



# Microservices

Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert

## From the Editor

Microservices are gaining momentum across industries to facilitate agile delivery mechanisms for service-oriented architecture and to migrate function-oriented legacy architectures toward highly flexible service orientation. The International Data Corporation has forecasted that by 2021, 80 percent of application development on cloud platforms will be with microservices. In this instalment of Software Technology, Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and I present a brief overview of microservice technologies and how to migrate to them. I look forward to hearing from both readers and prospective authors. —*Christof Ebert*

## INSPIRED BY SERVICE-ORIENTED

computing, microservices are small applications with a single responsibility that can be deployed, scaled, and tested independently.<sup>1</sup> This decomposition of the monolith (whose modules can't be executed independently) into a granular system interacting via messages (through RPC-based APIs or RESTful web services, for instance) enables organizations to achieve better time to market by means of swifter, more continuous deliveries. (REST stands for Representational State Transfer.) It also enables agile teams to structure their work around these services,<sup>2</sup> given that microservices are, by definition, autonomously developed.<sup>3</sup>

Connecting microservices with DevOps will increase software engineering's impact and benefits.<sup>4</sup> However, microservices also pose challenges and have disadvantages. Challenges include decomposing

the monolith into microservices; continuous architecture monitoring and deployment; more complex testing, versioning, and deprecating; and state management. A particular disadvantage is that microservices can involve soft factors such as the need for experienced staff and the difficulty of learning the technology.

Companies such as Amazon, Deutsche Telekom, LinkedIn, Netflix, SoundCloud, *The Guardian*, Uber, and Verizon are quickly adopting microservice-based approaches. Often, microservices are used to modernize legacy applications. The goal is to split such monolithic systems into microservices through refactoring. This supports the incremental modernization of legacy software and is perhaps less risky than completely redeveloping the whole system into microservices.

## Technologies for Microservices

Microservice software breaks systems and applications down to a more granular, modular level. The concept was created as a service-oriented architecture (SOA) follow-up some 10 years ago. It's about fragmenting complex applications into small pieces and a fluid delivery model that delivers services on demand, thus improving performance. DevOps provides the process framework for developing, deploying, and managing the microservice container ecosystem. With a service-oriented refactored architecture, DevOps can ensure fast delivery cycles by integrating the previous silo-style business processes of development and operations.

A variety of microservice technologies have evolved over the past two years.<sup>1,3</sup> Table 1 provides an overview of current technologies and

Table 1. Industry-grade microservice technologies.

	Technology			
	Azure Service Fabric	Lagom	MicroProfile	Spring Suite (Boot)
Authentication	Active Directory	Basic	JavaScript Object Notation (JSON) Web Token	Spring Security
Security	Security Center	Basic	JSON Web Token	Spring Security
Tracing	Event Tracing Windows	Basic	OpenTracing	Spring Cloud Sleuth
Deployment	Built-in	—	Java Platform, Enterprise Edition (J2EE)	Yes, especially for the Spring framework
Reliability	Reliable Collections	Via others	MicroProfile Fault Tolerance 1.0	Built-in
Cost	Paid	Free	Free	Free
Orchestration	Azure Container Service	ConductR	Via others	Spring Suite
Monitoring (health check)	Application, cluster, or service based	Not available	MicroProfile Health Check 1.0	Hystrix
Usability	High	Low	Low	Medium
Containers	Azure Container Service	Via others	Via others	Via others
Language	C#, .NET, Java	Java, Scala	Java	Java
URL	<a href="https://azure.microsoft.com/en-us/services/service-fabric">azure.microsoft.com/en-us/services/service-fabric</a>	<a href="http://www.lagomframework.com">www.lagomframework.com</a>	<a href="http://microprofile.io">microprofile.io</a>	<a href="http://projects.spring.io/spring-boot">projects.spring.io/spring-boot</a>

how we rate them qualitatively on the basis of our industry experience.

### Migration to Microservices

In our experience, when introducing a microservice migration, you should follow these five guidelines:

- *Be prepared for organizational changes.* Microservices are ahead of a new business culture.
- *Study the system.* Identify dependencies by means of tools (for example, Retrace, Dynatrace, or SchemaCrawler) or manually.
- *Define the architecture.* The system's architectural structure must be defined, including tools, frameworks, and so on. Although this isn't the time to choose the specific part of the system to migrate, don't postpone platform selection. And, if possible, include proper DevOps enablers in the architecture—the earlier, the better.
- *Prioritize your components for migration.* Develop your criteria for migration, and perform a study of each approach's risks before final adoption and migration.
- *Perform design, coding, testing, and integration.* Adopt an agile approach. If possible, use DevOps tools on each phase (for example, Fuge, Git and

GitHub, Jenkins, Phantom, or Kubernetes).

Microservices, as in any software evolution, can significantly affect quality attributes:

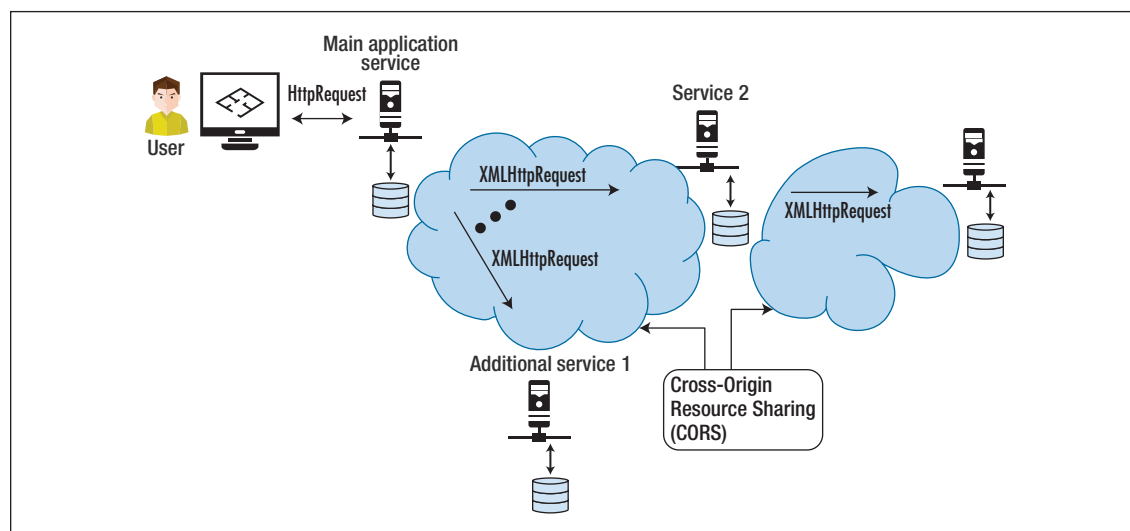
- *Security.* Given that data is flowing among microservices, the need exists to secure such communication through encryption techniques, but authentication mechanisms also must be implemented (see the case study sidebar).
- *Performance.* Generally, microservices perform worse than monoliths. The final performance depends on several

## CASE STUDY: MICROSERVICES AND SECURITY

While developing an application based on SonarQube and AngularJ, we faced several problems related to Cross-Origin Resource Sharing (CORS). We used SonarQube to analyze source code at different sites; a main application service gathered the information from these sites. By means of CORS, the application used additional HTTP headers to gain

permission to access selected resources from a server at a different origin (domain) (see Figure A).

Normally, browsers block such requests (see Table A) for security reasons—for instance, to avoid cyberattacks. So, in microservices scenarios, a particular microservice architecture can affect CORS behavior, leading to



**FIGURE A.** A common scenario in which a user accesses through a browser a website that gathers information from other services.

factors, including network aspects (latency, for instance) and virtualization (deployment in virtual machines imposes additional performance overhead).

- **Reliability.** Microservices by nature (because they're distributed) are less reliable than monoliths.
- **Availability.** In microservice architectures, a system's availability depends on not only the microservices' availability but

also their integration. On the other hand, microservices reduce deployment time, increasing availability.

- **Maintainability.** Microservices are, by nature, independent, making maintainability one of this approach's best aspects.
- **Testability.** A microservice is, initially, easier to test than a monolith. However, integration testing can be much more

complex, depending on the number of components and their connections.

Migration presents interrelated technical challenges regarding multi-tenancy, statefulness, and data consistency that migration teams must tackle to ensure success. In addition, more general aspects can potentially threaten migration, including monolith decoupling, data splitting,

the need to configure CORS options on each web browser.

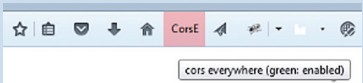
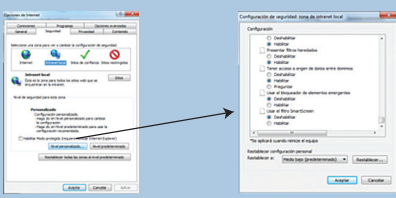
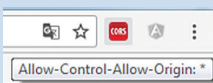
In fact, servers and browsers are implementing CORS. According to the W3C,

*User agents commonly apply same-origin restrictions to network requests. These restrictions prevent a client-side Web application running from one*

*origin from obtaining data retrieved from another origin, and also limit unsafe HTTP requests that can be automatically launched toward destinations that differ from the running application's origin.*<sup>6</sup>

On this basis, we mitigated the CORS security risk by properly configuring CORS in the set of available browsers.

**Table A. Browsers used in the environment, and how to enable or disable Cross-Origin Resource Sharing (CORS).**

Browser	Ver.	CORS plug-in	Actions for enabling or disabling CORS	Screenshot
Firefox	54.0	Cross Domain—CORS 0.1.1	A button on the toolbar enables or disables CORS.	
Internet Explorer	11.0.9600.18697	No specific plug-in	The toolbar has no specific button. Users must modify this option through Internet Options → Security → Custom → Miscellaneous → enable or disable.	
Chrome	58.0.3029.110	Allow-Control-Allow-Origin:* 1.0.3	A button on the toolbar enables or disables CORS.	

communication among services, effort estimation, DevOps infrastructure, and resistance to change. In sum, the challenges can be technical, economic, and psychological.

As we mentioned before, microservices typically are distributed. In agile and DevOps delivery models, each delivery has dependency impacts that must be analyzed, validated, and considered for packaging. When operated across networks,

microservices can incur significant performance penalties that must be compensated for by additional architectural tweaks, such as caching layers.

Consider the impacts on tools and application lifecycle management and product lifecycle management. Along with DevOps, microservices push automation from the application to the infrastructure. Compared with manual infrastructure

provisioning, configuration management tools will facilitate fast production provisioning. You can reduce configuration maintenance complexity with an optimized microservice architecture by recreating the production system in the development machines when moving from a monolithic block of software to a microservice approach.

Microservices will facilitate the convergence of classic IT and

## ABOUT THE AUTHORS



**XABIER LARRUCEA** is a senior project leader at Tecnalia. He is a conference correspondent for *IEEE Software* and teaches at the University of the Basque Country. Contact him at [xabier.larrucea@tecnalia.com](mailto:xabier.larrucea@tecnalia.com).



**IZASKUN SANTAMARIA** is a senior project leader at Tecnalia. Contact her at [izaskun.santamaria@tecnalia.com](mailto:izaskun.santamaria@tecnalia.com).




**RICARDO COLOMO-PALACIOS** is a full professor in Østfold University College's Computer Science Department. Contact him at [ricardo.colomo-palacios@hiof.no](mailto:ricardo.colomo-palacios@hiof.no).



**CHRISTOF EBERT** is the managing director of Vector Consulting Services. He is on the *IEEE Software* editorial board and teaches at the University of Stuttgart and the Sorbonne in Paris. Contact him at [christof.ebert@vector.com](mailto:christof.ebert@vector.com).

area of such evolution is serverless computing. This approach encapsulates common functionality (but not business functionality such as authentication, validation, or monitoring) into a hosting microservice handling these aspects and executing proper business functions. The system is then serverless or provides “function as a service” (FaaS), allowing Amazon Web Services Lambda, Iron.io, or Google functions to host your business functions.

Because microservices need DevOps, we recommend starting with a tailored DevOps strategy. It will have immediate value owing to better integration across the lifecycle and can gradually evolve to a microservice delivery model—if appropriate. 

## References

1. J. Thönes, “Microservices,” *IEEE Software*, vol. 32, no. 1, 2015, pp. 116, 113–115.
2. A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3, 2016, pp. 42–52.
3. C. Pautasso et al., “Microservices in Practice, Part 1: Reality Check and Service Design,” *IEEE Software*, vol. 34, no. 1, 2017, pp. 91–98.
4. R. Colomo-Palacios et al., “A Case Analysis of Enabling Continuous Software Deployment through Knowledge Management,” *Int'l J. Information Management*, Nov. 2017; [www.sciencedirect.com/science/article/pii/S0268401217308782](http://www.sciencedirect.com/science/article/pii/S0268401217308782).
5. C. Ebert and K. Shankar, “Industry Trends 2017,” *IEEE Software*, vol. 34, no. 2, 2017, pp. 112–116.
6. A. van Kesteren, ed., *Cross-Origin Resource Sharing*, W3C recommendation, 16 Jan. 2014; [www.w3.org/TR/cors](http://www.w3.org/TR/cors).

embedded (real time) systems.<sup>5</sup> Obviously, more complex and critical applications with availability and security constraints shouldn't be addressed entirely by a volatile cloud delivery model. To achieve continuous delivery for embedded systems, devices must be connected, so machine-to-machine communication and an Internet of Things architecture should be implemented. However, in critical applications in which failure isn't an option—for example,

in the automotive, medical, or aerospace domain—a more conservative approach with a strong focus on availability, safety, and performance should be considered.

**T**o be successful, a microservice-based solution needs an appropriate software architecture design for each application domain. Currently, microservice technologies are evolving fast. One