Towards a Taxonomy of Microservices Architectures

Martin Garriga(✉)

Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano, Milan, Italy
martin.garriga@polimi.it

Abstract. The microservices architectural style is gaining more and more momentum for the development of applications as suites of small, autonomous, and conversational services, which are then easy to understand, deploy and scale. However, the proliferation of approaches leveraging microservices calls for a systematic way of analyzing and assessing them as a completely new ecosystem: the first cloud-native architectural style. This paper defines a preliminary analysis framework in the form of a taxonomy of concepts, encompassing the whole microservices lifecycle, as well as organizational aspects. This framework is necessary to enable effective exploration, understanding, assessing, comparing, and selecting microservice-based models, languages, techniques, platforms, and tools. Then, we analyze state of the art approaches related to microservices using this taxonomy to provide a holistic perspective of available solutions.

1 Introduction

Microservices is a novel architectural style that tries to overcome the shortcomings of centralized, monolithic architectures [1,2], in which application logic is encapsulated in big deployable chunks. In contrast, microservices are small components, built around business capabilities [3], that are easy to understand, deploy, and scale independently, even using different technology stacks [4]. Each runs in a dedicated process and communicates through lightweight mechanisms, often a RESTful API.

Several companies have recently migrated, or are considering migrating, their existing applications to microservices [5], and new microservice-native applications and support tools are being conceived. While the adoption of this architectural style should help one address the typical facets of a modern software system: for example, its distribution, coordination among parts, and operation, some aspects are still blurred [6]. Like traditional developers, microservice adopters would benefit from a comprehensive support for the whole microservices lifecycle. All in all, the increasing number of microservice-based approaches calls for a systematic way to analyze and assess them as a completely new ecosystem: the first cloud-native architectural style [7].

In this context, this paper defines a preliminary analysis framework that captures the fundamental understanding of microservice architectures in the

A. Cerone and M. Roveri (Eds.): SEFM 2017 Workshops, LNCS 10729, pp. 203–218, 2018.
https://doi.org/10.1007/978-3-319-74781-1_15

form of a taxonomy of concepts, encompassing the whole microservices lifecycle, as well as organizational aspects. This framework enables effective exploration, understanding, assessing, comparing, and selecting microservice-based models, languages, techniques, platforms, and tools. We carried out an analysis of state of the art approaches (28 in total) related to microservices using the proposed taxonomy to provide a holistic perspective of available solutions.

The rest of the paper is organized as follows. Section 2 defines microservices architectures. Section 3 details the proposed taxonomy, and the analysis of state of the art approaches according to it. Section 4 discusses open challenges distilled from the previous analysis. Section 5 concludes the paper.

2 Microservices Architectures

The most widely adopted definition of microservices architectures is "an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP API" [1]. In contrast to monoliths, microservices foster independent deployability and scalability, and can be developed using different technology stacks [4].

However, this definition can be applied to traditional or RESTful services as well, which feeds the debate regarding microservices and SOA (Service-oriented Architectures). Although microservices can be seen as an evolution of SOA, they are inherently different regarding the sharing and reuse aspects. SOA is built on the concept of foster reuse: a share-as-much-as-possible architecture style, whereas microservices architecture is built on the concept of a share-as-little-as-possible style [8]. Given that service reuse has often been less than expected [9], instead of reusing existing microservices for new tasks or use cases, they should be "micro" enough to allow rapidly developing a new one that can coexist, evolve or replace the previous one according to the business needs [10].

3 A Taxonomy of Microservices Architectures

The first step to delineate the taxonomy comprised a literature review, following the guidelines for Systematic Literature Review (SLR) proposed in [11]. Although a SLR is outside the scope of this work, this helped us to organize the process of finding and classifying relevant literature. We searched for articles indexed in Scopus, Science Direct, IEEE Xplore, ACM Digital Library, SpringerLink, Google Scholar and Wiley Online. The search strings used were "microservice[s]", "microservice[s] architecture[s]", "cloud-native architecture[s]". The search comprised articles published up to 2016 (inclusive). Then, we applied snowballing [12], by looking for relevant references included in the works previously found, in order to identify other potential works. As microservices is a very recent topic, we considered journal, conference and workshop publications. Finally, we suppressed duplicated papers from the results, since the search engines and databases are overlapped to a certain extent. After these

steps, the initial collection consisted of 64 potentially relevant works from which we performed a detailed, qualitative analysis in order to exclude different papers of the same authors/group, incrementally reporting their results, and certain works that used the term "microservices" with a different meaning.

From this analysis we refined a collection of 46 relevant work[1], classified as: primary studies, that is, literature investigating specifically the research question (using microservices, proposing microservice-oriented frameworks, tools or architectures); and secondary studies, that is, literature reviewing primary studies (surveys, reviews and comparative studies assessing microservices or microservice-based approaches).

The former group (28 approaches, summarized in Table 1) became the target of our analysis (discussed in Sect. 4), while the latter was used to identify the concepts (and disambiguate their definitions) that are potentially relevant for the taxonomy. Additionally, we enriched this taxonomy by leveraging our previous experience with classification frameworks in the context of SOA, both for traditional Web Services [39] and the most recent RESTful services [40]. Figure 1 presents the taxonomy of concepts, which are defined below. For most of the concepts it is not possible to provide an exhaustive list of values, due to the wide (and growing) variety of approaches. Thus, we included *other* as a possible value for completeness.

Design implies thinking about the boundaries of microservices that will maximize their upsides and avoid some of the potential downsides, focused on loose coupling and high cohesion as the two key principles of SOA. The architects face this set of decisions, together with the possible choices, at the earliest stage of the lifecycle. The design space can be represented in a textual or graphical form, by means of architectural concepts. Design encompasses the following sub-concepts:

- *Design approaches* means the preexistence (or absence) of legacy software that should be transitioned to a microservices architecture, constraining its design [2]. Possible values: brownfield, greenfield.
- *Design practice*s to handle the complexity of microservices architectures into design time [41,42]. Possible values: domain-driven design, design for failure, other.
- *Architectural support* describing the obligations/constraints to be fulfilled by the microservices system, and how to apply them into a dynamic context environment [43]. Possible values: reference architectures, model-driven design, other.

Implementation implies being aware of program complexity, due to the thousands of microservices running asynchronously in a distributed computer network: programs that are difficult to understand are also hard to write and modify [23]. Also, the implementation should allow continuous evolution, which is often required by the application domain. Although implementation decisions

[1] Due to the space limit, the full list can be found in: https://goo.gl/j5ec4A.

Table 1. Microservices primary approaches

Id	Reference	Key points
01	Kratzke et al. [13]	Tradeoffs of containerized microservices and Software Defined Networks (SDNs)
02	Balalaie et al. [4]	Experience report and migration patterns for migration to microservices and DevOps
03	Bogner et al. [14]	Microservices integration from the enterprise architectures point of view
04	Toffetti et al. [7]	Cloud-native applications definition, self-managing monitoring and scaling
05	Ciuffoletti [15]	Monitoring-as-a-service for microservices
06	Florio et al. [16]	Autonomic and self-adaptable containerized microservices
07	Gabbrielli et al. [17]	Self-reconfiguring microservices using the ad-hoc Jolie Redeployment Optimizer tool
08	Gadea et al. [18]	Reference architecture to propagate database changes through microservices
09	Guo et al. [19]	PaaS Architecture based on microservices and containers
10	Gysel et al. [20]	Systematic approach to decompose a monolith into microservices
11	Heorhiadi et al. [21]	Failure testing framework for microservices
12	Kecskemety et al. [22]	Methodology to split traditional SOA architectures in microservices
13	Liu et al. [23]	Agents-oriented language and IDE for developing microservices
14	Grieger et al. [24]	Model-driven integration of microservices and self-adaptation with models at runtime
15	Safina et al. [25]	Data-driven workflows based on microservices, defined in the ad-hoc language Jolie
16	Nikol et al. [26]	Multi-tenancy microservice composition by adapting traditional BPEL workflows
17	Rahman et al. [27]	Automated acceptance testing architecture for Behavior Driven Development (BDD)
18	Savchenko et al. [28]	A framework and platform for microservices validation and testing
19	Sousa et al. [29]	Multi-cloud architecture with automatic selection of cloud providers for microservices deployment
20	Stubbs et al. [30]	Fully decentralized solution to the microservices discovery problem using Docker, Serfnode and gossip protocol
21	Sun et al. [31]	Extension to container's network hypervisor, enabling flexible monitoring and policy enforcement for microservices
22	Villamizar et al. [32]	Infrastructure cost comparison using monolithic and different microservices deployments on the cloud
23	Yahia et al. [33]	Event-driven lightweight platform for microservice composition, based on a DSL for describing orchestration
24	Bak et al. [34]	Context- and location-based microservices for mobile and IoT
25	Levcovitz et al. [35]	Identification and extraction of candidate microservices in a monolith by dependency graphs of modules and database tables
26	Meinke et al. [36]	Learning-based testing to evaluate the functional correctness and robustness of distributed microservices
27	Viennot et al. [37]	Middleware model for heterogeneous data-driven microservices integration
28	Amaral et al. [38]	Comparison between two models to deploy containerized microservices: master-slave and nested-container

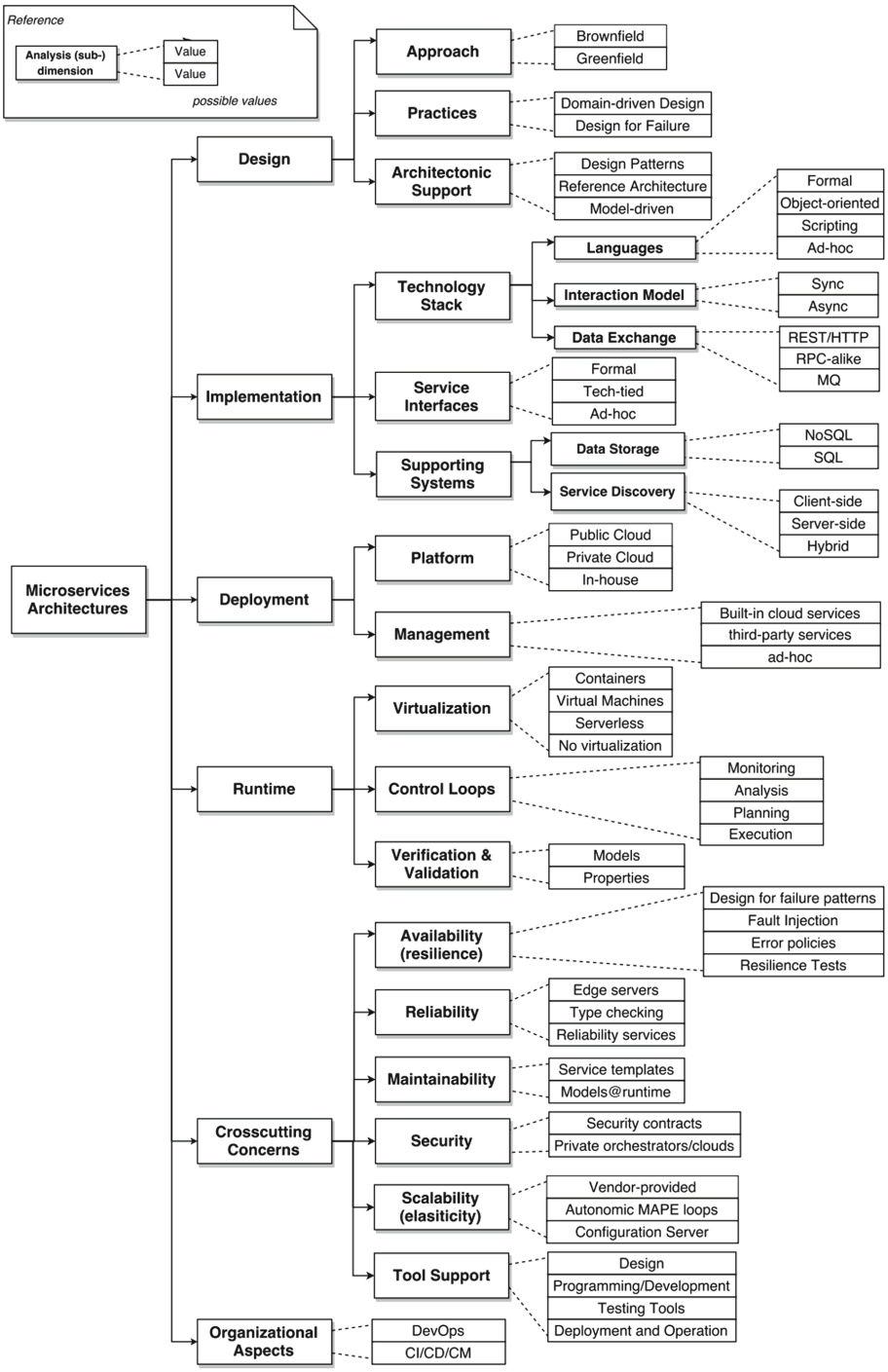

Fig. 1. Microservices taxonomy

are correlated with the (cloud) deployment of the architecture, we divided them for the sake of presentation. Thus, implementation encompasses the following:

- *Technology Stack*:
  Languages "the right tool for the right job" [44], since microservices foster polyglot languages. Possible values: formal (using formal languages to some extent), scripting, object-oriented, ad-hoc (developing a new language for microservices).
  Interaction Models refers to the communication flow among components. Possible values: synchronous, asynchronous.
  Data Exchange are the protocols used to represent the communication [44]. Possible values: REST/HTTP, RPC-alike, message queues, other.
- *Service Interfaces* are the different means of specifying contracts (if any) for the communication of microservices [45]. Possible values: formal (defined through a formal contract), tech-tied (the interface is tied to the implementation technology), ad-hoc (defined in a novel language).
- *Supporting Systems*:
  Data Storage usually integrated multiple services in legacy systems, but in microservices architectures it is mandatory to find seams in the databases and use the right technologies to split them out cleanly [3]. Possible values: SQL, graph-oriented, document-oriented, other.
  Service Discovery should allow clients to make requests to a dynamically changing, large set of transient service instances [30]. Possible values: client-driven (querying a service registry), server-driven (API gateway, load balancer).

Deployment encompasses how and where services are actually hosted and deployed. Although the cloud has been adopted as the de-facto platform for microservices [44], there are several alternatives into and out of the cloud.

- *Platform* can be customized due to privacy, security or business constraints. Possible values: public cloud, private cloud, in-house.
- *Management* encompasses the responsive reaction to failures and changing environmental conditions, minimizing human intervention [7]. Possible values: built-in cloud services (e.g., AWS Cloudwatch and Autoscaling), third-party services (e.g., Rightscale, New Relic), ad-hoc solutions (i.e., tied to the particular approach).

Runtime requires extra effort and attention given the number of independent components, log files and interactions, which can affect latency and reliability [3]. Certain microservices should be deployed and run together, and then monitored in order to detect performance degradation and perform (if possible), the self-adaptation actions to correct the behavior of the system.

- *Virtualization* encompasses the different degrees of platform abstraction, isolation and sharing. Possible values: virtual machines (VMs), containers, serverless (i.e., Functions-as-a-Service [46]), no virtualization.

- *Control loops* or MAPE loops (monitor-analyze-plan-execute) allow for different degrees of self-adaptation. This is challenging in a distributed setting since the overall system behavior emerges from the localized decisions and interactions. In our view, approaches can implement one or more stages of the control loop. Possible values: monitoring, analysis, planning, execution.
- *Verification & Validation* at runtime, concern the quality assessment of microservices throughout their lifecycle [47]. Possible values: models (at runtime), properties, other.

Crosscutting Concerns mostly regard QoS aspects that have to be tracked within the microservices lifecycle, supported by the infrastructure through specific artifacts and independent of individual microservices.

- *Availability and Resilience* imply handling both service-level and low-level failures that demand for persistence and recovery techniques [45]. Possible values: resilience patterns, fault injection, error-handling policies, resilience tests, other.
- *Reliability* refers to a system that is capable of perform well without halting, according to its requirements, and is fault-tolerant. This is particularly challenging for distributed microservices, threatened by integration and message passing mechanisms. Possible values: edge servers [4], type checking [25], reliability services [31,37], other.
- *Maintaintability* can be plainly defined by the premise "you build it, you run it" which claims for a better understanding of business capabilities, roles and operational details [48]. Possible values: service templates, models at runtime, other.
- *Security* vulnerabilities are those of SOA [45], plus the high distribution and network complexity that pose additional difficulty in debugging, monitoring, auditing and forensic [31]. Possible values: security contracts, private cloud, other.
- *Scalability* and elasticity refer to the capability to rapidly adjust the overall capacity of the platform by adding or removing resources, also minimizing human intervention [7]. Possible values: vendor-provided, autonomic MAPE loops, configuration servers, other.
- *Tool Support* should be provided given the program complexity, performance criticality and evolutive characteristics of microservices [3]. Possible values: design, programming/developing, testing, deployment/operation.

Organizational Aspects are crucial since organizations produce designs which are copies of their communication structures. Thus, a siloed organization will produce a siloed-system, while a DevOps one [4], with development and operations teams organized around business capabilities and collaboration (cross-functional teams) will be able to produce well-bounded microservices [1,8]. Although this concept is less technical when compared with the previous ones, it is important to understand the complex "organizational rewiring" scenarios that need

to be faced when transitioning to microservices [49], which may imply continuously adapting both the organization and architecture, and understand or mediate new requirements and concerns. Possible values: DevOps, Continuous delivery/deployment/integration practices, other.

Tables 2 and 3 summarize the analysis of the 28 approaches according to the taxonomy in Fig. 1. The following section discusses the findings and implications of such an analysis.

Table 2. Characterization of microservice approaches (Part 1)

	Concept	Value	Approaches	Total
Design	Approach	Brownfield	02, 03, 04, 10, 12, 16, 21, 22, 25	9
		Greenfield	06, 07, 08, 13, 15, 23, 27	7
	Practices	DDD	10, 22	2
		Design for failure	11	1
		Other	17	1
	Architectonic support	Design patterns	02, 15	2
		Reference architecture	03, 05, 08, 09	4
		Model-driven design	13, 14, 16, 19, 27	5
		Other	10	1
Implementation	Tech./languages	Semi-formal	03,05,16,25	5
		Object-oriented	02, 05, 22	3
		Scripting	11, 23, 28	3
		Ad-hoc	07, 10, 12, 13, 14, 15, 17, 19, 21, 23	9
	Tech./interaction model	Synchronous	02, 05, 16, 22, 25, 27, 28	7
		Asynchronous	13, 23, 25	3
	Tech./data exchange	REST/HTTP	01, 02, 06, 08, 10, 11, 21, 22, 23, 24	10
		RPC-alike	05, 10, 21, 25	4
		Other	13, 20	2
		Message queues	26, 27	2
	Service interfaces	Formal	05, 08, 14, 16	4
		Tech-tied	02	2
		Ad-hoc	06, 07, 10, 11, 12, 18, 23	7
	Supp. syst./storage	SQL	02, 22, 24, 25, 27	5
		NoSQL	08, 27	2
		Graph-oriented	27	1
		Document-oriented	27	1
		Other	15, 27	2
	Supp. syst./discovery	Client-side (discovery registry)	02, 04, 05	3
		Server-side (APIgateway/LB)	13	1
		Hybrid	20	1
Deployment	Platform	Public cloud	04, 06, 07, 08, 11, 12, 19, 21, 24	9
		Private cloud	04, 06, 13, 18, 19	5
		In-house	09, 28	2
	Management	built-in cloud services	22	1
		third-party services	02, 18	2
		ad-hoc solution	04, 05, 06, 07, 08, 09, 12, 13, 14, 16, 19, 20, 23, 27	14

Table 3. Characterization of microservice approaches (Part 2)

	Concept	Value	Approaches	Total
Runtime	Virtualization	Virtual machines	01, 04, 07, 12, 13, 19, 21, 22, 28	9
		Containers	01, 02, 04, 05, 06, 07, 08, 09, 11, 12, 14, 16, 19, 20, 22, 28	16
		Serverless	22	1
		No virtualization	01, 07, 28	3
	Control loop (MAPE-K)	Monitoring	05, 06, 08, 14, 20, 21, 24, 26	8
		Analysis	06, 08, 14, 26	4
		Planning	06, 26	2
		Execution	06, 26	2
		Shared knowledge	06	1
	Verification & Validation	Models	03, 14, 19, 26	4
		Properties	15	1
		Other	05, 11, 17, 18, 24	5
Crosscutting Concerns	Availability (resilience)	Patterns	02,11,20	3
		Fault injection	11, 26	2
		Error-handling policies	23	1
		Resilience tests	11	1
		Other	04, 10, 27	3
	Reliability	Edge servers	02	1
		Type checking	15	1
		Reliability services	21, 27	2
		Other	04, 13	2
	Maintainability	Service templates	02	1
		Models@Runtime	14	1
		Other	15	1
	Security	Security contracts	21	1
		Private orchestrators	23	1
		Other	10	1
	Scalability (elasticity)	Vendor-provided	04, 21, 22, 27	4
		Auto. MAPE loop	06	1
		Configuration server	02, 07	2
		Other	02, 27	2
	Tool support	Design	03, 09, 10, 13, 16, 25	6
		Programming	09, 13, 15	3
		Testing tools	09, 11, 17, 18, 26, 28	6
		Deployment/operation	02, 05, 07, 09, 16, 19, 27	7
Organizational aspects		DevOps	02, 07, 24	3
		CD/CI/CM	02, 13, 18	3

4 Discussion and Open Research Challenges

Design approaches are equally distributed between brownfield (9) and greenfield (7). The design phase is mainly supported through reference architectures and model-driven design. However, despite the hype and business push towards microservitization, there is still a lack of academic efforts regarding the design practices and patterns [10]. Design for failure and design patterns could allow to early address challenges as to bring responsiveness (e.g. by adopting "let-it-crash" models), fault tolerance, self-healing and variability characteristics. Resilience patterns such as circuit-breaker and bulkhead seem to be key enablers in this direction. It is also interesting to understand whether the design using a stateless model [50] can affect elasticity and scalability as well [43].

Another problem at design time is finding the right granularity level of microservices, which implies a tradeoff between size and number of microservices [10]. Intuitively, the more microservices introduced into the architecture, the higher the level of isolation between the business functionalities, but at the price of increased network communications and distribution complexity. Addressing this tradeoff systematically is essential for assessing the extent to which "splitting" is beneficial regarding the potential value of microservitization.

Implementation approaches mostly define their own, ad-hoc languages (9) for programming microservices or defining different parts of their architecture (e.g., in the form of DSLs). Some of those provide semi-formal support, by embracing standards such as MOF (Meta Object Facility) [14] or OCCI (Open Cloud Computing Interface) [15]. Interestingly, microservices architectures are intuitively associated to lightweight, scripting languages such as JavaScript or Python, which is not reflected explicitly in the analyzed literature [44].

Regarding the interaction model, the vast majority choose the synchronous one (7) rather than the asynchronous (3). Interestingly, microservices are most suitable for asynchronous communication, bringing performance, decoupling and fault-tolerance, but the paradigm shift implied has not been overtaken yet in practice. Synchronous request-response model is still easier to understand and implement, and much common in monolith systems (brownfield) but hinders decentralization, availability and performance. The transition from synchronous to asynchronous models calls for further analysis.

RESTful HTTP communication is the most widespread data exchange solution (10), being the de-facto standard to implement microservices. Message queues is not as adopted as expected, in concordance with the lack of proposals adopting asynchronous interaction models. Regarding interfaces, their ad-hoc definition (7) is the rule. This suggests not only that microservices are being used in-house, with contracts negotiated between different teams/people inside the company, but also that they are not supposed to be reused but to be (re)developed entirely from scratch to fulfill new requirements. The recent efforts on standardizing RESTful APIs through OpenAPI specifications[2] seem interesting and also applicable to specify microservices [51].

[2] https://www.openapis.org/.

Finally, SQL is still the common choice for storage (5) even considering the benefits and hype for NoSQL databases (5 among the different options). This can be related to the fact that brownfield approaches inherit legacy databases and their migration is not straightforward. This also opens a question mark regarding the splitting and migration of data to fully exploit microservices advantages of data governance and data locality. Finally, discovery approaches are not so common (4), which suggests that more research is needed in this topic, given its importance in microservice architectures [30].

Deployment appears as a broadly discussed topic in the literature. Public cloud is the de-facto standard for deploying microservice applications (9) which confirms the increasing adoption of XaaS platforms. For deployment management, several approaches (14) propose their own ad-hoc solutions. There seems to be a mistrust regarding built-in services of cloud providers, which sometimes result too rigid [52] or cumbersome to configure and adjust [50]. However, these solutions are growing in variety and usability (e.g., AWS offering around 1000 new features per year[3]), and we believe that they will become the standard to deploy and manage cloud microservices in the near future.

Runtime shows that containers and microservices seems to be the perfect marriage (16). Even though, virtual machines are also widespread (9). However, only one approach considered serverless functions, also known as Functions-as-a-Service or FaaS[4]. FaaS appeared as a disruptive alternative that delegates the management of the execution environment of application functionality (in the form of stateless functions) to the infrastructure provider [46]. However these new solutions bring together new challenges[5], among others: determine the sweetspots where running code in a FaaS environment can deliver economic benefits; automatically profile existing code to offload computation to serverless functions; bring adequate isolation among functions; determine the right granularity to exploit data and code locality; and provide methods to handle state (given that functions are stateless by definition).

Only one approach provides the full control loop to manage microservice applications at runtime, while the vast majority provide monitoring facilities (8) with other purposes rather than self-adaptation (e.g., profiling, verification, service discovery). Verification & Validation at runtime is not yet widespread among microservices approaches, which calls for further research. Some approaches provide static V&V at model level (4), while a few provide dynamic testing and monitoring.

Crosscutting Concerns. Regarding availability and resilience, a few approaches (3) use resilience patterns[6] such as circuit breaker and bulkhead, while others provide their ad-hoc availability solutions [7]. Reliability is another aspect

[3] https://techcrunch.com/2016/12/02/aws-shoots-for-total-cloud-domination/.
[4] https://martinfowler.com/articles/serverless.html.
[5] https://blog.zhaw.ch/icclab/research-directions-for-faas/.
[6] http://microservices.io/patterns/.

that calls for coverage (4). The challenges regarding reliability come from the microservices integration mechanisms: network integration and message passing is unreliable [45]. Maintainability is not particularly addressed (3), even if in practice an abuse of the freedom of choice (polyglot languages and persistence) could result in a chaos in the system and make it even unmaintainable [4]. Consequently, it is important to investigate how standards, good practices, processes and frameworks can help to organize (and automate) microservices maintainability.

Security is not extensively addressed (3), even though the microservices ecosystem makes monitoring and securing networks very challenging due to the myriad of small, distributed and conversational components: microservices are often designed to completely trust each other, therefore the compromise of a single microservice may bring down the entire application [31]. The main ongoing trends in security are either monitoring techniques for SDNs inspired by their physical counterparts TAP (Test Access Point) and SPAN (Switch Port Analyzer), which can then be combined with a policy enforcement infrastructure [31]; or application-based security approaches[7], which gather information to build ad-hoc application profiles and then use them to detect anomalous patterns. Surprisingly, only a few approaches addressed scalability, with four of them relying in the cloud vendor to achieve it. Again, the adoption of serverless functions can go one step beyond on this concern, since once deployed, the cost and effort in operation, scaling and load balancing these functions are reduced to zero.

Finally, various approaches provided tool support for the different activities. A few of them for programming activities, mostly relying in well known IDEs as common solution. For the rest of the lifecycle, design activity (6) is supported, for example, through an ad-hoc PaaS [19], a decomposition tool based on cross-cutting concerns [20], and a design tool to define multi-tenant BPEL-based microservices [26]. Testing (6) is supported through a resilience testing framework [21], and a tool to generate and manage reusable acceptance tests [27]. Finally, Deployment/Operation (7) is supported through a reconfigurator for the ad-hoc language Jolie [17], and a tool for automatic setup of multi-cloud environments for microservices [29].

Organizational Aspects are not fully or explicitly addressed yet, with 3 approaches mentioning the adoption of DevOps (which implies an organizational rewiring, equivalent to the adoption of Agile methodologies) and other 3 adopting only certain key practices (Continuous Delivery, Integration, Management). It would be interesting to link more explicitly microservices with the DevOps movement. DevOps seems to be a key factor in the success of this architectural style [4], by providing the necessary organizational shift to minimize coordination among the teams responsible for each component and removing the barriers for an effective, reciprocal relationship between the development and operations teams.

[7] E.g., Netflix Fido – https://github.com/Netflix/Fido.

Additionally, the literature reports different socio-technical patterns to ease organizational rewiring [49], which can pave the ground for the transition towards microservices. For example, Sociotechnical-Risks Engineering, where critical architecture elements remain tightly controlled by an organization and loosely coupled with respect to outsiders; or Shift Left, where organizational and operational concerns (for example, development-to-operations team mixing) are addressed earlier ("left") in the life cycle toward architecting and development.

5 Conclusions

Microservices architectures are fairly new, but their hype and success is undeniable. This paper presented a preliminary analysis framework that captures the fundamental understanding of microservices architectures in the form of a taxonomy of concepts, encompassing the whole microservices lifecycle, as well as organizational aspects. This framework is necessary to enable effective exploration, understanding, assessing, comparing, and selecting microservice-based models, languages, techniques, platforms, and tools. We carried out an analysis of state of the art approaches related to microservices using the proposed taxonomy to provide a holistic perspective of available solutions.

Additionally, from the results of the literature analysis, we identified open challenges for future research. Among them, the early use of resilience patterns to design fault-tolerant microservice solutions, the standardization of interfaces, and the development of asynchronous microservices. Special attention should be given to the latent use of serverless architectures to deploy and manage microservices. They have the potential to become the next evolution of microservices [53], to event-driven, asynchronous functions, because the underlying constraints have changed, costs have reduced, and radical improvements in time to value are possible.

References

1. Lewis, J., Fowler, M.: Microservices (2014). http://martinfowler.com/articles/microservices.html
2. Fowler, M.: Monolith first (2015). http://martinfowler.com/bliki/MonolithFirst.html
3. Newman, S.: Building Microservices. O'Reilly Media Inc., Newton (2015)
4. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Softw. **33**(3), 42–52 (2016)
5. Richardson, C.: Microservices architecture (2014). http://microservices.io/articles/whoisusingmicroservices.html
6. George, F.: Challenges in implementing microservices (2015). http://gotocon.com/dl/goto-amsterdam-2015/slides/FredGeorge_ChallengesInImplementingMicroServices.pdf
7. Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., Bohnert, T.M.: Self-managing cloud-native applications: design, implementation, and experience. In: Future Generation Computer Systems, vol. (2016, in Press)

8. Richards, M.: Microservices Service-Oriented Architecture. O'Reilly Media, Newton (2015)
9. Wilde, N., Gonen, B., El-Sheikh, E., Zimmermann, A.: Approaches to the evolution of SOA systems. In: El-Sheikh, E., Zimmermann, A., Jain, L.C. (eds.) Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures. ISRL, vol. 111, pp. 5–21. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40564-3_2
10. Hassan, S., Bahsoon, R.: Microservices and their design trade-offs: a self-adaptive roadmap. In: IEEE International Conference on Services Computing (SCC), pp. 813–818. IEEE (2016)
11. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. Technical report, Version 2.3 EBSE Technical Report. EBSE, sn (2007)
12. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, p. 38. ACM (2014)
13. Kratzke, N.: About microservices, containers and their underestimated impact on network performance. In: Proceedings of CLOUD COMPUTING 2015, pp. 165–169 (2015)
14. Bogner, J., Zimmermann, A.: Towards integrating microservices with adaptable enterprise architecture. In: 2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW), pp. 1–6, September 2016
15. Ciuffoletti, A.: Automated deployment of a microservice-based monitoring infrastructure. Procedia Comput. Sci. **68**, 163–172 (2015)
16. Florio, L., Di Nitto, E.: Gru: an approach to introduce decentralized autonomic behavior in microservices architectures. In: 2016 IEEE International Conference on Autonomic Computing (ICAC), pp. 357–362. IEEE (2016)
17. Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J., Montesi, F.: Self-reconfiguring microservices. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 194–210. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30734-3_14
18. Gadea, C., Trifan, M., Ionescu, D., Ionescu, B.: A reference architecture for real-time microservice API consumption. In: Proceedings of the 3rd Workshop on Cross-Cloud Infrastructures & Platforms, p. 2. ACM (2016)
19. Guo, D., Wang, W., Zeng, G., Wei, Z.: Microservices architecture based cloudware deployment platform for service computing. In: 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 358–363. IEEE (2016)
20. Gysel, M., Kölbener, L., Giersche, W., Zimmermann, O.: Service cutter: a systematic approach to service decomposition. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol. 9846, pp. 185–200. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44482-6_12
21. Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V.: Gremlin: systematic resilience testing of microservices. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), pp. 57–66. IEEE (2016)
22. Kecskemeti, G., Marosi, A.C., Kertesz, A.: The ENTICE approach to decompose monolithic services into microservices. In: 2016 International Conference on High Performance Computing & Simulation (HPCS), pp. 591–596. IEEE (2016)
23. Liu, D., Zhu, H., Xu, C., Bayley, I., Lightfoot, D., Green, M., Marshall, P.: CIDE: an integrated development environment for microservices. In: IEEE International Conference on Services Computing (SCC), pp. 808–812. IEEE (2016)

24. Derakhshanmanesh, M., Grieger, M.: Model-integrating microservices: a vision paper. In: Software Engineering (Workshops), pp. 142–147 (2016)
25. Safina, L., Mazzara, M., Montesi, F., Rivera, V.: Data-driven workflows for microservices: genericity in jolie. In: IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 430–437. IEEE (2016)
26. Nikol, G., Träger, M., Harrer, S., Wirtz, G.: Service-oriented multi-tenancy (SOMT): enabling multi-tenancy for existing service composition engines with Docker. In: 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 238–243. IEEE (2016)
27. Rahman, M., Gao, J.: A reusable automated acceptance testing architecture for microservices in behavior-driven development. In: 2015 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 321–325. IEEE (2015)
28. Savchenko, D.I., Radchenko, G.I., Taipale, O.: Microservices validation: mjolnirr platform case study. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 235–240. IEEE (2015)
29. Sousa, G., Rudametkin, W., Duchien, L.: Automated setup of multi-cloud environments for microservices-based applications. In: 9th IEEE International Conference on Cloud Computing (2016)
30. Stubbs, J., Moreira, W., Dooley, R.: Distributed systems of microservices using docker and serfnode. In: International Workshop on Science Gateways (IWSG), pp. 34–39. IEEE (2015)
31. Sun, Y., Nanda, S., Jaeger, T.: Security-as-a-service for microservices-based cloud applications. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 50–57. IEEE (2015)
32. Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A., et al.: Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 179–182. IEEE (2016)
33. Ben Hadj Yahia, E., Réveillère, L., Bromberg, Y.-D., Chevalier, R., Cadot, A.: Medley: an event-driven lightweight platform for service composition. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 3–20. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_1
34. Bak, P., Melamed, R., Moshkovich, D., Nardi, Y., Ship, H., Yaeli, A.: Location and context-based microservices for mobile and internet of things workloads. In: 2015 IEEE International Conference on Mobile Services (MS), pp. 1–8. IEEE (2015)
35. Levcovitz, A., Terra, R., Valente, M.T.: Towards a technique for extracting microservices from monolithic enterprise systems. In: 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM), pp. 97–104 (2015)
36. Meinke, K., Nycander, P.: Learning-based testing of distributed microservice architectures: correctness and fault injection. In: Bianculli, D., Calinescu, R., Rumpe, B. (eds.) SEFM 2015. LNCS, vol. 9509, pp. 3–10. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-49224-6_1
37. Viennot, N., Lécuyer, M., Bell, J., Geambasu, R., Nieh, J.: Synapse: a microservices architecture for heterogeneous-database web applications. In: Proceedings of the Tenth European Conference on Computer Systems, p. 21. ACM (2015)
38. Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M., Steinder, M.: Performance evaluation of microservices architectures using containers. In: 2015 IEEE 14th International Symposium on Network Computing and Applications (NCA), pp. 27–34. IEEE (2015)

39. Garriga, M., Flores, A., Cechich, A., Zunino, A.: Web services composition mechanisms: a review. IETE Tech. Rev. **32**(5), 376–383 (2015)
40. Garriga, M., Mateos, C., Flores, A., Cechich, A., Zunino, A.: Restful service composition at a glance: a survey. J. Netw. Comput. Appl. **60**, 32–53 (2016)
41. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, Boston (2004)
42. Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T.: Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications. Microsoft Patterns & Practices (2014)
43. Casale, G., Chesta, C., Deussen, P., Di Nitto, E., Gouvas, P., Koussouris, S., Stankovski, V., Symeonidis, A., Vlassiou, V., Zafeiropoulos, A., et al.: Current and future challenges of software engineering for services and applications. Procedia Comput. Sci. **97**, 34–42 (2016)
44. Schermann, G., Cito, J., Leitner, P.: All the services large and micro: revisiting industrial practice in services computing. In: Norta, A., Gaaloul, W., Gangadharan, G.R., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9586, pp. 36–47. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-50539-7_4
45. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow, arXiv preprint arXiv:1606.04036 (2016)
46. Roberts, M.: Serverless architectures (2016). http://martinfowler.com/articles/serverless.html
47. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_1
48. Bass, L.: Software Quality Assurance In Large Scale and Complex Software-intensive Systems, vol. 1. Morgan Kauffmann, San Francisco (2015). Ch. Forewords by Len Bass
49. Tamburri, D.A., Kazman, R., Fahimi, H.: The architect's role in community shepherding. IEEE Softw. **33**, 70–79 (2016)
50. Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Serverless computation with openlambda. Elastic **60**, 80 (2016)
51. Baresi, L., Garriga, M., De Renzis, A.: Microservices identification through interface analysis. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds.) ESOCC 2017. LNCS, vol. 10465, pp. 19–33. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67262-5_2
52. Baresi, L., Guinea, S., Leva, A., Quattrocchi, G.: A discrete-time feedback controller for containerized cloud applications. In: ACM Sigsoft International Symposium on the Foundations of Software Engineering (FSE). ACM (2016, accepted for publication)
53. Cockroft, A.: Evolution of business logic from monoliths through microservices, to functions (2017). https://goo.gl/H6zKMn