An Approach to Break Down a Monolithic App into Microservices

Article *in* Sylwan · January 2021

CITATIONS
0

READS
392

4 authors, including:

Taimoor Ali Syed
Central South University
7 PUBLICATIONS **1** CITATION
SEE PROFILE

Jun Long
Central South University
169 PUBLICATIONS **1,736** CITATIONS
SEE PROFILE

Mansoor Khuhro
Sindh Madressatul Islam University
35 PUBLICATIONS **32** CITATIONS
SEE PROFILE

An Approach to Break Down a Monolithic App into Microservices

Syed Taimoor Ali[1], Jun Long[1*], Vijay Kumar Khatri[1], Mansoor Ahmed Khuhro[2]

[1]School of Computer Science and Engineering Central South University, Changsha 410083, China;
[2]Department of Computer Science, Sindh Madressatul Islam University, Karachi- 74000, Pakistan;
Email: rizvisyedtaimoor@gmail.com

Abstract - IT is a field that is changing rapidly. For this reason, development in the field calls for a huge advancement of knowledge in the field. This article discusses how monolithic application design can be changed into microservices. The article details the comparison between applications of monolithic and microservice nature. This takes into consideration the theory behind decomposition and the various tactics that make it effective. The steps taken when implementing the decomposition process are then given providing a deep examination of the present application design and their technological background. After a detailed discussion of the applications, choices of steps to be followed to decompose them are made and discussed. Mature Amazon Web Services have been utilized in the article. The experiment work was done by decomposing a monolithic application into the microservice architecture, and the results of the architectural decomposition of a monolithic application to microservice and the utilization of the created process of evolution are then given.

Key Words: – Monolithic applications; Microservice; Strange application; Service Computing; Software Architecture;

1. Introduction:

Information technology (IT) is a wide field consisting of many tools, languages especially ones related to programming and a wide variety of backgrounds. This, however, does not mean that IT is a stagnant field since it evolves every day with advancement in knowledge and to cater to the growing day to day demand for IT products by consumers. This, therefore, requires that IT experts make the right choices of the materials to utilize in order to accomplish the goal of satisfying customers' demand for advanced products. On the other hand, the wrong choices of materials would greatly affect companies relying on these experts which would lead to stagnation and loses due to consumers' needs going unsatisfied. There would also be the presence of liability in the technical area of the IT companies that would not be noticeable in the beginning, but which would get glaring in the long term leading to incurring losses.

In the last 10 years, there has been a noteworthy change of mindset and way of doing things in matters concerning the development of the web. Out of date web applications that were based on servers and reinforced by the Relational database management system (RDMS) have been reducing by the day. There has also been a substantial increase in current JavaScript bases like Angular and React. The old structures of databases that were relative are being substituted by current ones that are leaning on documents. This has therefore led to increased intensity in competition and difficulty in the world dominated by web applications. Some of these challenges include the increased need for scalability, great need for quickness, and wild delivery at prices that can be met. To solve these problems, cloud computing has been introduced for it can overcome these barriers.

Testimonies on the effective functioning of cloud services are on the increase and so is the growth of the population of dependents of this useful package. This acts as verification that Amazon Web Services (AWS) is the most precious accreditation in the IT field currently [1]. Microservice design is important and of benefit to huge inventiveness applications that call for great scalability. Some companies are highly experienced when it comes to

handling web application commerce, which serves as a nice indicator of a monolithic application that could do with advantages of utilization and incorporation of a microservice approach [2]. This article is written to enlighten the readers on the progress and developments of the transformation of a monolithic blog application to microservice while it also surveys the downsides that are associated with this decomposition.

2. Research Challenge

There are bunches of sites with a huge number of users that were worked as one enormous monolith. Some of them will likely keep working in this mode for a few years, however, the greater parts of the organizations are endeavoring to separate their items into microservices. This is going on in view of some huge weaknesses of the monolithic applications. The troubles become especially observable when the administration and the group extend.

3. Comparison Between Applications of Monolithic and Microservices

Microservices is a different way of developing applications which instead of making only one application involves making them a set of little services that function to achieve a process. Lightweight devices are used to make communication possible and they include HTTP, special types of RPC and REST.

Having every part of the microservice working independently comes with the advantage of making them programmable using the most appropriate language for the specific service [3].

It is a common practice to have microservices compared to UNIX command services where every utility has one function and one that it accomplishes well. This should also apply to microservices where each set should deal with one business component platform and plays in the context of that unit. To develop each set of microservice a small independent team should be involved.

The size of the team will mostly vary from one case to another although in most cases the size is always small. One team mostly deals with the whole range of technologies that vary from UI, middleware and finally to database managers. This, therefore, is advantageous since less time is spent in communicated between varying teams classified by specialization. In microservice application development, these units can work with agility and react to changes in business quickly enough. This has an important effect on communication that can be reflected in the description found in Conway's law. Conway's law states that any company that designs a system will make a designation that looks just the structure of its communication [4].

Having to work with a small team also comes with the advantage of having small codes to write and correct from the beginning if errors arise which is not as simple when dealing with larger teams and complex codes [5]. Monolithic applications require larger teams to develop which takes much time and it is not simple to correct where many codes are involved. Working with a small number of codes makes it easy to incorporate new team members since understanding them is easy and it also means that members can be transferred from one business unit to another without investing a lot of money by the company [6].

As already mentioned, every microservice act independently which makes it easy for developers to use different technologies in different services when the need arises, and it also comes with an advantage of the possibility of upgrading them with new technologies separately when required. This

means that is possible for the programmers to use different languages without causing contradictions and getting stuck in using one technology on the whole service like it is the case with monolithic applications. By utilizing this they make a choice of the best technology and language to use based on the requirement of every single service. However, it is worth noting that avoiding programming with so many languages is important as it can cause limitations that are not noticeable at first sight.

In GOTO conference held in Chicago 2016, Matt Ranney says that programming using so many different languages may lead to arising of the disintegration of a working culture where people will be grouped into teams depending on the language they prefer.

TABLE 1
Comparison of Monolithic & Micro Services

Category	**Monolith**	**Microservice**
Time to market	Fast in the beginning, slower later as the code base grows.	Slower in the beginning, because of the technical challenges that microservices have. Faster later
Refactoring	Hard to do, as changes can affect multiple places.	Easier and safe because changes are contained inside the microservice.
Deployment	The whole monolith must be always deployed.	It can be deployed in small parts, only one service at a time.
Coding language	Hard to change. As code base is large. Requires big rewriting.	Language and tools can be selected per service. Services are small so changing is easy.
Scaling	Scaling means deploying the whole monolith.	Scaling can be done per service.
DevOps skills	It does not require much as the number of technologies is limited.	Multiple different technologies a lot of DevOps skills required.
Understandability	Hard to understand as complexity is high. A lot of moving parts.	Easy to understand as the code base is strictly modular and services use SRP.
Transactions	Easy to use ACID transactions supplied by the RDMBS.	Hard to implement. Eventual consistency must be agreed on some cases
CI, CD	CI is possible and should be used. CD is hard to achieve	Using CI is required, and the CD should be used. CD enables faster release cycles.

3.2 Decomposition of Already Present Monolithic Applications into Micro Services

It is a common occurrence for monolith beginning to get larger which is accompanied by their getting hard to manage which calls for the team of development making the system more current for it to remain active and competing. This begins with showing signs that include, long cycles of deployment, lack of module separation, the stack of legacy technology, being unable to scaling some parts of the application and difficulty in initializing the project by new programmers [7]. To correct these problems modernization of the system can be done in varying ways. One of the ways does complete rewriting of the codes used in programming of the application This, however, is a very risky way since some issues come up when the developers are required to retain important aspects of the old system as it means old bugs also remain after the rewriting. To avoid this there can be a creation of a new system surrounding the existing one where once the old one is substituted completely then only the current one remains [8]. This method of replacing the old system's parts with new ones when needed is known as stranger application which is adopted from vines in the rainforest which kills the host tree by climbing upon it extracting nutrients until the host tree is completely exhausted off nutrients. Similarly, incorporation of microservice on existing applications gradually grows as each part is rewritten to the point where a new application is formed [8]. The main point of all this is building new features and introducing new functionality to services of microservice nature while the best bits of the old technology is retained. This ensures that the monolithic applications do not grow to unmanageable levels by breaking into different units of microservice nature until the new technology takes over completely.

To achieve success in running the new applications that have been reprogrammed it is necessary to have enough storage. This comes from the fact that applications require storage for saving their state, files belonging to users and for backups. For storage to be passed as responsible it is required to make data durable, available and easy to upgrade. It is possible to achieve all the aspects stated although doing it alone gets complicated since it could involve investing in a lot of money.

To solve these issues Amazon S3 bucket is introduced since it is made to act on them specifically (Amazon Web Services, 2014) [10]. It is a tool that has been made with durability, availability, and scalability of data in mind all at a low cost. In the tool, data is stored in many copies across varying facilities which reduces the chances of losing the data and their size can expand to unlimited measures (Amazon Web Services 2017) [11, 12]. The tool can also be used for hosting static pages of the web which is accompanied by a high level of scalability and low cost than most web hosting services in the market. All these are advantages of using the Amazon S3 bucket in meeting the requirement of storage after decomposition of monolithic web applications.

In conclusion, the decomposition of monolithic web applications involves changing them by incorporating micro service knowledge to make them more effective and modern according to customer demand in the IT field. Making them current is required since it has the advantage of improving their manageability by involving different units with specific functions that can be programmed through different programming units and small cost. Stranger application is the strategy that can be used to achieve this that involves the gradual replacement of the old system's parts with new microservices ones which also requires enough storage. Storage can be improved using an Amazon S3 bucket which is a tool that offers all the requirements or successful decomposition, and which comes at a low cost [13,14].

4. Methodology

The main goal of this article is to help the readers to understand microservice as a trend that is architectural. To achieve this goal, the article will explain how a policy of architectural conversion was made which will prove that microservice is advanced and understandable while it also contrasts it to monolithic style. The realization of this conversion will be explained while the article will also talk about how microservices can be made around an existing monolith. To do this AWS was selected from the fact that it is the most commonly utilized platform and it is established too [9].

This article, it is written around four central fragments. The first of the four deals with cloud computing as a key innovation in hardware reserve provisioning. It is a fragment that will cover the four services provided by cloud dealers. Microservice is the second part, and this is where exploration of its advantages and disadvantages will be done as compared to monolithic style. The third fragment covers a description of decomposing monolithic application into microservice and will be based on theory covering the accomplishing of decomposition of IT application. It describes the various methods that can be used for decomposition to be achieved successfully. The last part is the implementation process where real-world examples of the application of decomposition are discussed.

The description of each fragment focused on a review of the existing data developed with a reflection on their description. As a result, there was a need to conduct a literature analysis focusing on the expectations of the market and the demands of the invoked technology. A review of the relevant literature to support the interests of the article adopted an explorative approach guided by the need to enlighten on the trends faced in the development of monolithic and microservice models. Also, the relevance of the isolated studies was prioritized.

4.1 Proposed Method for Converting Monolithic Modules into Microservices

The proposed method breakdown the monolithic application into microservice segments, where components of an application are divided into separate parts. That is executed and maintained separately from the mechanism system.

Figure 1 illustrates the process of breaking down application modules into the microservices. Steps are described next in this section.

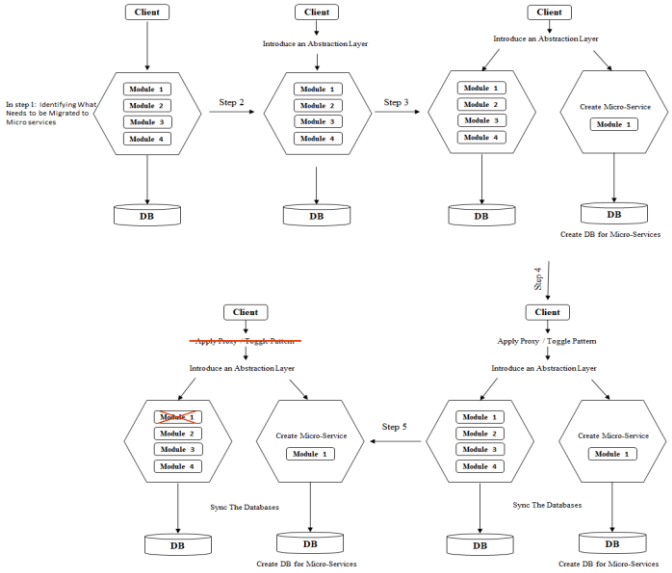

Figure 1. Proposed Method to Convert Monolithic to Micro Services

The process illustrated in figure 1 is defined as following in steps:

Step 1:

- Identifying What Needs to be Migrated to Microservices
- Start with the least complex modules in the legacy system that will have the greatest benefits.

Step 2:

- Apply Branch by Abstraction Pattern
- Introduce a deliberation layer before the old part
- Leave the old inheritance code in its present condition

Step 3:

- Apply Strangulate Pattern
- Create a parallel microservices execution for the recognized modules
- Create an autonomous Database for Each Microservice

Step 4:

- Apply the toggle Pattern.
- Introduce a run-time arrangement toggle behind the reflection layer.
- Deploy the refactored code underway
- Incrementally divert the traffic from the inheritance module to the recently made microservice
- Sync the database of the Legacy module and Microservice
- We can powerfully return to the old part in case of a disappointment or failure.

Step 5:

At the point when advancement and testing are finished, and the traffic is totally diverted to the microservice, wipe out the heritage module and refactor code and Database to delegate to the new segment.

The methodology of this article describes proposing a method for decomposition of a monolithic application into the microservices. Moreover, we implemented a monolithic application and performed the conversion of that app into the microservice architecture by applying our proposed conversion technique. That is discussed next in section 5.

5. Experiment & Result

5.1.1 Monolithic Blog Application

In experimental work, we implemented a monolithic application and breakdown designed monolithic blog application into microservice architecture by following the procedure and steps, that are defined above in section 4.

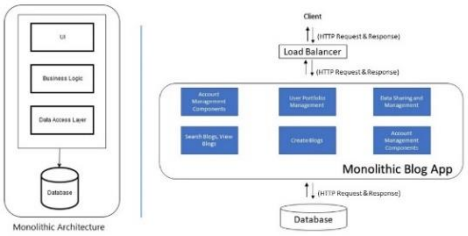

Figure 2. Implemented Monolithic Blog App

Figure 2 illustrates the monolithic structure of the blog application. There are six application modules, that are written in the same language, and can be deployed within a single application and high-performance server. The designed blog application is monolithic in nature. Therefore, the entire software application is composed in a single piece, the components of an application are interconnected and interdependent. If change is made in any one component, or anyone of them is failed to execute or any bug occurs, the whole system will be affected.

After going through the development phase of monolithic blog application we analyzed some challenges with the monolithic architecture, that are discussed as follow:

- **Unscalable**

  Makes caching less effective, increase memory consumption along with I\O traffic, each component may have different resource requirements, so in monolithic nature, it is difficult to scale each component independently.

- **Resist speed of continuous development of components**

  If an application is grown and became a large-scale software, it becomes difficult to modify. The application components have interdependency that resists the continuing development of components. If one component is going to be modified the entire application will be redeployed. Therefore, the application is modified once at a time and So it reduces the productivity of the application.

- **Large & Complex Software Application**

  If it is about a large-scale application, it becomes more complicated to extend, modify and manage the application. due to that quality of code declined over time

- **Unreliable**

  Because of tightly coupled components, if one component has faced bug the entire mechanism is interrupted.

- **Inflexible**

  The application with monolithic nature cannot compromise with the additional frameworks. The adoption of new technologies or frameworks in the monolithic application will consume more time and cost.

5.1.2 Transformation of Monolithic into Microservices

The application modules are classified separately when they come into microservice architecture [15], each component is treated independently as illustrated in figure 3.

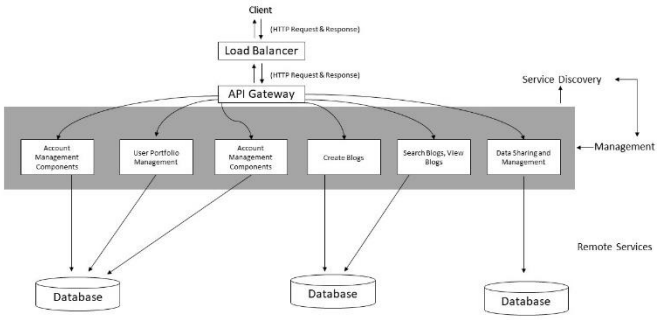

Figure 3. Blog App based on Microservices

Figure 3 illustrates the structure of transformed monolithic application into the microservices, where we have six services, that are small, independent and loosely coupled. Each of the services has a separate code base that is managed by a small development team. The development team is enabled to create, modify or update any service without deploying the whole application, each service can be deployed independently.

The components of the application are divided into small chunks in the shape of microservices, that are portable, flexible and interoperable in nature. The implementation of one service is hidden from other, as in monolithic application if one needs to modify any component, the whole system is modified and redeployed, but in the microservices each microservice works separately, the system will not face any problem if modification is made.

In figure 3 term service discovery manages a list of services and details about services. The term management is including management of service transactions like placing services over nodes, identifying service failure, service load balancing over a network. An API gateway is an intermediate between client and service provider, that is responsible to manage user requests and responses from client to service node.

Monolithic systems are not answerable for a solitary assignment, yet they need a few errands to finish an obligation. In monolithic, entire services are packaged into one bundle and run as one procedure. The UI, information access layer and information store layers are firmly coupled in monolithic applications. Generally, huge groups work with a monolithic system, and they are not appropriate for compartment-based arrangements.

The upsides of microservices appear to be sufficiently able to have persuaded some large endeavor players, for example, Amazon, Netflix, and eBay to embrace the technique. Contrasted with increasingly solid plan structures, smaller-scale administrations:

Improve deficiency separation: Larger applications can remain for the most part unaffected by the disappointment or failure of a solitary module.

Wipeout merchant or innovation lock-in: Micro-administrations give the adaptability to evaluate another innovation stack on an individual help as required. There won't be the same number of reliance concerns and moving back changes turns out to be a lot simpler. With less code in play, there is greater adaptability.

The simplicity of Understanding: With included effortlessness, designers can more likely comprehend the usefulness of help.

The more advantages are listed below.

- It can utilize the most recent innovative technologies to build up the application modules as microservices.
- Composability is very high.

- Can scale single microservice independently. No compelling reason to scale the entire framework.
- disappointment or failure of One segment won't cause the whole framework downtime.
- When building up a general arrangement, we can parallel the microservice implementation task with the little development groups. In this way, it diminishes the implementation time.
- CI/CD is very easy.
- Independent codebase support is troublesome.
- Monitoring the general framework is challenging of decentralization. Correspondence should be solid to speak with autonomous modules.
- Has extra execution overhead due to organizing dormancy.

Conclusion

This research was based on a comparison of monolithic and microservice architecture. The work proposes a new decomposition method to transform a monolithic application into the microservice architecture. To validate the proposed method an application was implemented and converted into microservice architecture by applying the proposed technique and results were found that the microservice architecture has more advantages over a monolithic application. Microservices are in depended and flexible in nature, the system designed with microservice architecture can be extended without affecting the entire mechanism of application. But in monolithic architecture, the whole system is redeployed. Therefore, over experience was good with microservices.

References

1. Columbus, L., 15 Top Paying IT Certifications In 2016: AWS Certified Solutions Architect Leads At $125K. Accessed on 8 April 2017. Retrieved from. 2016.
2. Fowler, M., Monolith First. Accessed on 1 April 2017. Retrieved from. 2015.
3. Lewis, J., & Fowler, M., Microservices a definition of this new architectural term. Accessed on 29 March 2017. Retrieved from. 2014.
4. Conway, M.E., How do committees invent? Datamation Magazine. 1968.
5. Goldsmith, K., Microservices @ Spotify. Video recording from GOTO Conference in Berlin 2015. 2015.
6. Ranney. What I wish I had known before scaling Uber to 1000 services. Video recording from GOTO Conference in Chicago 2016. 2016 [cited 2017 16 November]; Available from: https://www.youtube.com/watch?v=kb-m2fasdDY.
7. De Santis, S., Florenz, L., Nguyen, D. V., & Rosa, E., Evolve the Monolith to Microservices with Java and Node. Redbooks. 2016.
8. Fowler, M. Strangler Application. 2004 April 4]; Available from https://www.martinfowler.com/bliki/StranglerApplication.html.
9. Panettieri, J., Cloud Market Share 2017: Amazon AWS, Microsoft Azure, IBM, Google.
10. Jinesh Varia and Sajee Mathew, Overview of Amazon Web Services. January 2014]; Available at https://d36cz9buwru1tt.cloudfront.net/AWS_Overview.pdf.
11. Amazon Web Services, Inc, Overview of Amazon. 2019]; Available at

https://d1.awsstatic.com/whitepapers/aws-overview.pdf.

12. CloudFormation, A. Official AWS documentation. 2017; Available from: https://aws.amazon.com/cloudformation/.

13. Mayur Palankar, Adriana Iamnitchi, Matei Ripeanu and Simson Garfinkel, Amazon S3 for Science Grids: A Viable Solution, Computer Science and Engineering, University of South Florida 4202 E. Fowler Ave., Tampa, FL 33620, USA]; Available at https://www.ece.ubc.ca/~matei/PAPERS/S3.pdf.

14. Balakrishnan, A. AWS CEO Andrew Jassy's 2016 pay hits $35.6 million. 2017; Available from: https://www.cnbc.com/2017/04/12/aws-ceo-andrew-jassys-2016-pay-hits-35-6-million.html.

15. Richardson, C. Refactoring a Monolith into Microservices. 2016; Available from: https://www.nginx.com/blog/refactoring-a-monolith-into-microservices.

Funding: This research was funded by the National Natural Science Foundation of China (61402165, 61702560).