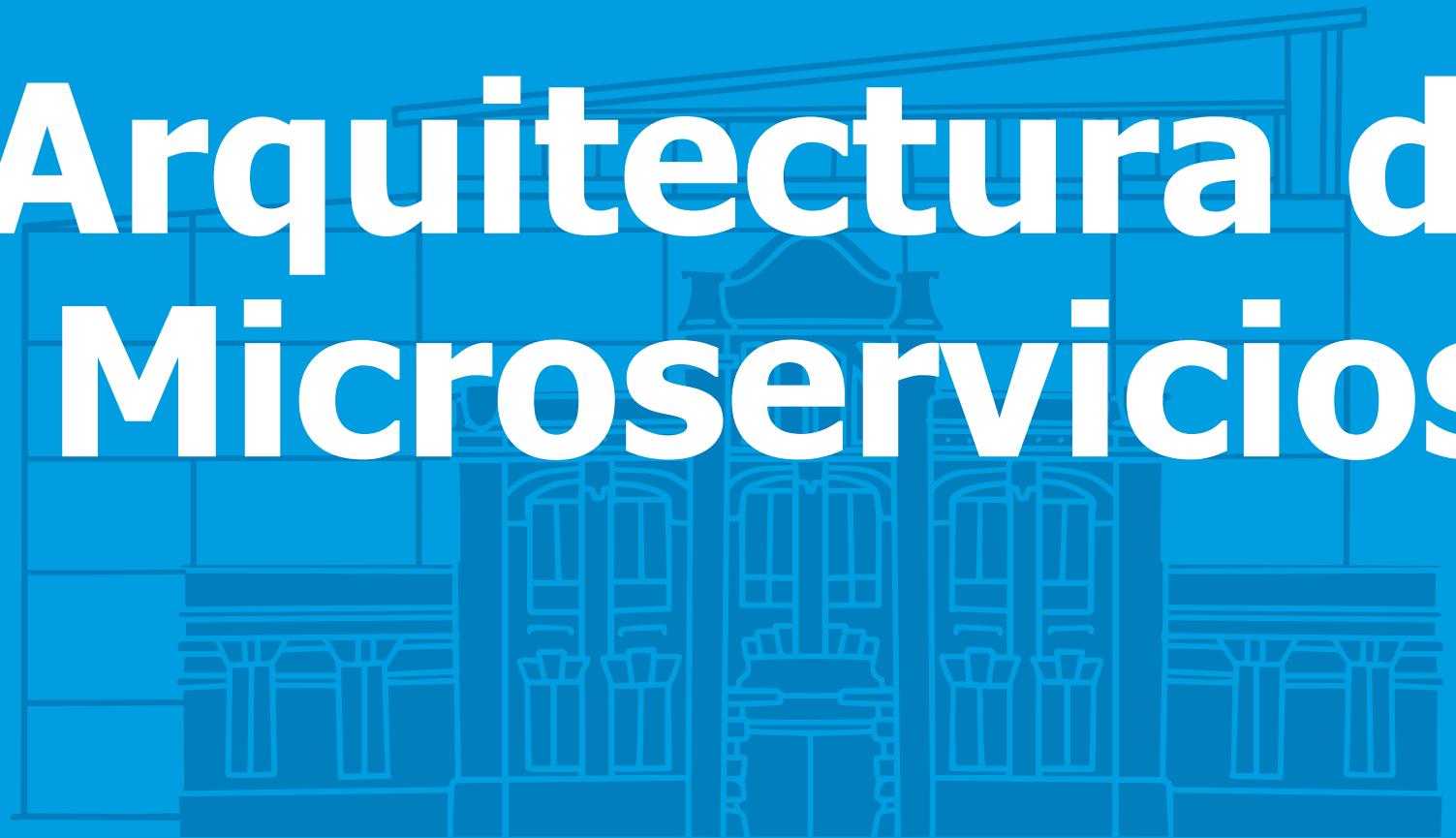
netmind
a bts company

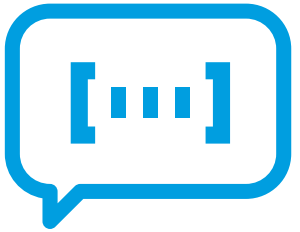
Spring Boot

Arquitectura de Microservicios



netmind
a bts company

01



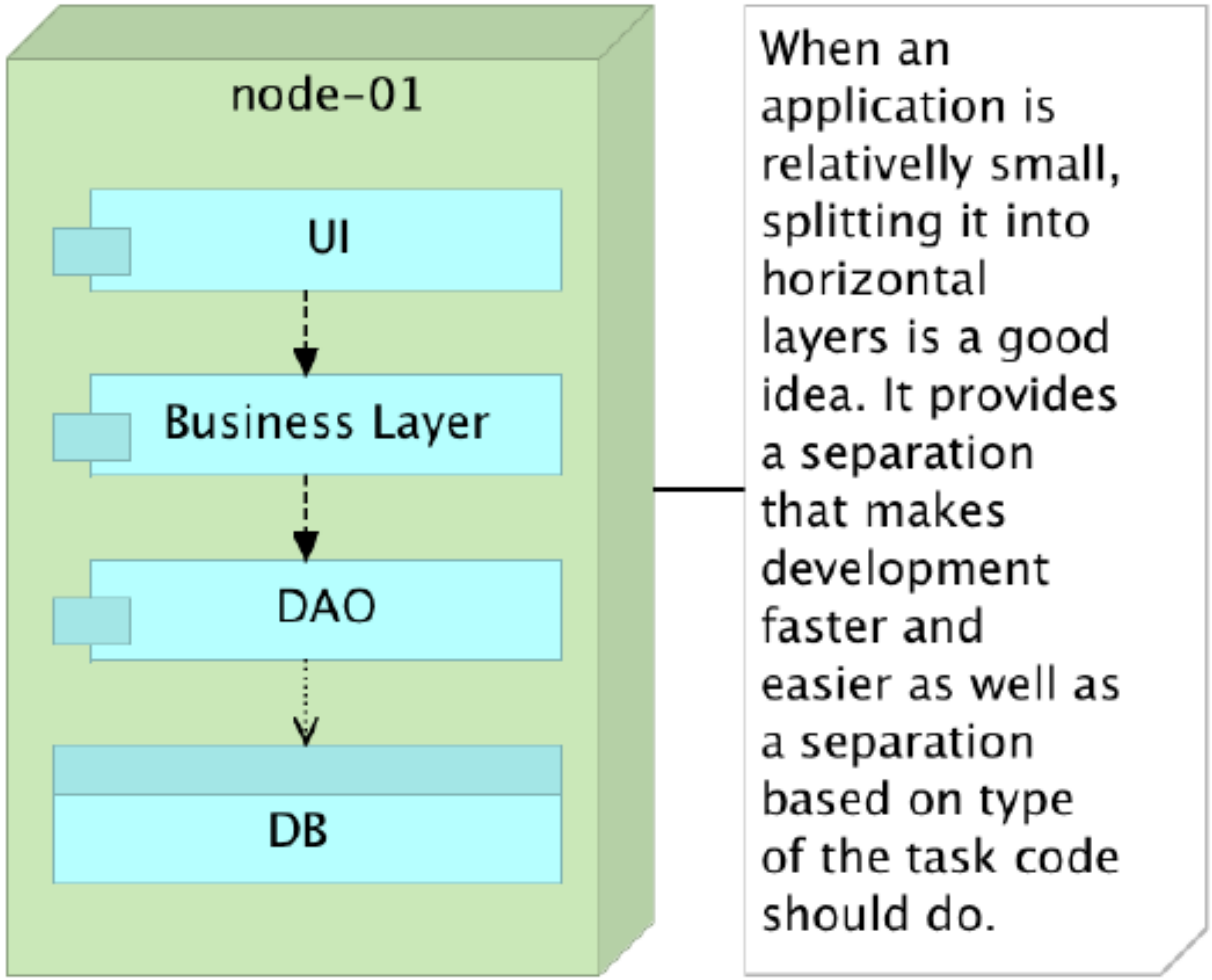
Aplicaciones Monolíticas

Aplicaciones tradicionales

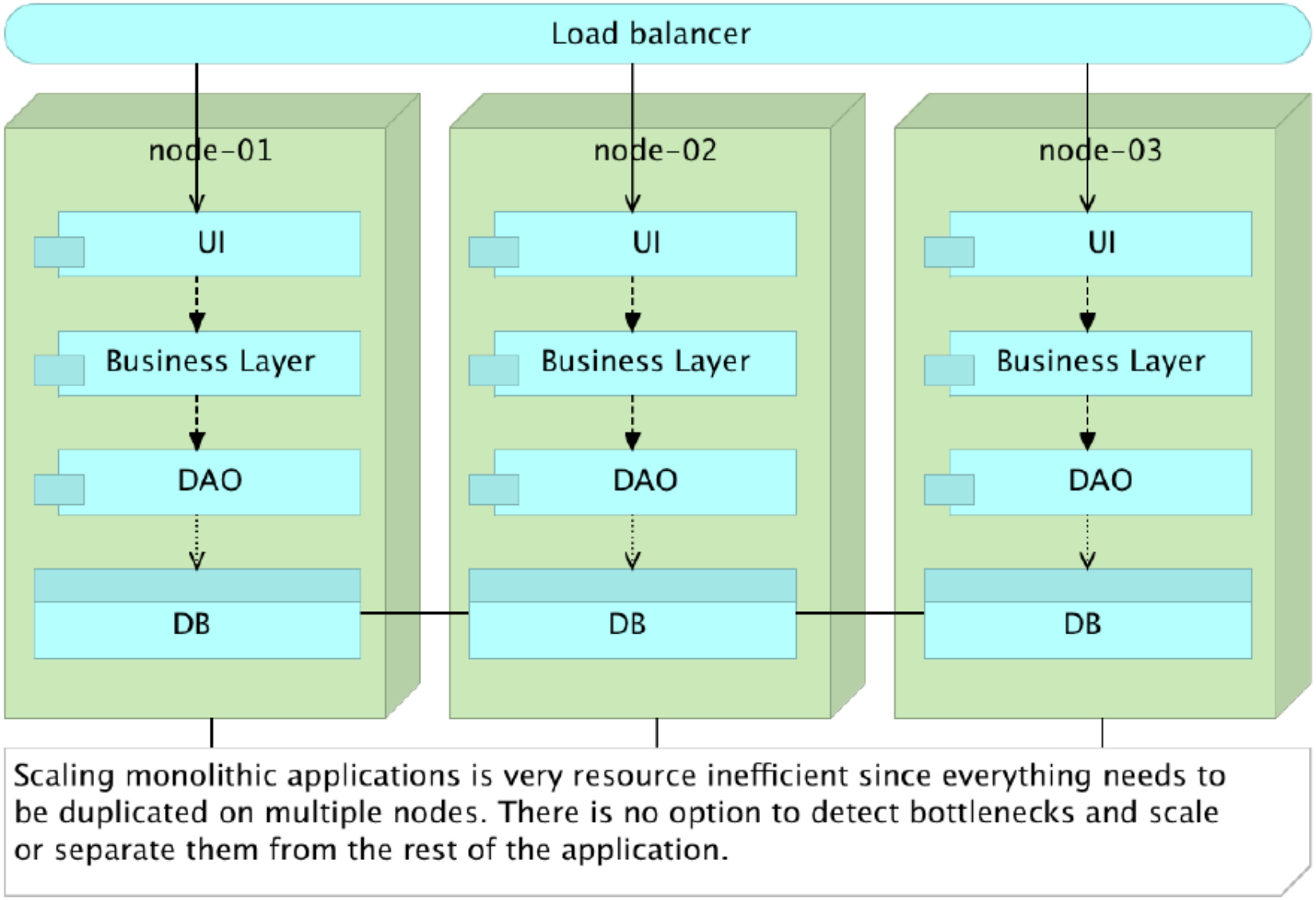
Pero qué pasa si...

- Queremos un casete doble
- O un reproductor de CD
- O una radio digital
- ...

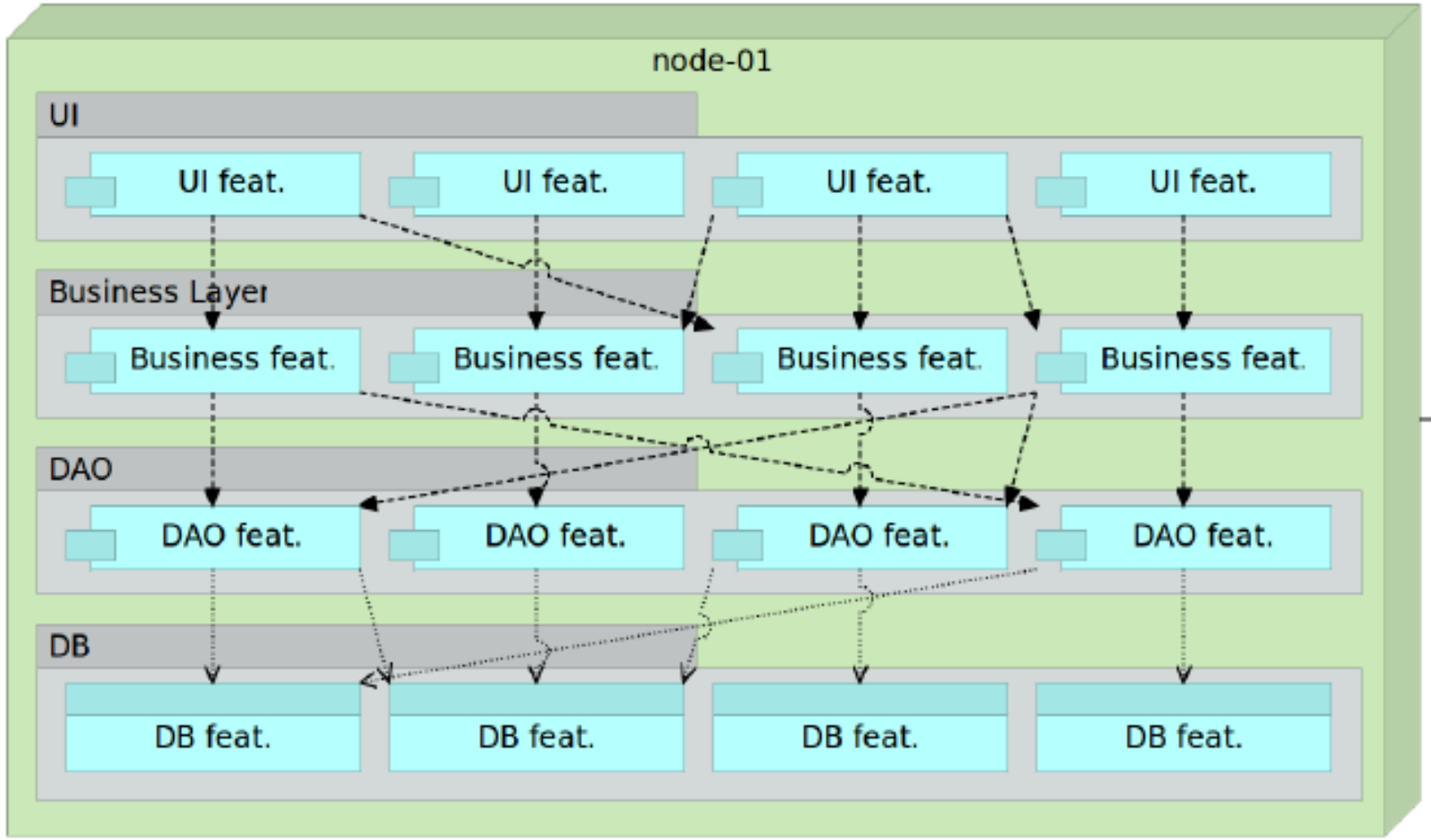
Aplicaciones monolíticias



Escalando aplicaciones monolíticas



Aplicación monolítica con mayor número de características



When an application becomes bigger and the number of features increase, initial design based on horizontal layers becomes less efficient. Tight coupling between separate features, longer paths for potentially simple solutions, increased complexity, increased development and testing time, and so on.

02

SOA

Cómo comprar tomates?

1. Buscar los vendedores de tomate
   - Páginas amarillas: contienen empresas que venden tomates, su ubicación e información de contacto.

2. Encontrar el servicio ofrecido de acuerdo a mis necesidades
   - ¿Dónde, cuándo y cómo puedo comprar tomates?

3. Comprar los tomates
   - Hacer la transacción

Cómo acceder a un servicio?

1. Buscar del proveedor de servicios
   - Registro: contiene proveedores que están vendiendo servicios, su ubicación y datos de contacto.

2. Encontrar el servicio ofrecido de acuerdo a mis necesidades
   - ¿Dónde, cuándo y cómo puedo obtener el servicio?

3. Acceder al servicio
   - Hacer la transacción

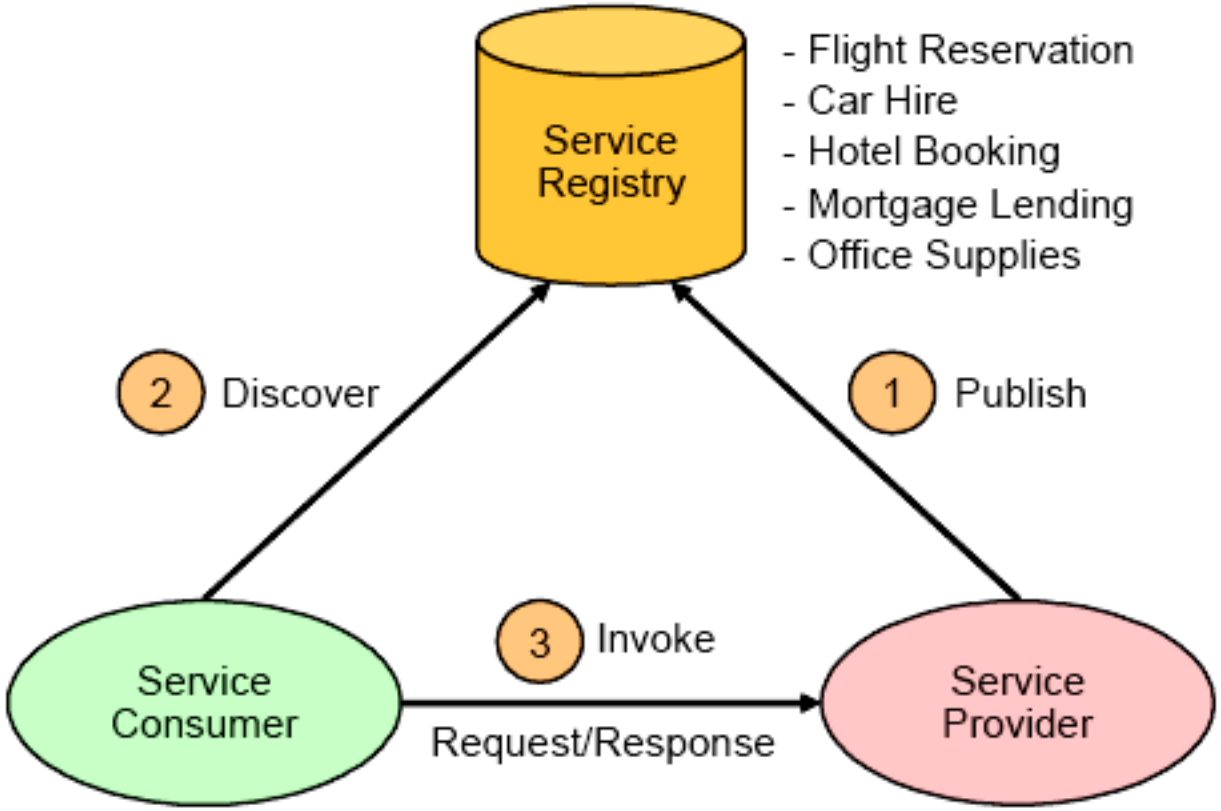
Service-Oriented Architecture (SOA):

- Las políticas, prácticas, frameworks que permiten proporcionar y consumir la funcionalidad de la aplicación como conjuntos de servicios publicados con una granularidad relevante para el consumidor del servicio.

- Los servicios pueden ser
  - invocados,
  - publicados y
  - descubiertos,

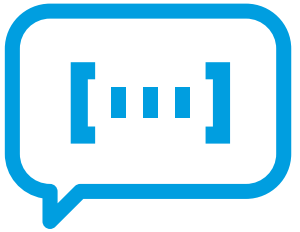
- y se abstraen de la implementación utilizando una única forma de interfaz basada en estándares.

SOA - Componentes y operaciones



Application-A
- Travel Agent
- Retail Bank
- Publishing House

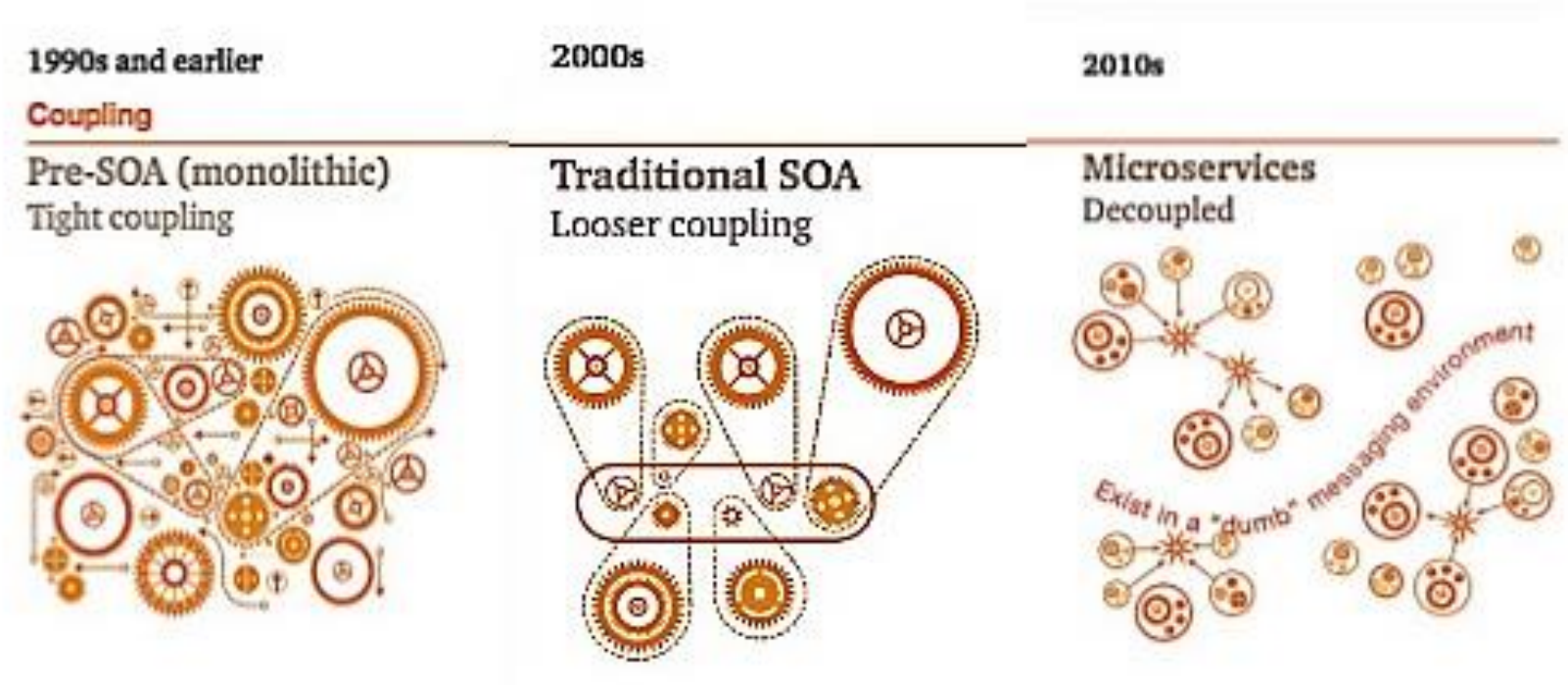
Application-B
- Airline/Car Rental/Hotel Chain
- Mortgage Specialist/Investment Banks
- Office Supplies Company



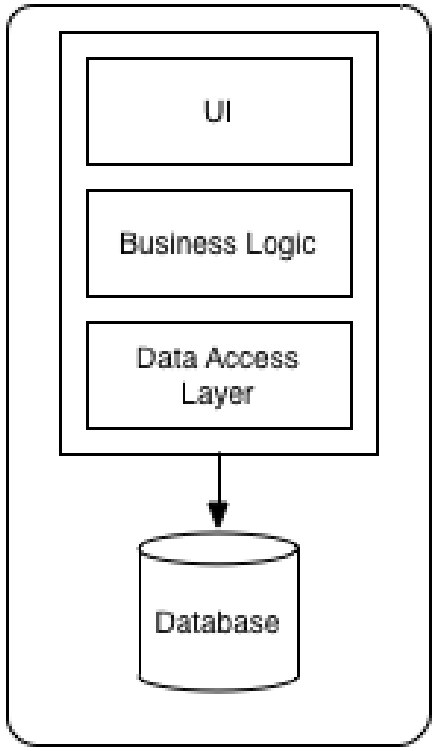
Microservicios

03

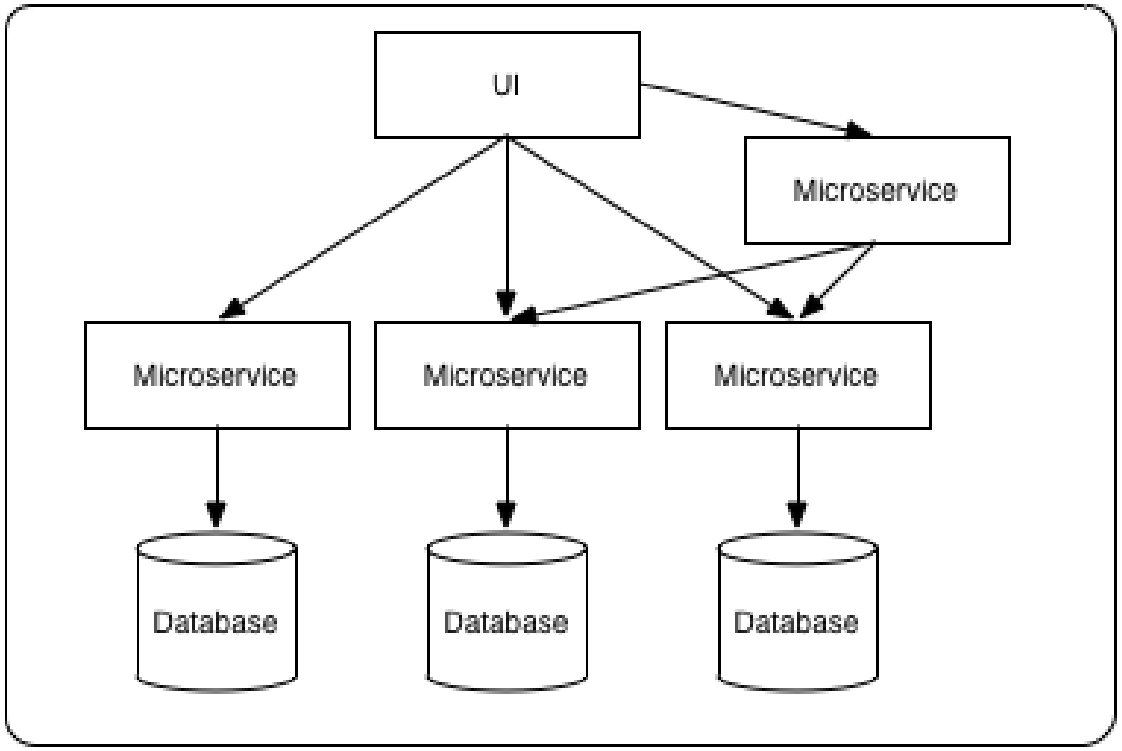
Microservicios



Microservicios

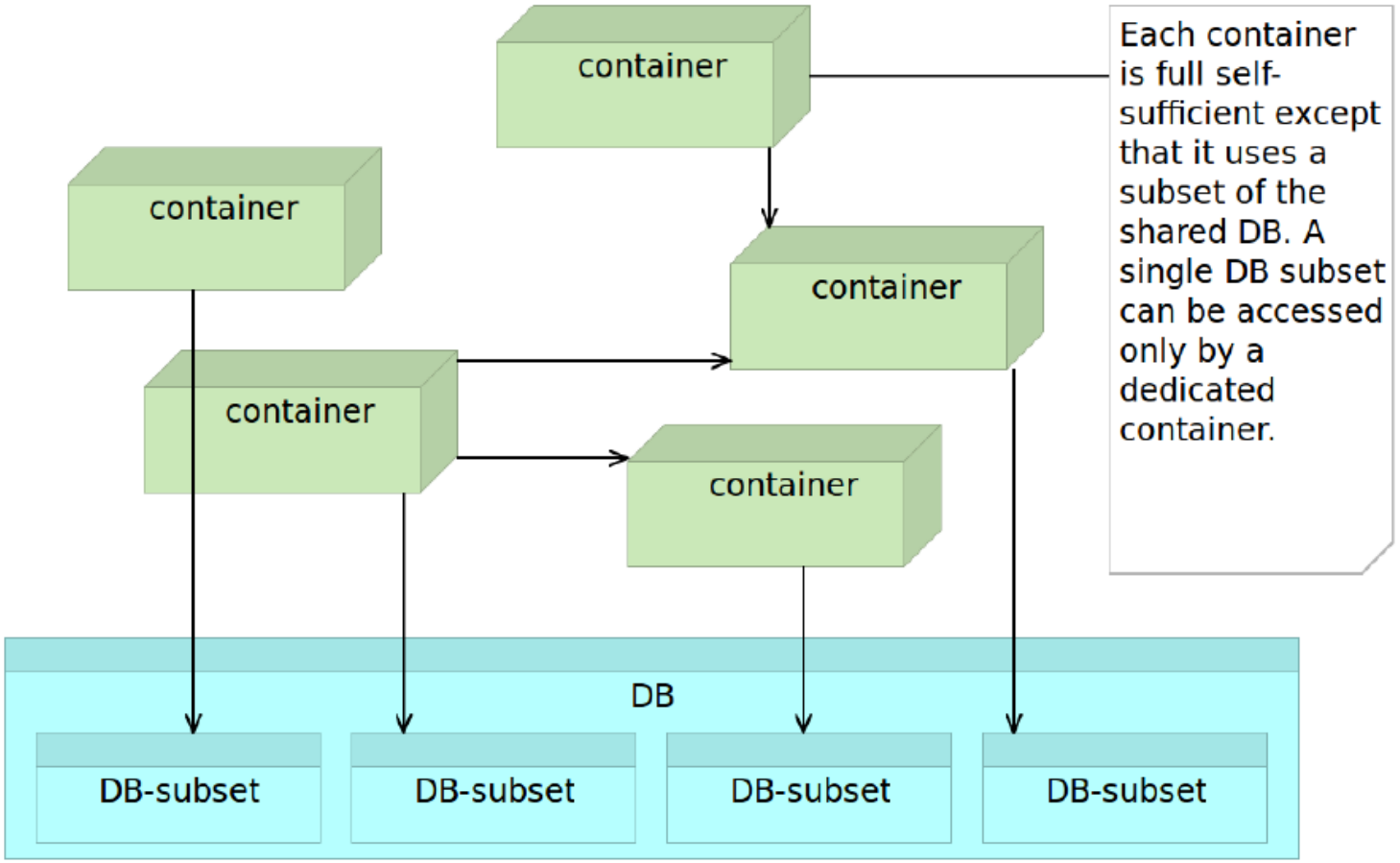


Monolithic Architecture



Microservices Architecture

Microservicios: accediendo a la base de datos compartida

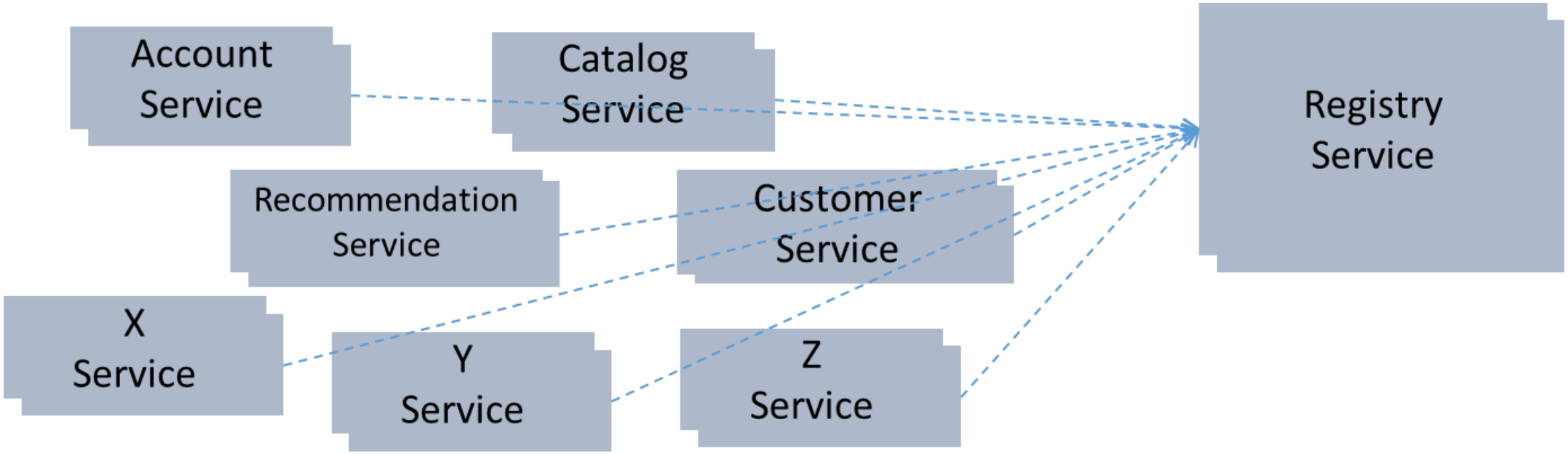


Características

1. Muchos servicios pequeños (de granularidad fina), con un alcance claro
   - Principio de responsabilidad única
   - Gestionado de forma independiente
2. La “propiedad” está clara para cada servicio
   - Por lo general, necesita/adopta el modelo "DevOps"

Descubrimiento de servicios

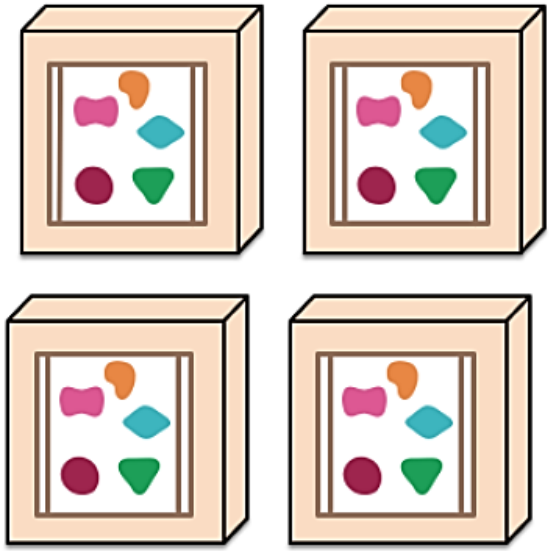
- 100s de MicroServicios
  - Necesita un servicio de registro de metadatos (servicio de descubrimiento)



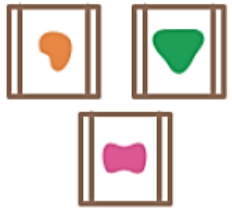
Escalabilidad de servicios

A monolithic application puts all its functionality into a single process...

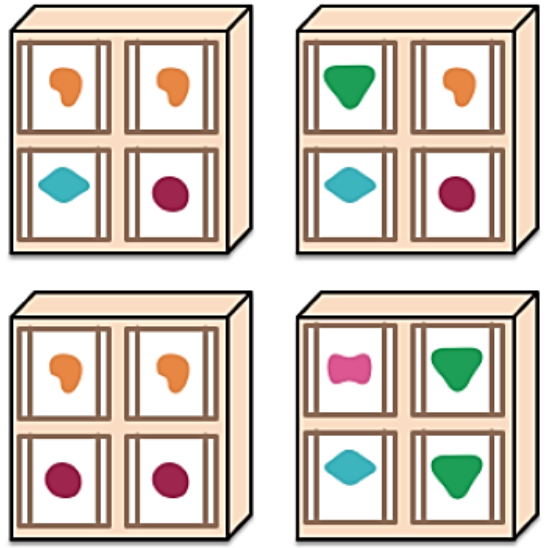
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Metodologías para definir microservicios

- Object Oriented Analysis and Design
- Test Driven Development
- Behavior Driven Development
- Domain-Driven Design

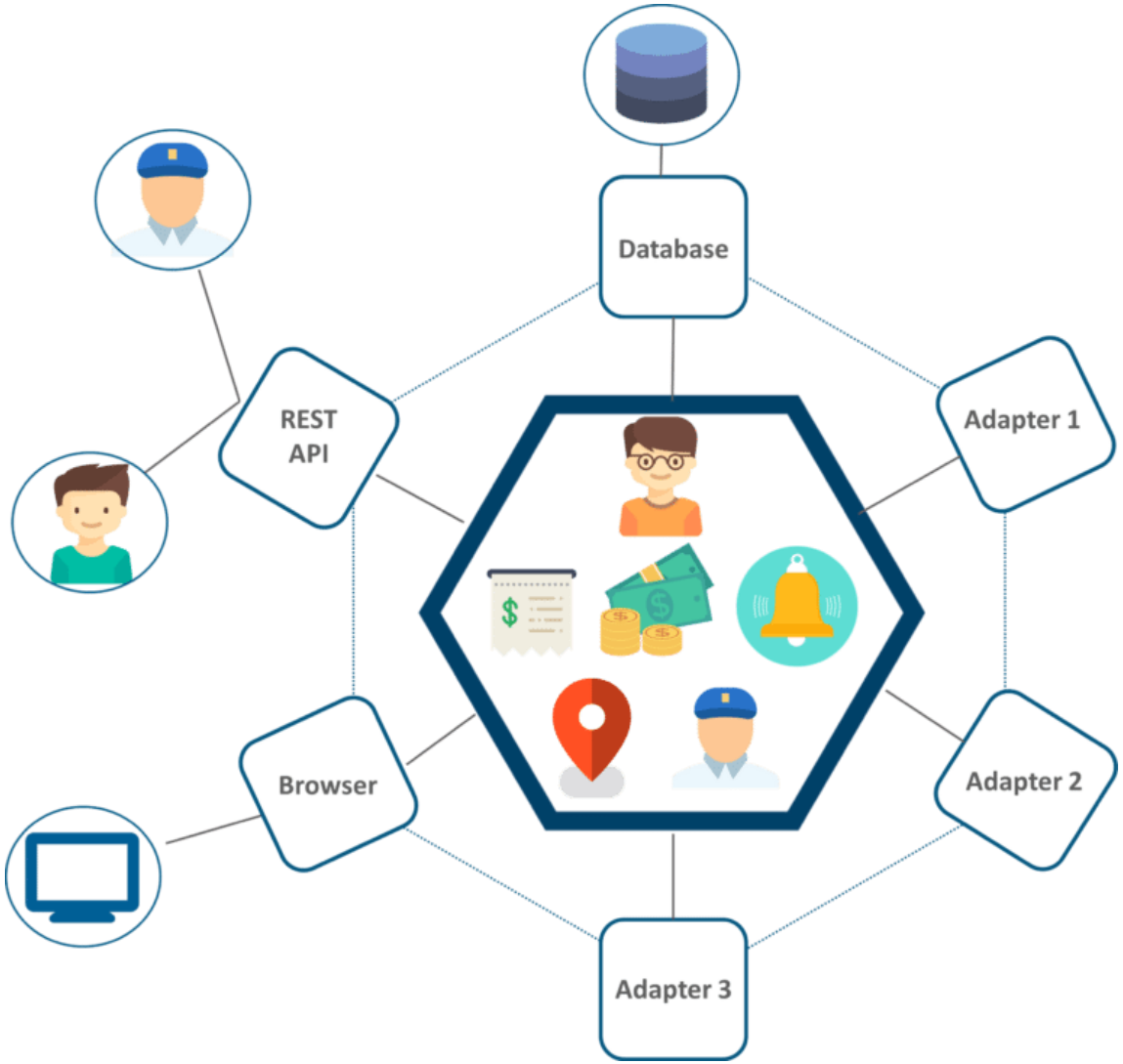
¿Cuándo necesitamos un arquitectura de microservicios?

- **Incorporación más fácil para un área de negocio**: ¿Los servicios representan casos/dominios de negocio no triviales y diferentes?

- **Gestión independiente para una implementación más rápida**: ¿los servicios deben implementarse y administrarse de forma independiente y más rápida?

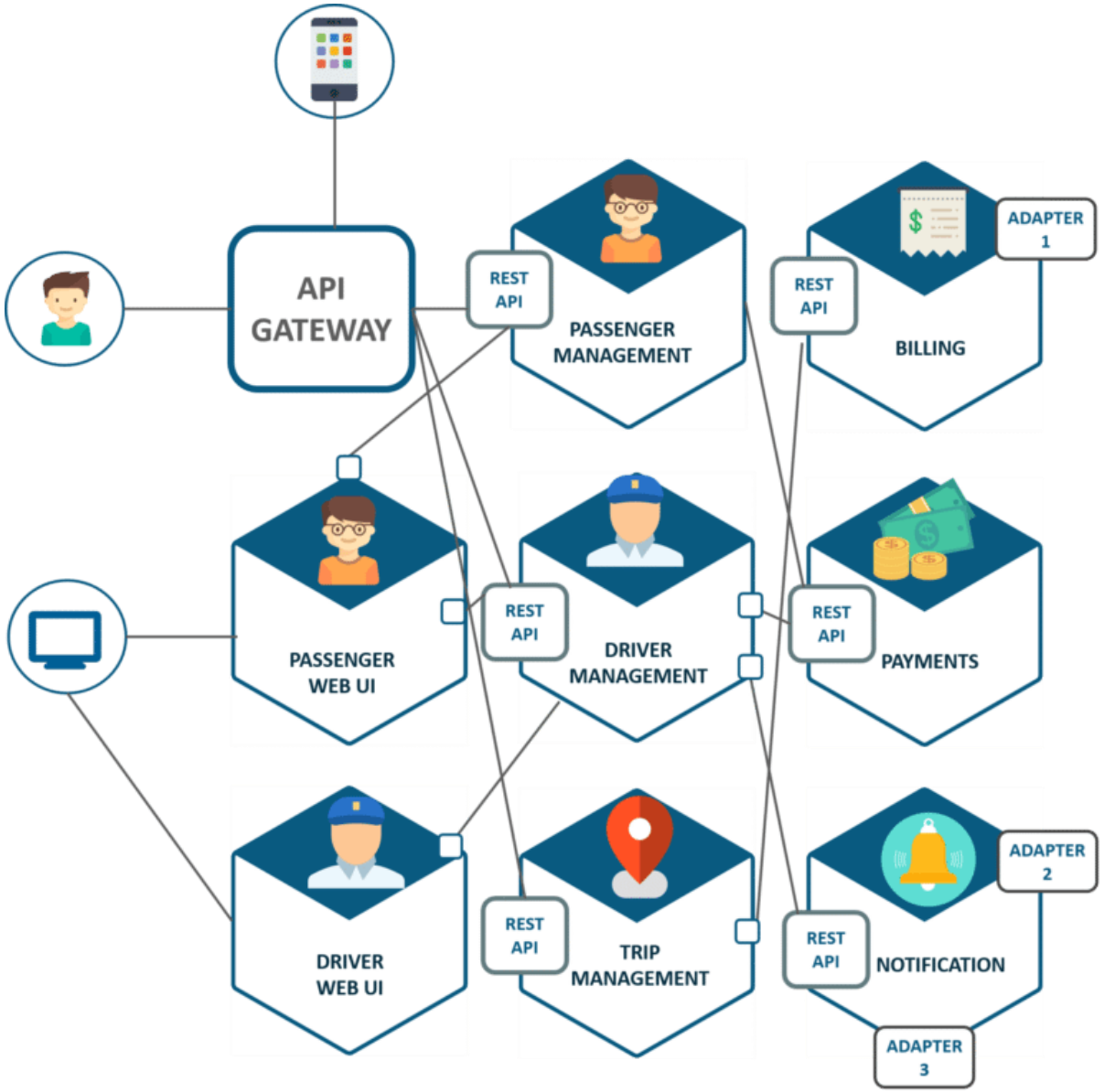
- **Escalado más inteligente:** ¿Las diferentes partes de la aplicación tienen diferentes necesidades de escalado/tecnología?

UBER CASE STUDY: inicial

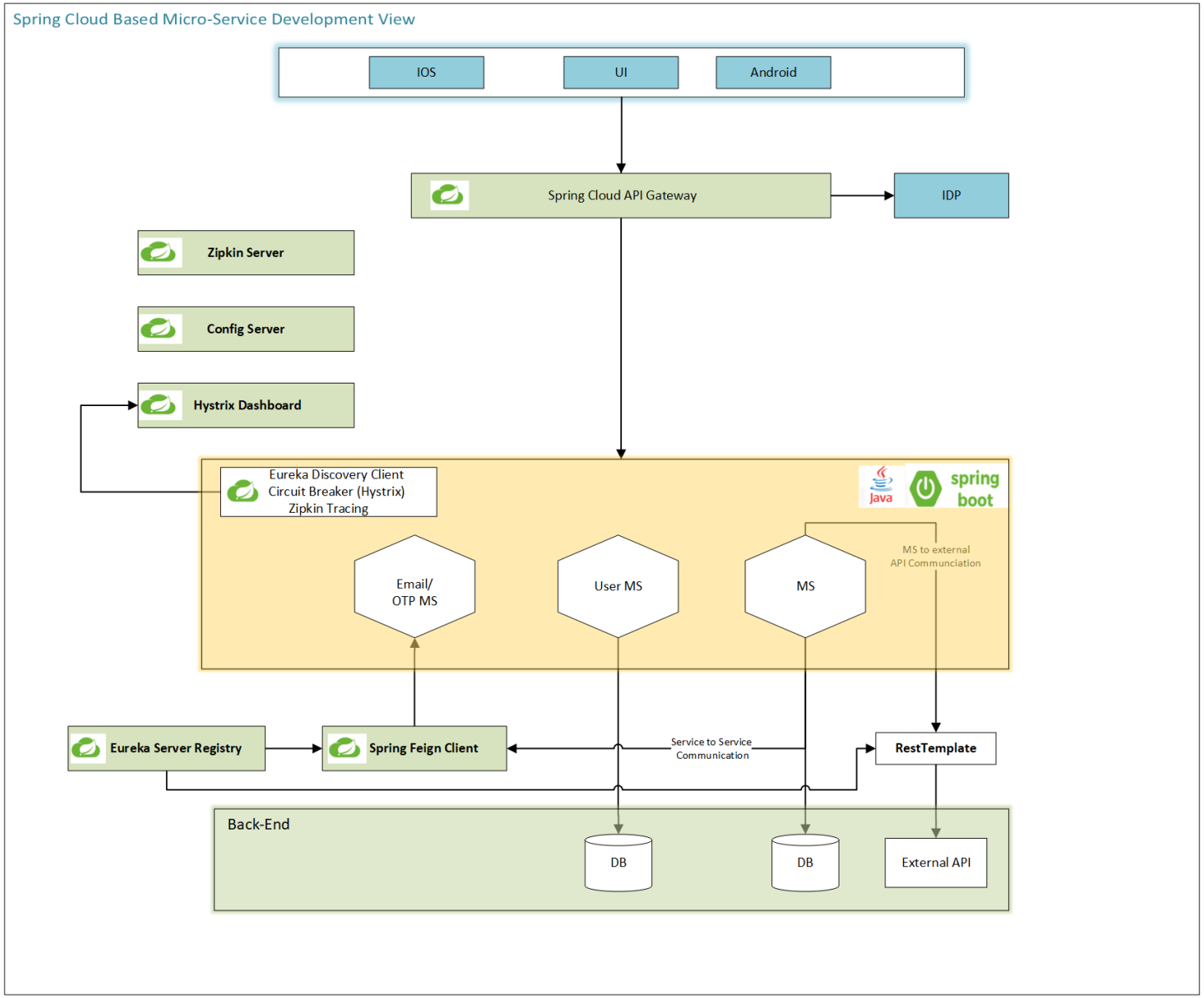


Ref: https://medium.com/edureka/microservice-architecture-5e7f056b90f1

UBER CASE STUDY: microservicios

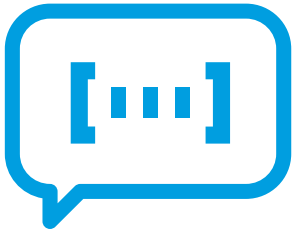


Microservicios y Spring boot



netmind
a bts company

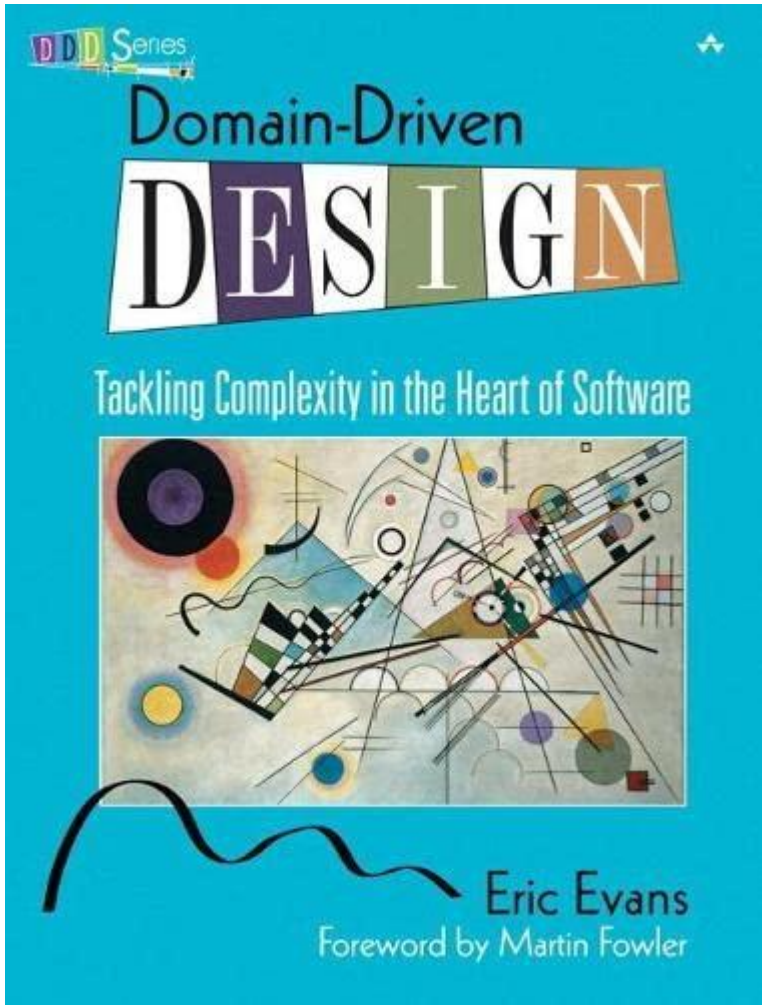
04



DDD

Domain Driven Design

- Propuesta de Eric Evans en el libro Domain- Driven Design. Tackling Complexity in the Heart of Software
- Nace dentro del movimiento XP.
  - Programadores hablando con usuarios… (Parece mentira eh…)
  - Refactorización continua.
- Se basa en:
  - Predominio del dominio sobre el resto de elementos de un sistema.
  - Es más importante un diseño consistente que las tecnologías empleadas.
  - Una propuesta contra el antipatrón AnemicDomainModel.



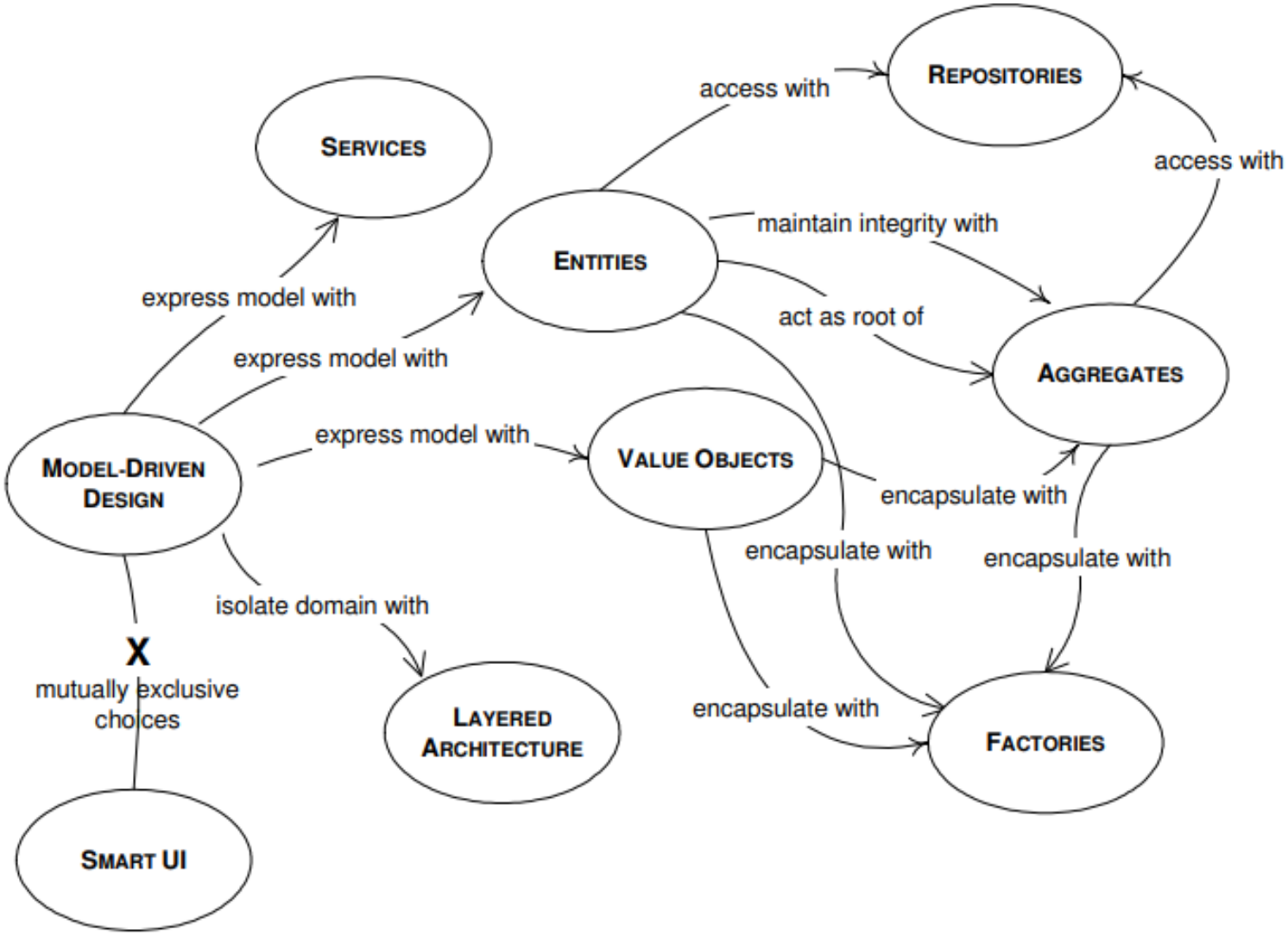
Conceptos de DDD

1. **Ubiquitous Language:** establecer un lenguaje que compartan todos los miembros de un equipo.
   - Un lenguaje común.
   - Es la base sobre la que construimos un modelo.

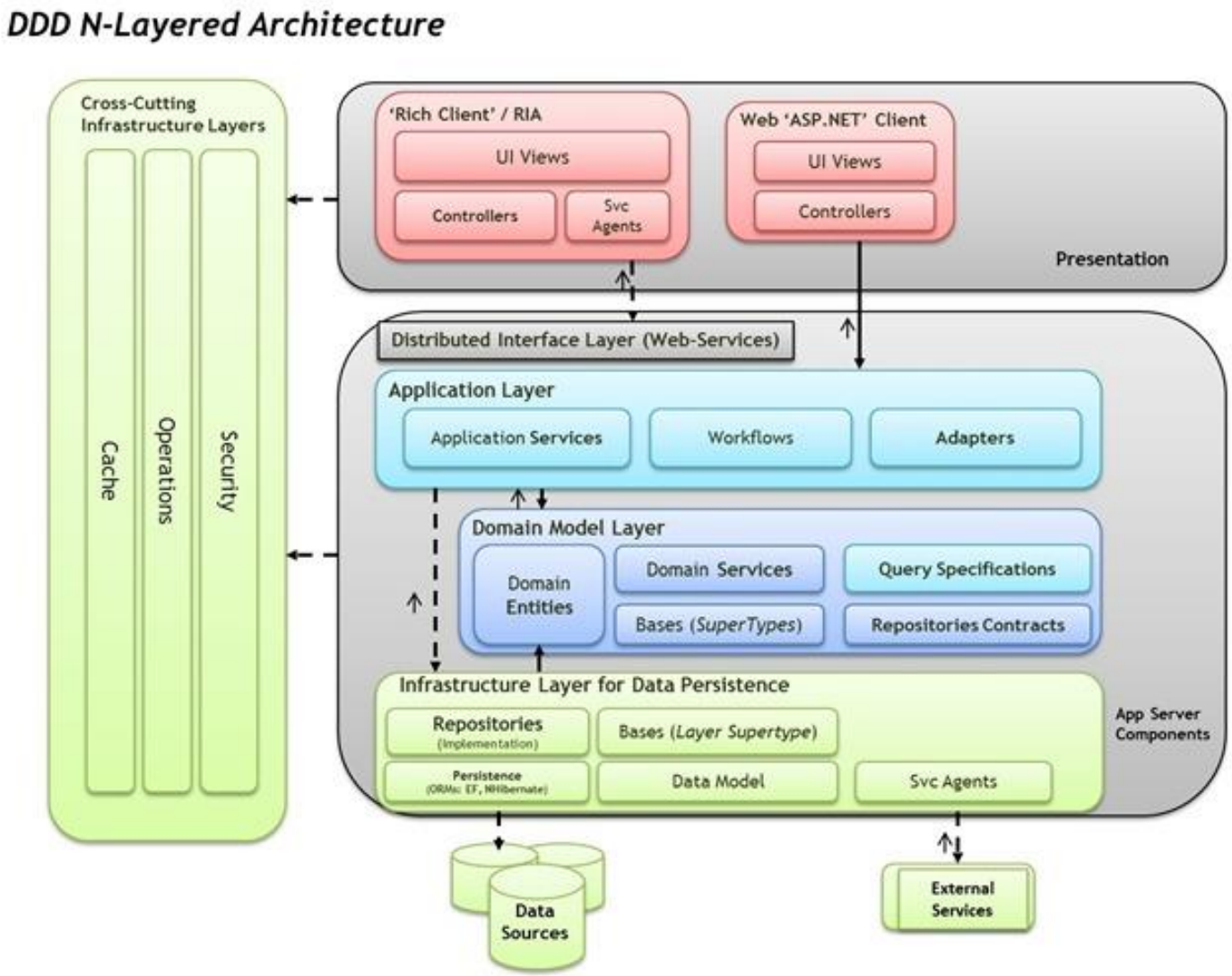
2. **Persistance Ignorance:** no se graba, se persiste.
   - Como almacene los datos no debe condicionar el modelo.
   - Tampoco cómo se muestre o explore.

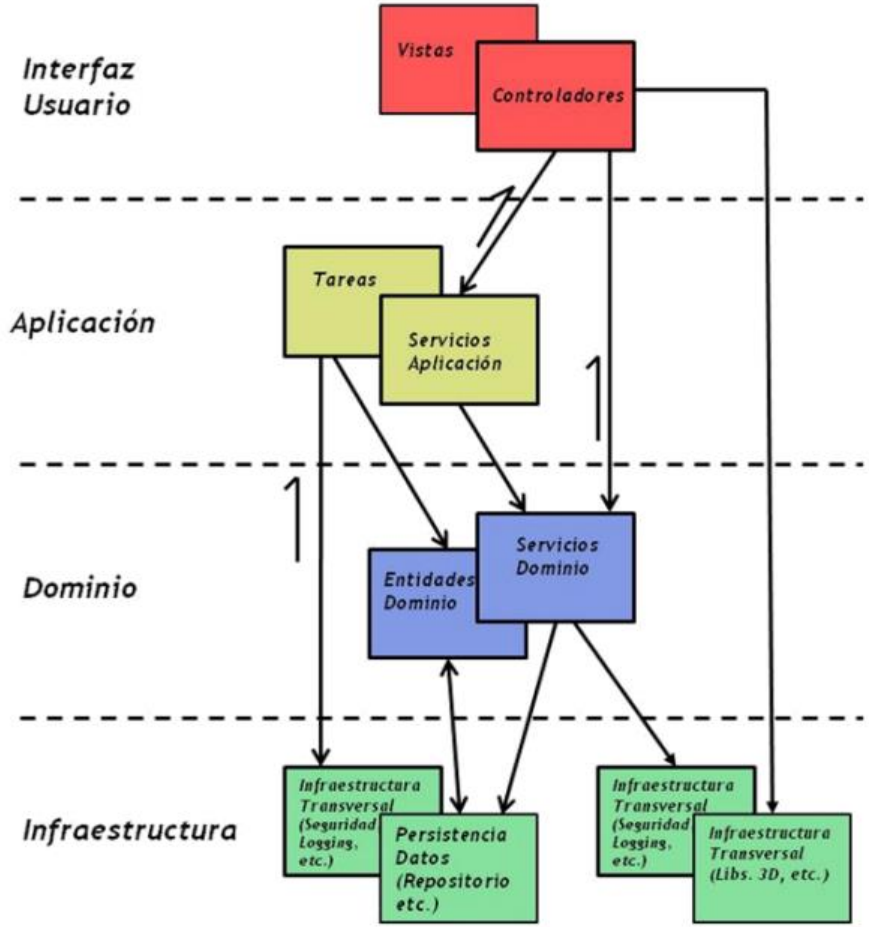
3. **El corazón del software:** construimos el corazón del software
   - El modelo es el corazón de nuestro software.
   - Debe ser consistente.
   - TDD. Mocks.

Modelo teórico (Evans)

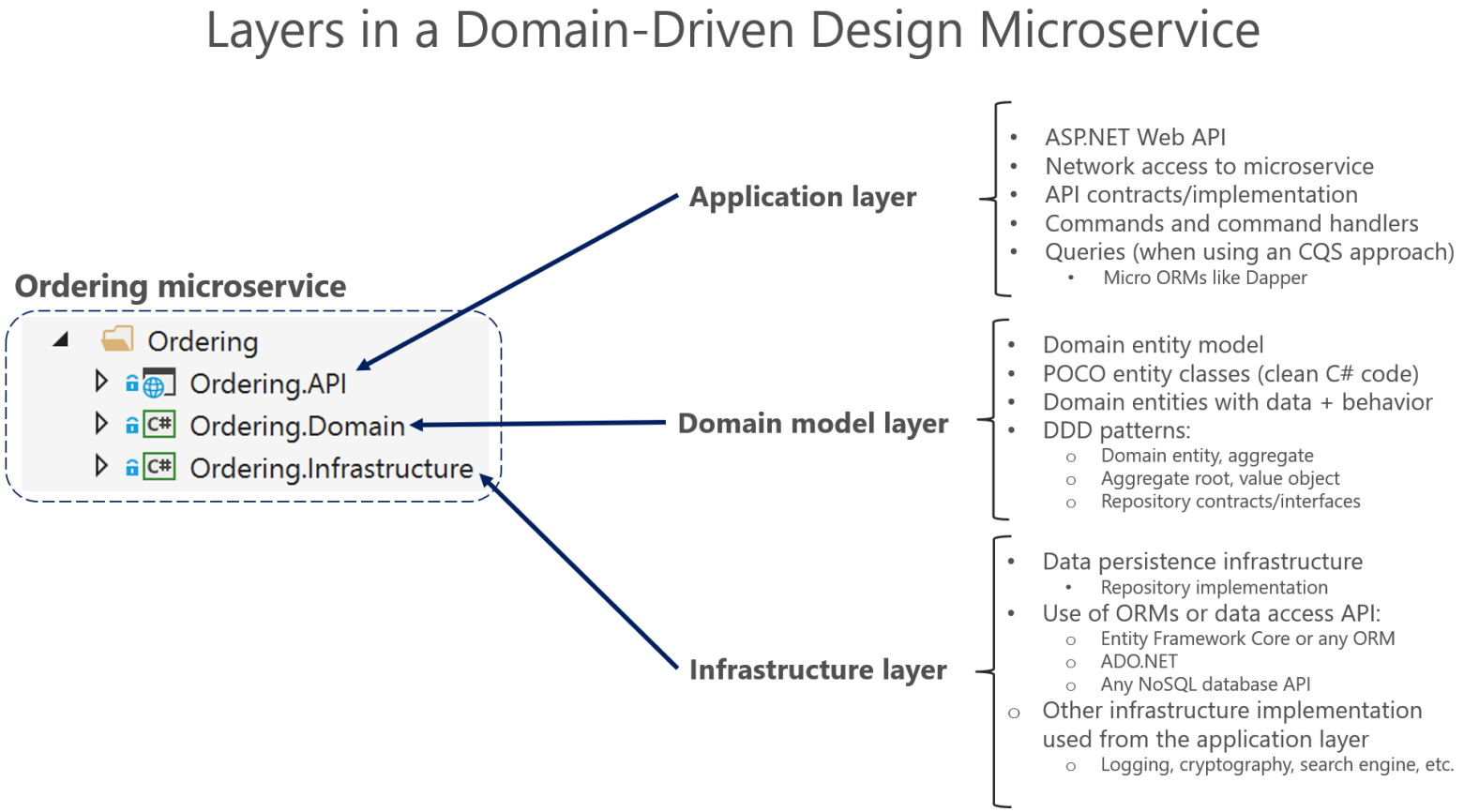


Modelo algo más práctico (Microsoft)

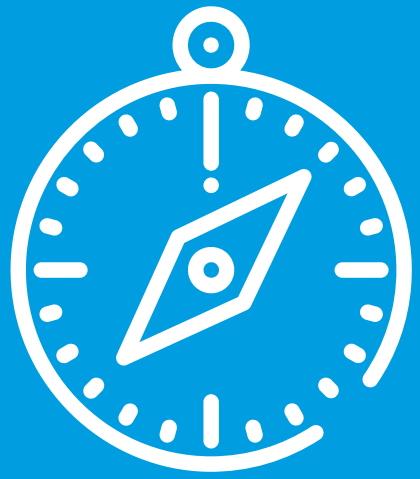




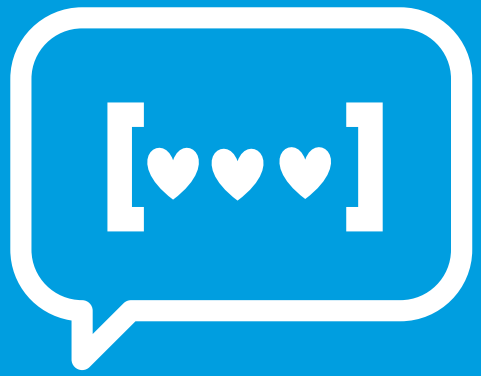
DDD y microservicios



https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice



Next steps



We would like to know your opinion!

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

Thanks!

Follow us: