MAKING THE SWITCH

Briefing Document for Transitioning from a Monolith System

HEP515 – Software Engineering

Abertay University

LUKE
GILBERT

INTRODUCTION

In an age where technology seems to be moving at light speed, some businesses feel like their software systems can get outdated and outclassed overnight. Due to its historic popularity, the system being transitioned is often Monolithic to something more modern which can keep up with industry demands for software such as Microservices.

This document has been produced to provide insight into this transition between an existing Monolith system which is employed at your other gym locations to the more robust modern Microservices system which has been developed for your gym management app.

This document will cover several different areas including the process to make this transition and the advantages and disadvantages of making the switch. It should be stated here that the information in this document will comprise research I have conducted along with my own opinion on the subject matter.

THE SWITCH

Switching from a Monolith System to Microservices is no simple process and must be carefully planned to ensure successful migration from one system to another.

This process can be broken down into 8 key phases (Frye, 2020).

Understanding the Existing System

This phase involves a review of the existing system, what data structures and operations it includes and what functionality it currently has. At this stage access to documentation for the system will be highly beneficial.

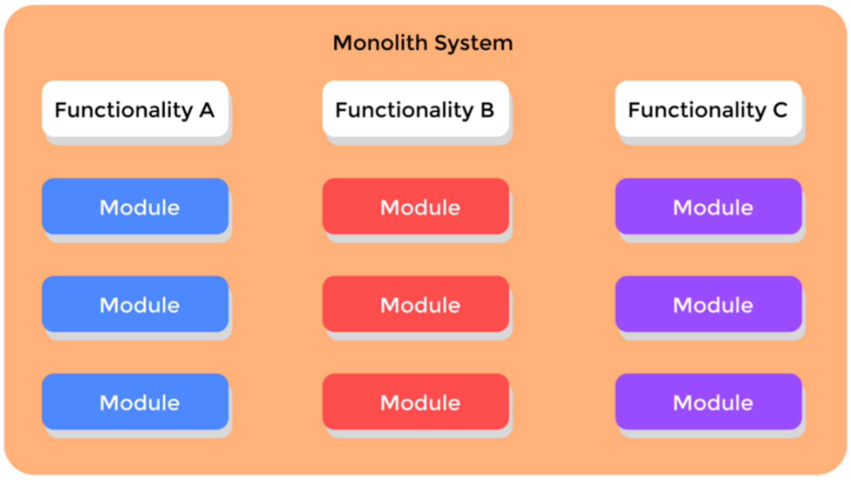

Figure 1.1 - Identification of Existing System Components

Review Components and Refactor

Once identified components should be reviewed, fully understood, and refactored to ensure there are no duplications in functionality. Microservices will only be responsible for one specific function so doing this ensures these will be easier to design.

Identify Component Dependencies

Due to the tightly coupled nature of Monoliths, there will be components which depend on one another, it will be key to identify these to determine which microservices will need to communicate with one another for the new system.

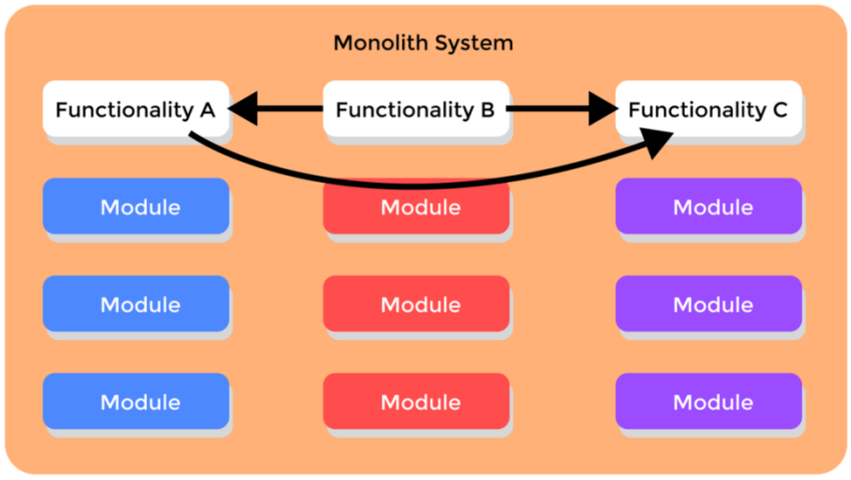

Figure 1.2 - Component Dependencies

Grouping Components

Once the dependencies have been identified they should then be grouped by functionality to form a homogenous group.

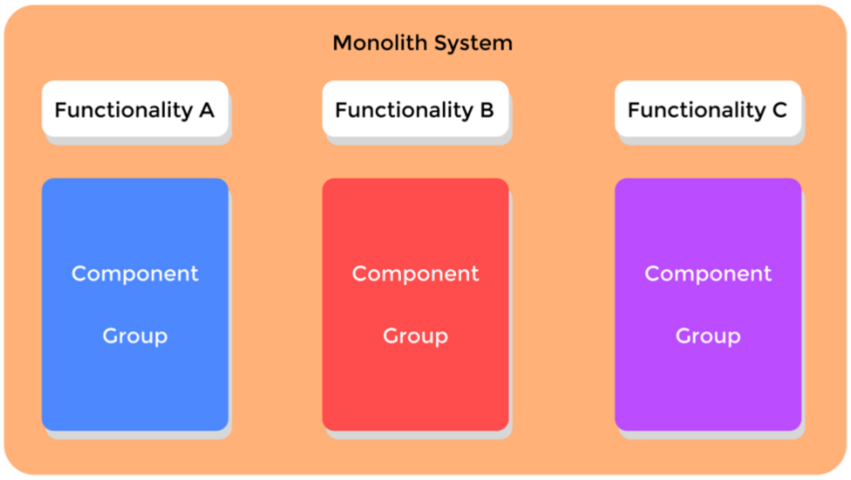

Figure 1.3 - Grouping Components

Create an API/Proxy

This API/proxy will form the only way users can interact with the system. During the transition, this is how the users will interact with the Monolithic system essentially just rerouting data through this back to the system. However, as you begin to develop the microservices system you can redirect the traffic. (Google Cloud Tech, 2019)

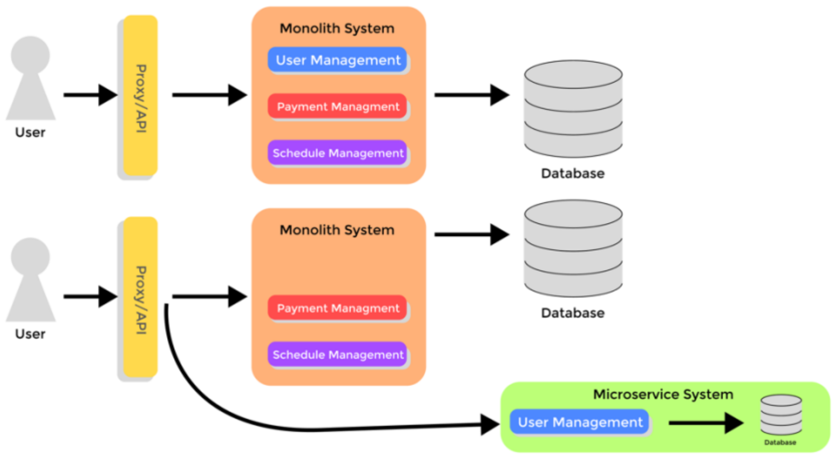

Figure 1.4 - Use of the Proxy Interface Before and After Microservice has been developed

Migrate Groups to Macroservices

This phase is optional and provides an intermediate step between the Monolith and Microservices. Due to the interconnected nature of a Monolith system problems can arise when trying to perform direct migration to microservices. Breaking the existing Monolith down into Macroservices (which are like smaller monoliths) these components involve breaking down the system into smaller components but these will still share the same database and access to data as before.

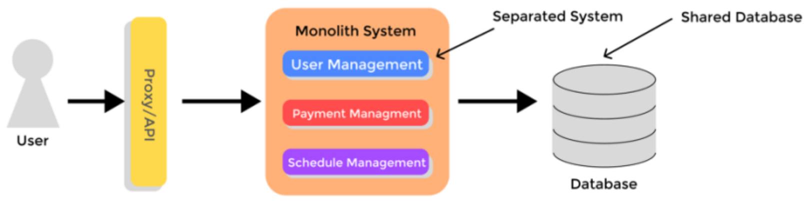

Figure 1.5 - Visual representation of Macroservices in a Monolith

Microservice Migration

This is the process now where services are migrated from the Monolith to Microservices. Each Microservice will have its own database and will be slowly integrated through the user of the proxy/API discussed previously.

Integration and Testing

This phase will be carried out with the previous phase and will ensure that not only does the Microservice function independently as expected but integrates both with the Monolith during the transition and with the other Microservices as they are developed and deployed.

Once the previously outlined steps are complete plans can then be made to decommission the now obsolete Monolith system.

WHY SWITCH?

Microservices offer several benefits when compared to Monoliths that should be considered before deciding to switch.

Robustness

One of the key downfalls of a Monolith system is the inherent single point of failure which comes with the coupled nature. Microservices does not suffer from such a flaw as each of the services is independent and an outage in one will not take down the entire system. This can help to save businesses a significant number of losses as other business functions can continue during an outage.

Faster Development and Deployment

With the decoupled nature of Microservices, the ability for teams to work independently on different services at the same time is enhanced. Since there is no worry that changes in one service will directly affect other services can be developed, tested, and deployed independently. Fortunesoft reported an average increase of 29% and 26% in faster time to market and staff productivity because of Microservice usage (Fortunesoft, 2023).

Scalability

Service and database independence allows for scalability to occur locally rather than system-wide. This can help to provide significant savings if a particular service is experiencing high traffic and needs to be scaled but other components do not. Research conducted by the IEEE showed a consistent reduction in costs as the system scaled sometimes as much as 50% (Villamizar et al., 2016).

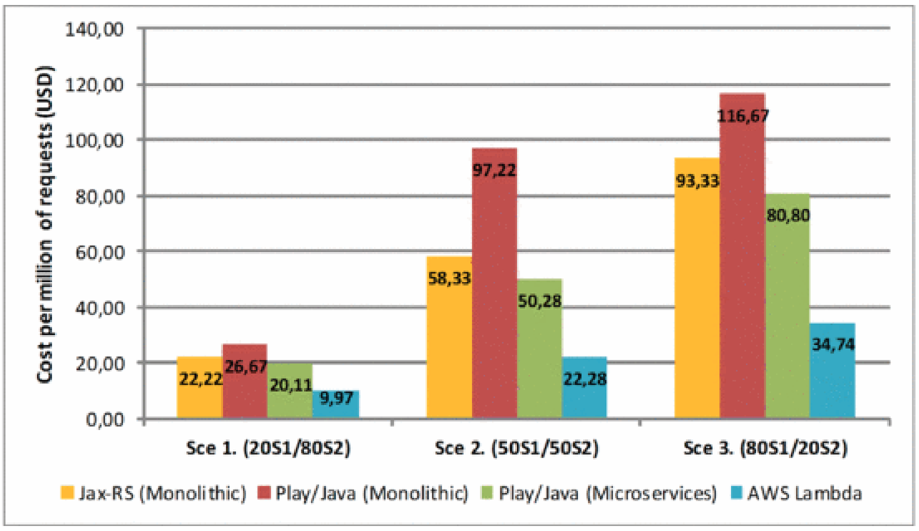

Figure 2.1- Cost comparison of the three architectures per million of requests (Villamizar et al., 2016)

Maintainability

Another benefit of the decoupling, inherent to Microservices, is that each of the services can be easily maintained and updated without causing issues in other components. It is also much easier to locate and isolate bugs as the services are sufficiently smaller in size. This can help to cut down significantly on both time and resources for location and fixing of software bugs throughout the software’s life.

Flexibility

Microservices give developers the ability to use the right tools for the job for each of the services using different languages and methodologies to make the most efficient system. This is something which can’t be accomplished in a Monolith. This allows enhanced flexibility to adopt new technologies. A great example of this is AI. If you wanted to integrate this into your Monolith and weren’t already using something well suited such as Python you might experience inefficiencies in the system but with Microservices this can be integrated much easier.

WHY NOT YET?

Whilst it might sound appealing to switch your existing Monolith to a Microservices architecture some drawbacks should be fully considered.

Investment

Transitioning from a Monolith to Microservices can be a complicated and cost-intensive process as outlined earlier in this document. In some cases, this cost might outweigh the numerous benefits that Microservices can provide to a business. In recent news, Amazon switched from Microservices for one of its prime video tools to a Monolith and saw a 90% cost reduction (Kolny, 2023), which indicates that Microservices might not always prove to be the most cost-effective solution.

System Latency

As the number of requests scales up there can be an increased latency in a Microservices system compared to Monolith as the communication between components is encapsulated in one codebase. This can result in an impact on system performance, however for simpler workloads average latency can be comparable to a similar Monolithic equivalent.

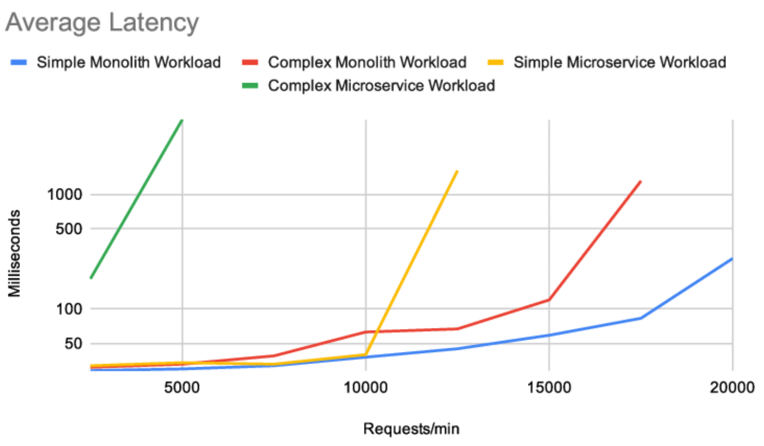

Figure 0.1 - Average Latency for the Different Workloads in Experiment 1 (Bjørndal et al., 2020)

Complexity

Whilst breaking down the system can provide some benefits by reducing complexity, complexity can still exist in a Microservice system with complex communication between services. This will require careful planning from a development team which could result in extra resources and costs spent on the product.

FINAL THOUGHTS

Considering the points which have been made in this document I believe that the transition would prove to be beneficial in the long run when compared to staying put with the existing Monolith.

Microservices can offer several benefits as a system architecture in terms of scalability, flexibility, and efficiency throughout the development process and after the system has been initially deployed.

Making the system more maintainable and robust will surely provide long-term benefits over the system life cycle which would outweigh the initial drawbacks of investment and communication complexity which can both be mitigated through careful planning from the development team.

References

Bjørndal, N., Bucchiarone, A., Mazzara, M., Dragoni, N. and Dustdar, S. (2020). Migration from monolith to microservices : Benchmarking a case study. doi:https://doi.org/10.13140/RG.2.2.27715.14883.

Fortunesoft (2023). *How Microservices are revolutionizing the IT sector?* [online] Fortunesoftit. Available at: https://www.fortunesoftit.com/how-microservices-are-revolutionizing-the-it/#:~:text=Companies%20achieved%2030%25%20greater%20customer [Accessed 25 Feb. 2024].

Frye, B. (2020). 8 Steps for Migrating Existing Applications to Microservices. *Carnegie Mellon University - SEI Blog*. Available at: https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/ [Accessed 25 Feb. 2024].

Google Cloud Tech (2019). *Migrating a Monolithic Application to Microservices (Cloud Next '19)*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=_azoxefUs_Y&ab_channel=GoogleCloudTech [Accessed 25 Feb. 2024].

Kolny, M. (2023). *Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%*. [online] Prime Video Tech. Available at: https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A. and Lang, M. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. pp.179–182. doi:https://doi.org/10.1109/CCGrid.2016.37.