Microservices for Scalability

[Keynote Talk Abstract]

Wilhelm Hasselbring
Software Engineering Group, Kiel University, D-24098 Kiel, Germany
hasselbring@email.uni-kiel.de

ABSTRACT

Microservice architectures provide small services that may be deployed and scaled independently of each other, and may employ different middleware stacks for their implementation. Microservice architectures emphasize transaction-less coordination between services, with explicit acceptance of eventual consistency. Polyglott persistence in this context means that the individual microservices may employ multiple data storage technologies. Microservice architectures are “cloud native” allowing for automated and rapid elasticity. Fault-tolerance mechanisms achieve that failures of individual microservices do not affect other services thanks to container isolation. Since services can fail at any time, it is important to be able to detect the failures quickly and, if possible, automatically restore services. Essential for success in such a setting is advanced monitoring.

In this keynote, I discuss how microservices support scalability for both, runtime performance and development performance, via polyglott persistence, eventual consistency, loose coupling, open source frameworks, and continuous monitoring for elastic capacity management.

CSC Concepts

• Software and its engineering → Software architectures; Software performance;

Keywords

Microservices; Scalability; Monitoring

Monoliths vs. Microservices

Traditionally, information system integration [16] and enterprise application integration [2] aim at achieving high (database) integrity among heterogeneous information sources [19, 25, 26]. Federated database systems achieve high integrity via tight coupling on the schema level [20], preferably based on standards [17]. For migration and modernization [27] of (legacy) monolithic information systems, an essential design decision is how to keep old and new databases consistent [21], particularly when migrating to the cloud [5, 14, 15]. However, a great challenge with tightly integrated databases is the inherently limited horizontal scalability.

Microservice architectures intend to overcome the limited scalability of monolithic architectures. Microservices are built around business capabilities and take a full-stack implementation of software for that business area. In particular, microservices prefer letting each service manage its own database, even with different database management systems (polyglott persistence with eventual consistency). Besides data, code should not be shared among microservices to avoid dependencies; only reuse of framework code as open source software is recommended [22]. The trade-off between many small microservices and a few more coarse grained services must be considered in microservice architectures, as in any other component and system design activities [18]. To achieve an appropriate granularity, we propose a vertical decomposition along business services.

Non-functional attributes, such as scalability and fault tolerance for high availability, are addressed by microservice architectures. A consequence of using microservices as components is that applications need to be designed such that they can tolerate the failure of individual services. Since services can fail at any time, it is important to be able to detect the failures quickly and, if possible, automatically restore services. Microservice applications put a lot of emphasis on real-time monitoring of the application, checking both technical metrics (e.g. how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received). Monitoring can provide an early warning system of something going wrong that triggers development teams to follow up. Besides Kieker [29], our ExplorViz approach [13] provides live visualization for large software landscapes introducing three hierarchical abstractions [10]. Live visualization with ExplorViz is scalable [6] and elastic in cloud environments [28]. Monitoring may provide runtime models [23] for system comprehension [9], trace visualization [4], architecture conformance checks [11], and a landscape control center [12] with performance anomaly detection [3, 24]. New perspectives on employing virtual reality [8] and physical models [7] are further explored. Regression benchmarking [31] should be integrated into continuous integration setups [30] of microservices. Microservices leverage techniques such as continuous integration and continuous deployment to promote DevOps [1].
1. REFERENCES


