

Arquitectura e Implementación de Microservicios con Spring Cloud y AWS



Instructor: Javier Dextre Pittman

- Arquitecto de software con más de 15 años de experiencia en empresas de prestigio, en el área de tecnologías de la información y desarrollo de Software, responsable del diseño, análisis, desarrollo, implementación y consultoría de sistemas de software. Ampla experiencia en desarrollo y arquitectura de microservicios para las empresas de telecomunicaciones (Telco).
- Certificación en Oracle SOA Suite y especialista en arquitectura de microservicios.

Linkedin: <https://www.linkedin.com/in/javier-dextre-pittman-3484abbb/>

Email: javidep@gmail.com

944433715

Objetivos de aprendizaje

- conocer los fundamentos de microservicios
 - diseñar y modelar aplicaciones usando la arquitectura de microservicios
- implementar microservicios usando:
 - patrones de diseño de microservicios.
 - herramientas como spring cloud y netflix oss framework
 - estrategias de testing para asegurar la calidad de los microservicios
 - monitoreo para mantener la disponibilidad del los microservicios.
- desplegar una arquitectura de microservicios a la nube AWS

Temática del curso

- 1. fundamentos de microservicios
- 2. diseño de microservicios
- 3-12. desarrollo de microservicios
 - spring cloud
 - docker/docker compose
 - netflix oss
 - hystrix
 - eureka
 - feign
 - zuul
 - ribbon
 - seguridad
 - oauth2
 - jwt
 - Testing
- Pruebas unitarias
- integración
- comunicación asíncrona
 - spring cloud stream
 - apache kafka
- monitoreo
 - openzipkin
 - spring sleuth
- 13. deploy
 - amazon aws

Metodología

- Horario
- Repositorio github
 - <https://github.com/javidep/curso-microservicios>
 - Codigo fuente, laboratorios, etc
- Maquina virtual Ubuntu
- Aplicación para compartir datos
 - <https://datajaw.web.app>

Que son los microservicios?

- Un enfoque de ingeniería basado en **descomponer** aplicaciones en módulos de **función única** , con **interfaces bien definidas**, que son **independientemente** desplegadas y operadas en **equipos pequeños** , responsables del **ciclo de vida entero** del servicio.
- Los Microservicios aceleran la liberación de software **minimizando la comunicación** y coordinación entre personas mientras **reduce el alcance** y riesgo del cambio.

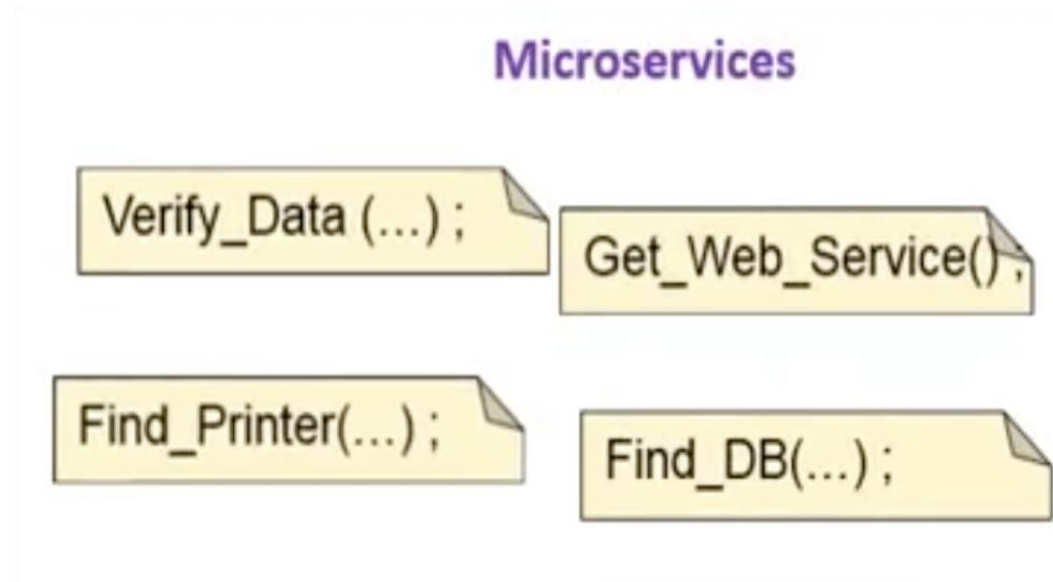
Descomponer

MyApplication

```
Verify_Data (...);  
Get_Web_Service();  
Find_Printer (...);  
Find_DB();
```

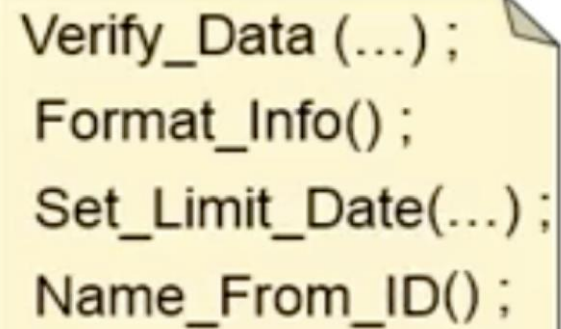
- Todos son Requeridos para completar el trabajo

Descomponer



- Servicios Pequeños individuales
- No dependen entre ellos

Función única

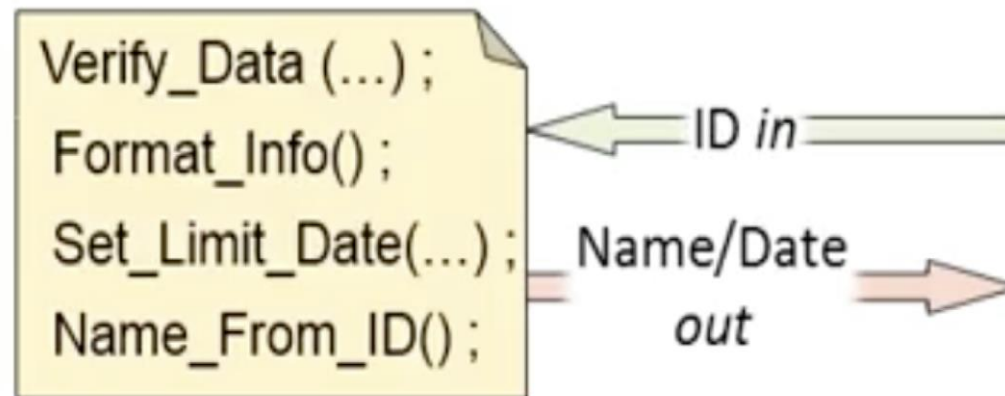


```
Verify_Data (...);  
Format_Info();  
Set_Limit_Date(...);  
Name_From_ID();
```

Conjunto de procesos autocontenidos

Interfaces bien definidas

- Definir data



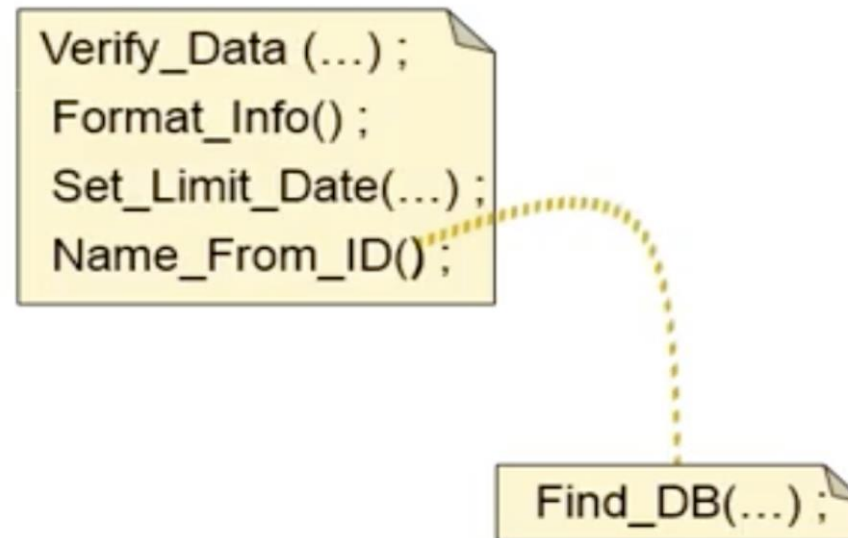
Independientes

- Mundo real



- Tren. Va a A hacia B
- Persona. Trabaja en B
- Servicios independientes

independientes



Tienen que trabajar juntos

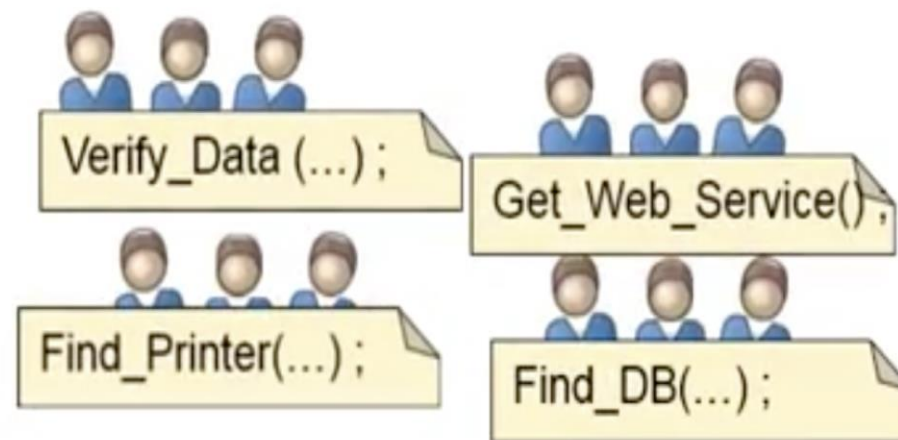
Equipos pequeños

- Coordinan el trabajo
- tradicional

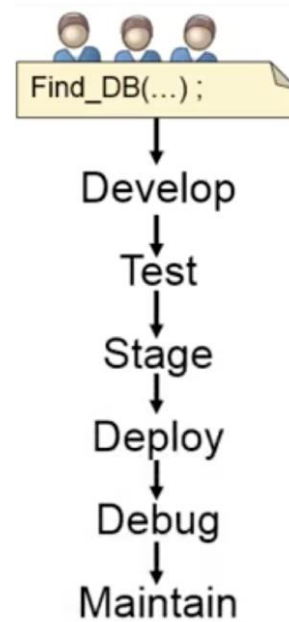


Equipos pequeños

- Servicios pequeños = Equipos pequeños
- Mas eficientes
- Mejor concentracion



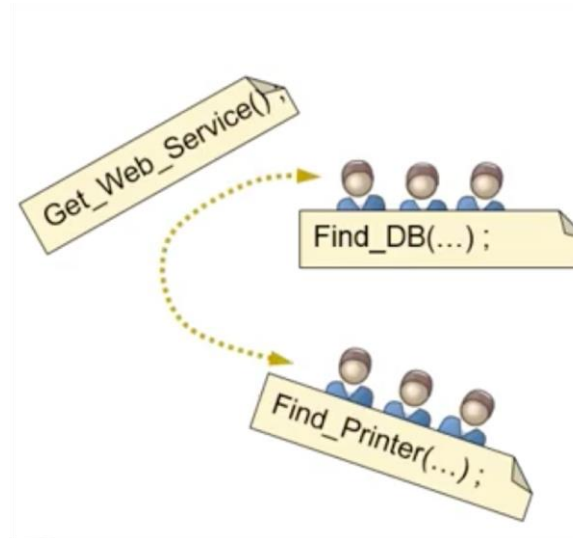
Ciclo de vida entero



- Responsabilidad clara
- Mas eficiente

Minimizando comunicacion

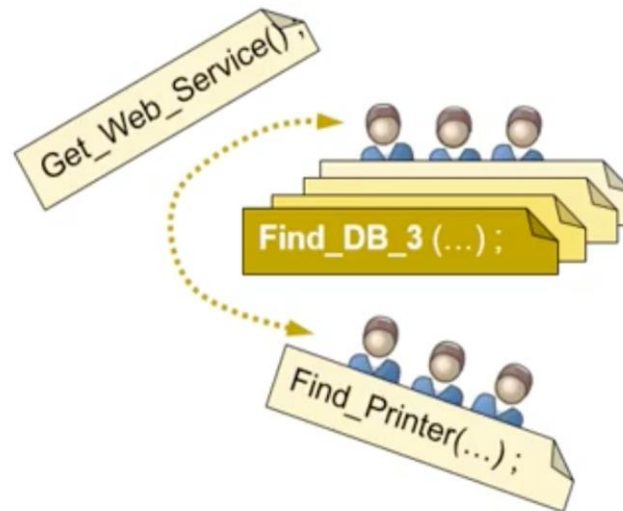
- Microservicios es acerca de la gente



Comunicación interservicios = interfaces

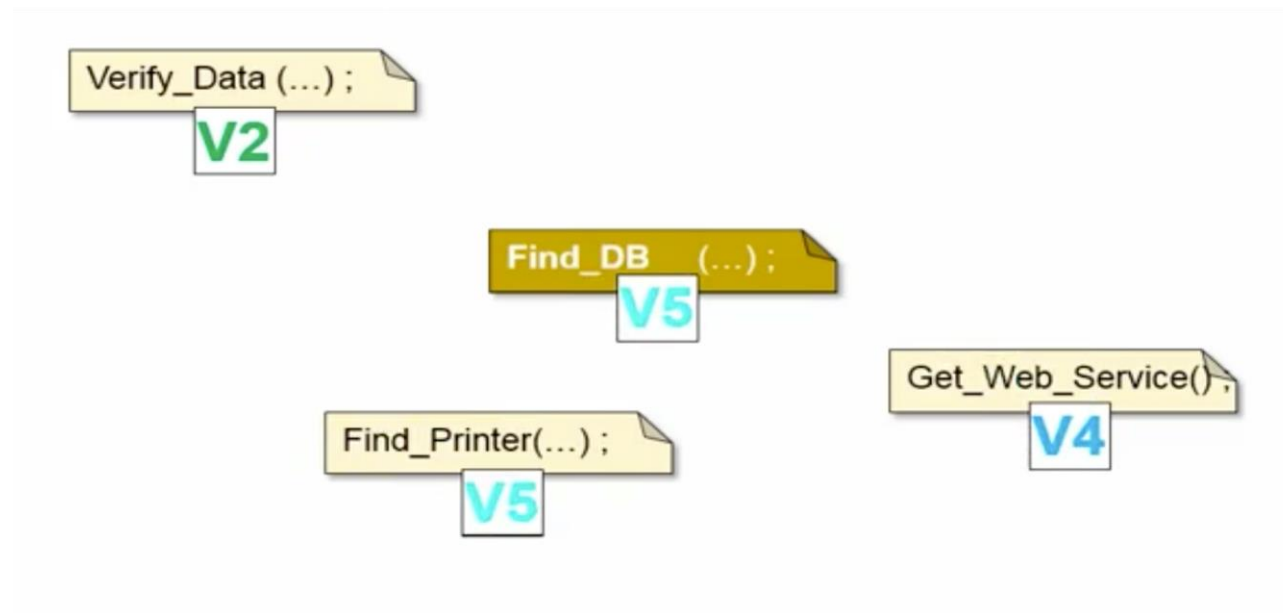
Reduce el alcance del cambio

- Que pasa si modifico algo?



- Alcance dentro del equipo
- Interfaces bien definidas

versionamiento



- Actualizacion individual

Que vino antes?

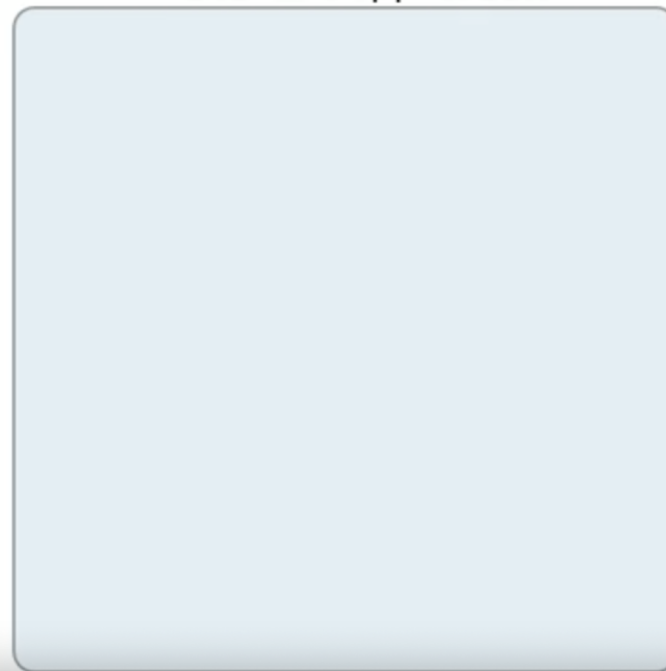
- 1 sola pieza, 1 solo trabajo
 - Ejemplo . Procesador de texto
 - Causa de microservicios

Monolito

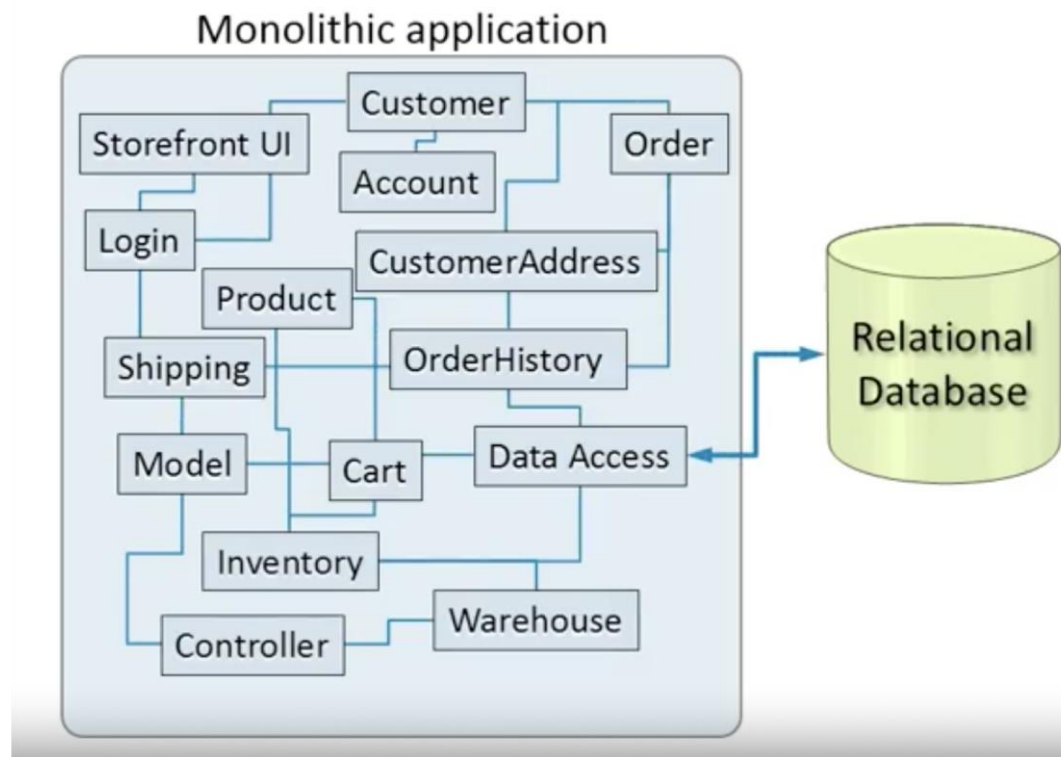


Pesado
Rígido
anticuado

Monolithic application



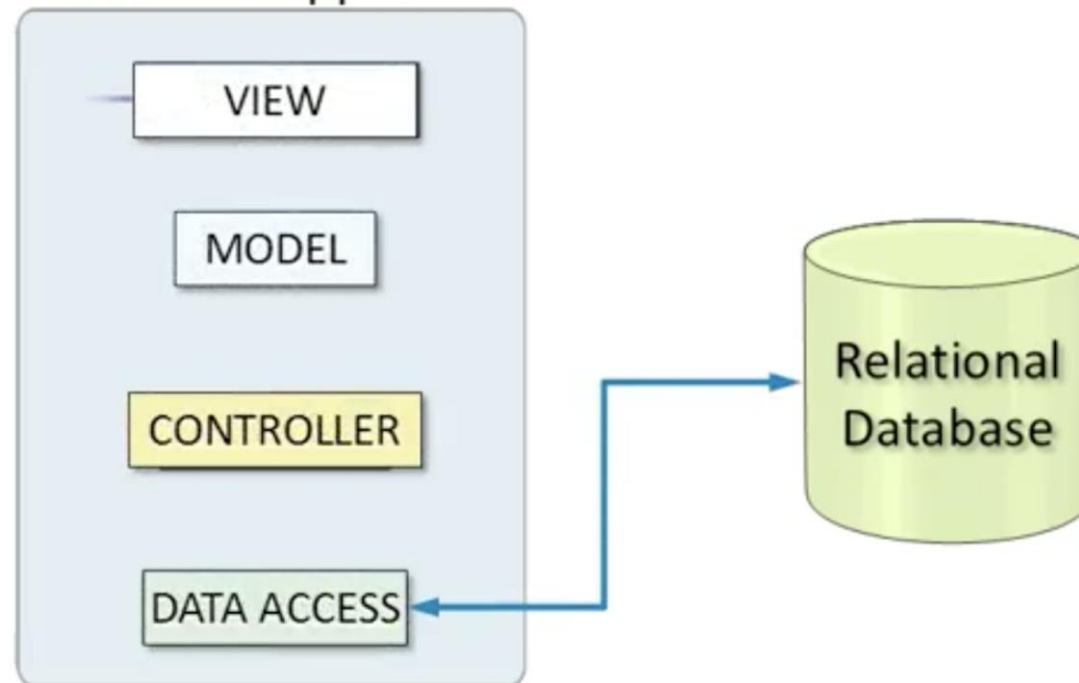
Relational
Database

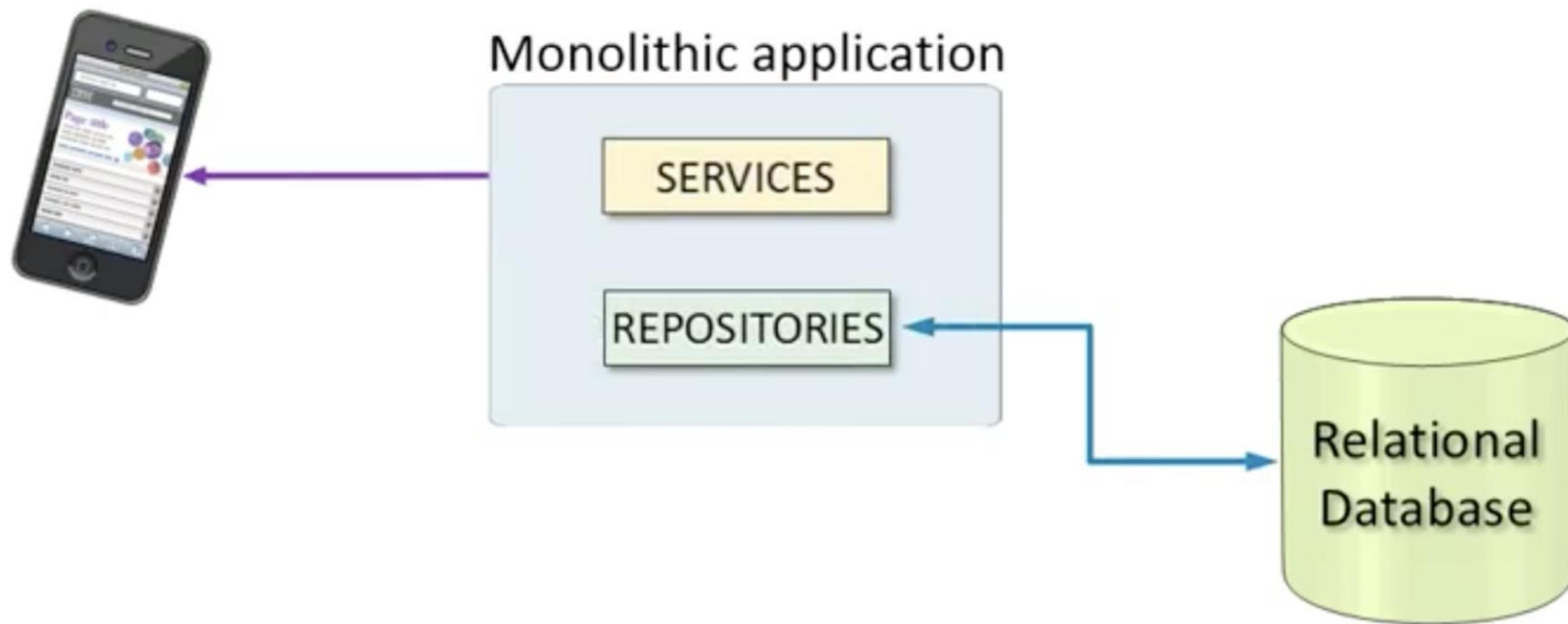


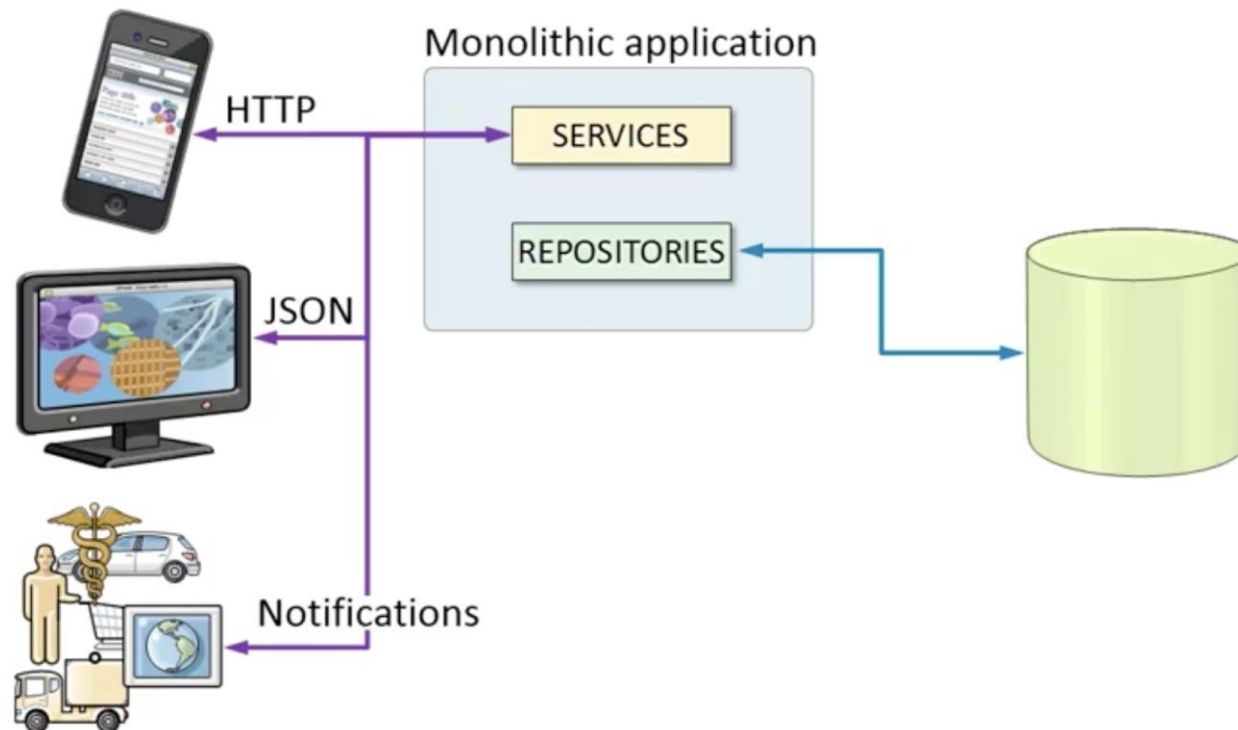
Monolithic application



Monolithic application





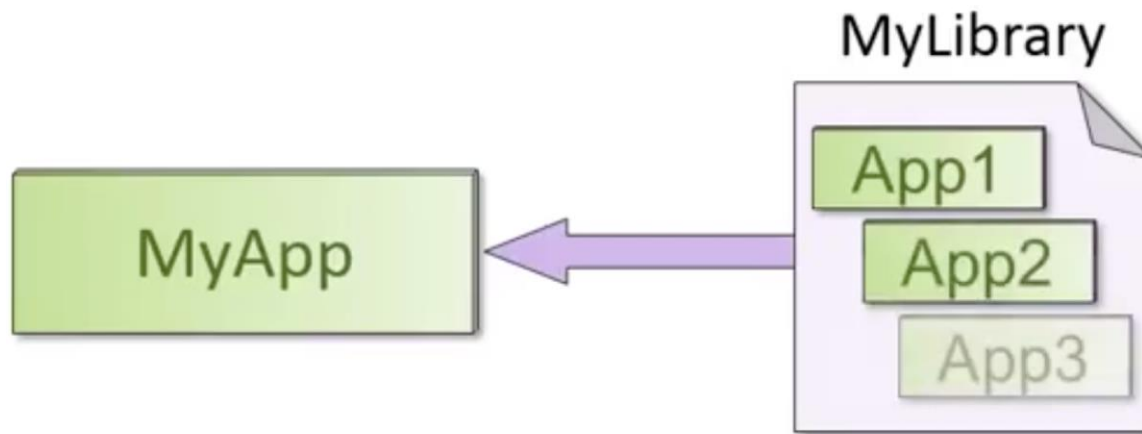


Analogia

- Un monolito es como una casa, una vez construida no puedes cambiarla.
- Si construiste la cocina de un tamaño solo puedes modificarla modificando muchas otras partes de la casa.
- La casa es una cosa maravillosa pero tiene sus limitaciones

Características de Arquitectura de microservicios

Componentes como servicios



Librerías generan acoplamiento

Componentes presentados como servicios

MyApp

App1

App3

App2

cualquiera de ellos puede ser actualizado sin afectar los otros.

Componentes presentados como servicios

- Todos son servicios
- Ventajas:
- Librería
 - mismo lenguaje que la App
 - Usar java JNI
- Servicio
 - Escrito en cualquier lenguaje
 - Solo necesario la interface

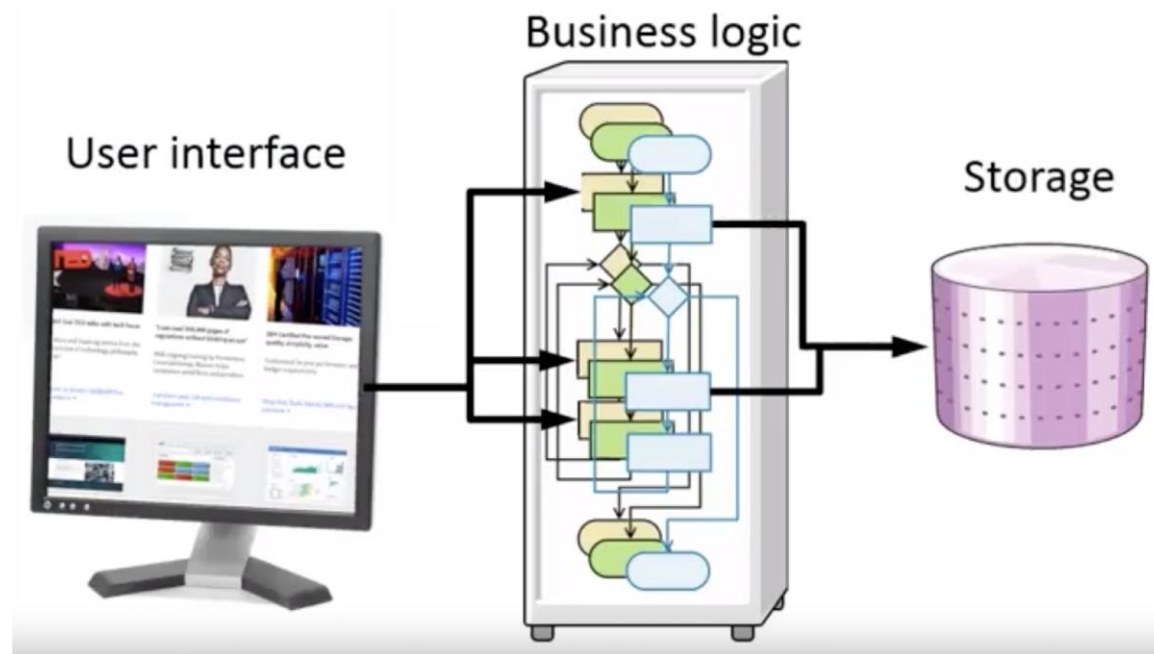
MyServ

Serv1

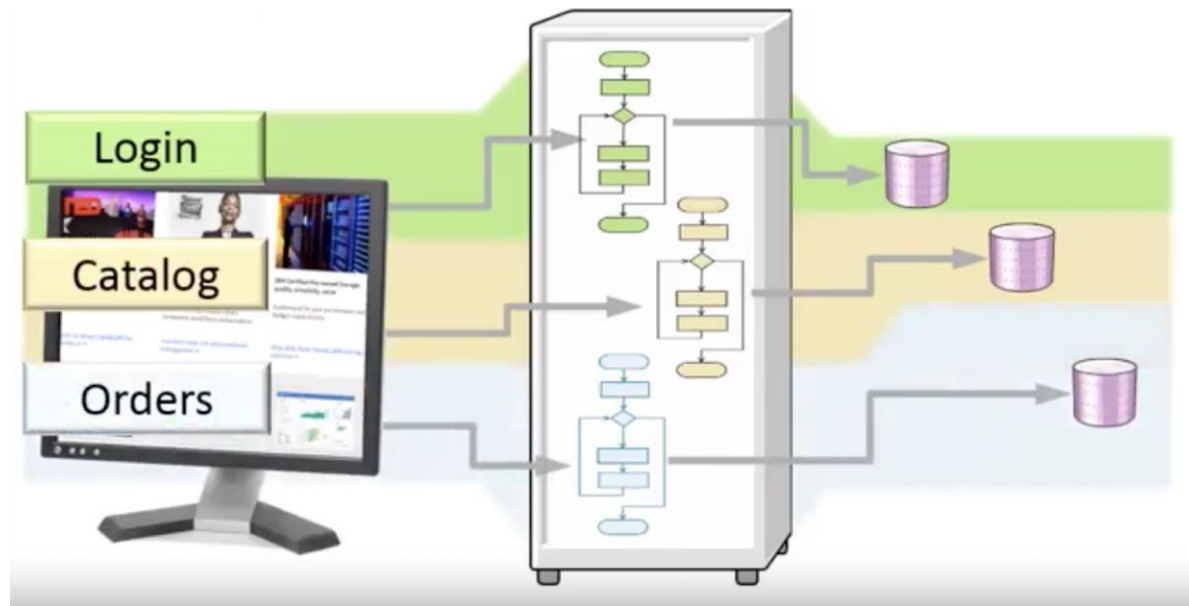
Serv3

Serv2

Diseño dictado por tecnología

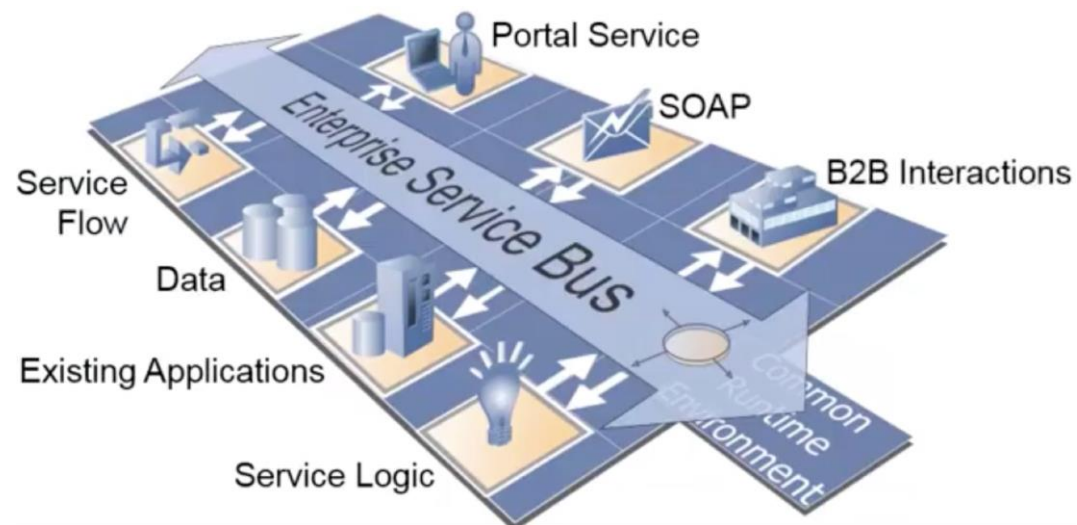


Dictado por el dominio del negocio

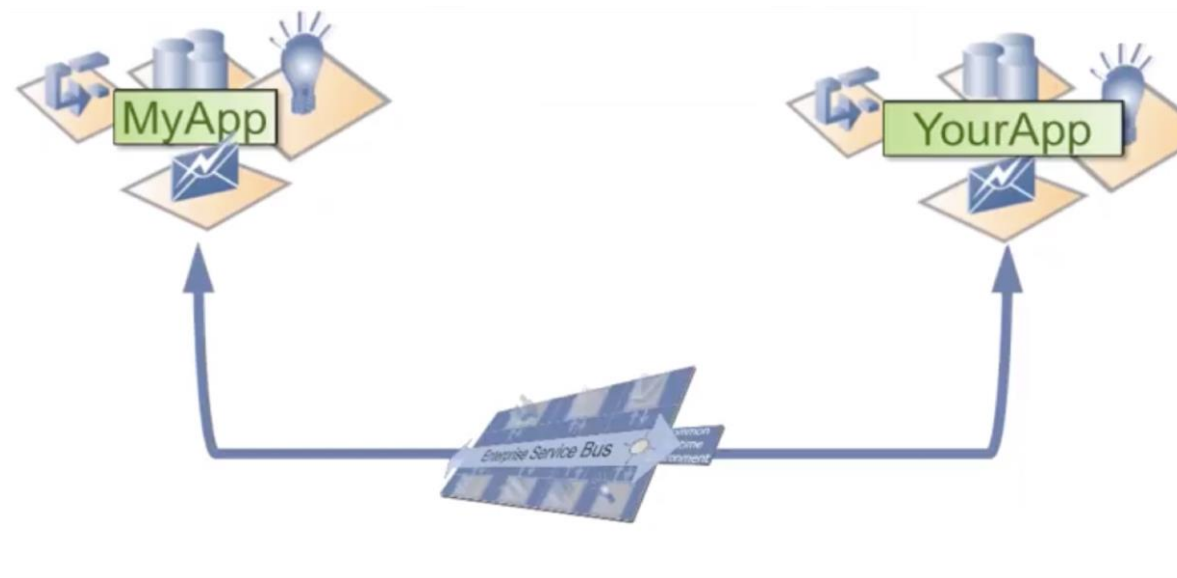


Smart endpoints – dumb pipes

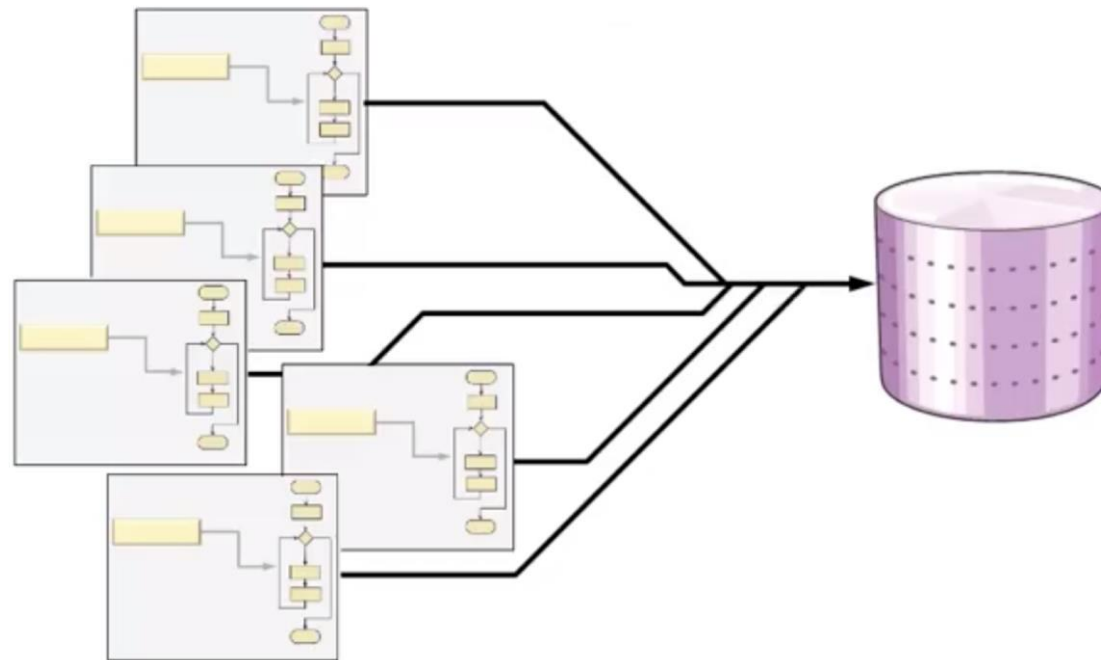
Aplicación tradicional



Smart endpoints – dumb pipes

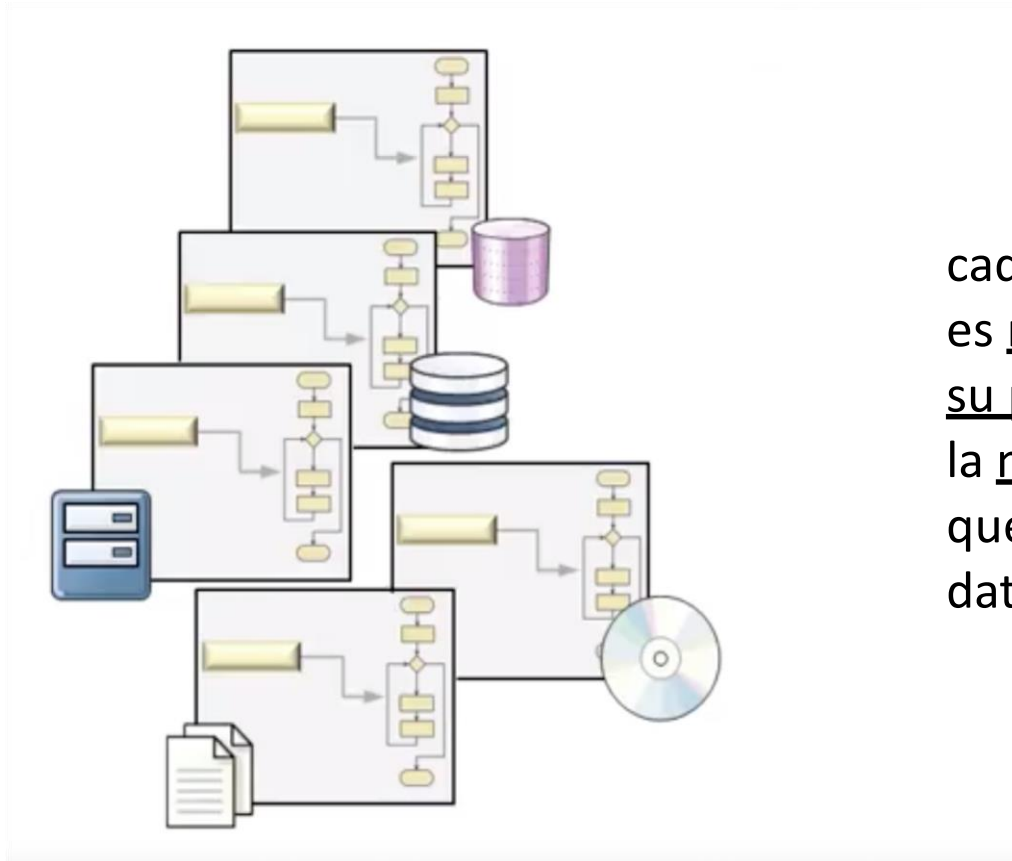


Cada microservicios sabe que hace y como lo hace



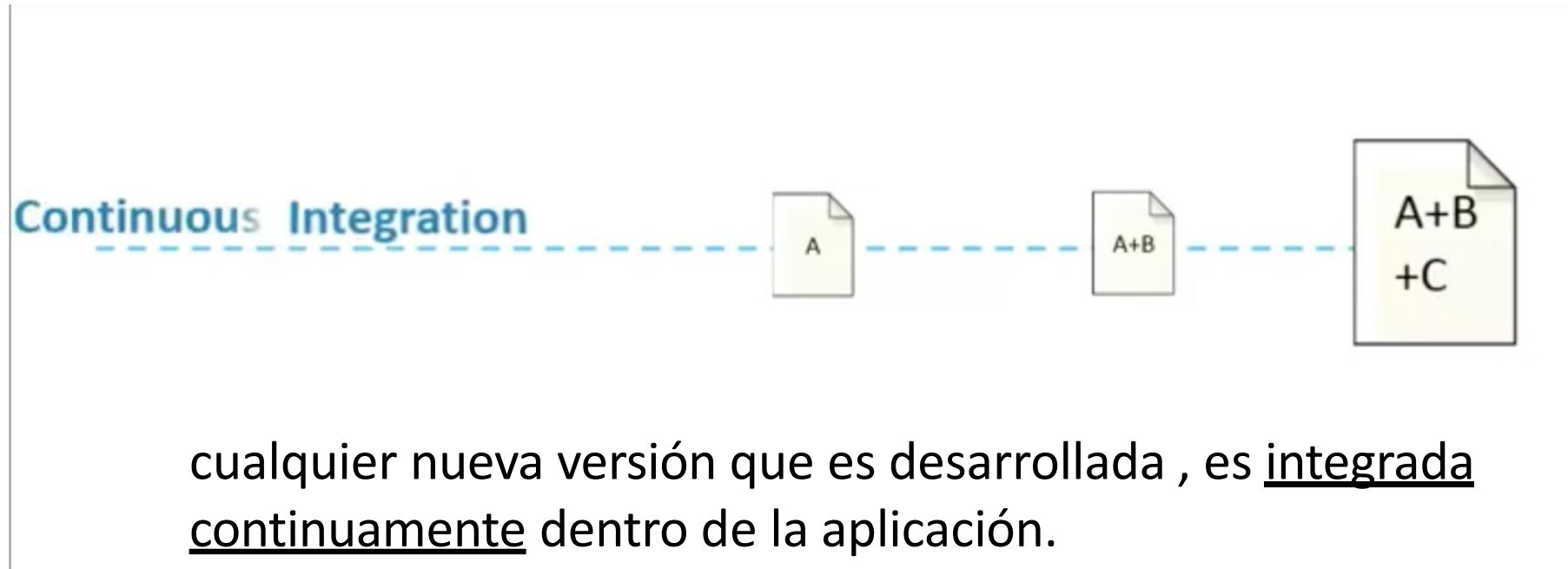
Mismo tipo de base de datos

Data distribuida



cada micro servicio es responsable de su propia data y de la manera en la que persiste esta data.

Integracion Continua



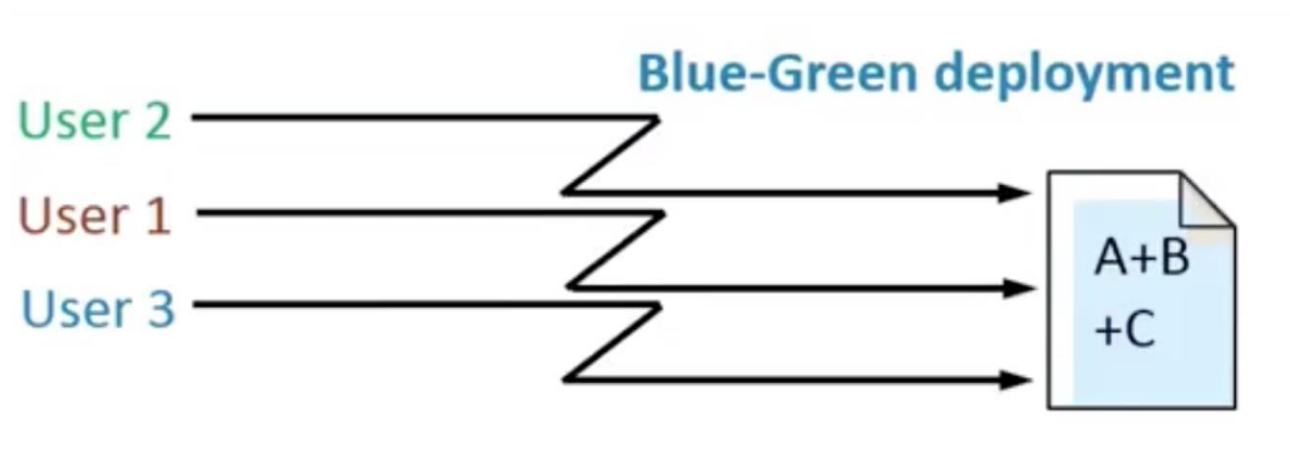
cualquier nueva versión que es desarrollada , es integrada continuamente dentro de la aplicación.

Infraestructura para automatizacion

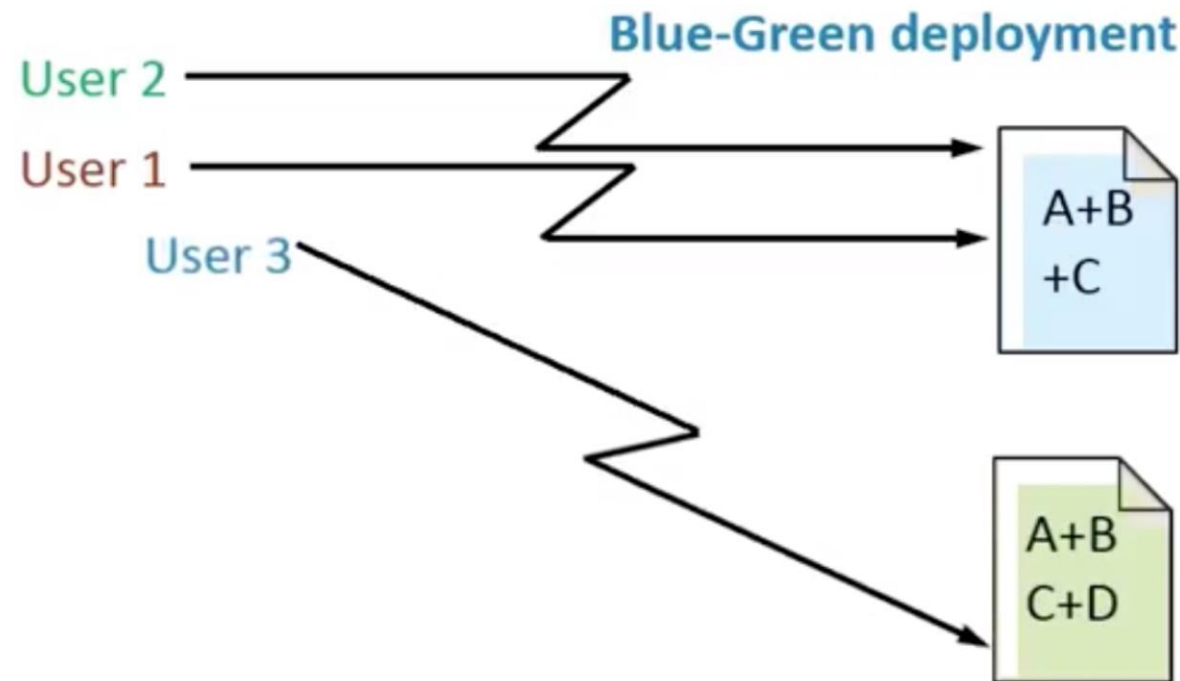
- CI/CD
 - Integración continua y Entrega continua
- Demasiados puntos de integración
- Posibilidad de errores humanos
- Desarrollo asociado a Operaciones
- DEVOPS
- Herramientas: Jenkins, Travis,etc

Infraestructura para automatizacion

Despliegue continuo

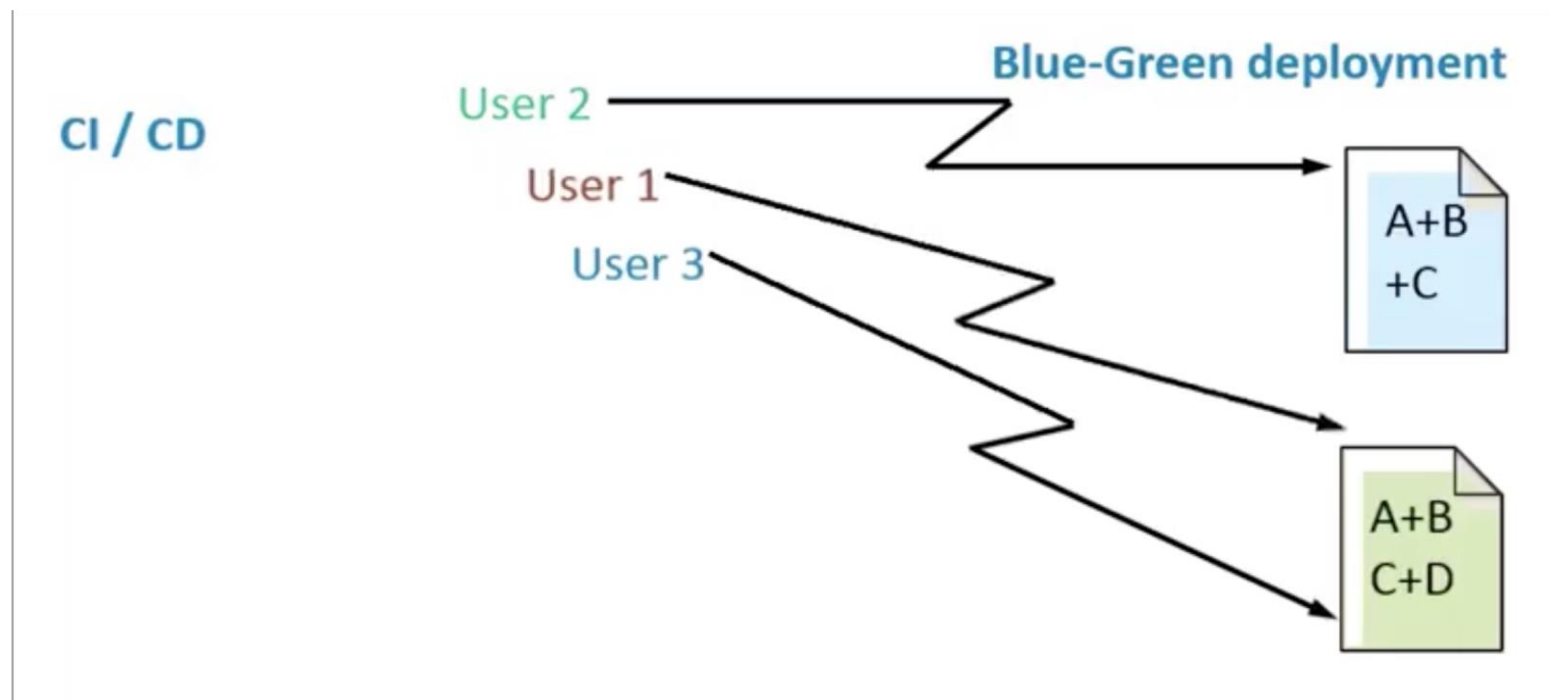


Infraestructura para automatizacion



si hay algún problema en la ejecución, solo afectaría a este usuario.

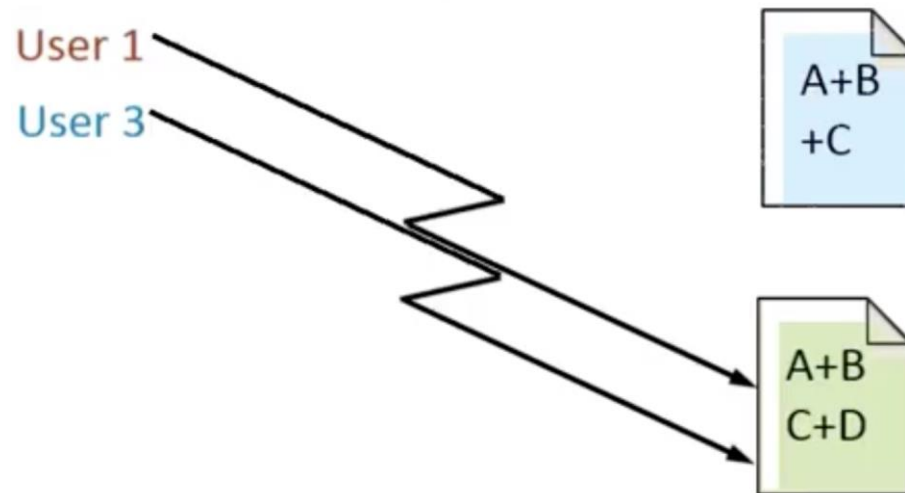
Infraestructura para automatizacion



voy incrementando la cantidad de usuarios que apuntan a la versión green

Infraestructura para automatizacion

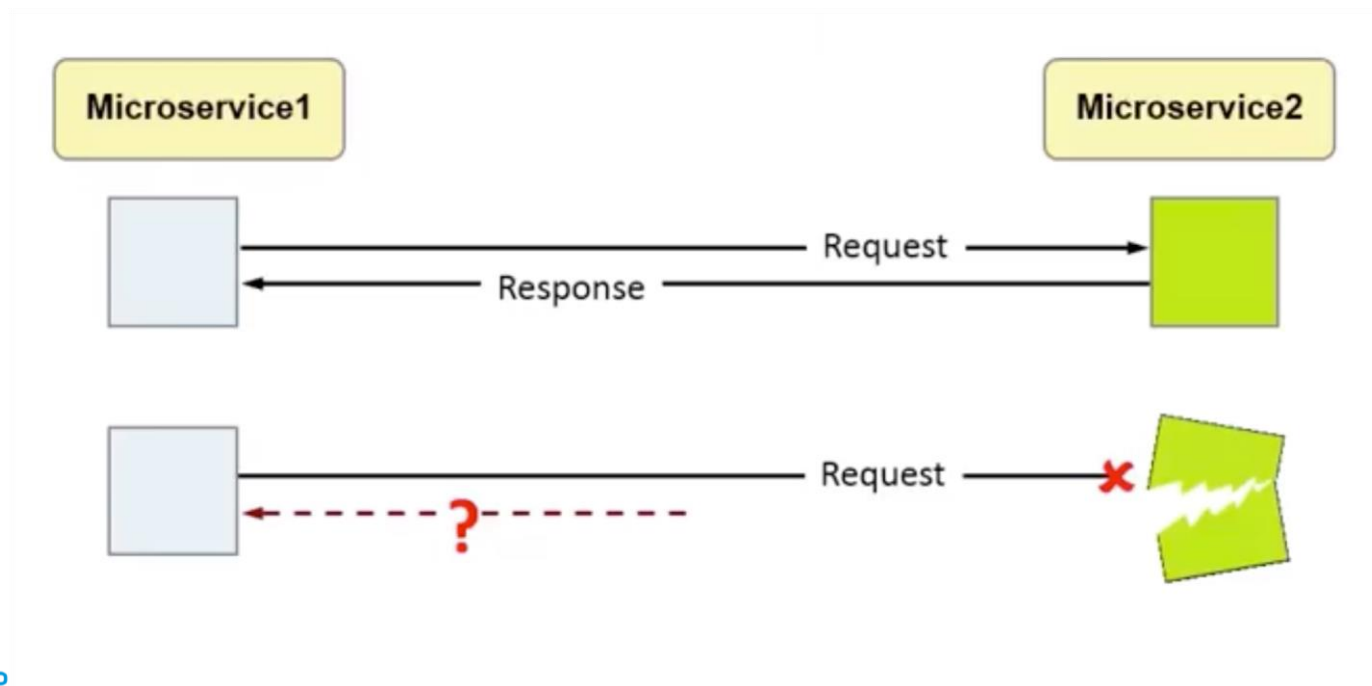
Blue-Green deployment



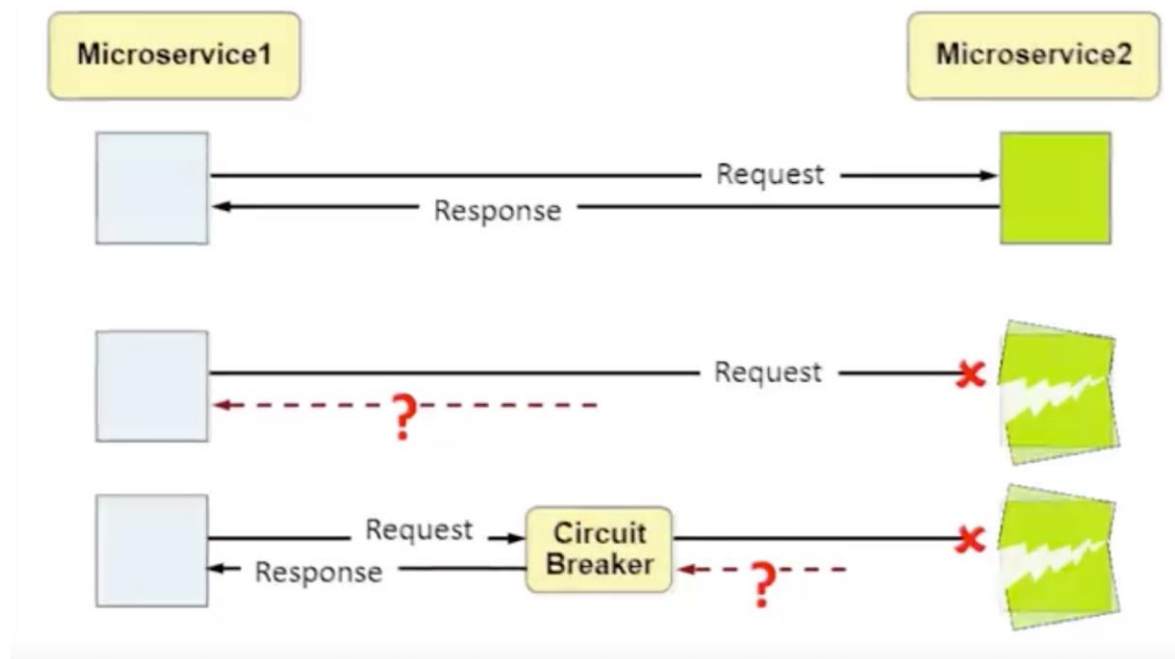
- la versión Green será invocada por cualquier nuevo usuario
- soportado por la mayoría de herramientas de despliegue como :
kubernetes, openshift, aws, docker

Tolerancia a Fallas

- Detectar Donde
- Evitar continuar llamadas



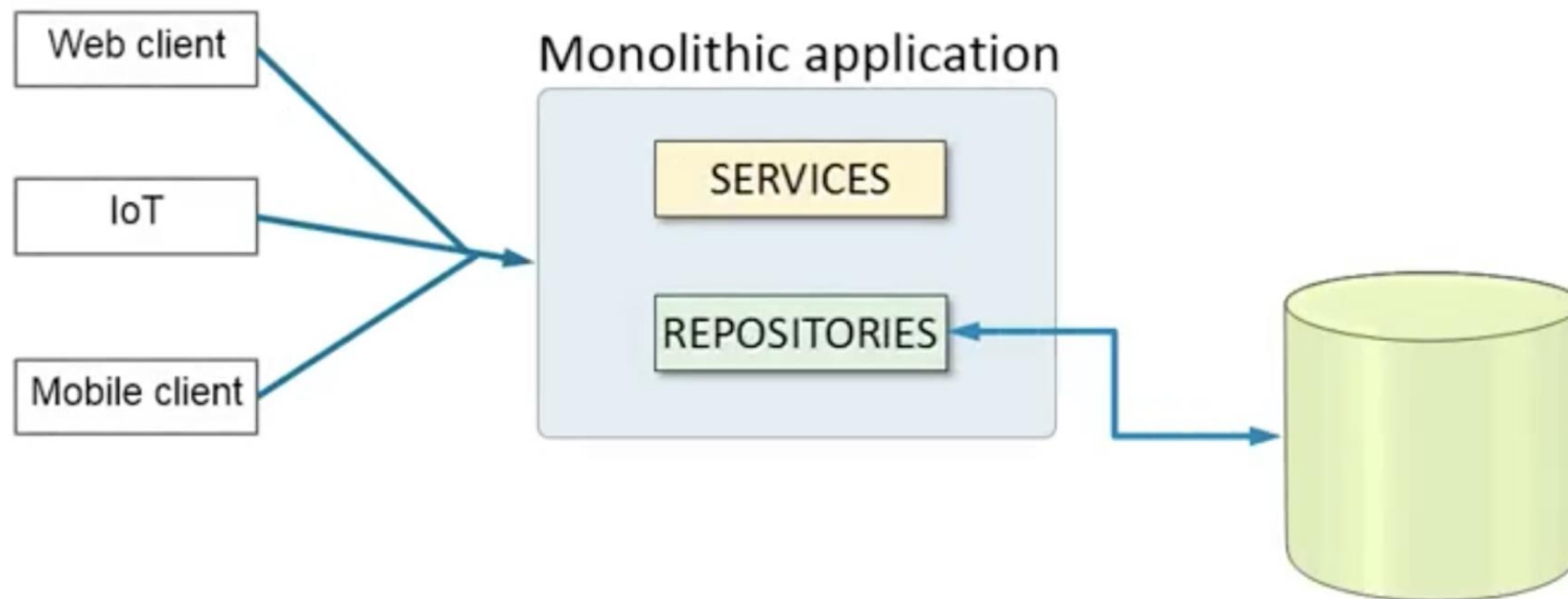
Tolerancia a Fallas



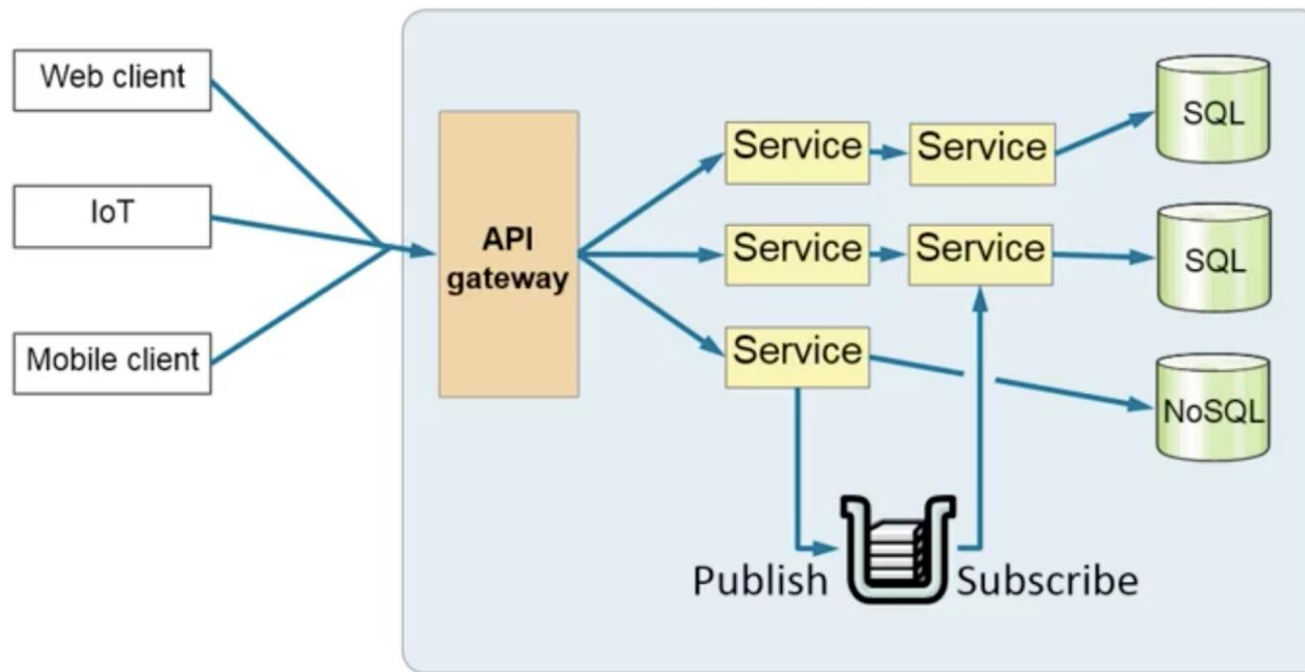
marca la instancia del microservicio como “fallida” y por lo tanto no pasar ninguna nueva petición a esta.

Ventajas de Microservicios

Arquitectura típica monolítica



Arquitectura típica de Microservicios

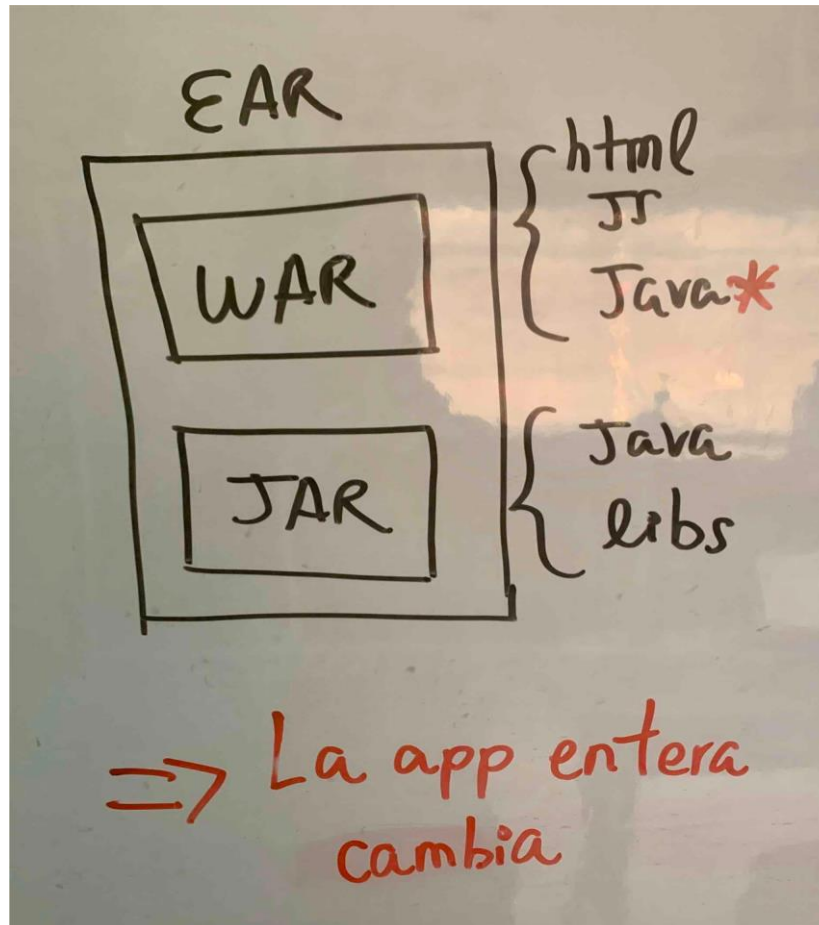


Comparacion

- Codigo
- Claridad
- Despliegue
- Lenguaje
- Escalamiento

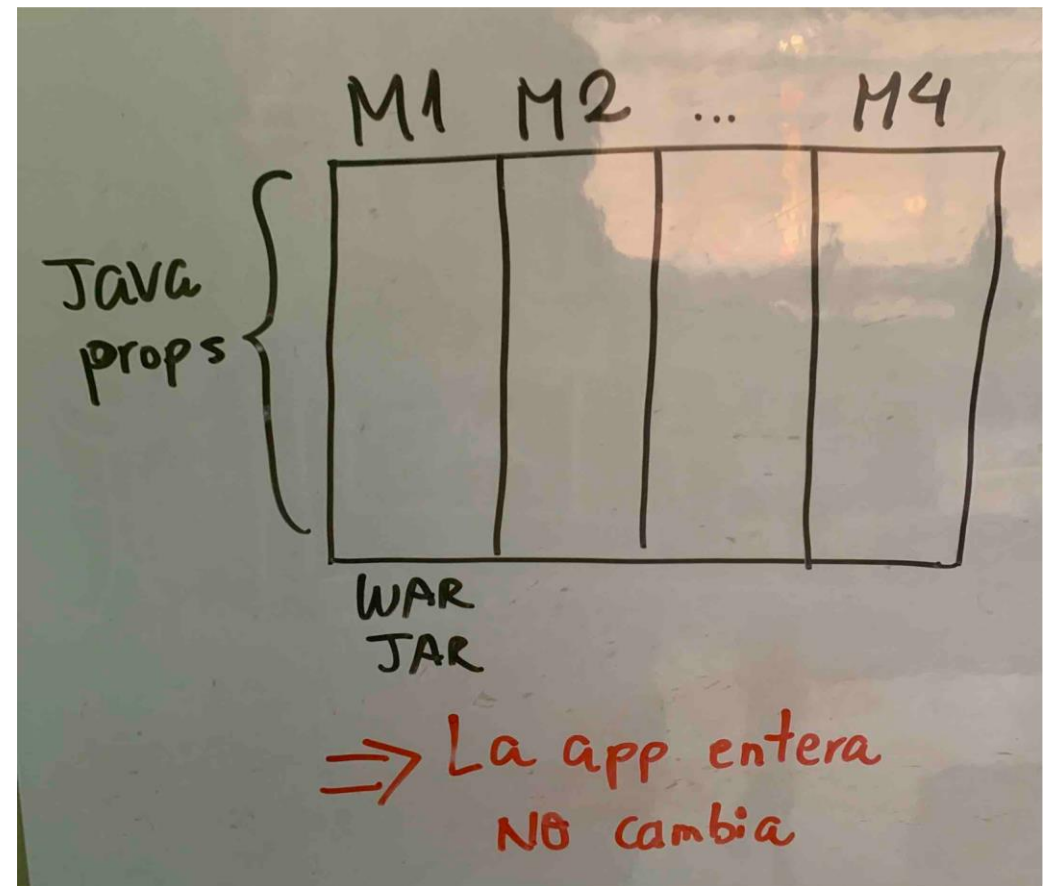
Codigo

- Aplicación Monolítica

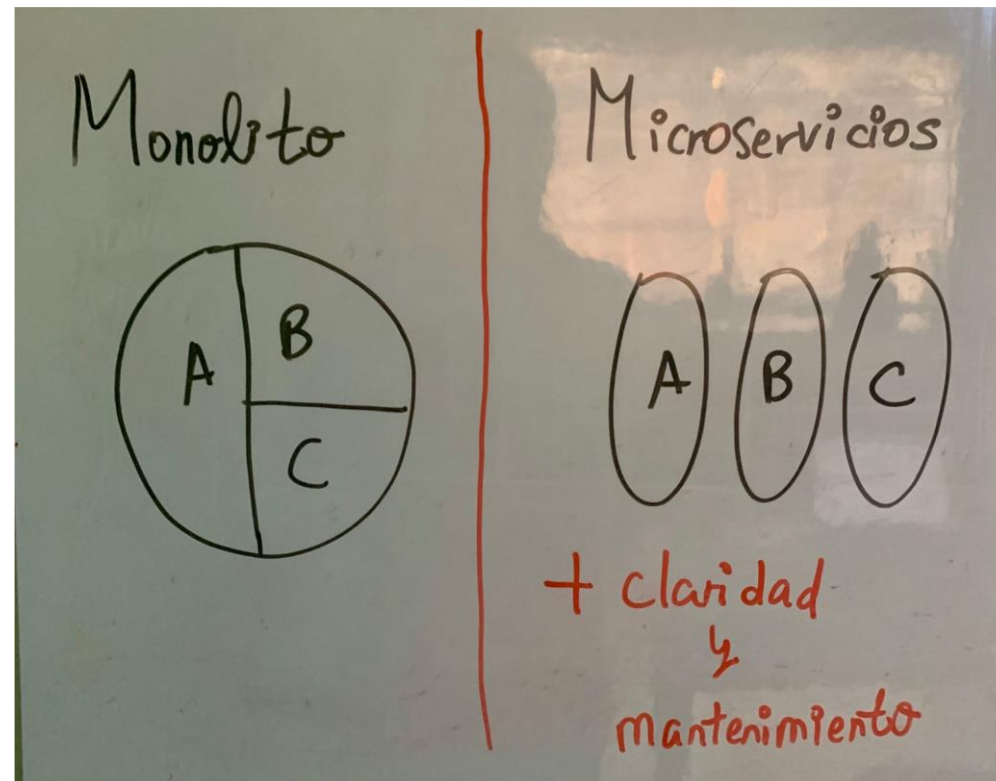


Codigo

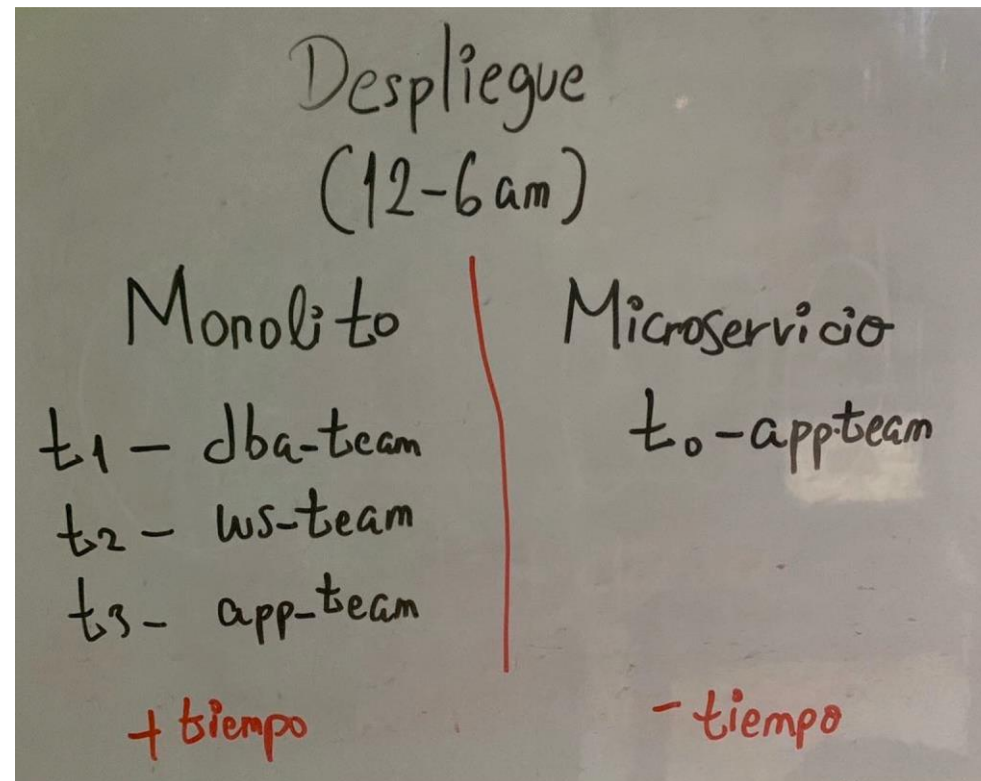
- Aplicación con Microservicios



Claridad



Despliegue



Lenguaje

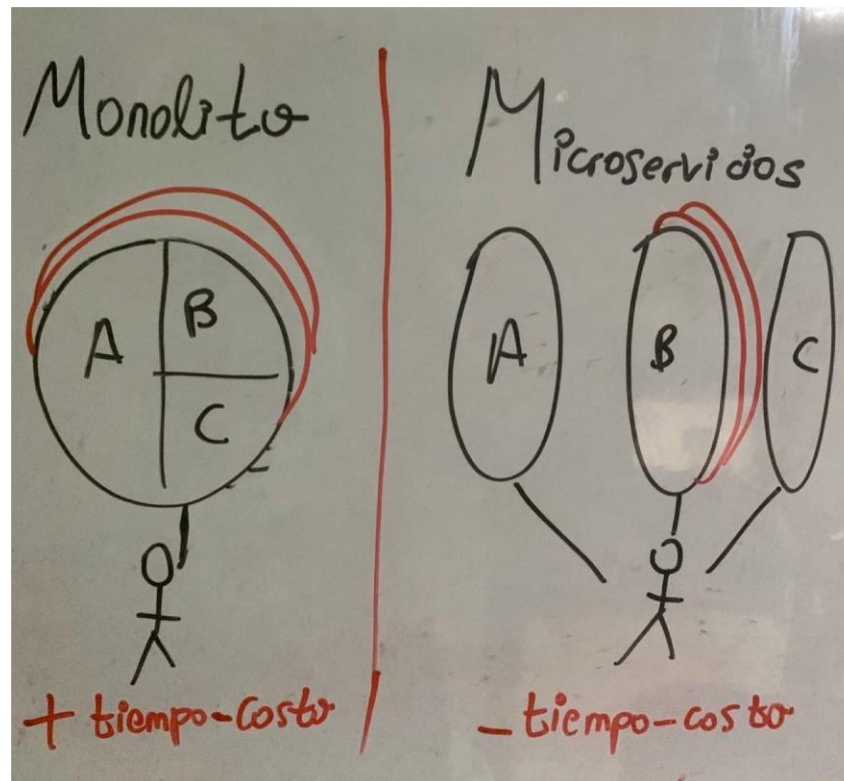
Monolito

- Un solo lenguaje
- Depende de la experiencia del desarrollador

Microservicios

- Cualquier lenguaje
- Depende del tipo de requerimiento
- Importa la interfaz

Escalamiento



Spring Framework

- framework java open source
- fue una alternative a j2EE
- **spring boot**
 - desarrollar y ejecutar rapidamente aplicaciones spring
 - provee configuraciones automaticas que son comunes a todos los aplicaciones spring
 - eleccion ideal cuando necesitamos crear microservicios
- **spring initializr**
 - genera rapidamente nuevas aplicaciones spring boot
 - generar proyectos en maven y gradle

Laboratorio spring

que necesitamos para implementar microservicios?

| | CI | CD |
|-------------|--|---|
| Desarrollo | <ul style="list-style-type: none">• Integrar los cambios sin introducir errores Escribir pruebas automatizadas | <ul style="list-style-type: none">• Despliegue debe usar contenedores Código debe estar programado para ser ejecutado en contenedores |
| Operaciones | <ul style="list-style-type: none">• Herramientas que haga la integración | <ul style="list-style-type: none">• Herramientas Que desplieguen los contenedores y que hagan la orquestación |

PATRONES DE DISEÑO DE MICROSERVICIOS

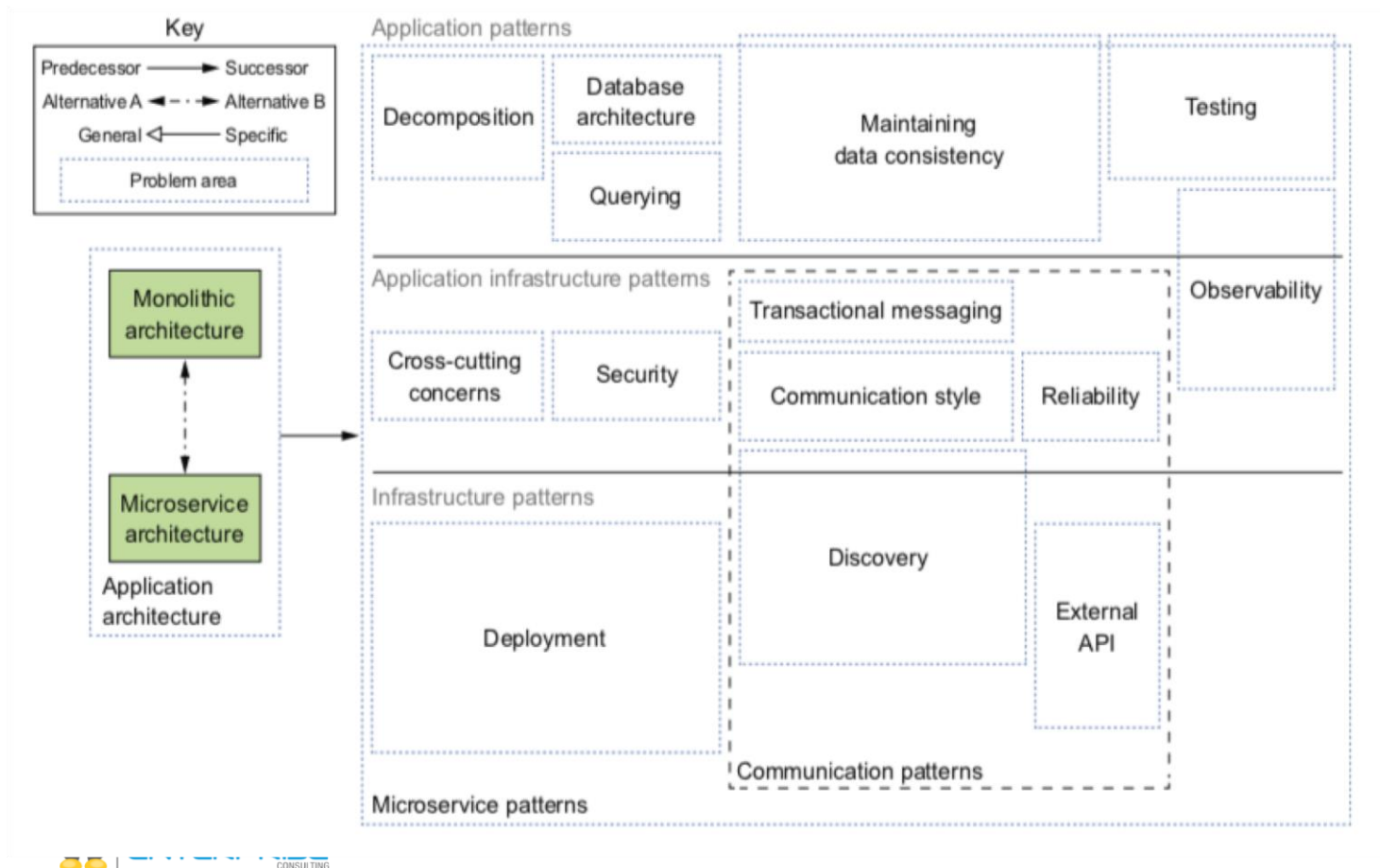
- Patrón de diseño

- Un patrón es una solución reusable a un problema que ocurre en un contexto particular.
- Tuvo su origen en la arquitectura de edificaciones y ha probado ser útil en la diseño y arquitectura de software.

- Aplicado a microservicios:

- Ayudan a construir arquitectura
 - Se debe usar microservicios?
 - Beneficios y desventajas
 - Si microservicios, solucionan problemas de arquitectura y diseño.

Clasificación



- Patrones de infraestructura
- Patrones de infraestructura de la aplicación
- Patrones de la aplicación

Patrones de descomposicion

Como identificar servicios?

Usando Capacidades de negocio

- Describen el negocio y son estables.
- Como conducen el negocio cambia con el tiempo.
 - Transferencias bancarias
 - Ventanillas
 - Cajeros automáticos
 - Smartphones
 - Estable: la transferencia.
 - El que.
 - Cambia: el canal de la transferencia.
 - el como

Identificar capacidades de negocio

- Propósito, estructura y procesos de negocio
- Identificada como un servicio
- Especificación: entradas, salidas, SLA.
 - Ejemplo: vender seguro vehicular.
- Identificando capacidades de negocio en un Restaurant

Capacidades

| Capacidades |
|----------------------------|
| Administración del cliente |
| |
| Toma de ordenes de pedidos |
| Preparación de ordenes |
| |
| Administración de Courier |
| Entrega de pedidos |
| Contabilidad |

Sub-capacidades

| Capacidades |
|----------------------------|
| Administración del cliente |
| Administración de Ordenes |
| Toma de ordenes de pedidos |
| Preparación de ordenes |
| Logística |
| Administración de Courier |
| Entrega de pedidos |
| Contabilidad |

De capacidades de negocio a servicios

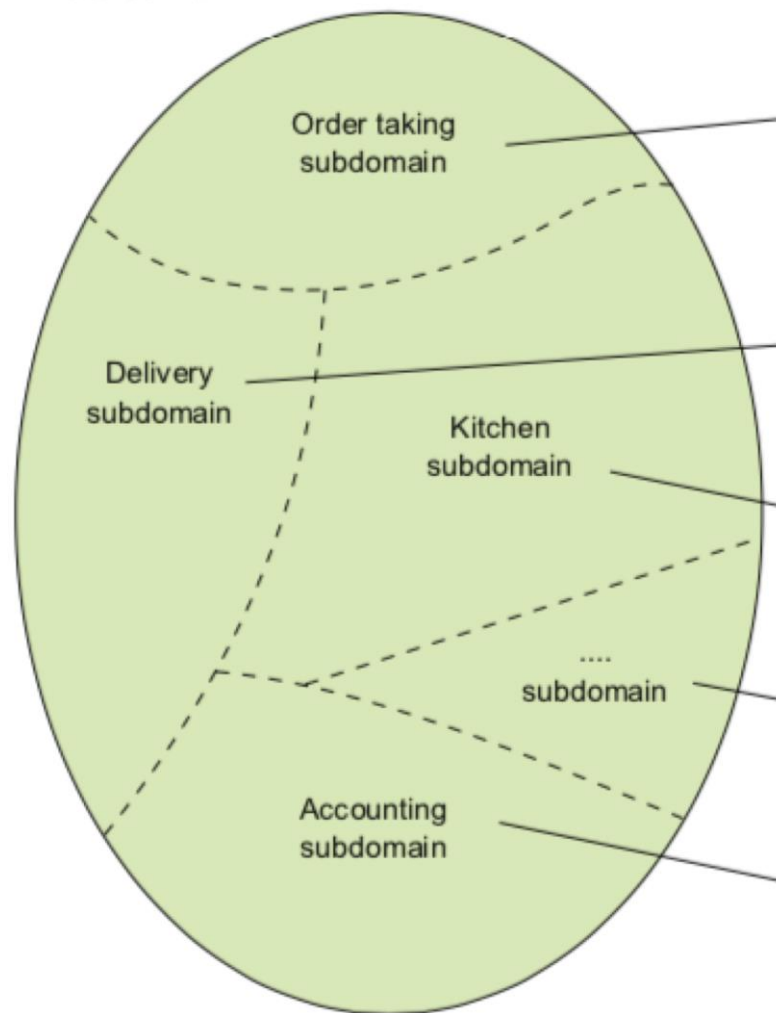
| Capacidades | Servicios |
|----------------------------|----------------------|
| Administración del cliente | Cliente Service |
| Administración de Ordenes | |
| Toma de ordenes de pedidos | Orden Service |
| Preparación de ordenes | Cocina Service |
| Logística | Entrega Service |
| Administración de Courier | |
| Entrega de pedidos | |
| Contabilidad | Contabilidad Service |

- Beneficio: Servicios y Arquitectura estable
- Servicios cambian y evolucionan
- Investigar como servicios colaboran
 - Si Comunicación excesiva=combinar servicios

Descomposición usando Dominios de negocio

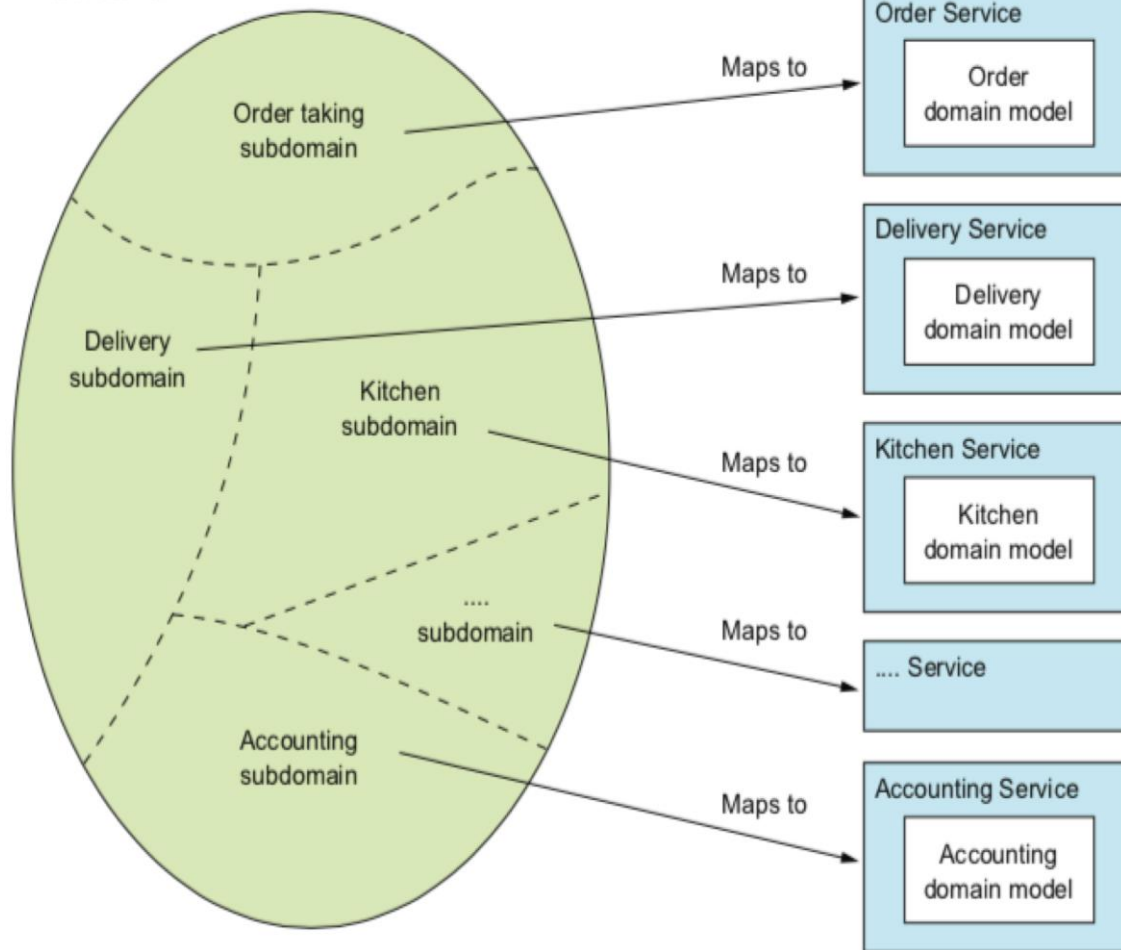
- Usa Diseño dirigido por el dominio. DDD.
 - diseño dirigido por el negocio, es un enfoque para construir aplicaciones de software complejos centrada en el desarrollo de un modelo de dominio orientado a objetos.
- Captura y Soluciona problemas del dominio
- Enfoque tradicional, crea un Unico modelo.
 - DDD, crea Multiples modelos de dominio
 - Subdominios
- Subdominios Identificados como capacidades de negocio
 - Analizar el negocio
- Modelos resultantes similares

FTGO domain



Podemos descomponer el dominio en subdominios

FTGO domain



- Subdominios asociado con modelo de dominio
- Modelo de dominio asociado con un servicio

Ddd y la arquitectura de microservicios

- DDD alinea con microservicios
 - Subdominios=microservicios
 - independientes y pequenos
 - Equipos autónomos
 - Desarrollado por un único equipo

GRACIAS