Lund&Bendsen
developing developers

Architecture

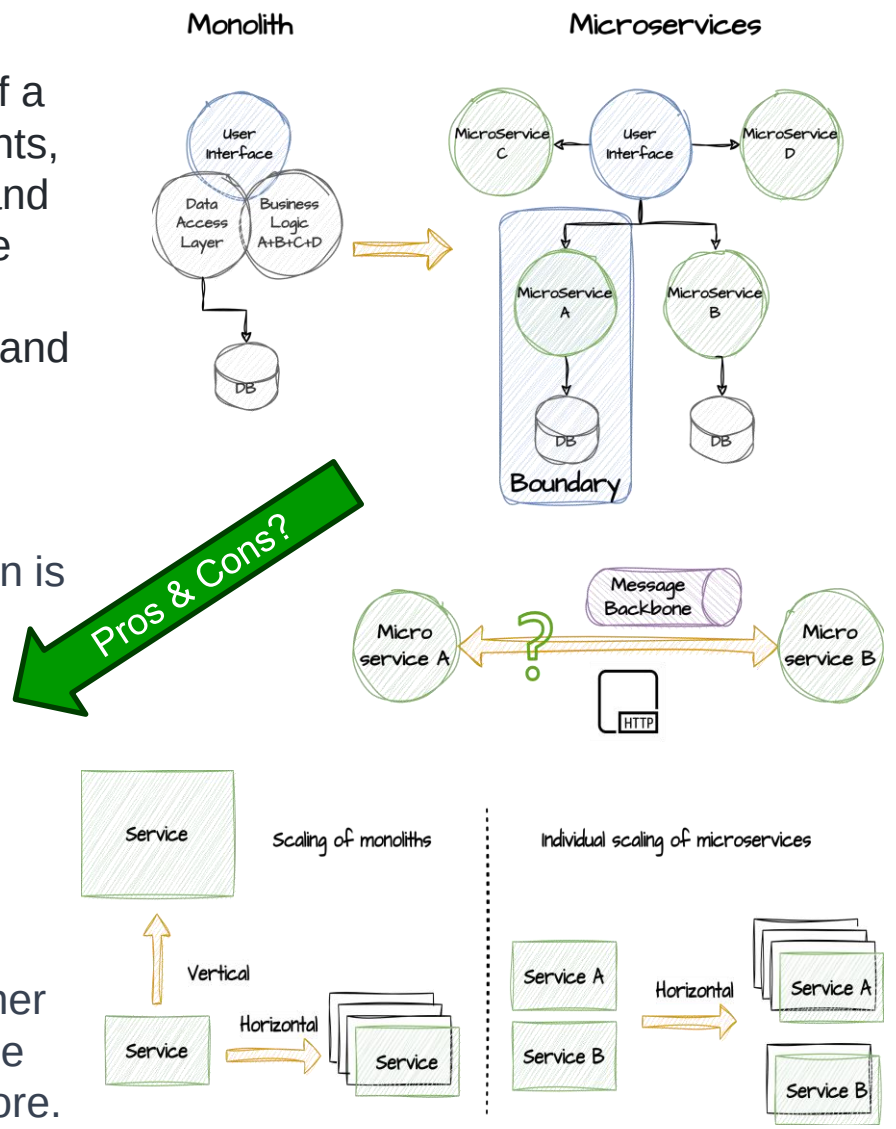
Monolith & microservices

Lund&Bendsen
developing developers

Software architecture refers to the fundamental structure of a software system. It's a blueprint that outlines the components, their relationships, and the principles guiding their design and evolution. This design includes various aspects such as the system's components, their functionalities, the interactions between them, and the properties that govern their design and evolution. Often when discussions on enterprise software architecture is made, two different approaches are being mentioned. The non-distributed and the distributed:

Monolith: In a monolithic architecture, the entire application is built as a single, tightly integrated unit. All components and modules are interconnected, and there is typically a single codebase and database. Calls between modules are in-process.

Microservices: Microservices architecture divides the application into small, isolated, independently deployable services. Each service focuses on a specific business capability (single responsibility) and communicates with other services via well-defined APIs or preferably over a message backbone. Each microservice typically has its own state store.

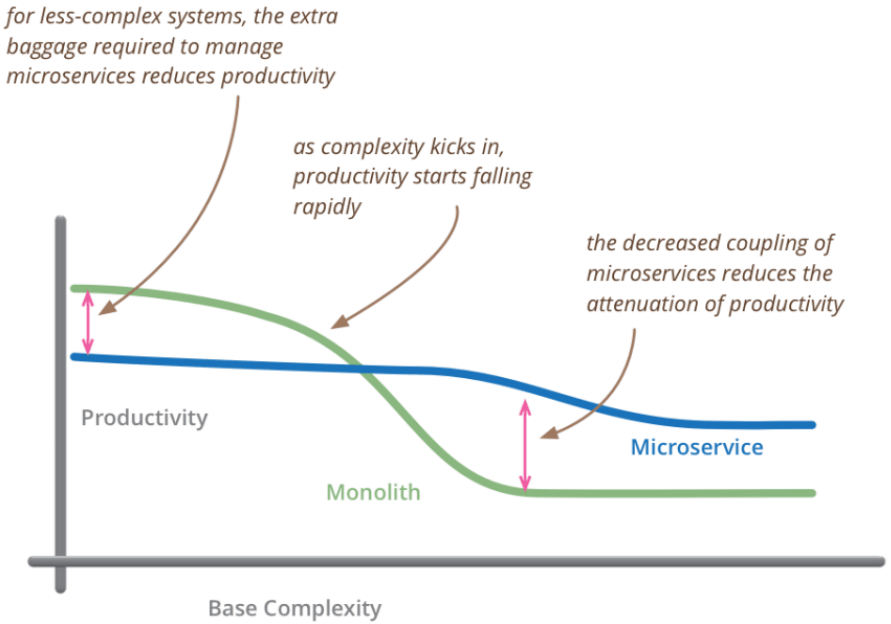


Monolith & microservices

Lund&Bendsen
developing developers

Monolith and **microservice** architectures are two fundamentally different approaches to designing and building software systems. They differ in various aspects, including their scalability, development process, and deployment.

The choice between a monolithic and microservices architecture depends on various factors, including the size and complexity of the application, the development team's expertise, scalability requirements, and organizational preferences. It's important to carefully consider these factors when deciding on the architecture for a software project.



but remember the skill of the team will outweigh any monolith/microservice choice

Microservice architecture

Lund&Bendsen
developing developers

Scalability: Microservices allow for granular scalability, where you can scale individual services independently based on their specific resource needs.
Flexibility: You can use different technologies and programming languages for different microservices, allowing you to choose the right tool for each job.
Isolation: Failures in one microservice generally don't impact the entire system, thanks to isolation, reducing the blast radius of issues.
Continuous Deployment: Microservices are conducive to continuous deployment and easier rollbacks since changes are limited to individual services.

Complexity: Microservices introduce additional complexity in terms of service discovery, load balancing, and inter-service communication.
Testing and Debugging: Testing and debugging can be more complex, especially when dealing with interactions between multiple services (integration testing).
Operational Overhead: Managing and monitoring multiple services can require additional operational overhead.
Latency: Inter-service communication over a network can introduce latency compared to in-memory calls in a monolithic application.

Monolithic architecture

Lund&Bendsen
developing developers

Simplicity: Monoliths are typically simpler to develop and manage, especially for smaller applications, as there's only one codebase to deal with.
Performance: Monoliths can be more efficient when all components are tightly integrated, as there's no overhead for inter-service communication.
Easier Debugging: Debugging and troubleshooting can be simpler in a monolithic application because there are fewer moving parts and dependencies to consider.
Testing: Testing is often easier since the entire application can be tested as a whole.

Scalability: Scaling a monolithic application can be challenging, as you often need to scale the entire application even if only certain components require more resources.
Maintainability: As monolithic applications grow, they tend to become harder to maintain and update due to their size and complexity.
Technology Stack: You're limited to a single technology stack and programming language for the entire application.
Deployment: Deployments can be riskier because any change affects the entire application, potentially leading to downtime.

Martin Fowler

Lund&Bendsen
developing developers