Point to Point Responses to Comments

January 30, 2023

Comment	Response
The authors identified a list of microservice patterns to research, and validated those with interviews. According to the discussion chapter, there are 3 major types of feedback that is relevant IMHO - it just does not come across that way yet: Type 1 -patterns that are already recognised as being relevant and actually used by the industry experts (like API gateway and anti-corruption layer) Type 2 -patterns that the experts might be interested in checking Type 3 -patterns that the experts actually recommend against (like CQRS) Recommendation: it would be nice to see a table that summarises this information as part of this article's conclusion or discussion. Recommendation: for future work, types 2 and 3 are probably most interesting to focus on, besides confirming the applications of type 1. For instance, checking the applicability of type 2 patterns, and/or challenging the inapplicability of type 3 patterns by proving the critics wrong?	Row 1, Column 2

Microservices (and patterns) are one type of architecture, mostly driven by the need for independent teams to deploy independently (either in release cadence or technologies employed). Microservices make this more flexible at the expense of some overhead (both co-ordination and network / performance as mentioned in the text). There are other patterns that may be interesting, but not necessarily "services" deployed independently but rather "intra-service" modules that do, for instance, input data validation. It would be interesting to see future work also incorporate that, since microservices are not necessarily the best architectural style. Many people and teams actually go against it, and Fowler himself said that many successful microservice projects started off as monoliths - refactored along the way. So I would suggest "architectural patterns" as a broader scope for future research work. On the side: refactoring is (in my experience) the best way to apply patterns since this ensure that patterns are only used to simplify existing code - rather than being a holy grail from the outset. I have had many worries over team members that learned a pattern in some training and started using it all over the code base the weeks after (alas, also in cases where it made things worse).	Row 2, Column 2
Some typos identified (I probably missed some): -page 17: "gonna" -¿ should be "going to" -p11: "to sketch" -¿ should be "to a sketch" -p1: "descriebd" -¿ should be "described"	Row 3, Column 2
A (CQRS style) event store could be an ideal way to refactor BD architectures afterward, since one can re-init a new system based on the historical "replay" of all past events.	Row 3, Column 2
Competing consumers don't really need a circuit breaker in my understanding?	Row 3, Column 2
I strongly recommend the authors look at the "event storming" approach to identify and structure DDD / CQRS / microservice systems.	Row 3, Column 2