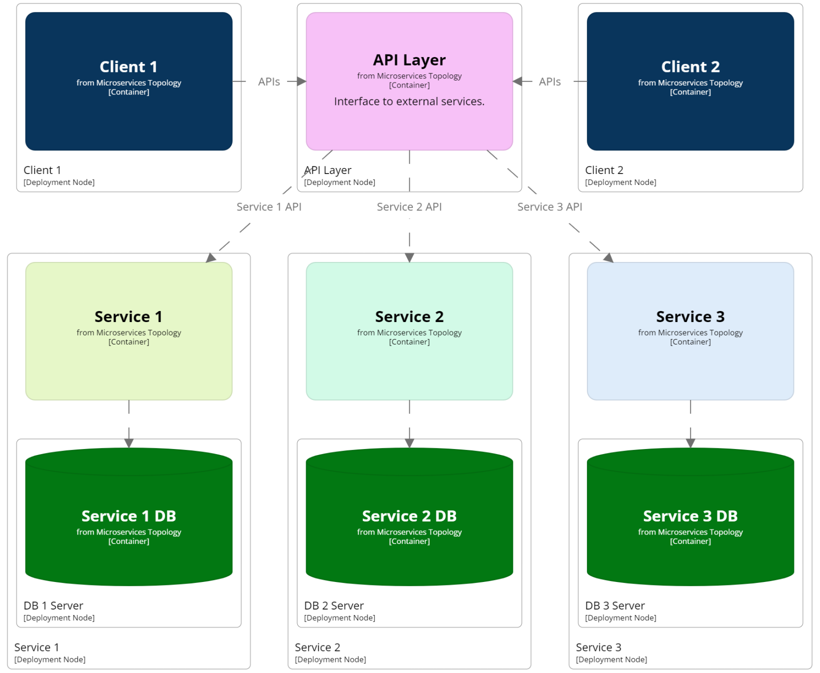
Microservices Architecture

Software Architecture

Richard Thomas

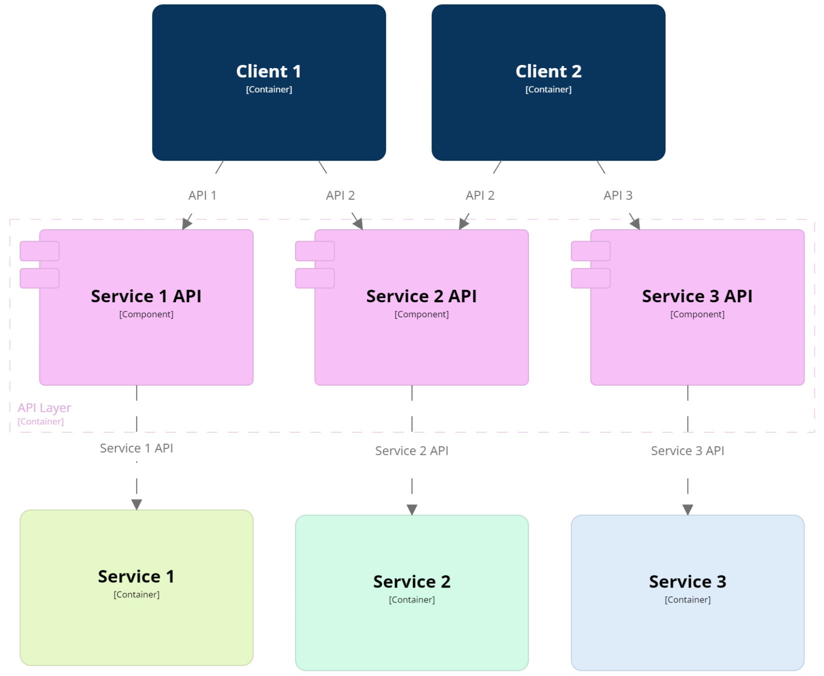
May 8, 2023

Microservices General Topology



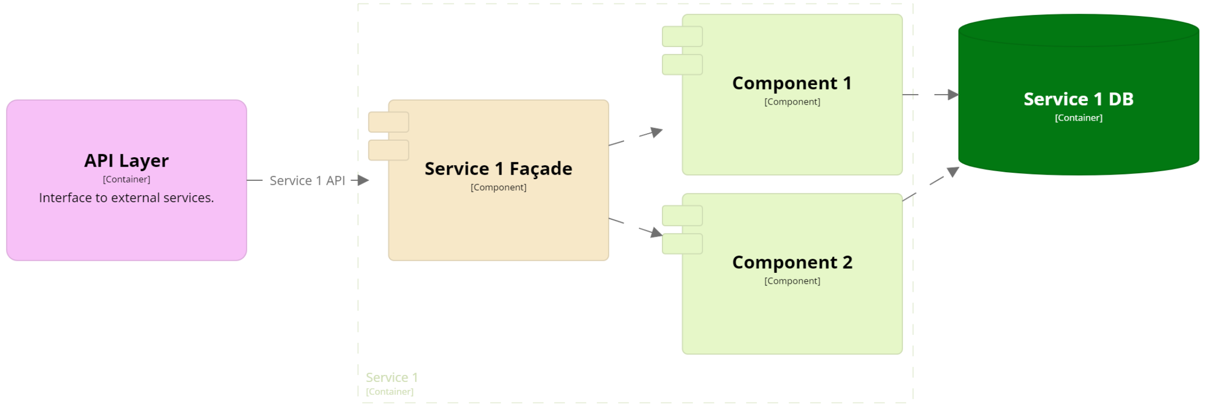
- Multiple clients demonstrates common scenario of multiple interfaces to system (e.g. mobile, web).
- Client UIs may be monolithic to provide a rich interface.

API Layer Components

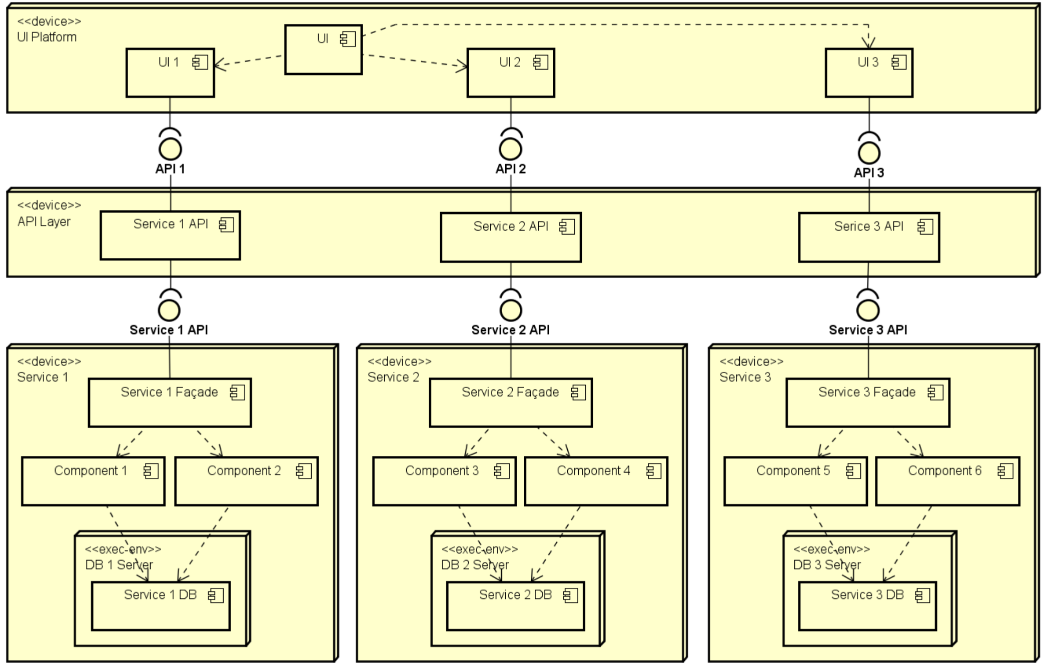


- Each client may use a different combination of services.
- API layer provides reverse proxy or gateway services, see Service-Based Architecture notes & slides.
- Typically Service APIs in this layer have a one-to-one relationship with Services and are designed by the Service teams.
- Routing behaviour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

Service 1 Components



Services 2 & 3 are essentially the same.



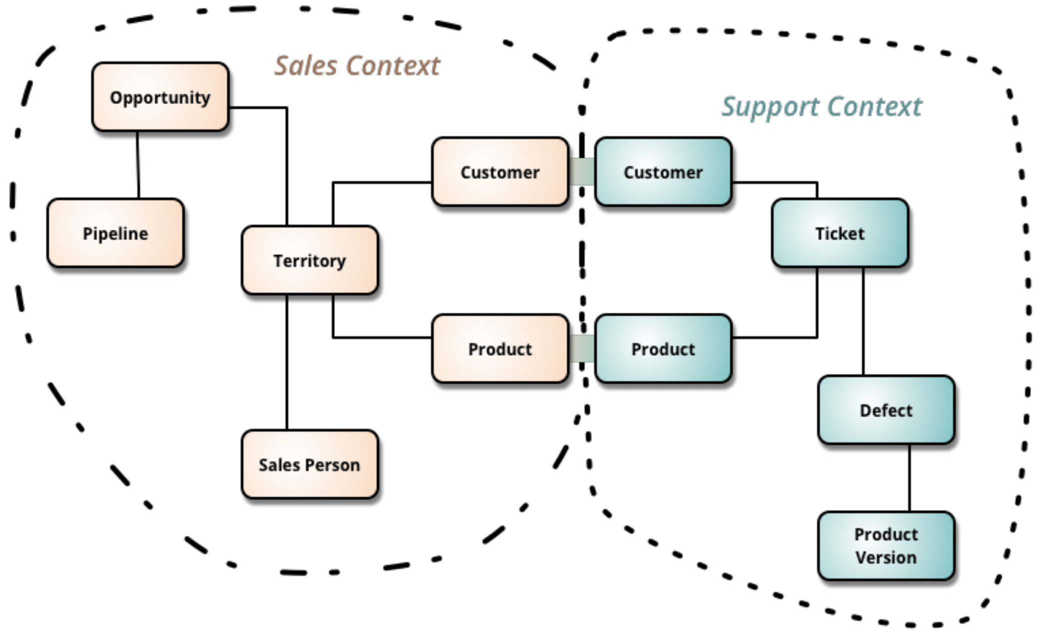
- More like a purist microservices architecture, where each service development team builds the service's UI(s).
- Typically needs some coordinating activity in the UI.
- Can still have multiple UIs (e.g. web, mobile, ...).

DDD Influence

Services are *bounded contexts*.
Bounded contexts are not necessarily *services*.

Definition 1. Bounded Context

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



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

Definition 2. Service Cohesion Principle

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

Large Bounded Contexts

A bounded context may be too large to be a single service.

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

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.

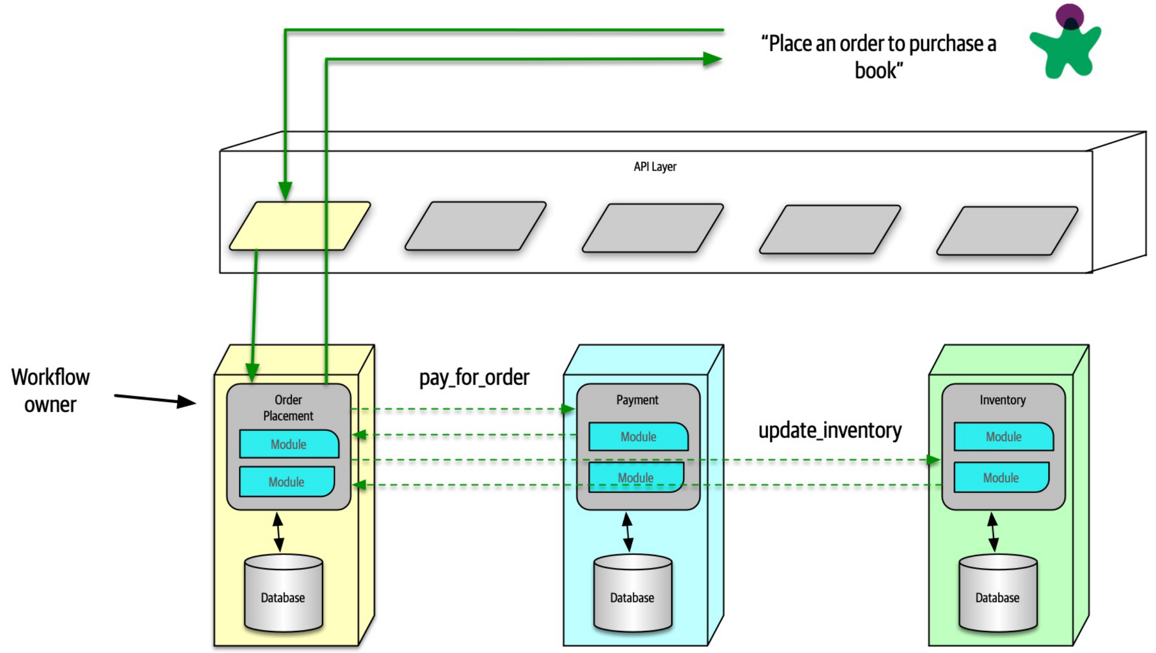
Corollary 2. No Reuse

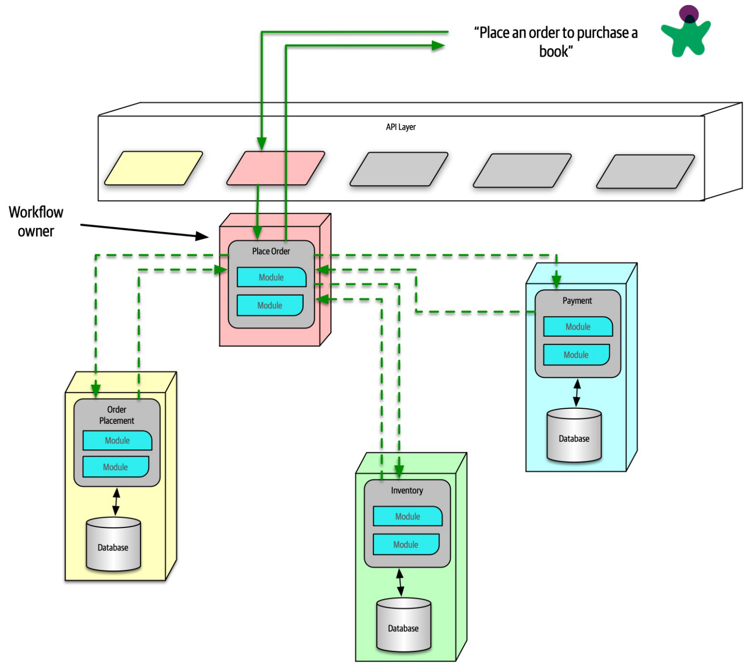
Avoid dependencies between services.
Do not reuse components between services.

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?

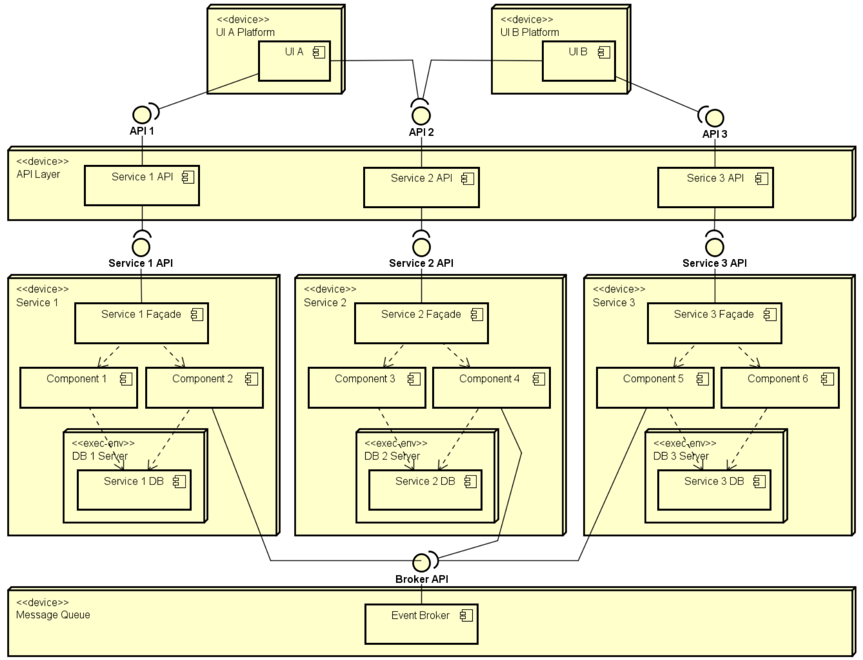
Question

How bad is the coupling with choreography or orchestration?

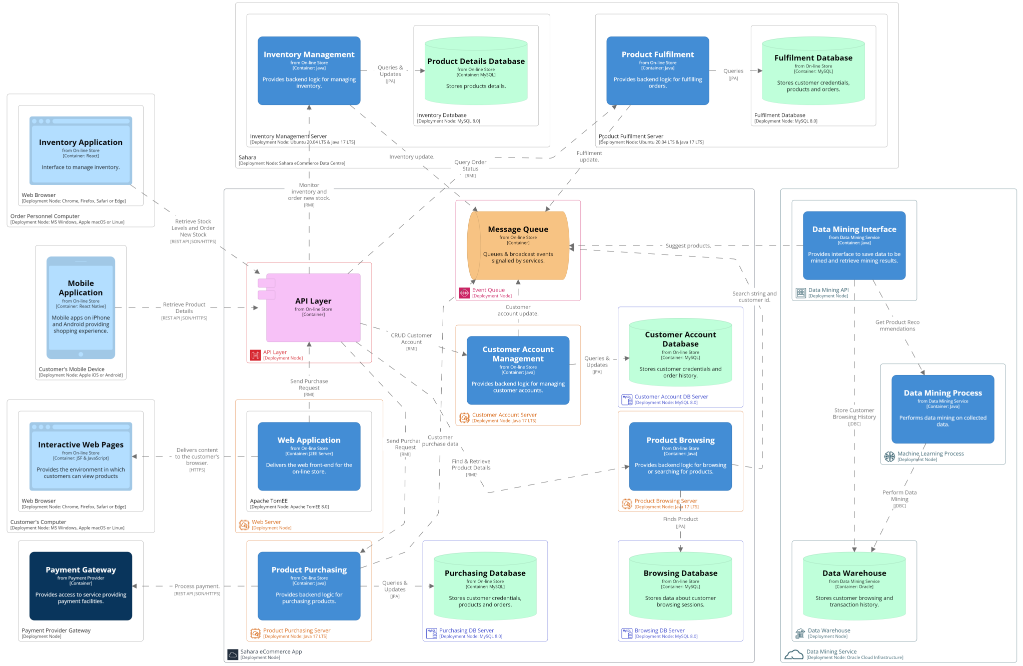
Answer

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.


Inventory Management
from On-line Store
[Container: Java]
Provides backend logic for managing inventory.
Queries & Updates
[JPA]
Product Details Database
from On-line Store
[Container: MySQL]
Stores products details.
Inventory Database
[Deployment Node: MySQL 8.0]
Inventory Management Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Sahara
[Deployment Node: Sahara eCommerce Data Centre]
Product Fulfilment
from On-line Store
[Container: Java]
Provides backend logic for fulfilling orders.
Queries
[JPA]
Fulfilment Database
from On-line Store
[Container: MySQL]
Stores customer credentials, products and orders.
Fulfilment Database
[Deployment Node: MySQL 8.0]
Product Fulfilment Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Inventory update.
Query Order Status
[RMI]
Fulfilment update.
Inventory Application
from On-line Store
[Container: React]
Interface to manage inventory.
Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]
Order Personnel Computer
[Deployment Node: MS Windows, Apple macOS or Linux]
Retrieve Stock Levels and Order New Stock
[REST API JSON/HTTPS]
Monitor inventory and order new stock.
[RMI]
Message Queue
from On-line Store
[Container]
Queues & broadcast events signalled by services.
Event Queue
[Deployment Node]
Suggest products.
Data Mining Interface
from Data Mining Service
[Container: Java]
Provides interface to save data to be mined and retrieve mining results.
Data Mining API
[Deployment Node]
Search string and customer id.
Get Product Reco mmendations
Mobile Application
from On-line Store
[Container: React Native]
Mobile apps on iPhone and Android providing shopping experience.
Customer's Mobile Device
[Deployment Node: Apple iOS or Android]
Retrieve Product Details
[REST API JSON/HTTPS]
API Layer
from On-line Store
[Container]
API Layer
[Deployment Node]
CRUD Customer Account
[RMI]
Customer account update.
Customer Account Management
from On-line Store
[Container: Java]
Provides backend logic for managing customer accounts.
Customer Account Server
[Deployment Node: Java 17 LTS]
Queries & Updates
[JPA]
Customer Account Database
from On-line Store
[Container: MySQL]
Stores customer credentials and order history.
Customer Account DB Server
[Deployment Node: MySQL 8.0]
Send Purchase Request
[RMI]
Interactive Web Pages
from On-line Store
[Container: JSF & JavaScript]
Provides the environment in which customers can view products
Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]
Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]
Delivers content to the customer's browser.
[HTTPS]
Web Application
from On-line Store
[Container: J2EE Server]
Delivers the web front-end for the on-line store.
Apache TomEE
[Deployment Node: Apache TomEE 8.0]
Web Server
[Deployment Node]
Send Purchas Request
[RMI]
Customer purchase data
Find & Retrieve Product Details
[RMI]
Product Browsing
from On-line Store
[Container: Java]
Provides backend logic for browsing or searching for products.
Product Browsing Server
[Deployment Node: Java 17 LTS]
Finds Product
[JPA]
Store Customer Browsing History
[JDBC]
Data Mining Process
from Data Mining Service
[Container: Java]
Performs data mining on collected data.
Machine Learning Process
[Deployment Node]
Perform Data Mining
[JDBC]
Payment Gateway
from Payment Provider
[Container]
Provides access to service providing payment facilities.
Payment Provider Gateway
[Deployment Node]
Process payment.
[REST API JSON/HTTPS]
Product Purchasing
from On-line Store
[Container: Java]
Provides backend logic for purchasing products.
Product Purchasing Server
[Deployment Node: Java 17 LTS]
Queries & Updates
[JPA]
Purchasing Database
from On-line Store
[Container: MySQL]
Stores customer credentials, products and orders.
Purchasing DB Server
[Deployment Node: MySQL 8.0]
Browsing Database
from On-line Store
[Container: MySQL]
Stores data about customer browsing sessions.
Browsing DB Server
[Deployment Node: MySQL 8.0]
Data Warehouse
from Data Mining Service
[Container: Oracle]
Stores customer browsing and transaction history.
Data Warehouse
[Deployment Node]
Sahara eCommerce App
[Deployment Node]
Data Mining Service
[Deployment Node: Oracle Cloud Infrastructure]

- 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	👎