Do Microservices Pass the Same Old Architecture Test? or: SOA is Not Dead – Long Live (Micro-)Services

Position Paper for SEI SATURN 2015 Workshop

Olaf Zimmermann[1]

[1] University of Applied Sciences of Eastern Switzerland (HSR FHO), Oberseestrasse 10, 8640 Rapperswil, Switzerland
{firstname.lastname}@hsr.ch

My pre-workshop position is that microservices architectures do not constitute a new architectural style. They rather qualify as service-oriented architecture (SOA) done right, facilitated by contemporary software engineering practices and recent advances in Web application development – for instance, a) xDD (with x = {B, D, T, …}), pipes-and-filters chaining of fine-grained processing logic, polyglot programming and persistence, b) build and test process automation and continuous deployment, e.g., into lightweight containers and cloud computing environments and c) modern, lean approaches to systems management closely intertwined with software construction.

I developed this position in unsystematic literature review (unSLR) that started form the workshop reading list. This unSLR indicates that the main differences between microservices and previous attempts to service-oriented computing do not concern the architectural style as such (e.g., its design intent and its platform-independent principles and patterns), but its implementation. For instance, many microservices architects decide for Web-centric technology stacks combining pre-packaged open source assets, and their logical application and integration designs are geared towards continuous delivery and cloud deployment. The unSLR also unveiled that, just like any incarnation of SOA, microservices architectures are confronted with a number of nontrivial design challenges that are intrinsic to any distributed system – including data integrity and consistency management, service interface specification and version compatibility (both syntax and semantics), and application and infrastructure security; such architecture design issues transcend both style and technology debates.

My position is also supported by a comparison of a) microservices architectures tenets and definitions (drawn from literature 2009-2015), b) SOA principles and patterns (drawn from literature and project experience 2001-2008), and c) plain old distributed computing knowledge essentials. Finally, a closer look at common (mis-)perceptions and gaps of SOA leads to questions about potential microservices pendants and reconfirms that SOA and microservices are close relatives that have a lot in common.

In summary, I welcome the current microservices movement from an SOA point of view as it brings new momentum and community support for this architectural style. From a neutral application and integration architect point of view, I appreciate the promises and opportunities of microservices, but I also acknowledge the additional cognitive load, as well as the increased design, testing and maintenance efforts that are required to design microservices architectures properly. These efforts are caused by a number of additional, nontrivial design choices to be made and novel options for the many “distribution classics” type of architectural decisions required.

Workshop questions. Microservices are young and emerging still; therefore, many questions seem to remain open in the online resources that I consulted for the unSLR. Specifically, I look for answers to the following questions:

1. *How fine is too fine?* The project team-focused two-pizza rule or the code-centric few-lines-per-service heuristic may establish some rules-of-thumb regarding the goals of the functional decomposition (a.k.a. logical partitioning) of a software system into microservices; but in my opinion such rules-of-thumb cannot guide project teams in how to achieve these goals. While it is commonly agreed that business input and domain expertise are needed during this journey, e.g., when following the agile principle to welcome changing requirements even late in development, it is not yet clear to me when and how such insights should be considered during microservices synthesis. A fallback position would be to apply general-purpose methods such as object-oriented analysis and design (e.g., OpenUP elaboration and construction techniques) or their agile substitutes.
2. *Which guidance on the selection and adoption of domain-driven design during functional partitioning of a system into discrete microservices and resources is applicable?* For instance, is any instance of the "Aggregate" pattern a micro-service candidate, or is its "Root Entity" a better choice? And is a "Bounded Context" a composite "Service" then, comprising of a cohesive collection of Aggregates/Root Entities and being exposed as a REST resource in an "API Gateway"? Or should each Bounded Context form a single microservice? Either way, how is the microservices design mapped to REST resource design – e.g., how many resources per microservice should be defined?
3. *If there are microservices, is there a place for macroservices as well?* A variant of this question is how the microservices concept relates to plain old Patterns of Enterprise Application Architecture (PoEAA) such as "Service Layer", "Remote Façade" and "Data Transfer Object". For instance, can a Remote Façade be considered a macroservice to be exposed by an API Gateway (with the Service Layer then being realized by this API Gateway, and business logic layer patterns such as "Transaction Script" and "Domain Model" supplying the underlying microservices)? Or is the Remote Façade the better microservice candidate? How are the macroservices linked to the microservices from a REST perspective?
4. *How should overall data integrity be ensured in microservices architectures?* This is a variant of the strict vs. eventual consistency discussion around the Consistency, Availability, Partition Tolerance (CAP) theorem in cloud computing, which becomes particularly relevant when parts of the domain model are replicated (e.g., master data) – what are the implications of using eventual consistency in the context of rich domain models facing challenging audit requirements such as non-repudiation and Completeness, Accuracy, Validity, Restricted Access (CAVR) compliance (e.g., each order must refer to exactly one customer, and this customer- must correspond to an existing legal entity)? A subsequent question concerns business-level undos and compensating actions:
5. *How can microservices and RESTful HTTP resources support the "Business Transaction" pattern to coordinate updates to non-transactional resources?* Is the compensating action a different verb/method or a separate resource – and, assuming it is a separate resource, how is this resource linked to the original one, e.g., is there a predefined link type or another naming convention? Yet other

subsequent questions deal with the required substitute for foreign key relationships and SQLish query capabilities (joins) as well as conflict resolution if the database is partitioned and does not guarantee ACID properties across partitions – e.g., does the foreign key management code and query execution code have to be written, tested, optimized, and maintained by the programmers of API Gateway resources and/or end user applications?

6. *How to avoid microservice deployment dependency and invocation "spaghetti"?* This question is a variant of one of the main points of critique for SOA: ESBs seem have to have the reputation of being fat and always containing business logic (such ESB design actually violates several defining SOA principles and recommended ESB practices). Macroservices and API Gateways in microservices architectures run the same risk of becoming stateful (e.g. over time during micro-services maintenance); they may even be more vulnerable due to increased flexibility of RESTful resources on higher levels of maturity and absence of static contracts. Stitching the end user application together in the client tier (e.g., in JavaScript running in a Web browser) is not an option in many business contexts and production settings due to the negative impact on certain quality attributes.
7. *How to identify and overcome (or manage) general SOA pitfalls and inhibitors?* Assuming that microservices become mainstream, they will have to deal with counterproductive side effects of success such as misuse of terms, vision, and concepts, e.g., in the form of scope creep, ambiguity, terminology abuse for non-technical purposes (e.g., business development in professional services or software firms or due to the "looks-good-on-my-CV" factor). Earlier technology trends have also suffered this fate – including (but not limited to) SOA.

Answers to these questions will not only help me to complete my analysis en route to a full comparison, but will also help application and integration architects to successfully create and implement microservices architectures. Hence, I look forward to the SATURN workshop, and I am curious to find out whether and how my post-workshop position will differ from the pre-workshop one articulated in this paper.

References

1. Agile Manifesto, http://www.agilemanifesto.org/principles.html
2. Bass, L., Clements, P., Kazman, R., Software Architecture in Practice, Second Edition. Addison Wesley, 2003.
3. Evans, E., Domain-Driven Design. Tackling Complexity in the Heart of Software. Addison Wesley, 2003.
4. Fowler, M., Patterns of Enterprise Application Architecture. Addison Wesley, 2003.
5. Julisch, K., Suter, C., Woitalla T., O. Zimmermann, O., Compliance by Design. Bridging the Chasm between Auditors and IT Architects. In: Computers & Security, Volume 30, Issue 6-7 2011, Elsevier
6. Mason, R., ESB or not to ESB Revisited – Part 1, http://blogs.mulesoft.org/esb-or-not-to-esb-revisited-part-1/
7. Newman, S., Building Microservices – Designing Fine-Grained Systems, O'Reilly, 2015.
8. Richardson, C., Microservices: Decomposing Applications for Deployability and Scalability, InfoQ 2014, http://www.infoq.com/articles/microservices-intro
9. Reading list for SEI SATURN 2015 Microservices Architecture Workshop (as of April 8, 2015), https://github.com/michaelkeeling/SATURN2015-Microservices-Workshop