Journal of Computer Information Systems

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/ucis20

# Architecting Microservices: Practical Opportunities and Challenges

Saša Baškarada , Vivian Nguyen & Andy Koronios

**To cite this article:** Saša Baškarada , Vivian Nguyen & Andy Koronios (2020) Architecting Microservices: Practical Opportunities and Challenges, Journal of Computer Information Systems, 60:5, 428-436, DOI: 10.1080/08874417.2018.1520056

**To link to this article:** https://doi.org/10.1080/08874417.2018.1520056

JOURNAL OF COMPUTER INFORMATION SYSTEMS
2020, VOL. 60, NO. 5, 428–436
https://doi.org/10.1080/08874417.2018.1520056



# Architecting Microservices: Practical Opportunities and Challenges

Saša Baškarada[a], Vivian Nguyen[b], and Andy Koronios[a]

[a]University of South Australia, Adelaide, Australia; [b]Defence Science and Technology Group, Fishermans Bend, Australia

**ABSTRACT**

Contemporary highly dynamic technology and business environments, coupled with digitally savvy customers, are forcing both private and public organizations to continuously innovate and update their Information and Communication Technology (ICT) service offerings. In order to decrease the development cycles and ensure continuous delivery, many organizations are adopting new practices aimed at unifying software development and operations (i.e., DevOps). However, monolithic architectures of many large systems underpinning ICT services are severely restricting the effectiveness of such efforts. Although the latest architectural trend, microservices, is promising to solve many of the problems associated with monolithic software architectures, there is significant disagreement on when microservice architecture should be adopted, as well as how it may be best implemented. Given the limited empirical research on the topic, this paper identifies and discusses a range of opportunities and challenges associated with the adoption and implementation of microservices. The findings presented in the paper have been derived from in-depth interviews with 19 ICT architects with significant experience in large corporate systems, middleware, service oriented architectures, and, to a somewhat more limited extent, microservices.

**KEYWORDS**

Microservices; service oriented architecture (SOA); DevOps; orchestration; choreography

## Introduction

Organizational agility, the capacity to flexibly respond to changes in the environment by quickly adjusting product and service offerings, is increasingly becoming critical to achieving sustained competitive advantage. [1] Information and Communication Technologies (ICT) are a key enabler of organizational agility, not just in the technology sector, but more broadly (e.g., Uber and Airbnb). Many organizational capabilities and functions are underpinned by large monolithic software systems, including Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM); where a monolith has been defined as "a software application whose modules cannot be executed independently".[2] As such systems grow and become more complex, they also become more difficult to change. As a result, large monolithic software systems may slow organizational ability to quickly respond to changes in the environment; e.g., changes in customer and competitor behaviors. In other words, rather than enabling, large monolithic software systems may hinder organizational agility. [3]

One of the most recent architectural trends, microservices (see Table 1), has been proposed as a potential solution to the above-mentioned problem.[5–7] The term "microservices" was reportedly first discussed at a workshop of software architects near Venice in May 2011.[8] While there is arguably still no precise definition, in contrast to monolithic software architectures, microservices are small applications (generally less than a couple of thousand lines of code) with a single responsibility (a functional, non-functional, or cross-functional requirement) that can be independently deployed, scaled, and tested.[9] They support agility since they can be independently developed and modified using different programming languages and product stacks.[10] Some of the more popular technologies used for implementing and deploying microservice include Docker, Node.js, MySQL, PostgreSQL, Java, MongoDB, and PHP.[11] A microservice architecture has been described as a "distributed application where all its modules are microservices".[2]

While each individual microservice is relatively simple compared to a large monolithic application, such a comparison is misleading since the complexity resides in the whole system composed of microservices. Due to their distributed nature, such systems may be even more complex than monolithic applications.[8] Furthermore, since each microservice may have a separate back-end, and be developed by a different team on a different technology stack, the resulting complexity may not be manageable without a DevOps capability.[12]

DevOps is a set of practices aimed at unifying software development (Dev) and IT operations (Ops) that make heavy use of automation and monitoring to decrease the development cycles while ensuring high software quality.[13] Continuous delivery is a DevOps practice aimed at automating the process of software testing and deployment into production environments, while continuous monitoring is a DevOps practice aimed at providing developers with real-time performance data.[5]

A major difference between microservices and traditional Service-Oriented Architecture (SOA) is that microservices do not make use of an Enterprise Service Bus (ESB), a defining feature of SOA.[9] Instead, they generally

**CONTACT** Saša Baškarada baskarada@gmail.com University of South Australia, North Terrace, Adelaide, SA 5000 Australia
Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/ucis.

**Table 1.** Monolithic vs. microservice architecture; developed from [2,4].

| Monolithic Architecture | Microservices |
|---|---|
| Due to their large size and complexity, they are difficult to debug, maintain, and evolve. | Due to their small size and low complexity, they are relatively easy to debug, maintain, and evolve. |
| Development and testing is slowed-down as, due to the dependencies between modules, making changes to one module may require restarting the whole application. | Accelerated development and testing enabling continuous delivery as, due to their independence, each microservice can be separately changed and deployed. |
| Since various modules may have different resource requirements, deploying monolithic applications in a single environment generally leads to sub-optimal performance. | Individual microservices may be deployed and optimized in separate environments. |
| Scalability is limited since it is not possible to scale only a subset of modules. | Each microservice can be independently scaled leading to improved resource utilization. |
| Since they are built on the same technology stack, development flexibility is limited. | Each microservice may be developed using a different technology stack, leading to greater flexibility. |

communicate directly with each other through standards like HTTP, REST, and JSON. [12] Another key difference between SOA and microservices is that SOA generally exposes services from existing monolithic applications. Since such services are tightly integrated with the monolithic applications, they cannot be independently changed, scaled, and deployed.

Although Amazon, Netflix, and the Guardian are frequently highlighted as early adopters of microservices, most of the academic research on microservices is still in a formative stage, [2,14,15] and at a relatively low technology readiness level (formulated, validated, or at most demonstrated in a lab).[16] Furthermore, empirical research on microservices is limited.[11] As a result, this paper answers calls for qualitative studies with practitioners aimed at better understanding the state of the practice on microservices.[14,16]

While prior studies have explored a range of technical issues associated with the implementation of microservices, including communication/integration, service discovery, performance, fault tolerance, security, tracing and logging, application performance monitoring, and deployment operations,[15] this paper aims to contribute to the literature by also exploring organizational, cultural, and institutional factors. Specifically, this study aims to answer the following research question:

*What are the practical opportunities and challenges associated with the adoption and implementation of microservice architecture?*

The rest of the paper is organized as follows. Section two provides an overview of related research, and section three describes the research method employed. Section four presents the findings by discussing a range of opportunities and challenges associated with the adoption and implementation of microservice architecture. Section five concludes the paper by discussing the research limitations and identifying opportunities for future research.

## Literature review

Research on microservices is recent and limited.[2,11,14,15] A recent literature review identified 71 relevant studies, most of which were published after 2014. [16] Another recent literature review found that most of the published papers on microservice are non-empirical. [11] Although there is no formal definition, a common view is that microservices are "cohesive, independent processes interacting via messages".[2] Common characteristics of microservice architecture include:

- componentization via services—the application is broken-down into independent services that run in different processes, share no resources, and communicate via lightweight messaging;
- being organized around business capabilities—built by cross-functional teams;
- focus on products rather than projects—cross-functional teams are responsible for the whole lifecycle of a microservice;
- smart endpoints and dumb pipes—microservices communicate via RESTful interfaces and avoid any complex orchestration and choreography busses;
- decentralized governance and data management—microservices may be developed using different technology stacks with no common data model;
- infrastructure automation—microservice architecture generally presupposes extensive experience with continuous integration and delivery;
- design for failure—given that services can fail at any time, quickly detecting failures and automatically restoring services becomes important; and
- evolutionary design—microservices can be easily substituted. [8]

As a result of their independence, each microservice runs in a different process (frequently deployed on different virtual machines), and has an independent back-end database.[3] It is important to clarify that separate data stores do not necessarily imply separate database servers. In other words, different microservices could share a database server as long as they are using different schemas. Individual microservices may be dynamically located via service registries and discovery mechanisms.[5,17] While, in theory, the strict autonomy of microservices is thought to enforce modularization, how well this actually works in practice is a still unanswered empirical question.[8]

Since each microservice has its own data store, there is no single source of truth, and maintaining data consistency may become problematic.[12] While some published prototypes have avoided this problem by implementing microservices with a shared database,[17] such solutions are generally seen as being out of pattern with the share-nothing microservices architecture. Implementing complex business processes through microservices may require the addition of a message broker. Such asynchronous messaging may lead to inconsistencies in individual data stores.[12]

The effectiveness of microservice architecture is largely determined by the skills of the cross-functional implementation teams.[7] Implementing and sustaining distributed systems is inherently more difficult than monolithic applications.[2] For instance, microservices developers and architects need be aware of the fallacies of distributed computing, including assuming that the network is reliable, that latency is zero, that the bandwidth is infinite, that the network is secure, that topology doesn't change, that there is one administrator, that transport cost is zero, and that the network is homogeneous.[18] Thus, while more experienced teams may be able to derive many of the advantages associated with microservices, less experienced/skilled teams may be more effective at implementing traditional monolithic systems.[8]

Although some have argued that most organizations implementing microservices (e.g., Netflix and Amazon) start by decomposing existing large monolithic applications rather than building new applications comprised of microservices,-[9,19] others have found that decomposing monolithic applications into microservices is one of the key barriers to the adoption of microservices.[20] For instance, while describing the migration of an on-premises commercial Mobile Backend as a Service (MBaaS) platform to a microservices architecture in the cloud, Balalaie, Heydarnoori[5] highlight a number of key challenges associated with such efforts, including architectural refactoring, incremental migration, and the adoption of DevOps practices (continuous integration, delivery, and monitoring).

Taibi, Lenarduzzi[20] interviewed 21 practitioners, who had adopted microservice architecture, on their motivations for migrating to microservices, as well as on the pros and cons of microservices versus monolithic applications. They found that organizations tend to migrate to microservice architecture to improve software maintainability and scalability, as well as to support DevOps. Many organizations also reported adopting microservices simply because other organizations were doing it. Taibi, Lenarduzzi[20] also identified a number of key issues when migrating from monolithic software to microservices, including the difficulty of decomposing the monolith. Decomposing of the data layer was identified as being particularly difficult, so much so that many interviewees reported adopting microservices with a centralized legacy database.

Microservices architecture is closely tied with continuous delivery through DevOps, as well as the use of container technologies (lightweight virtualization packaging mechanism) and Platform as a Service (PaaS) cloud infrastructure.[5,14] However, recent experiments have indicated that container technologies like Docker may have potentially significant performance impact of 10%–20%. [21]

Implementing complex business processes that span the boundaries of individual microservices is no trivial matter. Although it is generally recommended that microservices should avoid complex orchestration,[8], exactly how this may best be accomplished in practice is not well understood. Orchestration, which originates with SOA, relies on a centralized process/service that coordinates all other microservices partaking in the execution of a business process. Since such orchestrators assume all responsibility for command and control, this approach increases coupling and complexity in the system. Instead, the recommendation is to employ choreography, which distributes the coordination responsibilities to individual microservices, generally in the context of event driven architecture.[3] However, since individual microservices are not prohibited from making calls to other microservices, it is not clear at what point a regular microservice becomes an orchestrator. Furthermore there is still significant debate on whether orchestration can be entirely avoided, especially when implementing business processes with complex states.[19]

## Methods

This paper adopts evidence-based software engineering (EBSE), which recommends starting with a research question, identifying the best data to answer that question, analyzing and critically evaluating the data collected, integrating the critical appraisal with our expertise and stakeholders' values and circumstances, and finally critically reflecting on our execution of the study with a view to identifying lessons for future research.[22] We chose to frame this study in the context of EBSE because of the strong grounding of the research question in the practice of software engineering; e.g., most of the relevant research and discussion has so far taken place in the software engineering community. Nevertheless, while framing this research as EBSE, we need to acknowledge significant methodological overlap with qualitative research in information systems and social sciences more broadly. Given that we were more interested in exploring the *how* and *why* relating to practical opportunities and challenges associated with adoption and implementation of microservice architecture, rather than in questions like *who, what, where, how many*, and *how much*, a qualitative research approach was deem suitable.

Accordingly, qualitative data were collected through semi-structured interviews with 19 ICT architects. The number of interviewees is broadly in line with recommendations from methodological literature, which notes that although the appropriate number of interviews depends on the the phenomenon under investigation, the scope of the study, and the timeframe available,[23] studies should generally aim for at least 15 interviews.[24] All participants were interviewed face-to-face, and each interview generally lasted between 30 and 60 minutes. Most interviews were recorded (several participants did not agree to being recorded) and subsequently transcribed.

All interviewees had more than 10 years of professional ICT experience, and more than five years of experience as domain, solution, and/or enterprise architects (see Table 2). While there is a significant overlap between the different roles, enterprise architects generally aim to simplify and optimize the entire ICT landscape of an organization, domain architects specialize in particular industries (e.g., telecommunication) or broad technology types (e.g., cloud computing), and solution architects generally specialize in particular products or technology stacks (e.g., Big Data analytics). Most of the interviewees had had a range of roles in both private and public organizations, and at the time of the study, eight of the interviewees were employed as consultants/contractors, seven were employed in public

**Table 2.** List of Interviewees.

| # | Position | ICT Experience | ICT Architecture Experience | Employment Type |
|---|---|---|---|---|
| 1 | enterprise architect | > 25 years | > 15 years | public organization |
| 2 | enterprise architect | > 30 years | > 15 years | public organization |
| 3 | enterprise architect | > 20 years | > 10 years | public organization |
| 4 | solution architect | > 10 years | > 5 years | consultant/contractor |
| 5 | domain architect | > 15 years | > 5 years | public organization |
| 6 | solution architect | > 20 years | > 10 years | consultant/contractor |
| 7 | solution architect | > 30 years | > 15 years | public organization |
| 8 | domain architect | > 10 years | > 5 years | private organization |
| 9 | enterprise architect | > 30 years | > 15 years | public organization |
| 10 | solution architect | > 20 years | > 10 years | consultant/contractor |
| 11 | domain architect | > 15 years | > 10 years | private organization |
| 12 | solution architect | > 10 years | > 5 years | private organization |
| 13 | domain architect | > 20 years | > 10 years | consultant/contractor |
| 14 | solution architect | > 10 years | > 5 years | consultant/contractor |
| 15 | solution architect | > 15 years | > 5 years | consultant/contractor |
| 16 | solution architect | > 20 years | > 10 years | consultant/contractor |
| 17 | solution architect | > 10 years | > 5 years | consultant/contractor |
| 18 | solution architect | > 20 years | > 10 years | public organization |
| 19 | enterprise architect | > 25 years | > 10 years | private organization |

organizations, and the remaining four were permanent full-time employees of large private organizations.

As this study adopts a qualitative rather than a quantitative approach, there was no need to select a statistically representative sample. Instead, participants were selected based on their subject matter expertise, availability, and willingness to participate in the study. All interviewees were identified through authors' professional networks. All of the interviewees had extensive experience with large corporate systems, middleware, service-oriented architectures, and to a somewhat more limited extent microservices. Although a set of predefined questions was used to guide the interviews (see Table 3), the semi-structured approach provided the authors with the flexibility to refocus questions, or prompt for more information, when an interesting or novel topic emerged. [24] The high-level interview questions detailed in Table 3 were largely based on the main research question. The first four questions were aimed at building rapport with the interviewees as well as at establishing their level of expertise. Questions four and five were directly derived from the main research question, and the final question is a standard closing question generally used in qualitative studies.

The constant comparative method of qualitative data analysis was used to identify key factors, themes, and patterns.[25,26] Qualitative data analysis in general, and the constant comparative method in particular, aim to add structure to unstructured text. For instance, thematic analysis, where categories are grouped into overarching themes, is one of the most frequently employed approaches to qualitative data analysis.[27] Such structure, which is introduced in stages, forms the basis of any resulting frameworks, theories, and explanations.[28] The constant comparative method was first outlined by Glaser [25] in a seminal paper that highlighted the importance of simultaneous coding and analysis. The method initially involves the identification of as many categories of analysis as possible. According to Glaser, this data categorization directly leads to the identification of theoretical properties relating to categories. Such theoretical properties may include causes, conditions, consequences, dimensions, types, and processes. Once identified, categories and their properties may then be integrated into a preliminary theory. While Glaser and Holton[29] explain what needs to be compared, exactly how this is to be done remains relatively ambiguous. When comparing coded items, other scholars have suggested looking for equivalence, similarity and difference.[28]

We followed three sequential stages: open coding, axial coding, and selective coding.[30] During open coding, we broke down the data into codes and categories. During axial coding, we explored and explained the relationships between those codes and categories. During selective coding, we identified a couple of central categories (opportunities and challenges), related other codes and categories to that core, and explained them in terms of that core.

The coding was iteratively performed by the lead researcher and reviewed by the co-authors. Following each review, any coding-related disagreements were discussed by the researchers and the codebook was updated accordingly.[31] While no quantitative inter-coder reliability assessments were undertaken, inter-coder reliability was not considered problematic as the number of coding-related disagreements reduced (and inter-coder consensus increased) with each iteration. In order to ensure a level of face-validity, the outcomes of the analysis were reviewed by several subject matter experts.

**Table 3.** Interview Questions.

| # | Question |
|---|---|
| 1 | Could you please tell us about your current role (position, organization, etc.)? |
| 2 | Could you please tell us about your professional ICT experience? |
| 3 | Could you please tell us about your professional ICT architecture experience? |
| 4 | Could you please tell us about your experience with microservices and related architectural trends (e.g., SOA)? |
| 5 | In your opinion, what are the advantages of microservice vs. monolithic architectures? |
| 6 | What are some of the key practical challenges associated with the adoption and implementation of microservices? |
| 7 | Is there anything else you would like to add? |

## Results and discussion

Although all 19 interviewees had extensive experience with large corporate systems, middleware, and service-oriented architectures, their experience with microservices was significantly more limited. This can be explained by the newness and as yet relatively limited adoption of the architectural trend. While all interviewees had some exposure to microservices, only seven had implemented major production systems through microservice architecture. Three of those had implemented new applications, while the other four had decomposed existing monoliths into microservices. Nine interviewees had experimented with a small number of relatively simple microservices, either in sandpit and/or production environments. The remaining three interviewees had considered the adoption of microservices, but for various reasons never proceeded with implementation. Based on our discussions with the interviewees, we identified two main opportunities and 10 key challenges associated with the adoption and implementation of microservice architecture.

### Opportunities

Perhaps unsurprisingly, the opportunities associated with the adoption of microservice architecture identified by the interviewees agreed with the purported benefits promoted by various vendors and enthusiasts. They were centered around improving agility in software development (Dev) well as around optimizing scalability in IT operations (Ops).

### Development/deployment agility

All of the interviewees highlighted the lack of agility associated with large monolithic corporate systems as a major issue. Such systems and the associated governance procedures are seen as barriers to modern digital business initiatives. While microservices were acknowledged as a potential solution to some of the problems associated with simpler monoliths, none of the interviewees suggested that complex corporate systems of record (e.g., ERP and CRM) should be decomposed into microservice architecture. Large corporate systems of record generally support complex business processes that may span the entire organization (as well as external suppliers and service providers). As a result, individual modules of such systems are highly interdependent. Furthermore, corporate systems of record should be fairly stable, without a need for frequent changes. Instead, the interviewees suggested that microservice architecture may be more suitable for corporate systems of engagement (used for communication and collaboration), systems of differentiation (bespoke systems underpinning organizational business models), and systems of innovation—experimental systems developed to address ad hoc business requirements that may be incorporated into future systems of differentiation (see Figure 1). Using this terminology, five of the architects interviewed had implemented corporate systems of differentiation using microservice architecture, while 11 had utilized microservices for systems of innovation. None of the interviewees had seriously considered applying microservice architecture to reengineer corporate systems of record.

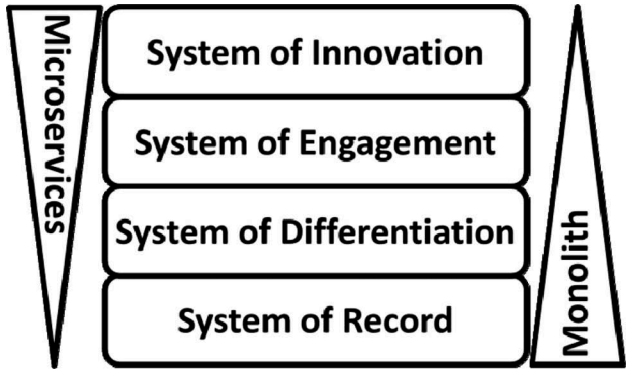


Figure 1. Types of Corporate Systems.

The opportunity to use heterogeneous technology stacks was singled out by most interviewees as almost an end in itself. Existing monoliths are generally based on relatively fixed technology stacks, some of which may be relatively old (e.g., several of the interviewees had recently worked on monoliths written in COBOL). Relatively old technology stacks may not provide the best system performance and/or user experience, and relatively rare technologies may be expensive and difficult to maintain (e.g., the supply of COBOL developers is much more limited that the supply of Java developers). Thus, the possibility for heterogeneous technology stacks in the microservice architecture was seen as a significant enabler of experimentation and innovation. Different teams and/or developers could experiment with different technologies and the best technologies could be identified through the process of natural selection. Since individual developers may be more productive in particular technologies, giving developers the freedom to choose their own tools was also seen as a boost to productivity.

In addition to the higher agility in software development, the interviewees also stressed more agile softer deployment. Since microservices can be independently deployed, there is no need to deploy the entire system/monolith each time any of the microservices are updated. Such approach is also less risky, since any problematic deployments can also be rolled back relatively quickly.

### Operational scalability

Besides development/deployment agility, operational scalability was identified as another key potential benefit of microservice architecture. Most interviewees highlighted the difficulty of appropriately scaling monolithic applications. Given the ubiquitous adoption of cloud infrastructure and platform services, scaling out (adding extra virtual servers) was preferred to scaling up (adding extra components like RAM and CPU to existing servers). Although different application tiers (presentation, logic, and data) may be scaled separately, this was seen as insufficiently flexible as different modules from each tier may have different scaling requirements (see Figure 2). Moreover, several interviewees were of

the view that microservice architecture was well suited to serverless (function as a service computing); e.g. Azure Functions and Amazon Web Services Lambda. Serverless computing aims to simplify IT operations by abstracting away virtual machines, virtual network infrastructure, and scaling.

### Challenges

Most of the interviewees had significant reservations regarding the adoption of microservice architecture. These included the lack of relevant skills, the reliance on Software as a Service (SaaS) and commercial off the shelf (COTS) on-premises applications, organizational culture and structure, governance, difficulties associated with monolith decomposition/refactoring, master data management, orchestration and choreography, testing, and performance.

### Lack of relevant skills

While developing individual microservices may not presuppose any particular skillsets (since they may be developed with heterogeneous technology stacks), effectively implementing and efficiently managing a large application developed using microservice architecture requires significant advanced skills in distributed system development and DevOps. Given that individual microservices can be developed and deployed independently, and considering that non-trivial applications may be composed of a large number of microservices, the resulting architectural complexity (continuous development/delivery, monitoring, scaling, recovery, etc.) cannot be managed manually. Popular DevOps tools include Jenkins (for continuous integration), Docker (for containerized deployment), Docker Swarm (for container clustering and orchestration), and Kubernetes (for load balancing, scaling, and discovery). Moreover, the assumption that, due to their relative simplicity, individual microservices are easy to develop is also misleading. Compared with large teams developing monoliths, where individual developers can specialize in particular technologies/features, microservice developers need to have a solid understanding of a range of non-functional aspects, including security, system fault-tolerance, microarchitecture recoverability, latency, and the like.

### Saas and COTS applications

Only a small number of the interviewees reported developing bespoke corporate applications (mainly systems of innovations and some systems of differentiation), and none of the interviewees reported developing bespoke systems of record and systems of engagement. In general, the interviewees were of the view that most organizations prefer buying established COTS products and/or subscribing to cloud-based SaaS solution than developing bespoke systems. Such reliance on monolithic COTS and SaaS application places obvious limits on the potential uptake of microservice architecture. Finally, in contrast to microservices, which imply decomposition of monoliths, SOA integrates perfectly well with monolithic COTS and SaaS products.

### Organizational culture

Since individual microservices are owned by individual developers and/or small teams, microservice architecture requires a significant amount of organizational trust. Each microservice developer/owner needs to trust that other developers/owners will produce and operate efficient and reliable microservices. Because of network effects, microservice architecture becomes more useful as more microservices partake in it. As a result, for microservice architecture to be effective, the organization needs to be accepting of microservices.

### Governance

The distributed nature of microservices requires a significant shift from traditional governance frameworks and processes. Governance of microservices architecture is by definition distributed. While individual owners are responsible for their microservices, they also need to consider potential impacts of changes on other dependent microservices. Given the distributed nature of microservice architecture, governance is

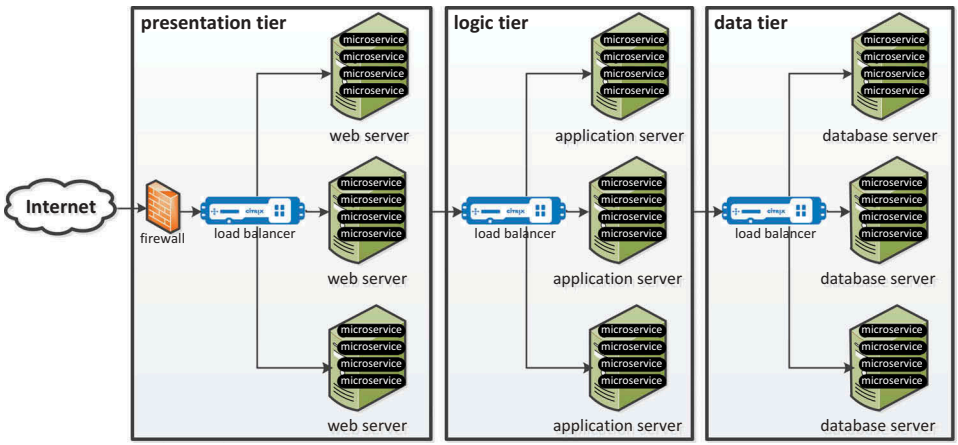


Figure 2. Three tier architecture.

generally ensured through interface/API contracts. Any changes to interfaces of microservices generally have significant broader effects. Although interface changes can be discouraged and minimized, they cannot be entirely avoided. As a result, changes to any interfaces need to be broadly communicated and coordinated with related microservices.

### Organizational structure

The fact that most IT departments are organized around "plan, build, run" silos was identified as perhaps the biggest obstacle to microservice architecture. Given that many (perhaps most) organizations predominantly rely on SaaS and COTS software suggests that traditional "plan, build, run" structures of IT departments are probably here to stay for some time. The fact that such organizational structures run counter to DevOps and microservice architecture philosophies was seen as a major barrier to effective adoption.

According to Conway's Law, "organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations".[32] For instance, the fact that more loosely-coupled organizations tend to produce more modular designs has found strong empirical support.[33]

DevOps and microservice architecture presuppose the control of the entire deployment pipeline, from development to operations. This is generally not the case in siloed IT departments (see Figure 3). Given that microservices and DevOps largely originated in IT organizations like Google, Amazon, and Netflix, whose business models revolve around IT services, there was some concern among the interviewees about how such structures could be effectively implemented in organizations whose business model does not revolve around IT. In such organizations, effective adoption of microservices and DevOps may require major structural changes to existing IT departments; for instance, by distributing relevant IT expertise among various business units.

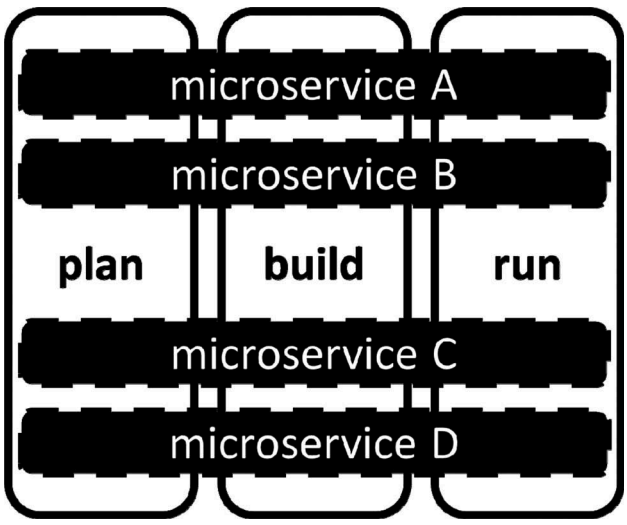


Figure 3. microservices across the plan, build, run organizational structure.

### Decomposition/refactoring

Decomposing large existing monoliths was seen by most interviewees as largely impractical, especially in the context of systems of record. The four interviewees who had decomposed existing monoliths into microservices confirmed the enormity of the task and suggested that in some cases it may even be easier developing a new application from scratch rather than decomposing an existing monolith. The interviewees noted that the lack of documentation and the fact that many of the original architects, designers, and developers had moved on, meant that the monolith had to be reengineered before it could be decomposed. Whether the result is worth the effort may be determined by the business criticality of the application in question. All four architects who reported decomposing an existing monolith into microservices had done it for a business critical system of differentiation.

### Master data management

Given that microservice architecture discourages a common data model, and encourages decentralized data management, interviewees expressed significant concerns regarding master data management. There was little understanding regarding how data consistency may best be ensured in microservice architecture, given that the same data may be stored in multiple databases in a range of formats. This can cause obvious problems with data updates. While it was acknowledged that using a centralized data store (which some of the interviewees had done with microservice-based systems of innovation) may alleviate some of these problems, there was also general agreement that this would introduce an unacceptable level of centralization that is counter to the spirit of decentralized microservice architecture.

### Orchestration and choreography

There was little consensus on when to use orchestration and when choreography in microservice architecture. While most of the interviewees had experience with SOA orchestration, very few had experience with event-based architectures and service choreography. And although most of the interviewees understood that microservice architecture encourages choreography over orchestration, there was a general consensus that some level of orchestration may be required with more complex business processes.

### Testing

While individual microservices may be independently developed, tested (analogous to traditional unit testing), and deployed, microservice architecture presupposes multiple interdependent/coordinating microservices. As a result, whenever an individual microservice is changed, it needs to be tested together with any other microservices that may be impacted. This may include integration, system, and acceptance testing. Furthermore, testing of distributed systems built using microservice architecture is inherently more difficult than testing of centralized monoliths. One of the key

difficulties associated with distributed systems is that they are inherently less stable than non-distributed systems, and that trying to anticipate all possible modes of failure is pretty much impossible. In addition, many of the non-functional failure modes may be difficult to automate in a DevOps microservice architecture, and when automated, comprehensively testing a large microservices application can be very slow.

### Performance

Given the distributed nature of microservice architecture, and especially the fact that what would have been method/procedure calls in a monolith are replaced with RESTful API calls over a network, many of the interviewees were concerned about potential performance penalties associated with microservice architectures. This concern was amplified by the fact that few microservice-based applications are currently available as COTS products with clearly specified performance parameters. Developing distributed microservice-based applications in-house was seen as particularly risky since estimating the likely performance of such applications was seen as being more difficult. Several interviewees raised a concern relating to reporting from microservice-based applications. Given that data are distributed across numerous microservices/databases, any complex reporting may require significant inter-microservice communication through RESTful APIs, which may introduce significant performance penalties.

## Conclusion

Microservice architecture is the latest architectural trend that has been significantly gaining in popularity over the last few years. While there is a lot of hype associated with microservices, and while many vendors and enthusiasts promote a range of architectural benefits purportedly associated with microservices, with the exception of a small number of frequently mentioned success stories (e.g., Amazon, Netflix, and the Guardian), few organizations seems to have refactored or developed from scratch large organizational systems of record or systems of differentiation using microservice architecture. Our study with 19 experienced ICT architects has found that there is a lot of professional uncertainty associated with adoption of microservice architecture. Although most of the interviewees understood purported benefits of microservice architectures, namely improved development/deployment agility and operational scalability, they also had significant reservations relating to a range of technical and organizational aspects of microservice adoption.

While some of the opportunities and challenges discussed in this paper have also been identified in other studies, others, to the best of our knowledge, provide a novel contribution to the microservices literature. Factors discussed in other studies include development/deployment agility,[10] operational scalability,[9] lack of relevant skills,[7] master data management,[12] decomposing/refactoring,[20] and orchestration and choreography.[19] Factors not previously considered in the context of microservices include SaaS and COTS applications, organizational culture, governance, organizational structure, testing, and architecture-related performance.

Our findings further indicate that not all types of corporate systems may equally benefit from microservice architecture; there was a general consensus that large corporate systems of record (e.g., ERP and CRM) may not be appropriate targets for microservice architecture. Similarly, while organizations that predominantly deal with ICT services (e.g., Amazon, Netflix, and the Guardian) may find it easier to take advantage of microservice architecture, many traditional organizations (where IT is merely seen as an enabler or "necessary evil") may struggle to overcome organizational challenges associated with adoption of microservices and underpinning DevOps methodologies.

Like any study, this paper has a number of limitations. The obvious limitation is that as a qualitative study we are unable to claim that our findings apply across any particular populations. On the other hand, we are not claiming any universality with our findings. Instead, we are simply presenting a range of concerns identified from a non-representative sample. This paper aims to add to the literature by identifying and discussing a range of issues associated with adoption and implementation of microservice architecture. To what extent these issues apply to organizations more generally may be investigated in future research if and when required.

## References


1. Singh J, Sharma G, Hill J, Shnackenberg A. *Organizational agility: what it is, what it is not, and why it matters*. in *Academy of Management Proceedings*. 2013. Academy of Management.
2. Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, Safina L. *Microservices: Yesterday, today, and tomorrow*, in *Present and Ulterior Software Engineering*; Cham, Switzerland: Springer:2017. 195–216.
3. Newman S. Building microservices: Designing fine-grained systems. Sebastopol, CA, USA: O'Reilly Media, Inc; 2015.
4. Dragoni N, Lanese I, Larsen ST, Mazzara M, Mustafin R, Safina L. *Microservices: how to make your application scale*. in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. 2017. Springer.
5. Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables devops: migration to a cloud-native architecture. IEEE Software. 2016;33:42–52.
6. Hasselbring W *Microservices for scalability: keynote talk abstract*. in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. 2016. ACM.
7. O'Connor R, Elger P, Clarke PM. *Exploring the Impact of Situational Context—A Case Study of a Software Development Process for a Microservices Architecture*. in *Software and System Processes (ICSSP), 2016 IEEE/ACM International Conference on*. 2016. IEEE.
8. Lewis J, Fowler M. *Microservices: a definition of this new architectural term*. 2014 Feb 2018; Available from: https://martinfowler.com/articles/microservices.html.
9. Thönes J. Microservices. IEEE Software. 2015;32:116–116.
10. Hunter T II. Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling. San Francisco, California, USA: Apress; 2017.
11. Vural H, Koyuncu M, Guney S. *A Systematic Literature Review on Microservices*. in *International Conference on Computational Science and Its Applications*. 2017. Springer.

12. MuleSoft. *Best Practices for Microservices*. 2018 Feb; Available from: https://www.mulesoft.com/lp/whitepaper/api/microservices-best-practices.
13. Bass L, Weber I, Zhu L. DevOps: A Software Architect's Perspective. New York, NY: Addison-Wesley Professional; 2015.
14. Pahl C, Jamshidi P. *Microservices: A Systematic Mapping Study*. in *6th International Conference on Cloud Computing and Services Science (CLOSER 2016)*. 2016.
15. Alshuqayran N, Ali N, Evans R. *A systematic mapping study in microservice architecture*. in *Service-Oriented Computing and Applications (SOCA) IEEE 9th International Conference on*. 2016. IEEE.
16. Di Francesco P, Malavolta I, Lago P. *Research on architecting microservices: trends, focus, and potential for industrial adoption*. *Software Architecture (ICSA) IEEE International Conference on*. 2017. IEEE.
17. Le VD, Neff MM, Stewart RV, Kelley R, Fritzinger E, Dascalu SM, Harris FC. *Microservice-based architecture for the NRDC*. *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. 2015. IEEE.
18. Wilson G *Eight fallacies of distributed computing - tech talk*. 2015 Feb 2018; Available from: http://blog.fogcreek.com/eight-fallacies-of-distributed-computing-tech-talk/.
19. Dragoni N, Dustdar S, Larsen ST, Mazzara M. *Microservices: migration of a mission critical system*. arXiv preprint arXiv:1704.04173, 2017.
20. Taibi D, Lenarduzzi V, Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: an Empirical Investigation. IEEE Cloud Computing. 2017;4:22–32.
21. Kratzke N, *About microservices, containers and their underestimated impact on network performance*. arXiv preprint arXiv:1710.04049, 2017.
22. Kitchenham BA, Dyba T, Jorgensen M. *Evidence-based software engineering*. in *Proceedings of the 26th international conference on software engineering*. 2004. IEEE Computer Society.
23. Pan SL, Tan B. Demystifying case research: a structured–pragmatic–situational (SPS) approach to conducting case studies. Inf Organ. 2011;21:161–76.
24. Baškarada S. Qualitative Case Study Guidelines. Qual Rep. 2014;19:1–25.
25. Glaser BG. The constant comparative method of qualitative analysis. Soc Probl. 1965;12:436–45.
26. Spradley JP. The ethnographic interview. Fort Worth (TX): Holt, Rinehart & Winston; 1979.
27. Aronson J. A pragmatic view of thematic analysis. The Qualitative Report. 1995;2:1–3.
28. LeCompte MD. Analyzing qualitative data. Theory Pract. 2000;39:146–54.
29. Glaser BG, Holton J. Remodeling Grounded Theory. Forum Qualitative Sozialforschung/Forum: Qualitative Social Research. 2004;5(02).
30. Glaser BG, Strauss AL. The Discovery of Grounded Theory. London, UK: Weidenfeld and Nicolson; 1967.
31. Hruschka DJ, Schwartz D, St. John DC, Picone-Decaro E, Jenkins RA, Carey JW. Reliability in Coding Open-Ended Data: lessons Learned from HIV Behavioral Research. Field Methods. 2004;16(3):307–31.
32. Conway ME. How do committees invent. Datamation. 1968;14:28–31.
33. MacCormack A, Baldwin C, Rusnak J. Exploring the duality between product and organizational architectures: a test of the "mirroring" hypothesis. Res Policy. 2012;41:1309–24.