Microservices

Xabier Larrucea[1], Izaskun Santamaria[1], Ricardo Colomo-Palacios[2], Christof Ebert [3]

[1]TECNALIA, Bizkaia, Spain. {xabier.larrucea;izaskun.santamaria}@tecnalia.com
[2] Østfold University College. ricardo.colomo-palacios@hiof.no
[3] Vector Consulting Services. Christof.Ebert@vector.com

[Editor introduction]
Microservices are gaining momentum across industries to facilitate agile delivery mechanisms for SOA and to migrate function-oriented legacy architectures towards highly flexible service-orientation. IDC forecasts for 2021 that 80 percent of application development on cloud platforms will be with microservices. Authors Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and myself present a brief overview on Microservices technologies and how to migrate. I look forward to feedback from both readers and prospective column authors.
—Christof Ebert

Inspired by Service-Oriented computing and opposite to monoliths, whose modules cannot be executed in an independent way, microservices are small applications that can be deployed independently, scaled independently and tested independently and that has a single responsibility [1]. This decomposition of the monolith into a granular system interacting via messages (RPC-based APIs or RESTful, for instance) enables organizations to achieve better time to market by means of swifter and more continuous deliveries but also enables agile teams to structure their work around these services [2], given that microservices are, by definition, autonomously developed [3, p. 1].

Connecting microservices with the DevOps will boost continuous software engineering impact and benefits [4]. However, there are also issues and disadvantages in the use of microservices. On the challenges side are the difficulty behind the decomposition of the monolith into microservices, aspects related to continuous architecture monitoring and deployment, more complex testing, versioning and deprecating and state management. On the disadvantages side, one can find soft factors like the need of seniority and the difficulty to learn.

Companies like Amazon, Deutsche Telekom, LinkedIn, Netflix, SoundCloud, The Guardian, Uber or Verizon are fast adopting microservices-based approaches. Often microservices are used to modernize legacy applications in organizations. Consequently, the goal is splitting such monolithic systems into microservices by means of refactoring. This supports the incremental modernization of legacy software as maybe the less risky option compared to the complete re-development of the whole system into microservices. This approach is easing the low-risk, small-scale incremental modernization that is often preferred to large-scale approaches.

1 Technologies for Microservices

Microservices software breaks systems and applications down to a more granular, modular level. It has been created as a SOA follow-up some ten years ago. It is about fragmenting complex applications to small pieces and a fluid delivery model where services are delivered on demand thus improving

performance. DevOps provides the process framework for developing, deploying, and managing the microservices container ecosystem. With a service-oriented refactored architecture DevOps can ensure fast delivery cycles by integrating the before silo-style business processes of development and operation.

A variety of technologies for microservices have been evolved during the past two years [1,3]. Table 1 provides an overview on current technologies and how we rate them qualitatively from our industry experiences.

Table 1. Industry-grade Microservice Technologies

	Azure Service Fabric	Lagom	MicroProfile	Spring Suite (Boot)
Authentication	Active Directory	Basic	JSON Web Token	Spring Security
Security	Security Center	Basic	JSON Web Token	Spring Security
Tracing	Event Tracing Windows (ETW)	Basic	OpenTracing	Spring Cloud Sleuth
Deployment	Built-in		J2EE	yes, especially for Spring framework
Reliability	Reliable Collections	via others	MicroProfile Fault Tolerance 1.0	Built-in
Cost	pay	free	free	free
Orchestration	Azure Container Service	ConductR	via others	Spring Suite
Monitoring (Health check)	Application/Cluster /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
Website	https://azure.microsoft.com/en-us/services/service-fabric/	https://www.lagomframework.com/	https://microprofile.io/	https://projects.spring.io/spring-boot/

2 Migration to Microservices

To our experience from introducing Microservices, there are some general aspects to consider when introducing a microservices migration:

1. Be prepared for organizational changes. Microservices are ahead of a new business culture.
2. Study the system. Identify dependencies by means of tools (e.g. Retrace, Dynatrace, SchemaCrawler …) or manually.
3. Define the architecture. Architectural structure of the system must be defined including tools, frameworks…. Although it is not the time to choose the specific part of the system to be migrated, do not postpone platform selection and, if possible, include in the architecture proper DevOps enablers. The earlier, the better.
4. Prioritize your components for migration. Develop your criteria for migration and include a study on the risks of each of the approaches before final adoption and migration.
5. Design / Coding / Testing / Integration. Adopt an agile approach and, if possible, use DevOps tools on each phase e.g. Fuge; Git/Github; Jenkins; Phantom; Kubernetes…)

Microservices, as any software evolution, present significant impact on quality attributes.

- Security. Given that data is flowing among microservices, there is a need to secure such communication by means of encryption techniques but also it is needed to implement authentication mechanisms (see case study sidebar)
- Performance. Microservices in general present lower performance than monoliths. Final performance depends on several factors, including network aspects (latency for instance) and virtualization (deployment in virtual machines imposes additional performance overhead).
- Reliability. Again, microservices are, in general and by nature given their distributed essence, less reliable than monoliths.
- Availability. In microservices architectures, the availability of a system depends on the availability of the microservices but also on the integration of such components. On the other hand, microservices reduce deployment time, increasing availability.
- Maintainability. Microservices are, by nature, independent, making maintainability one of the best aspects in this approach.
- Testability. A microservice is, initially, easier to test than a monolith. However, integration test can be much more complex depending on the number of components and their connections.

There are obvious challenges with microservices. Migration presents also interrelated technical challenges regarding multi-tenancy, statefulness and data consistency that migration teams must tackle to ensure success. Beyond these purely-technical challenges, there is a set of more general aspects that can be potential threats for the migration process including monolith decoupling, data splitting, communication among services, effort estimation and DevOps infrastructure and resistance to change . In sum, challenges can be technical, economical and psychological.

Microservices by nature are typically distributed. In agile and DevOps delivery models, each delivery has dependency impacts which must be analyzed, validated and considered for packaging. When operated across networks microservices can cause significant performance penalties that must be accommodated with additional architectural tweaks, like caching layers.

Consider the impacts on tools and ALM/PLM. Along with DevOps Microservices push automation from application to infrastructure. Compared with manual infrastructure provisioning, configuration management tools will facilitate fast production provisioning. Configuration maintenance complexity can be reduced with an optimized Microservice architecture by recreating the production system in the development machines when moving from a monolithic block of software to a microservices approach.

Microservices will facilitate the convergence of classic IT and embedded (real-time) systems [5]. Obviously more complex and critical applications with availability and security constraints should not be addressed entirely by a volatile cloud delivery model. To achieve continuous delivery for embedded systems devices must be connected, so machine to machine communications and an IoT (Internet of things) architecture should be implemented. However, in critical applications where failure is not an option, for example automotive, medical or aerospace, a more conservative approach with high focus on availability, safety and performance should be considered.

3 Summary and Outlook

Microservices are bringing several benefits, but they are also implying several hurdles to be overcome. An appropriate software architecture design for each application domain is needed for a successful microservices based solution. Currently microservice technologies are fast evolving. One of these evolutions is serverless computing. By means of this approach, common functionality (not business functionality as authentication, validation, monitoring…) is encapsulated into a hosting microservice handling these aspects and executing proper business functions. The system is then serverless or "function as a service (faas)", allowing AWS Lambda, Iron.io or Google functions hosting your business functions.

Since microservices need DevOps, we recommend starting with a tailored DevOps strategy. It will have immediate value due to better integration across the life-cycle and can gradually evolve to a microservices delivery model – if appropriate.

Sidebar: Case Study Microservices and Security

During the development of an application based on Sonarqube and AngularJ, authors faced several problems related to Cross Origin Resource Sharing (CORS). In this case study, Sonarqube is used to analyze source code in different sites; a main application service gathers the information from these sites. By means of CORS, additional HTTP headers are used to gain permission to access selected resources from a server on a different origin (domain) (figure 1). This kind of requests are normally blocked (table 2) by browsers for security reasons, for instance, to avoid cyber-attacks. Therefore, in microservices scenarios, a particular microservice architecture can affect CORS behavior leading to the need to configure CORS options on each web browser.

In fact, servers and browsers are implementing CORS and according to W3 (https://www.w3.org/TR/cors/) “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“. On this basis the CORS security risk is mitigated by means of a proper configuration of this aspect in the set of available browsers.

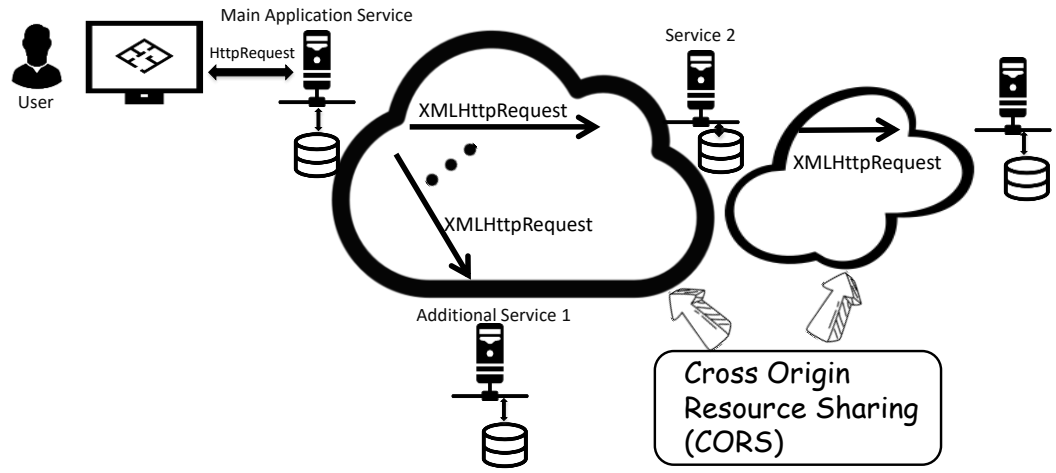
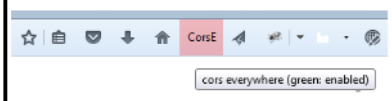
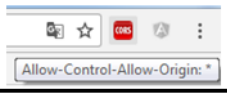


Figure 1: Common scenario where a user access through its browser to a website which gathers information from other services.

Table 2. Browsers used in the environment, and how to enable/disable CORS

Browser	version	CORS plugin	Actions for enabling/disabling CORS	Screenshot
Firefox	54.0	Cross Domain - CORS 0.1.1	There is a button on the toolbar for enabling and disabling CORS	
Iexplorer	11.0.9600.18697	No specific plugin	There is not specific button on the toolbar. User is required to modify this option on the "Internet Options"-->Security-->Custom-->Miscellaneous-->enable or disable.	
Chrome	58.0.3029.110	Allow-Control-Allow-Origin:* 1.0.3	There is a button on the toolbar for enabling and disabling CORS	

4 References

[1] J. Thönes, “Microservices,” IEEE Softw., vol. 32, no. 1, pp. 116–116, Jan. 2015.

[2] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[3] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, “Microservices in Practice, Part 1: Reality Check and Service Design,” *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, Jan. 2017.

[4] R. Colomo-Palacios, E. Fernandes, P. Soto-Acosta, and X. Larrucea, “A case analysis of enabling continuous software deployment through knowledge management,” *Int. J. Inf. Manag.*, Nov. 2017.

[5] Ebert, C. and K. Shankar: Industry Trends 2017. IEEE Software, ISSN: 0740-7459, vol. 34, no. 2, pp. 112-116, Mrc/Apr 2017

5 Authors

Xabier Larrucea is a senior project leader at Tecnalia. He is on the IEEE Software initiatives team and teaches at the University of the Basque Country. Contact him at xabier.larrucea@tecnalia.com.

Izaskun Santamaria is a senior project leader at Tecnalia. She has more than 18 years' experience in industrial contexts related to software engineering. Contact her at: Izaskun.santamaria@tecnalia.com

Ricardo Colomo-Palacios is a full professor in the Computer Science Department at Østfold University College. Colomo-Palacios received a PhD in computer science from the Universidad Politécnica of Madrid. Contact him at 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