UNIVERSITY OF WATERLOO
Faculty of Mathematics

A LOOK INTO MONOLITHIC VS. MICROSERVICES SOFTWARE ARCHITECTURE

PD11: Process For Technical Report Writing
Waterloo, Ontario

Prepared by
Raphael Koh
2B Computer Science
ID 20509965
November 27, 2017

Table of Contents

List of Tables and Figures

Summary

This report defines two types of software design, namely, the monolithic architecture and microservices architecture. The advantages and disadvantages of both are explored and, in doing so, the report provides a clearer understanding of why many larger companies are making a move from a monolithic software architecture to the microservices architecture. The report analyses Loblaw Digital's current use of a monolithic structure in its code base and the drawbacks it has on the company and its development cycle. Loblaw Digital is scaling rapidly in terms of both its code base and its team size. Thus, its monolithic code base does not scale well and a restructure to a microservices architecture is needed.

1.0 Introduction

Software architecture plays a major role in the success of a technology company. The choice and design of the company's software architecture affects the scalability of the code base and the productivity of its developers. This report focuses on 2 different types of software architecture, namely the monolithic architecture and microservices architectures. The monolithic architecture is used at Loblaw Digital and, as a result, developers face long application start up times, the code base is hard to understand for new developers on the team, and continuous deployment is difficult. Although many companies use the monolithic software architecture for their code base, there is an increasing number of companies, such as Netflix, Amazon, and PayPal (Bhatia, 2015), who are adopting the microservices architecture for various reasons explored in this report. This report aims to examine the differences between the monolithic and microservices architecture by comparing pros and cons of each type of architecture in relation to the a tech company's development productivity. We discuss how a move to the microservices architecture could resolve the problems faced by growing companies such as Loblaw Digital.

2.0 Analysis

Software architecture is the set of decisions that define the organization of a software system to achieve a solution that meets all of the technical and operational requirements (Microsoft, 2009). The initial choice of software architecture for a company has a big impact on the scalability of the code base. Failing to consider certain requirements by choosing a certain architecture can lead to long-term consequences such as a failure to meet business requirements or a difficult deployment process. The goal of software architecture is to create a software system that is able to meet both technical and business requirements and is flexible enough to be able to adapt to changes in requirements over time. The monolithic and microservices architectures are two types of software architectures that are prevalent among technology companies today.

2.1 Monolithic Architecture

A monolithic software architecture is one in which the components of the software system exist within the same code base, as seen in Figure 1. All the logic and computation occurs in one large code base (server-side application) which serves as an interface between the client, or end user, and the database.

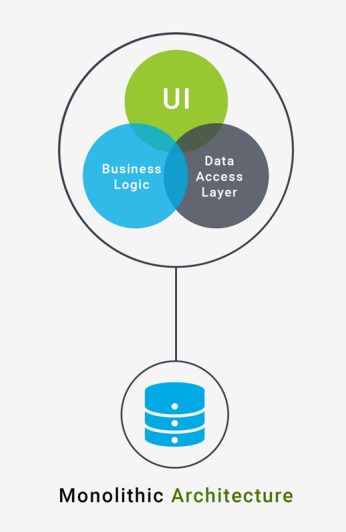

Figure 1: Diagram of a monolithic architecture (Awasthi, 2017)

2.1.1 Advantages of Monolithic Architecture

Many companies adopt the monolithic architecture as it is very simple to structure and develop. Most IDEs (Integrated Development Environments), like Eclipse and NetBeans, are geared towards the development of monolithic applications (Richardson, 2017) as older IDEs were made for corporate consumption and enterprise software. There are tools, such as the Selenium framework which allows for end-to-end testing for web applications (Kharenko, 2015), that makes testing easier for monolithic applications. Monolithic software are also simple to deploy as you just have to build the application and upload it to the server, as compared to having to upload and manage multiple services with the microservices architecture. It is simple to scale a monolithic app by running copies of the application behind a load balancer. The load bal-

ancer splits up the workload of the application among multiple servers, increasing the amount of allowable traffic.

The monolithic architecture is preferred for smaller software systems as it makes it simple to design and deploy. This allows companies to iterate over their software more frequently and ship their product to consumers faster. However, as the code base grows, the monolithic architecture design becomes an increasing disadvantage.

2.1.2 Disadvantages of Monolithic Architecture

Most of a monolithic system's functionality and implementation are interwoven and have huge dependencies on one another. Its high coupling is the major disadvantage that a monolithic architecture has. Good software design advocates low coupling to ensure that even if one component goes down, the other components can still function independently without crashing the entire system.

One major disadvantage of a monolithic architecture is the inverse relationship between the size of the code base and the developer's productivity. As the code base grows, it gets more complex to understand and code readability decreases. This translates to longer onboarding times for new developers to learn the code base. It also takes developers a longer time to understand the issue they are tackling and figure out how to correctly implement the solution. Larger applications are

also victims to slower start up times. These factors lead to a slower development cycle and a gradual decrease in code quality.

Continuous deployment is also increasingly difficult as the code base grows. A monolithic architecture has low modularity as each of its components are highly interconnected. This creates high coupling which is undesirable in software design because a potential fix in one component could inadvertently create a bug in a dependent component. Furthermore, a small update in one component requires redeployment of the entire system (Richardson, 2017). The possible introduction of new errors due to high coupling further increases the risk associated with redeployment, also known as "dependency hell" (Merkel, 2014). This potential for unintended mistakes and errors make the development process harder and thus makes continuous deployment more difficult.

Lastly, the monolithic architecture makes it costly to adopt new technologies. For example, if a company wanted to change the query language its database uses, the company would have to rewrite all its database queries and restructure its database schema depending on the difference between the old and the new query languages. Due to the high coupling of the monolithic application, the migration to a different language is not an easy feat. A slight change in code could have big repercussions throughout the rest of the code base. Thus, it is difficult and costly to adopt new technologies in a monolithic architecture as

the entire application will be affected (Kharenko, 2015).

2.2 Microservices Architecture

The microservices architecture tackles the main problems that monolithic architecture software has. It divides your application into smaller components that work independently from one another (see Figure 2). Each service implements one functionality, is independent from other services, can be deployed independently, and can use a different stack of technology (Bhatia, 2016). If one service goes down, the rest of the application isn't affected. Each service might have its own database schema which could result in duplication of data (Kharenko, 2015). The goal of a microservice architecture is to achieve low coupling. One way it enforces low coupling is allowing the services to communicate between themselves only through network calls, or application programming interfaces (APIs) (Newman, 2015). The API is like a recipe or a set of instructions that allows other services to know the exact data being transferred. The computation that occurs within each service behind the API is opaque to the other services and allows the service to function like a black box. This provides the system with the flexibility to make changes or even revamp a component without affecting other components.

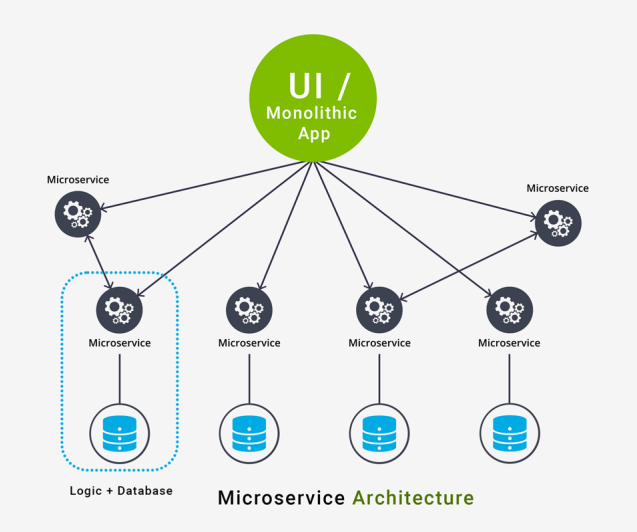

Figure 2: Diagram of a monolithic architecture (Awasthi, 2017)

2.2.1 Advantages of Microservices Architecture

The low coupling, modularity, and flexibility that the microservices architecture gives a system is beneficial for companies looking to scale its software. Many large tech companies, like Netflix and Amazon, have made the decision to employ the microservices architecture in their code base due to its scalability.

One of the big advantages of the microservices architecture is the ease of continuous deployment. Applications that are architected with a microservices framework are quick to deploy because any change to a single service can be deployed independently of the other systems. This facilitates continuous deployment for complex applications. If there is a problem, the change can be rolled back and the error tracked easily

due to the small scale of the individual service (Newman, 2015). Development is also made significantly easier because each service can be developed independently by a team dedicated to that service. Changes can be made to the service without fear of breaking another service. Furthermore, having a team specialize in a service improves their productivity as it limits their required scope or breadth of knowledge. This allows them to gain better depth of knowledge and be subject matter experts of their service. An additional benefit is minimizing the team size working on each service, thus optimizing the company's overall productivity (Newman, 2015).

Developers are more productive because the application's code base is segmented up into individual services, making it easier to understand. Developers can focus on the particular services relevant to their work without having to worry about the inner functionalities of other services. This greatly reduces the onboarding time required and gives way to increased scaling velocity.

The microservices architecture allows for technology diversity and flexibility (Fowler, 2017). Different languages, frameworks, and tools can be integrated seamlessly together into the application. Changing these technology for others can be done easily without affecting the rest of the application because the services are not highly coupled and only interact via an API interface. As long as the API interface stays constant or

is updated throughout the migration, the application as a whole would still maintain invariance. This allows the companies the freedom to choose a technology that fits the current purpose and not be bound to the choices made at the beginning of the project. This tackles a fundamental problem with the monolithic architecture and legacy code bases that cost companies too much time and money to replace.

2.2.2 Disadvantages of Microservices Architecture

Even though the microservices architecture solves many of the problems raised by the monolithic architecture, it comes with inherent problems. Although it is helpful for scaling large applications, it complicates smaller code bases by adding unnecessary abstractions and interactions between components.

The microservices architecture introduces the complexity of a distributed system. A distributed system is a network of components that are connected via a distribution middleware or API Gateway (Techopedia, 2017). A distributed system can be costly to manage because each interaction through the API Gateway has to be checked for potential errors. Distributed systems also face the problem of updating multiple databases. Since the application's database is partitioned according to the individual services, transactions performed by the application have to update multiple databases. This complicates the code and creates a

wider surface area for potential errors. If one database or part of the transaction goes down, the transaction would not function as it should. A solution to this is the eventual consistency approach. Whenever a service updates its data, it publishes an event that other services can subscribe to. Once notified of the change, the other services updates its data to match the published event (Richardson, 2017). This allows for all services to be "eventually consistent", however, this approach is challenging to develop. The application and its components are also harder to test. Having a distributed system of services makes testing a particular system more complicated as its dependencies have to be run before it can be tested.

Deployment is more difficult in microservice applications. Firstly, changes are difficult to implement across multiple services. In a monolithic architecture, you can integrate and deploy the changes after modifying the appropriate component. On the other hand, changes for a microservices architecture that affect multiple services have to be coordinated between the components. This requires communication across teams to figure out the cross-system deployment and might complicate or slow down the development process. Manual deployment for microservices is unfeasible. Each microservice has to be configured, deployed, scaled, and monitored. This makes a manual approach almost impossible and requires a high level of automation (Richardson, 2017). Even though continuous deployment is easier, coming up with a plan to roll out the

changes and automate the testing makes it more complex.

2.3 Loblaw Digital And Software Architecture

Loblaw Digital uses the monolithic architecture in its code base. This was a decision made early on in its early days as a new off-shoot project from Loblaws. Loblaw Digital began its digital project back in 2013 with a small team size of 3 developers and 1 business (C. Qu, personal communication, November 14, 2017). It operated as a startup with a huge corporate financial backing and made technical decisions fitted for a tech startup. Specifically, the team made the decision to use the Java programming language with Spring Boot integration for web development and SAP Hybris for content management. Java, along with other older programming languages like C and C++, tends towards a monolithic software architecture.

Today, Loblaw Digital has over 80 developers and manages the digital platform of 5 businesses (C. Qu, personal communication, November 14, 2017), which includes its grocery stores, like Loblaws and Canadian Superstore, along with other subsidiaries like Shoppers Drug Mart and Joe Fresh. This results in an enormous code base that is responsible for providing the functionalities for the websites of each store. It is quite evident that this rapidly growing code base is detrimental to the company's growth because the monolithic architecture is not suited to a software system of this scale.

Loblaw Digital's code base has grown by 8 to 10 times since the first year of development (C. Qu, personal communication, November 14, 2017). The majority of the company's massive code base is written in Java and is packaged into a single application that is deployed to the server every 2 weeks. Having a monolithic architecture makes it easier for developers to run the whole application on their system, but requires a lot of setup time. Setting up the necessary environment for development on the developer's local machine can take up to 3 hours! This eats up a lot of development time and is costly to the team's productivity and the company's profits.

Every minor code change in one part of the system requires days of testing the internal Automation Testing team. The code first undergoes unit tests developed by the software developer, integration tests by the Quality Assurance engineer, followed by regression tests leading up to live deployment of the code. All this testing has to be done over the whole application to ensure that the code change does not affect any other part of the system. Live redeployments happen every two weeks at Loblaw Digital. This is really slow when compared with Amazon's 11.6 second release cycle, as of 2011 (Jenkins, 2011). Continuous deployment and a fast development cycle are desirable traits that tech companies want to have as it allows for their product to reach their customer faster. However, this is difficult to achieve because the monolithic architecture does not lend well to frequent continuous deployment.

The size and complexity of a monolithic architecture makes it tough for new Loblaw Digital employees to learn the code base. New developers have to understand the structure of the monolithic code base and how each component interacts with one another before being able to implement correct solutions to the code. Loblaw Digital hires about 5 new employees per month (C. Qu, personal communication, November 14, 2017) and will continue to increase its aggressive hiring. The monolithic architecture will be costly for the company when new developers have to take time to understand the code before starting development.

2.3.1 Adopting The Microservices Architecture

As Loblaw Digital looks to scale, a shift to a microservices architecture is required. However, the shift from a monolithic architecture to a microservices architecture is not easy. Migrating the code base to a different software architecture is akin to "changing the engine of a running truck" (C. Qu, personal communication, November 14, 2017). The team must ensure that the product on the consumer's side interfaces as per normal while the migration is happening.

The change of software architecture affects other parts of the orga-

nization as well, and not just the tech division. There will be business factors that have to be considered. For example, large corporate businesses favour enterprise software. Enterprise software provides all the relevant tools the business needs rather than requiring the company to build its framework from the ground up, costing time and money. Loblaw Digital is no exception; the code base is built on top of the monolithic hybris. Thus, the company needs to reconcile these two contrasting architectures and decide on one that results in a net gain.

It takes a lot of time and effort to restructure not only the code base, but also the teams within the company. The company will have to re-evaluate the current team organization and perform structural changes to minimize team sizes and allow each team to focus on a microservice. A web messaging protocol, or API, will have to be clearly established to enable the teams and their software to effectively communicate with one another. The team's automation process would have to be improved to handle the set up and deployment of the entire application and its various microservices.

2.3.2 The Future Of Loblaw Digital

Loblaw Digital already has a plan set in motion for the migration of its code base away from a monolithic architecture. However, the company will not be implementing a true microservices architecture due to

its monolithic hybris core. Instead, they will follow a broader Service-Oriented Architecture (or SOA) by looking at what components of its code base can be extracted from hybris and be placed within its own microservices. In essence, the code base will comprise of multiple services with one monolithic core service. As shown in Figure 3, the code base will be divided up into smaller services represented by the grey box. Each service operates separately from the rest and communicates with each other via API calls. If one of these services malfunctions, the system as a whole can still run independently. Teams will be designed such that their scope will be within one service to optimize domain-specific knowledge. Code changes within one service will not be able to change code in other services; only valid data changes can be done through the APIs.

Each service has its own database. This is in line with the microservices architecture and restricts database access to the enclosing service. It should be noted that there can be duplicated database types across multiple services (i.e. product data in Figure 3). Product data is required across multiple services, but due to the microservices architecture, each of these services has to have its own database schema of product data. However, this is not necessarily undesirable; each service uses different properties or data of the products. For example, the "Product Data/Metadata Service" service uses properties like category and price, while the "Ecommerce Services" service uses properties like

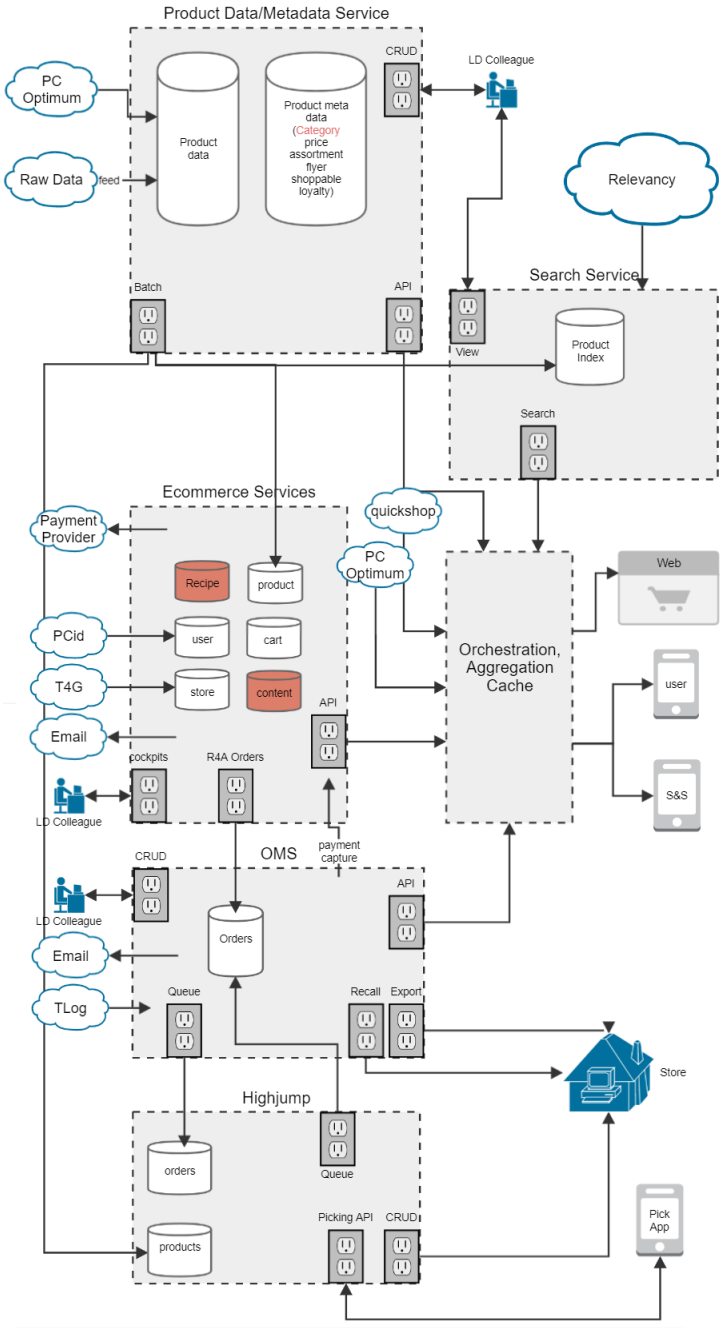

Figure 3: Diagram of Loblaw Digital's plan for a service-oriented system (Qu, 2017)

product ID. This is an example of Domain-Driven Design, which states that types can span over multiple services, but the type can take on a different meaning based on the domain or service it is used in. This is efficient because the application is not retrieving useless data when querying for an item. The database only holds the data relevant to the enclosing domain or service.

All in all, Loblaw Digital will see a drastic shift over time in the way its software development process works as it transitions from a monolithic to microservices architecture. The company has already migrated a few of the services over from the original code base and plans to finish the migration by the end of 2018.

3.0 Conclusions

Loblaw Digital's monolithic architecture approach definitely worked well when it first started developing its software due to the small code base. However, as the company is looking to scale aggressively, it needs to rethink its approach and look to restructure and rearchitect its software from monolithic to microservices architecture.

4.0 Recommendations

Loblaw Digital is beginning to see some minor integrations of the microservices architecture into its code base. A few smaller and newer teams are taking a shot at structuring their code with a microservices architecture. This gives them the autonomy to develop code independently and to have a release cycle that is separated from the rest of the company. In the coming years, more segments of Loblaw Digital's code base will be rewritten and restructured using a microservices architecture and, in the end, there will be a code base that allows the company to ship its code under 10 seconds!

References

1 Awasthi, M. (2017, May 12). Microservice vs Monolithic Architecture [Web log post]. Retrieved November 13, 2017, from http://nodexperts.com/blog/microservice-vs-monolithic/

2 Bhatia, A. S. (2016, February 03). Why Are Organisations Moving to Microservices? Retrieved October 31, 2017, from http://opensourceforu.com/2015/11/why-are-organisations-moving-to-microservices/

3 Fowler, M. (2017, October 18). Microservices Guide. Retrieved October 31, 2017, from https://martinfowler.com/microservices/

4 Jenkins, J. [O'Reilly]. (2011, June 20). Velocity 2011: Jon Jenkins, "Velocity Culture" [Video file]. Retrieved from https://www.youtube.com/watch?v=dxk8b9rSKOo

5 Kharenko, A. (2015, October 9). Monolithic vs. Microservices Architecture [Web log post]. Retrieved October 31, 2017, from https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

6 Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal. Retrieved October 31, 2017, from http://www.linuxjournal.com/content/docker-

lightweight-linux-containers-consistent-development-and-deployment

7 Microsoft. (2009, October). Microsoft Application Architecture Guide, 2nd Edition. Retrieved October 31, 2017, from https://msdn.microsoft.com/en-ca/library/ee658098.aspx

8 Newman, S. (2015). Building Microservices. Sebastopol, CA: O'Reilly Media, Inc. Retrieved October 31, 2017, from https://cdn.baker.com/documents/Building%20Microservices/9781491950357_sampler.pdf.

9 Techopedia. (n.d.). What is a Distributed System? - Definition from Techopedia. Retrieved November 27, 2017, from https://www.techopedia.com/definition/18909/distributed-system

10 Qu, C. (2017, November 8). Services, Components, Architecture things. Retrieved November 14, 2017, from https://loblaw.atlassian.net/wiki/spaces/ chao.qu/pages/119703024/ Services Components Architecture things

11 Richardson, C. (2017). Microservices Pattern: Monolithic Architecture pattern. Retrieved October 31, 2017, from http://microservices.io/patterns/monolithic.html

Acknowledgements

I would like to thank Kristian Reyes, my manager at Loblaw Digital, for letting me bounce ideas off him in spite of his busy schedule and his help with the report even after work hours.

I would like to thank Chao Qu, my supervisor at Loblaw Digital, for answering my questions and giving me a really detailed explanation of Loblaw Digital's plans towards a service-oriented architecture.

I would like to thank Clayton Halim and Liam Horne, who helped me structure my work term report.

Lastly, I would like to thank Michelle, my best friend and girlfriend for being patient with me while I procrastinate and stay up late working on this report.