An Architectural Approach to Creating a Cloud Application for Developing Microservices

A. N. M. Sajedul Alam[1], Junaid Bin Kibria[1], Al Hasib Mahamud[1], Arnob Kumar Dey[1], Hasan Muhammed Zahidul Amin[1], Md Sabbir Hossain[1], Annajiat Alim Rasel[1]

[1] Department of Computer Science and Engineering, School of Data and Sciences (SDS), BRAC University, 66 Mohakhali, Dhaka 1212, Bangladesh
{a.n.m.sajedul.alam, junaid.bin.kibria, al.hasib.mahamud, arnob.kumar.dey, muhammed.zahidul.amin, md.sabbir.hossain1}@g.bracu.ac.bd, annajiat@bracu.ac.bd

Abstract. The cloud is a new paradigm that is paving the way for new approaches and standards. The architectural styles are evolving in response to the cloud's requirements. In recent years, microservices have emerged as the preferred architectural style for scalable, rapidly evolving cloud applications. The adoption of microservices to the detriment of monolithic structures, which are increasingly being phased out, is one of the most significant developments in business architecture. Cloud-native architectures make microservices system deployment more productive, adaptable, and cost-effective. Regardless, many firms have begun to transition from one type of architecture to another, though this is still in its early stages. The primary purpose of this article is to gain a better understanding of how to design microservices through developing cloud apps, as well as current microservices trends, the reason for microservices research, emerging standards, and prospective research gaps. Researchers and practitioners in software engineering can use the data to stay current on SOA and cloud computing developments.

Keywords: Microservices, Architecture, Cloud Application

1 Introduction

As monolithic architecture is not considered anymore to use, microservice architecture is considered one of the trending topics in the software industry. It is very necessary to have a clear idea about the design pattern of microservice architecture. Microservice designs have recently been introduced to divide up architectures into independent services which are deployable and these services can be quickly deployed to any resource as needed [1].

Microservice Architecture (MSA) is considered a subset of Service Oriented Architecture (SOA) which focuses on certain elements. SOA and MSA have a lot of

differences. MSA's service architecture is guided by a share nothing mentality in terms of facilitating agile methodologies. It encourages isolation and autonomy [2]. Rather than that, SOA promotes a high level of reuse by adopting a share as much as you can philosophy.

Monolithic techniques were formerly used to construct apps. This signifies that a complete program was executed from a single code base [3]. It is well-known that designing a monolithic program is easier than a distributed one. Contrarily, putting them into action comes with a number of difficulties. A lack of scalability and adaptability are two of the most important drawbacks of monolithic apps because they are built as a single unit, monolithic apps are very complicated to change. Such applications have a high degree of connection, which makes them problematic because of their enormous size, scaling monolithic programs is a significant challenge. When just a portion of a monolithic application has to be scaled, it must be scaled as a whole. Such challenges, along with the emergence of innovative technologies, have prompted the software industry to look for alternatives. It is possible to overcome many of these concerns by using microservices. Microservices are mainly a collection of interconnected services that compose an application. Every service is designed to perform a certain function. The deployment, as well as replacement processes, are made easier for their reduced sizes. As a result, organizations are able to do activities that would have been challenging or impossible to complete with traditional monolithic software.

Microservices may be deployed separately, generally with the help of a deployment and orchestration framework, such as in the cloud, allowing them to deploy frequently and on their own timetables [4]. The deployment and orchestration of microservices across the cloud's dispersed and layered nature are important architecture issues. For their variable deployment timetables and provisioning orchestration demands, clouds provide a management platform.

In this paper, a microservice was built using different technologies. For some specific services which were required for another distributed project. The main goal of this work is that a hybrid microservice will be built for solving our project issues, by which we can use it for saving time while working with cloud services.

2 Literature Review

Dmitry Namiot et al. [5] proposed the overview of the microservice architecture including the pros and cons of the approach and provided the deployment pattern. Authors have also shown partitioning the system into microservices by use case and for this purpose, they have utilized the M2M ETSI model. Nabor C.Mendonc et al. [6] has discussed the challenges of existing and emerging microservice technologies using an example of a cloud-based intelligent video surveillance device. When designing the microservice architecture, the interaction between self-adaptive systems and microservices was shown. Lianping Chen et al. [7] has proposed an approach to tackle the challenges that may arise in a microservice architecture. Indicating the areas which are needed to study more to build microservice architecture, the authors have also discussed the main advantages of microservice architecture.

Muhammad Waseem et al. [8] has shown the challenges of Microservice software architecture with DevOps with the definition of design Microservice architecture, refactoring, and evolution in Microservice software architecture. Claus Pahl et al. [4] has compared the current status of research on microservice architecture and its cloud applications. Based on the study of 21 articles that are related to microservice architecture, authors have proposed a comparison on the selected articles where the structure for characterization is the main concuss. The comparison has shown evolving issues within the microservice architecture.

Nuha Alshuqayran et al. [9] proposed implementation on a study based on systematic mapping in a microservice architecture. The implementation mainly focuses on finding architectural issues, views of the architecture, and quality factors connected to microservice systems. The authors have chosen 33 publications from which to perform an investigation. The most common study categories, according to this research, were assessment studies followed by solution recommendations. Modularity, scalability, and maintainability appear to be among the top discussed traits, according to their research. They further point out that the most underrated discussed attribute is security. According to this research, DevOps approaches are the way of the future.

H.Vural et al. [10]. have done a survey on 37 papers on microservice architecture to compile a list of microservices' characteristics. The proposed approach of this research might be seen as a suitable beginning point for developing a common

microservices definition. The authors have provided a list of a few of the new technologies. REST was at the top of the list, followed by swagger. A list of the most often utilized technologies is compiled where Docker comes in first, followed by Node.js and MySQL.

Dharmendra Shadija et al. [11] compared microservices architecture and SOA based on several papers written about SOA and microservices. The process of defining microservices is also an important topic in this work. While similar attempts have been undertaken in other articles, the study stands out for the breadth of its discussion.

3 Methodology

Service-one and service-two are the two services offered by the proposed application. Each service has a database of its own, such as service-1-db and service-2-db. At the time of service starts, it stores the name of the service and a UUID that is auto-generated in relation to the point of view of the database and then transmits the information to the exchange of RabbitMQ, which shows the data to every thread depending on the key for routing. All microservice attend to a queue provided by RabbitMQ and update the database as soon as new data arrives.

3.1 Service Registration

There are servers namely discovery and registration. To set it up the service has to be registered on both the servers(in our case it's a Consul named HashiCorp) as shown in **Fig. 1**.

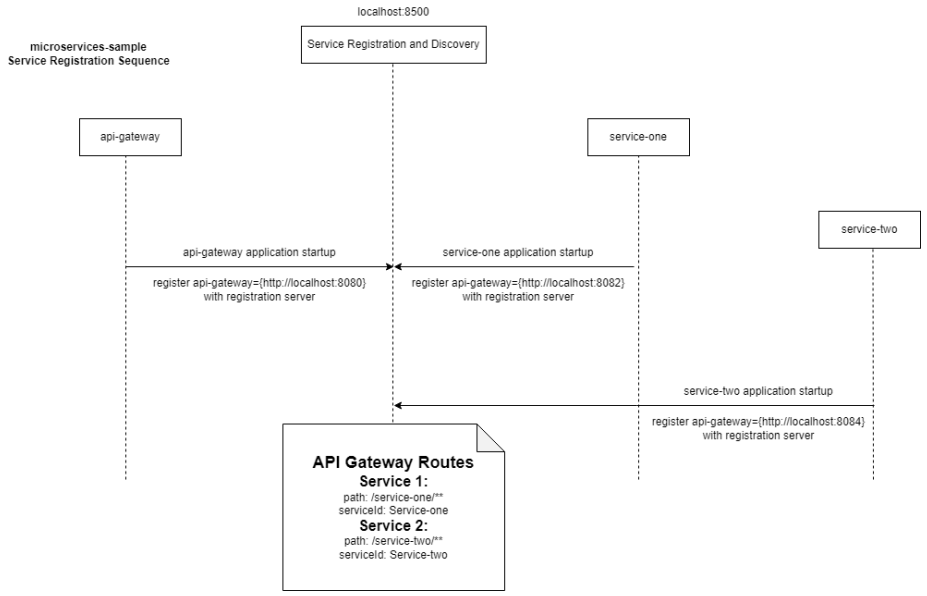

Fig. 1. Service Registration Sequence

3.2 Service Discovery

Once a service (like a gateway of API) allows for it to update a particular commodity from a different service (for example, service one), it only needs to request the identification and server of registration (Consul) for one instance of a service's details. The whole process is figuratively described in **Fig. 2.**

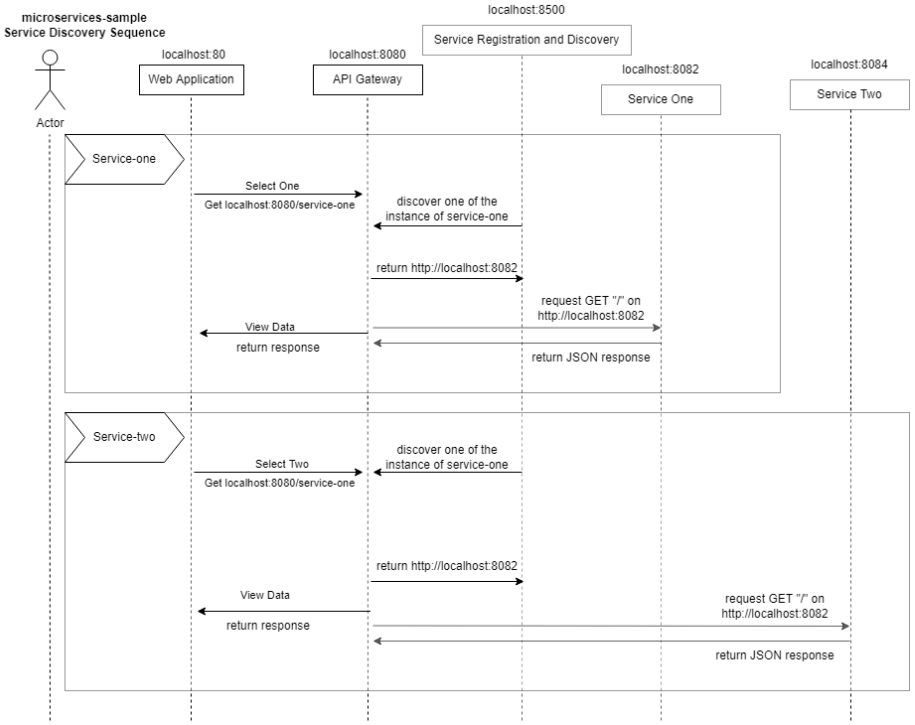

Fig. 2. Service Discovery Sequence

3.3 Architecture

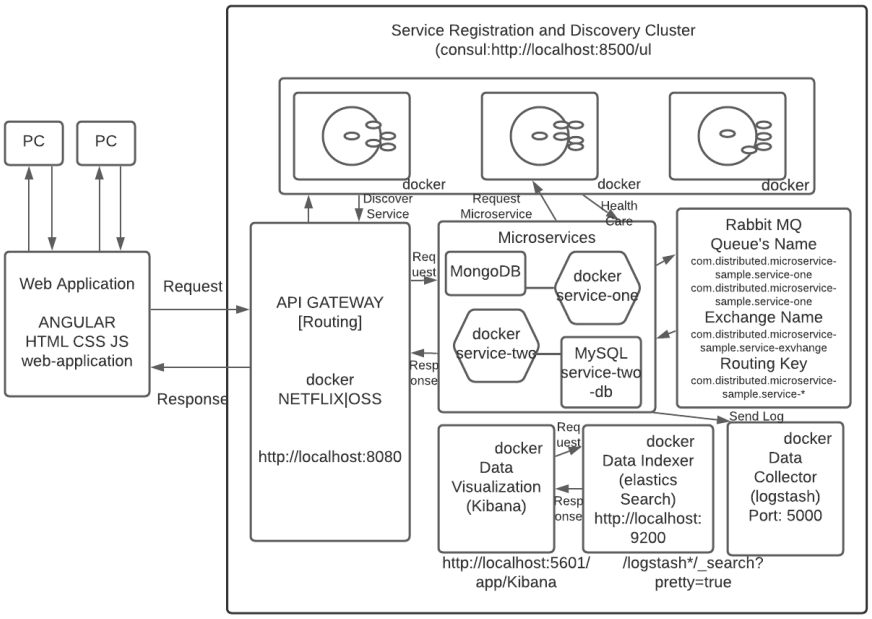

Fig. 3. Architecture

Netflix Zuul is a reversal server of proxy that serves as a Gateway of API for connecting micro-services at the back of the same gateway that directs requests to the appropriate service. Microservices are hidden there behind the reversed proxy server and must be accessed through an API gateway. The whole architecture is shown as a flow in **Fig. 3.**

3.3.1 Routing configuration finished in Gateway of API:

```
zuul:
  ServicesIgnored: "*"
  routes:
     First:
      (path): {/service-1/}
      service_Id: Service-1
     Second:
      (path): {/service-2/}
```

```
      service_Id: Service-1
```

The Consul of HashiCorp is in charge of registration and discovery. Individual services register their details with the service registration service during launch, including such hostname, port, and so on, whereby the accessibility of the services are possible. Again when the registration of the service was completed with the help of the consul, the consul examined the quality of the service by dispatching a signal for the path of the quality-check path and the quality-check interval which was specified. Requests involving microservices have to be routed with the help of the gateway of API, which uses the gateway of API to reach out to the service of discovery to gather relevant data necessary to dispatch the request to the correct microservice.

3.3.2 Microservices configuration for Consul registration:

Management_info:

```
context_Path: {/actuator_name}

  Spring_info:

  application_name: service-1

  Cloud_info:

    Consul_info:

      host_name: consul

      port_number: {-8500 or the value the will be set}

     Discovery_panel:

        host_Name: service-1


instance_Id:${Spring_info.application_name}:${Spring.appl
ication_instance_id:${Random.value}}

        health_Check_Path:
${Management_info.context_Path}/quality

        health_Check_Interval: 20 seconds
```

4 Result Analysis

To begin, the main application is located at localhost:80. The three services, 'Service-1,' 'Service-2,' and 'Service-3,' are readily visible in this application, and by clicking any of them, the user is directed to the API gateway, which determines where the request should be sent. If you select Service-1, the APIs will be contacted at localhost:8080, and the call will be internally redirected to localhost:8082, where our Service-1 is located. Service-2 is available at localhost:8084. Furthermore, the last one, Service-3, is accessible at localhost:8086. Basically, when someone clicks on Service-1, it sends a request to the API gateway on port 8080, which then determines where Service-1 is running by speaking with Consul. Consul actually gives the port details of the services that are coming up, and every time they are called by APIs, the APIs ask the consul for their information. This is the same for all of the services. Kibana has been our consolidated logging system. Second, weave scope was used to monitor and visualize the containers since it provides a bird's-eye view of your app and its entire architecture, as well as the ability to detect any difficulties with your distributed containerized software in real-time while it's being deployed to a cloud provider. Finally, we used ELK for centralized logging. When Logback is paired with a microservice-Logstash, it generates application logs and delivers them to a logging server. Using Logstash-Elasticsearch, the data is prepared and transmitted to the indexing server. Kibana may be used to display data from an elasticsearch server visually. Furthermore, asynchronous microservices communication, which is essentially intercommunication between microservices, is carried out asynchronously using simply RabbitMQ.

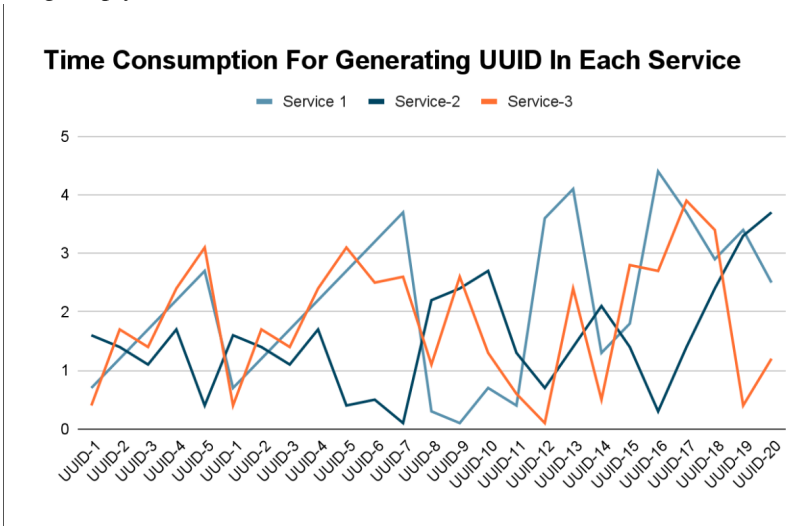

Fig. 4. A line chart for showing a performance comparison between services

From the above **Fig. 4.** it can be seen that for each service, in this microservice, we observed that, for generating UUID each time and storing it into their own database, the time consumption varies. In the above For example, we picked a specific time range which is 0-4 seconds, for each service, and took 20 samples to check, how much time it takes for each service to generate and store UUID into their database. By seeing the line chart, we can tell that Service-1 is working best till the end because it is taking lesser time than the other two services, though, in the beginning, Service-2 was in top performance, but after that only Service-1 was second-best compared to Service-3, in the mid. Eventually, Service-1 turned out to be the best service.

Table 1. 20 samples for each service.

Unique Keys	Service-1	Service-2	Service-3
UUID-1	0.7	1.6	0.4
UUID-2	1.2	1.4	1.7
UUID-3	1.7	1.1	1.4
UUID-4	2.2	1.7	2.4
UUID-5	2.7	0.4	3.1
UUID-6	3.2	0.5	2.5
UUID-7	3.7	0.1	2.6
UUID-8	0.3	2.2	1.1
UUID-9	0.1	2.4	2.6
UUID-10	0.7	2.7	1.3
UUID-11	0.4	1.3	0.6
UUID-12	3.6	0.7	0.1
UUID-13	4.1	1.4	2.4
UUID-14	1.3	2.1	0.5
UUID-15	1.8	1.4	2.8
UUID-16	4.4	0.3	2.7
UUID-17	3.7	1.4	3.9
UUID-18	2.9	2.4	3.4
UUID-19	3.4	3.3	0.4
UUID-20	2.5	3.7	1.2

This **Table 1** was the table of our 20 samples, all of which were taken between 0 and 4 seconds. The time of response was counted for each UUID generation in these 20 samples for Service-1, Service-2, and Service-3.

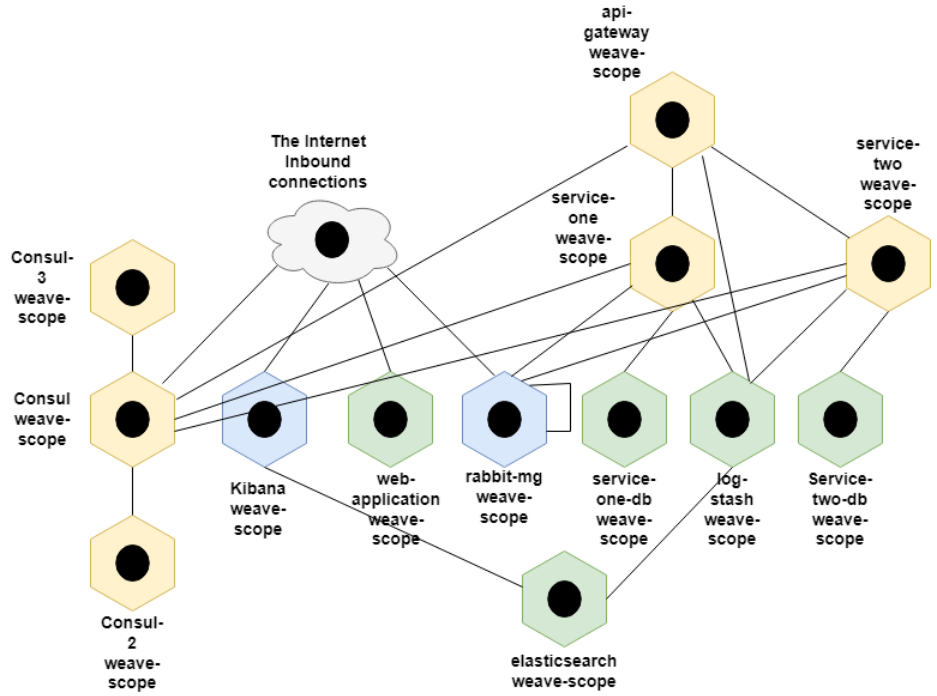

Fig. 5. Connections between services

From **Fig. 5.** we can see that Consul, Consul-2, Consul-3 are the three main Consuls. Consul is connected with both Consul-2 and Consul-3. Consul is connected with the internet inbound connections as well. Kibana, Web-application, Rabbit-mg are connected with the internet inbound connections. Consul is connected with API-Gateway. API-Gateway is connected with Service-One. Service-one is connected with Consul and Rabbit-mg. Rabbit-mg is innerly connected with itself as a loop and also it is connected with service-one. Service-one-db is connected with service-one. Additionally, Service-one is connected with long-stash. Long-stash is connected with elasticsearch. Also, elasticsearch is connected with Kibana. Service-two is connected with API-Gateway. Service-two-db is connected to Service-two, Consul, and Rabbit-mg.

5 Conclusion

This paper has been successful in demonstrating how to design an architecture of microservices by creating cloud apps. We also shed light on some of the trends of

microservices, the purpose and vision for investing in such research. Discussed some of the technical gaps while also touching upon the upcoming standards design standards. Future work could be to develop a more robust architecture and enhance the services in such a way so that the time consumption is lesser than before, hence saving time.

References

1. Lewis, J. and Fowler, M. (2014). Microservices. http://martinfowler.com/articles/microservices.html.
2. P. D. Francesco, I. Malavolta and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption," 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 21-30, DOI: 10.1109/ICSA.2017.24.
3. Hamzehloui, Mohammad Sadegh et al. “A Study on the Most Prominent Areas of Research in Microservices.” International Journal of Machine Learning and Computing (2019): n. pag.
4. Claus Pahl and Pooyan Jamshidi. 2016. Microservices: A Systematic Mapping Study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2 (CLOSER 2016). SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 137–146. DOI:https://doi.org/10.5220/0005785501370146
5. Namiot, Dmitry, and Manfred Sneps-Sneppe. “On micro-service architecture” International Journal of Open Information Technologies 2, no. 9 (2014): 24-27.
6. N. Mendonca, P. Jamshidi, D. Garlan and C. Pahl, "Developing Self-Adaptive Microservice Systems: Challenges and Directions" in IEEE Software, vol. 38, no. 02, pp. 70-79, 2021.DOI: 10.1109/MS.2019.2955937
7. L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps," 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 39-397, DOI: 10.1109/ICSA.2018.00013.
8. M. Waseem and P. Liang, "Microservices Architecture in DevOps," 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), 2017, pp. 13-14, DOI: 10.1109/APSECW.2017.18.
9. N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), 2016, pp. 44-51, DOI: 10.1109/SOCA.2016.15.
10. Vural, Hulya et al. “A Systematic Literature Review on Microservices.” ICCSA (2017).

11. D. Shadija, M. Rezai and R. Hill, "Towards an understanding of microservices," 2017 23rd International Conference on Automation and Computing (ICAC), 2017, pp. 1-6, DOI: 10.23919/IConAC.2017.8082018.