Microservices Architecture

Software Architecture

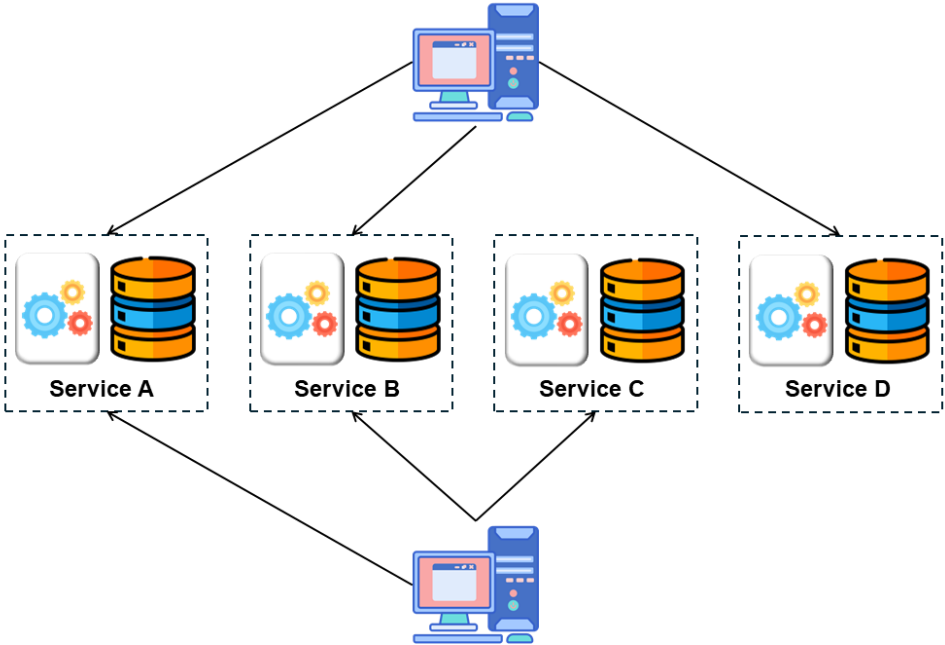
Richard Thomas

April 7, 2025

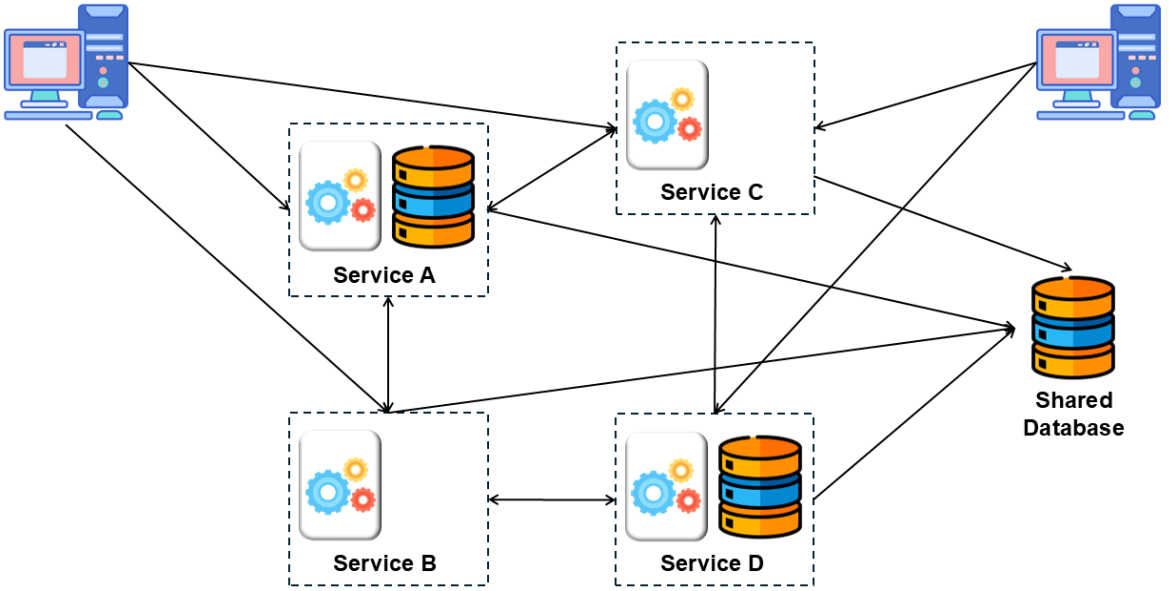
History

- Service Oriented Architecture (SOA)
- Enterprise Service Bus (ESB)
- RESTful APIs
- Microservices

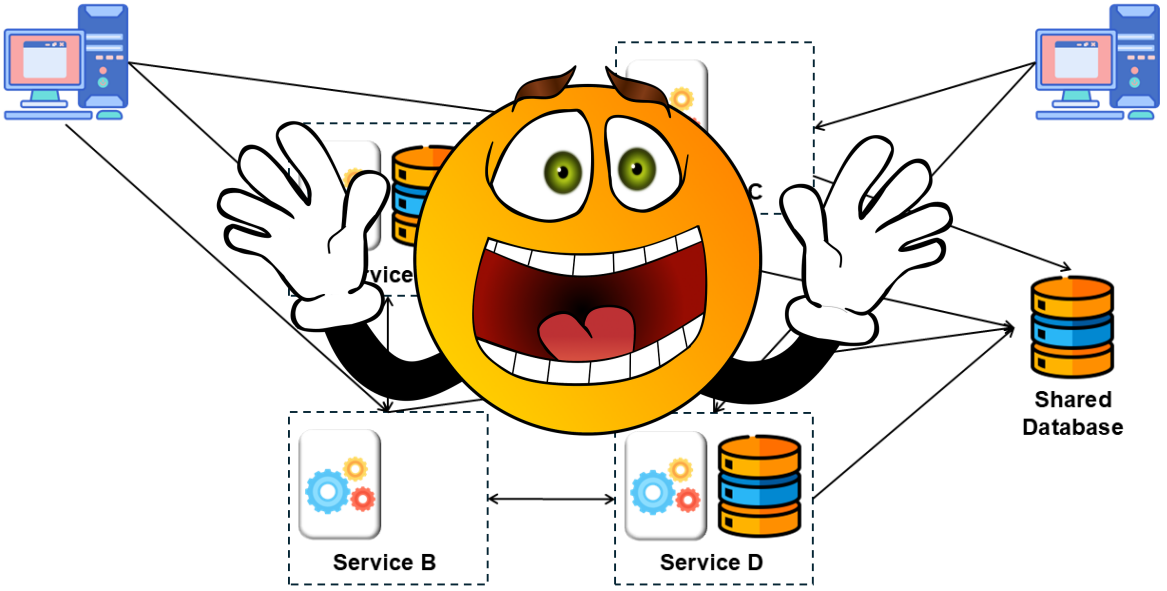
SOA



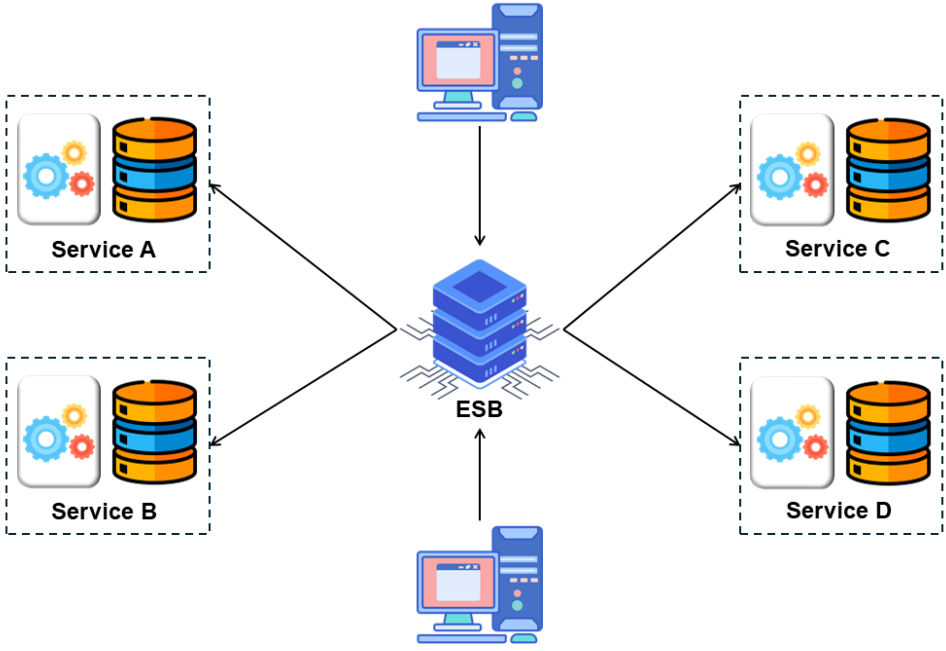
SOA Circumvented



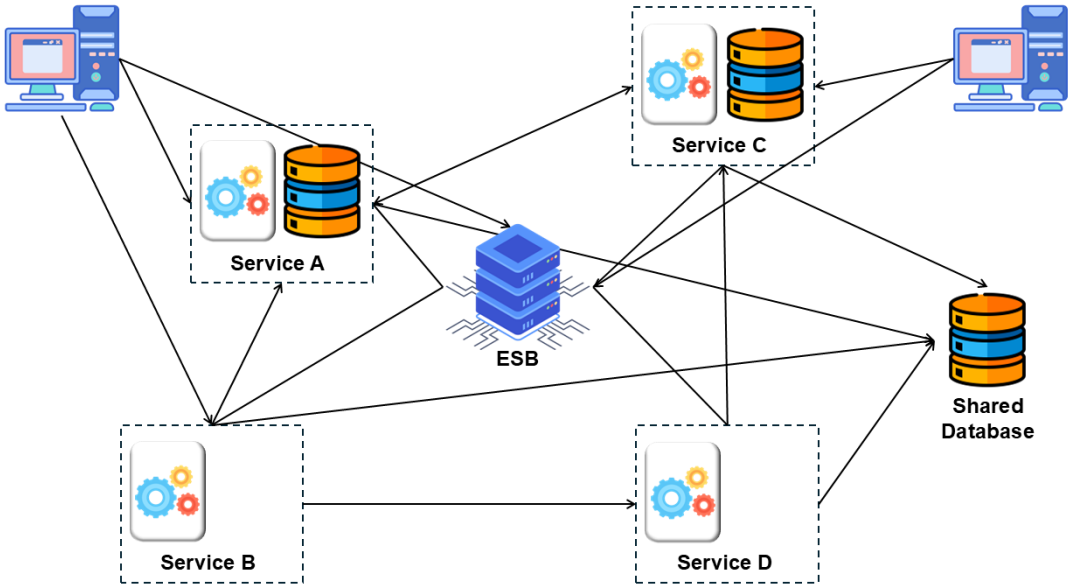
SOA Circumvented



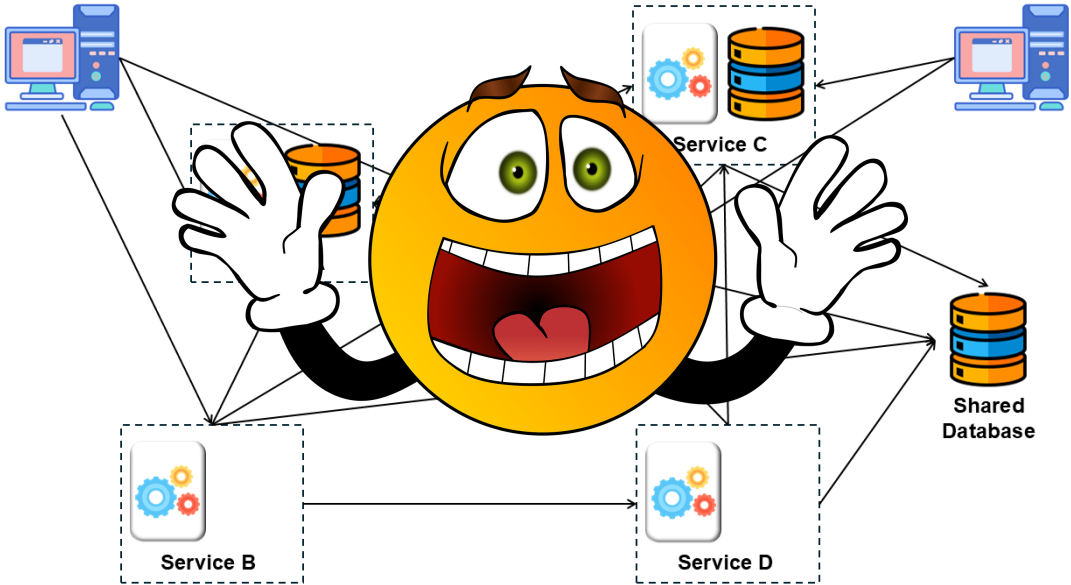
ESB



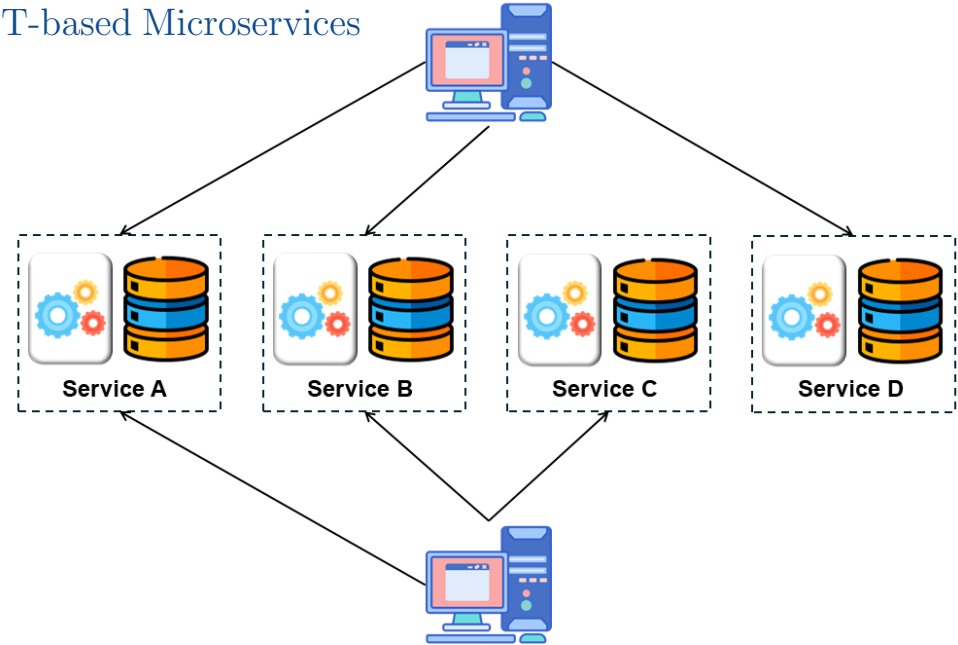
ESB Circumvented



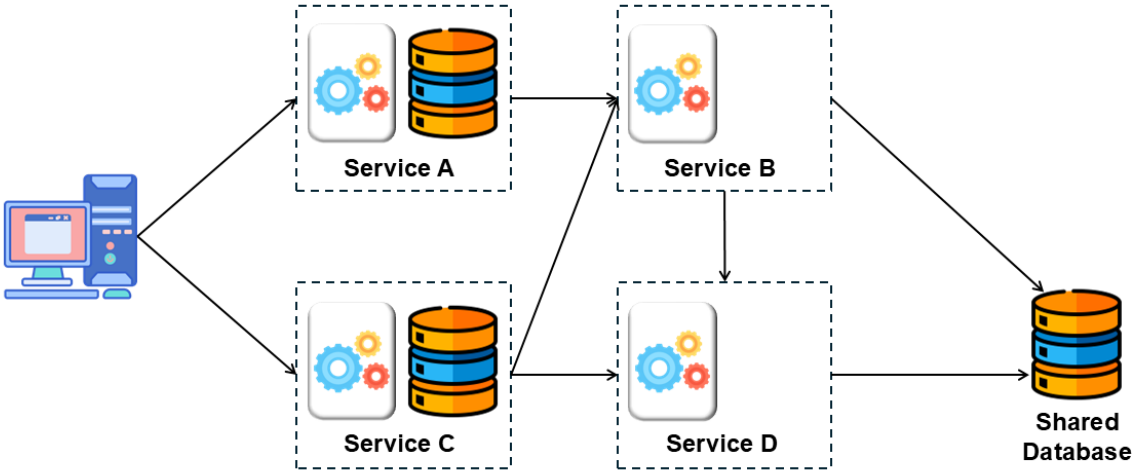
ESB Circumvented



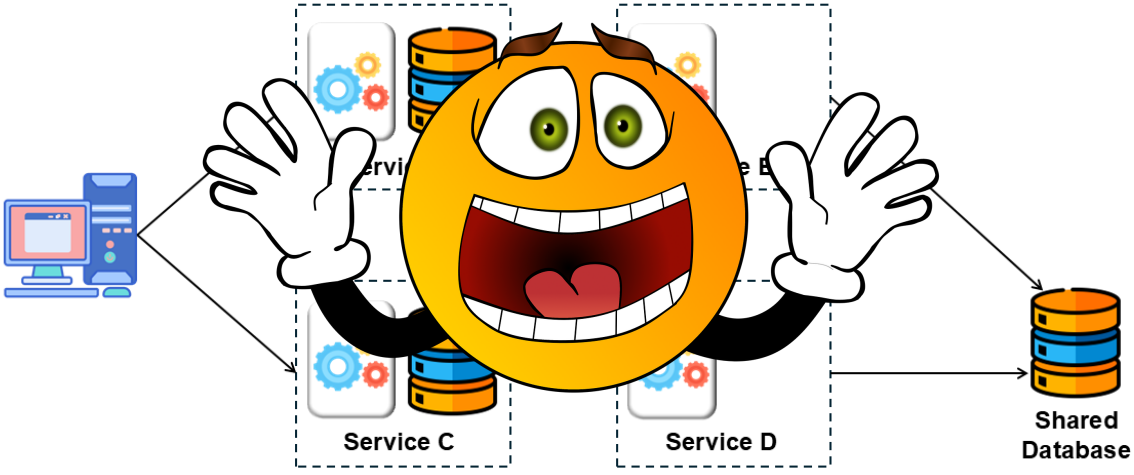
REST-based Microservices



REST-based Microservices Circumvented



REST-based Microservices Circumvented



Question

Why does it go wrong?

Question

Why does it go wrong?

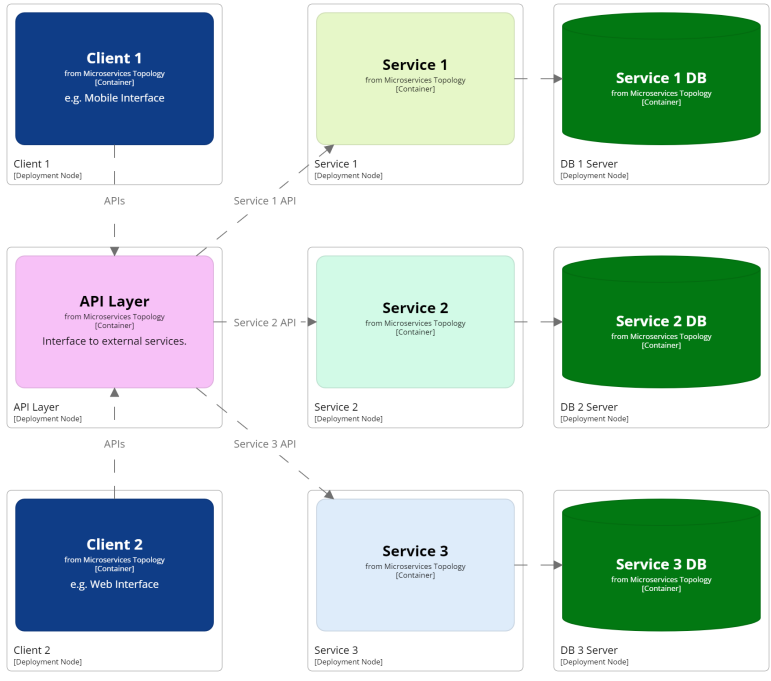
Answer

Getting caught up in the technology!

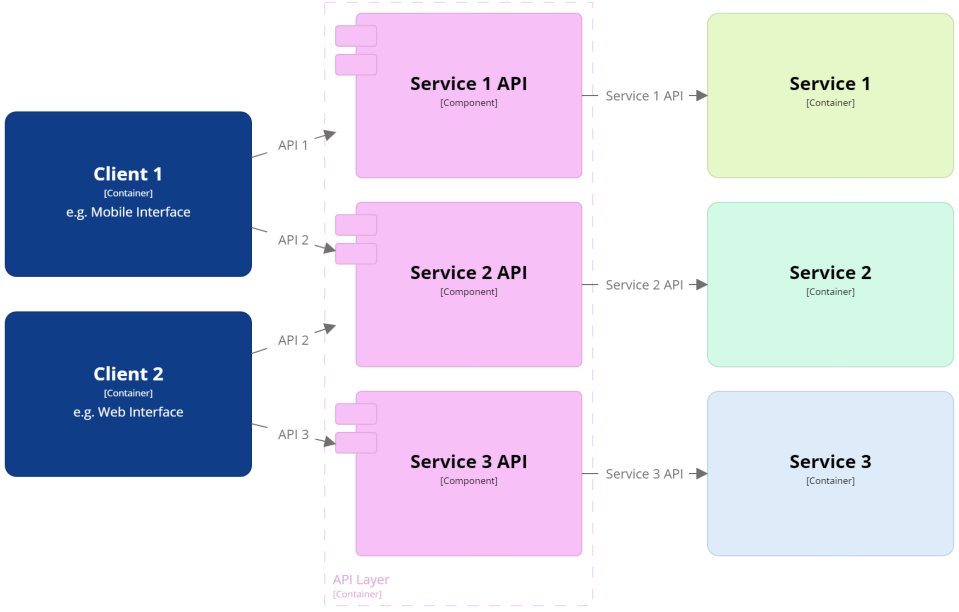
If you haven't shipped *one* service.
How will you ship a *dozen*?

§ *Microservices*

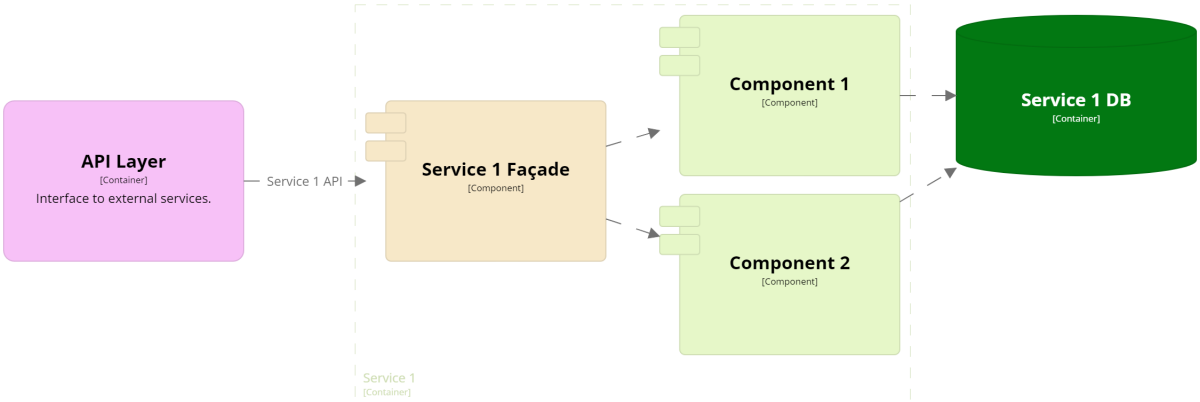
Microservices General Topology



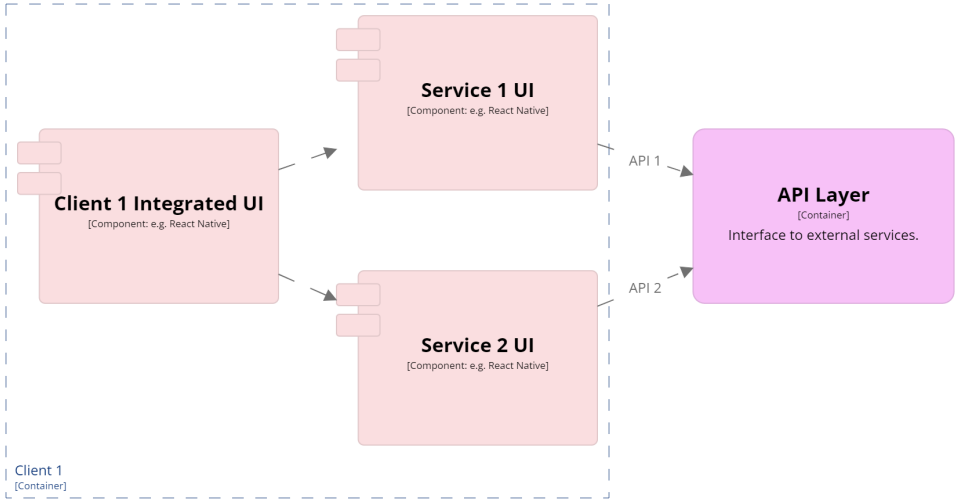
API Layer Components



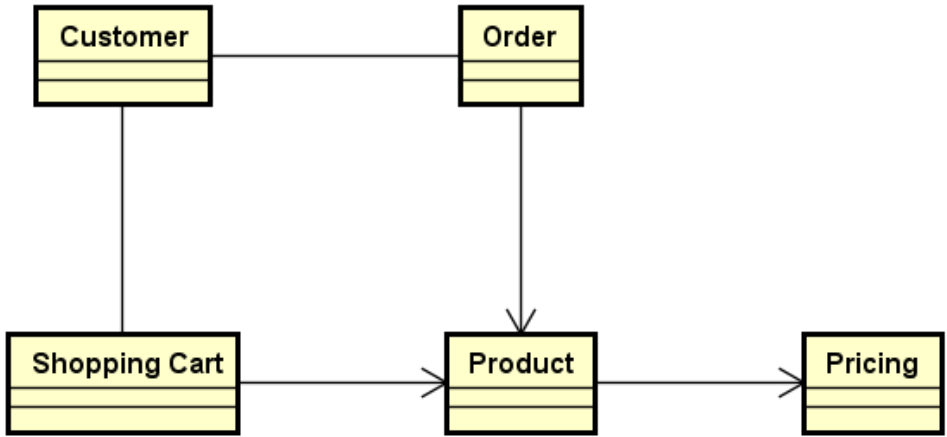
Service 1 Components



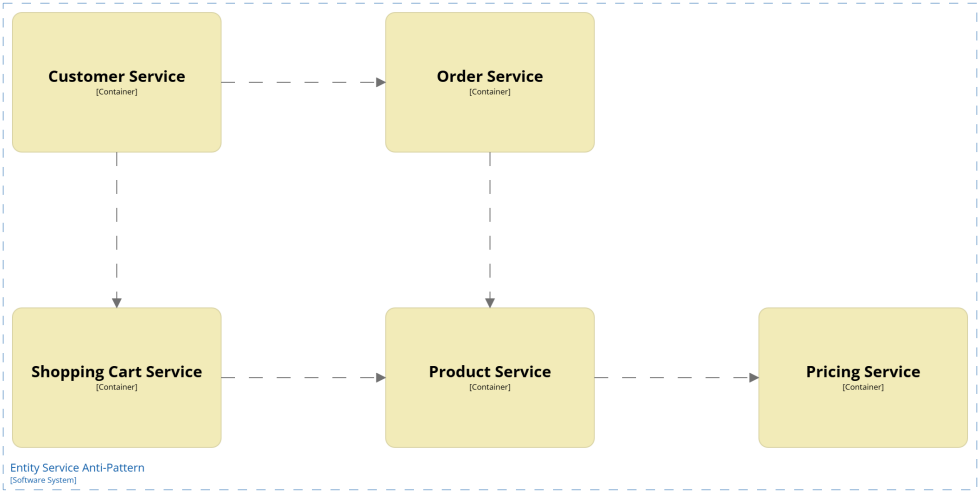
Client with Monolithic UI



Entity Service Anti-Pattern

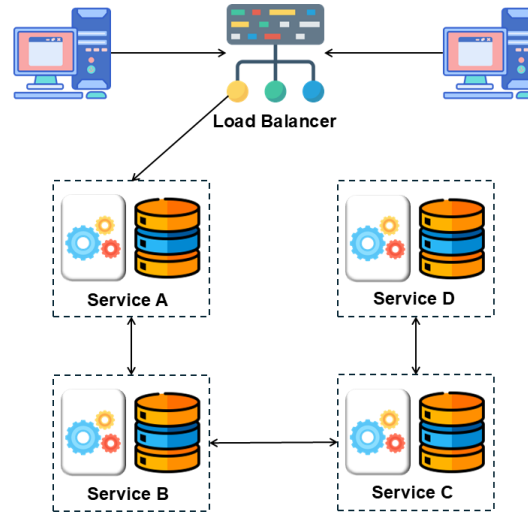


Entity Service Anti-Pattern



Problem with Synchronous Logic

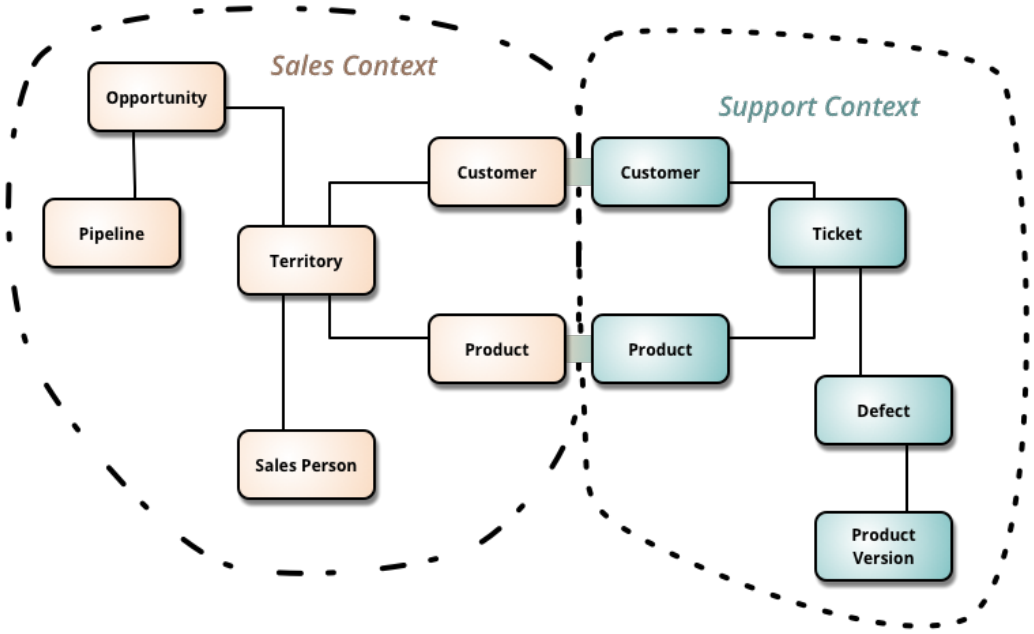
- Potential chain of failure
- Failures are multiplied
- Latency is multiplied
- Deployment is hard
- Scaling is hard
- Lazy design



§ Domain Driven Design

Definition 0. Bounded Context

Logical boundary of a domain where particular terms and rules apply consistently.



From https://martinfowler.com/bliki/BoundedContext.html

Definition 0. Service Cohesion Principle

Services are cohesive business processes.

DDD Consideration

Services are *bounded contexts*.

Bounded contexts are not necessarily *services*.

Large Bounded Contexts

Bounded context may be too large to be a single service.

Split it into services that are *independent* sub-processes.

Definition 0. Service Independence Principle

Services should not depend on the implementation of other services.

Corollary 0. Low Coupling

There should be minimal coupling between services.

Corollary 0. No Reuse

Avoid dependencies between services.

Do not reuse components between services.

Restaurant Examples

Bad Restaurant

Restaurant Examples

Bad Restaurant

Good Restaurant

Lessons Learnt

Chains of request don't *scale*.

People who have the *information* to do their job are more effective.

Service boundaries that follow these *principles* work better.

Human Shaped Microservices

Look at *behaviour* rather than entities.

- Who & How
- Not the Thing

What would people *do*?

How would they *communicate*?

Bounded Domains Implications

- Duplication
  - Entities specialised for domain
    - Requires mapping of entity data between domains

Bounded Domains Implications

- Duplication
  - Entities specialised for domain
    - Requires mapping of entity data between domains
  - Should everything be duplicated?

Bounded Domains Implications

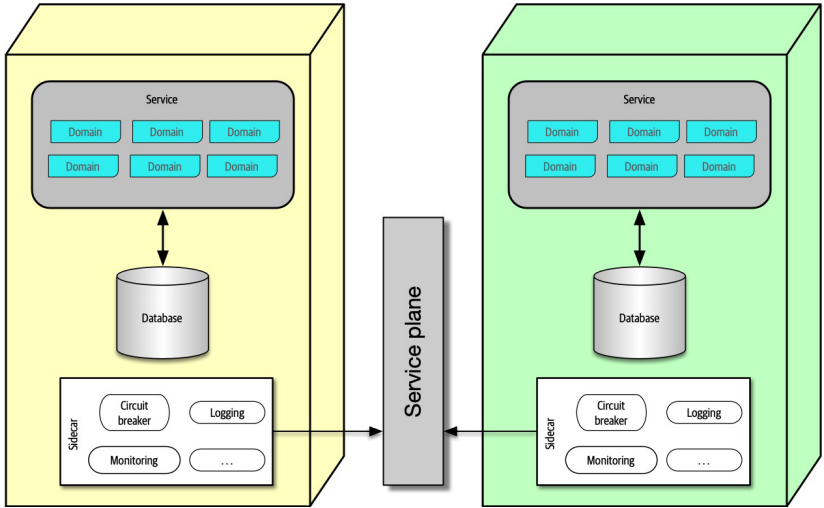
- Duplication
  - Entities specialised for domain
    - Requires mapping of entity data between domains
  - Should everything be duplicated?
    - What about common services (e.g. logging, ...)?

Bounded Domains Implications

- Duplication
  - Entities specialised for domain
    - Requires mapping of entity data between domains
  - Should everything be duplicated?
    - What about common services (e.g. logging, ...)?
- Heterogeneity
  - Services can use different implementation technologies

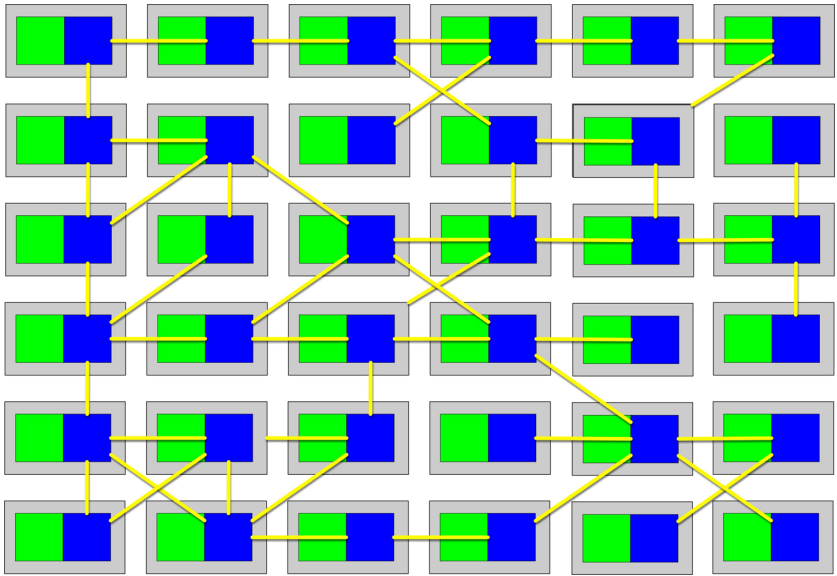
§ Microservices Design Options

Service Plane



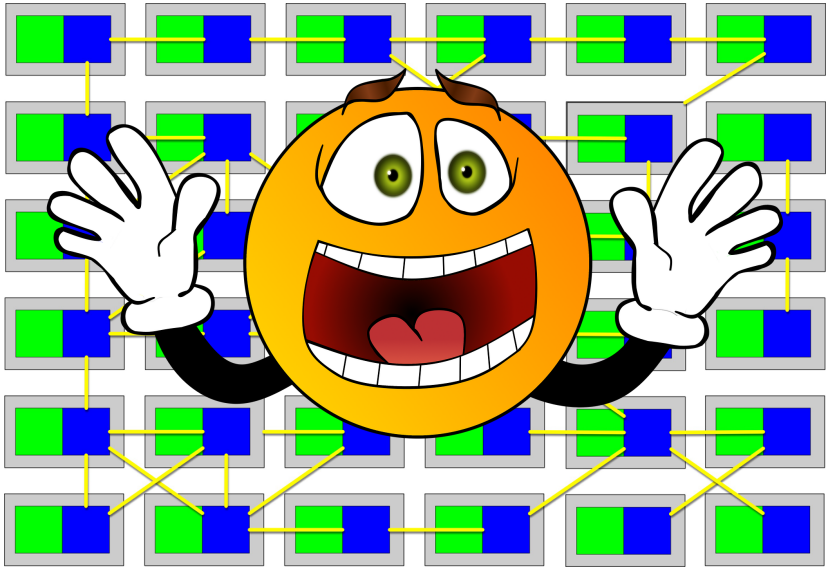
From *Fundamentals of Software Architecture*

Service Mesh



From *Fundamentals of Software Architecture*

Service Mesh



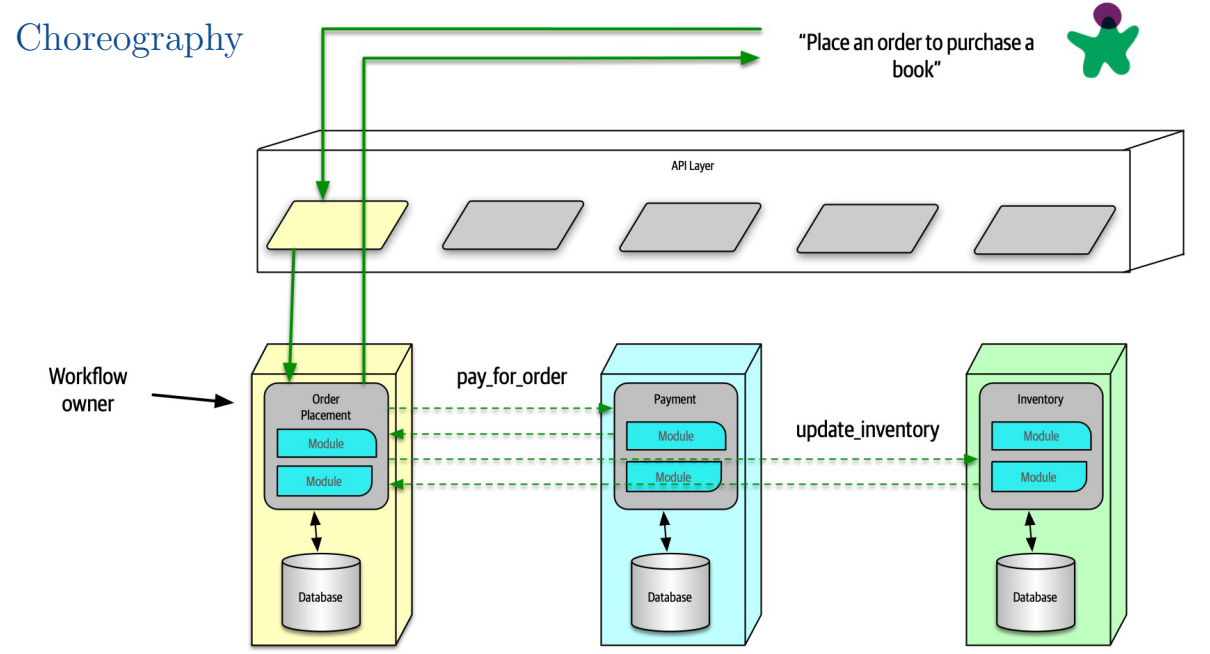
From *Fundamentals of Software Architecture*

Choreography & Orchestration

Choreography Similar to event-driven *broker*

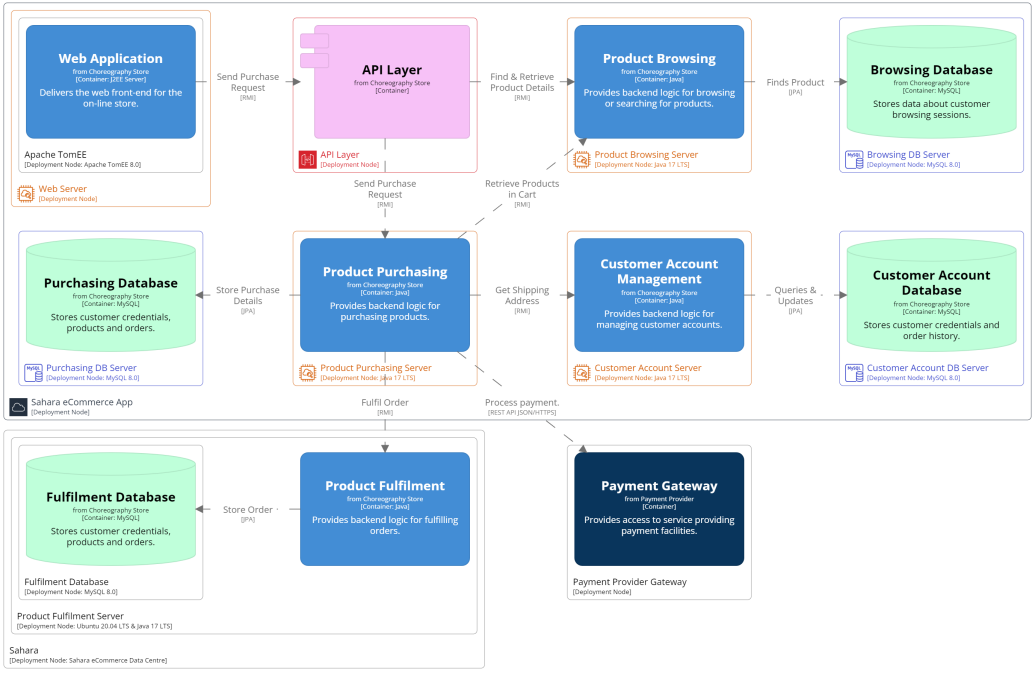
Orchestration Similar to event-driven *mediator*

Choreography

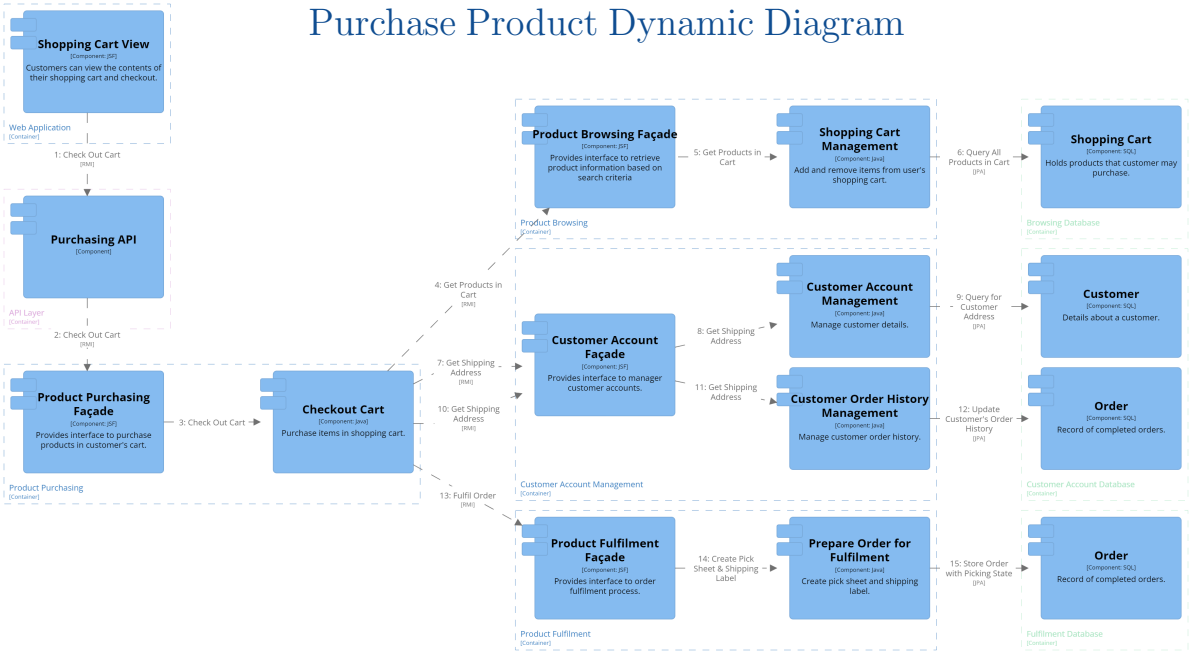


From *Fundamentals of Software Architecture*

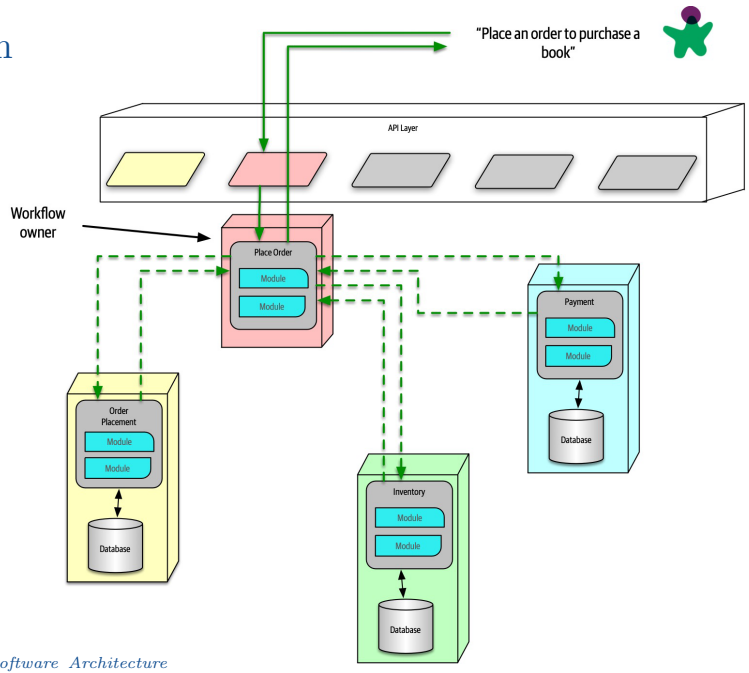
Sahara using Choreography



Purchase Product Dynamic Diagram



Orchestration



From *Fundamentals of Software Architecture*

Question

How bad is coupling with choreography or orchestration?

Question

How bad is coupling with choreography or orchestration?

Answer

For a large system, *very bad*.

Question

How do we scale large microservices-based systems?

Question

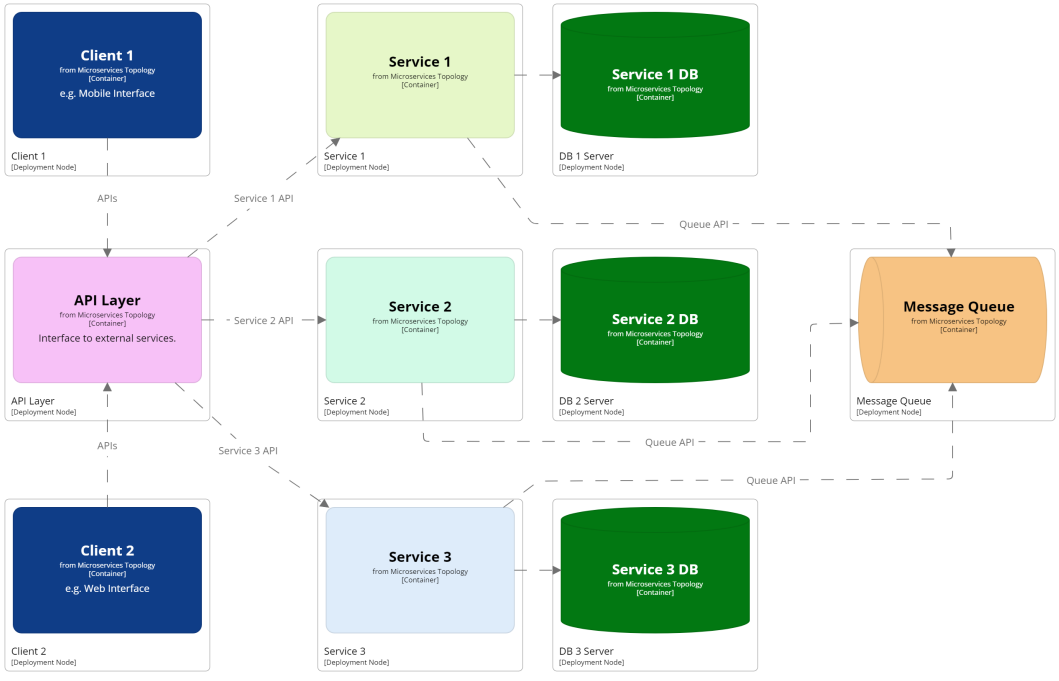
How do we scale large microservices-based systems?

Answer

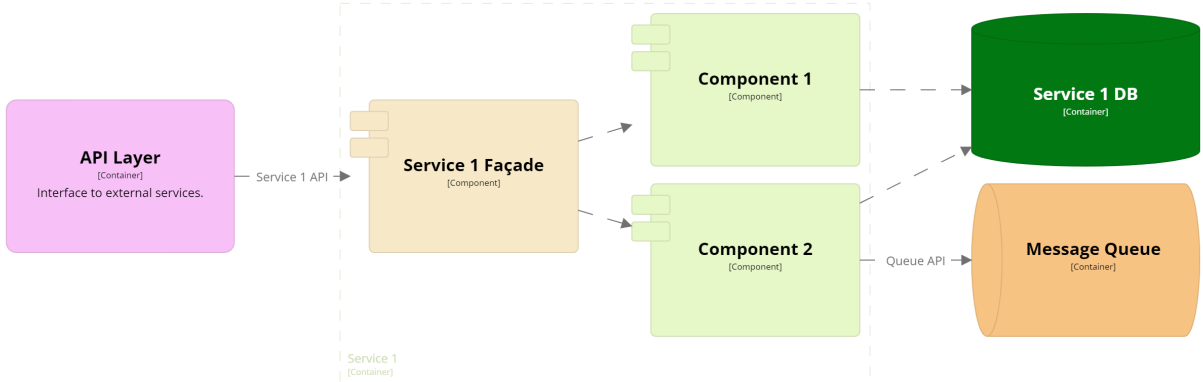
Combine architectural patterns

- Event-Driven Microservices

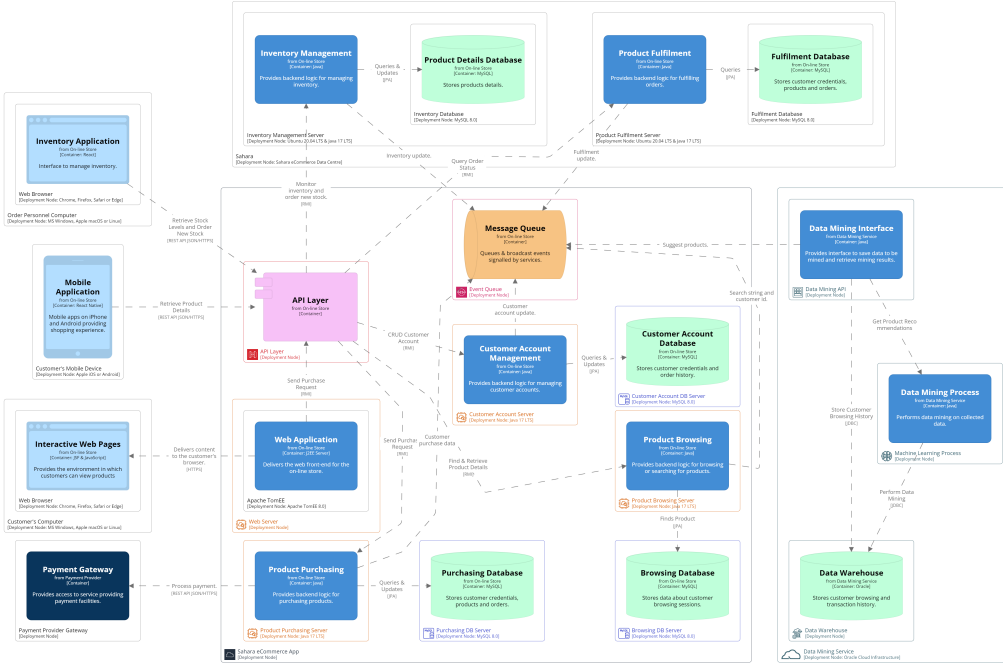
Microservices with Event Queue



Service 1 Components with Event Queue



Sahara using an Event Queue



Question

Are *browsing* and *purchasing* separate contexts?

Question

Are *browsing* and *purchasing* separate contexts?

Answer

- Are they a single business process or different processes?
- Do they share much or little data?

Question

- What about *inventory management* and *browse*?
- How do they maintain a consistent product database?

Model Behaviour

- *Commands* & *Events* describe *behaviour*
- They will help you better model your *domain*
- Leading to *independent*, *scalable* services

Pros & Cons

- Modularity 
- Extensibility 
- Reliability 
- Interoperability 
- Scalability 
- Security 
- Deployability 
- Testability 
- Simplicity