In Microservices We Trust — Do Microservices Solve Resilience Challenges?

Marcus Hilbrich
marcus.hilbrich@informatik.tu-chemnitz.de
Technische Universität Chemnitz
Germany

ABSTRACT

Resilience is an open challenge. In this paper we look into microservices–a concept that argues to be resilient. We look into the definition of microservices and argue whether the definition provides the promised advantages regarding to resilience.

KEYWORDS

Microservice, Resilience, Concept, Discussion

1 INTRODUCTION

Building high quality software is not easy. The software needs to provide a functionality, to offer a use. In addition, the software has to offer non-functional properties to be useful. It needs to deliver correct results and it needs to be available. In an defined working environment, this is still a challenge and we have to consider various system properties (safety, security, fault tolerance, performance, down times, etc.) and we have seen various methods to foster this properties (redundancy, threat modeling, test driven development, verification, certification processes, system/software modeling processes, exception management etc.). When we consider a unforeseeable environment, it gets even more complicated. We have to offer the same system properties without knowing what we can rely on. Each hard- and software we are using to operate our system, can get unavailable (or be even corrupted) at any time. Thus, to build a system that can uphold its system objective as good as possible (in short it is resilient), is even more complicated.

The open challenge is: how to constructs resilient systems? The architecture of software and the process of software creation should foster resilience. It should offer concepts to increase resilience directly or at least offer properties to simplify to develop a resilient system.

In this paper, we look into microservices. Microservices is a concept that claim to support the development of huge and complex systems that have to be operated in an unforeseeable environment. Thus, microservices should foster resilience. We discuss whether microservices really help to develop resilient systems. Therefore, we have to find out the definition of microservices and other terms (Sec. 2). Afterwards, we discuss the relation of microservices and resilience (Sec. 3) and close with summery and conclusion (Sec. 4).

https://doi.org/10.18420/fbsys2019-02

2 DEFINITIONS

Before we can argue whether microservices solve resilience challenges, we need to define the terms we speak about. We start with resilience, go on with the system we have to consider. Afterwards we define the term "functional concern" that is defined in the microservice context and allows to define microservices. Based on the definitions, we can argue about resilience for microservices in the next section.

2.1 Resilience

Resilience is the ability of a system to deal with an unforeseeable changing environment in a useful meaner (according to [2]). Avizienis et al. [1] describe resilience as synonym for fault tolerance which includes also the system itself as source of faults. This means, whatever goes wrong (broken network connections, failing hardware, incorrect software, attacks, etc.) the system protect its objectives as good as possible–whatever the system's objectives are.

This offers two aspects of resilience:

(1) we have to deal with an unpredictable environment (including the system itself), and
(2) we have to consider the system's objectives.

2.2 Considered System

The system is the object we are looking for. Thus, we need to consider resilience based on the system we are observing. In case we look to a single component, we observe the interfaces to measure the quality of reached objectives. Faults can be caused by the software itself, from used hardware, or other components etc.. In case we observe the overall system, we observe the system's interfaces (e.g. interface to the user). The individual interfaces of the components the system is built of are not directly relevant.

2.3 Functional Concerns

To define microservices, we need to define functional concerns (or business concerns) as it is used in microservice context. A functional concern is an (isolated) interest of an user to the system (this can be a human or another system). The functional concern contains functionality and covers everything that is needed to fulfill this concern. An example is, the user wants to delete the user's account. This would include the tasks persistence of accounts, authorization, authentication, etc.. A task to fulfill a concern can be helpful for other concerns, too. Demands like the system has to be secure are no functional concerns, we call them cross cutting concerns.

2.4 Microservices

The term microservice has no clear definition. We observed different not useful definitions, like "microservices are small" according to

[5, 6]. Smartphone applications, projects of end user programming, and scripting are typical examples for small projects or systems. In most persons understanding, these examples are no microservices. So, the definition is not discriminating and not really defining microservices.

We need a clear and discriminating definition. Therefore, we considered common definitions (e.g., [3, 4]) and consolidated them. We decided to consider the definition of the system's structure as architectural style and the process of development and operation by the trailering of the software process model. To display the difference to the crowded microservice definitions, we also use an new term.

The slice service style (microservices) is an architectural style, where the functional concerns of the system are encapsulated to services (slices or vertical services) that deliver the functionality to end-users and have no (or minimal) dependencies to other slices of the system. This includes code sharing, usage of interfaces, sharing of manpower, and management of e.g., creation, deployment, and operation.

3 MICROSERVICES AND RESILIENCE

Based on the microservice definition, it is clear that a microservice system consists of independent microservices (slices) that are (mostly) independent to each other. So, we can use the overall system as reference or a single microservice.

3.1 Overall System

In case we consider the overall system, microservices can be helpful to foster resilience. In case a microservice is failing (e.g. based on a non-working execution environment), it cannot hinder another microservice in operation, based on the independence of separate microservices. So, all other microservices offer the functional concerns they are representing. As result, a failure that influences only individual microservices influences not the functional concerns of additional microservices.

In systems with other structures the impact of a failure can be more dramatic. E.g., in an monolithic system, a failure can escalate and hinder the complete system from operation (no functional concern is fulfilled), or a failing layer in an layered system can result in a failing of all layers (transitively) depending on it.

Thus, a microservice system can keep alive objectives defined by functional concerns in case of a single microservice is failing based on the separation of functional concerns.

Beside, a microservice represents one functional concern of the system. Thus, the complexity and size of a microservice is less than the overall system. A reduction of complexity and size simplifies the development of a (resilient) system. This advantage is similar to other concepts that divide a system in subsystems e.g., component based development or layered systems.

3.2 One Slice or Microservice

In case we consider a single microservice or slice, the definition does not give an direct benefit in relation to resilience. It is not defined to develop a single microservice with resilient in mind at all.

However, the definition fosters the development of resilient microservice (at least indirectly):

- The overall objectives of the system are separated in functional concerns with their individual non-functional concerns. So, each microservice has its own individual objectives that can be achieved independent of objectives of other microservices. Thus, the objectives that have to be covered by resilience are more clear and less complex then to consider the overall system. Therefore, it is more clear which objectives have to be considered by resilience in contradiction to e.g., a layered system, where a layer has to support all (probably contradicting) objectives of the system.
- The development team of a microservice is responsible for development and operating the microservice, also the microservice is not reused (based on its independence). So, the developers and operators of the microservice are aware of the limitations of the microservice and of the current manner of operation and it is not needed to consider a general usage of the microservice. This concept is also known as DevOps.
- One microservice covers all aspects, there are no layers, services, component usage etc. that are not part of the microservice. Thus, it is possible to have the microservices resilience aims in mind for each part of the microservice development.

4 SUMMERY AND CONCLUSIONS

Microservices are no silver bullet to answer all challenges of resilience. The organization of the system based on functional concerns results at least in an avoidance of microservices to harm other parts of the system. This is at least an clear positive impact in relation to resilience.

In addition, the separation of functional concerns has advantages that at least foster resilience by reduction of complexity. A guaranty for resilience cannot be given, the microservice still needs to be developed to be resilient.

Furthermore, we need to consider additional challenges. Cross cutting concerns like security, performance, authentication, etc. are known open challenges for microservice systems. Sharing of code and knowledge across microservices is another open challenge. Beside microservices, this is a challenge that was answered by different concepts of reuse like component based design, object orientation, reusable libraries, etc.. Moreover, the definition of functional concerns by dividing the overall system in independent parts is a major challenge that is not easy to answer.

REFERENCES

[1] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (Jan 2004), 11–33. https://doi.org/10.1109/TDSC.2004.2

[2] AXELOS Limited. 2011. ITIL® glossary and abbreviations. https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_GB-v1-0.pdf

[3] James Lewis and Martin Fowler. 2014. Microservices: a Definition of this new Architectural Term.

[4] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, and Mike Amundsen. 2016. *Microservice Architecture : Aligning principles, practices, and culture.* O'Reilly Media.

[5] S. Newman. 2015. *Building Microservices.* O'Reilly Media.

[6] Stefan Tilkov. 2019. Microservices: A Taxonomy. In *International Conference on Microservices.* University of Applied Sciences and Arts Dortmund, Germany. https://www.conf-micro.services/2019/papers/Microservices_2019_paper_29.pdf