Microservices Architecture

CSSE6400

Richard Thomas

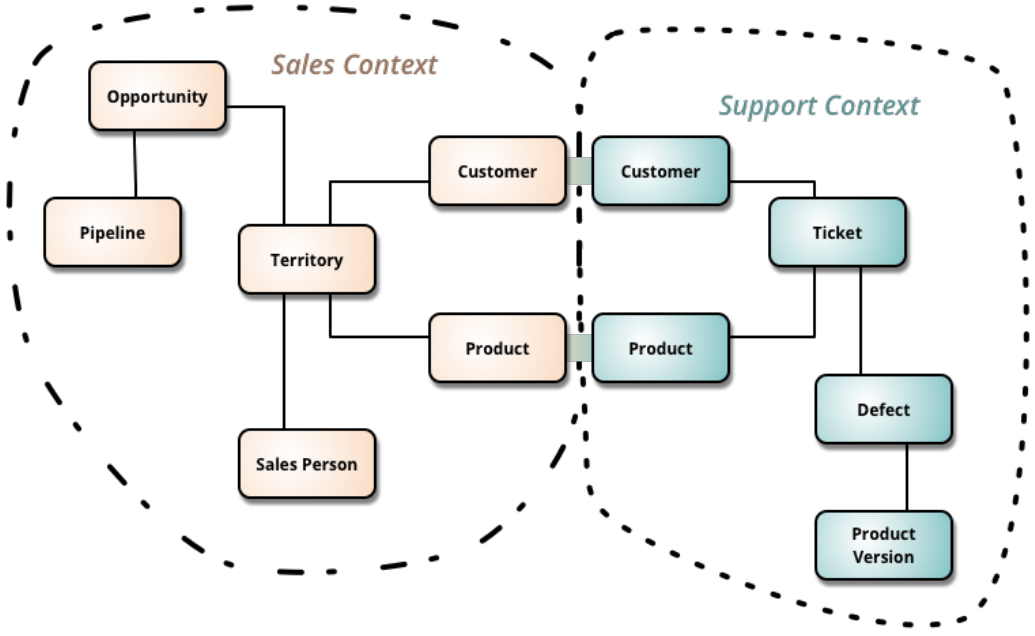
May 9, 2022

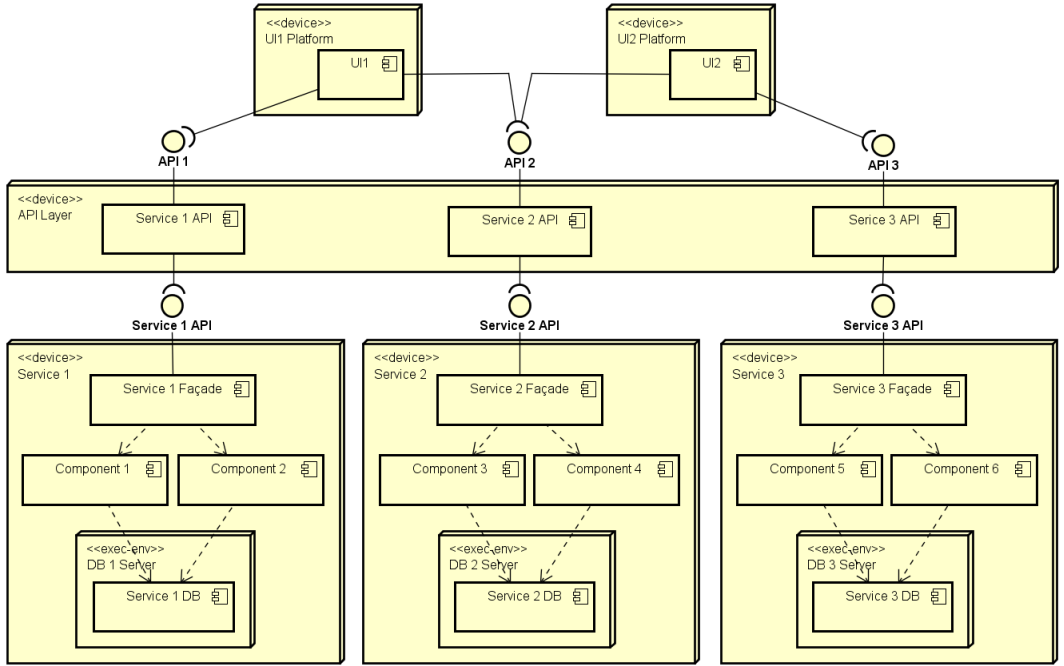
Microservices

Inspired by DDD

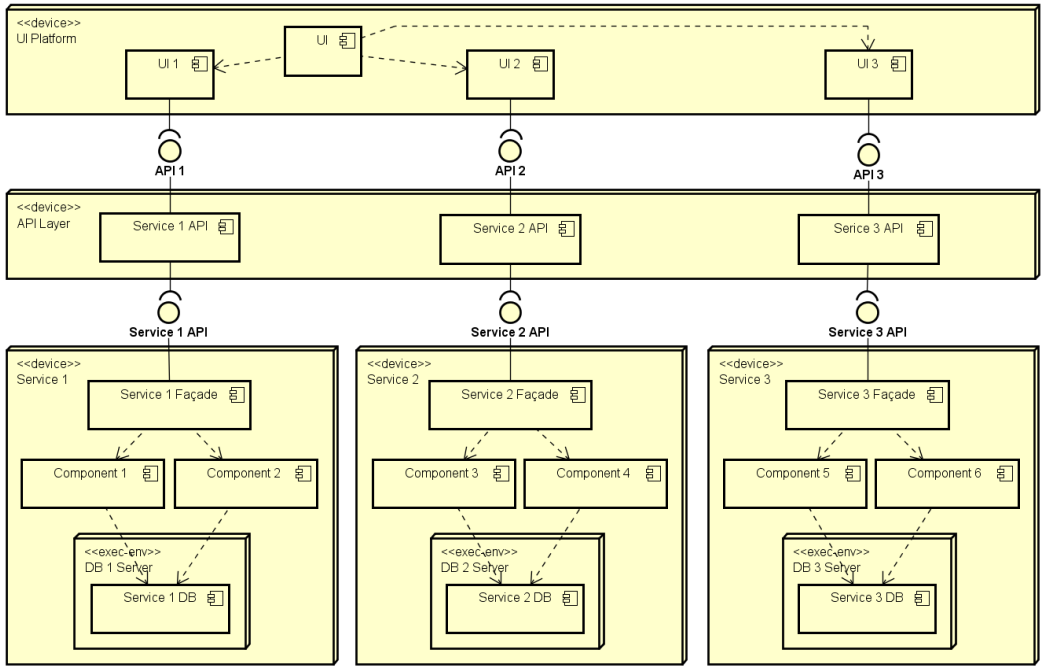
Definition 1. Bounded Context

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


Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Product
Sales Person
Customer
Ticket
Product
Defect
Product Version



- Basic structure of a microservices architecture.
- UIs are fairly monolithic to provide a rich interface.
- Fairly common to have multiple UIs, some of which use a different combination of services.



- More like a purist microservices architecture, where each service development team builds the service's UI(s).
- Typically needs some coordinating activity in the UI.
- Can still have multiple UIs (e.g. web, mobile, ...).

Definition 2. Service Cohesion Principle

Services are cohesive business processes. They are a bounded context.

Definition 3. Service Independence Principle

Services should not depend on the implementation of other services.

Corollary 1. Low Coupling

Services should have minimal coupling with other services.

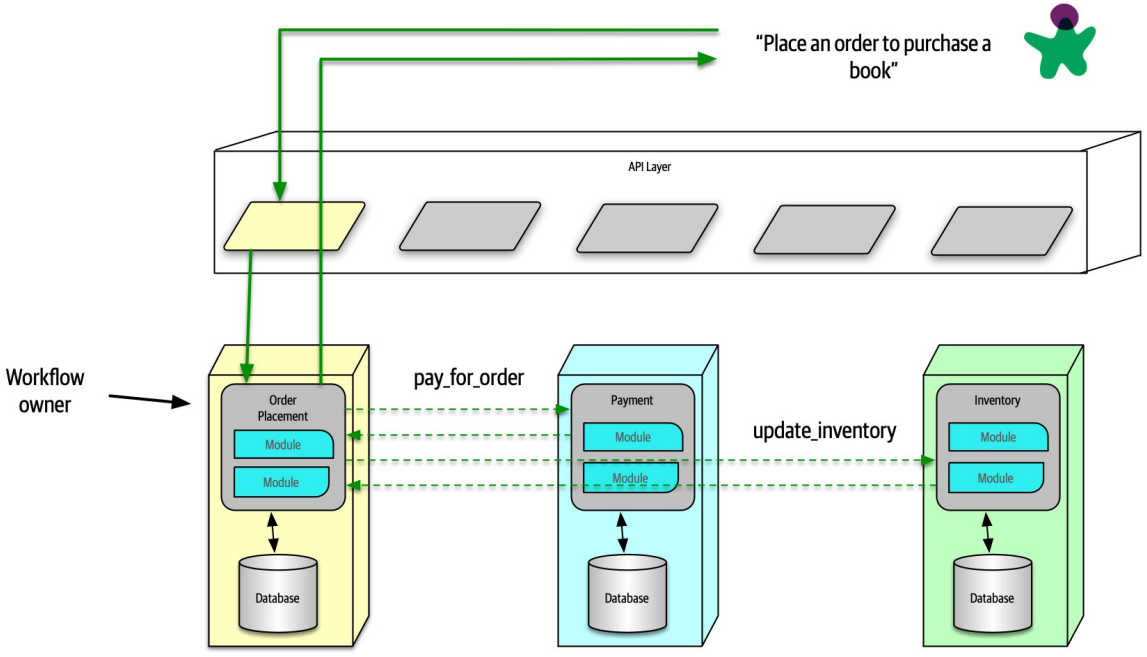
Corollary 2. No Reuse

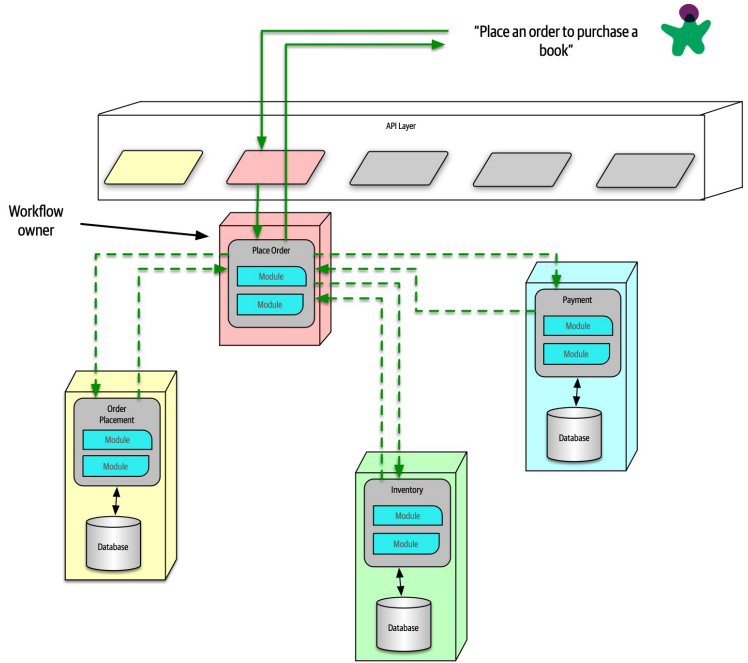
Services do not reuse components from other services, to avoid dependencies.

Choreography & Orchestration

Choreography Similar to event-driven *broker*

Orchestration Similar to event-driven *mediator*


"Place an order to purchase a book"
API Layer
Workflow owner
Order Placement
Module
Module
Database
pay_for_order
Payment
Module
Module
Database
update_inventory
Inventory
Module
Module
Database


"Place an order to purchase a book"
API Layer
Workflow owner
Place Order
Module
Module
Payment
Module
Module
Database
Order Placement
Module
Module
Database
Inventory
Module
Module
Database

Question

How bad is the coupling with choreography or orchestration?

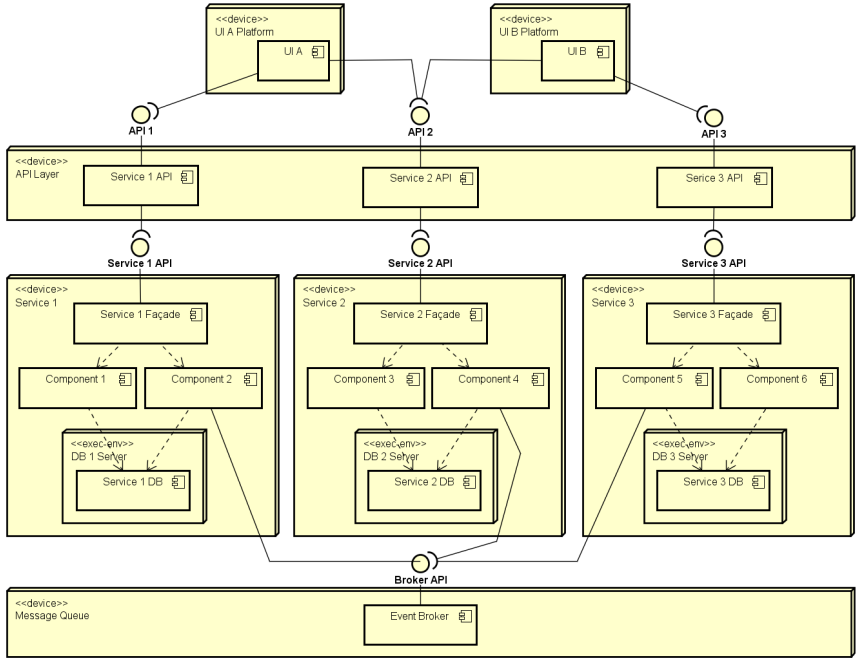
Question

How bad is the coupling with choreography or orchestration?

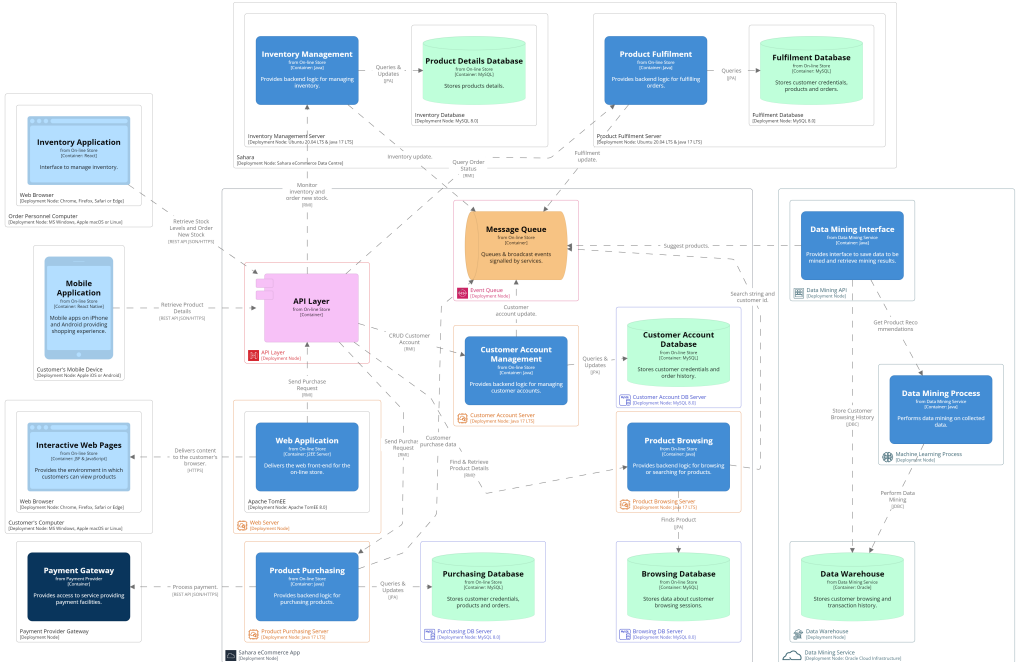
Answer

For a very large system, very bad.

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



- 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 what to coordinate system behaviour.



- 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 the 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