Microservices: A Taxonomy

Stefan Tilkov[1]

innoQ Deutschland GmbH, Monheim, Germany
stefan.tilkov@innoq.com

Abstract

A microservices services architecture has become a standard way to approach the modular design of large scale systems. But while there are some traits that most practitioners can agree on, such as independent deployment, choice in implementation details, polyglot persistence, and benefits such as isolation, better parallelization, and improved scalability, there are still vast differences between the diverse approaches taken in practice. In this talk, we will categorize different ways to approach the architectural style, and highlight differences, benefits, and downsides of various interpretations found in projects.

1 Microservices: A Taxomony

The most important criteria for the taxonomy are individual microservices size, the number of microservices resulting from the choice of size, and the communication patterns that emerge to connect them. Based on these aspects, we can derive at least four main categories of microservices architecture:

- The minimalistic style used when services are built with the goal to be as small as possible, labeled FaaS (Function as a Service)
- The style of small services, mostly communicating synchronously, forming larger units that collaborate to achieve a business goal, using approaches well known from service-oriented architecture, thus labeled Micro-SOA
- An approach where larger units are formed that are able to fulfill most of their tasks independently. They mostly rely on asynchronous communication and accept some redundancy as the price for increased autonomy. Synchronous communication is only used to fulfill service requests and asynchronous communication is used to support synchronization between services, labeled DDDD (for "distributed domain-driven design" due to the similarity with DDDs bounded contexts)
- A system built from larger-scala subsystems where each of them contains not only data handling and business logic, but also its own UI. Integration is done mostly on the frontend, and communication among services is minimized (called SCS for self-contained systems).

2 Comparison

For each of the different styles, it can be shown that they are suitable for different goals. The FaaS style works well when a strong dependency to some infrastructure component (such as a cloud vendors serverless platform) is acceptable and the main goal is to build "glue code" to connect the vendors services. For the Micro-SOA style, the most important benefit is runtime scalability, whereas the DDDD approach is well-suited when it development-time scalability of

teams is at the center of attention. Due to its embracing the UI aspect, the SCS approach is very well suited to systems where modularity is required in the frontend parts of a system to a similar degree as in the backend. The presentation will thus show that no single solution is the best for all cirumstances, and highlight how one might approach selecting the one that works best.

3 Conclusion

While the categorization that will be presented has been successfully used in practice, it is most definitely not complete. It can serve as a starting point to categorize more proven approaches, ideally assigning them clear and unambiguous names, helping to clarify the pros and cons of different architectural choices.