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 mircroservices 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 mircoservices 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.

CCS 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 mircoservices. Microservices leverage techniques such as continuous integration and continuous deployment to promote DevOps [1].

ICPE'16 March 12-18, 2016, Delft, Netherlands
ACM ISBN 978-1-4503-4080-9/16/03.
DOI: http://dx.doi.org/10.1145/2851553.2858659

1. REFERENCES

[1] A. Brunnert et al. Performance-oriented DevOps: A Research Agenda. Technical report, SPEC Research Group, Aug. 2015.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch. *Enterprise Application Integration.* Spektrum Akademischer Verlag, 2005.

[3] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In *Proceedings of ICAC 2011*, pages 197–200. ACM, 2011.

[4] F. Fittkau, S. Finke, W. Hasselbring, and J. Waller. Comparing trace visualizations for program comprehension through controlled experiments. In *Proceedings of ICPC 2015*, pages 266–276. IEEE, 2015.

[5] F. Fittkau, S. Frey, and W. Hasselbring. CDOSim: Simulating cloud deployment options for software migration support. In *Proceedings of MESOCA 2012*, pages 37–46. IEEE, Sept. 2012.

[6] F. Fittkau and W. Hasselbring. Elastic application-level monitoring for large software landscapes in the cloud. In *Proceedings of ESOCC 2015*. Springer, Sept. 2015.

[7] F. Fittkau, E. Koppenhagen, and W. Hasselbring. Research Perspective on Supporting Software Engineering via Physical 3D Models. In *Proceedings of VISSOFT 2015*, pages 125–129, 2015.

[8] F. Fittkau, A. Krause, and W. Hasselbring. Exploring software cities in virtual reality. In *Proceedings of VISSOFT 2015*, pages 130–134, 2015.

[9] F. Fittkau, A. Krause, and W. Hasselbring. Hierarchical software landscape visualization for system comprehension: A controlled experiment. In *Proceedings of VISSOFT 2015*, pages 36–45, 2015.

[10] F. Fittkau, S. Roth, and W. Hasselbring. ExplorViz: Visual runtime behavior analysis of enterprise application landscapes. In *Proceedings of ECIS 2015.* AIS, 2015.

[11] F. Fittkau, P. Stelzer, and W. Hasselbring. Live visualization of large software landscapes for ensuring architecture conformance. In *Proceedings of ECSAW 2014*. ACM, Aug. 2014.

[12] F. Fittkau, A. van Hoorn, and W. Hasselbring. Towards a dependability control center for large software landscapes. In *Proceedings of EDCC 2014*, pages 58–61. IEEE, May 2014.

[13] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. In *Proceedings of VISSOFT 2013*, Sept. 2013.

[14] S. Frey and W. Hasselbring. The CloudMIG approach: Model-based migration of software systems to cloud-optimized applications. *International Journal on Advances in Software*, 4(3 and 4):342–353, 2011.

[15] S. Frey, W. Hasselbring, and B. Schnoor. Automatic conformance checking for migrating software systems to cloud infrastructures and platforms. *Journal of Software: Evolution and Process*, 25(10):1089–1115, Oct. 2013.

[16] W. Hasselbring. Information system integration. *Communications of the ACM*, 43(6):32–36, 2000.

[17] W. Hasselbring. The role of standards for interoperating information systems. In *Information Technology Standards and Standardization: A Global Perspective*, pages 116–130. Idea Group Pub., 2000.

[18] W. Hasselbring. Component-based software engineering. In *Handbook of Software Engineering and Knowledge Engineering*, pages 289–305. World Scientific Publishing, 2002.

[19] W. Hasselbring. Web Data Integration for E-Commerce Applications. *IEEE Multimedia*, 9(1):16–25, 2002.

[20] W. Hasselbring. Formalization of federated schema architectural style variability. *Journal of Software Engineering and Applications*, 8(2):72–92, Feb. 2015.

[21] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. The Dublo architecture pattern for smooth migration of business information systems. In *Procceedings of ICSE 2004*, pages 117–126. IEEE, 2004.

[22] W. Hasselbring and A. van Hoorn. Open-source software as catalyzer for technology transfer: Kieker's development and lessons learned. TR-1508, Department of Computer Science, Kiel University, Aug. 2015. http://eprints.uni-kiel.de/29463/.

[23] R. Heinrich, E. Schmieders, R. Jung, K. Rostami, A. Metzger, W. Hasselbring, R. Reussner, and K. Pohl. Integrating run-time observations and design component models for cloud system analysis. In *Proceedings of the 9th Workshop on Models@run.time*, volume 1270, pages 41–46. CEUR, Sept. 2014.

[24] N. S. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring. Automatic failure diagnosis in distributed large-scale software systems based on timing behavior anomaly correlation. In *Proceedings of CSMR 2009*, pages 47–57. IEEE, 2009.

[25] H. Niemann, W. Hasselbring, T. Wendt, A. Winter, and M. Meierhofer. Kopplungsstrategien für Anwendungssysteme im Krankenhaus. *Wirtschaftsinformatik*, 44(5):425–434, 2002.

[26] M. Roantree, J. Murphy, and W. Hasselbring. The OASIS multidatabase prototype. *ACM SIGMOD Record*, 28(1):97–103, Mar. 1999.

[27] A. van Hoorn et al. Dynamod project: Dynamic analysis for model-driven software modernization. In *Proceedings of MDSM 2011*, pages 12–13, 2011.

[28] A. van Hoorn, M. Rohr, I. A. Gul, and W. Hasselbring. An adaptation framework enabling resource-efficient operation of software systems. In *Proceedings of WUP 2009*. ACM, 2009.

[29] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of ICPE 2012*, pages 247–248, Apr. 2012.

[30] J. Waller, N. C. Ehmke, and W. Hasselbring. Including performance benchmarks into continuous integration to enable DevOps. *SIGSOFT Softw. Eng. Notes*, 40(2):1–4, Mar. 2015.

[31] J. Waller and W. Hasselbring. A comparison of the influence of different multi-core processors on the runtime overhead for application-level monitoring. In *Proceedings of MSEPT 2012*, pages 42–53. Springer, June 2012.