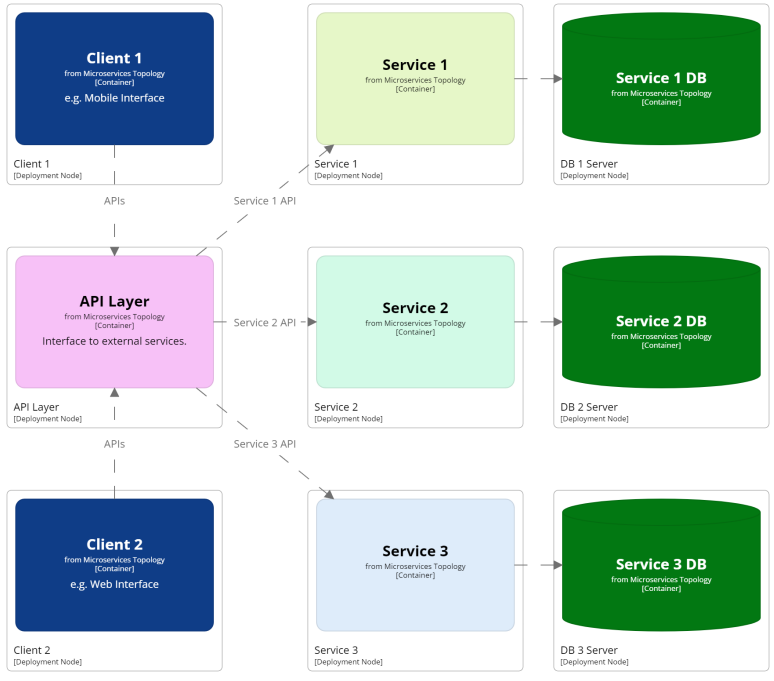
Microservices Architecture

Software Architecture

Richard Thomas

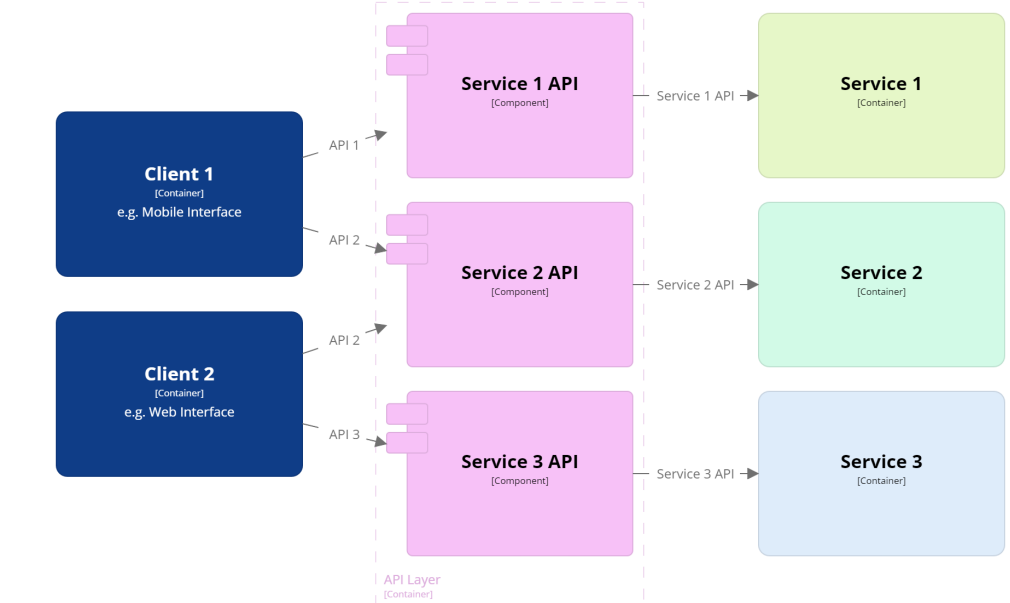
April 14, 2025

Microservices General Topology



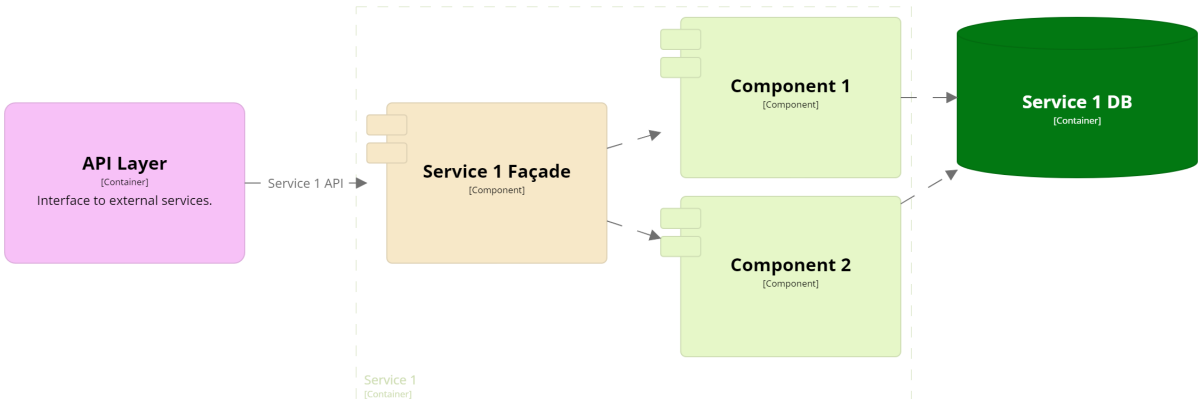
- Multiple clients demonstrates common scenario of multiple interfaces to system (e.g. mobile, web).
- Client UIs may be monolithic to provide a rich interface.

API Layer Components



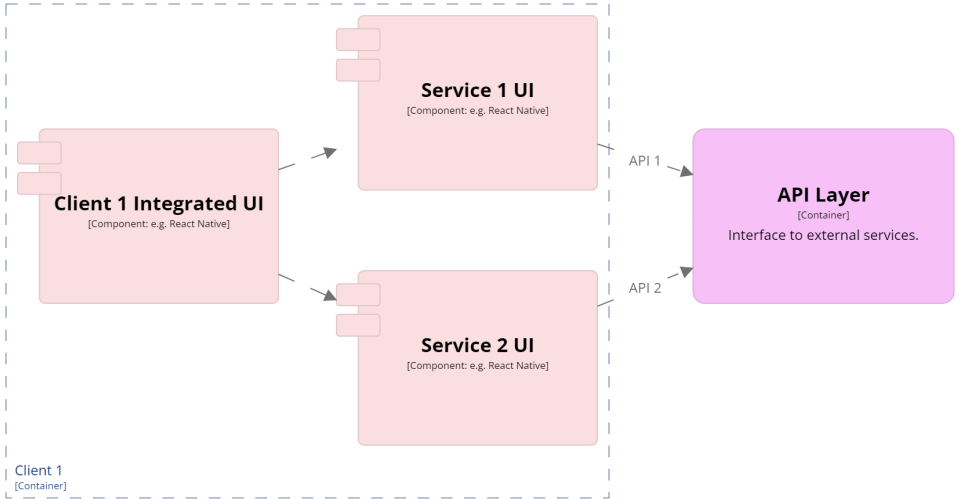
- Each client may use a different combination of services.
- API layer provides reverse proxy or gateway services, see Service-Based Architecture notes & slides.
- Typically Service APIs in this layer have a one-to-one relationship with Services and are designed by the Service teams.
- Routing behaviour may not be required.

Service 1 Components



Services 2 & 3 are essentially the same.

Client with Monolithic UI



- Purist Microservices architecture – each service development team builds their service's UI(s).
- Typically needs some coordinating activity in the UI.
- Can still have multiple UIs (e.g. web, mobile, ...).

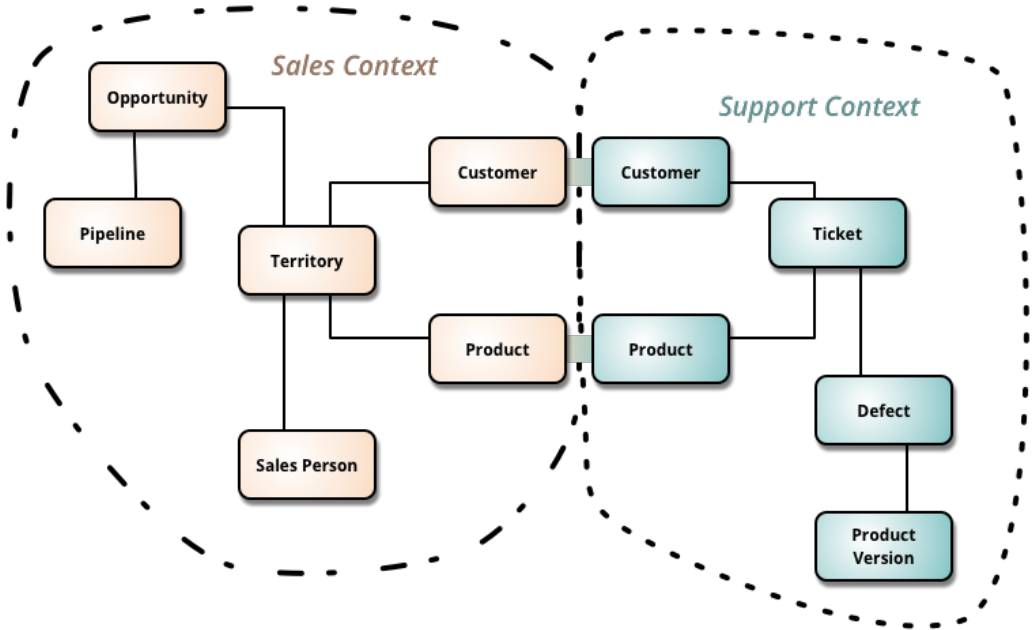
DDD Influence

Services are *bounded contexts*.

Bounded contexts are not necessarily *services*.

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.
They are a bounded context.

A 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.

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?

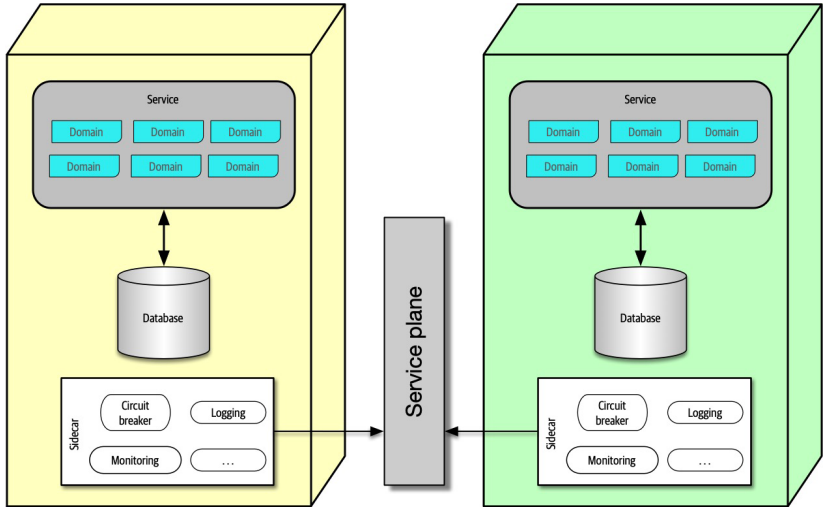
Bounded Domains Implications

- 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

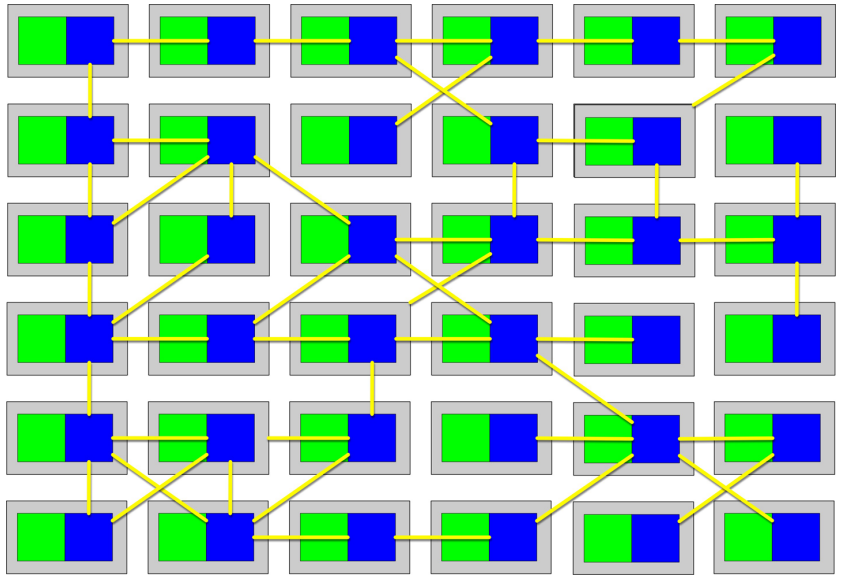
- Duplication
  - Entities specialised for domain
    - Requires mapping of entity data between domains
  - Should everything be duplicated?
    - What about common services (e.g. logging, ...)?
- Heterogeneity
  - Services can use different implementation technologies

Service Plane



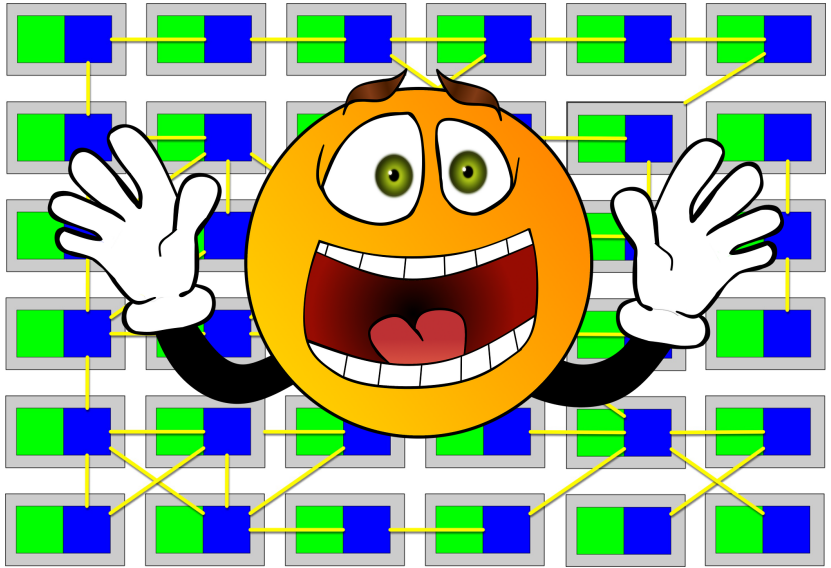
From *Fundamentals of Software Architecture*

Service Mesh



From *Fundamentals of Software Architecture*

Service Mesh

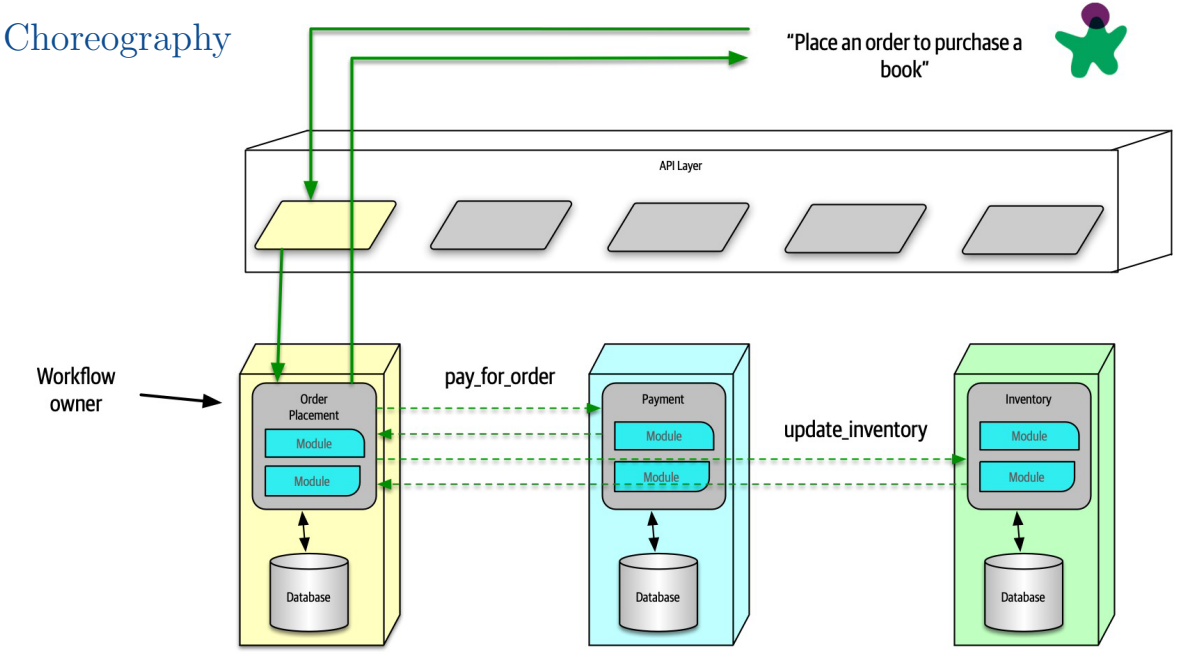


From *Fundamentals of Software Architecture*

Choreography Similar to event-driven *broker*

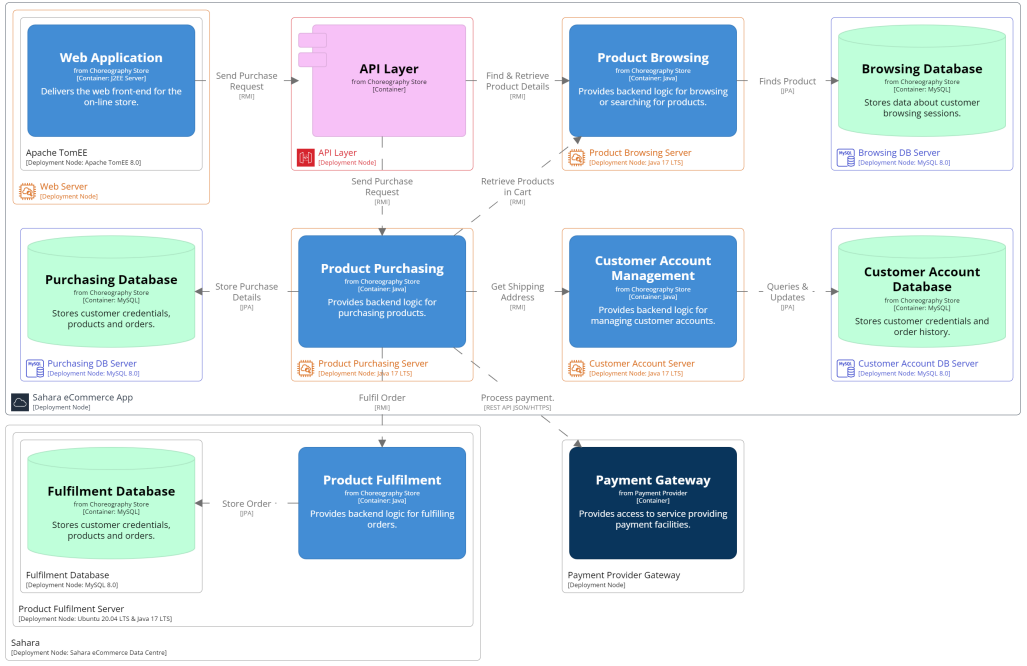
Orchestration Similar to event-driven *mediator*

Choreography

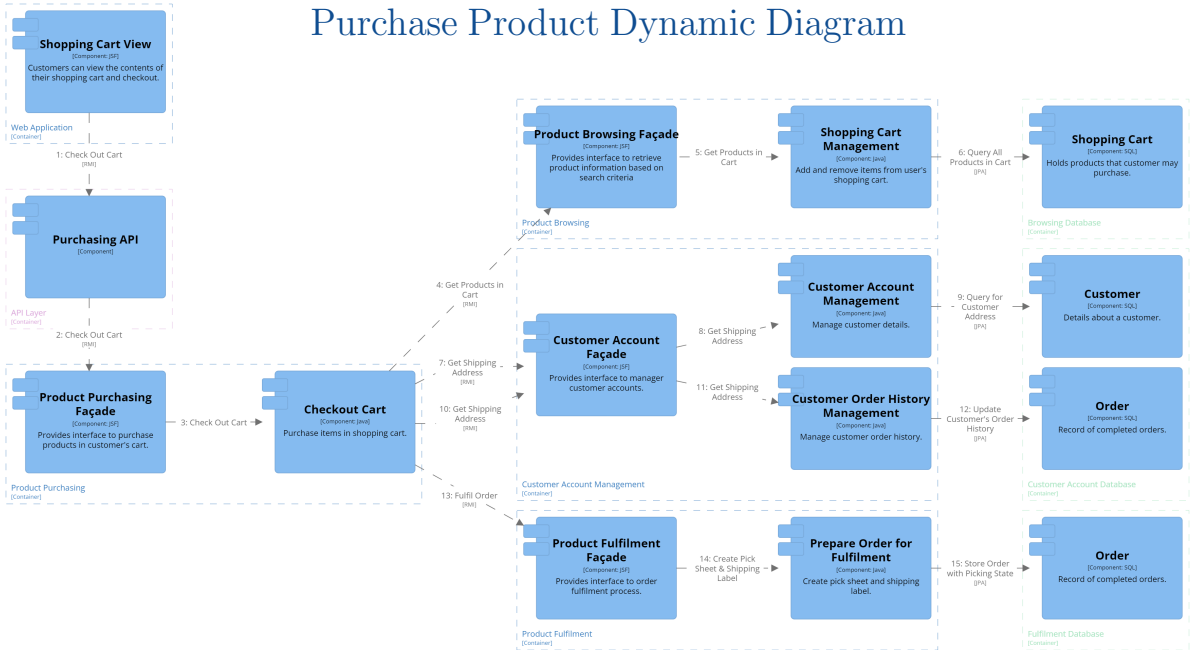


From *Fundamentals of Software Architecture*

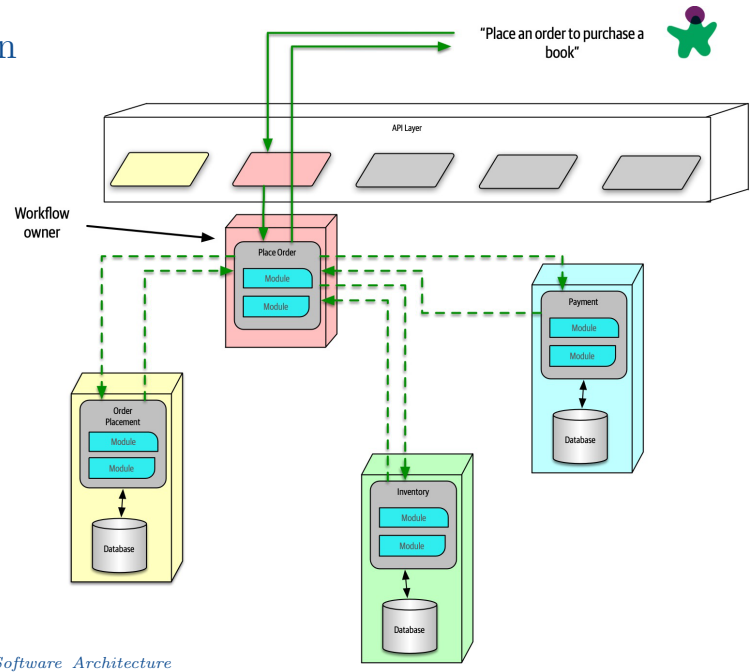
Sahara using Choreography



Purchase Product Dynamic Diagram



Orchestration



From *Fundamentals of Software Architecture*

Question

How bad is the coupling with choreography or orchestration?

Question

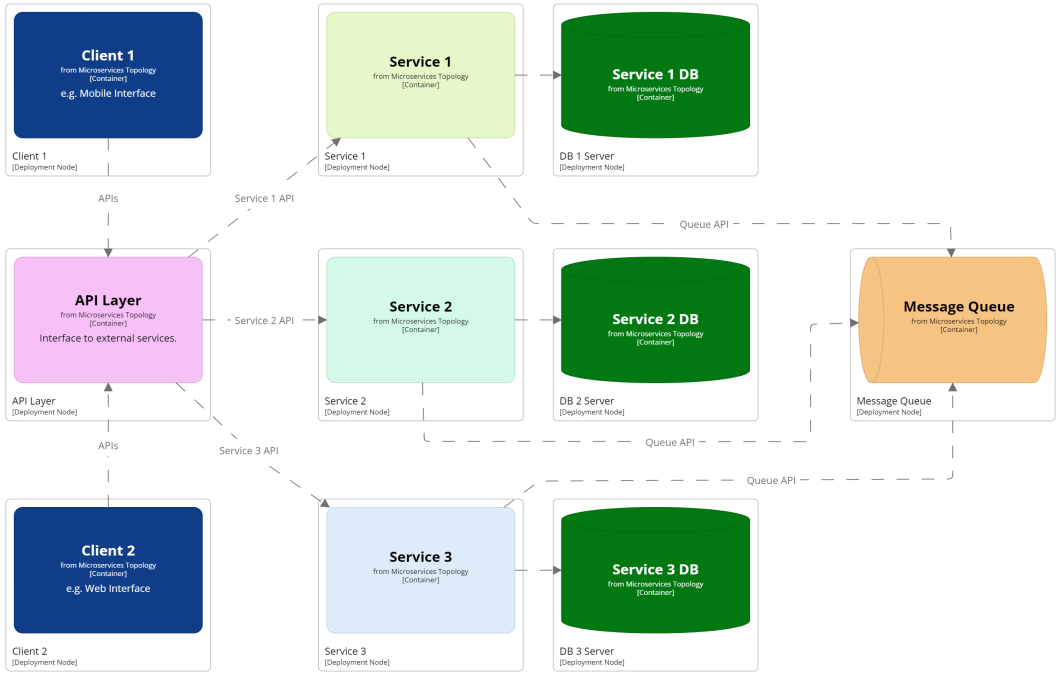
How bad is the coupling with choreography or orchestration?

Answer

For a large system, *very bad*.

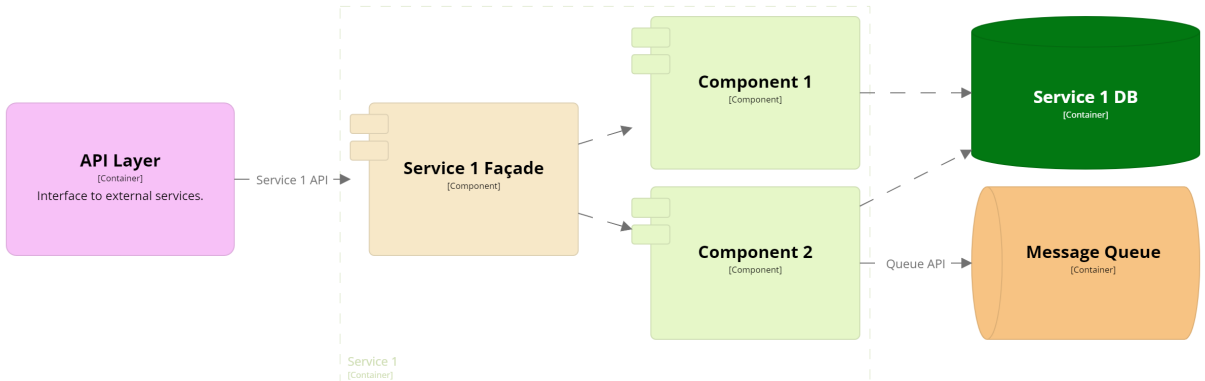
In 2017, Uber had over 1400 services ... consider how bad coupling would be with either approach.

Microservices with Event Queue



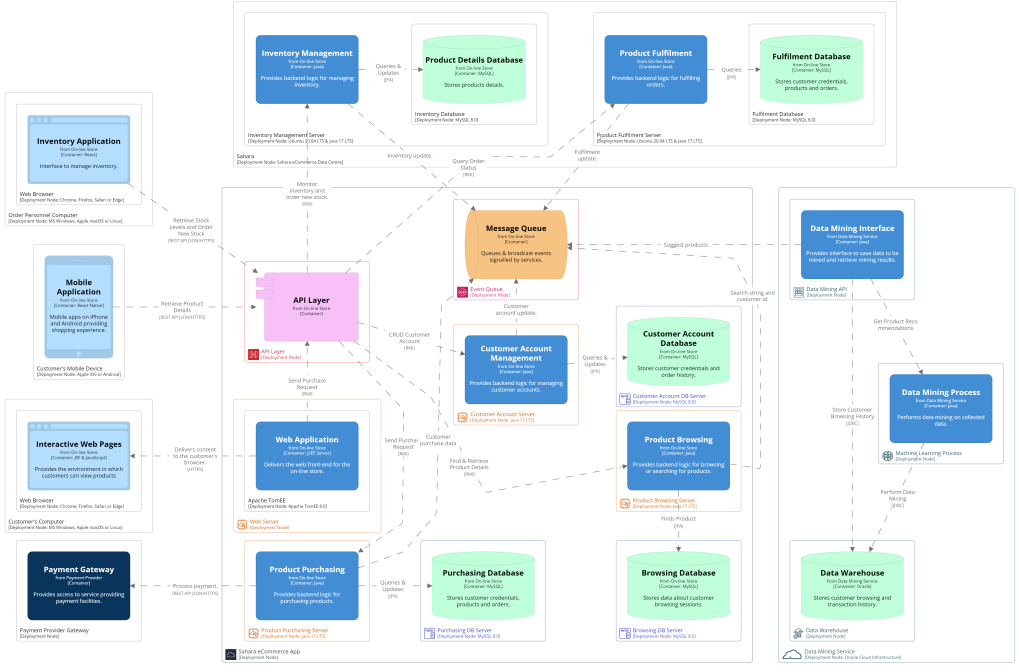
- Use the tried and true Observer pattern, with the event-driven architecture pattern.
- Services publish events indicating what they have been done.
- Services listen for events to decide how to coordinate their part of the system behaviour.

Service 1 Components with Event Queue



Services 2 & 3 are essentially the same.

Sahara using an Event Queue



- Sahara eCommerce system as a simple microservices architecture, using event-driven messaging between services.
- Services publish events indicating what they have been done.
- Also an example of a multi-tenanted system built across in-house servers, AWS and OCI.

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?

- Probably different business processes, but possibly the same context.
- If separate services, browse needs to send an event for every change to the shopping cart, and purchase needs to listen for these.
- Possibly merge into one service, as one context.

Question

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

Pros & Cons

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