Microservices vs. Monolithic Architectures

Nada SalahEddin ElGheriani[1]
nada.slhxx@gmail.com

Abstract: Software development seems to be on the rise, owing to growing necessity for businesses to go online. There is a push to develop upgraded applications that will help businesses become more efficient and grow. Today's software architectures include monolithic and microservices, which are both popular and powerful. For microservice architecture brings practical benefits such like scalability and flexibility, as well as being a cost-effective means of developing large applications. On the flipside, the monolithic approach is losing favor since it endangers current software delivery methodologies. In this paper, we will discuss the differences between Microservices and Monolithic Architectures, highlighting their strengths and weaknesses in each, and minding a comparative depending on selecting a simple travel application structure, which wins to be chosen as the best choice in software business world.

Keywords: Microservices Architecture, Monolithic Architecture, Software Architecture, Application

[1] Researcher: College of Computer Technologies-Tripoli [CCTT], Tripoli, Libya

1. Introduction

When it comes to the question of how to structure code, all software need an architecture for better understanding, communication, consensus, and negotiation among the various stakeholders. The security, dependencies, guidance, and implementation of defined guidelines are all determined by the software architecture, which is the backbone of the software project. Many companies, as Netflix, Amazon, and eBay, have shifted their applications and systems to the cloud because the cloud computing paradigm allows them to scale their computer resources according to their needs. Microservices Architecture is also regarded as a method for developing software. A software application created using a self-contained set of components that operate each application process as a service is described here. A component in a loosely connected architecture is independent of the others, can be written in different programming languages, use multiple data storages, and only performs one function. At the other hand, Monolithic architecture is an integrated approach to software development. All software components are linked and interdependent in this environment. To run code or compile effectively in such a tightly connected architecture, each component and its related components must be functional.

1.1. Monolithic Architecture

One of the oldest software architectures is monolithic architecture. The goal of this design is to construct an application with all of the required components. All components are interdependent and, in many cases, cannot run or even compile independently. Monolith programs have a lot of distinct libraries inside of them. Monoliths, on the other side, as companies grow, the monoliths grow with them, where monolith holds all classes, functions, and namespaces for the entire application, the benefits of employing a monolithic architecture include that all logic for handling a request runs in a single process. Certain jobs, such as testing, become trivial for developers as a result of this. Monolith applications can be divided into three categories. Monolithic applications with only one process are the simplest. Many components can be combined into a single process in such applications. This is the most popular sort of monolith system. There are also monolithic systems that are widely distributed. Distributed monolith systems may include numerous distributed components that are incompatible with one another and must be installed together. In most cases, monolithic applications installed as a single service, with the exception of distributed applications. Monolith applications are frequently a single process, therefore monitoring them is straightforward. End-to-end testing of monolith applications is definitely simple. Monolithic application developers do not have to consider aspects of securing

communication between several services, which is an advantage. Certainly, cheaper hosting costs for monolithic programs can be significant. Monolithic applications often perform better than microservices applications. The lack of requirement for communication with another process or service is the major reason for this. Monolithic applications have a number of problems, one of which being their vulnerability. It is common for one issue to have a huge impact on the overall application, rather than simply a small fraction of it. Monolithic applications are also technologically driven. The same technological stack must be used to construct a complete application. It can lead to trouble when updating a software's code to a newer framework version. Another challenge with monolithic applications is that they can be difficult to interface with other software.

The development team is bound to commit to a single technology in monolithic architecture, which has its own set of limitations. For example, if the application's framework becomes obsolete over time, shifting to a newer, better framework can be difficult. It's possible that the development team will have to rewrite the entire application in a new language and on a different framework, which is both risky and time consuming. Monolithic features a massive codebases that is made up of interconnected components. Because modifying a single component requires the entire application to be redeployed, such an application structure makes frequent deployments difficult. This not only causes problems with background processes, but it also affects the operation of associated services. Due to the sheer risk of redeployments, frequent application changes are forbidden, Table 1 presents strengths and weakness of Monolithic architecture according to its base structure.

1.2. Microservices architecture

Business logic has split down into minimal, single-purpose self-contained services in microservice architecture. Each architectural service is in charge of achieving a certain commercial aim. In essence, the microservice architecture resembles a Lego construction that has been broken down into several components. Application Programming Interface (API) guarantees that the system's components communicate with one another. The phrase "microservice" has recently received a lot of traction. Starting with Netflix, where has been adopted by a number of companies, including Amazon, Spotify, and Sound Cloud. The microservice architectural style is a method of developing distributed systems that developed to address the shortcomings of monolithic architecture. It is an extension of classic service-oriented architecture (SOA), highlighting the division of a system into small, compact, and loosely connected services, each executing on its own process space and developed with the goal of performing a highly integrated business function. The autonomy of services is one of the primary characteristics of microservices. While microservices are independent of one

another and should not share the same data source, they can also be deployed independently. Each service is in charge of a little amount of logic that is centered on the same business domain. The question is whether the microservice should be modest or large. It can be defined as an application that can be rewritten in two weeks. Instead of having one large-scale, service that handles all business logic, microservice architecture is a distributed system where all its modules are microservices, each dedicated to a single business capability. This foster separation of concerns and allows each service to be independently replaceable, upgradeable and redeploy able at any time.

Table 1: Strength and weakness of Monolithic architecture

Strengths of Monolithic Architecture	**Weakness of Monolithic Architecture**
Less cross-cutting concerns Logging, handling, caching, and performance monitoring are all aspects that affect the entire application. This section of functionality only affects one application in a monolithic application.	Understanding When a monolithic application grows in size, it becomes complicated to understand. Furthermore, managing a complicated coding structure within a single application is difficult.
Easier debugging and testing Debugging and testing monolithic applications is significantly easier. Whereas the monolithic software is one indivisible item.	Making changes It is harder to implement changes in such a large and complex application with highly tight coupling. Any code change affects the whole system so it has to be thoroughly coordinated. This makes the overall development process much longer.
Simple to deploy There is no need to deal with several deployments when it comes to monolithic applications, just one file or directory.	Scalability Only the whole application.
Simple to develop Any technical team with proper abilities and knowledge can design a monolithic application as long as the monolithic approach is a typical way of building applications.	New technology barriers Applying a new technology to a monolithic application is exceedingly difficult because the entire application must be rebuilt.

Characterized control and data management are also provided by the independence. The database per service pattern refers to each service having its own, independent storage system that can be built with the most appropriate technology stack for the work at hand. Microservice architecture is flexible and creates opportunities for companies to respond faster to inevitable change because of its evolutionary design. Microservice architecture can increase agility, developer productivity, resilience, scalability, reliability, maintainability, separation of concerns, and ease of deployment. However, all of this comes with its own set of challenges. Services communicate over the network as microservices are built independently. This requires service discovery, enhanced security management, improved communication, and load balancing. Microservices architecture allows developers to build applications utilizing a variety of technologies (languages, frameworks, and operating systems). With a single technological stack, this eliminates dependency and long-term commitment. A new, better technology stack can be implemented whenever a new service is established or an old service is updated.

This also reduces the development team's dependence on a single resource for creating or updating services. Each component of an application is operated and scaled separately in microservices. This means that if one of the application's services is changed, the other services will remain unaffected. Furthermore, microservices do not exchange code or implementation with other microservices. Microservices architecture makes continuous development and deployment of large, complicated applications simple. Using these little standalone, applications that provide specific functionality have both strengths and Weakness points, Table 2 - presents the strengths and weakness of Microservices architecture.

1.3. Research Questions

How monolithic Works?

The monolithic architecture is made up of various components that work with each other to define it. These include the following:

Business layer; this layer specifies what the application must do to meet with the business logic.

Database layer; It serves as a repository for all required data objects.

Presentation layer; this layer improves the communication between the user interface and the browser.

More specifically: the application's requests and responses.

Persistence layer; it takes care of object-relational mapping and other tasks.

In a monolithic architecture, all the layers act in isolation from each other. It implies that any change in any layer requires an update on the remaining layers to be executable. How microservices work?

Table 2: Strength and weakness of Microservices architecture

Strengths of Microservices Architecture	Weakness of Microservices Architecture
Independent components As starting with, all of the services can be installed and upgraded independently, allowing for greater flexibility. Moreover, a problem in a single microservice affects only that service and does not affect the entire application. In addition, adding new features to a microservice application is significantly easier than adding them to a monolithic program.	Extra complexity Whereas microservices architecture is a cloud - based system, the connections between all of the modules and databases must be established and set up. Furthermore, if an application has independent services, each one must be deployed separately.
Easier understanding Microservice applications are better to understand and manage since it is broken down into smaller and simpler components.	System distribution Since a microservices architecture is a complicated system with many modules and databases, all of the connections must be carefully managed.
Better scalability The microservices architecture also has the advantage of allowing each component to scale independently. As a result, the entire process is less expensive and time-consuming than with monoliths, where the entire application must be scaled even if it is not required.	Cross-cutting concerns To deal with a number of cross-cutting concerns when developing a microservices application. Externalized configuration, logging, metrics, health checks, and other features are among them.
The higher level of agility Any flaw in a microservices application only affects one service, not the entire solution. As a result, all of the modifications and experimentation are carried out with fewer risks and errors.	Testing Testing a microservices-based solution is significantly more difficult due to the large number of independently deployable components.

The microservices architecture divides functions into smaller parts known as services. These services are set up to carry out specific activities or operations.

Moreover, the database structure is such that depending on the type of data needed to handle specified tasks, each microservice has its own database or may share one. The microservices communicate with one another using an API. An application-programming interface (API) is a collection of instructions that allows users to call the application and receive the necessary data. The API retrieves data from numerous microservices in order for the end client's mobile or desktop applications to conduct the appropriate requests.

Why microservices?

Each microservice can be deployed independently There is no need to re-deploy the entire application with each update, and for complicated applications, continuous deployment is conceivable. The team dedicated to that service in terms of development can build Microservice independently. Decomposing an application makes it considerably easier to design applications that are more comprehensible and maintainable. Adapting to new technologies is simple since developers are free to choose the technologies that make sense for their service rather than being restricted to the choices made at the start of the project.

Which software architecture to choose?

To answer this question, business requirements such as project budget, estimates, and revenue must first be stated. Small enterprises and startups will thrive from the monolithic approach since that allows for easy development and deployment. But at the other hand, microservices will aid in the implementation of changes and upgrades to the application structure.

By considering the following scenarios, for making the choice to choose a monolithic architecture firstly, project must be informed in many different ways, such as:

- Plan to create a basic application that will not require future upgrades. The project's complexity will be a disadvantage in this situation due to the microservices architecture's complexity.
- The product must be deployed as soon as possible.
- Will not have a staff of specialists who can partition the system into different functionalities and assign roles.

Reasons for making the choice to choose a microservices architecture secondly:

- Set to deploy a large-scale, sophisticated application. The implementation of new technologically advanced stacks will be the ideal answer if new functionalities and upgrades to the application that required in the future. Microservices strategy will be particularly useful for gaining a competitive advantage in this regard.

- If there were specialists who are strongly tied to microservices and have extensive knowledge in this industry, this would be ideal for microservices architecture strategies.

Not all architecture created equal. Likewise, although not all applications created similar, microservices are better suited to sophisticated and dynamic applications, on the other hand, will be a monumental task without sufficient competence in these technologies. In addition, many people think differently in architecture. Some people believe that the application should build as a monolith initially for being easier at first then switched to microservices when growing. However, the main point in software architecture that there would be no need to start with monoliths if the goal was to construct a microservices application. Choosing the best architecture always depends on software's structure and the purpose that created for service.

The differences in software development methodologies among microservices and monolithic architecture are shown in Table 3.

1.4. Software Industry Issues and Priorities

All software companies aspire to be more agile and faster, to deliver products more quickly, to have shorter release cycles, and to help their customers achieve Digital Transformation. Hundreds of legacy apps are sitting in on premise data centers for most mid-to-large businesses, and they want to shift them to the cloud. They commonly transfer their applications to the cloud in their current state, but this is ineffective. Cloud applications must build in such a special manner. This paper will show how shifting an application structure from monolithic to microservices can change the whole business strategy, Where Microservices provide lots of benefits over monolithic systems, despite the fact that they are a relatively new idea. Numerous monolithic applications previously been developed. They are not really cloud natives

2. Methodology

To compare the structure of two different architectures on travel application, initially identify the first application, which built using monolithic and prove the challenges of shifting to microservices, then understand how microservice application can provide backend capabilities by exposing APIs, where the gateway displayed in the frontend of the entire system that includes all of the APIs of every microservice application in the system.

Table 3: Development methodologies among microservices and monolithic architecture

Monolithic Architecture	Microservices Architecture
Follows a basic architecture with tightly connected service components.	The architecture is structured into a variety of loosely coupled service components.
Usually takes longer to develop.	Quick to develop and takes little time.
All of the components are interdependent, and if one goes down, the entire application falls.	Any service that goes down can be addressed independently without affecting the application.
Requires less resource distribution because the services are It follows a centralized approach dependent.	Each team can work independently under a microservice architecture.
Continuous and rapid application delivery is tough for the application development process must be initiated from scratch.	The entire team contributes to numerous advancements, allowing for speedier development and ongoing delivery.
Less scalable	Highly scalable
Difficult to communicate amongst teams.	The teams' communication is clear and effective.
Project-based approach.	Product-based approach.
One technology can be utilized at a time, and the application cannot be upgraded with newer technologies.	Various technologies can be applied to different service components.

2.1. Microservices Platform

For this type of application, microservices-based system would be suitable. A microservice would handle each phase of the data preparation process: data collecting, filtering, normalization, enrichment, aggregation, and reporting. The microservices approach naturally provides a trackable lineage, making it very easy to trace back, which microservices may need to be modified if data issues happen. In the case of microservices architecture works through component isolation. If one microservice fails in this loosely connected architecture, it is easy to spot and

assures that other sections of the program are unaffected. The proactive avoidance of platform inefficiencies and failure through microservice autonomy is becoming increasingly relevant for industries and enterprises that frequently deal with a high volume of requests or the interchange of sensitive data across multiple applications. Furthermore, the separation of distinct platform components makes monitoring easier, technical staff can also tap into deeper functionality from a service-layer perspective, and empower continuous development, integration, and refinement of their important processes in a way that can actually transform and move ahead due to the power of microservices architecture; see Figure 2.

2.2. Travel application architecture Scenario

Software Company has implemented a travel platform; this application has implemented using monolithic backend system. While it was difficult to integrate hundreds of service providers because its technology greatly differs and has some minimal technical expertise, the replacing architecture base needed eventually.

2.3. Monolithic architecture in use

Serving so many routes with so many different transport modes, plethora of offers, promotions, rules, dynamic timetables, moving grounds, and a seemingly unlimited amount of journey segment permutations is no small engineering achievement. The magic that transpired between pressing the search button on the landing page and showing a list of the most intriguing connections was managed by a single monolithic application at the beginning, for when the volume of the traffic, data, codebase, and number of external integrations was significantly was much smaller. However, as everything continued to rapidly increase in numerous dimensions, it became increasingly difficult to build and operate. The whole stiff working probably know what happened next, by keeping up with current software architecture and technical advances. Everything concerning provider integration was built as a monolithic service because the design was simple enough. Figure 1 illustrates the Monolithic architecture.

Integrating providers is just one part of the puzzle when it comes to creating a search result. Originally, all of the code was contained within a single monolithic application. Conflicts arise frequently among teams working on the same codebase. Changes in one place had unintended consequences in other others. It was difficult and time-consuming to coordinate releases amongst teams. A large number of changes released at the same time raised the chance of failure. Difficult to trace the particular commit that caused a failure if one occurred, and it may not always be possible to reverse it because there may have been a lot of other changes stacked on top of it.

2.4. Application Architecture

Travel: Airfares displayed and airline tickets purchase.
Hotel: Hotel room listings and reservation.
Purchase: Providing users with the ability to make payments.
Authentication: Identification service for signups, logins, and logouts, among other things.

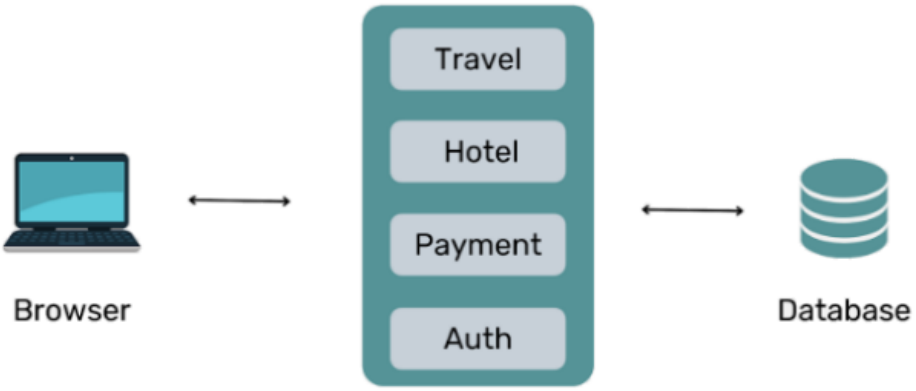

Figure 1: Monolithic Architecture

2.5. Thinking Microservices

Application growing study: By choosing to use feature toggles, one may easily enable or disable new code execution pathways and functionality based on database entries. This would be, nevertheless, frequently difficult and blunder, as there was sometimes a need to branch the logic in multiple locations. Away from the increased development complexity, the application grew increasingly difficult to start and demanded increasing amounts of resources. As a result, increasing the number of replicas in production proved challenging and inefficient in terms of CPU and memory usage.

First, a single architecture that controls the layout of basic search results, bookings, and travel modes, among other things. The model comprises a search result structure that incorporates travel legs, segments, stations, trains, stops, and offers, among other things.

Second, a bootstrap application that serves as a runtime framework around the code for specific provider search integration and displays all of the required API endpoints as specified in the Search Core contract.

Finally, when building a new integration, a code-generation tool based on Rhythm templates are used to generate all of the glue code.

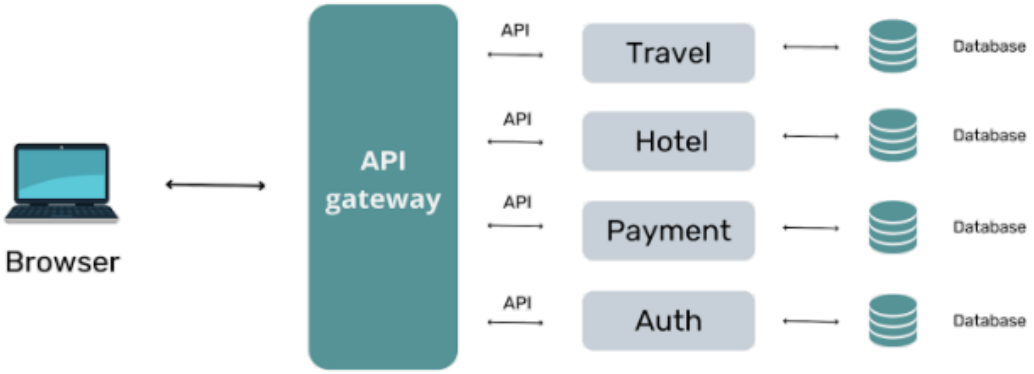

Figure 2: Microservices Architecture

2.6. Layout service in Microservices

Since microservices are lightweight, self-contained, and Cloud hosting that are simple to comprehend, deploy, and scale. While constructing this application as collection of small services, each executing in its own process and communicating using lightweight mechanisms [HTTP resource, and API].

Description: Microservices supports dividing a monolithic application into smaller bits or components. In other words, it separates each module from the monolithic architecture as a standalone monolithic architecture. It takes a modular approach to huge application development. The microservices' individual services or components are loosely coupled with one another. Additionally, they function as a self-contained unit. By comparing the monolithic architecture to microservices, standing on three basic steps (Development, Troubleshooting, and Deployment) greatly enhances the company's experience when developing an application.

2.7. Development

The monolithic design, most likely, use a single codebase or repository. This implies that all developers contribute separately to a single codebase. They are pulling the most recent code from a remote repository, as well as pushing and merging their own. This creates a lot of code clashes as a development workflow. As a result, resolving these issues may cause the workflow to slow down. As going in-group of developers is working on the payment mechanism of application, another team could be working on the authentication module at the same time. However, they are both bringing in new code and pushing it to the

same repository. This is not necessary because their activities are unrelated and could generate confusion. The payment team no longer has to be concerned about the authentication team's most recent codebase changes. Similarly, the authentication team can push their code without involving the payment team.

2.8. Troubleshooting

The architecture in monoliths is tightly coupled frequently leads to difficult-to-trace faults and defects. This is due to the fact that modifications in one component or module can frequently result in unintended changes in other connected modules. However, as working on a flaw in the authentication module, as it is so close to the payments module in the same codebase, technical staff accidentally introduced an unwanted behavior in the payments module. In this situation, the monolith suffers from the drawback of being more vulnerable to application bugs. In addition, microservices are more flexible in this situation. Because of payment module is lightly connected with the authentication module, no matter how many flaws is create in the authentication module, the payment module will stay intact. Since the authentication and payment modules are different projects, even inadvertent changes to the authentication module will not affect the payment module.

2.9. Deployment

In monolith, updating the authentication module on the server requires redeploying the entire application. One of the major drawbacks of monoliths is that a slight change in a supposedly isolated module can result in the entire application being deployed. Furthermore, if the factor in the resulting downtime, it could be a difficult condition for company, users, and entire product. In this situation, the related delay is only visible in the authentication microservice. All of the other microservices continue to function normally. Figure 3 and Figure 4 illustrate the full application deployment–monolithic and individual services deployment–microservices respectively.

2.10. Cost and Complexity

For microservices the more components the architecture has, the more complex it is to manage and maintain them. As a result, the whole application's cost and complexity increases, having many services, each hosted separately means having independent codebases. In addition, each microservice should have its own database, the additional cost and complexity of hosting databases could be significant. Monolithic is straightforward, not lavish. In terms of infrastructure, just one server is required for the application and another for the database.

Figure 3: Full Application Deployment – Monolithic

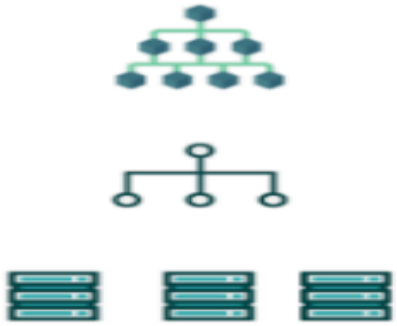

Figure 4: Individual Services Deployment - Microservices

2.11. Communication between sectors

To book a flight, the user launches the application. Two application modules are used in this scenario. The first is the travel one, which displays all available flights, reserves a seat for user, and generates e-tickets, and so on. Second, payment; without paying, no reservation can be made.

By mentioning how the travel module communicates with the payment module in monolithic, first; at the time of payment, invoking the payment module from the travel module. Furthermore, in monolithic architecture, all modules are in one location. The code works from any location.

The travel microservice must communicate with payment microservice. Each microservice needs to interact with the others if user wanted to book a complete travel package like, airline tickets, hotel room. Furthermore, some use cases may require a lot of back-and-forth across multiple microservices. For example, user may buy a flight and a hotel for themselves, and then book a hotel for a business

visitor. Clearly, monolithic systems have the ability to be faster than their microservice equivalents.

3. Results

Users are always looking for services that are faster, more efficient, and more performant. Monolithic services might be faster than their microservice equivalents. Inter-services communication is where this performance comparison differs. Each inter-services connection with microservices will almost certainly take the form of an API call. This API call is transformed into code calls or function invocations in monoliths.

Scalability

Using a load balancer and many server nodes to achieve horizontal scaling in monolithic architecture, Monolithic would be able to scale effectively without a single point of failure. As a result, if one of the server nodes or the API server fails, the other nodes can step in to help. However, one disadvantage of scaling monoliths will always be the separation of business logic. Figure 5 illustrates the horizontal scaling- Monolithic Architecture.

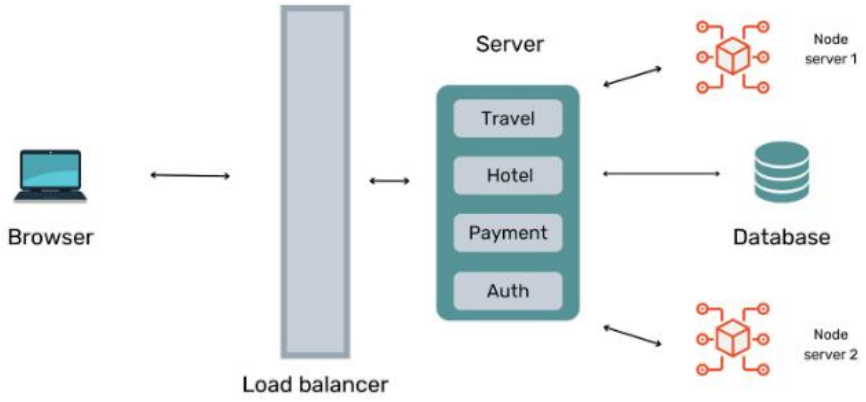

Figure 5: Horizontal scaling- Monolithic Architecture

At the business level, the requirement to divide modules or components becomes more critical. Microservices are incapable of horizontal scaling. Individual microservices can be scaled as and when needed, allowing for granular scaling. Figure 6 illustrates the granular scaling- Microservices Architecture.

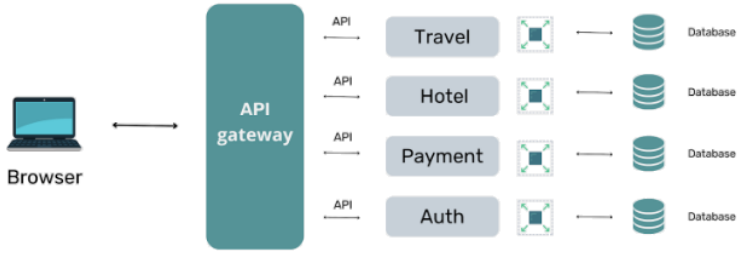

Figure 6: Granular scaling- Microservices Architecture

4. Conclusions

According on the comparisons, the software firm must select which factors are crucial. Monoliths are a good place to start. It is also a good idea to have a small team for monolithic. When it goes to scale the application from a medium to a somewhat large size, horizontal scaling in the monolithic is good. As microservices, fits with a large and diverse workforce, and a diversity of tech stacks, wins this one. As a result, the decision is to bear the cost and complexity of microservices.

References

[1] Lebedev, A. (2021). monolithic and microservices-based architecture. Retrieved from: https://morioh.com/p/54271e7358ad?f=5c224490c513a556c9042463&fbclid=IwAR0Uly4YTIS0cKeKPCR5I_e8ZcrIRJEwA62AUoAAciOT4vhD8VcHZcetkfU.

[2] Dissanayake, B. (Nov 17, 2019). Level up coding. Monolithic vs. Microservices Architecture. Retrieved from: https://levelup.gitconnected.com/monolithic-vs-microservices-architecture-b333c8754187.

[3] Kharenko, A. (2015). Monolithic vs. Microservices, Monolithic Architecture. Microservices Practitioner Articles. Retrieved from: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59.

[4] Jain, P. (October 28, 2020). Monolithic vs Microservices – Difference, Advantages & Disadvantages. Retrieved from: https://k21academy.com/docker-kubernetes/monolithic-vs-microservices/?utm_source=facebook&utm_medium=referral&utm_campaign=kubernetes31_jan21_cloud_for_beginners&fbclid=IwAR3KFlLGXZt_dVIh6DitflogpS2JubJo0DZswr4ods0GYakwHBdb6vrshNU#8.

[5] Spratshi. (June 11, 2020). Monolithic Vs Microservices Architecture. Retrieved from: https://readsngeeks.blogspot.com/2020/06/monolithic-vs microservices.html?fbclid=IwAR0Hbf0Nkqvp0F1qxQAUgHEVBS9acU hYjsTlif5hsvE2aSpCoBV0-7n_mEQ.

[6] Eisele, M. (June 16, 2019). Monolithic Vs Microservices. Retrieved from: https://programmerfriend.com/monolith-vs-microservices/?fbclid=IwAR0YO3B8H2Q43YG2INYHIO7AZ5I6yAb2 UwavO3O_s0K16_9Rf1cDbI_Cirk.

[7] Kaczmarek, A. (Jan 8, 2021). Monolithic vs Microservices - which architecture to choose?. Retrieved from: https://softwaremill.com/monolithic-vs-microservices. architecture/?fbclid=IwAR0083PnR_6Klf0236Lnz8LBMwV22GJewmp ZoucgnFWW63HD-0Rsv-amVg0.

[8] Matloka, M. (Aug 03, 2020). How to design microservices architecture?. Retrieved from: https://softwaremill.com/designing-microservices-architecture/.

[9] Maddikera, B. (Mar 20, 2021). How to Build Micro Services - Real World - Travel App. Retrieved from: https://www.linkedin.com/pulse/micro-services-travel-app-bhargav-maddikera.

[10] Vistola, L. (July 23, 2021). The Move Away from Monolithic Application Development. Retrieved from: https://devops.com/the-move-away-from-monolithic-application-development/.

[11] Sontz, M. (2022). From Monoliths to Microservices – Benefits and Challenges. RCG Global Services. Retrieved from: https://rcgglobalservices.com/monoliths-microservices-benefits-challenges/?fbclid=IwAR0YO3B8H2Q43YG2INYHIO7AZ5I6yAb2UwavO3O_s0K16_9Rf1cDbI_Cirk#:~:text=Refactoring%20a%20monolithic%20application%20to,ahead%20and%20plan%20for%20failure.

[12] Garcia, L.M., Aciar, S., Mendoza, R., & Puello, J.J. (2018). Smart Tourism Platform Based on Microservices Architecture and Recommender Services. Mobile Web and Intelligent Information Systems, pp.167-180.

[13] Blinowski, G. Ojdowska, A. & Przybylek, A. (Nov 30, 2021). Monolithic vs. Microservices Architecture: A Performance and scalability Evaluation. IEEE Access.

[14] Na, N., Wang, W., Xu, Y,. & Luo. W. (2019). A Microservice–Based Big Trajectory Data Processing Platform for Multimodal Trip Planning. International Conference on Big Data, Electronics and Communication Engineering.

الخدمات المصغرة مقابل البنى المتجانسة

ندى صلاح الدين[1]
nada.slhxx@gmail.com

المستخلص: يبدو أن تطوير البرمجيات آخذ في الازدياد ، بسبب الحاجة المتزايدة لأن تدخل الأعمال التجارية عبر الإنترنت. هناك دفعة لتطوير التطبيقات التي تمت ترقيتها من شأنها أن تساعد الشركات على أن تصبح أكثر كفاءة ونموًا. تشتمل هياكل البرمجيات الحالية على خدمات متجانسة وصغيرة ، والتي تحظى بشعبية وقوية. بالنسبة لبنية الخدمات المصغرة ، فإنها توفر مزايا عملية مثل قابلية التوسع والمرونة ، فضلاً عن كونها وسيلة فعالة من حيث التكلفة لتطوير التطبيقات الكبيرة. على الجانب الآخر ، يفقد النهج المتآلف تفضيله لأنه يعرض منهجيات توصيل البرامج الحالية للخطر. في هذه الورقة ، سنناقش الاختلافات بين الخدمات المصغرة والبنى المتجانسة ، مع إبراز نقاط القوة والضعف في كل منهما ، والتفكير في المقارنة اعتمادًا على اختيار هيكل بسيط لتطبيق السفر ، والذي يفوز باختياره كأفضل خيار في عالم البرمجيات التجارية.

الكلمات المفتاحية: هندسة الخدمات المصغرة، الهندسة المعمارية المتجانسة، هندسة البرمجيات، التطبيق

[1] باحثة: كلية تقنيات الحاسوب - طرابلس - ليبيا