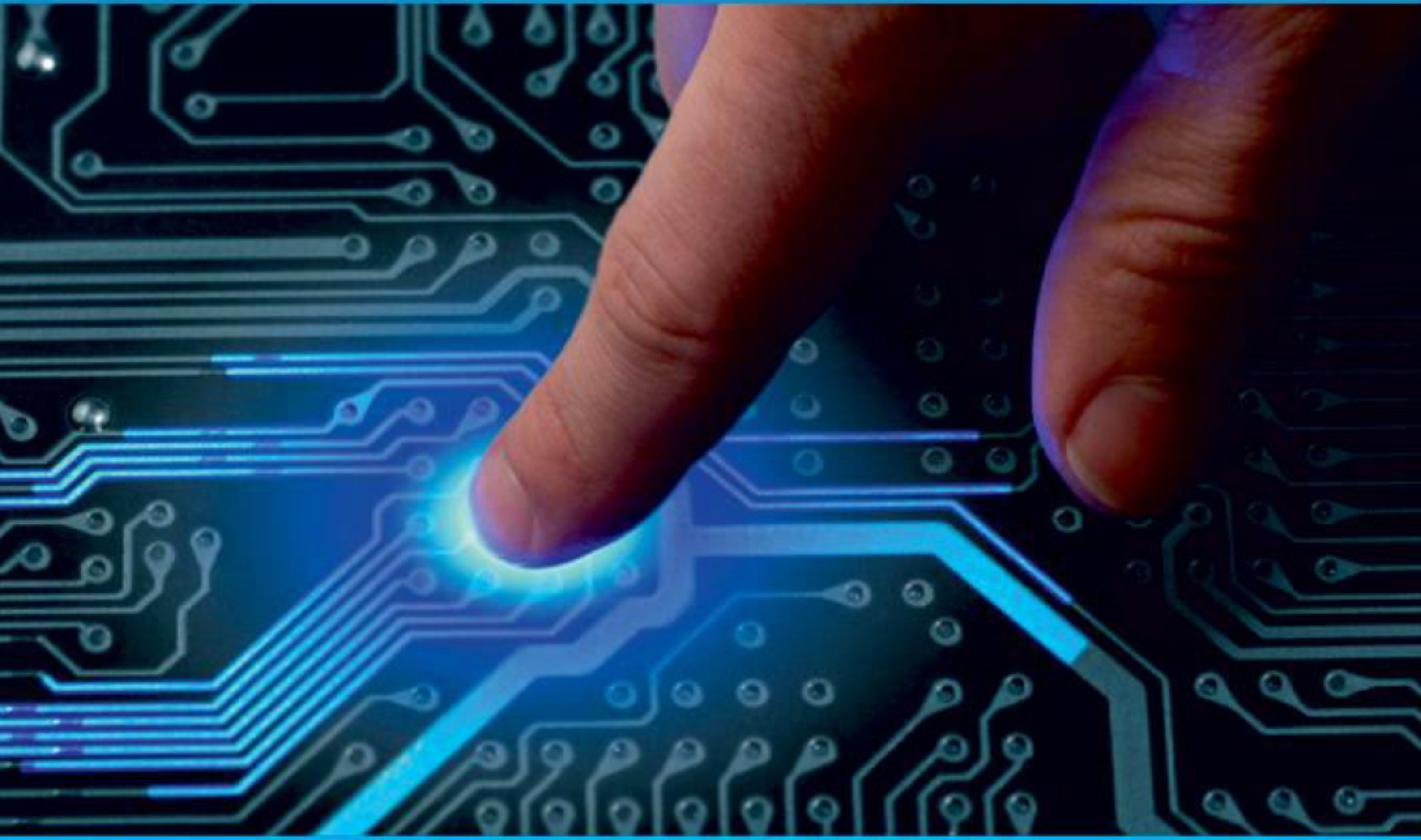
IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 10, Issue 7, July 2022

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.165

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165

|| Volume 10, Issue 7, July 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1007098 |

Microservices Architecture Based E-CommerceService

Mohammed Hassan Shahid, Sabarinathan C

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India

ABSTRACT : In the modern era, when convenience is key to a comfortable lifestyle, people are finding ways of buying and selling products online. This has brought a revolution in the e-commerce domain. Unlike before, the e-commerce domain is not just relatable to wholesalers but to a massive amount of consumers. One of the problems and possibilities in the industry of acquiring, selling, and marketing goods is the rapid rise of e-commerce. E commerce necessitates a variety of tactics to boost income and consumer happiness. In order to improve client happiness, e-commerce businesses must share a variety of services. The issue arises because e-commerce requires reusability services, automated service deployments, rapid service scaling, and a secure process. This makes it immensely difficult for small-scale businesses to handle the huge amount of requests and improve scalability, and maintain the modules separately. To accommodate current e- commerce needs, the ideal approach is to leverage microservices architecture.

Microservices are a novel computing paradigm that can be used in conjunction with cloud computing. Traditional monolithic programs are broken down into a set of individual services that can be built, tested, and deployed independently using microservices. Microservices characteristics include independent d Development, deployment, synchronous execution, high availability, and loosecoupling are all separate. This project aims to build a microservice-based e-commerce platform using the Saga Orchestration pattern and other development practices to ensure that the development and deployment of the entire service is suitable for small- scale e-commerce companies to maintain and deploy.

KEYWORDS: Microservices, Cloud computing, AWS.

I. INTRODUCTION

Cloud computing enables businesses to deploy enterprise applications with the ability to grow computer resources on demand. Businesses can host their own applications using Infrastructure as a Service or Platform as a Service solutions, or they can buy ready-to-use software using the Software as a Service (SaaS) model. When businesses deploy their own apps on PaaS or IaaS platforms, they face challenges in order to make use of cloud platform features like auto scaling, continuous integration, continuous deployment, and variable monitoring, among others. Most firms that have attempted to migrate their monolithic applications to an IaaS or PaaS provider and have done so with a monolithic payload. A monolithic application, in this sense, is a single codebase that provides various services via numerous interfaces such as web applications, and REST services.

In a monolithic architecture, since all the functionality is contained in a project repository, its modules cannot be run individually. This approach is tightly integrated, and a single process contains all of your logic for handling a request. Using the essential characteristics of your programming language, you can divide the programme into classes, functions, and namespaces.

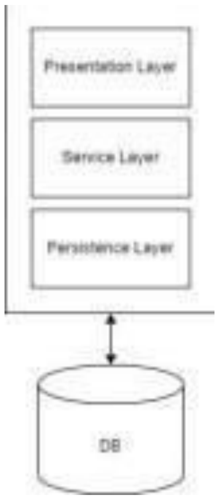


Fig. 1. Monolithic Architecture

Because monolithic programmes generally provide a range of services, some of which are more popular than others, they are difficult to scale. If popular services must be scaled owing to high demand, the entire set of services must be scaled simultaneously, leading non-popular services to consume large server resources even when they are not in use.

According to the development strategy, most cloud based software companies need to be able to adapt as soon as possible since their competitors may threaten their organization if they do not improve their product experiences and add new features. Monolithic apps use a single codebase that is used by several developers to generate all services. When these programmers create or upgrade services, they must make certain that all other services continue to function. The complexity of services grows as more are introduced, reducing organisations' ability to innovate on latest models and features. Additionally, when new application versions are deployed, the entire set of services is restarted, resulting in unsatisfactory user experiences. A single point of failure characterises a monolithic deployment; if the application fails, the entire collection of services collapses as well.

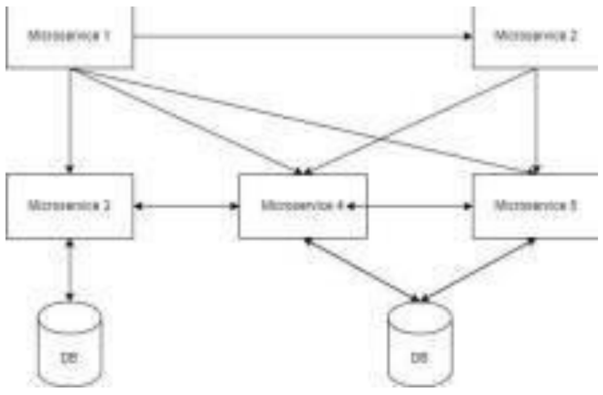


Fig. 2. Microservices Architecture

Microservices architecture presents a way for efficiently scaling computing resources as well as resolving many other challenges that plague monolithic designs. Microservices architectures, on the other hand, present additional issues, such as the work necessary to install each microservice as well as scaling and running them on cloud providers. To solve these problems, platforms such as AWS Lambda enable the delivery of microservice architectures without the requirement for server management. As a result, it makes it easier to create functionalities (i.e. microservices) that can be simply deployed and automatically scaled.

II. RELATED WORKS

1. One of the issues with deploying microservices in the cloud is the work necessary to deploy each microservice. Firms deploying microservices may utilise a variety of DevOps automation technologies, including Ansible, Docker, Puppet, and auto scaling, among others, but putting such tools in place takes time and resources. To solve this issue, cloud providers like as AWS have recently released services such as Lambda, which allows microservices to be deployed without the requirement for server management. This service is meant to be costed per request, so programmers must only worry about coding individual services then launching them on Lambda, which is ideal for small development teams and start-ups.

2. Microservices architectures are cloud-based systems that differ from monolithic system architectures in several ways. Microservices for architecture include a communication mechanism that allows them to collaborate with other services. Because of the various frameworks, platforms, and databases, this service will be simple to develop using a programming language. Microservices are frequently combined with DevOps approaches for continuous integration and automated system development. Application monitoring also allows architecture microservices to receive input in order to improve their performance. Improvements to the system result in a shorter development cycle.

3. The microservice design recommends dividing an application into a collection of granular services S (S1, S2, Sn), each of which provides a subset of the application's services S (S1, S2, Sx). Each microservice is built and tested separately by a development team µT, using the technical stack most suited to the microservice's functions. Each microservice is built on its own codebase, and the µT team is responsible for deploying, growing, managing, and upgrading the microservices on a cloud computing IaaS or PaaS solutions. The services are provided in the presentation layer utilising the REST software architectural style due to its un-complicated and widespread adoption by large Internet enterprises.

4. A study on migration practises toward microservices adoption in sector in [15]. They created and executed a survey intended at practitioners who were in process of migrating their applications, gathering information on the activities takenand the problems faced throughout the move.

5. Because of the uprising of serverless computing, microservices architectures are no longer the only choice for modernising a monolithic software. There are some differences of opinion about whether to use serverless or microservices. The properties of microservices and serverless architecture decisions were expounded upon by Jambunathan et al. From the perspective of service deployment, serverless has infrastructure limitations that necessitate cloud service support and hosting by cloud service platforms. A microservices architecture, on the other hand, can be implemented in either a privateor public data centre. Serverless, on the other hand, has a deployment benefit over microservices due to auto-scaling without complex server settings.

III. SYSTEM DESIGN

This section will talk about the proposed methodology which is implemented as an e-commerce application named BuyersClub. The implementation of the application can be categorised into the various modules as follows:

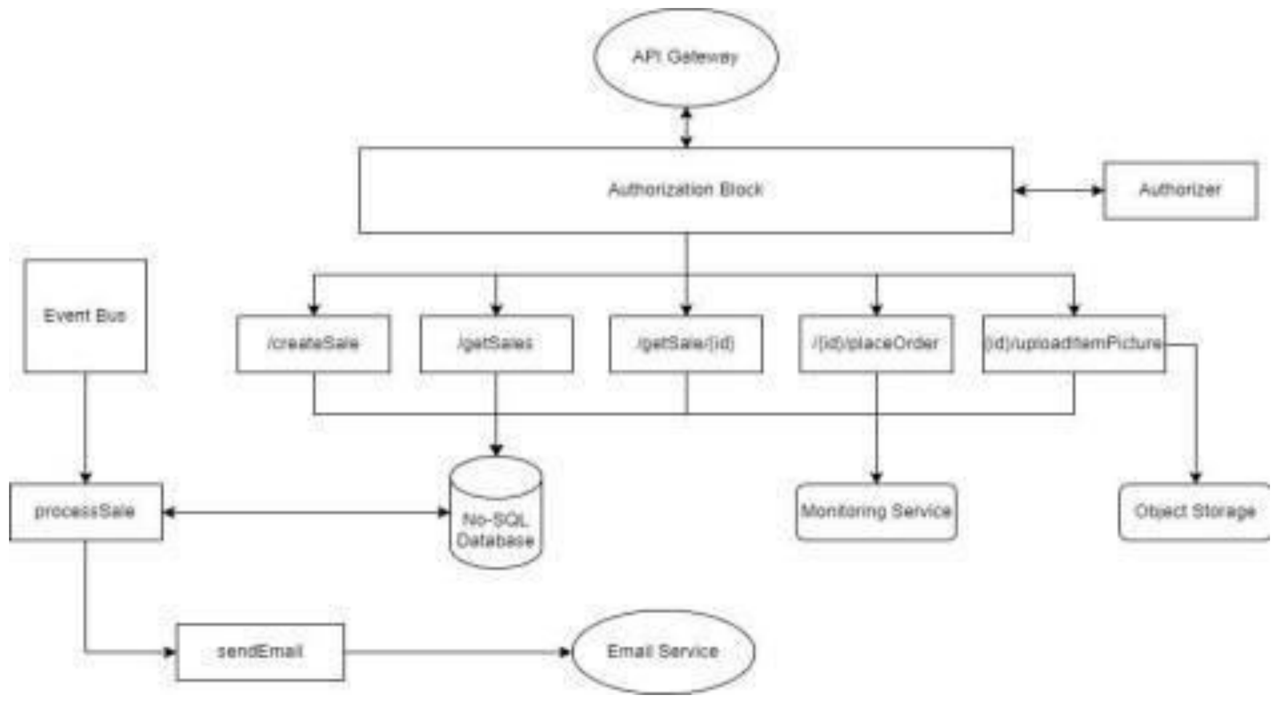


Fig. 3. Architecture diagram of BuyersClub service A. Application Overview

The application used as part of the research is an e commerce system where users can put up their items for sale and other users of the application can place orders on them. The application consists of a front -end module developed in Flutter capableof running on multiple platforms, while the back-end module of the project is developed using Node.js and contains the following sub-modules:

- Authentication

  JSON Web Token is a standard for securely transmitting information within a JSON object between users in a simple and self contained manner. This information can be reviewed and trusted since it is digitally signed. RSA or ECDSA canbe used to sign JWTs with a secret or a public/private key pair.

  For this application, this project is going to use JWT for authentication purposes. This is one of the most common scenarios for using JWT. To implement JWT authorization, the use of Auth0 has been implemented. Soon as the user has signed in, the JWT will be included in each subsequent request, allowing the user to access route keys, services, and resources which the token permits. The authentication module checks the client's JWT in the Authorization header of the API request for validity.

- Sale Items

  It consists of items that are marked open for sale. When a user visits this module, it triggers a GET request to the API to fetch a list of all the items which have the status 'OPEN' for sale in the database which can be used to display in the front-end module. The user can also invoke another GET request with the item's ID to fetch all the details of the respective item.

Users can invoke a POST request to create items for sale that they intend to sell on the platform by entering the title, price, description, and image of the item in the body of the request. Similarly, users can invoke a PATCH request to place orders on items that they desire using the item's ID which changes then changes the status of the item to 'CLOSED' which will prevent other users to place orders on the respective item and pass the item's details to the notification module.

The structure and choreography of the microservices architecture in this application is represented in *Fig. 3* mentioned above.

- Notification Module

This module is used in the application for notifying buyers and sellers regarding the sale/order of their items. On the successful sale of the item, both the buyer and the seller will be notified. In case, an item has passed its expiration time, the seller will be notified of the item not being able to sell on the platform. The notification module has been implemented using AWS SES. AWS Simple Email Service is a low-cost, and manageable email service that allows developers to transfer emails from a source application.

- Miscellaneous

All the above modules require user data to work on. In order to store details of each item, an AWS DynamoDB table has been set up for the project. DynamoDB is a No-SQL document type database that stores data in objects with key-pair values. To store images of the items, an object storage system is also required for the project. In this case, S3 (Simple Storage Service) has been used to store the images of the items.

  - Application Front-End

A front-end application has been developed using Flutter and Dart. Flutter is a cross platform front-end development SDK based on Dart. Flutter can be used to create applications for Android, iOS, Web, Windows, macOS, and Linux. The application serves as an interface to allow users to buy or sell items on the platform. The applicationis also integrated with the Auth0 library to allow users to log in. On successfully signing in, Auth0 generates the JSON Web Token which can be used to trigger the API endpoints embedded in the application.

B. Microservices Based Deployment Strategy

The back end of the project is deployed using Lambda functions and API Gateway, as shown in *Fig. 3*. The AWS API Gateway acts an access point for the API that routes requests to the appropriate Lambda function. After interpreting the request and retrieving the data from the database, the Lambda function executes the business logic. AWS Lambda has a scaling strategy that can handle up to 1000 concurrent requests for on-demand scaling, depending on the workload. We have 8 lambda functions with distinct API calls in the present setup. Moreover, Lambda is a container based serverless computing resource which allows programs to be run in a contained isolated environment.

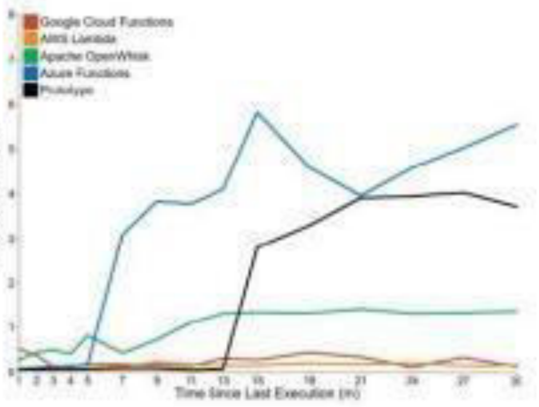


Fig. 4. Execution time comparison between serverless functions

While there are other alternatives to AWS Lambda functions, such as, Google Cloud functions, Azure Functions, and Apache OpenWhisk. In a study, it has been found that AWS Lambda provides the least execution latency of all. While considering the network latency which is going to be generated while delivering users, going for AWS Lambda functions for a serverless computing resource to organise microservices seem like the best choice to deliver content efficiently.

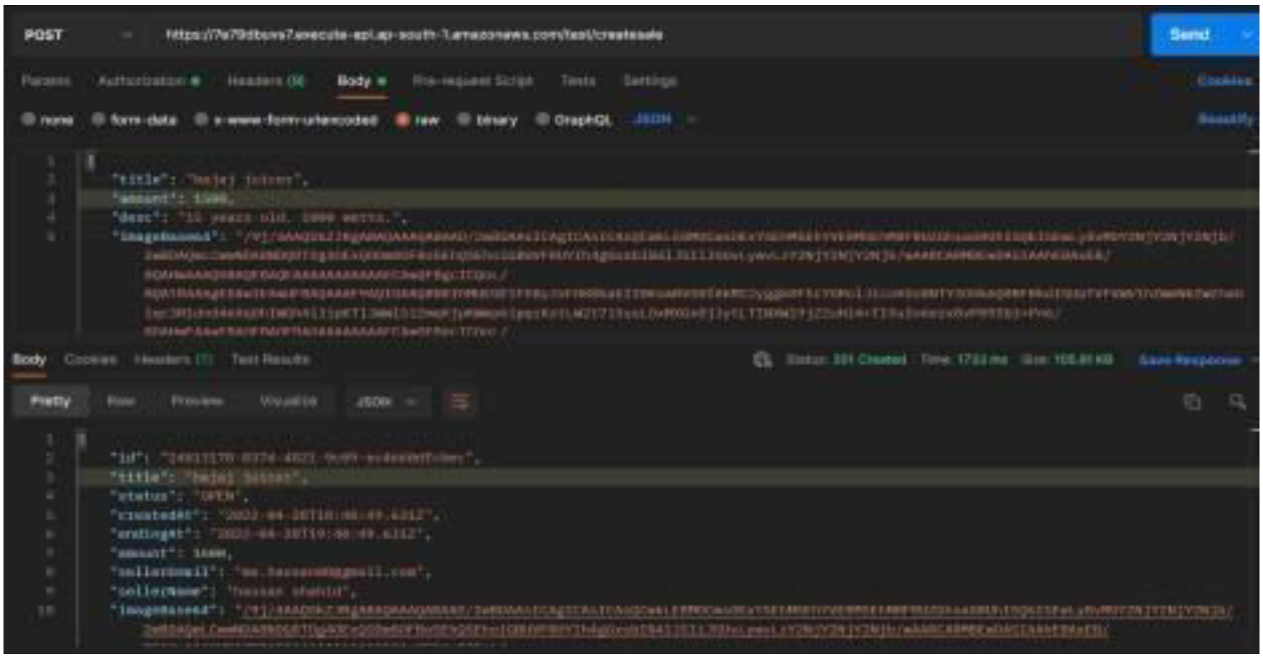
The microservices along with the other cloud services are deployed using Terraform. Terraform is an Infrastructure as Code (IaC) tool that allows provisioning and configuring infrastructure on the cloud using code. With HCL (Hashicorp Configuration Language), Terraform allows writing concise descriptions of resources using blocks, arguments, and expressions. This method allows error detection in resource configuration before deployment.

IV. RESULT

The objective of this project to deploy an e-commerce application using microservices architecture as access the performance and latency of the execution of the microservices. The application after deployment allows users to put their items for sale and allow other users to place orders on them using the most suitable architectural pattern for the business logic and deployment techniques.

A. API Responses

Following are the various outputs of performing CRUD operations using the developed microservices on Postman. Postman is an API testing platform for developers. The operations on these APIs were performed by entering the required body parameters and authorization headers.



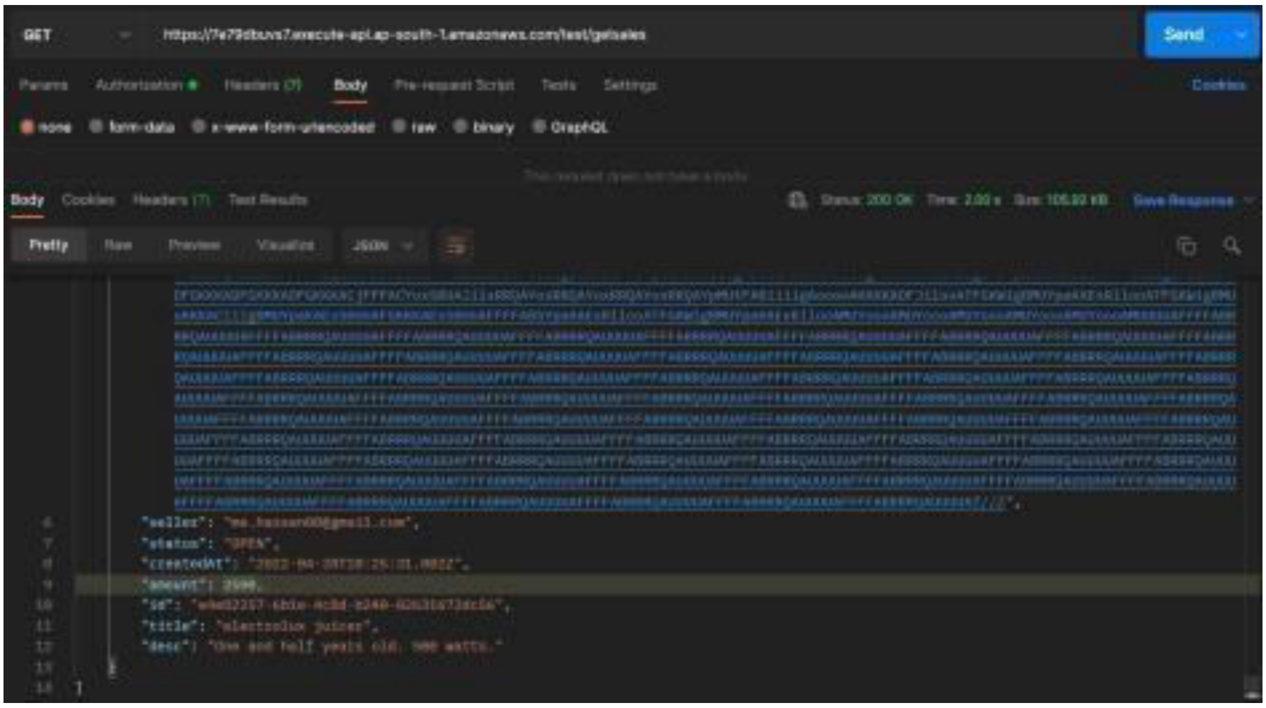
```
POST    https://7e79dbuvs7.execute-api.ap-south-1.amazonaws.com/test/createsale

{
    "title": [illegible],
    "amount": [illegible],
    "desc": [illegible],
    "imageBase64": [illegible]
}
```

```
Status: 201 Created   Time: 1733 ms   Size: 105.81 KB

{
    "id": [illegible],
    "title": [illegible],
    "status": "OPEN",
    "createdAt": [illegible],
    "endingAt": [illegible],
    "amount": [illegible],
    "sellerEmail": [illegible],
    "sellerName": [illegible],
    "imageBase64": [illegible]
```

Image 1. POST /createsale API

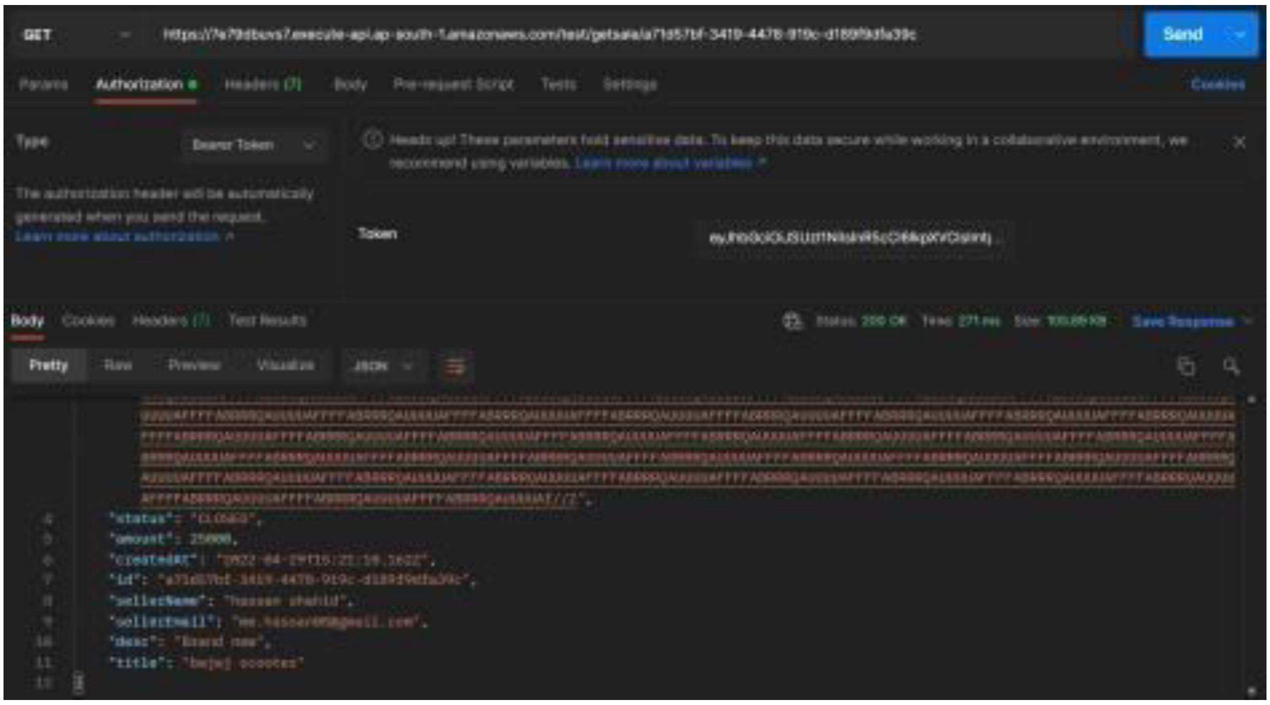


```
GET  https://7e79dbuvs7.execute-api.ap-south-1.amazonaws.com/test/getsales
Status: 200 OK   Time: 2.00 s   Size: 105.92 KB
```

[illegible]

```
"seller": "ms.hassan00@gmail.com",
"status": "OPEN",
"createdAt": "2022-04-28T18:25:31.082Z",
"amount": 2500,
"id": "[illegible]",
"title": "electrolux juicer",
"desc": "One and half years old. 900 watts."
}
]
```

Image 2. GET /getsales API



```
GET  https://7e79dbuvs7.execute-api.ap-south-1.amazonaws.com/test/getsale/a71d57bf-3419-4478-919c-d189f9dfa39c
Type: Bearer Token
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables
The authorization header will be automatically generated when you send the request. Learn more about authorization
Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtj...
Status: 200 OK   Time: 271 ms   Size: 105.89 KB
```

[illegible]

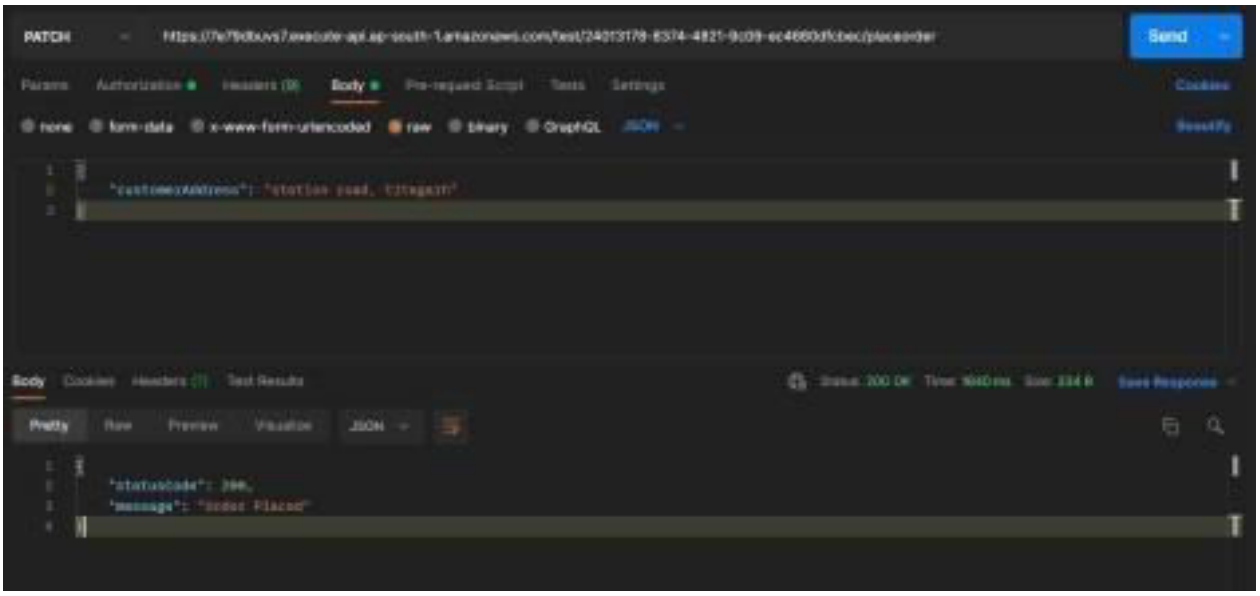
```
"status": "CLOSED",
"amount": 25000,
"createdAt": "2022-04-29T15:21:10.162Z",
"id": "a71d57bf-3419-4478-919c-d189f9dfa39c",
"sellerName": "hassan shahid",
"sellerEmail": "ms.hassan00@gmail.com",
"desc": "Brand new",
"title": "bajaj scooter"
}
```

Image 3. GET /getsale/{id} API

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.165

|| Volume 10, Issue 7, July 2022 ||

| DOI: 10.15680/IJIRCCE.2022.1007098 |



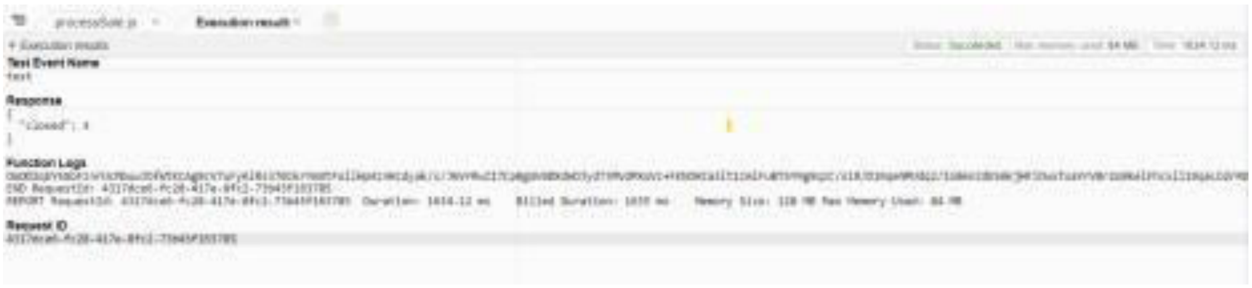
```
PATCH   https://7e79dbuvs7.execute-api.ap-south-1.amazonaws.com/test/24013178-8374-4821-9c09-ec4660dfcbec/placeorder
```

```
{
    "customerAddress": "station road, Citagarh"
}
```

Status: 200 OK Time: 1660 ms Size: 334 B

```
{
    "statuscode": 200,
    "message": "Order Placed"
}
```

Image 4. PATCH /{id}/placeorder API



Status: Succeeded Max memory used: 84 MB Time: 1614.12 ms

Test Event Name
test

Response

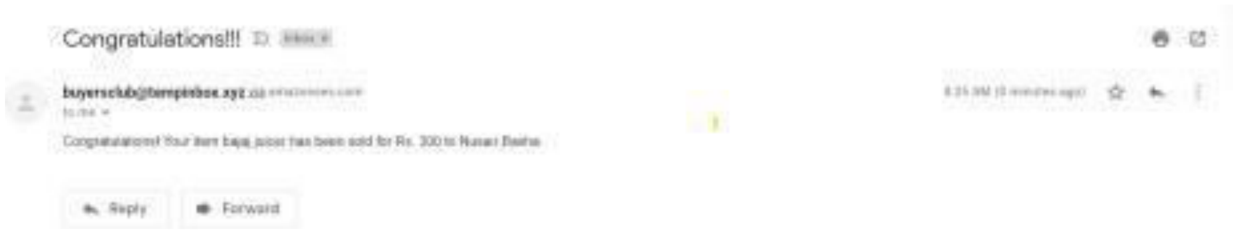
```
{
  "closed": 4
}
```

Function Logs

```
[illegible]
END RequestId: 4317dca6-fc28-417e-8fc1-73945f1037B5
REPORT RequestId: 4317dca6-fc28-417e-8fc1-73945f1037B5  Duration: 1614.12 ms    Billed Duration: 1615 ms    Memory Size: 128 MB Max Memory Used: 84 MB
```

Request ID
4317dca6-fc28-417e-8fc1-73945f1037B5

Image 5. processSale API execution response

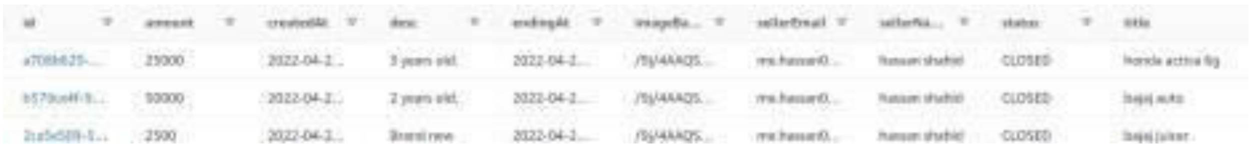


Congratulations!!! Inbox

buyersclub@tempinbox.xyz via amazonses.com
to me
8:25 AM (0 minutes ago)

Congratulations! Your item bajaj juicer has been sold for Rs. 300 to Nusari Basha

Image 6. Email received on the successful order placement



id	amount	createdAt	desc	endingAt	imageBa...	sellerEmail	sellerNa...	status	title
a708b625-...	25000	2022-04-2...	3 years old.	2022-04-2...	/9j/4AAQS...	ms.hassan0...	hassan shahid	CLOSED	honda activa 6g
b573ce4f-3...	50000	2022-04-2...	2 years old.	2022-04-2...	/9j/4AAQS...	ms.hassan0...	hassan shahid	CLOSED	bajaj auto
2ca5e589-3...	2500	2022-04-2...	Brand new	2022-04-2...	/9j/4AAQS...	ms.hassan0...	hassan shahid	CLOSED	bajaj juicer

Image 7. Database Table Items

B. Performance Monitoring

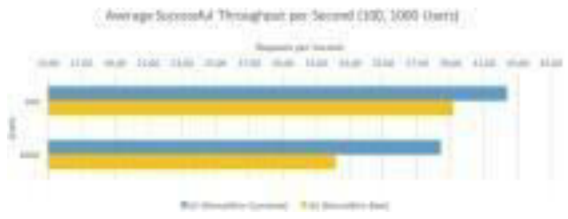


Fig. 7. Average Successful throughputs per second

Fig. 5. Performance metrics of processSale lambda function

The following performance data for the microservices have been recorded using AWS CloudWatch and Postman. The invocation shown in Fig. 7. is performed by the Event Bus which triggers the processSale function at regular intervals to close items for sale after they have surpassed their expiration deadline. The duration of the invocation can vary depending on the number of items on which the function is performing the operation.

Sl. No	Microservice Function	Time consumed (ms)	MemoryUsed (kb)
1	createSale	1733	105.91
2	getSales	2000	105.92
3	getSale	1605	105.89
4	placeOrder	1840	0.334
5	processSale	1635	324.80

Fig 6. Time and Memory consumption of microservices functions

average successful throughputs per second between monolithic and microservices architecture, microservices architecture performed significantly better than monolithic in both cases where the number of users in one case was 100 and 1000 in another.

V. CONCLUSION

In this section, the results from the proof of concepts of microservices architecture for e-commerce are already presented. According to the research findings from microservices architecture, loading time is longer for APIs accessible to the publi c via API endpoints. On the other hand, for choreographed APIs running internally in the application, the time consumed is significantly less than the prior. The loading time is determined by the web service's performance and network connectivity. Furthermore, due to the large number of development teams working on monolithic systems, any modifications require extensive coordination.

We also find that deploying microservices on Infrastructure as a Service (IaaS) platforms such as AWS leads in lower infrastructure expenses owing to their pay-as-you-go pricing model. However, as compared to container-based services such as AWS ECS, the adoption of new cloud services such as Lambda, which is solely intended to deploy microservices at a more detailed level, can dramatically lower infrastructure expenses.

Microservices transform large scale application into a collection of small sub-application that can be separately deployed and managed allowing codebases to be managed efficiently by employing a more realistic method in which incremental enhancements are carried out by small development teams on distinct code bases. Cost deduction, scalability, and agility should be balanced against the development efforts, technical issues, and costs paid by businesses as a result of microservices that necessitate the adoption of new techniques, procedures, and methods.

REFERENCES

[1] R. Chen, S. Li, and Z. Li, “From Monolith to Microservices: A Dataflow-Driven Approach,” in 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017, pp. 466–475.
[2] M. Richards, “Microservices vs. Service-Oriented Architecture,” p. 57
[3] S. Newman, Building microservices: designing fine-grained systems, First Edition. Beijing Sebastopol, CA: O’Reilly Media, 2015.
[4] R. Buyya, “Cloud computing: The next revolution in information technology,” in Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on, pp. 2–3, Oct 2010.
[5] Microservices, “Pattern: Microservices architecture,” 2014.
[6] https://registry.terraform.io/providers/hashicorp/aws/l atest/docs
[7] https://docs.aws.amazon.com/
[8] M. Villamizar et al., “Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures,” Serv. Oriented Comput. Appl., vol. 11, no. 2, pp. 233– 247, Jun. 2017.
[9] M.C. MacKenzie et al., Reference Model for Service Oriented Architecture 1.0, OASIS, 12 Oct. 2006.
[10] Schneider, T. (2016). Achieving cloud scalability with microservices and devops in the connected car domain. In Software Engineering.
[11] Schneider, T. (2016). Achieving cloud scalability with microservices and devops in the connected car domain. In Software Engineering.
[12] Baldini, I., Castro, P. C., Chang, K. S., Cheng, P., Fink, S. J., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R. M., Slominski, A., and Suter, P. (2017). Serverless computing: Current trends and open problems. CoRR, abs/1706.03178.
[13] S.Kannadhasan and R.Nagarajan, Development of an H-Shaped Antenna with FR4 for 1-10GHz Wireless Communications, Textile Research Journal, DOI: 10.1177/00405175211003167, March 21, 2021, Volume 91, Issue 15-16, August 2021
[14] K. Mikhail et al., “Domain Objects and Microservices for Systems Development: A Roadmap,” Proc. 5th Int’l Conf. Software Eng. for Defence Applications, Springer, 2016, pp. 97–107.
[15] I. W. Len Bass and L. Zhu, DevOps: A Software Architect’s Perspective. Addison-Wesley Professional, 2015.
[16] P. Di Francesco, P. Lago, and I. Malavolta, “Migrating towards microservice architectures: An industrial survey,” in 2018 IEEE International Conference on Software Architecture (ICSA), pp. 29– 2909, April 2018.