Migrating Monoliths to Microservices: A Survey of Approaches

Eberhard Wolff[1]

innoQ Deutschland GmbH, Berlin, Germany
eberhard.wolff@innoq.com

Abstract

Usually microservices are not built on a greenfield. Instead, a monolith is migrated into a microservices architecture. This presentation shows goals and constraints of a migration strategy. A blueprint strategy is presented which is not limited to architecture but also covers infrastructure. Finally, alternative approaches with their advantages and disadvantages are discussed.

1 Introduction

The architecture of a software system must evolve to support functional and non-functional requirements. The shift to faster deployment make microservices a great new architecture approach. This raises the question of how an existing system can be migrated into a microservices architecture.

2 Goals and Constraints

The goals of the migration are important for choosing the best migration approach. A typical goal is to enable independent deployment of parts of the system, simplifying continuous delivery. Also, microservices are an important step to make the teams more independent and scale the development to a larger team.

However, there might be other goals. For example, microservices also provide much better stability and resilience. Of course that goal leads to a completely different approach.

Typical constraints for a migration include:

- The migration should be iterative. Doing the migration step-by-step mitigates the risk and leads to a faster pay-off.
- Usually the existing system is hard to understand and analyze. So ideally the migration should require as little knowledge about the existing system as possible.
- The reason for the migration is often the desire to build a new structure of the system and to use more modern technologies. That means it does not make a lot of sense to reuse the existing code and structure.

3 Blueprint Approach

The blueprint migration approach represent an approach to a migration that can serve as a starting point to build custom migration strategies. It consists of two parallel tasks: building the infrastructure and the actual migration.

The migration make use of Domain-driven Design's *Bounded Contexts*. The system is separated into bounded contexts. Then a bounded context is chosen and migrated into one or more microservice. The migration continues to migrate bounded contexts. This approach is iterative. Also it requires little understanding about the implementation details of the existing system: Bounded contexts can be identified by the functionalities and use cases. The newly implemented bounded contexts can use any kind of technology and do not depend on the existing structure. The microservice should include the UI, logic, and persistence for the bounded context. That ensure that a change just requires changes to one microservice even if it causes changes to UI, logic, and data structure aspects.

Parallel to the migration of the system, an infrastructure must be set up. In the long run, a large number of microservices must be operated. That requires a different infrastructure than deployment monoliths. However, for the first microservice the existing infrastructure might be sufficient. So the infrastructure should be set up stepwise. If there is just one microservice per bounded context, the number of microservices might be quite low and no different infrastructure might be needed.

Even though the term "blueprint approach" has been coined by the author of this paper [7], it is generally agreed that microservices should be build around bounded contexts and that the migration should be based on migrating to bounded contexts [5][6][1]. There are numerous case studies about migrations based on bounded contexts.

4 Alternative Approaches

Besides the blueprint approach, there are alternative approaches. These include for example:

Fit Organization Microservices must not be shared by more than one team. To develop a bounded context requires a cross functional team that can change UI, logic, and persistence. However, that might require a reorganization. The alternative is to accept the organizational constraints. The code each team works on is migrated into multiple microservices. These might implement a bounded context but that is not always the case.

Change by Extension This strategy closes the old system for changes. New functionalities must be implemented in new microservices. While this strategy is very easy to follow, it means that new microservices are created ad-hoc depending on the current changes. It is hard to ensure that the system will have a good architecture. It is also hard to ensure that all significant parts of the system will be migrated at some point.

Strangler *Strangler* [4] is a typical migration approach. The idea is to create new systems around the edges of the old system and 'strangle' the old system. Techniques include event interception: The new system taps into the stream of events the system works on. An alternative is asset capture: The new system works on specific assets, e.g. simple orders or specific customers. *Strangler* is a rather general description of a strategy that matches the *blueprint approach* as well as e.g. *change-by-extension*. Strangler is probably the most popular approach. But because it is such a generic term, this actually says little about the concrete strategy used in a project.

Other approaches include Front End Switch, UI, Backend, Change-by-Copy, Qutside-in Interface, Zombie, and Natural Death as defined by aim42 [3].

5 Conclusion and Outlook

Migration to microservices is a common scenario to establish a microservices architecture. This presentation shows several approaches for a migration, each with different benefits. It is therefore an great foundation for a custom migration strategy.

The approaches can be applied to improve an existing microservices architecture by adding new microservices, splitting existing microservices, and eliminating some microservices.

However, currently little is known about how popular the different approaches are and how likely they are to provide good results. First papers [2] are beginning to appear. Such statistics are a very interesting subject for further research. But because every system is different, such statistics might not be too useful to select a strategy for a case at hand. A specific strategy might be a good fit for one project but a very bad fit for other projects.

References

[1] Zhamak Dehghani. How to break a monolith into microservices. https://martinfowler.com/articles/break-monolith-into-microservices.html, 2018.

[2] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. Migrating towards microservice architectures: an industrial survey. In *2018 IEEE International Conference on Software Architecture, ICSA 2018, Seattle, USA, April 30 - May 4, 2018*, pages 29–38, May 2018.

[3] Gernot Starke et al. aim42 architecture improvement method. aim42.org, 2014–.

[4] Martin Fowler. Strangler application. https://www.martinfowler.com/bliki/StranglerApplication.html, 2014.

[5] Sam Newman. *Building microservices - Designing Fine-grained Systems*. O'Reilly, 2015.

[6] Eberhard Wolff. *Microservices: Flexible Software Architecture*. Pearson Education, 2016.

[7] Eberhard Wolff. Monolith to microservices — a comparision of strategies. https://speakerdeck.com/ewolff/monolith-to-microservices-a-comparison-of-strategies, 2019.