

Microservices trade-offs and fallacies

An overview of the
misconceptions and constraints of
microservices



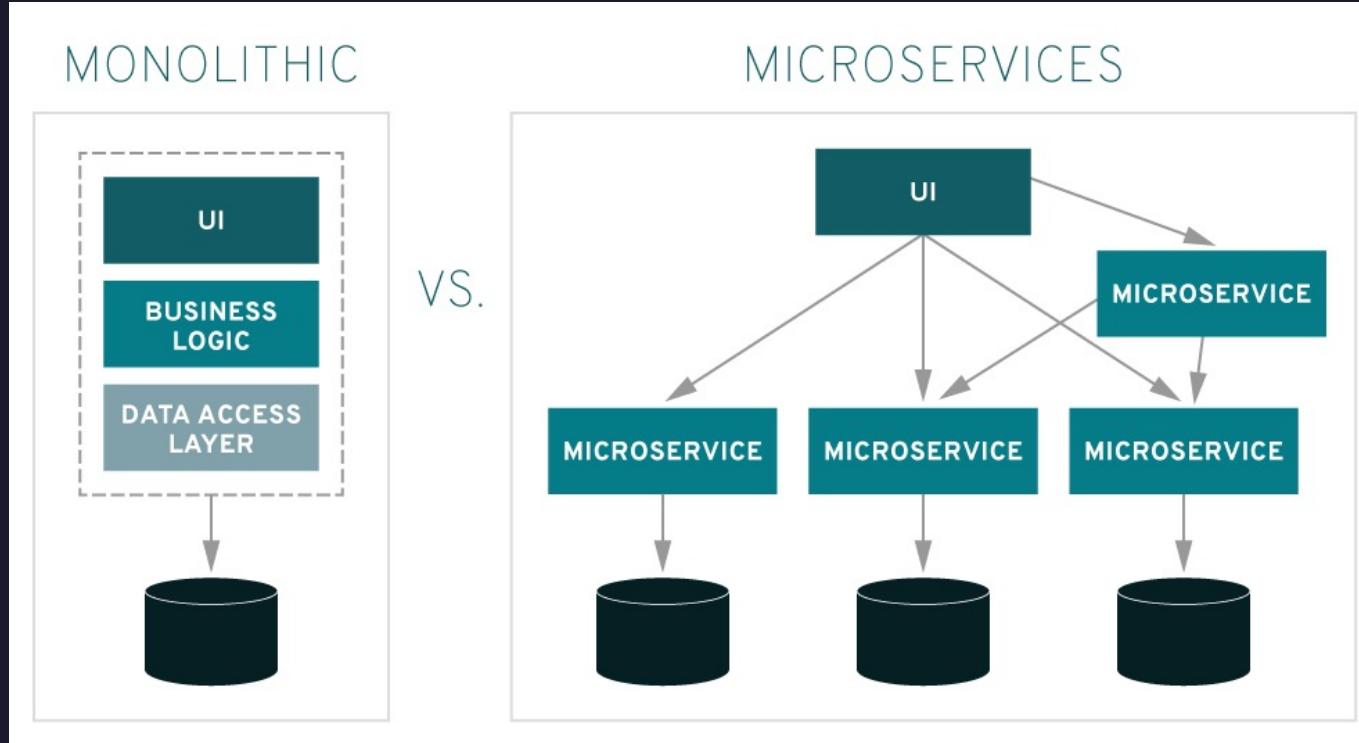
Daniel Barrientos Iglesias
Raúl Núñez García

Index

- Monolithic solutions vs. Microservices: an overview
- The rise of microservices and the general discourse
- Fallacies and misconceptions
- Conclusions
- Alternatives



Brief recap: Monolithic solutions vs. Microservices



The rise of microservices

- The hyperscalers fed the hype
- A lot of expectations and recent developments
- The “trendy” solution

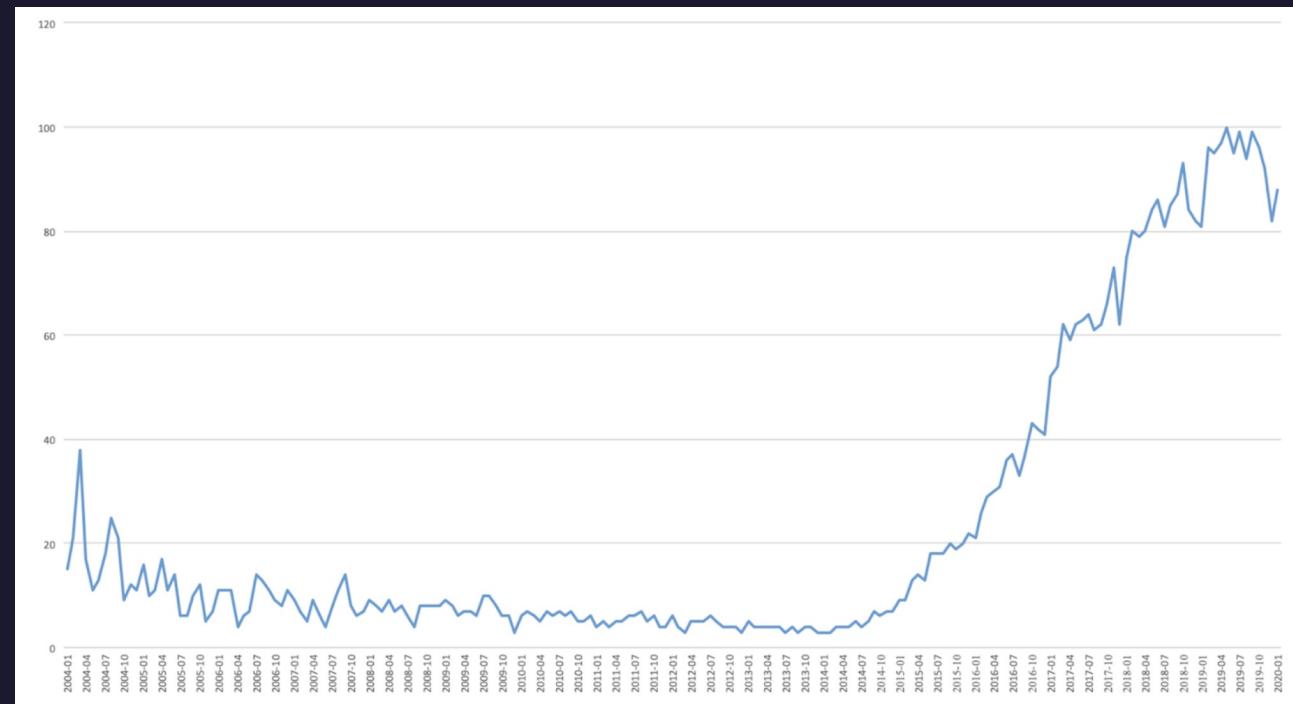


The mainstream discourse

- Widely seen as the way of the future
- “Adaptable and flexible”
- “The monolithic approaches are outdated”



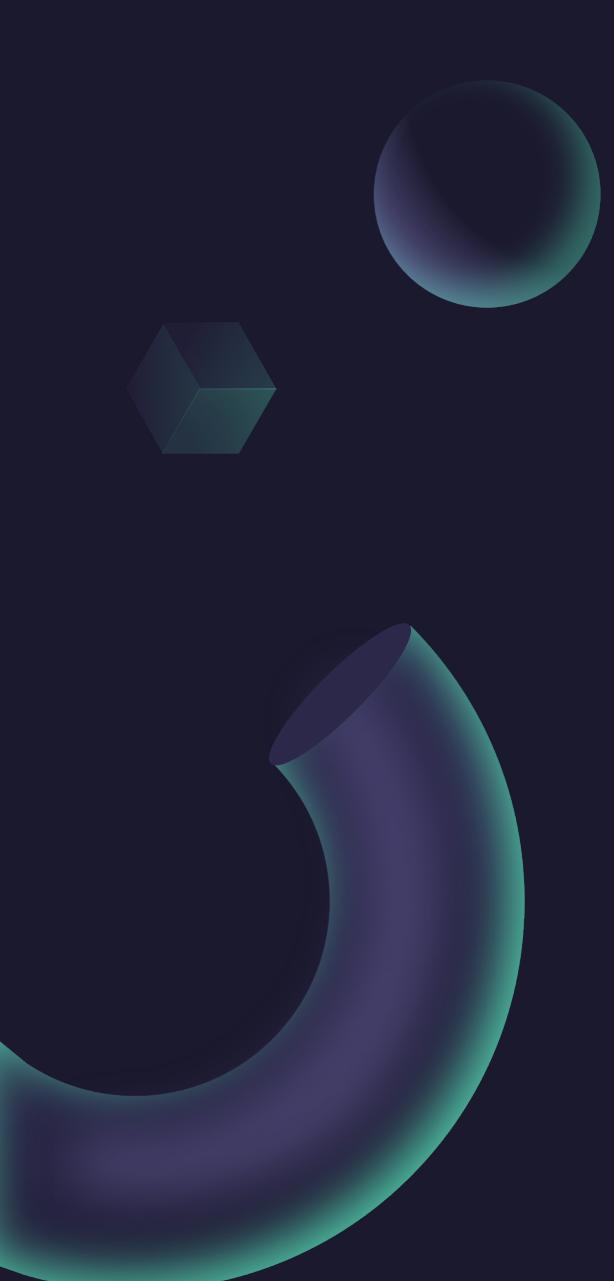
Google Trends metrics for search of microservices over time



False notions and confusion



- Certain ideas about microservices are misconceptions
- Sometimes chosen for the sake of it or out of fashion
- They are not the solution for everyone, and can be detrimental
- The fallacies continue to propose microservices as a silver bullet

The background features several abstract geometric shapes in a dark navy blue space. In the upper left, a large sphere with a purple-to-teal gradient sits above a small, faceted hexagonal prism. In the lower left, a large, translucent cylinder with the same gradient overlaps a solid dark circle. The text is positioned in the center-right area of the slide.

The fallacies themselves

Fallacy n°1: Scalability

- “We need microservices to scale our applications dynamically”
- Reality: You don’t need microservices to satisfy regular enterprise scalability demands.



Fallacy n°2: Simplicity

- “Microservices are simpler than monoliths”
- Essential complexity persists
- Microservices failures:
 1. Crash failures
 2. Omission failures
 3. Timing failures
 4. Response failures
 5. Byzantine failures



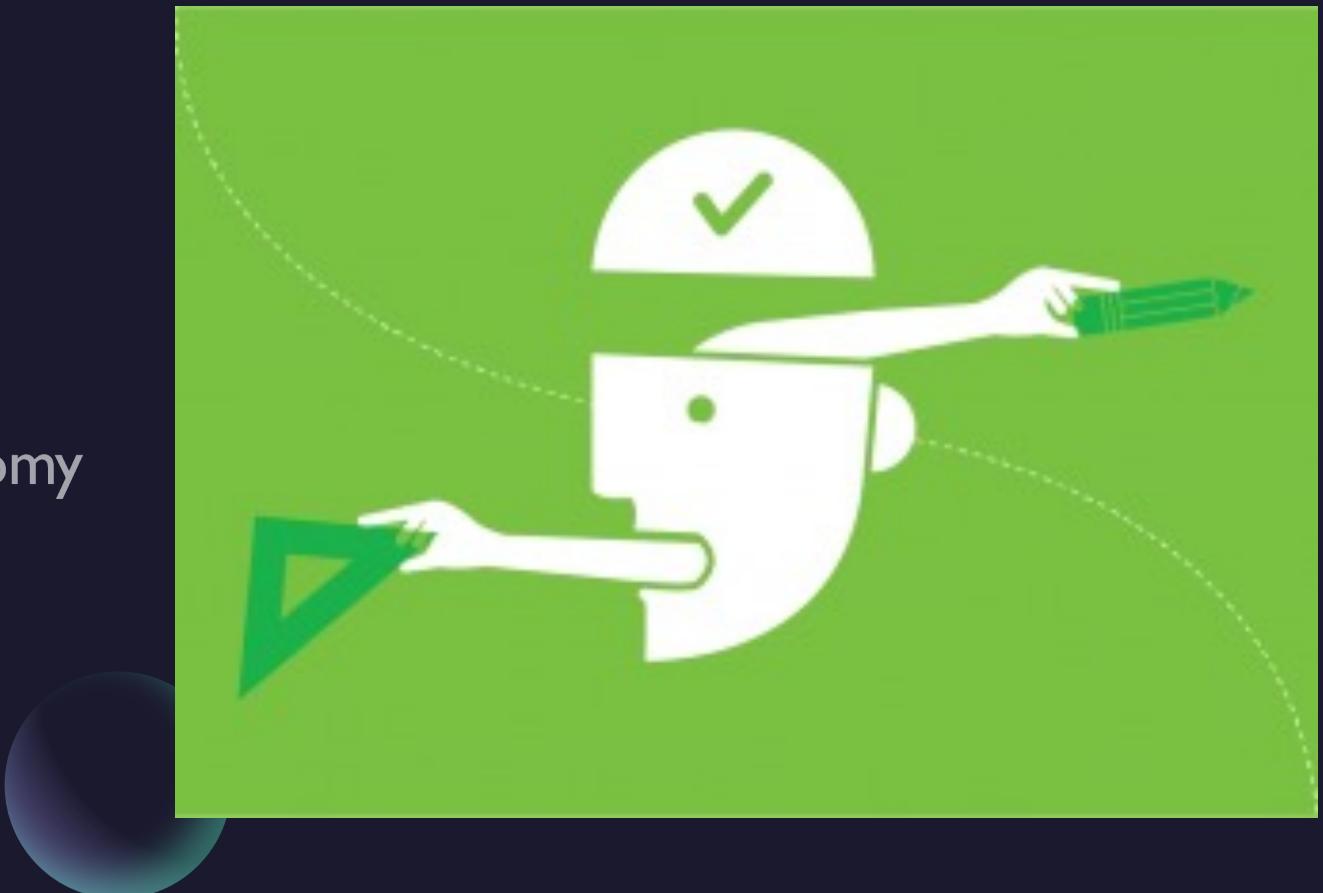
Fallacy n°3: Reusability

- Aiming for reusable microservices means deliberately crippling availability.



Fallacy n^o4: Autonomy

- Microservices cannot improve autonomy by themselves





Fallacy n°5: Microservices lead to better solution design

- Microservices enforce their boundaries due to their nature
- Monoliths can be well-structured as well
- Definitely, the chosen runtime structure doesn't impede the source code structure by any means

Fallacy n°6: Microservices make technology changes easier

- *It takes the same effort to migrate an existing system to a new technology as it took to develop the system to its current state*
- It's true that it's a bit easier...
- Do not blame monoliths... The guilt is on
 - I. Broken IT governance processes
 - 2. Application dependencies on the OS level



Conclusions

Reasons to use microservices

- You need to move fast
- You have very disparate NFR's



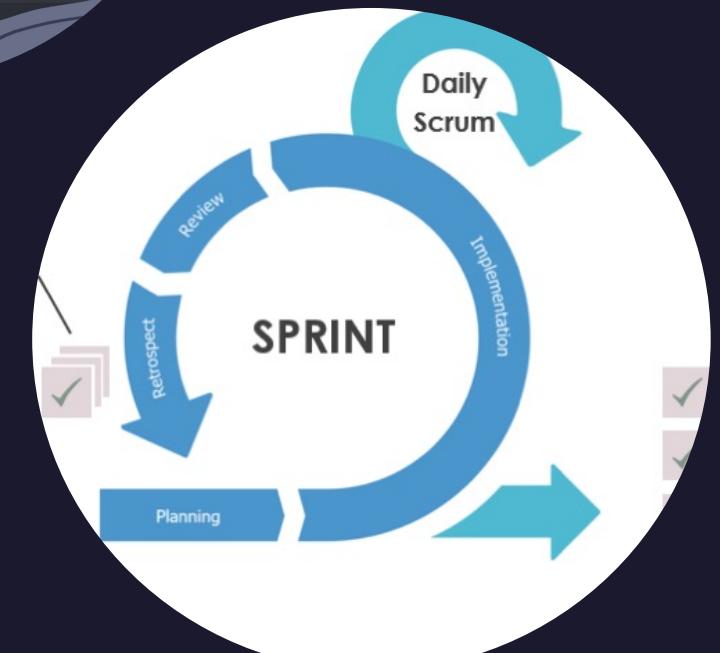
"IT DEPENDS"

When to use
microservices

YEAH.. NOW I DON'T HAVE TO
EXPLAIN ANYTHING...

The good

- Suitable if you have short cycle times
- Also if you have tons of concurrent requests and very high availability requirements



The bad

- If you are not able to cope with the side aspects of the microservices you should not try to implement them



The ugly

- Microservices don't suppose any advantage if you are a single team

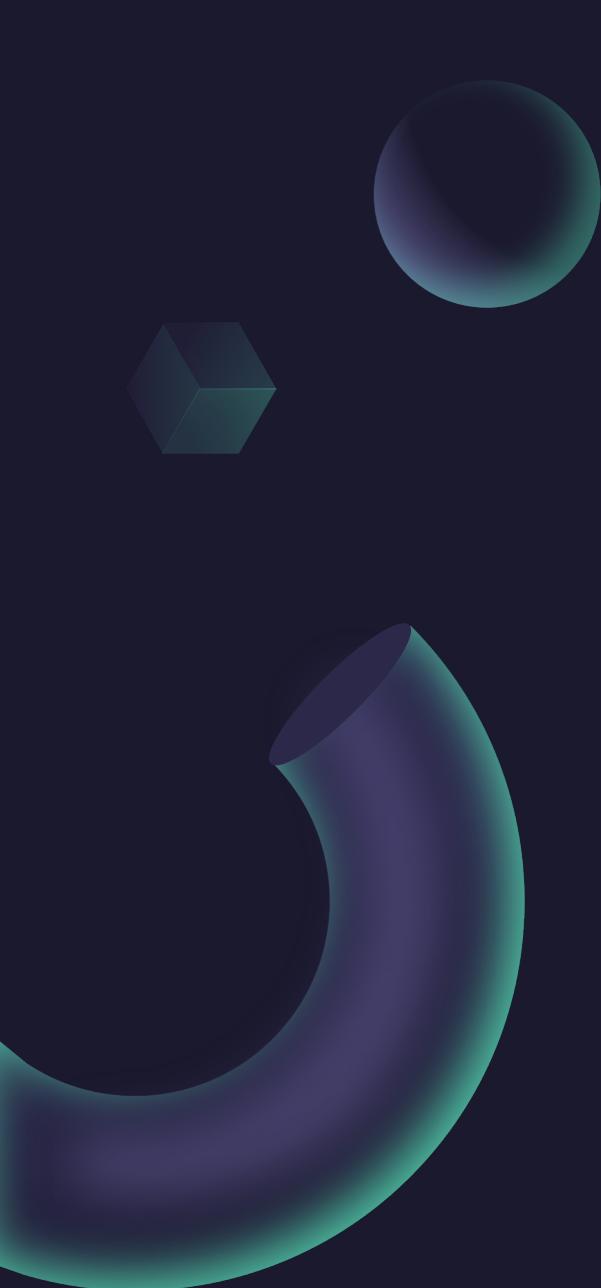




¿What are the alternatives?

What happens if we...

- Do not need the speed?
- Do not have multiple teams?
- Do not want to pay the price for microservices?



First alternative: Moduliths

What are those?

- Division of monoliths into clearly defined and isolated modules
- Solves the problem of the maintenance, which can be a nightmare in monoliths
- Good idea if we:
 - I. Do not need to move fast and/or do not have multiple teams (and exponential scaling is not in sight)
 2. do not have any special runtime requirements in terms of very disparate non-functional requirements



Moduliths constraints



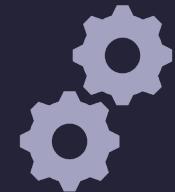
Requires decent design skills
to find the right module
boundaries and contents



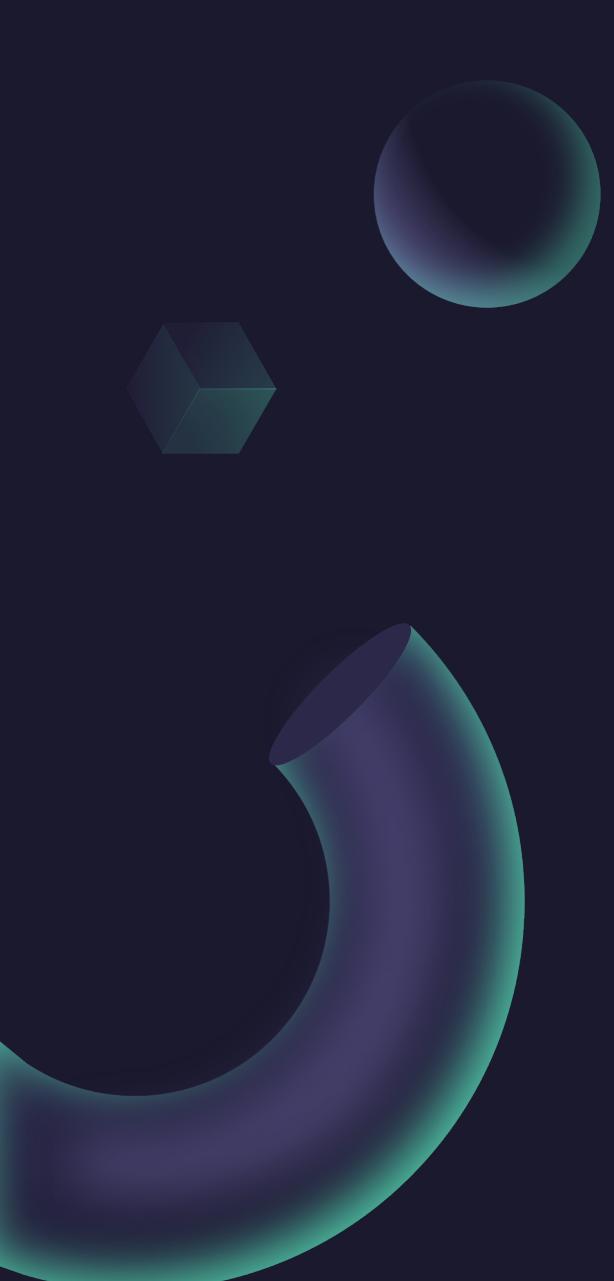
Requires that everyone
adheres to the rules



The modules should be
weakly coupled if the design
is OK



Code from modules
shouldn't be traced.

The background features abstract, semi-transparent 3D geometric shapes in dark teal and purple. A large, rounded rectangular shape is on the left, a smaller sphere is at the top center, and a hexagonal prism is on the far left.

Second alternative: Microliths



What are those?

- Service that is designed using module design principles, but that avoids calls between them
- Good idea if we:
 1. Do not need to move fast or have multiple teams
 2. Have special runtime requirements in terms of very disparate Non Functional Requirements
 3. We meet the preconditions for using microservices, but we are not willing or able to pay for them

Microliths constraints

- Require some status and data reconciliation technique
- To satisfy this we have to meet several design features:
 1. All functionality needed for a use case must be implemented inside a single microlith.
 2. All data required to serve an external request must be in its database.
 3. Reusable functionality needs to be provided via libraries or some other implementation or build time modularization mechanism, not via other services.



Thank you for your attention

Question time