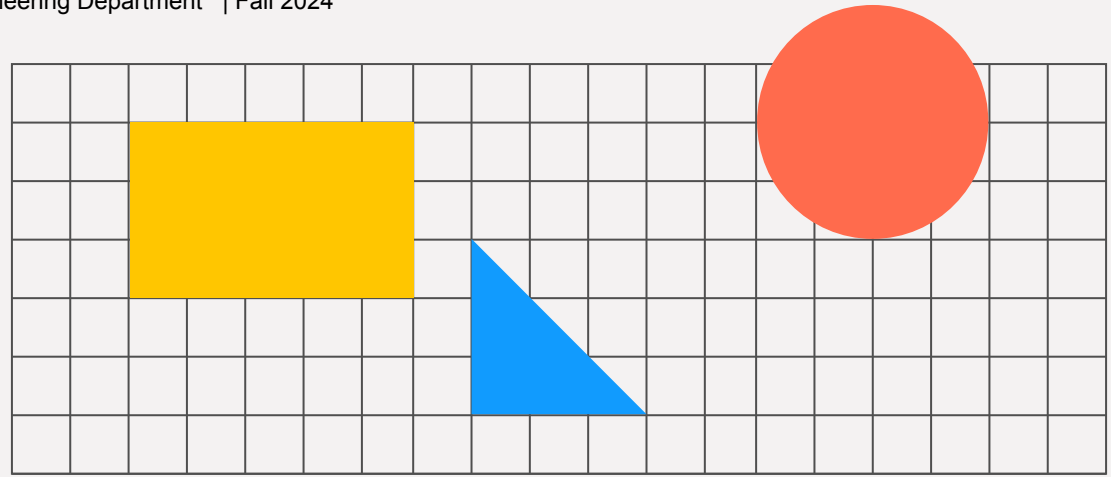
Microservices Architecture

A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges *2023*

Keywords:
- Microservices
- Monolithic
- Decomposition
- Principle
- Patterns
- Migration

Alireza Soltani Neshan, Mohadese Salem, Milad Shadan Kohankhah

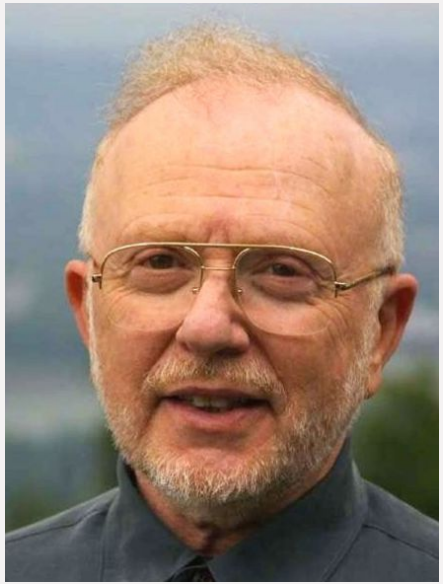
Computer and Software Engineering Department | Fall 2024



Thanks to

Martin Fowler
Software Engineer

Sam Neuman
Software Architecture Engineer



David Parnas
Information Hiding forerunner

Software Architecture Activities

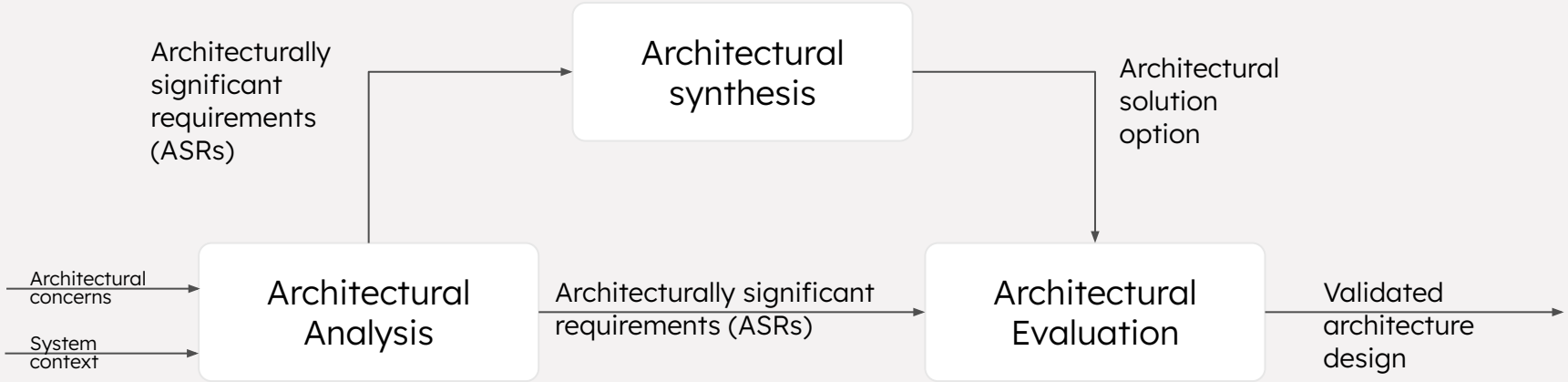
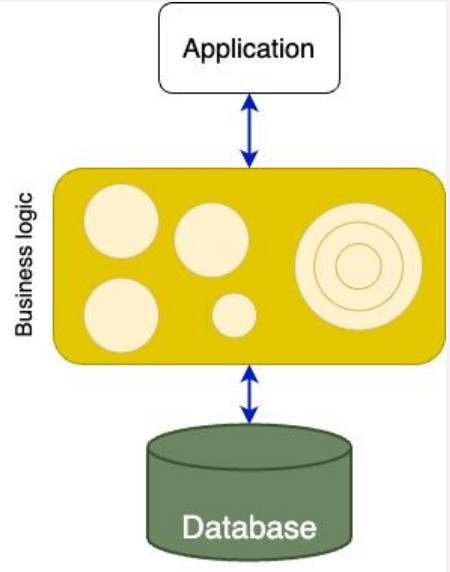
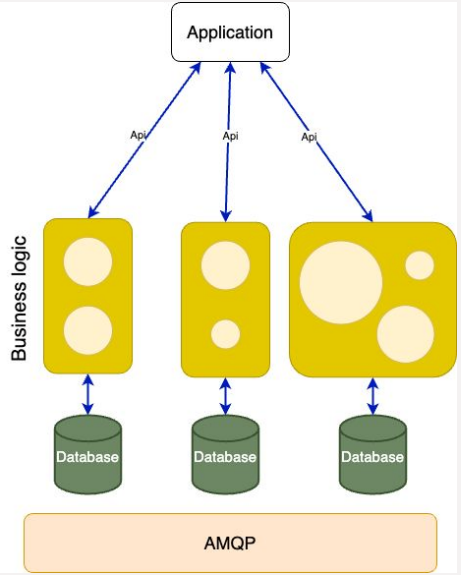


FIG 1. Architecture activities from **Wikipedia**.

Styles

Software Architecture styles





Monolithic

MicroService

POV

Another Point of View from Monolithic and Microservice styles

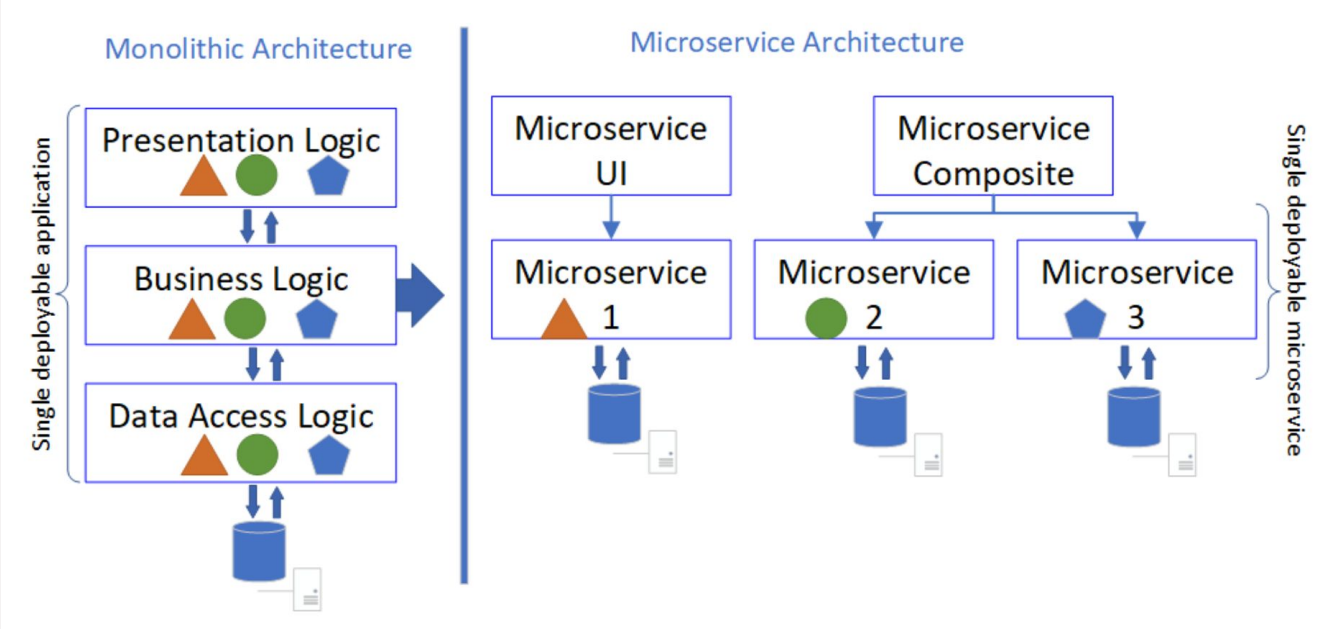


FIG 2. Monolithic vs. Microservice Architecture activities.

When?

When should we migrate from a monolithic architecture to microservices?

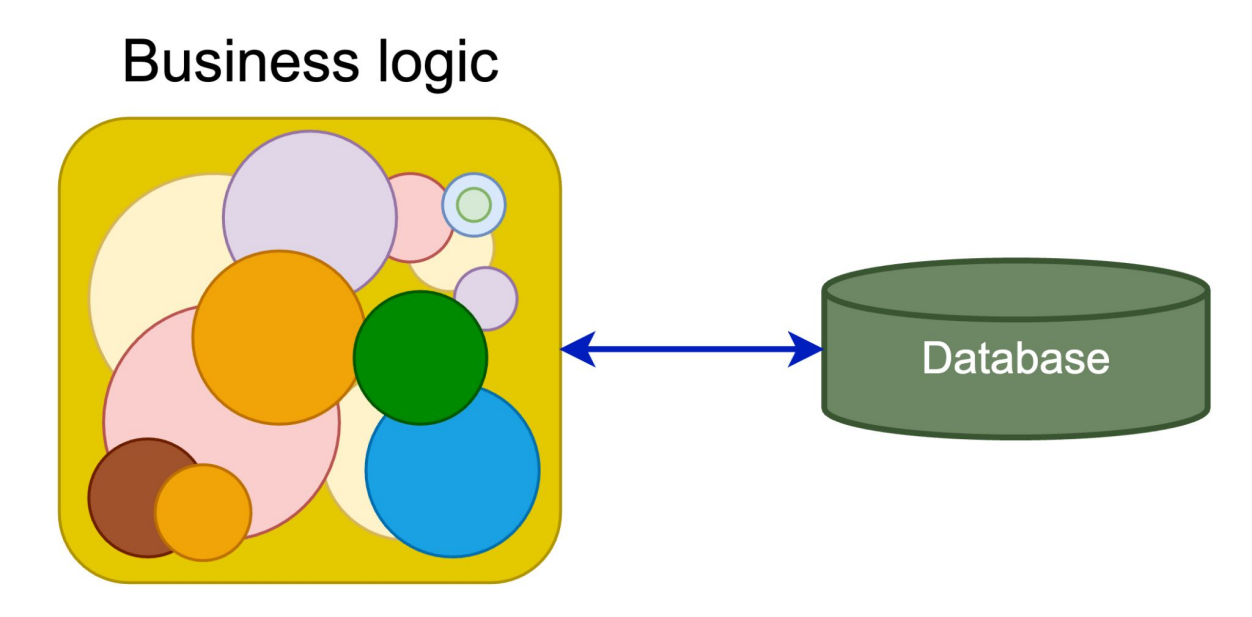


FIG 3. Monolithic Style, Complexity, Failures

When?

When should we migrate from a monolithic architecture to microservices?

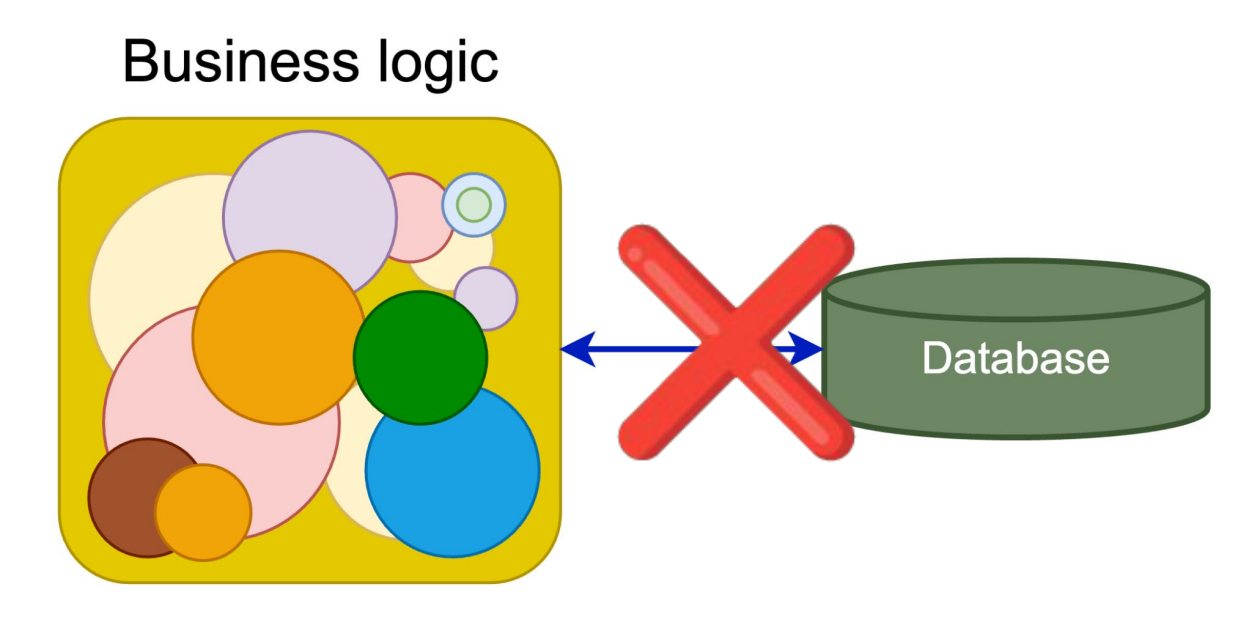


FIG 3. Monolithic Style, Complexity, Failures

When?

When should we migrate from a monolithic architecture to microservices?

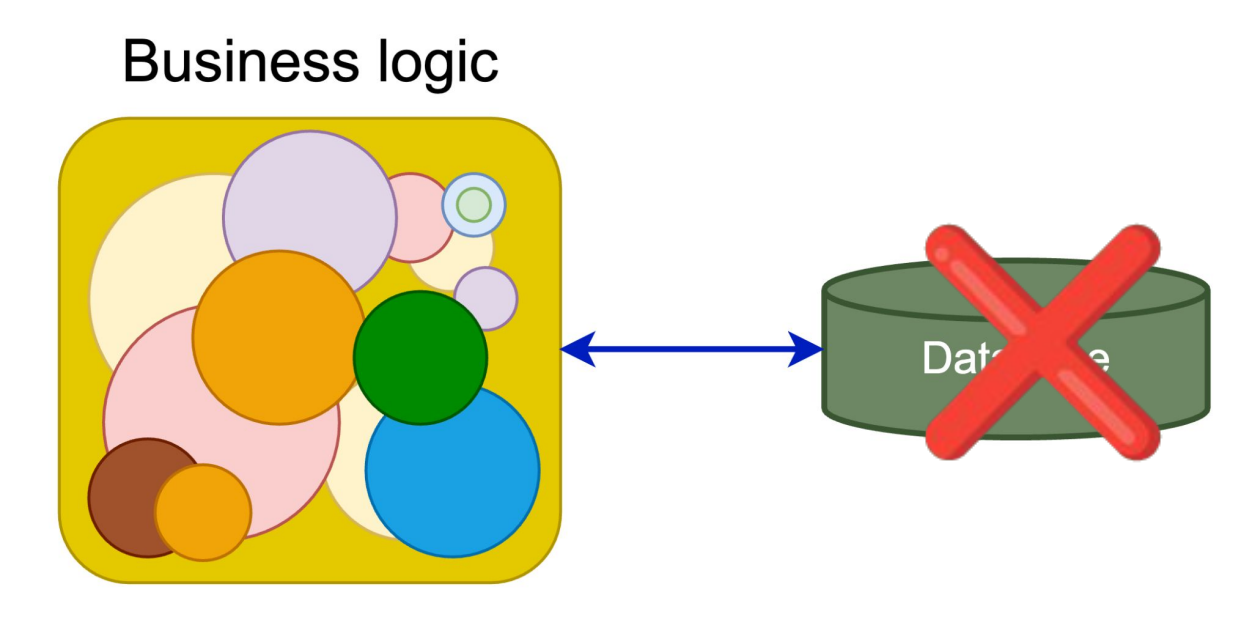


FIG 3. Monolithic Style, Complexity, Failures

When?

When should we migrate from a monolithic architecture to microservices?

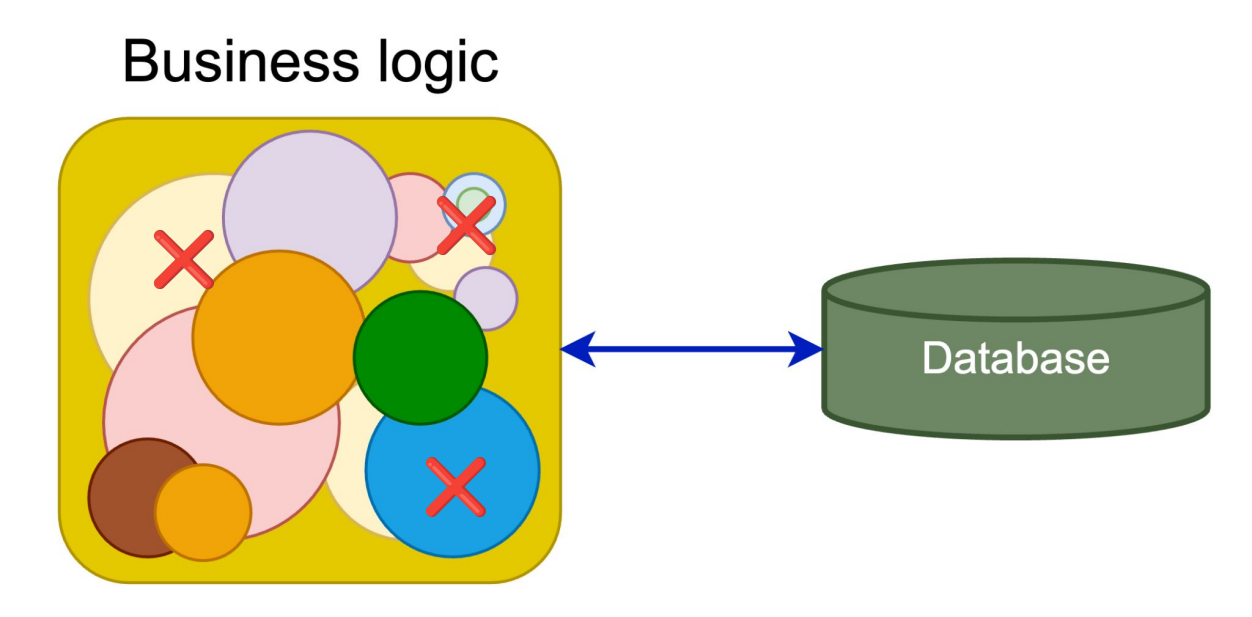


FIG 3. Monolithic Style, Complexity, Failures

Availability

Escalating restart

Degradation

No matter how small, can affect the entire system behavior

Horizontal, vertical, containerization scaling

If a module is the most used, it is not possible to scale only this modules, but entire monolith must be scaled, consequently increasing costs.

High Availability

Horizontal scaling

If a module is the most used, it is not possible to scale only this modules, but entire monolith must be scaled, consequently increasing costs.

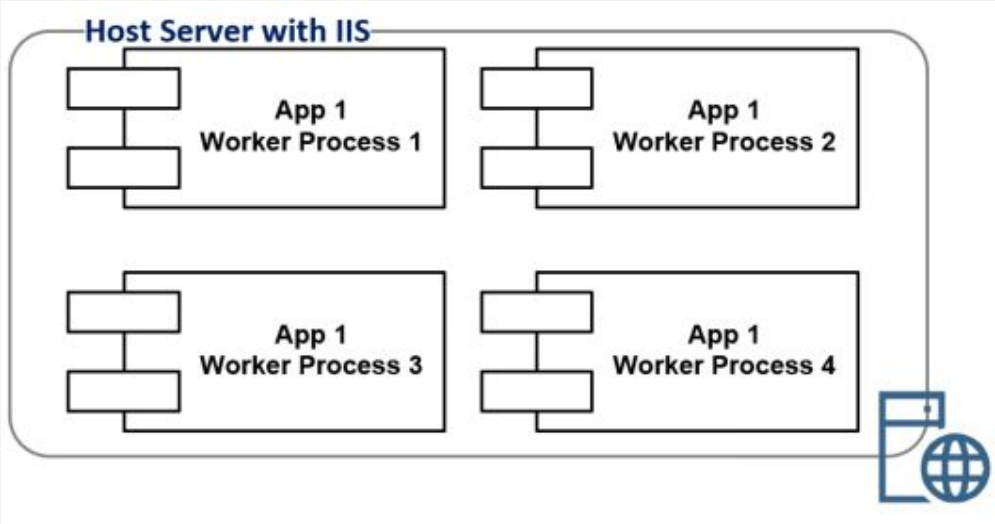


FIG 4. Monolithic scalability duplicating worker process.

Vertical scaling

If a module is the most used, it is not possible to scale only this modules, but entire monolith must be scaled, consequently increasing costs.

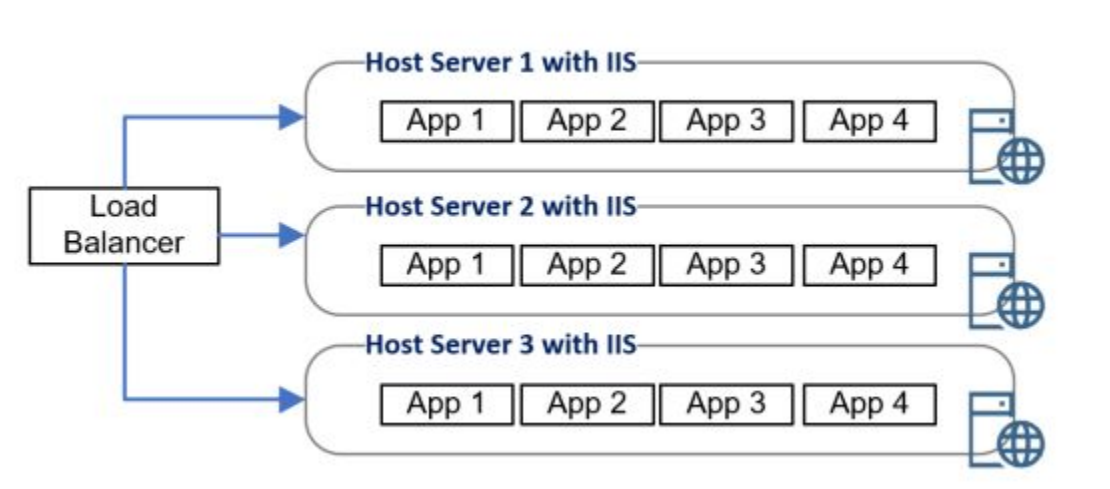


FIG 5. Monolithic scalability in servers.

Containerization scaling

If a module is the most used, it is not possible to scale only this modules, but entire monolith must be scaled, consequently increasing costs.

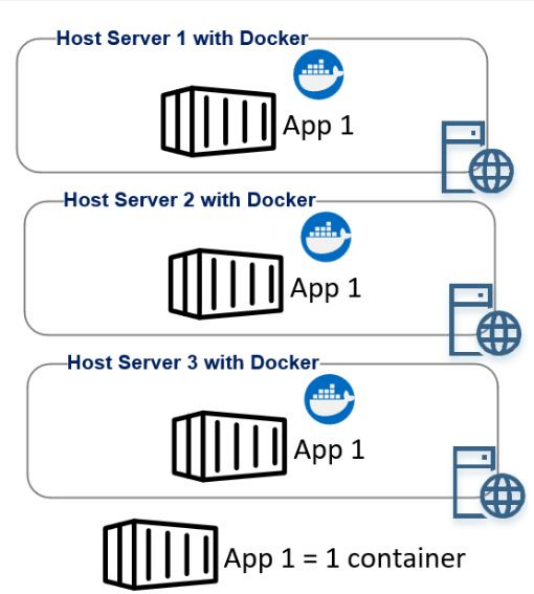


FIG 6. Monolithic scalability in containers.

4 properties of SOA

Loosely coupled

+ Undependability
+ Flexibility & SoC (Separation of Concerns)
+ Easy update
+ Better NSF

Reusability

+ Modular design
+ Development & maintenance cost reduction

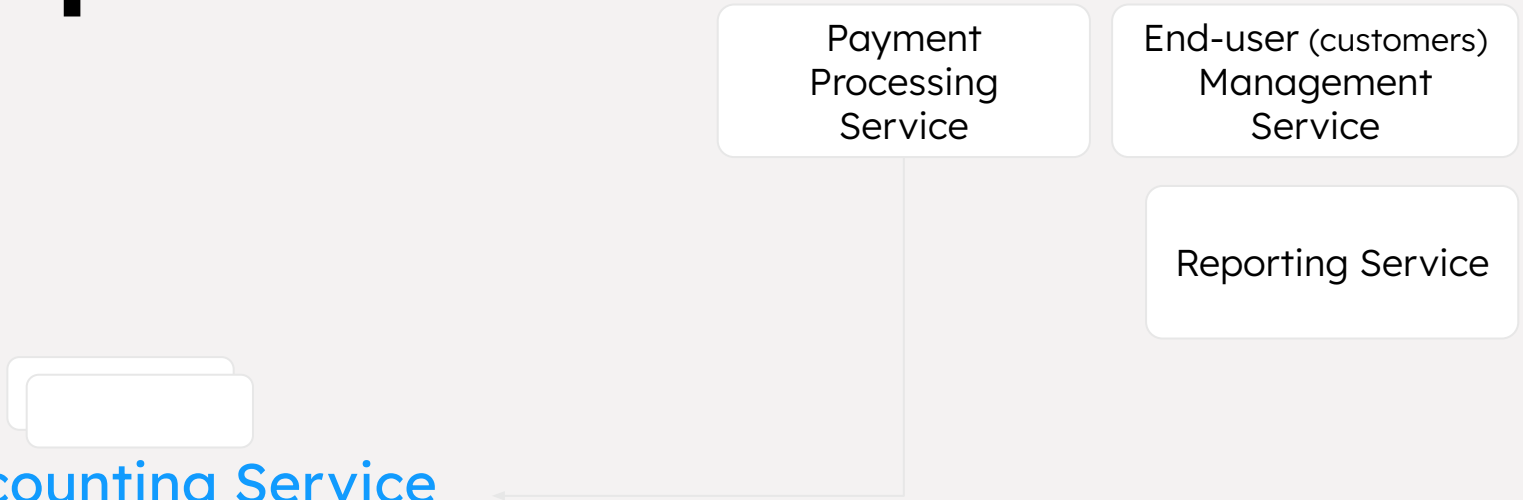
Platform Independence

+ Use any programming languages
+ Use any efficient hardware
+ Use any appropriate infrastructure

Composability

+ Fine-Grained
+ Independence management

Service Composition



Which one?

Which one is better? Microservice or Monolithic architecture?

- Project requirements
- Project constraints

Costs

Microservice

- High initial development costs
- Needs for more resources, tools, and expertise to manage multiple services
- Lower operational costs through independent scaling in long-term

Monolithic

- Minimum initial development costs
- When the application grows, maintenance costs, can escalate du to increased complexity
- Extensive testing and debugging

Monolithic

Pros

- Simplicity: ease of deployment, easy to develop, easier to debugging
- Performance: everything runs within a single process
- Code reuse: no need to ssh, all code will be in one place without accepting distributed systems.

Cons

- Scalability
- High maintenance cost
- limited flexibility in making changes
- compromised availability and reliability
- Longer development times
- Technology lock-in: single technology stack

Microservices

Pros

- Flexible coupling
- High functional cohesion
- Comfort for scaling
- Easy to deploy with DevOps tools
- Resilience

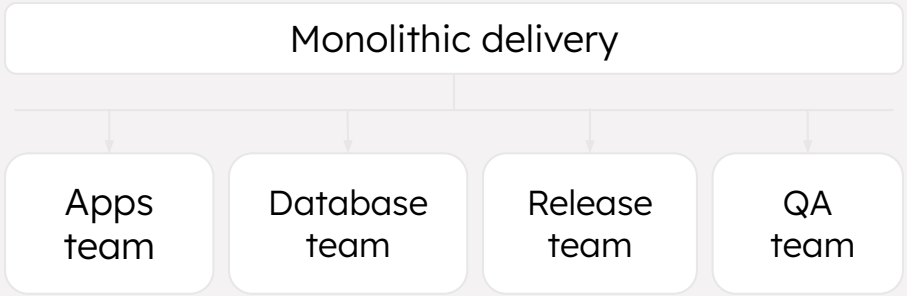
Cons

- Lack of experience: steep learning curve
- Challenges in managing and monitoring
- Increased complexity: debugging, error tracing, leading to higher operational costs
- Network overhead: introduce latency and careful network architecture planning
- Data duplication: each microservices have its own database

Team management

Better team management

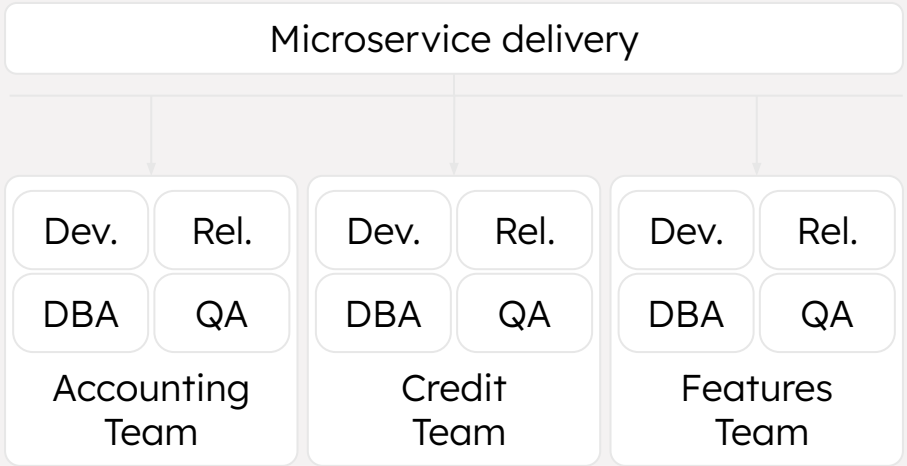
Monolithic organizations



Team management

Better team management

Microservices organizations



Patterns

Methodologies and patterns for microservices architecture

Backend for Frontend Pattern

Shared Data Microservice Design Pattern

Aggregator Microservice Design Pattern

Patterns

Isolate backend from front-end

Backend for Frontend Pattern

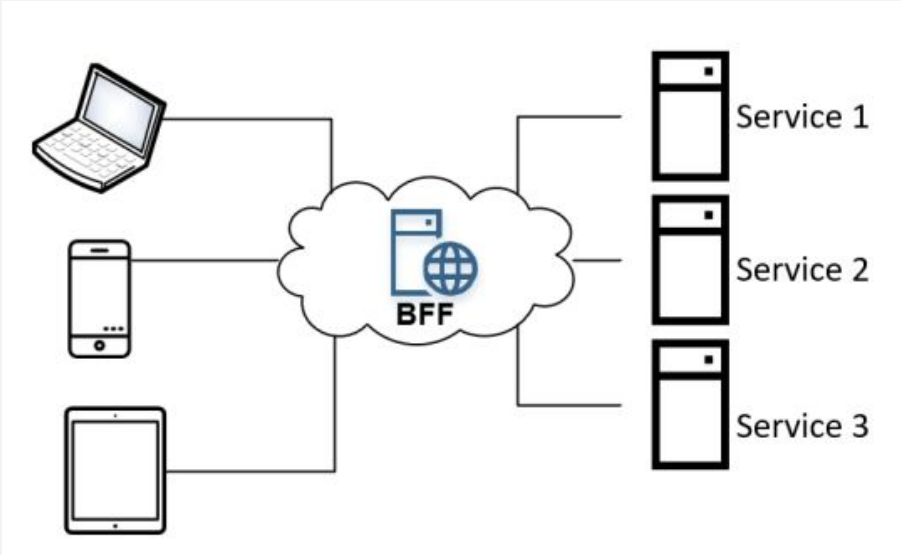


FIG 7. BFF design pattern.

Patterns

Methodologies and patterns for microservices architecture

Backend for Frontend Pattern

Shared Data Microservice Design

Aggregator Microservice Design Pattern

Patterns

Shared data repository, loose coupling and high cohesion

Shared Data Microservice Design

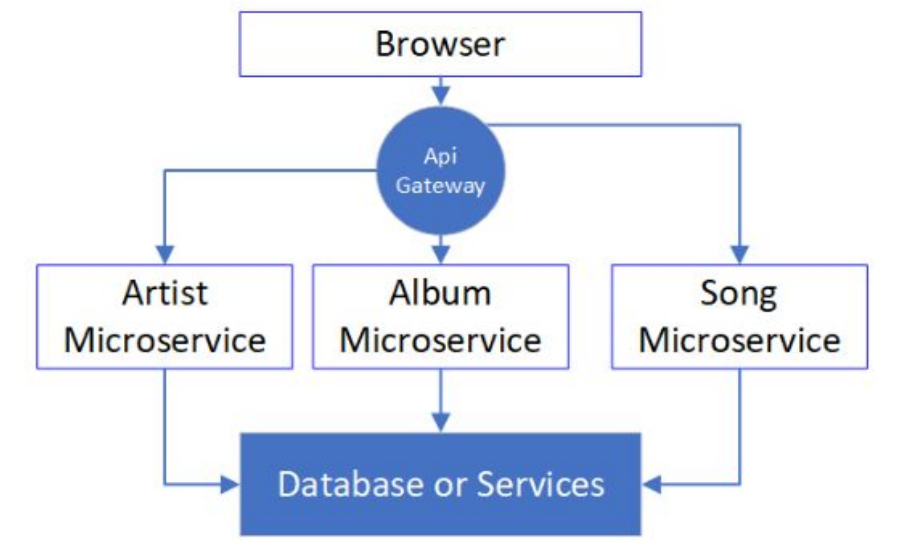


FIG 8. Shared data microservice design pattern example

Patterns

Methodologies and patterns for microservices architecture

Backend for Frontend Pattern

Shared Data Microservice Design

Aggregator Microservice Design

Patterns

Centralized data aggregation, reduced client complexity, improved performance, reduce network overhead

Aggregator Microservice Design

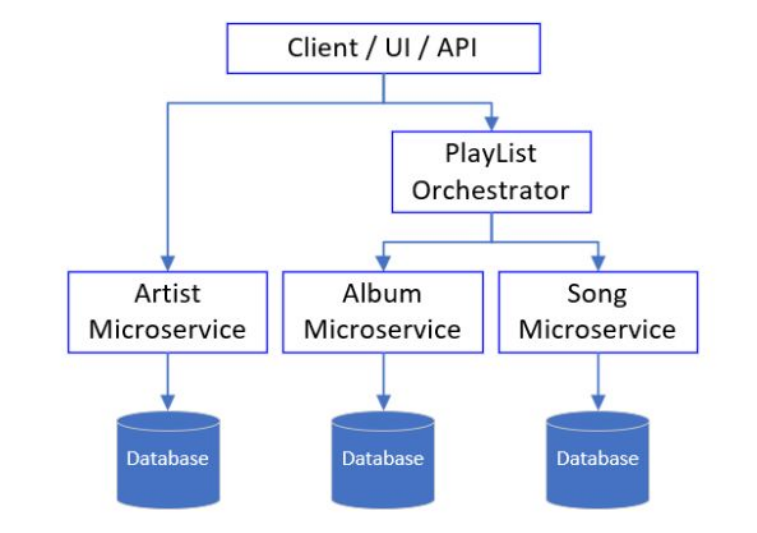


FIG 9. Shared data microservice design pattern example

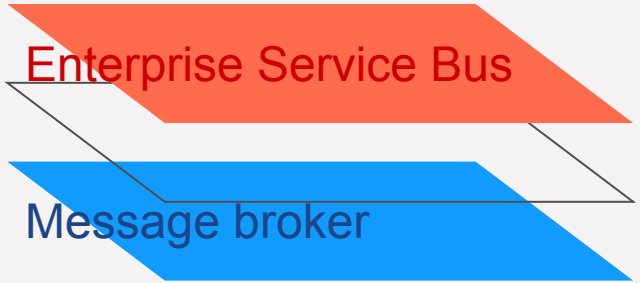
AMQP vs. ESB

Without any logic only streaming and queuing

- RabbitMQ
- Apache Kafka

With business logic such as data transformation & data normalization & smart routing

- Mule ESB
- Apache Camel



Orchestrator?

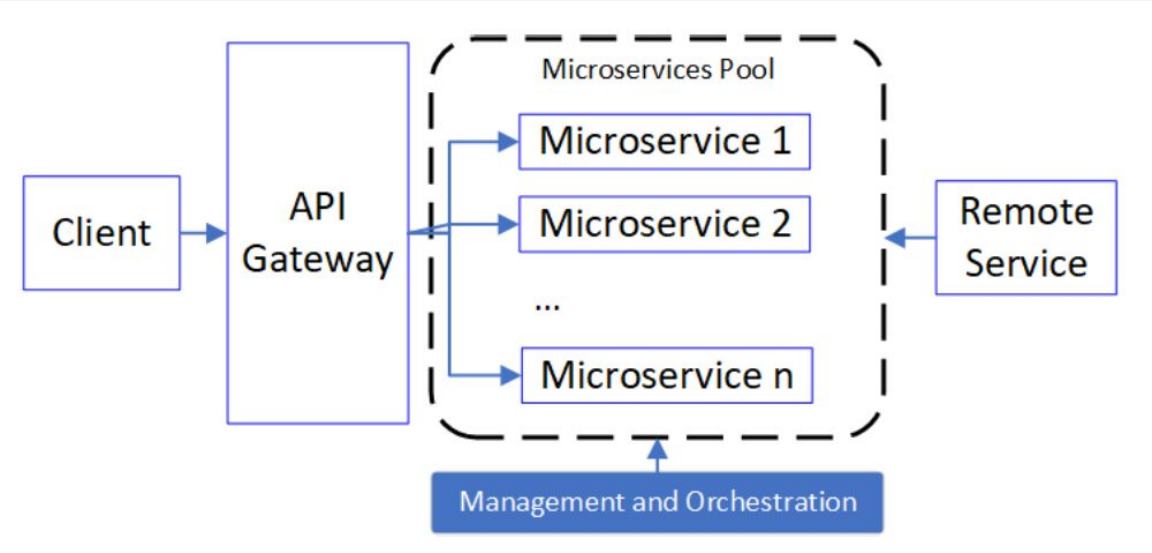


FIG 3. Microservice reference architecture.

Thanks