IMPROVING BUSINESS PERFORMANCE THROUGH MICROSERVICES

Krasimir Baylov, Aleksandar Dimov
Department of Software Engineering – University of Sofia, Bulgaria
krasimirb@uni-sofia.bg, aldi@fmi.uni-sofia.bg

Abstract: *Microservices is an emerging software architectural style that is meant to deal with many of the limitations of contemporary monolith applications. In the recent years, this architectural style has become the preferred choice for building large and complex heterogeneous software systems. Many companies identify microservices as a means to build competitive advantage. Main reason for this is the flexibility that the microservice approach provides in terms of developing software intensive systems. Companies that incorporate microservices claim to be able to respond much faster to changing business environment. Microservices have both technical and cultural aspects. They provide a great flexibility in selecting the proper technologies. On the other hand, they result in a culture shift in each organization. Developers tend to become more productive and reduce the total number or defects. This paper makes an overview of the microservices architectural style for developing software intensive systems and analyzes the key aspects that result in improving the business performance. It discusses the main characteristics of microservices and provides guidelines on how to apply them in real business scenarios.*

Keywords: SERVICE ORIENTED ARCHITECTURE, MICROSERVICES, SOFTWARE ARCHITECTURE, BUSINESS PERFORMANCE

1. Introduction

In the past, developers used to build and deploy their enterprise applications packaged in a single piece of block that contains all the code and is called a *monolith*. Such an architectural approach has lots of benefits when it comes to small applications that are developed by a single team. However, contemporary applications tend to be more complex. They are often so large, that developers spend more time on identifying sources of issues and problems, than fixing them. As systems grow larger, the monolith approach implies a set limitations that prevent developers from working efficiently. Such applications get tied to a single technology stack. Upgrading to a newer version or changing the technology is an extremely difficult task. Components in monoliths are so tightly coupled that a single change results in updating multiple components. However, one of the main issues with monolith applications is that they cannot scale independently. In other words, either everything scales together or nothing scales at all. This forces companies to search for new ways of building software systems – such that could increase their Time to Market (TTM), reduce the overhead of supporting them and have strongly positive effect on their business performance in general.

Service Oriented Architecture (SOA) tends to solve many of the limitations implied by monolith applications. It is a paradigm that utilizes distributed capabilities under different ownership [1]. In contrast to monolith applications, SOA allows splitting a single system into multiple sub-systems that can communicate with each other through service calls. Such distribution allows separate teams to work on each sub-system and introduce changes independently. Moreover, SOA has been the preferred choice for enterprise system integration because of its flexibility in design and component communication.

Despite the multiple benefits of SOA, in the recent years a new architectural style is emerging - *microservices*. Microservices are small, autonomous services that work together [2]. In many ways microservices resemble SOA. It is often considered that microservices is a proper way of building SOA [3]. However, microservice architectural style can be characterized by a set of practices and principles that are common across different systems. Such systems are organized around business capabilities, i.e. each company organization is responsible for developing and supporting their own services. Each service can be built using the technology that is most suitable for it. In addition, each service has its own data store. This allows teams to store data in formats that make it easy to process. Communication between services is achieved only though the services application programming interfaces (API).

In this paper we present microservices as an architectural style for improving teams' efficiency. We discuss some of the principles that directly impact the business results of companies. We also compare microservices to SOA in order to highlight what distinguishes both architectural styles and when architects should prefer microservices to SOA.

The rest of the paper is organized as follows: Section 2 makes on overview of related work in the field; Section 3 introduces service-based architectures as opposed to monolith approach; Section 4 compared microservices and SOA; Section 5 provides a more detailed overview of microservice by describing their main benefits; and Section 6 concludes the paper and points steps for future research.

2. Related Work

Microservices has become a topical research direction recently and there exist a lot of efforts both from academy and industry researchers in the field. In this section we focus on some related papers that deal with the connection between SOA, microservices and business improvement.

For example, authors of [4] argue that software architecture has a strong impact on systems flexibility. It can help in keeping technology investment costs within acceptable ranges. Authors propose costs and benefits dimensions for evaluating the business impact of SOA systems. They have identified six cost dimensions (*organizational, business process, technology, servicification, integration* and *people*) and five benefits dimensions (*reuse, flexibility, integration, open standards* and *people*).

Service reusability is key aspect of service-based architectures. Authors of [5] provide an assessment of the business value from service reuse on SOA-based e-business platform. They analyze various models for business evaluations of software systems like Cost-Volume Profit (CVP), Break-Even Point (BEP) and apply them against an existing SOA system to evaluate the benefits of reusability.

Mark Richards [3] provides a detailed comparison between microservices and SOA. He argues that both are considered service-based architectures. They are highly distributed and communicate over well-defined APIs. However, there are many differences when it comes to components reusability, transaction management, service ownership and coordination, etc.

3. Service Oriented Architectures

The concept of Service Oriented Architecture (SOA) emerged over a decade ago in order to respond to the need of improved reusability in software engineering. Main goal of SOA is to decrease the complexity by providing means for successful development of large-scale software systems and at the same time to encourage further reuse of newly developed software artefacts. Service is the key artefact in SOA and it should be deployable and

accessible independently of programming languages, technologies and execution platforms. It is a broad term that has different meaning depending on its usage context [10]. For example, in many domains, the terms of web-service, e-service and business service have common meaning. In this paper, we consider service to be a: loosely coupled reusable software component that encapsulate functionality and is distributed and programmatically accessible over some previously defined networking protocols. In the case of web services, these protocols are standard and based on XML and users may search and discover web services in dedicated registries. This way, services offer standard approach for interaction in distributed environment. Additionally, services may be composed with each other in order to implement more complex functionalities and may be consumed by clients for use in different applications or in execution of sophisticated business processes. The promise of services is for increased maintainability and scalability of systems, achieved at predictable time with less efforts and decreased cost of development.

In this respect, reusability in SOA is enabled via the mechanisms of service composition and interaction [11]. SOA conforms to a service-orientation design paradigm, supports the implementation of services and services compositions together with the creation and evolution of a service repository. The basic principles of SOA design is based on the well-known publish-find-bind triad [12], as shown in fig. 1.

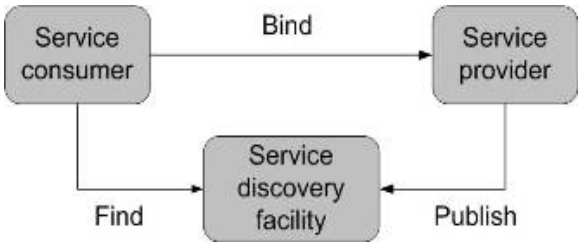

Fig. 1. *The basics of service-oriented architecture*

To summarize - service-based architectures emphasize on services as their main building block. As such, they share many common characteristics. First, they are distributed architectures. This means that services are accessed remotely over some remote-access protocol. Distributed architectures have lots of advantages compared to monolith ones. They scale independently, allow better control over the system development, components are loosely coupled, etc. Such benefits can help teams optimized their work in developing new components and features much faster. Companies can expect better business results as they optimize the overall effort in development and increase the time for delivering new functionality to the end customers. A key advantage of service-based architectures is their ability to scale. Companies that have incorporated service-based architectures can scale the service they need to. Other services that are not impacted by the increased load of requests may continue working without scaling. Independent scalability helps companies reduce cost, compared to the monolith approach. The main reason is that with monoliths everything scales at once – even the services or components that don't need to scale at all. Comparison between monolith and service-based scaling is presented in Fig. 2.

Another key characteristic for service-based architectures is that they are very complex. Their distributed nature introduces a new set of problems that architects need to consider when designing their systems. For example, service ownership is distributed across multiple teams and they may need to coordinate their actions when complex project timelines are followed. In many cases transaction management is extremely hard to achieve. When transactions span across multiple services developers need to come up with mechanisms for rolling back and accept eventual consistency. In a distributed architecture, service may fail to respond because of non-stable network communication. Architects need respond by designing systems that can handle failures at runtime and use complex monitoring mechanisms.

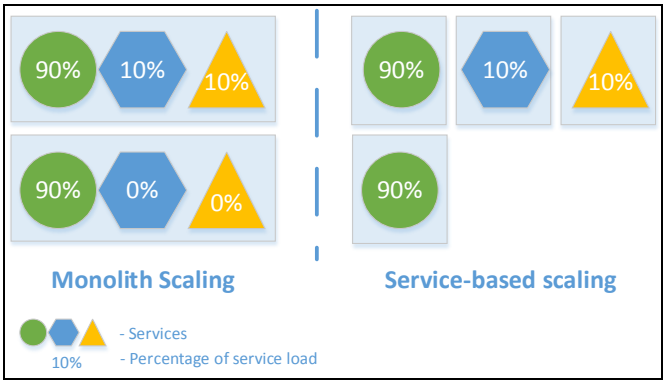

Fig. 2 *Comparing Monolith and Service-Based Scaling*

In general, service-based architectures are the preferred architectural style when it comes to large software systems. In many ways, they overcome the limitations of monolith applications but they also introduce a new complexity layer that should be considered.

4. Microservices and SOA

Both microservices and SOA are service-based architectures. As such, they share all common characteristics for service-based architecture and add new ones that are specific for each of them. In this section we discuss the differences between SOA and microservices and what makes microservices unique when in it comes to optimizing business efficiency.

A) Service Granularity

Microservices are small and fine-grained. Their small codebase allows developer to quickly identify and fix issues. This can be critical when teams support service level agreements (SLA) to their customers and they need to respond quickly to reported incidents. Small codebase means less dependencies between service components. This reduces the risk or introducing bugs to the code. Another indirect benefit of small codebases is that they could easily the tested in isolation. Implementing a set of automated tests can reduce risk of introducing defects significantly.

SOA doesn't provide any recommendation regarding service granularity. Services could vary in size significantly and developers could easily be tempted to build large enterprise services that are hard to maintain.

B) Component Sharing

Microservices follow the principle of *share-as-little-as-possible*. The reason is that this makes each service independent to other shared components. A common approach for building microservices is copying the same components to all services and evolving the components independently. This approach gives developers freedom to update their services independently by reducing the external dependencies and team coordination.

SOA approach for component sharing is completely different. SOA follows the principle of *share-as-much-as-possible*. Although this approach solves the problem with code duplication, it increases the coupling between service components. When a shared component needs to be updated, teams need to analyse the impact on other services and align their release plans to the other services.

C) Remote Access Protocols

Microservices tend to favour Representational State Transfer (REST) [6] for exchanging messages. Compared to other protocols

REST is relatively simple and lightweight. It is mainly based on Hyper Text Transfer Protocol (HTTP) and uses HTTP verbs (GET, PUT, UPDATE, etc.) to retrieve and manipulate external resources. REST is best suited for basic non-complex integrations [7], which fits perfectly into the microservice style. Moreover, REST allows easy ad hoc development and integrations. This helps development teams to quickly implement and deploy new services that can exchange messages with others.

SOA doesn't have any specific prescriptions regarding remote access protocols. However, more often than not, you will encounter SOA applications that extensively use Simple Object Access Protocol (SOAP). This protocol is more suitable for complex integrations since it supports additional features like web service security, quality of service (QoS), reliable messaging, etc. However, this makes the protocol heavyweight for simple and well-separated applications.

D) Heterogeneous Interoperability

Heterogeneous interoperability refers to the ability of a system to integrate with others based on other technologies. Microservices, limit the options to two – *REST* and *simple messaging*. This choice limitation eliminate the need of using complex middleware systems that translate, route and mediate messages. This can have a strong positive cost impact as such middleware systems are often too expensive to buy. Also, introducing such a system increases the development and administrative efforts, as well as service coupling because all services need to use this middleware.

SOA systems do not have specific limitation for interoperability. They could integrate with various heterogeneous systems using its messaging middleware. However, this comes at certain price – increased complexity and cost.

E) Simplicity

Simplicity is one of the fundamental factors that distinguishes SOA and microservices. Microservices strive for simplicity at any level of implementation. Services are fine-grained because they are simpler to support. Protocol and message options are reduced because it is much easier to operate with limited set of interfaces that need to be supported. Application simplicity has a long-term impact on business results. Such applications are more robust and resilient. They could easily be adapted to new market requirements and attract new customers.

SOA doesn't provide any guideline in terms of simplicity. Although service-oriented systems can be built for simplicity, most times you will encounter large and complex service-based applications. They can integrate with a large set of heterogeneous systems, work with large amount of data and support complex rules for processing messages. But such applications are hard to support and change rapidly. If such application complexity is not required, then it introduces more troubles that solutions.

In general, microservices are a form of SOA. Since SOA practices are significantly less restrictive than microservices it is reasonable to think of microservices as a "form of SOA, perhaps service orientation done right" [8]. This relationship between microservices and SOA is presented in Fig. 3.

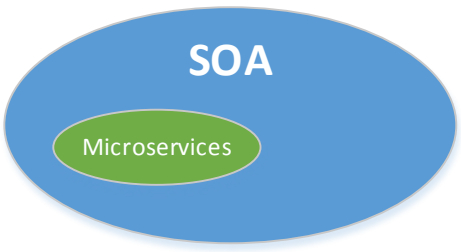

Fig. 3. *SOA and Microservices*

5. Microservice Benefits

In this section we discuss the key benefits of using microservices [9]. Those benefits can directly influence the business results of companies in either short or long term.

A) Independent Deployments

Microservices can be deployed independently from other services. This means that teams can establish different release cycle for each service and avoid any coordination efforts for deployments. Deployment independence is critical when it comes to large systems that require frequent releases to multiple services.

B) Small Services

Microservice have a small codebase. Developers don't need to know the entire system to implement a change in a service. This reduces the time in troubleshooting problems and fixing them. Small services can be built, deployed and ran quickly. With high level of automatization this could be achieved within several seconds.

C) Independent Scalability

Microservices can scale independently. When some of the services is overloaded it can scale without affecting other services. When the service scales by duplicating itself or increasing its computational resources companies need to pay only for that service. Other services don't get affected by this and continue using the same amount of computing resources.

D) Development Scalability

The large number of small and independent services mean that teams can be split around those services. Management could easily analyze the skills of the developers and form new smaller and easily manageable teams for the services. In this way, teams could work in parallel as they have nearly no dependencies among them.

E) Improved Fault Isolation

The large number or services and the loose coupling between them results in improved fault isolation. If any of the services fails this will not affect any of the other services. Therefore, the system can continue operating while developers identify the issues with the failed service and fix it.

F) Technology Diversity

A key benefit of microservices is their technology diversity. Since the entire system is split into small services that communicate over some remote access protocol (usually REST), developers are free to select any technology for building their services. Different service have different purpose, so developers can select the technologies that best suit in solving their technical problems. Technology diversity has another great benefit over monolith (and sometimes SOA) applications – it doesn't tie companies to a single technology stack. When companies decide that there is a new technology that better fits into their business strategy, they could easily replace the old technology with the new one.

6. Conclusion and Future Work

In this paper we have described microservices as a way of building software systems. We presented the downsides of monolith applications and discussed how microservices can be used to overcome many of the monolith limitations and improve the overall efficiency of business. Microservices is form of SOA that provides specific prescriptions and guidelines how to be implemented. A key aspect of microservices is that systems are split into small independent services which allows them to evolve, scale and operate without affecting other services.

Further steps in our research would include comparing microservices to other architectural styles and analysing how they fit into different software domains, e.g. embedded systems, business

intelligence, web applications, etc. Because of the high level of automation and their distributed nature, we plan further research in applying self-adaptive mechanisms in microservices.

Acknowledgements

Research, presented in this paper was partially supported by the DFNI I02-2/2014 (ДФНИ И02-2/2014) project, funded by the National Science Fund, Ministry of Education and Science in Bulgaria.

References

[1] MacKenzie, C. Matthew, et al. "Reference model for service oriented architecture 1.0." OASIS Standard 12 (2006).

[2] Newman, Sam. Building Microservices. " O'Reilly Media, Inc.", 2015.

[3] Richards, Mark. Microservices vs. Service-Oriented Architecture. "O'Reilly Media, Inc.", 2015

[4] SOA World Magazine. 2008. The Business Value of SOA- Based Agile IT Architectures. SYS-CON Media Inc. http://webservices.sys-con.com/read/275128_p.htm

[5] Kryvinska, Natalia, et al. "Assessment of business value from services re-use on SOA-based e-business platform." Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services. ACM, 2010.

[6] Fielding, Roy Thomas. Architectural styles and the design of network-based software architectures. Diss. University of California, Irvine, 2000.

[7] Pautasso, Cesare, Olaf Zimmermann, and Frank Leymann. "Restful web services vs. big'web services: making the right architectural decision." Proceedings of the 17th international conference on World Wide Web. ACM, 2008.

[8] James, Lewis; Fowler, Martin. Microservices: A definition of this new architectural term. http://martinfowler.com/articles/microservices.html, 2014, March 25

[9] Richardson, Chris. "Microservices: Decomposing applications for deployability and scalability." (2014).

[10] Baida, Z., Gordijn, J., Omelayenko, B. (2004). A shared service terminology for online service provisioning. Proceedings of the 6th international conference on Electronic commerce, , Delft, The Netherlands, pp. 1-10.

[11] Erl, T. (2008). SOA: Principles of Service Design, Pearson Education, Inc., Boston, USA.

[12] Weerawarana, S, et all. (2005). Web Services Platform Architecture, Pearson Education, Inc., Upper Saddle River, NJ, USA