Tools for describing architecture of distributed systems based on microservices: overview and improvements

***Abstract*—Microservices represent a software development technique for developing an applications as a collection of small, loosely-coupled services. This approach of developing software applications is gaining more and more popularity because of its potential to deliver scalable and resilient solutions. However, one of the main technical challenges that appear when implementing microservices is lack of tools that would facilitate development, deployment, and monitoring of microservices. In this paper, we provide an overview of currently available tools for describing the architecture of microservice-based distributed systems. The tools analyzed in this paper allow users to create models of their application, that are afterwards transformed into runnable applications. Nevertheless, the application of these tools is rather limited to a certain domains because the generated applications are (mostly) Java-based and use synchronous communication style. In order to mitigate the shortcomings of the current tools in the domain, we came up with the feature set for a generic tool that will allow users to experiment with different programming languages, communication styles and deployment strategies, but also provide a mechanism for evaluating the architecture of the system.**

I. Introduction

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API[1]. Because microservices are independent from each other, developers can use the most suited technologies for their implementation.

Microservice architectures (MSA) can be seen as a descendant of Service-Oriented Architectures (SOA). However, there are significant differences between the two styles, both in practical and conceptual nature[2]. For example, SOA relies on the Enterprise Service Bus (ESB) whose primary duties are to route, mediate and transform messages between services, while, in contrast, MSA spreads these duties across multiple infrastructural components such as: *API Gateway*, *Circuit Breaker*, *Service Discovery*, *Load Balancer*, etc.

In recent years, MSA is gaining popularity for its potential to deliver scalable and resilient software solutions. According to the Jakarta EE Developer Survey[1] that was conducted over a two-week period in March 2018, about 46% of the organizations are already developing their application by applying the microservice architectural style, with another 21% planning to join them in the coming years.

[1]Jakarta EE Developer Survey 2018 - https://jakarta.ee/documents/insights/2018-jakarta-ee-developer-survey.pdf

However, there are certain technical challenges that need to be overcome when it comes to the implementation of microservices. Studies presented in [3] and [4], and a survey conducted by the *RedHat*[5], clearly point to the lack of software tools that would facilitate development, deployment, managing and monitoring of microservices. Additionally, the development of a microservice-based application is marked by writing a lot of boilerplate code to set up the necessary infrastructure, e.g. service discovery or an API gateway, rather than implementing actual business logic[6].

Rest of the paper is organized as follows. Section II provides an overview of currently available tools for describing the architecture of microservice-based distributed systems. Section III presents the tool that addresses the issues with analyzed tools. Finally, Section IV concludes the paper.

II. Current state of the art

In this section we are presenting current state of the art tools for describing the architecture of distributed systems based on microservices. We will shortly describe each of the tools and, in the end of the section, we will point to a possible improvements.

Currently available tools for describing the distributed systems based on microservices include MAGMA[7], AjiL[6], TheArchitect[8], and Microbuilder[9].

All of these tools are designed for the purpose of accelerating the development of microservices. Users are able to model their systems by using specially designed user interfaces, and to use the model to generate runnable applications. The text that follows will provide a brief overview of each of the tools.

MAGMA (*Maven Archetype for Generating Microservice Architectures*) is a tool based on the Maven[2] build management system. It aims at accelerating the development of microservices architectures (MSA) by generating infrastructure code that is: (i) specifically configured for the target application domain; (ii) directly runnable; (iii) extensible with user-defined templates. Since it's based on Maven, MAGMA can only be used to generate microservices in Java programming language. Fig. 1 shows how MSA concepts are represented in MAGMA. Besides MAGMA, Java ecosystem offers a set of similar solutions like **JHipster**[3] and **Spring Initializr**[4].

[2]Maven - https://maven.apache.org/
[3]JHipster - https://www.jhipster.tech/
[4]Spring Initializr - https://start.spring.io/

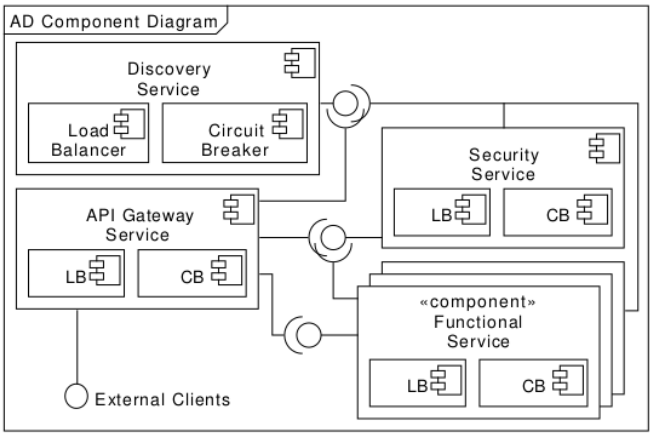

Fig. 1. MSA concepts in MAGMA[7]

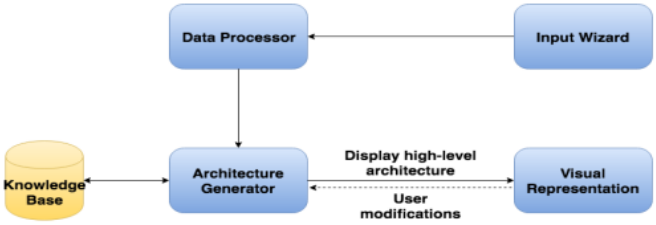

Fig. 2. High-level architecture of the TheArchitect[8]

JHipster is a web-based tool that can generate code for MSA-based applications. Applications generated by JHipster are based on Spring[5] and AngularJS[6] frameworks. Spring Initializr is a tool for generating Maven configurations that serve as a basis for Spring-based MSA applications.

AjiL is a modeling language for creating and describing MSAs based on Eclipse Modeling Framework (EMF)[7]. Besides textual, AjiL comes with graphical notation for modeling microservices, their interfaces and data that they exchange. AjiL uses specially tailored code generator to transform the model into a runnable Spring-based application.

TheArchitect is a rule-based system for serverless-microservices based high-level architecture generation. In order to create an MSA via TheArchitect, it is necessary to provide system requirements into its input wizard. The obtained information is then processed by the predefined set of models, which are subsequently analyzed by the architecture generator. After the analysis, the application code is generated. The high-level architecture of *TheArchitect* is shown in Fig. 2.

MicroBuilder is a tool for generating REST-based microservices. It comprises MicroDSL and MicroGenerator modules. MicroDSL is a Domain-Specific Language (DSL) with both textual and graphical notation, used to specify the

[5]Spring - https://spring.io/
[6]AngularJS - https://angularjs.org/
[7]Eclipse Modeling Framework - https://www.eclipse.org/modeling/emf/

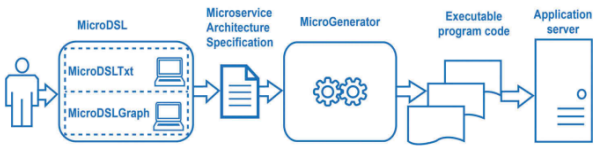

Fig. 3. Overview of the MicroBuilder architecture[9]

architecture of the microservice. MicroGenerator uses the MicroDSL specification to generate Java application based on Spring framework. Overview of the Microbuilder architecture is show in Fig. 3.

The world of MSA is a heterogeneous one. According to the study presented in [10], possibility to experiment with different technology is an important reason why companies migrate their monolithic applications to microservices. In addition to that, more than 33% of correspondents in Jakarta EE Developer Survey are using multiple languages to construct more than 80% of their applications. With that in mind, ability to generate application's code in multiple programming languages would be a great improvement for the aforementioned tools.

It is not uncommon to have a system where microservices are written in different programming languages. Because of this, communication between them must be implemented in a language-agnostic manner. The chosen communication style directly affects scalability and performance of the system. Microservices generated with the tools described above can only use synchronous REST-style communication, which narrows their domain of applicability.

Another two important features where the tools analyzed in this section could improve are: (i) support for multiple deployment strategies, and (ii) support for multiple service discovery strategies. Microservices are independently deployable, which means different microservices can be deployed on a potentially different platforms. One of the most popular deployment techniques for deployment of microservices is containerization. Most frequent tool used for containerization of microservices is Docker[8]. Since number of instances of a microservice and its location can change dynamically, it is necessary to implement a service discovery mechanism that will enable clients to communicate with the microservices. That discovery mechanism is usually a *Service Registry* where microservices register themselves during start-up. There are two basic service discovery patterns that can be implemented, each with its benefits and drawbacks: client-side discovery, and server-side discovery. Benefit of client-side discovery is that it has fewer moving parts than the server-side discovery, but it also tightly couples client with a *Service Registry*. Because they generate Spring-based applications, *MAGMA*, *AliJ* and *MicroBuilder* have possibility to use *Eureka*[9] which supports only client-side discovery. *TheArchitect*, however, provides no

[8]Docker - https://www.docker.com/
[9]Eureka - https://github.com/Netflix/eureka

solution for service discovery.

III. Proposed feature set for a tool for describing the architecture of microservices

Even though microservices as a style of developing distributed systems emerged quite recently, the design patterns and guidelines on how to develop microservices are already there. Based on the tools analyzed in Section II and the design patterns defined in [11], we came to a conclusion that the generic tool for describing architecture of distributed systems based on microservices should have the following features:

- Support for both textual and graphical notation.
- Support for multiple target languages.
- Support for multiple communication styles (Remote Procedure Call, messaging, Domain-Specific Protocol) - provides ability to switch between different communication styles allows developers to choose the best suited communication style for their needs.
- Support for multiple deployment strategies - provides ability to choose how microservices are packaged and deployed.
- Support for multiple service discovery strategies - provides ability to chose how microservice instances are being located.
- Ability to evaluate system's architecture - evaluation should be focused on following:
  - Coupling calculation - if a microservice *A* is calling a method of a microservice *B*, that means that the microservice *A* depends on the microservice *B*. The main thought behind coupling calculation is that dependence on many different microservices is bad for reusability, extendibility and maintainability[12]. Microservices should be loosely coupled, with high cohesion.
  - Bottleneck detection - if a large number of microservices relay on a certain microservice, this could mean that this microservice is a possible bottleneck. It also indicates that the dependency on the specific microservice is high, which can lead to both scalabilly and performance problems.
  - Failure analysis - show in which degree microservice is affected by failure of microservices that it depends upon.
- Ability to suggest the architecture of the system based on the user's input - similarly to *TheArchitect*, the tool should be able to suggest the best way in which the end application should be implemented, based on the design patterns and guidelines defined in [11]. This feature would enable less experienced developers to come up with the best possible architecture of the system.

IV. Conclusion

In this paper, we presented currently available tools for describing the distributed systems based on microservices. Given the rise in popularity of microservice-based applications, we argue that such tools will be of great value in the upcoming years. Tools analyzed in this paper aim at accelerating the development of microservice-based applications by enabling users to create models of their systems, which are afterwards transformed into a runnable applications via code generators. All of these tools have been used to develop real-world applications. However, potential improvements are pointed out in Section II.

Based on these tools, and the already-existing design patters for microservice-based applications, we came up with the feature set for a generic solution that would mitigate shortcomings of existing solutions presented in this paper. The feature set is presented in Section III, and it will serve as a base for our future work.

References

[1] M. Fowler and J. Lewis, "Microservices." https://www.martinfowler.com/articles/microservices.html. Accessed: 2017-04-30.

[2] F. Rademacher, S. Sachweh, and A. Zündorf, "Differences between model-driven development of service-oriented and microservice architecture," in *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*, pp. 38–45, IEEE, 2017.

[3] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in *CLOSER (1)*, pp. 137–146, 2016.

[4] P. Di Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *Software Architecture (ICSA), 2017 IEEE International Conference on*, pp. 21–30, IEEE, 2017.

[5] C. Saavedra, "The state of microservices survey 2017 – eight trends you need to know." https://developers.redhat.com/blog/2017/12/05/state-microservices-survey-2017-eight-trends-need-know/. Accessed: 2018-04-23.

[6] J. Sorgalla, "Ajil: A graphical modeling language for the development of microservice architectures," in *Extended Abstracts of the Microservices 2017 Conference*, 2017.

[7] P. Wizenty, J. Sorgalla, F. Rademacher, and S. Sachweh, "Magma: build management-based generation of microservice infrastructures," in *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, pp. 61–65, ACM, 2017.

[8] K. Perera and I. Perera, "A rule-based system for automated generation of serverless-microservices architecture," 2018.

[9] B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljević, and I. Luković, "Development and evaluation of microbuilder: a model-driven tool for the specification of rest microservice software architectures," *Enterprise Information Systems*, pp. 1–24, 2018.

[10] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation," *IEEE Cloud Computing*, no. 5, pp. 22–32, 2017.

[11] C. Richardson, *Microservice patterns*. Manning Publications, 2017.

[12] J. Muskens, M. Chaudron, and R. Westgeest, "Software architecture analysis tool: Software architecture metrics collection," in *Proceedings 3rd PROGRESS Workshop on Embedded Systems (Utrecht, The Netherlands, October 24, 2002)*, STW Technology Foundation, 2002.