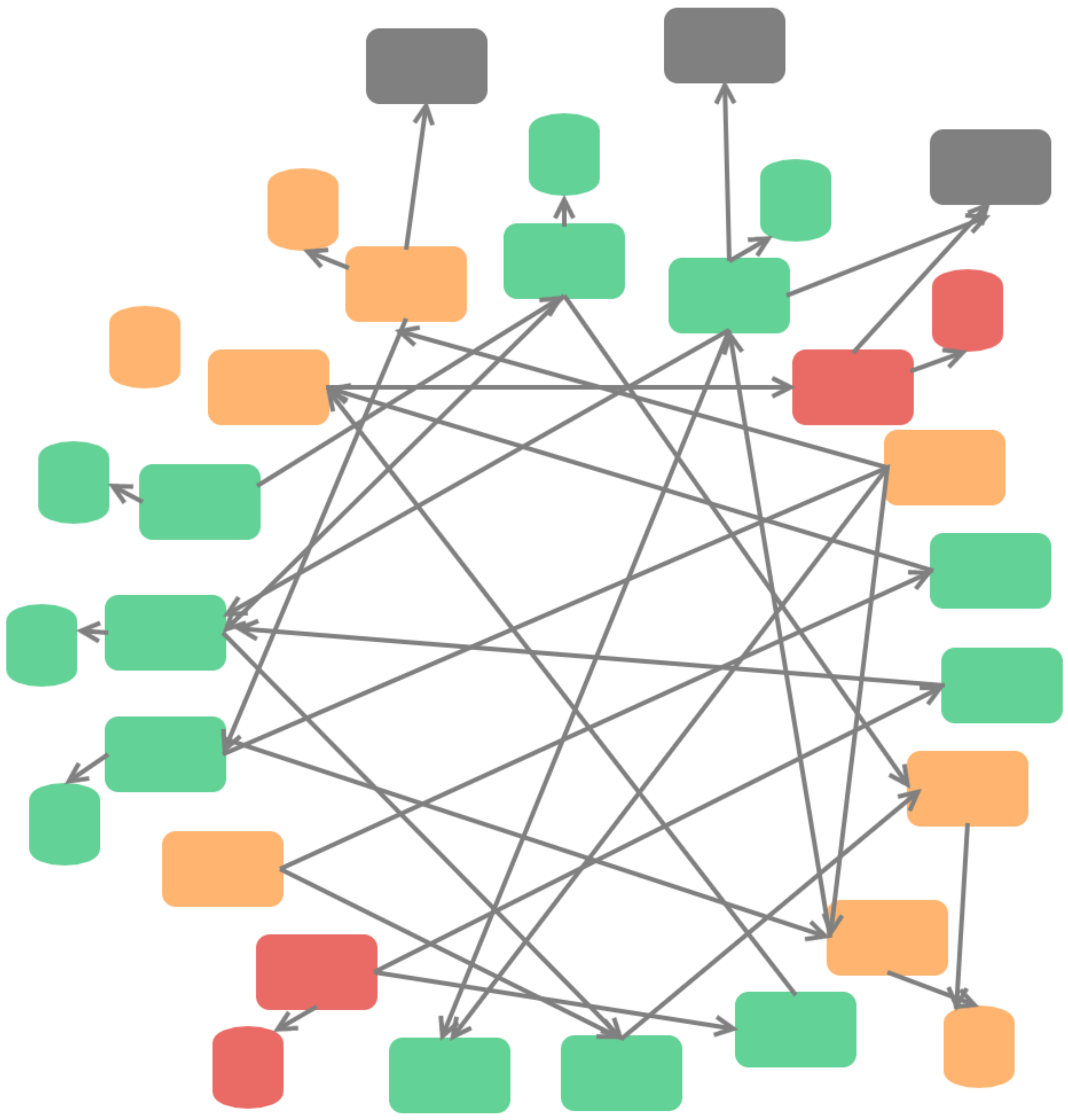
medium.com

How small should Microservices be - Grygoriy Gonchar - Medium

Grygoriy Gonchar

5-6 minutes



Death star of microservices

One of the most frequent questions within an implementation of microservices architecture is about size and granularity of microservices: how small should be microservices, should a piece of software be split into multiple microservices or built as a single one. Trying to answer these questions I came to certain criteria which help me to make a decision.

Rather than focusing on pizza-size teams or single responsibility principle, it's better to stay pragmatic and focus on possible advantages and drawbacks of the design decision and challenge why a particular piece of software needs to be split. To do this we need to model a part of the system we want to build as multiple microservices candidates: define bounded contexts, main entities and service capabilities around these entities. With having a rough picture in front we can think around benefits and drawbacks of such design from multiple perspectives.

Isolation of changes perspective: do you expect changes in requirements of one candidate to be independent of another? By isolating changes risk to break something that wasn't changed is reduced. Usually changes in one Bounded Context change independently of others. Organization of microservices around business capabilities is a key success factor here. Since microservices provide a high level of isolation they can help isolate changes.

Isolation of static services: do we isolate services that don't change frequently. While some requirements change frequently others might change so rarely that we can treat microservice as *static*. Examples of such static microservices could be services responsible for sending emails or SMS, document or image format conversion, localization or translation etc. In most of the system, such functionality changes rarely while main business features might change quickly. By isolating static functionality we reduce the risk of breaking it and the necessity of testing.

Isolation of business-critical services: do we isolate business critical operations form less critical. In some systems, we can divide functions that are vital and stop the system from being useful when they are unavailable. Such functionality for web-shop could be purchasing functionality while recommendations system or wishlist are not that critical. Business critical operations must be as much isolated as possible from non-critical functionalities. This will minimize probability to affect business-critical operations by troubles with non-critical functionality. Separately isolated microservice for pure critical functionality is usually a good idea.

Cohesion perspective: is one of the candidates fully independent in runtime from another? Mutual dependencies are a sign of high cohesion and bad candidates for different microservices.

Team setup perspective: do you expect those candidates to be implemented by different development teams? Microservices approach is especially good to isolate the work of one team from another.

Technology perspective: can the candidates be implemented more effectively with different technologies? Microservices make possible polyglot implementations where different programming languages and frameworks might be used.

Data consistency and transaction boundaries perspective: can you accept eventual consistency between candidates? If not this might be an issue since strong consistency in distributed systems is a complex topic. Are there any atomic business operation across the candidates? This might be not trivial to achieve with separate services.

Non-functional requirements perspective: are performance and availability requirements for those candidates different? It’s easier to deal with small and simple performance models where critical operations are isolated and implemented in the most optimal way.

On the other hand, each remote call is latency and risk of failure. Availability of the business process relying on multiple microservices by default is lower then if the process relies just on one microservice.

Security perspective: is one of the candidates processing high-sensitive data while another is not? Least common mechanism security principle says that it's safer to make processing of highly sensitive data separate from other data.

Give a priority to questions that matter for your system. If the answer is "yes" to most of them then it might be a good idea to continue with those microservices candidates. Reiterating over each candidate and thinking if any further segregated microservices would make sense will result in a pragmatic decision on microservices granularity.