EPiC Series in Computing
Volume 75, 2021, Pages 41–48
CAINE 2020. The 33rd International Conference on Computer Applications in Industry and Engineering

Microservice-based Architectures: An Evolutionary Software Development Model

Aziz Fellah and Ajay Bandi

Northwest Missouri State University, School of Computer Science & Information Systems
Maryville, MO 64468
afellah@nwmissouri.edu, ajay@nwmissouri.edu

Abstract

Microservices have recently emerged as an architectural style that gained widespread popularity in industries. Not long time ago, software applications were designed *monolithically*, that is all components were woven together as one single executable artifact unit sharing the resources of the same machine. In this paper, we look at microservice architectures through evolutionary lenses as it does not capture the essence of a new software movement. Microservices offer a new trend in software architecture and deliver a set of benefits and best practices. However, this is by no means without their own share of challenges and problems that are self-inflicted or inherited from its predecessors (*i.e.,* component-based software architecture (CBSA), service-oriented architecture, (SOA), and service-oriented computing (SOC). The evolution of these different paradigms and their gradual interweaving have fostered the development of microservices afterwards. We introduce two finite state-based formalisms called, *monitoring microservice automata* (MMA) and *container microservice automata* (CMA). The former is a powerful and parallel formalism to model microservices' infrastructures, including monitoring microservices' functionalities, resource usage, compositions, and interface operations. The later models each microservice functionality independently as an automaton that accounts for local behavior that contains a microservice and its code. Such as code is required to run within an isolated environment and a system which is fully supported by MMA. As another phase of the evolution of agile software development, microservice architectures have made their footprints in several industries such as Amazon, Twiter, PayPal, LinkedIn, Netflix, and SoundCloud.

1 Introduction and Background

Microservices have recently emerged as an architectural style that gained widespread popularity in industries. Not long time ago, software applications were designed *monolithically*, that is all components were woven together as one single executable artifact sharing the resources of the same machine. In monolithic centralized architectures, the self-contained components can communicate via method invocations or function calls such as, for example, Network Objects, RMI or CORBA [21, 23, 26]. However, the deployment of monolithic applications suffer from several issues such as maintainability, dependency, growing complexity, and scalability. Then, software system development has shifted its emphasis from traditional monolith building to a component-based approach. In general, component-based software architecture (CBSA) [10, 8] has been emerged as a feasible approach to overcome and address the software complexity in different domain areas. The component-based paradigm is composed of a collection of functional building blocks components as well as their interactions which have became a system blueprint in modern software engineering development life cycle. These components are computational elements with different responsibilities and functionalities, and consist of self-contained software artifacts have given better control over design, implementation and evolution of software systems. On an object-oriented programming language level, components stem fundamentally and loosely from modules, classes, objects, and functions in the source code. In the last decade, we have seen a further shift towards the concept of service-oriented computing, an emerging paradigm for distributed computing and e-business processing that entangles both objects' and components' computations. The evolution of these different paradigms, monolithic, component-based software architecture (CBSA), service-oriented architecture, (SOA) service-oriented computing (SOC), and their gradual interweaving have fostered the development of microservices afterwards.

In this paper, we look at microservice architectures through evolutionary lenses and not through revolutionary lenses as it does not capture the essence of a new software movement. Microservices offer a new trend in software architecture and deliver a set of benefits and best practices. However, this is by no means without their own share of challenges and problems that are self-inflicted or inherited from its predecessors (*i.e.,* SOA, SOC). In fact, microservices have made their footprints in several industries such as Amazon, LinkedIn, Netflix [16], PayPal, Twiter, and SoundCloud but sill they are not widely adopted by both industry and academia.

Although there is no uniform definition of microservices, but as a whole microservices have different definitions in the literature [23, 26]. Terminologically, microservice and microservices are used for both singular and plural terms in the concordant context of the sentence. In this paper, we give a general idea of what a microservice is while highlighting some of its advantages such as reducing complexity, minimizing coupling, maximizing cohesion, and increasing scalability. Microservices are a new trend in software architecture for developing a single application as a series of small, autonomous, and distributed services that communicate with lightweight mechanisms, often an HTTP API's (examples, REST, SOAP, RPC)[14]. This architecture decomposes the services into components, commonly by functionality and relies on two main enablers containerization and virtualization platforms such as Docker to create reusable and independently container images as a viable solution.

Although there are many advantages of microservice-based architectures over their monolithic counterparts, several challenges remain and still permeate their design and implementation. That is, configuring, deploying and maintaining cross-domain microservices can be error-prone, costly and time-consuming. For example, for a microservice to be successful it has to be permanently embracing and scaling changes in requirements without duplicating efforts and replicating instances of a specific microservice.

In this paper, we propose and discuss strategies, deciding factors, and formalism for building a software microservice-based architecture research infrastructure. Such an architecture reflects the academic/industry desire that a microservice-based application should follow key design and development practices of the microservice deployment pipeline that is aligned with agile and DevOps processes (*i.e.* continuous integration (IC)). There is not one-size-microservive that fit all strategies for developing microservices. Each solution to a given functionality has a different strategy. There were bottlenecks and other issues that are not part of this investigation and could be interesting topics in research such as automation, architectural smells, layering, visibility, load balancing, and orchestration along the side of deployment and language issues

mandated by microservice-based architectures and the IT community.
In this paper, we will devote Sections 2 and 3 to evaluate web services, monolithic, and microservices architectures from a research perspective and potentially adopt the results of our study in academia and industry. In Section 4, we introduce two finite state-based formalisms called, *monitoring microservice automata* (MMA) and *container microservice automata* (CMA). The former is a powerful and parallel formalism to model microservices' infrastructures, including monitoring microservices' functionalities, resource usage, compositions, and interface operations. The later models each microservice functionality independently as an atomaton that accounts for local behavior that contains a microservice and its code that is required to run it within an isolated environment within the system which is fully supported by MMA. These small pieces of code are good candidates to refactor microservices. In Section 5, we conclude our investigation by a broader guideline that provides a starting point for researchers and practitioners in the discipline which exhibits the characteristics of this evolutionary microservices architecture.

2 Web Services Architecture

The term *web services* (successors of CORBA) also referred to as *e-services*, *web-based applications* have no universal and exact definition in the computer literature. Web services are a collection of protocols that allow applications developed in different technologies to communicate and exchange information with each other over a network. For example, Php, iOS, Android, .Net, and Java applications can communicate with each other through a variety of formats such as XML. Web services are not tied to a specific platform, operating system or programming languages [9]. Possible definitions range from a simple and generic application over the web to more accurate and dependent software components that provide distributed services to potential clients over the web. On the surface, a web service is simply an application invoked over the web. For example, an application (*i.e.,* client, browser window) would send an HTTP GET request to get a specific web service. In general, web services are much more complicated than this simple example, they may be dynamically generated and run on diverse hardware and software platforms.

Research on web services spans over a spectrum of issues, ranging from fundamental questions concerning behavior descriptions (*i.e.,* conversations) of services to the design and analysis of composite web services. Moreover, several other research aspects of web services have been addressed in the literature. For example, modeling, testing, parsing XML web services and developing new formal languages for web services. A landscape of techniques for specifying and modeling web services have been proposed, see for example [25, 11, 7, 13, 12, 6, 14, 5, 24]. Web service framework is mainly based on the fundamental standards such as SOAP [2], WSDL ((Web Services Description Language) [3, 1], WSFL (Web Services Flow Language) [15] and WSCL (Web Service C Language). The purpose of these languages is for describing protocol conversations and defining interfaces of web services.

3 Microservices Architecture

In traditional monolith architectures, software applications are developed as a collection of modules, all woven together as a single-tiered piece of executable artifacts, called monoliths. Monoliths are independently executable, share the same resources (*i.e.* memory, files, databases), and are difficult to maintain and scale due to their complexity. The microservice architecture has emerged as a new software development that functionally decomposes/splits a system into a set of autonomous services modeled around a business domain. Microservices decouple a standard monolithic application into several discreet services. Microservices can be implemented on different platforms using different programming languages and heterogeneous software tools. This allows each service to be managed independently, explicitly characterized by interfaces, and run as small autonomous processes which communicate through REST APIs. For asynchronous communication using messaging instead of REST APIs provides better performance and reliability [18]. Microservices should be small and independent but convenient in their smallness and independence. That is, promoting microservice's self-containment and proper infrastructure. Moreover, these microservices can have their own data model concealed in separate databases, or share the same database accessible to some or all instances of the application. Other key major benefits of microservice architectures are the independent life cycle of each service, flexibility, simplicity, loose coupling, ease of maintenance and scalability. Figure 1 illustrates the difference between monolith and microservice architecture. Figure 2 shows a simple microservice business example. Microservices architectures are particularly well suited for distributed systems [4]. A microservices architecture remains easy to maintain while the system evolves and features are created and updated. Some examples of API types are in either SOAP and RPC among others [17, 19, 20, 22]. interfaces such as REST (Representational State Transfer), APIs and HTML pages [24]. The API implementation is beyond the scope of this paper but suffice to say that the most common implementations rely on communication over HTTP.

`

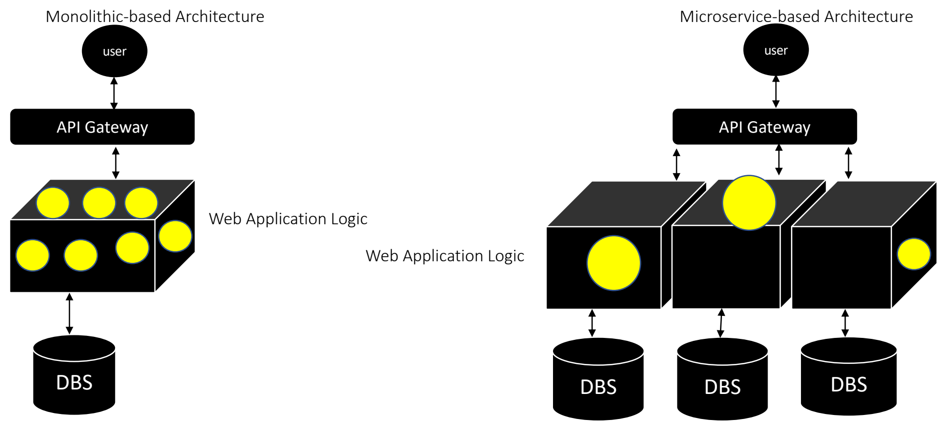

Figure 1: Monothilic-based vs microservice-based architectures.

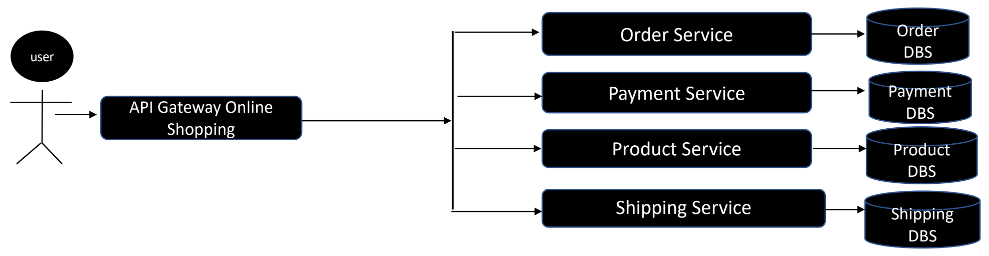

Figure 2: Components of a basic microservice architecture.

4 Monitoring and Container Microservice Automata

In this section we propose a finite state based formalism called, *Monitoring Microservice Automata* (MMA), a powerful and parallel formalism to model microservices' infrastructures, including monitoring microservices' functionalities, resource usage, compositions, and interface operations. In addition, we model each microservice functionality as a *Container Microservice Automaton* (CMA), that accounts for local behavior that contains a microservice and its no-shared code that is required to run it. CMA can be executed within an isolated environment within the system and supported by MMA to achieve performance efficiency through parallelism and independence. These small pieces of code are good candidates to refactor microservices. MMA interact through sequences of *messages*. A message can be defined as a quadruple *<e, s, p, o>* where e, s, p, o indicate *event, sender, parameters*, and *output* respectively. An event can be an input command, sender is the MMA invoking the input command and parameters are used to trigger some computations performed by CMA and requested by the invoking MMA for the specific task requested through the command. The output message could be possibly used for different purposes, for example, termination of the task and requested output.

Definition 4.1. *Let S, M be finite, disjoint set of microservices, and messages (input, output). A Monitoring Microservice Automaton (MMA) is six-tuple $D = (\Sigma_i, \Sigma_o, Q, s, F, \delta)$ where* (a) *$\Sigma_i, \Sigma_o \subseteq M$ are disjoint, finite set of input and output messages,* (b) *Q is a non-empty finite set, the set of states,* (c) *$s \in Q$ is the starting state,* (d) *$F \subseteq Q$ is the set of final states,* (e) *δ is a set of deterministic transition functions of the form $\delta : Q \times \Sigma_i \times \Sigma_o \rightarrow Q$.*

Definition 4.2. *An Extended Monitoring Microservice Automaton (EMMA) is an MMA augmented with queue and a tuple of register variables.*

The main purpose of adding a queue and a set of register variables is to store the incoming messages and support the conversation operation. An MMA extended with a FIFO queue is given in Figure 3.

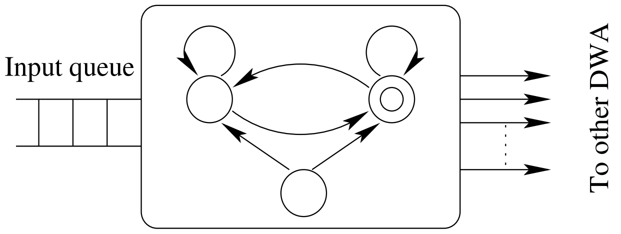

Figure 3: An extended monitoring microservice automaton (EMMA)

Let $\mathcal{C}$ be a CMA, a *container microservice configuration (cmc)* of $\mathcal{C}$ is a sequence in $Q \times \Sigma_i^* \times \Sigma_o$ where Σ_i^* denotes the Kleene star operation of Σ. Let $v, w \in \Sigma_i$, $x, y \in \Sigma_o$, and $p, q \in Q$. By $pvx \models qwy$, $pvx \overset{j}{\models} qwy$, and $pvx \overset{*}{\models} qwy$ we denote the fact there exists a simple microservice invocation, j-invocation, or an arbitrary number of invocations, respectively, from pvx to qwy. An *execution* of a microservice is a finite, nonempty sequence of configurations as follows. $stc \models pux \cdots \overset{j}{\models} qvy \cdots \overset{*}{\models} rwz$ where s is the initial state, $r \in F$ is the final state, $s, p, q, r \in Q$; $t, u, v, w \in \Sigma_i$, and $c, x, y, z \in \Sigma_o$. The set of all executions of $\mathcal{C}$ represents the behavior of the microservice modeled by $\mathcal{C}$. A monitoring microservice automaton (MMA)can trigger all container microservice automata (CMA) to work in parallel on the same input. The starting node can be any CMA where the query is initiated. Thus, the execution of an MMA represents the parallel execution of $\bigcup_{i=1}^{n}$CMA and the sequential execution of CMA, where n indicates the number of CMA. We assume that at any time of the composition operation there is only one active MMA and m CMA where $1 \leq m \leq n$. Monitoring microservice automata (MMA)are based on the *universal* and *existential* quantifiers during the course of a computation. From a practical point of view, it means that a client has started the execution of a set of interactive microservices, allowing the composition and conversation operations. In addition, it is always possible to check the existence of any composition among CMA. The results of a composition that exists is returned to the MMA node. Similarly to local container microservice configuration, we define *global monitoring microservice configurations* (*gmmc*). A run of an MMA is a rooted tree rather than a sequence of gmmc.

Now we define the *characteristic microservice vector* of $\mathbb{F} \in \mathcal{B}^{\mathbb{Q}}$ as follows:

$$f_q = 1 \Longleftrightarrow q \in \mathbb{Q}$$

$f_q = 1$ affirms that the request made by client and initiated by an MMA at node q is satisfied. We can extend ρ to a mapping of $\mathbb{Q}$ into the set of all mappings of $\Delta_i^* \times \mathcal{B}^{\mathbb{Q}} \to \mathcal{B}$ to include many but finite messages.

Theorem 4.1. *If $\mathcal{M}$ is an MMA with k states satisfying a microservice query ξ. Then ξ is satisfied by CMA with 2^{2^k} states. Moreover, the web service query ξ is satisfied by a 2^k nondeterministic microserice automata (NMA) in the worst case.*

Theorem 4.2. *Let ξ be a web service query and M an MMA. Then, ξ is logarithmic-time computable on MMA.*

Proof: We omit the proofs due to space constraints and the lengthy proofs of Theorems 4.1, 4.2. Interested readers may directly contact the authors. □

Monitoring microservice automata are a generalization of nondeterminism monitoring microservice automata (NMMA) in the following sense: if in a given state q MMA reads a message m, it will activate all CMA to work in parallel on the input. Once the CMA have completed their tasks, q will evaluate their results using a Boolean function and pass on the resulting value to the state by which it was activated. A query is satisfied if the starting state computes the value of 1. Otherwise, the query is not satisfied.

5 Conclusion

Microservices have emerged as a software development perspective and an innovative guideline at designing an architectural solution to manage complexities of a system by decomposing a monolithic application. Changes to a single service are done independently and distinctly by different teams without the involvement of any other microservices. Each microservice contains its own database and communicate with each other over the network APIs. We also have considered finite state-based automata as the basic model for exploring behaviors of microservices. Monitoring and container microservices automata can be used effectively for designing and implementimng microservices and their operations (*i.e.,* composition and orchestration) in a distributed environment. An extension of this work is being addressed in a paper in preparation.

References

[1] Bpws4j java implementations. http://www.alphaworks.ibm.com/tech/bpws4j, 2007.

[2] Soap version 1.2: Messaging framework. http://www.w3.org/TR/2007/REC-soap12-part1-20070427, 2007.

[3] Web services description language (wsdl) 2.0: Core language. http://www.w3.org/TR/2007/REC-wsdl20-2007, 2007.

[4] N. Dragoni and al. Microservices: yesterday, today, and tomorrow. *Mazzara M., Meyer B. (eds) Present and Ulterior Software Engineering. Springer Internationa Publishing, Cham*, pages 195–216, 2017.

[5] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591, 2009.

[6] T. Erl and et al. Web service contract design and versioning for soa. *Pearson: The Pearson Service Technology*, 2017.

[7] S. Zannettou et al. Understanding web archiving services and their (mis)use on social media. In *Proceedings of the Twelfth International AAAI Conference on Web and Social Media (ICWSM)*, pages 454–463, 2018.

[8] A. Fellah and A. Bandi. Automata-based timed event program comprehension for real-time systems. In *Proceedings of FASSI 5th International Conference on Fundamentals and Advances in Software Systems Integration*, pages 21–28, 2019.

[9] A. Fellah and A. Bandi. Learning language equations and regular languages using alternating finite automata. *Journal of Computing Science in Colleges*, 35(2):19–28, 2019.

[10] A. Fellah and A. Bandi. Moving towards program comprehension in software development: A case study. In *Proceedings of the Fourth International Conference on Computing Methodologies and Communication (ICCMC 2020)*, pages 660–665, 2020.

[11] X. Fu, T. Bultan, and J. Su. Analysis of interactive bpel web services. *IBM Corporation*, 2007.

[12] S. Hale, G. Blank, and V. Alexander. Live versus archive: Comparing a web archive and to a population of webpages. *UCL Press*, 2017.

[13] S. Hale, G. Blank, and V. Alexander. Dzone. retrieved from microservice testing: Coupling and cohesion (all the way down). https://dzone.com/articles/microservice - testing-coupling-and-cohesion-all-the, 2018.

[14] F. Halili. Web services: A comparison of soap and rest services. *Modern Applied Science*, 12(3:175), 2018.

[15] F. Leymann. Web services flow language (wfsl). *IBM Corporation*, 2011.

[16] T. Mauro. Adopting microservices at netflix: lessons for team and process design. https://dzone.com/articles/adopting-microservices-netflix, 2015.

[17] M. Richards. Software architecture patterns. *California: O'Reilly Media Inc.*, 2015.

[18] M. Richards. Microservices antipatterns and pitfalls. *O'Reilly Media, Inc.*, 2016.

[19] M. Richards. Microservices vs service-oriented architecture. *California: O'Reilly Media Inc.*, 2016.

[20] C. Richardson. Microservice architecture. http://microservices.io/patterns/monolithic.html, 2017.

[21] K. SeongKi and H. Sang-Yong. Performance comparison of dcom, corba and web service. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time*, pages 106–112, 2009.

[22] J. Thones. Microservices. *IEEE Software*, 32(1):113–116, 2015.

[23] G. Toffetti, S. Brunner, M. Blochlinger, F. Dudouet, and A. Edmonds. An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. ACM, 2015*, pages 19–24, 2015.

[24] M. Villamizar and et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *Proc. Computing Colombian Conference*, pages 583—-590,

2015.

[25] S. Watts. Microservices vs soa: What's the difference? https://dzone.com/articles/microservices-vs-soa-whats-the-difference, 2019.

[26] E. Wolff. Microservices: Flexible software architecture. *Addison- Wesley Professional*, 2016.