From Monolithic To Modern Software Development: An approach to migrate and scale legacy applications using software containers, microservices and cloud computing

Marius Krämer
Karlsruhe University of Applied Sciences
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Patrick Wiener
Bosch
11 PUBLICATIONS **92** CITATIONS
SEE PROFILE

Andreas Abecker
disy Informationssysteme GmbH
250 PUBLICATIONS **3,846** CITATIONS
SEE PROFILE

Jens Nimis
Karlsruhe University of Applied Sciences
64 PUBLICATIONS **1,164** CITATIONS
SEE PROFILE

From Monolithic To Modern Software Development

An approach to migrate and scale legacy applications using software containers, microservices and Cloud Computing

Marius Krämer[1], Patrick Wiener[1], Andreas Abecker[2] and Jens Nimis[1]
Faculty of Management Science and Engineering
[1]Karlsruhe University of Applied Sciences
[2]Disy Informationssysteme GmbH
me@mariuskraemer.com

Keywords: Software Container, Microservices, Cloud-based Scaling, Software-as-a-Service

Digitalization is one major trend in the industry at which big data, cloud computing or artificial intelligence define the cornerstones of new business models. Today, more often than not, a myriad of software companies lack corresponding strategies to modernize their monolithic application stack due to predominant scepticism about new technologies as well as new development approaches. However, to foster a competitive and innovative market position, it is inevitable to migrate from monolithic to modern software development. The related challenges when dealing with monolithic applications are threefold, including the (1) deployment, (2) update process and (3) scaling of individual components, which make improvements in organizing, operating and scaling necessary. Some of those challenges include local installations, cross checking existing references and libraries when updating, or not knowing which piece of code is located in which part of the application. Monolithic applications tend to be huge collections of code thus requiring a lot of effort when scaling. While many software companies successfully apply modern technologies and strategies to move towards distributed systems and cloud-native applications there are still a lot of competitors following a traditional monolithic approach for software development.

Our goal is to develop a generic concept that addresses the presented challenges by leveraging three key technologies, namely (1) software containers, (2) microservices and (3) cloud-based scaling to modernize legacy applications eventually operating as a Software-as-a-Service (SaaS). While it is not necessarily required to apply these technologies and architectural design, they greatly intertwine and are beneficial for the overall transformation process. The three stages of the concept that evolve around the aforementioned cornerstones are described in the following as depicted in Figure 1.

The first stage uses software containers as a fundamental building block. A containerised monolith allows for the use of Container-as-a-Service, which is defined as a forth cloud computing service level that unites developing, deploying, updating and running applications (Burns, 2015). The containerisation process of the application is exemplified using Docker[1], demonstrating that a monolith can be dockerised without further changes to its architecture, as shown in Figure 1.

In the second stage, we introduce microservices to the concept. With microservices there is the need to introduce organizational changes to a development process, by using concepts like agile and lean development (Anderson, 2003) such as an iterative SCRUM (Schwaber, 2004) process instead of the commonly known waterfall approach (Bell & Thayer, 1976).

[1] https://www.docker.com/

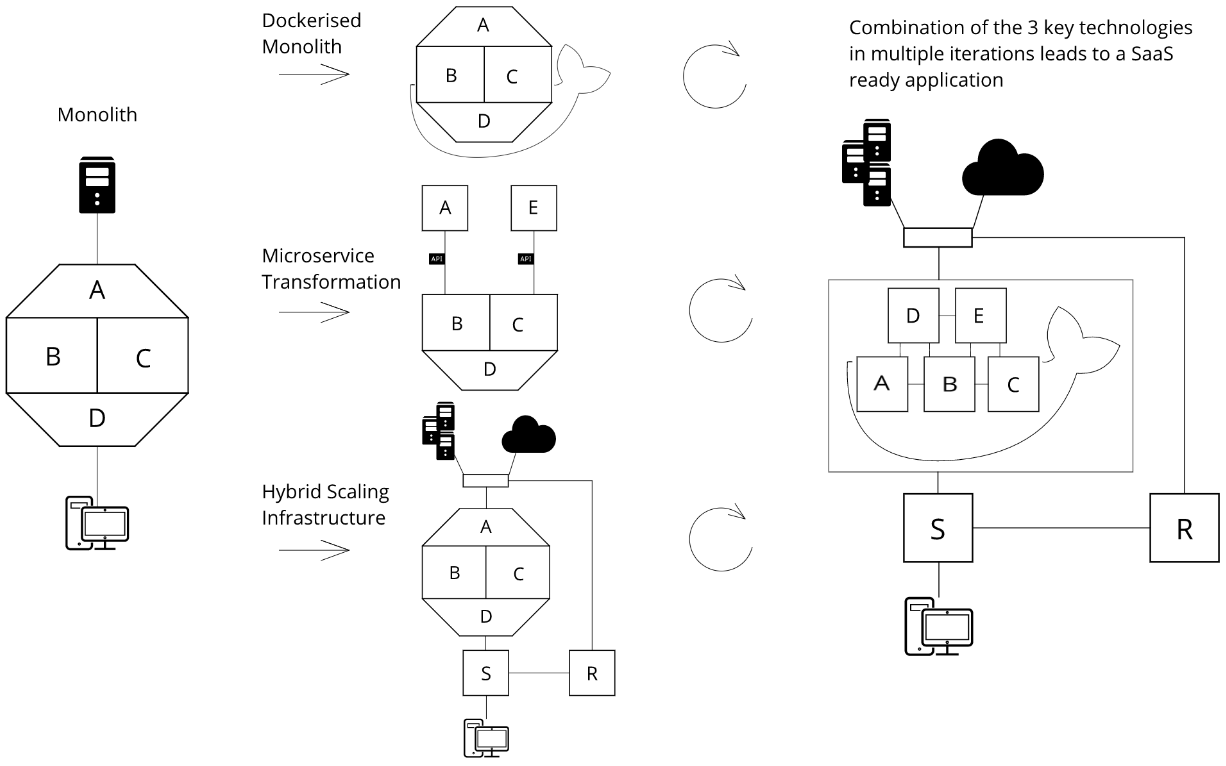
Monolith
A
B
C
D
Dockerised Monolith
Microservice Transformation
API
E
Hybrid Scaling Infrastructure
S
R
Combination of the 3 key technologies in multiple iterations leads to a SaaS ready application

Fig. 1 Approach to migrate and scale legacy applications using software containers, microservices and cloud computing.

Three different approaches can be used to transform an application's architecture into multiple microservices. The first approach is based on the law of holes meaning that if in an untenable position, it is best to stop carrying on and exacerbating the situation (Healy, 2013). Thus, new functionalities and features are attached to the monolith as additional microservices by using an API, as shown in Figure 1 by microservice “E”. The second approach separates front- and backend, but is only applicable to some special use cases, which is why this approach is not further discussed. The most common approach can be called “extract don't expand”. It combines the first approach by gradually extracting existing services from the monolith and attaching them as microservices, as shown in Figure 1 by extracting functionality “A” from the monolith and re-attaching as a microservice. To decide on the order of extraction, we developed a prioritisation matrix.

The third stage centres around scaling an application in a hybrid or entirely cloud-based environment. We evaluate existing scaling mechanisms, which are used in practice and are currently under development in research (Xiao et al., 2014; AWS, 2017; Scalr, 2017). One example is the prediction-based scaling mechanism Scryer, developed by Netflix (Jacobson et al., 2013). We leverage different scaling mechanisms to design a holistic approach to scale applications including (1) decision-based scaling, (2) reactive scaling and (3) predictive scaling element, as represented by the scaling engine “S” in Figure 1. The decision-based scaling enables the user to scale an application based on guidelines across a hybrid environment. The reactive scaling element scales an application based on metrics of the current workload. The predictive scaling element uses past metrics to scale an application in advance of the arising workload. An automatic scaling mechanism further requires a routing service, which is represented by the element “R” in Figure 1.

Eventually, the three key technologies can be combined to easily migrate into a SaaS-model. Any combination showed as valid and brings significant benefits to deploying, updating and scaling an application. For example, it is more efficient to scale the service, on which the actual workload appears, instead of scaling the entire monolithic application. Containerised microservices make this scaling even faster and allow for a clear separation between services and the monolith. The final decision is about deploying the newly formed application with in a single or multi-tenant architecture. With multiple iterations of all three key technologies it is possible to successfully transform a monolithic application into a SaaS-ready application, as shown in the final architecture in Figure 1.

We conducted a proof of concept in cooperation with a local software company to modernize their monolithic application design. The monolith includes a desktop and a web application, while the company's long-term goal is to develop an independent web application. At first the monolith was containerised and could be deployed as a Docker container. In a first step, the web monolith was extracted, which was consequently missing some functionalities from the desktop application. We identified the needed modules in the desktop application and prioritised them for extraction according to the proposed prioritisation matrix. We evaluated a common use case, in which the traffic will randomly exceed the applications capacities. The usual scaling approach was to manually start new instances. We showed how to displace this approach with an automatic scaling mechanism. A second use case features the seasonal customer base, which uses the application mainly for 3 months of the year. In addition to that the main traffic is generated at two fixed times of the day. A predictive scaling mechanism can greatly improve this use case. As a forth step we stated how to combine all technologies and how to deploy the application in a SaaS-model, using a single-tenant architecture.

This work is not only a generic conceptual guidance for companies to modernize their legacy software but further proves that the new technologies and development approaches can be used to cope with the presented challenges. Despite the initial success, the concept still needs some further testing and adjustment. Hence, supplementary research will focus on deploying the concept to a wide range of use cases and refining the individual stages.

References

Anderson, David J. (2003). *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Pearson Education.

AWS (2017). *Auto Scaling Documentation*. Available at https://aws.amazon.com/de/documentation/autoscaling/ [24.08.2017].

Bell, T. E. & Thayer, T. A. (1976). Software requirements: Are they really a problem?, *Proc. 2nd Int. Conf. on Software engineering*. IEEE Computer Society Press.

Burns, Brendan. (2015). Brendan Burns, *Google: An Overview of Kubernetes*. Available at https://soundcloud.com/thenewstackmakers/brent-butler-of-google-gives-an-overview-of-kubernetes [24.08.2017].

Healy, Dennis (1996). *Healy's First Law of Holes is to stop digging*. Available at https://business.highbeam.com/794/article-1G1-19124747/healey-first-law-holes-stop-digging-so-he-believes [24.08.2017].

Jacobson, D., Yuan, D. & Joshi, N. (2013). *Scryer: Netflix's Predictive Auto Scaling Engine*. The Netflix Tech Blog. Available at https://medium.com/netflix-techblog/scryer-netflixs-predictive-auto-scaling-engine-a3f8fc922270 [24.08.2017].

Scalr (2017). Documentation. Available at https://scalr-wiki.atlassian.net/wiki/ [24.08.2017].

Schwaber, K. (2004). *Agile Project Management with Scrum*. Microsoft Press. Redmond, Washington.

Xiao, Z., Chen, Q. & Luo, H. (2014). Automatic Scaling of Internet Applications for Cloud Computing Services, in: *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1111-1123.