# BOOTCAMP JAVA-MICROSERVICIOS | SESIÓN 04

BOOTCAMP JAVA-MICROSERVICIOS | Perú 2022

Por: Angelo Rafael Angulo Méndez

aangulom@emeal.nttdata.com

# AGENDA

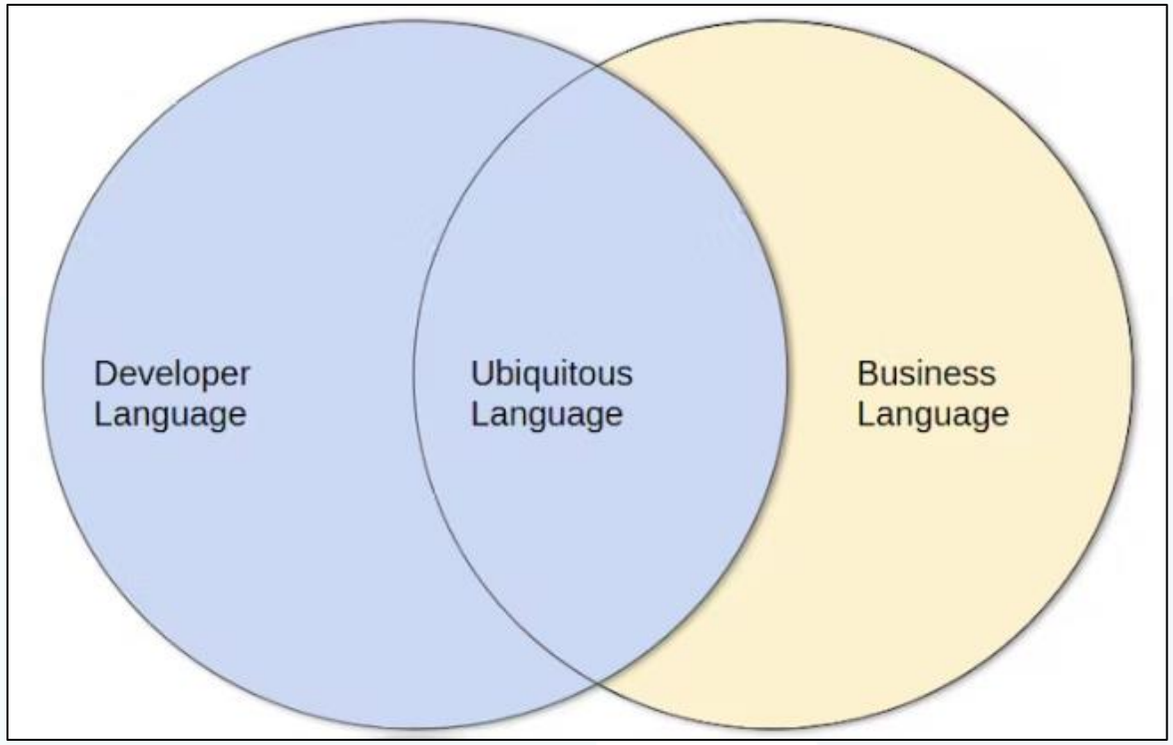
DISEÑO DIRIGIDO POR DOMINIO (DDD)

STACK DE SPRING CLOUD: CONFIG SERVER

NTT DATA

# Diseño dirigido por dominio

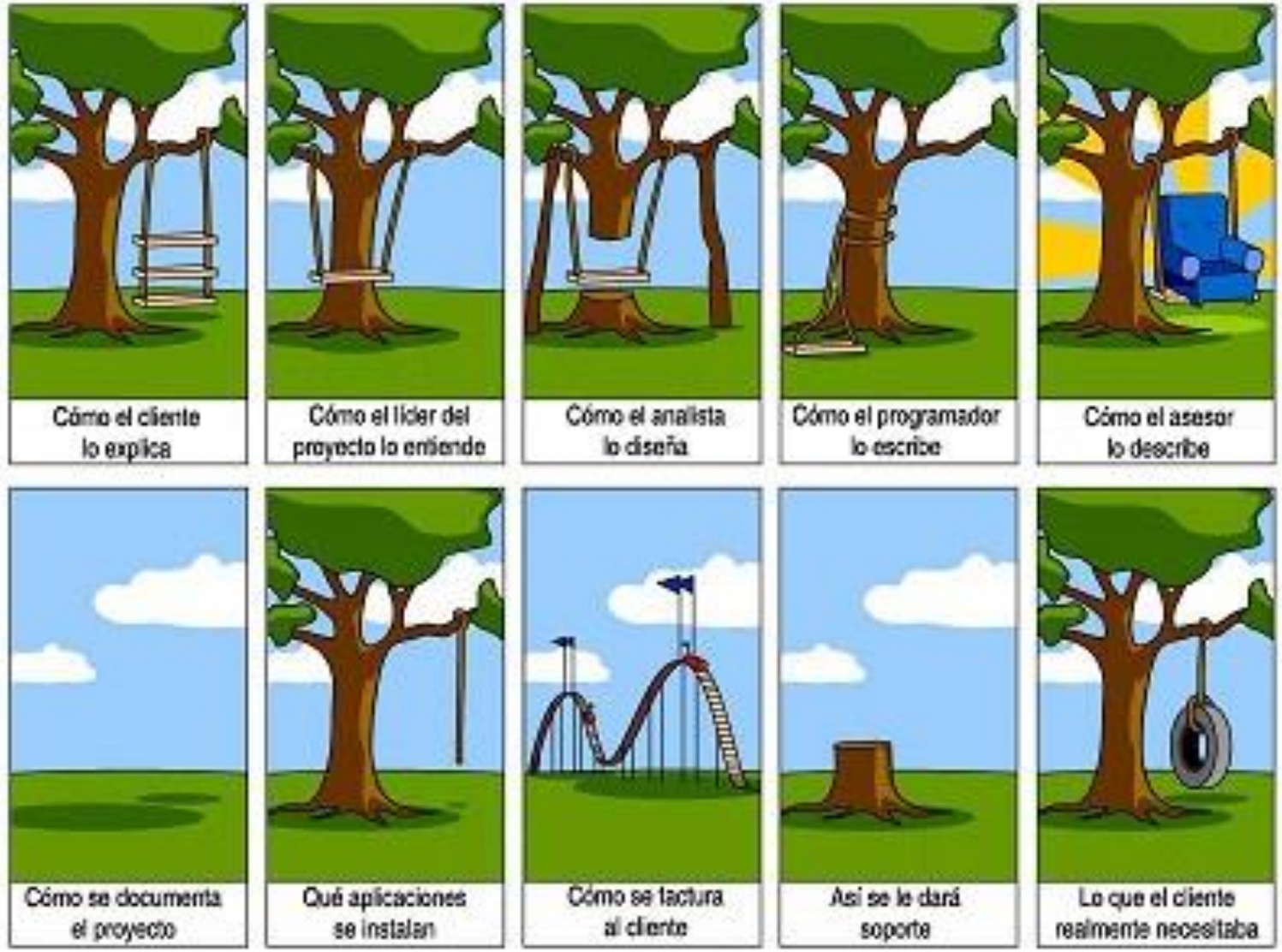
# Diseño basado en Dominio




- ✓ ***El diseño basado en dominio (DDD)*** *aboga por el modelado basado en la realidad del negocio como relevante para sus casos de uso. En el contexto de la creación de aplicaciones, DDD habla de problemas como* ***dominios****. Describe áreas de problemas independientes como* ***contextos*** *acotados (cada contexto acotado se correlaciona con un microservicio) y enfatiza un* ***lenguaje común*** *para hablar sobre estos problemas*

- ✓ *No es una* ***tecnología ni una metodología****, este provee una estructura de prácticas y terminologías para tomar decisiones de diseño que enfoquen y aceleren el manejo de dominios complejos en los proyectos de software.*

# Diseño basado en Dominio



# Ventajas de usar DDD

Comunicación efectiva entre expertos del dominio y expertos técnicos a través de ***Ubiquitous Language***.

Foco en el desarrollo de un área dividida del dominio (subdominio) a través de ***Bounded Context's***.

El software es más cercano al dominio, y por lo tanto es más cercano al cliente.

Código bien organizado, permitiendo el testing de las distintas partes del dominio de manera aisladas.

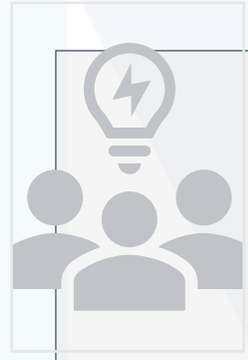
Lógica de negocio reside en un solo lugar, y dividida por contextos.

Mantenibilidad *a largo plazo*.

# Premisas al usar DDD

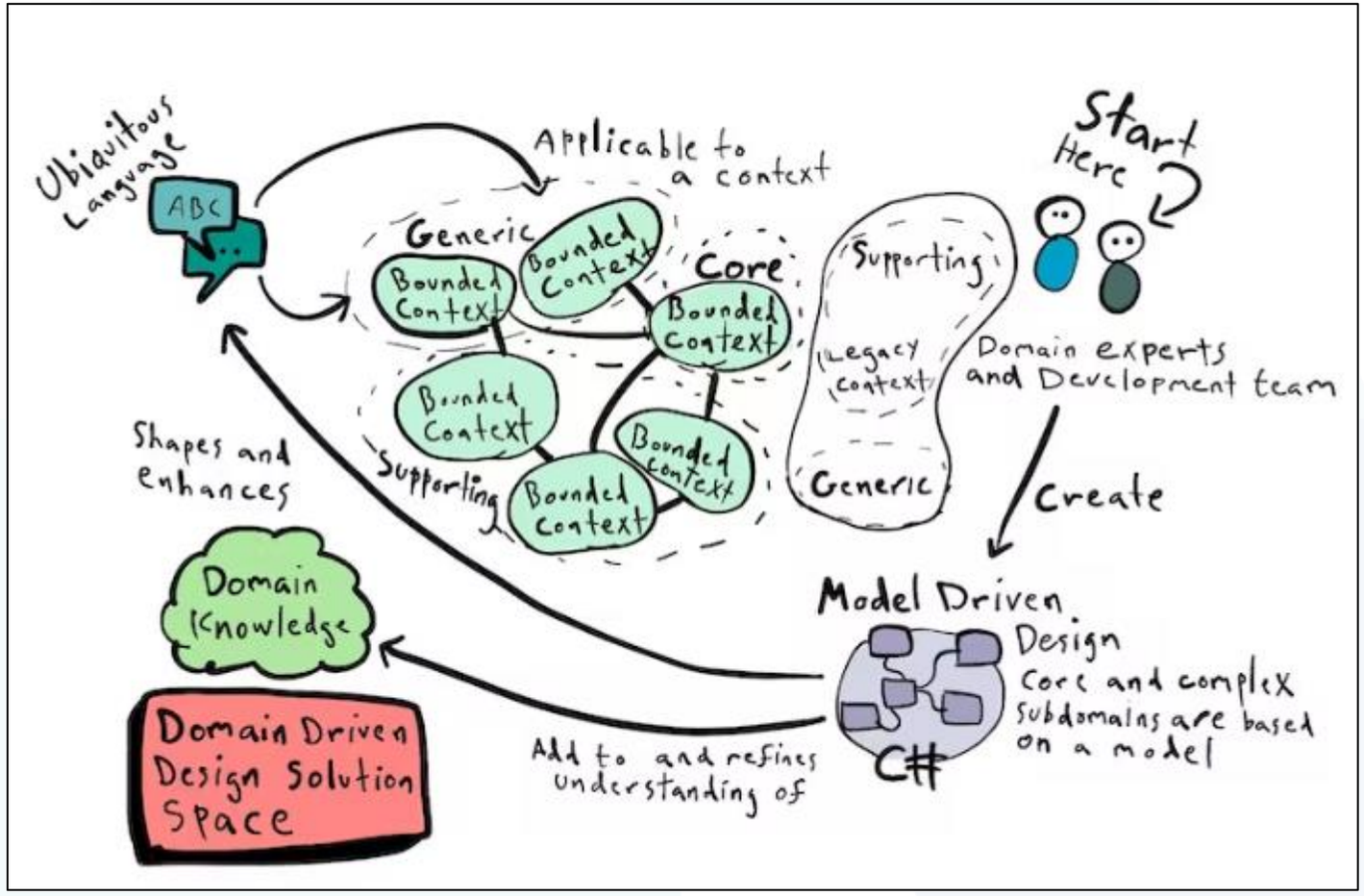
Poner el foco primario del proyecto en el núcleo y la lógica del dominio.

Basar los diseños complejos en un modelo.



Iniciar una creativa colaboración entre técnicos y expertos del dominio para interactuar lo más cercano posible a los conceptos fundamentales del problema.

# Fases del DDD



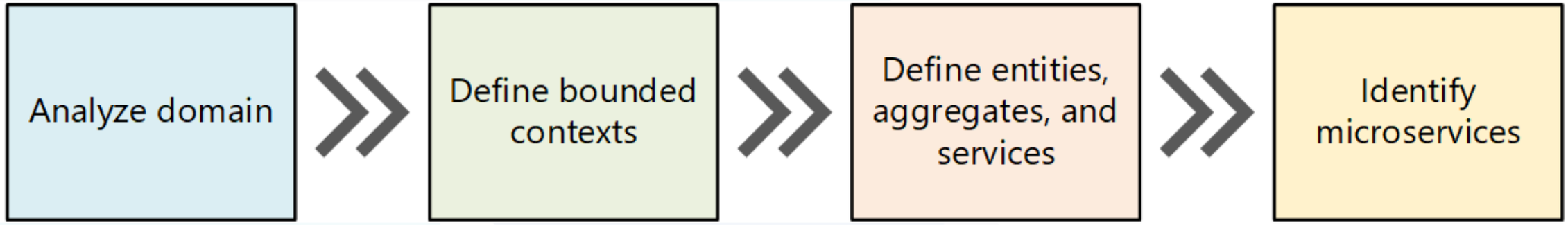

- ✓ *El diseño basado en dominios tiene dos fases distintas*
- ✓ *En el diseño basado en **dominios estratégico**, se define la estructura a gran escala del sistema. Ayuda a garantizar que la arquitectura permanece centrada en las funcionalidades del negocio.*
- ✓ *El diseño basado en **dominios táctico** proporciona un conjunto de modelos de diseño que puede usar para crear el modelo de dominio*

# Ejemplo uso DDD en el diseño de Microservicio – Caso de Uso

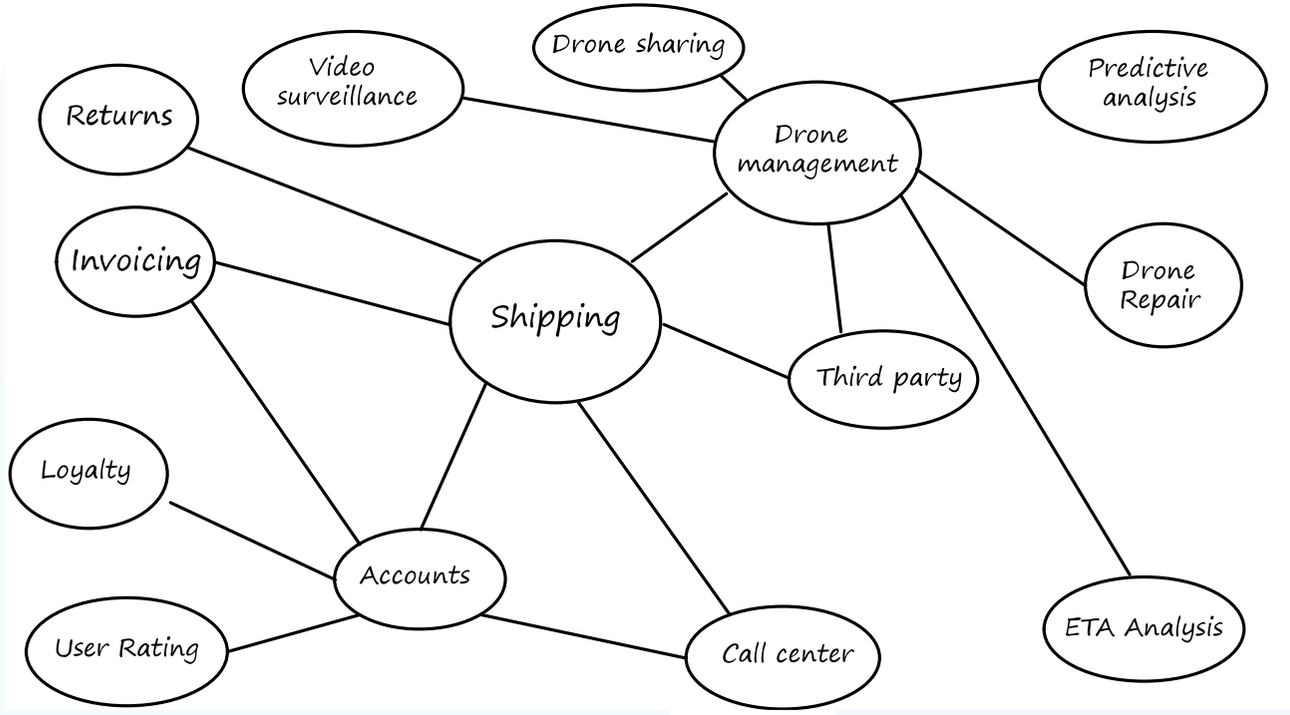

- Fabrikam, Inc. está iniciando un servicio de entrega con drones. La empresa administra una flota de drones. Las empresas se registran en el servicio y los usuarios pueden solicitar que un dron recoja los bienes para la entrega cuando un cliente programa una recogida, un sistema back-end asigna un dron y notifica al usuario con un tiempo de entrega estimado. Con la entrega en curso, el cliente puede realizar el seguimiento del dron, con una fecha estimada que se actualiza constantemente.

https://learn.microsoft.com/es-es/azure/architecture/microservices/model/domain-analysis

# Ejemplo uso DDD en el diseño de Microservicio - Fases



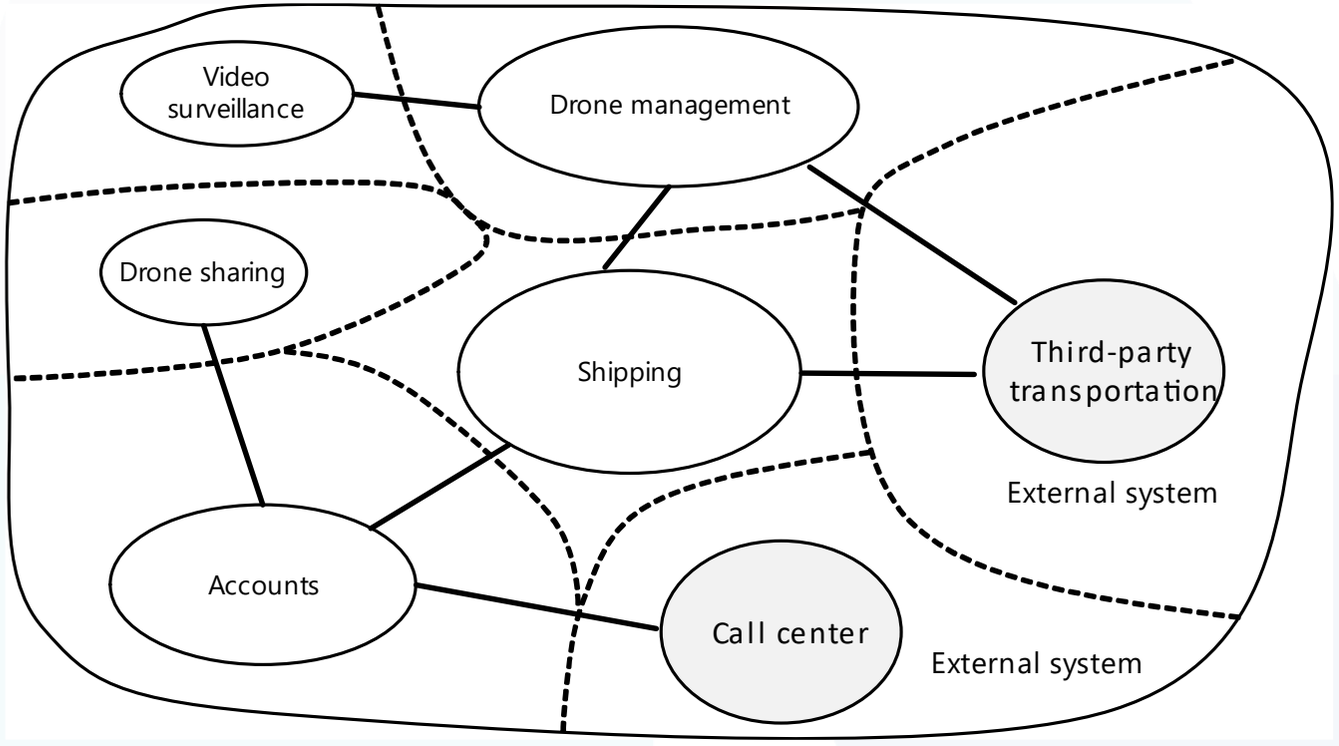
# Ejemplo uso DDD en el diseño de Microservicio - Analizando el Dominio




https://learn.microsoft.com/es-es/azure/architecture/microservices/model/domain-analysis

- ✓ *Antes de escribir ningún código, necesita una **visión general del sistema** que se va a crear.*
- ✓ *Con el diseño basado en dominios, se empieza por modelar el **dominio empresarial**.*
- ✓ *Empiece por asignar todas las **funciones empresariales** y sus conexiones. Probablemente, esto requerirá la colaboración de los expertos en dominios, los arquitectos de software y otros actores implicados*
- ✓ *A medida que rellene el diagrama, puede empezar a identificar **subdominios discretos.***

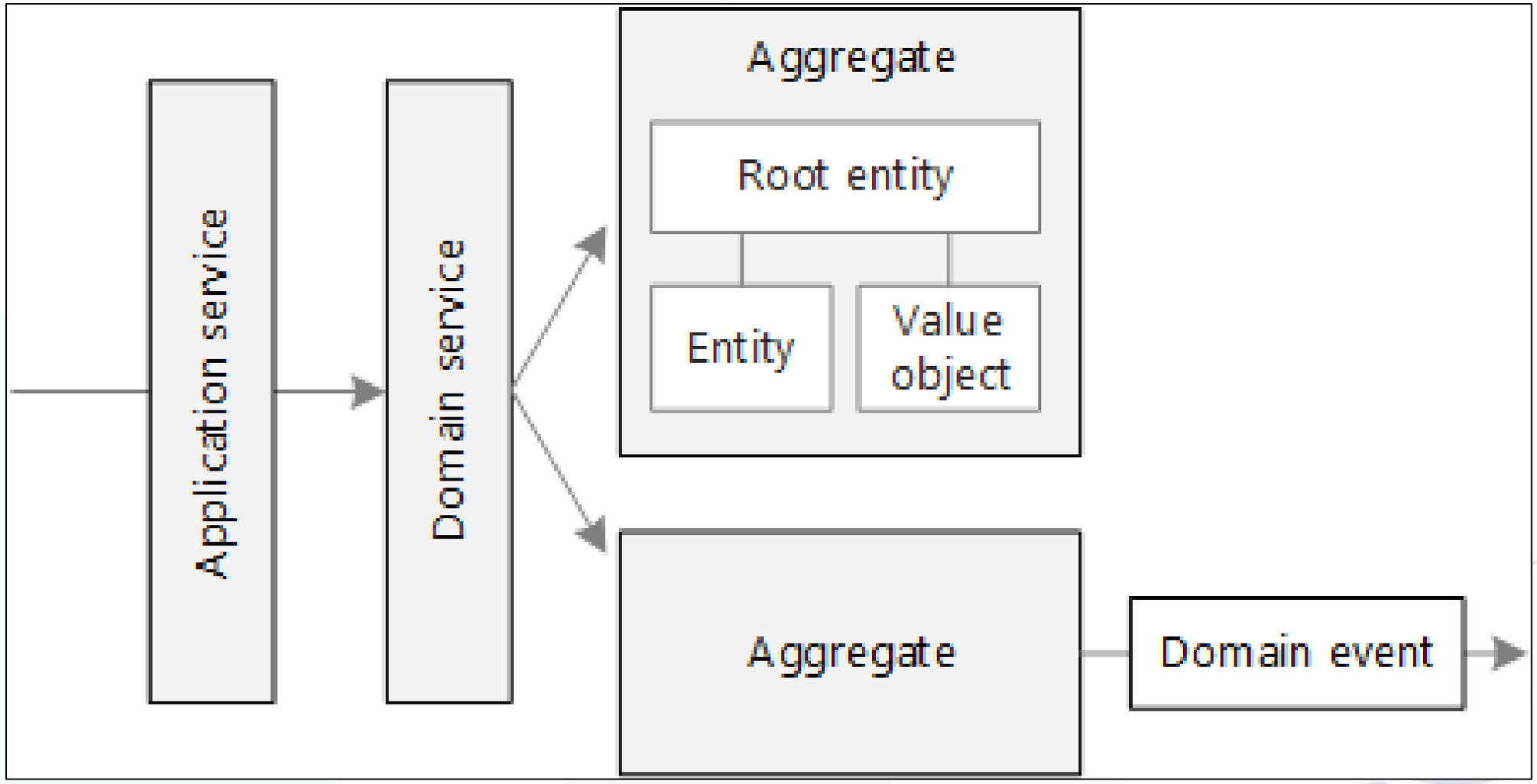
# Ejemplo uso DDD en el diseño de Microservicio - Definiendo contextos limitados




https://learn.microsoft.com/es-es/azure/architecture/microservices/model/domain-analysis

- ✓ *Un **contexto delimitado** es simplemente el límite dentro de un dominio donde se aplica un modelo de dominio en particular*
- ✓ *En el libro Domain Driven Design de Eric Evans, se describen varios **patrones** para mantener la integridad de un modelo de dominio cuando interactúa con otro contexto delimitado.*
- ✓ *Uno de los principios fundamentales de los microservicios es que los servicios se comunican a través de API bien definidas. Este método se corresponde con dos patrones que Evans llama **Open Host Service** (servicio de host abierto) y **Published Language** (lenguaje publicado).*

# Ejemplo uso DDD en el diseño de Microservicio - Definir entidades agregado, servicios



# Patrones de diseño táctico - Entity

```java
@Data
@Builder
@ToString
@EqualsAndHashCode(of = {"identityNumber"})
@AllArgsConstructor
@NoArgsConstructor
@Document(value = "customers")
public class Customer {

    @Id
    private String id;

    @NotNull
    @Indexed(unique = true)
    private String identityNumber;

    @NotNull
    private String name;

    @NotNull
    private String lastName;

    @NotNull
    private String businessName;

    @NotNull
    @Indexed(unique = true)
    private String email;

    @NotNull
    @Indexed(unique = true)
    private String phoneNumber;
```

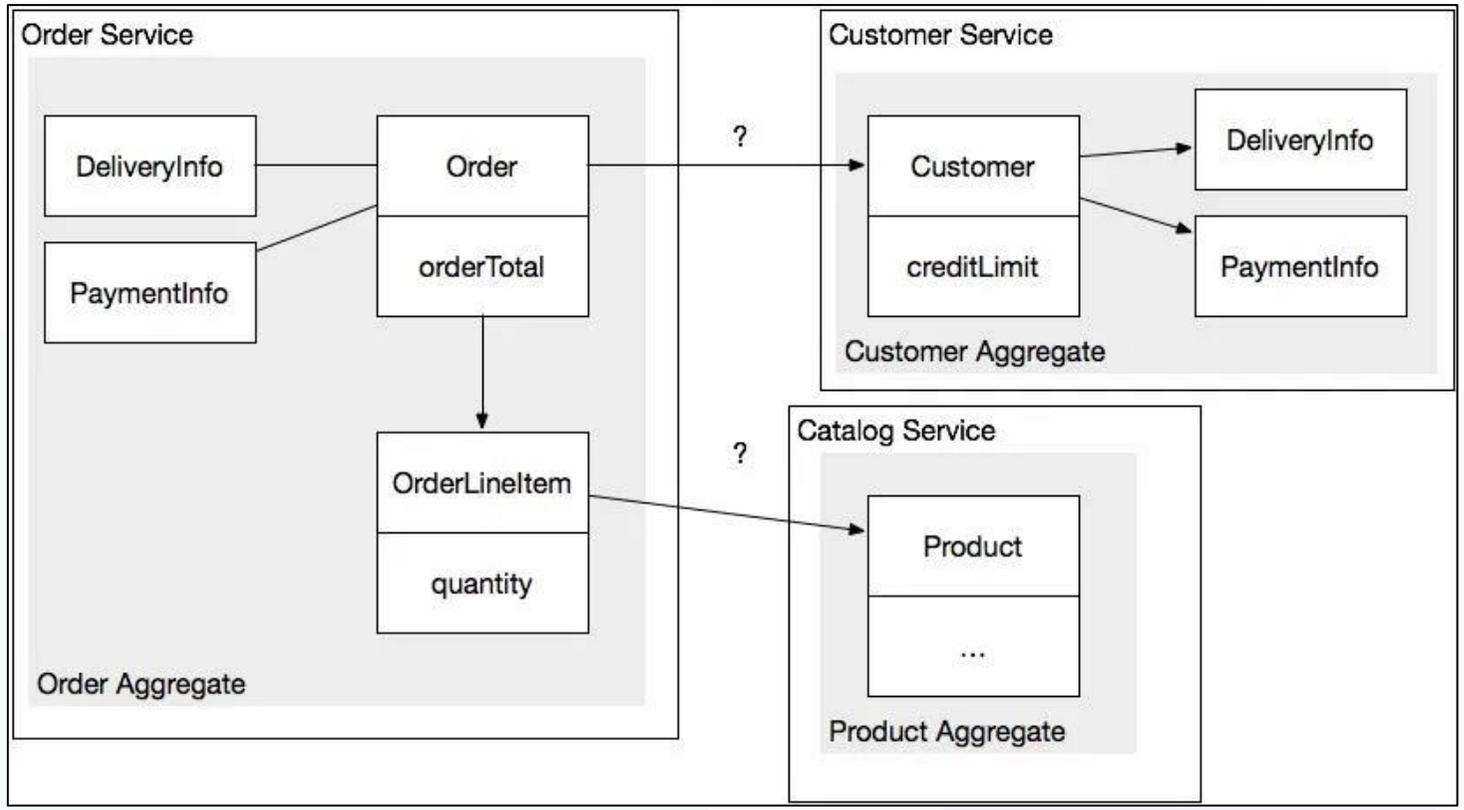
- ✓ Una **entidad** es un objeto con una **identidad única que persiste en el tiempo**. Por ejemplo, en una aplicación bancaria, las cuentas y los clientes serían entidades.
- ✓ Una entidad tiene un **identificador único en el sistema**, que se puede usar para buscar la entidad o para recuperarla.
- ✓ Una identidad puede abarcar **varios contextos** delimitados y puede **durar más que la aplicación**.
- ✓ Los atributos de una entidad pueden **cambiar con el tiempo.**
- ✓ Una entidad puede contener **referencias** a otras entidades.

# Patrones de diseño táctico - Objetos de valor

```
public enum CustomerTypeEnum {
    PERSONNEL("PERSONNEL"), BUSINESS("BUSINESS");

    private String value;

    CustomerTypeEnum(String value) {
        this.value = value;
    }

    public String getValue() {
        return value;
    }
}
```

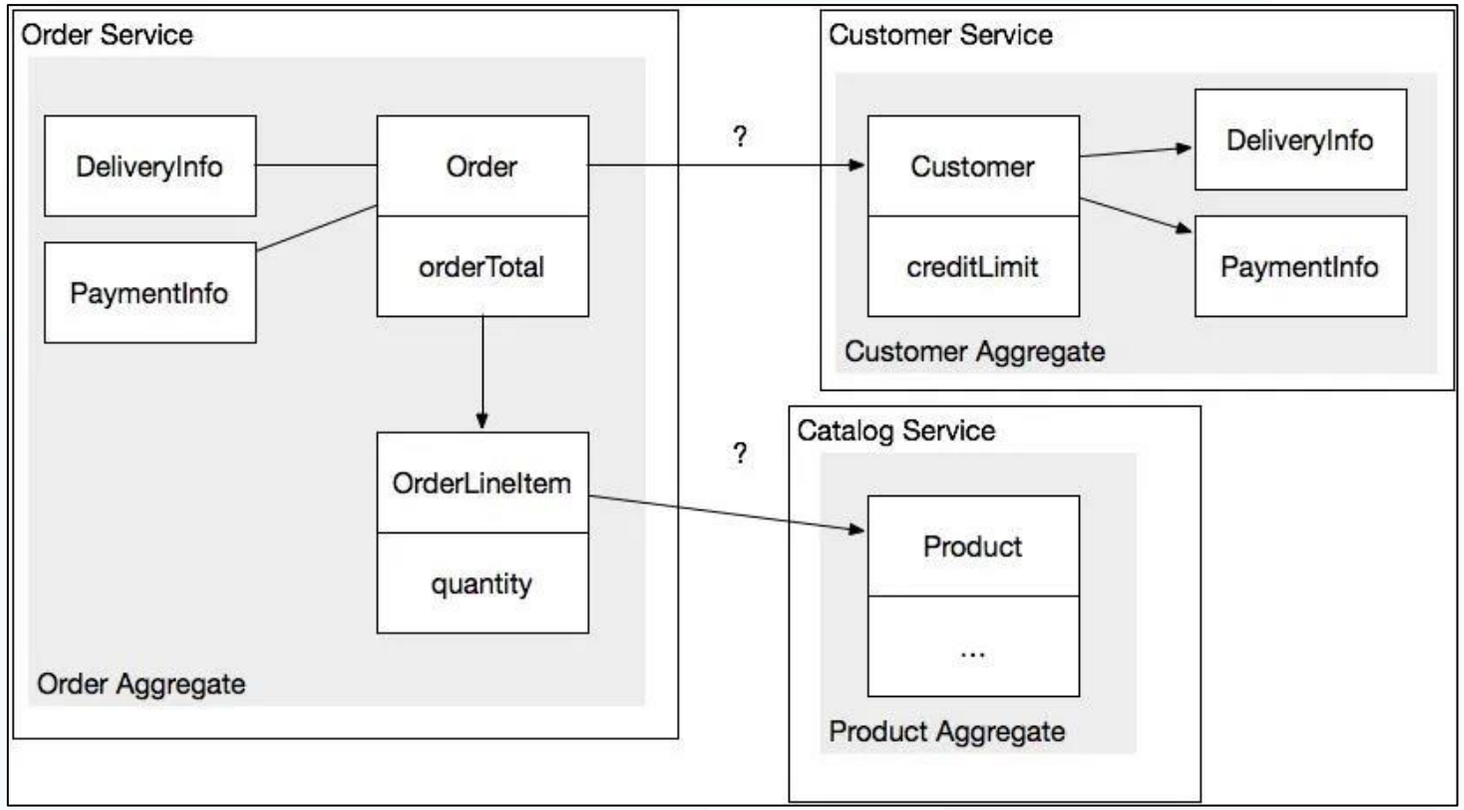
- ✓ Un objeto de valor **no tiene identidad**. Se define únicamente mediante los valores de sus atributos.
- ✓ Los objetos de valor también son inmutables. Para actualizar un objeto de valor, siempre hay que crear una nueva instancia que reemplace a la anterior.
- ✓ Los objetos de valor pueden tener métodos que encapsulen la lógica del dominio, pero esos métodos no deben afectar al estado del objeto. Ejemplos típicos de objetos de valor son los **colores, las fechas y horas, y los valores de divisa.**

# Patrones de diseño táctico - Agregados



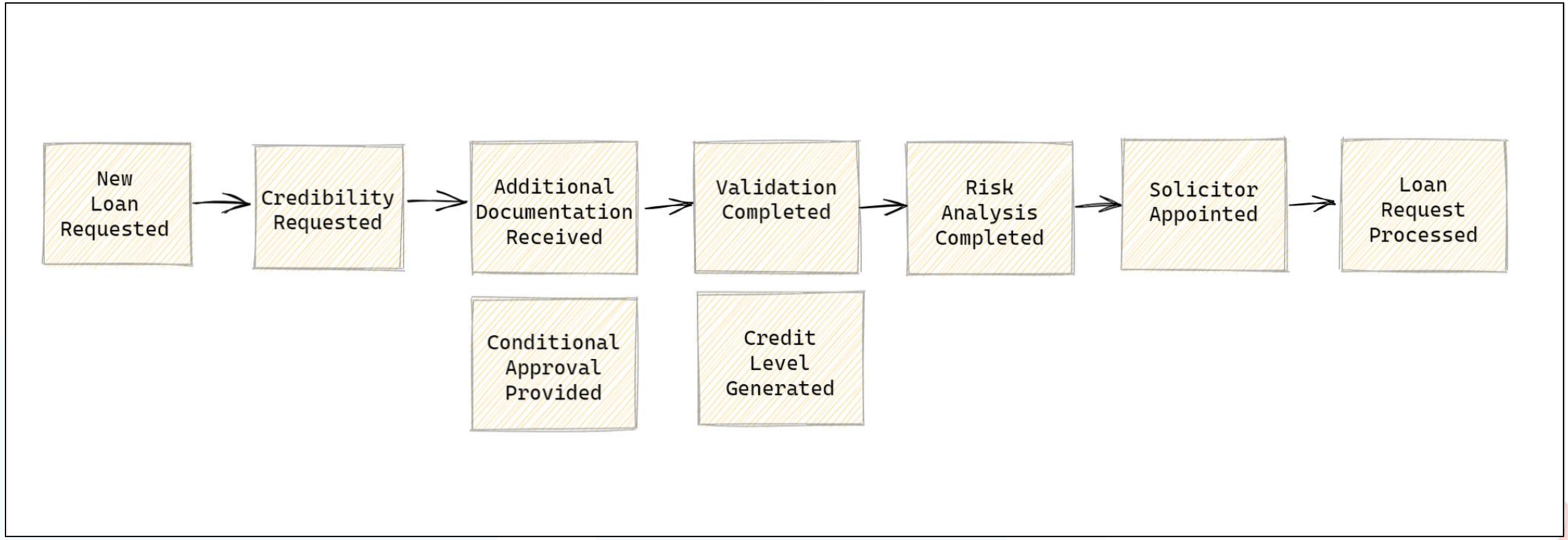

- ✓ Un agregado define un límite de coherencia alrededor de **una o varias entidades**. Una entidad exacta en un agregado es la raíz. La búsqueda se realiza con el **identificador** de la **entidad raíz**. Cualquier otra entidad en el agregado es secundaria de la raíz y se hace referencia a ella siguiendo punteros desde esta.

# Patrones de diseño táctico - Servicios de aplicación y de dominio



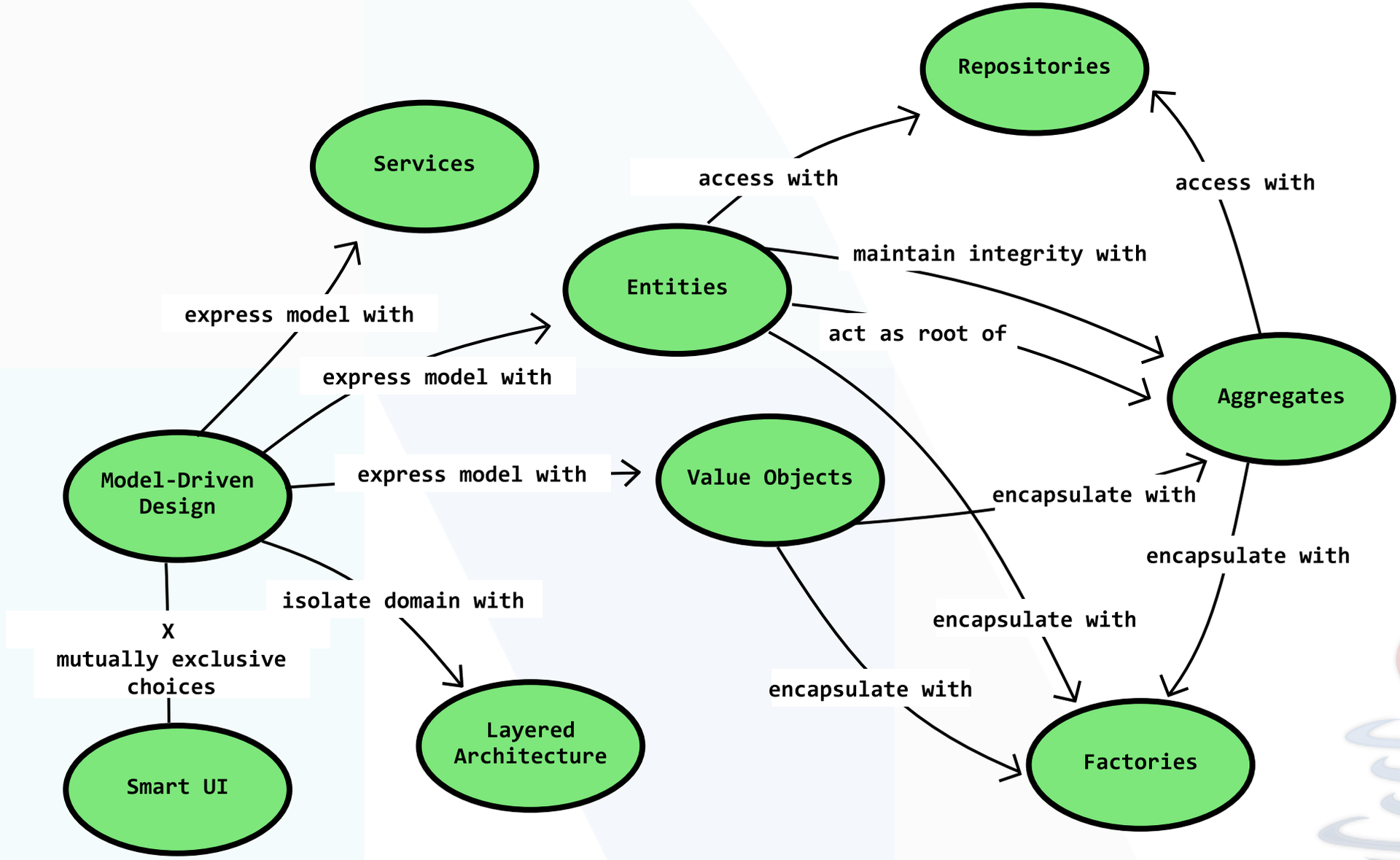

- ✓ En la terminología del diseño basado en dominios, un **servicio** es un objeto que implementa alguna **lógica** sin mantener ningún estado.
- ✓ Evans distingue entre **servicios de dominio**, que encapsulan la **lógica del dominio**, **y servicios de aplicación**, que proporcionan la **funcionalidad técnica**, como la autenticación del usuario o el envío de un mensaje SMS.

# Patrones de diseño táctico - Eventos de dominio

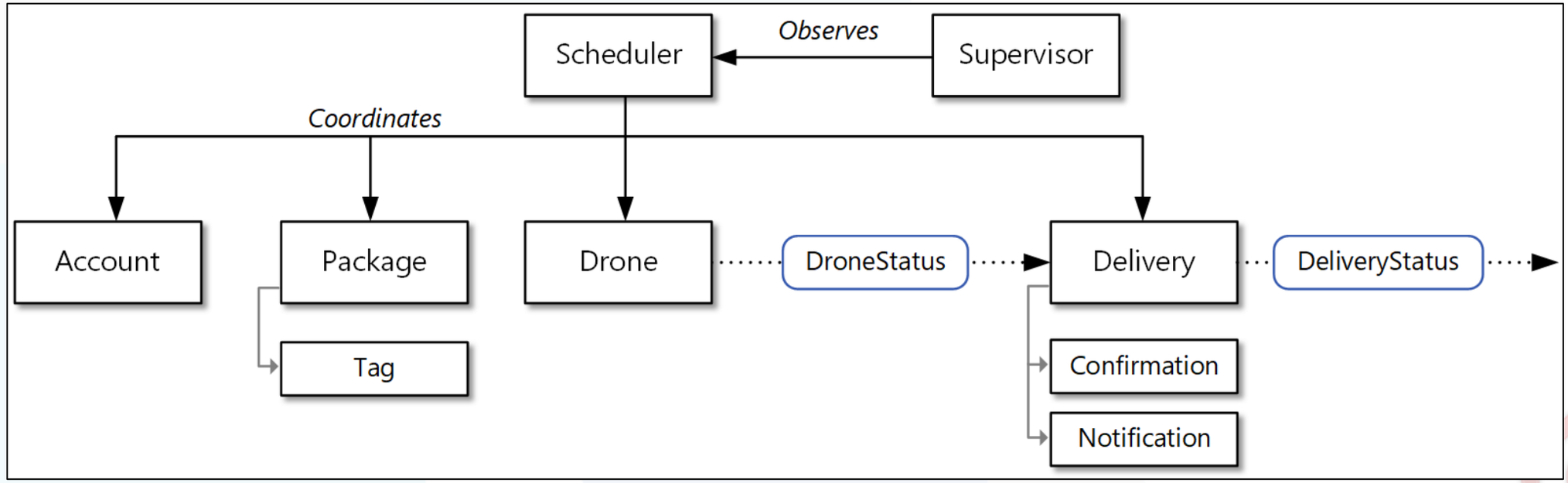



- ✓ Los eventos de dominio se pueden utilizar para **notificar** a otras partes del sistema cuando **sucede algo**. .
- ✓ Como sugiere su nombre, los **eventos de dominio** deben significar algo dentro del dominio. Por ejemplo, "se inserta un registro en una tabla" no es un evento de dominio. "Se canceló una entrega" es un evento de dominio.

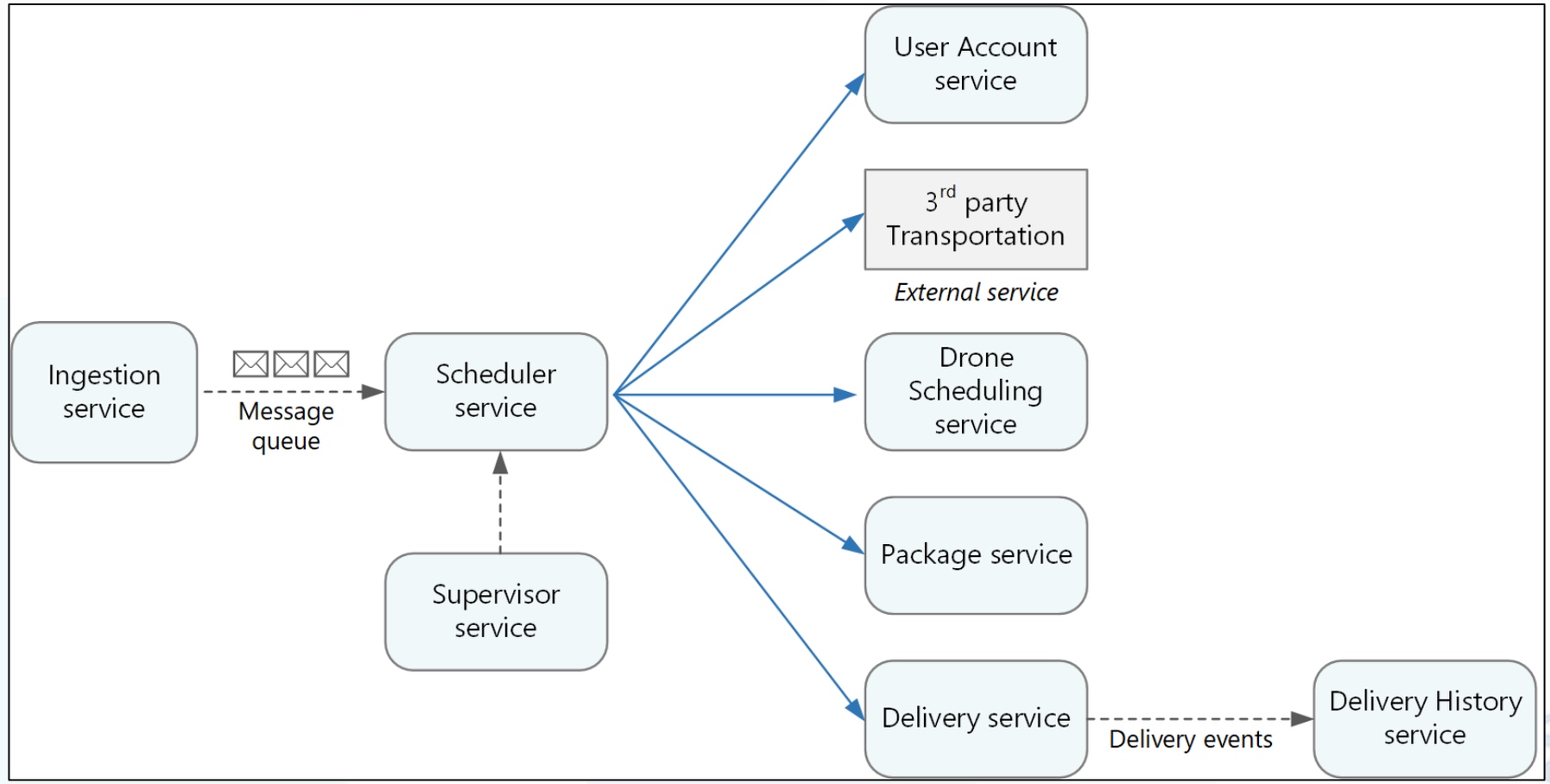
# Patrones de diseño táctico – Vista General



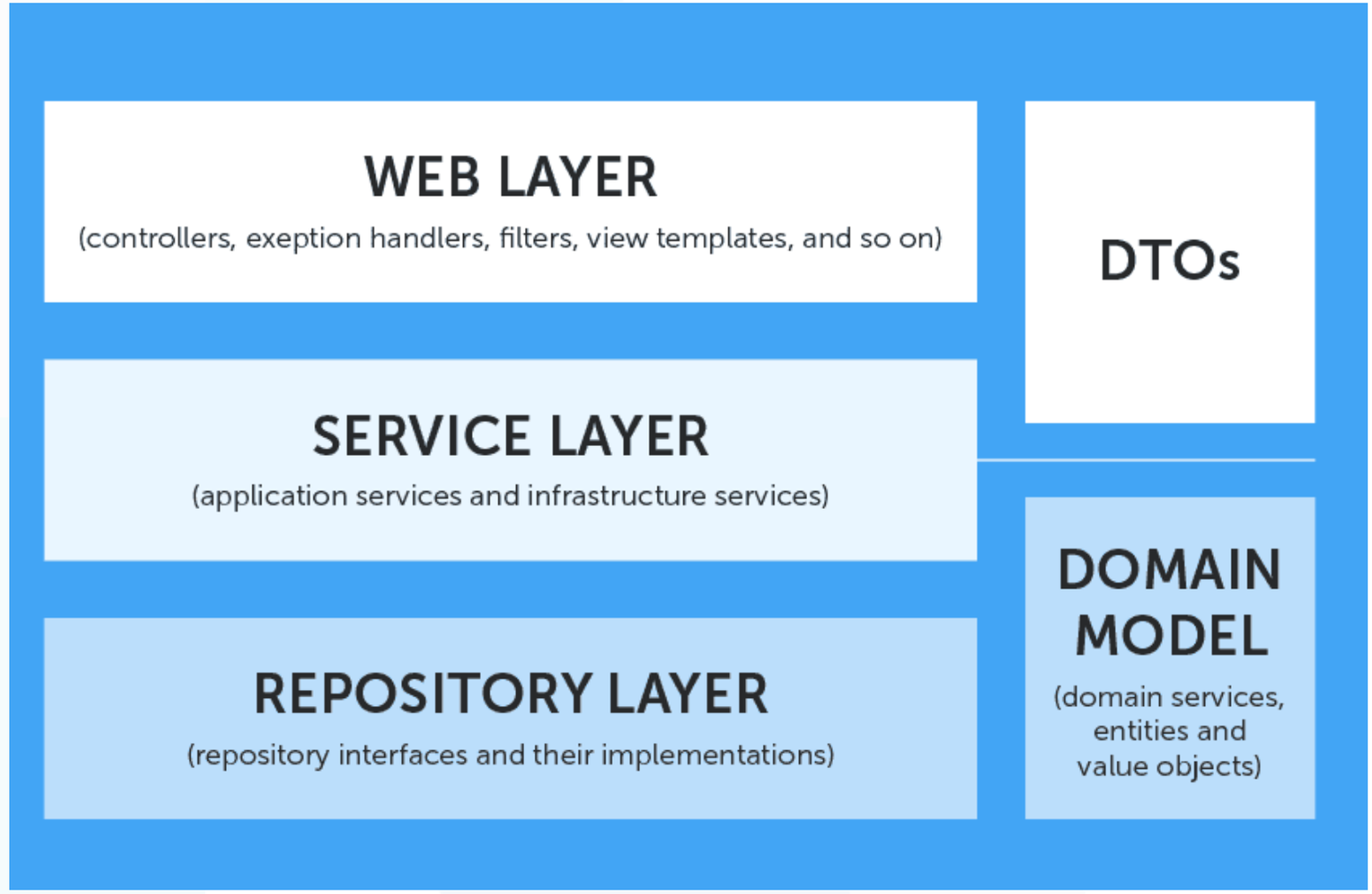
# Ejemplo uso DDD en el diseño de Microservicio - Definir entidades agregado, servicios



# Ejemplo uso DDD en el diseño de Microservicio - Identificar Microservicios

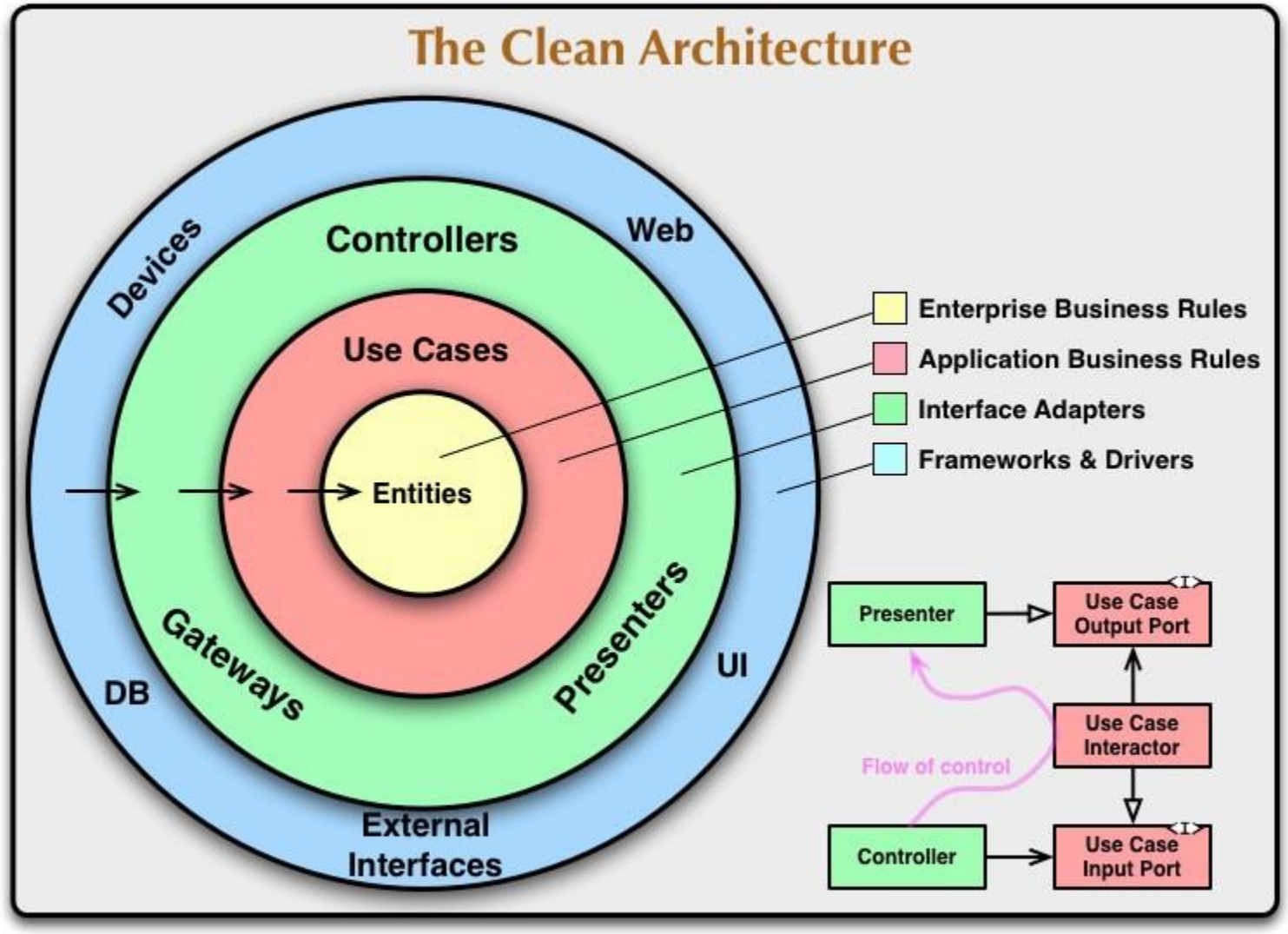


# 3-tier Web Applications Architecture y DDD




https://www.codenomads.nl/soc-spring-boot-and-microservices/

# Clean Architecture y DDD

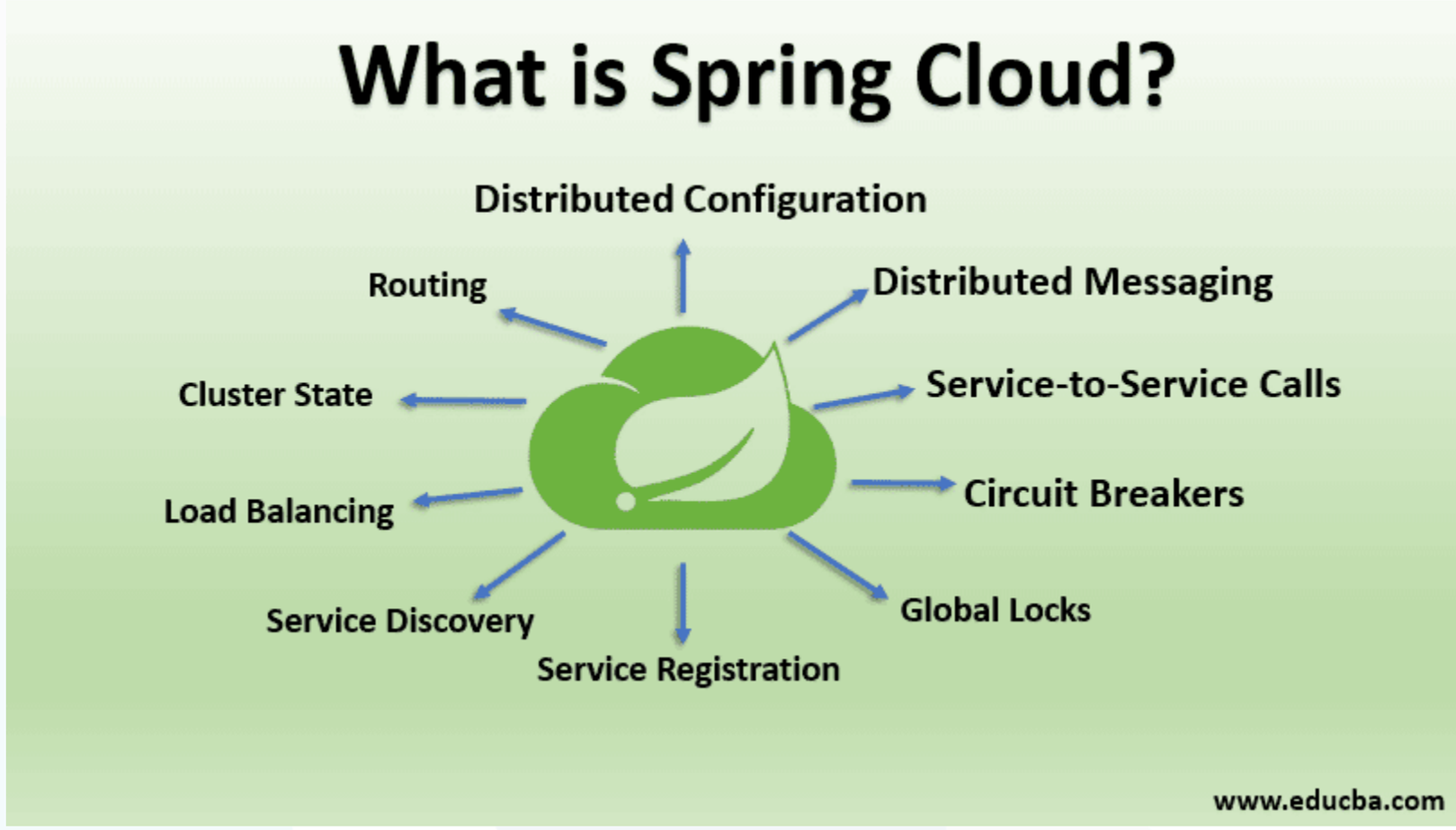



https://github.com/carlphilipp/clean-architecture-example

NTT DATA

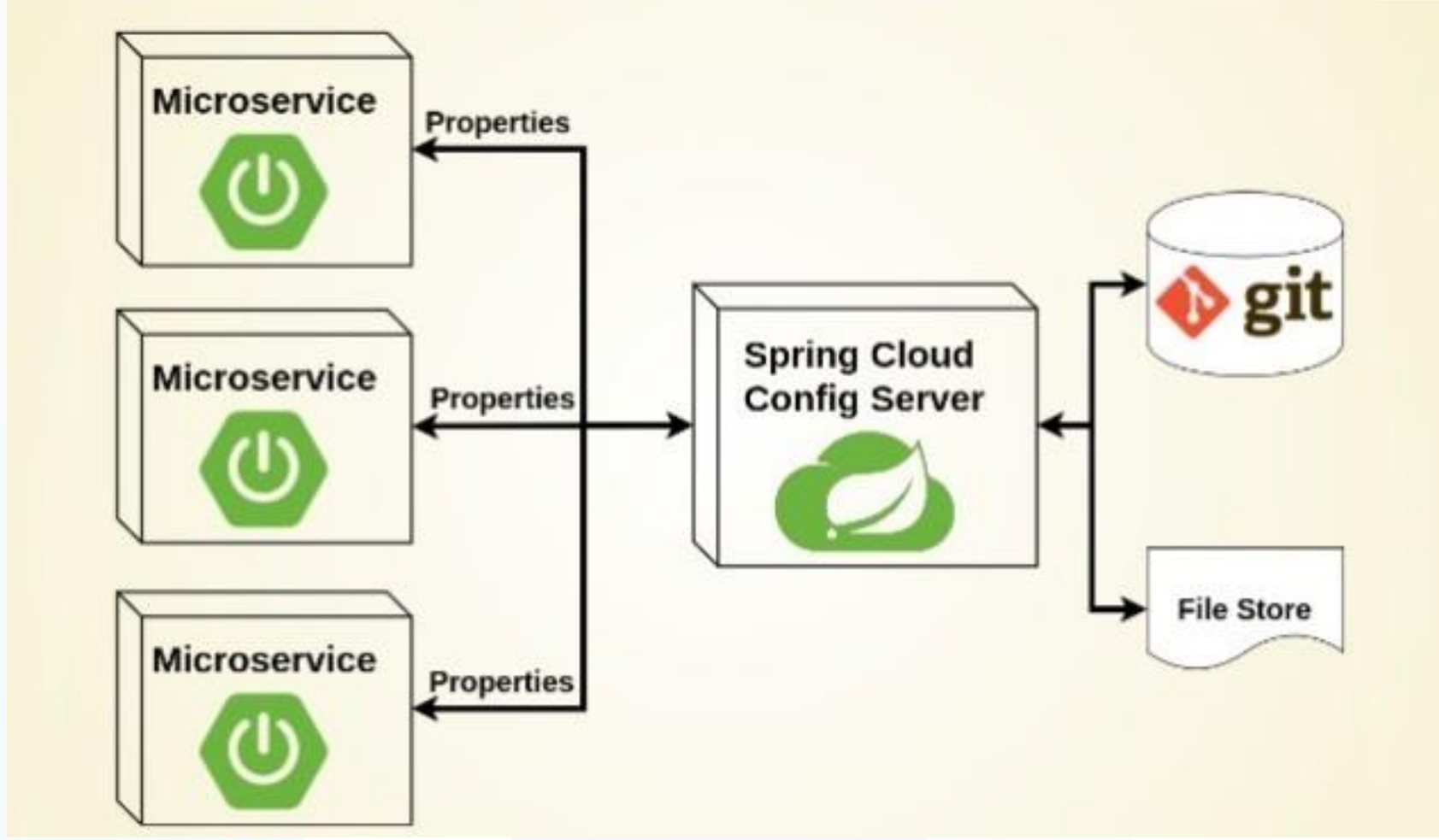
# Stack de Spring Cloud: Config Server

# Spring Cloud




https://spring.io/projects/spring-cloud

# Spring Cloud Config




https://spring.io/projects/spring-cloud

NTT DATA

# Gracias

aangulom@emeal.nttdata.com