Optimizing the Mid-Size Microservice Architecture Ecosystem

Nikoloz Katsitadze
University of Georgia
Tbilisi, Georgia
e-mail: n.katsitadze@ug.edu.ge

Zurab Chachava
University of Georgia
Tbilisi, Georgia
e-mail: chachavazuka@gmail.com

https://doi.org/10.62343/csit.2024.6

***Abstract* — Starting a new IT project requires careful planning, including understanding the business, analyzing data, and making smart decisions about how to build the project. While many are drawn to using microservices for their scalability, they can be complex, costly, and time-consuming. On the other hand, traditional monolithic architectures are simpler but can be hard to scale and maintain as the project grows.**

A good compromise is the modular monolith approach. It breaks the project into smaller parts with clear boundaries, making it easier to manage and reuse code. By using a simple communication method between these parts, called RPC-like communication, we can keep things efficient and avoid complicating the developers' work with networking details. If needed, we can later transition smoothly from a modular monolith to microservices, saving time and avoiding many bugs.

Overall, choosing practical architectural approaches like modular monoliths helps keep projects straightforward while preparing them for future growth and upkeep.

***Keywords* — Microservice architecture, optimization, monolith architecture.**

I. INTRODUCTION

Planning modern IT projects requires hard work. Understanding business domain, making analytics, choosing relevant technical project architecture are the key concepts of IT project management. The work becomes harder when the team is trying to find the best architecture solutions while dealing with business problems. Most of the time modern architect guru's advice to use microservice architecture, but is it really a good idea?

II. WHY MICROSERVICES ARE NOT THE BEST OPTION AT THE BEGINNING.

Even though microservices are one of the most scalable architectural solutions, but as always great power comes with great responsibilities. While it is a practical choice, it has many disadvantages as well: Technical complexity, high cost, time intensive.

When talking about technical complexity, it refers to following problems:

- Complexity in communication: microservices should communicate to each other, which introduces complex APIs, network congestion, latencies and so on;
- Data Management: managing data consistency across services and handling transactions among these processes can be challenging;
- Service Discovery: Keeping track of numerous services and instances for service discovery can be complex in a microservices architecture;
- Security: Ensuring secure communication between services and managing the increased attack surface due to exposed APIs requires more work;
- Monitoring and Troubleshooting: Monitoring and troubleshooting a microservices-based application can be more complex than a monolithic one due to the different programming languages and environments used for each service.

According to these problems, using microservice architecture from the beginning of the project, especially in startups, can cause fatal issues and most probably the project will not see light of the sun.

"You shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile." - Martin Fowler.

III. PROBLEM WHILE WORKING WITH MONOLITH

Another common architectural approach can be "good old" monolith. monolith refers to a software application that is built and deployed as a single unit. Therefore, all the components of the application, including the user interface, business logic, and data storage, are tightly integrated and run in a single process. This solution is not as bad as modern developers say. It gives us simplicity because all components are part of a single codebase. Most of the time, it is more performant due to avoiding additional network communication. Debugging and monitoring are also simplified when using single process.

Nothing is perfect, so that monolithic architecture also has disadvantages:

- Scalability: Monolithic applications can become difficult to scale, as the entire application needs to be scaled, regardless of the specific component that requires more resources.
- Deployment: Deploying a monolithic application can be time-consuming, as the entire application needs to be built, tested, and deployed as a single unit.
- Maintenance: As the application grows, it can become more difficult to maintain and update, as changes to one component can potentially affect other components, leading to unintended consequences.

At the end of the day, we get a big giant codebase, which hates changes, and any attempt causes big trouble and pain to the developers.

IV. THE GOLDEN EDGE SOLUTION – MODULAR MONOLITH

A modular monolith is an architectural approach that divides an application's domain into smaller, more manageable components or modules, providing a balance between monolithic and microservices architectures.
The project is segmented, and each piece contains individual features and business logic. This kind of approach promotes high cohesion and low coupling. Modules in a modular monolith communicate through well-defined interfaces, which refers to lose coupling and independent development of functionalities.
Benefits:

- High Reusability: Logic encapsulation enables high reusability while maintaining data consistency and simple communication patterns.
- Simplicity: Easier to manage than multiple microservices, keeping infrastructural complexity and operational costs low.
- Easy to migrate modular monoliths can be separated without much effort. This benefit can be attractive to startups, who seek simplicity and future improvements.

Challenges:

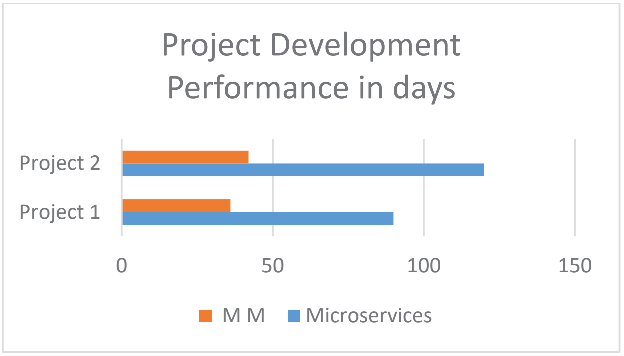
- Limited Technological Diversity: Modular monoliths restrict the use of diverse technologies and languages compared to microservices due to executing code within a single runtime.
- Scalability Constraints: Scaling a modular monolith can face limitations as the entire application needs to be scaled together, potentially leading to inefficient resource utilization.

V. MODULAR MONOLITH INTERNALS

It is simple to imagine how modules are defined in the project. Each module has its own project layers such as domain, application, infrastructure and so on. But what happens when communication between modules is required. There are several approaches, such as sending data through a service bus or declaring module interfaces.
According to my experience, using a service bus gives us a clear border between modules and transferring modular monolith into microservices requires less effort. But it mostly misses the main goal, which is to build a project simply. Service bus is an additional infrastructure unit, and it increases project complexity.
Using interface is easy and performant because direct function calls are always faster than network calls. Unfortunately, standard interface implementation still pairs modules tightly. But what about using the rpc approach? especially grpc which is well supported by Google. RPCs always look like a normal function, even thou they make network calls. What if we use RPC-like structure to call directly referenced modules. When using this approach developer is step ahead and if in the future the project is scaled, fake RPC functions will be replaced by real RPC calls.
It is fact, that transferring from fake RPCs to real RPCs is not straightforward, but this process is easy. According to my bank experience building projects for experimental products using modular monolith and RPC approach gives better development performance, because dev team members do not need to worry about sending messages, network failures and so on. The development time was mostly reduced about 30%-40%, which is pretty much attractive for businesses people and product managers.

VI. SPEED DIFFERENCE BETWEEN MONOLITH AND MODULAR MONOLITH WHILE SEPARATING INTO MICROSERVICES

Transferring into microservices became a smooth process. The transferring time was also reduced by 50%-70% and not many bugs were developed during this period. Modules are easily decomposed as a bunch of units. Units became docker containers and were deployed in the cloud.



Picture 1

The following chart displays the development performance of two real financial projects. According to the results, using modular monolith significally reduces the development time compared to microservices. This big difference is caused by avoiding additional infrastructural components.

VII. SUMMARISE

As reality has shown us, using the most advanced architectural approaches for newly started projects is not good decision. Choosing easy structures is the best way to build new digital products. But developers must be wise and plan the project like this so that in the future it will be easily maintainable scalable and reliable. Modular monoliths are one of the best options to achieve this goal.

REFERENCES

1. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. First edition. Martin Kleppmann, 2017
2. System Design Interview – An insider's guide volume 1 – An insider's guide. Alex Xu, 2020
3. System Design Interview – An insider's guide volume 2 – An insider's guide. Alex Xu, Sahn Lam 2022
4. Domain-Driven Design: Tackling Complexity in the Heart of Software 1st Edition. Eric Evans 2003
5. Building Microservices 1st Edition – Sam Newman 2015