SOA: conceptos básicos

SOA

Es un concepto de arquitectura de software que define la utilización de servicios, programas o rutinas que realizan una función específica, para dar soporte a los requisitos del negocio.

La Arquitectura Orientada a Servicios, lo que permite es la creación de sistemas de información ampliables, versátiles y flexibles que pueden ayudar a las organizaciones a impulsar el rendimiento y, al mismo tiempo, reducir costes de IT y mejorar la flexibilidad en los procesos del negocio. Además, brindan una forma bien definida de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y da soporte a las actividades de integración y consolidación de los datos de cualquier organización.

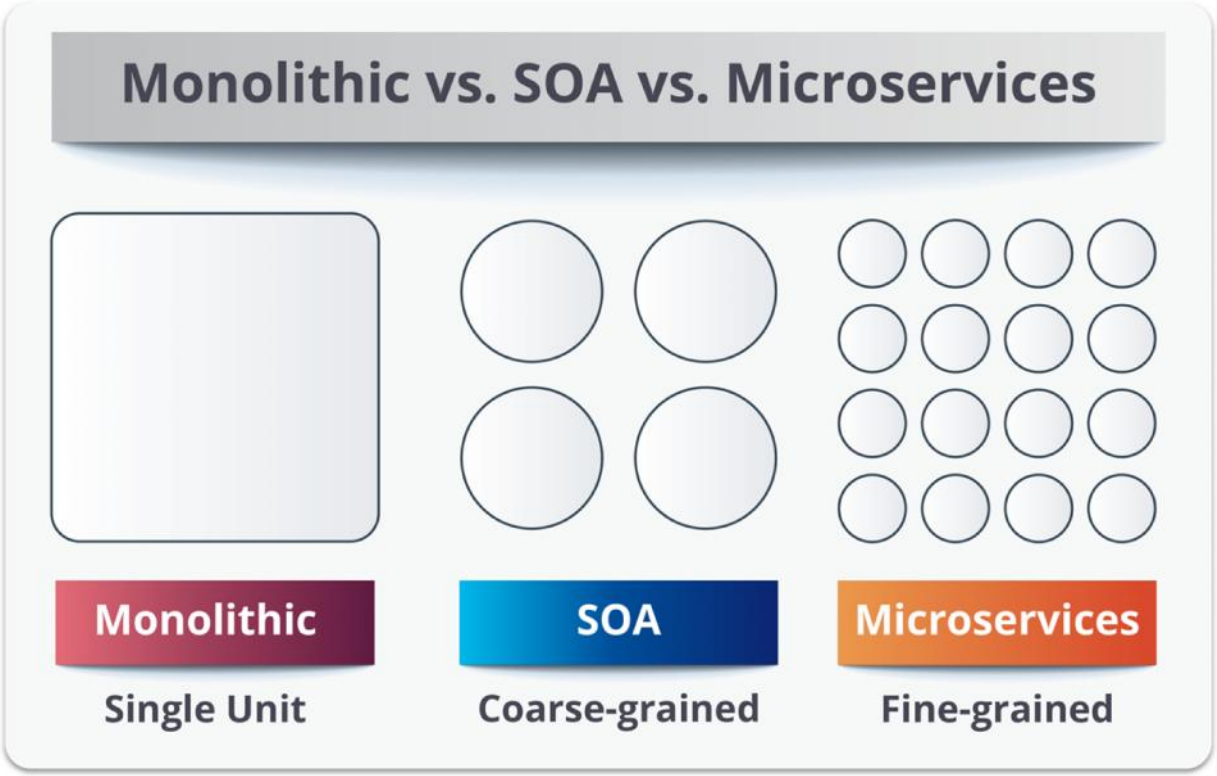
SOA

- Principio de diseño de software
- Microservicios como componentes de la aplicación
- Usando un protocolo de comunicación
- Las aplicaciones en los entornos de SOA carecen de sentido individual, ya que o bien es un conjunto de servicios, o bien es en sí mismo un servicio para otro sistema mayor.

Beneficios de SOA

- Aumento de la eficiencia en los procesos.
- Amortización de la inversión realizada en sistemas.
- Reducción de costes de mantenimiento.
- Facilita la adaptación al cambio, con la integración con sistemas heredados.
- Fomento de la innovación orientada al desarrollo de servicios, acordes con el dinamismo de mercado. Se modernizan los sistemas obsoletos por razones económicas, funcionales o técnicas.
- Simplificación del diseño, optimizando la capacidad de organización.

Monolítico vs Soa y Microservicios



Propiedades de cada microservicio

- Representa una actividad lógica de negocio
- Está auto-contenido
- Es una caja negra para sus consumidores
- Puede contener servicios anidados

Principios (técnicamente relevantes)

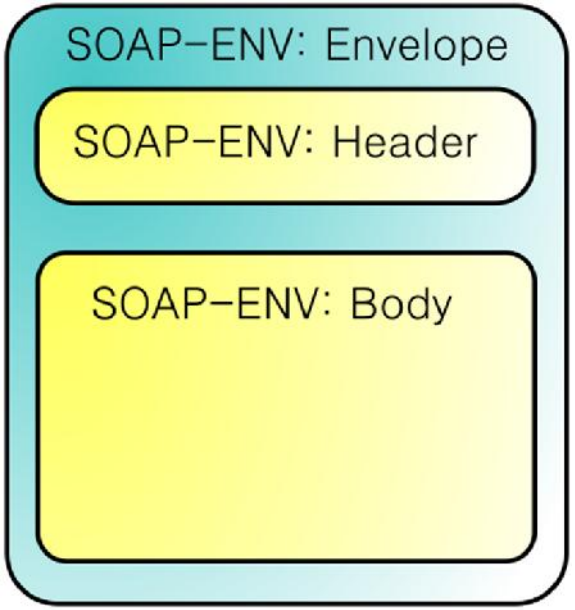
- Loose coupling
- Autonomía
- Statelessness (carencia de estado)
- Granularidad
- Descubrimiento
- Reusabilidad
- Encapsulamiento

Implementaciones más comunes

- **WSDL** (Web Services Description Language)
- **SOAP** (Simple Object Access Protocol)
- **Colas de mensajes (RabbitMQ, etc)**
- **RESTful**
- **WCF**
- **OPC-UA**

SOAP

SOAP define el protocolo de los mensajes a través de XML



WSDL

Lenguaje de descripción de servicios. Define qué tiene disponible el web se

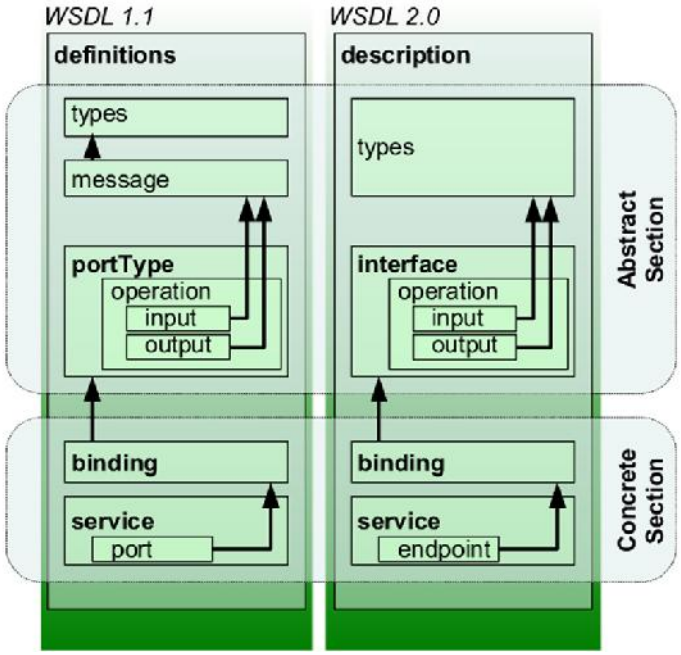
Operaciones

Puertos

Binds

Mensajes

Tipos



RESTful

Basado en la implementación de HTTP

- GET = list / item
- POST = create
- PUT = update
- DELETE = delete

Rutas

https://sitio.com/recurso/:identificador

Ejemplo:
GET https://sitio.com/producto/1

Códigos (relevantes) HTTP

2xx: Respuesta exitosa

4xx: Error de cliente

5xx: Error de servidor

Métodos vs respuestas REST

Método	Operación	Éxito	Fracaso
GET	Leer	200	404
POST	Crear	201	406
PUT	Editar	200	404/406
DELETE	Borrar	200	404/406

Seguridad en Servicios Web

Autenticación: HTTP / Basic

- Consiste en el envío de un usuario y contraseña en texto plano, en un campo de encabezado (header) de HTTP
- La autenticación está cifrada en base64
- Contenido: usuario:password
- Ejemplo:
  Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l

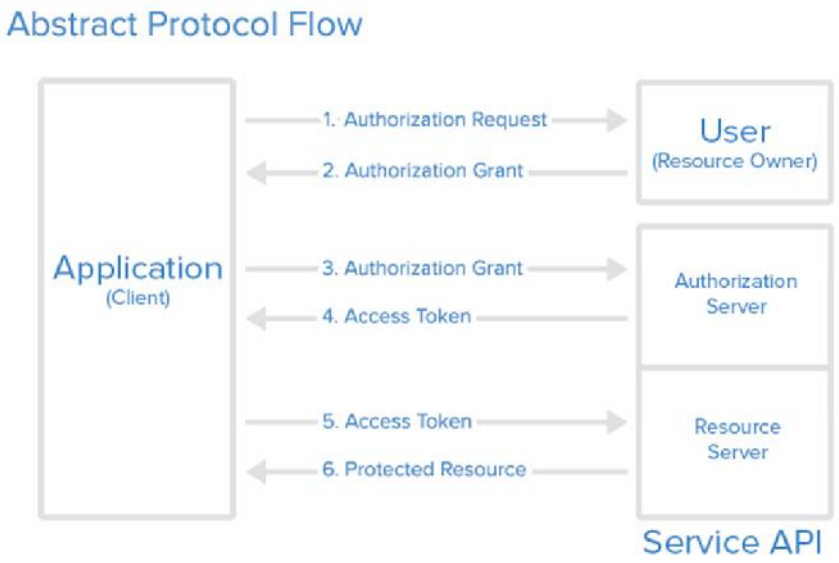
Bearer token

Consiste en autenticar mediante un token, por medio de encabezado **Authentication**. El token autoriza al “portador” (bearer).

Authorization: Bearer <token>

Autenticación: oAuth2

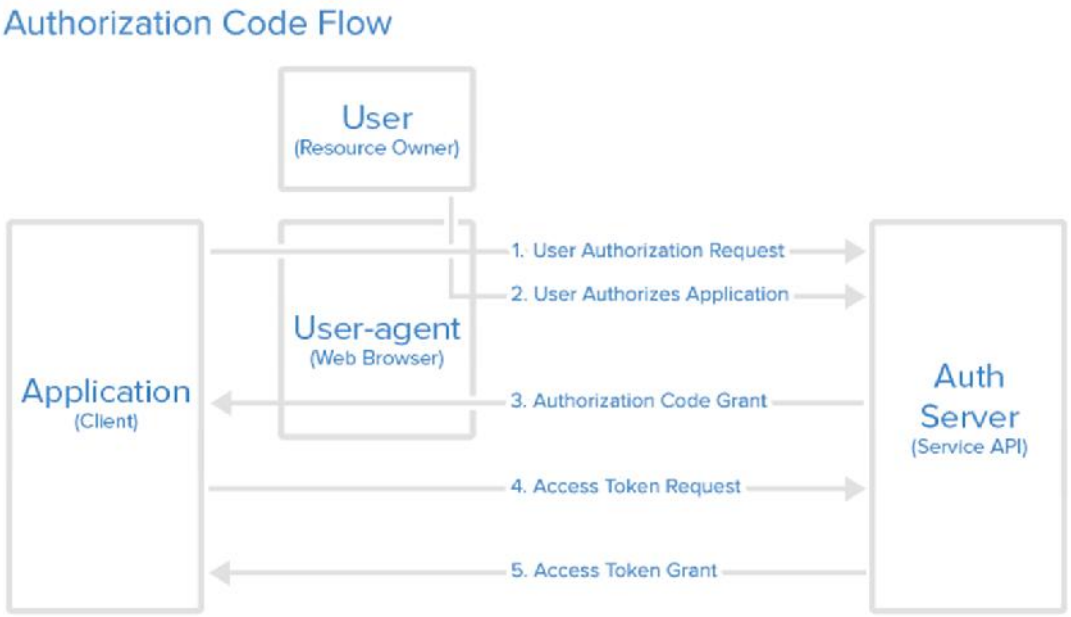
- Es un sistema de autenticación que permite a las aplicaciones obtener acceso limitado a servicios HTTP.
- El esquema divide los servicios en:
  - Servidor de autorización
  - Servidor de recursos



Protocolo oAuth2

Autorizaciones oAuth2

1. Authorization Code. Usado para acceso a recursos de terceros.
2. Implicit. Usado para apps móbiles.
3. Resource Owner Password Credentials
4. Client Credentials
5. JWT (java web token)



Authorization Code

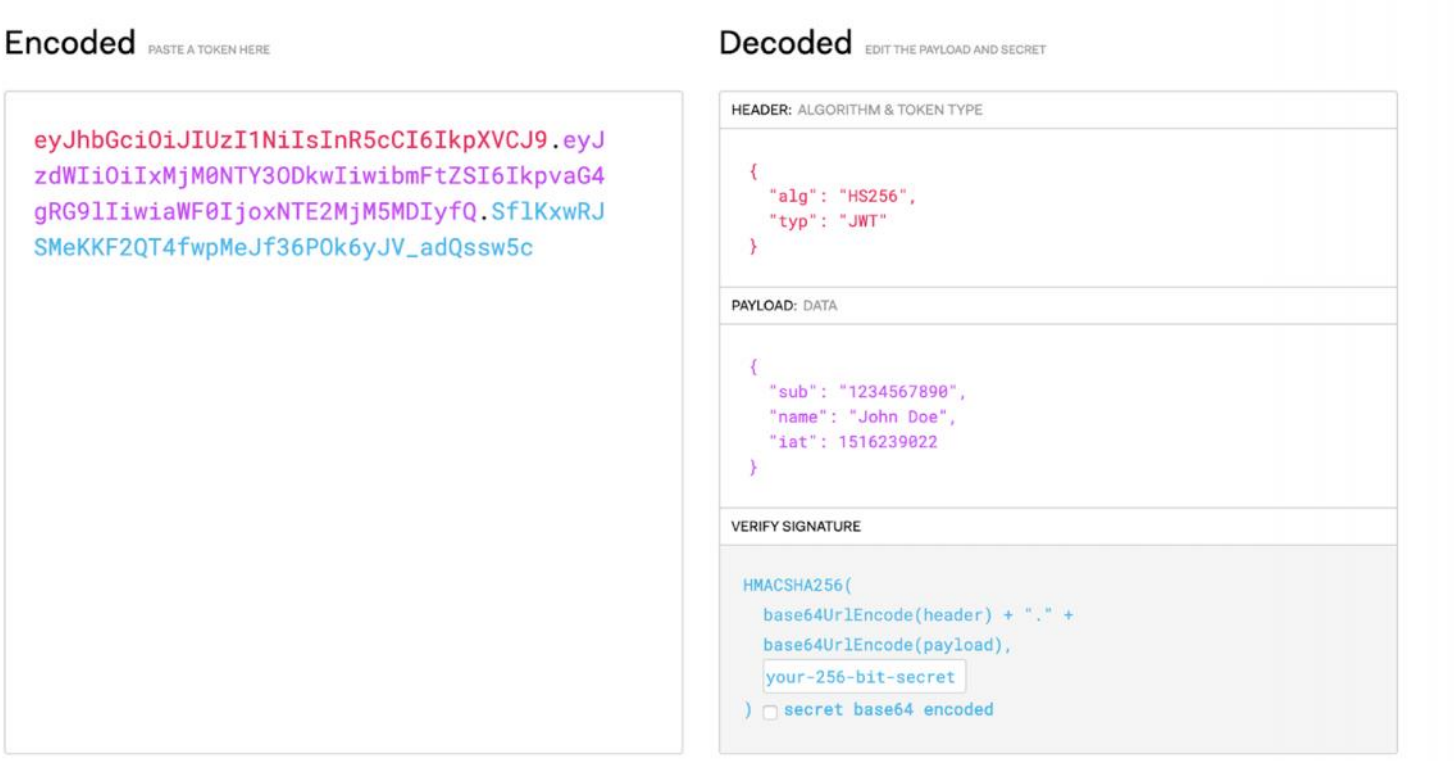
Client Credentials

https://oauth.example.com/token
?grant_type=client_credentials
&client_id=CLIENT_ID
&client_secret=CLIENT_SECRET

JWT

1. Tokens auto-contenidos
2. No requieren un servidor de autorización, sino que contienen la autorización en sí
3. Contenido encriptado mediante par de llaves pre-autorizadas

Ejemplo JWT

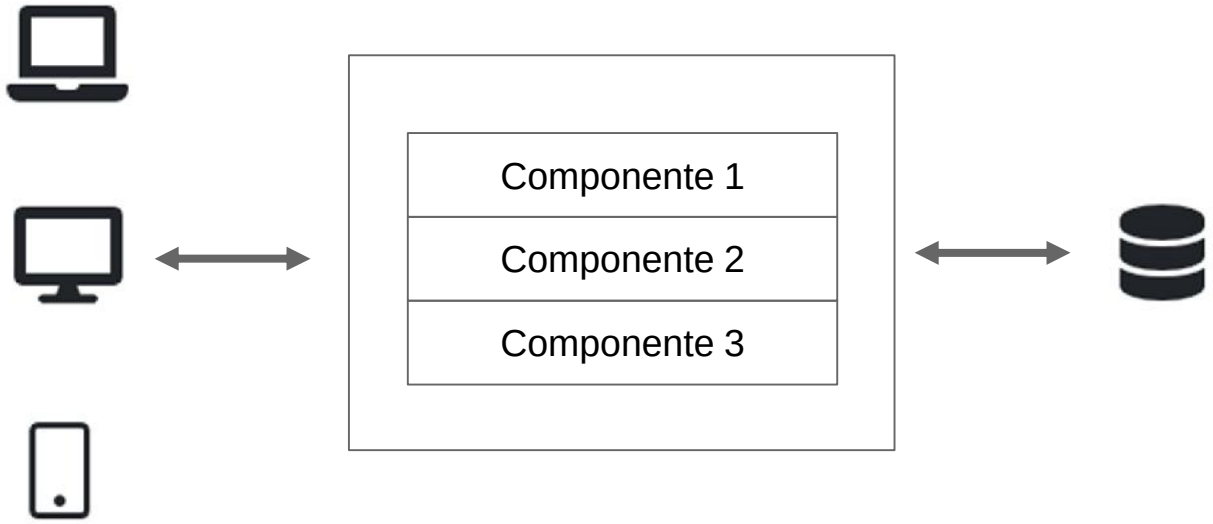

Encoded
PASTE A TOKEN HERE
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
Decoded
EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "HS256",
"typ": "JWT"
}
PAYLOAD: DATA
{
"sub": "1234567890",
"name": "John Doe",
"iat": 1516239022
}
VERIFY SIGNATURE
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
your-256-bit-secret
) secret base64 encoded

SOA: Service Choreography

Microservicios

vs sistemas
monolíticos

Sistemas monolíticos



Microservicios

Tres claves para el uso de microservicios

1. Ideales para sistemas grandes
2. Arquitectura orientada a objetivos
3. Enfocados a ser reemplazables

Características de los microservicios

- Pequeños en tamaño
- Habilitados para mensajería
- Ligados a contextos
- Desarrollados en forma autónoma
- Liberación independiente
- Descentralizados
- Construidos y liberados con procesos automatizados

Desarrollo de caso SOA

Aplicación: TengoHambreApp

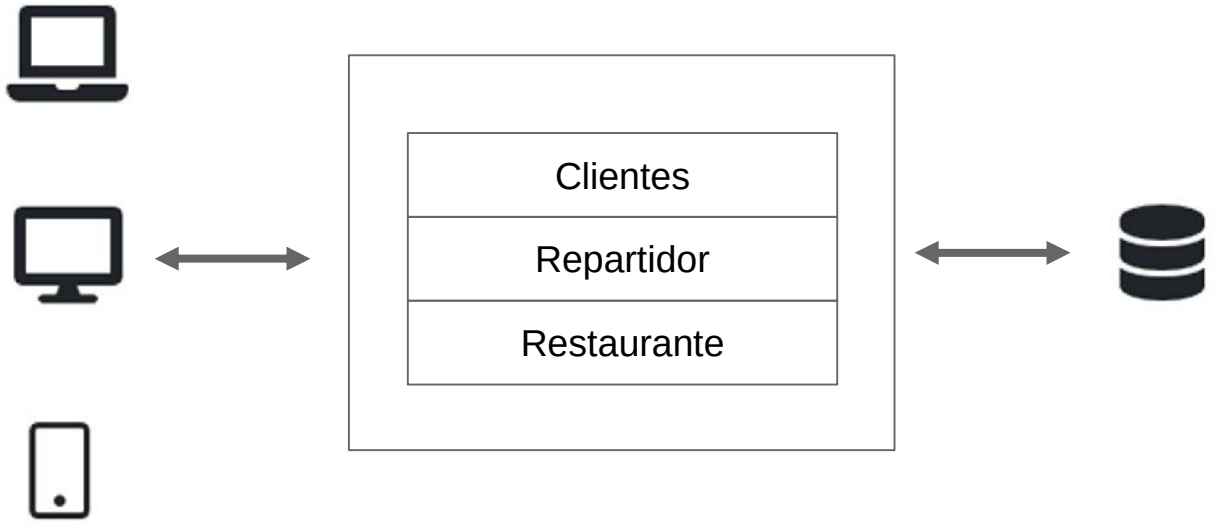
Creación de una aplicación que permita hacer pedidos a domicilio dentro de la Ciudad de Guatemala, contando con un catálogo de múltiples restaurantes, cada uno con un menú de platos.

La compra será 100% en línea y por medio de tarjeta de crédito, percibiendo una comisión sobre la venta.

Ejercicio en grupos

1. Identificar los actores de la aplicación.
2. Listar los posibles casos de uso de cada uno de los actores.
3. Detallar los componentes de software que se podrían utilizar para crear la aplicación.

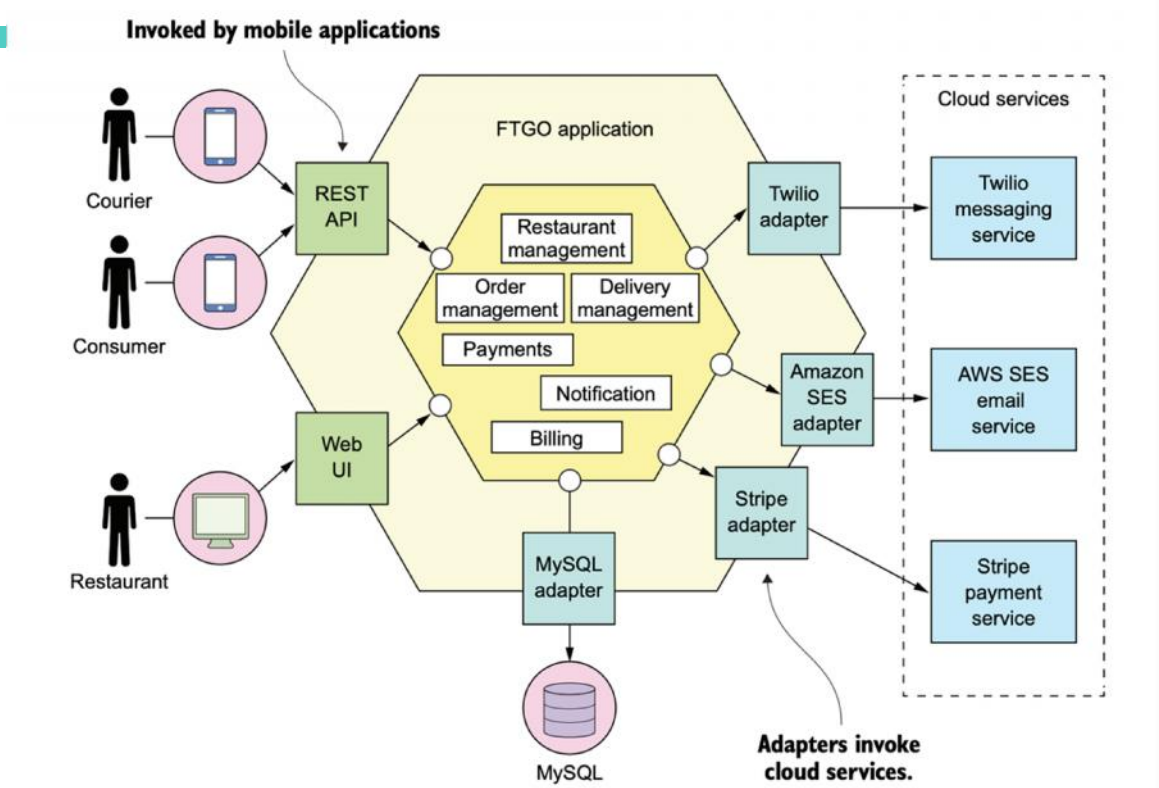
Sistemas monolíticos



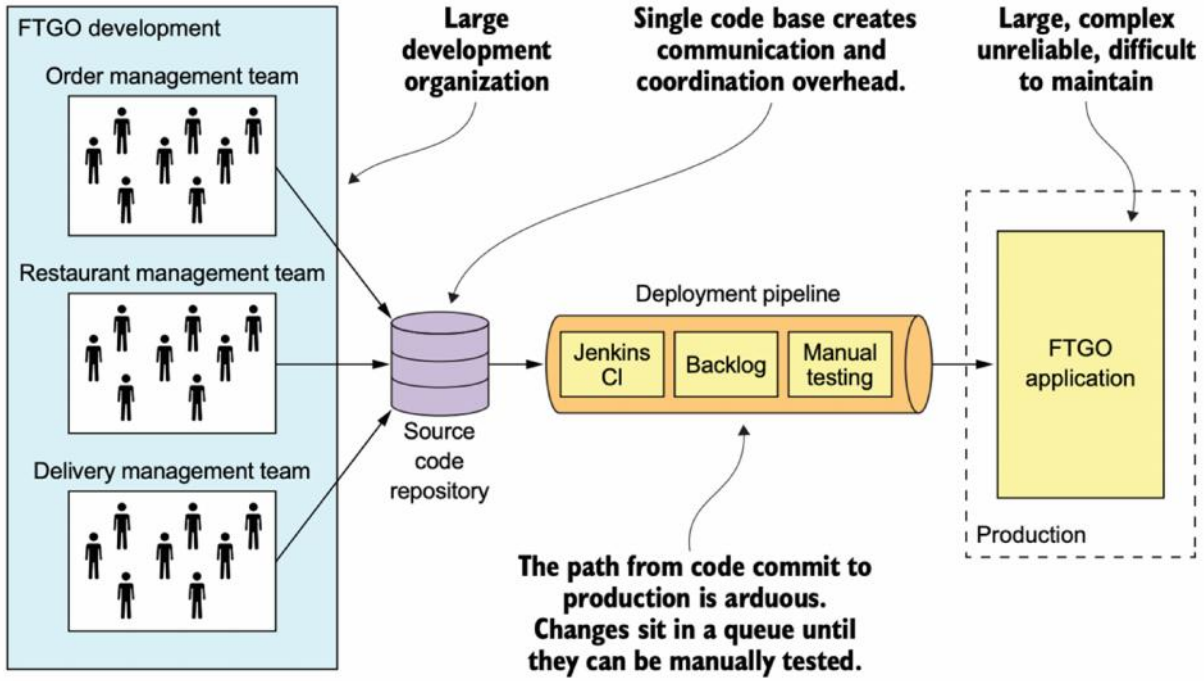
Microservicios

Rompiendo los sistemas monolíticos

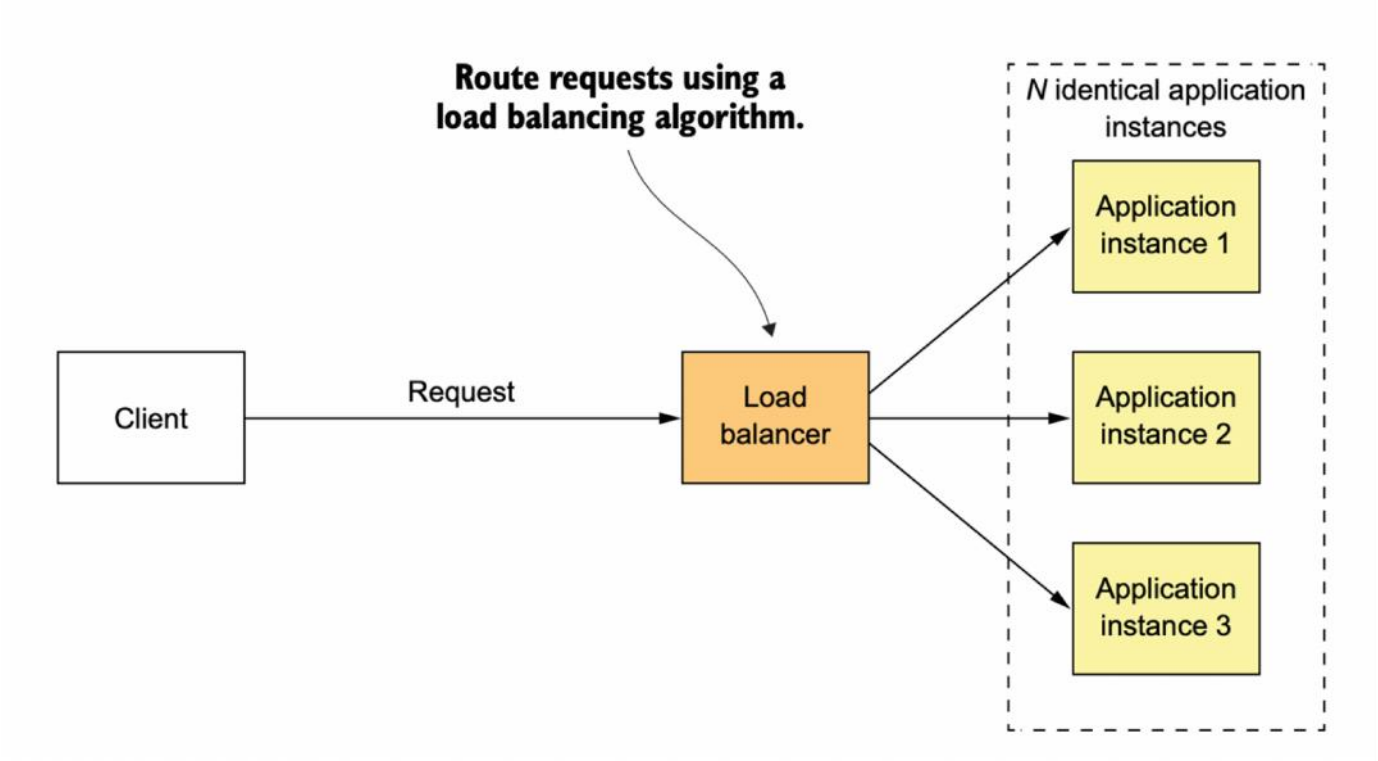
Ejemplo de aplicación monolítica



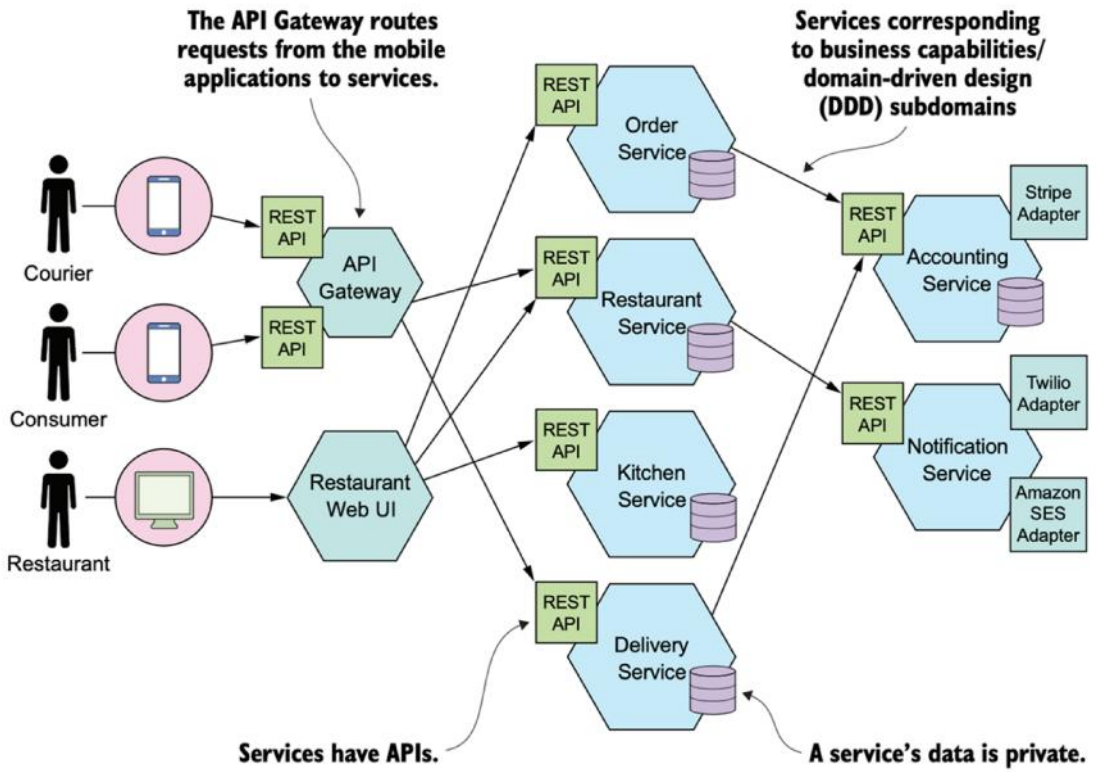
Proceso de construcción monolítico



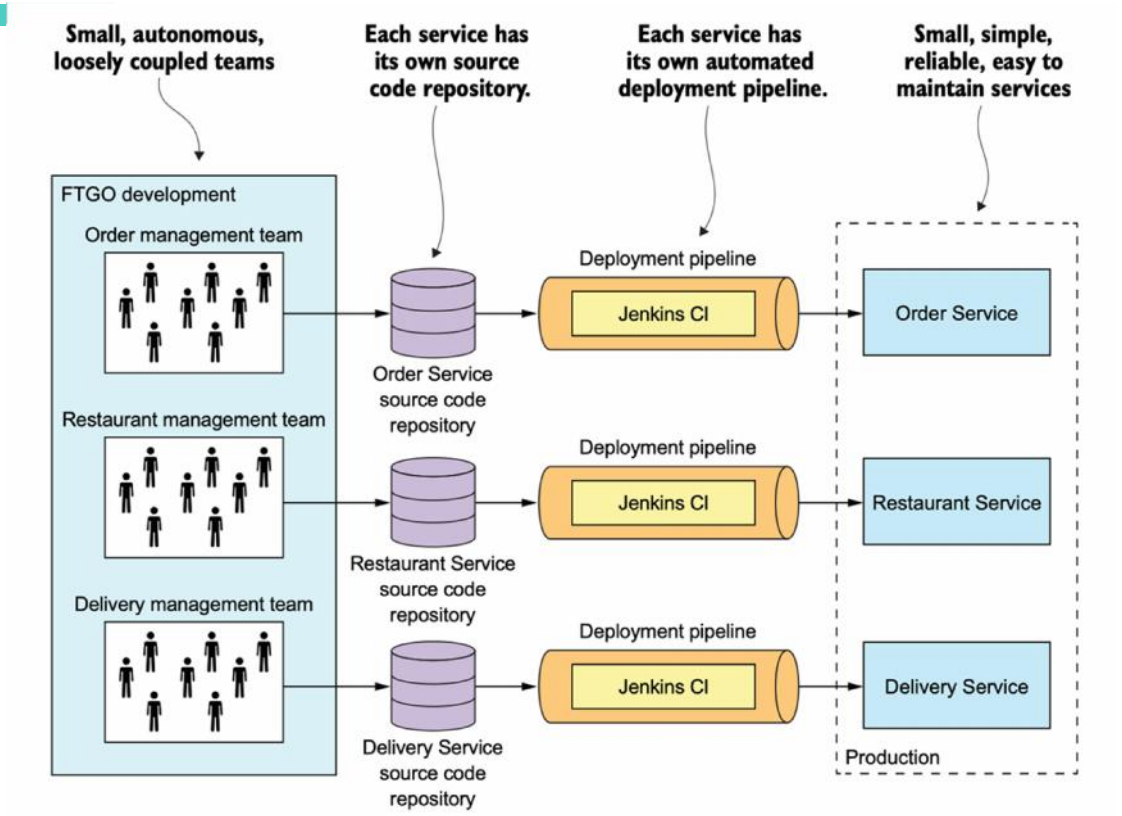
Retos de capacidad



Alternativa en microservicios



Nuevo proceso de desarrollo



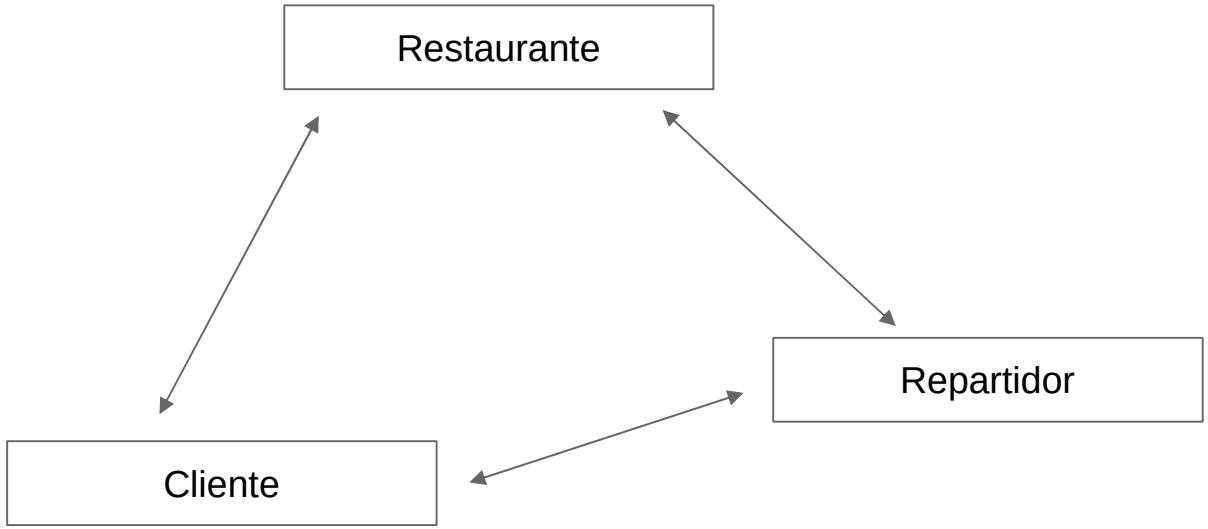
Coreografía y orquestación

de microservicios

Coreografía de servicios

- Utilizado en ambientes multi-corporación especialmente
- Cada servicio define su comunicación con los restantes
- Autonomía de servicios
- Servicios de bajo nivel.

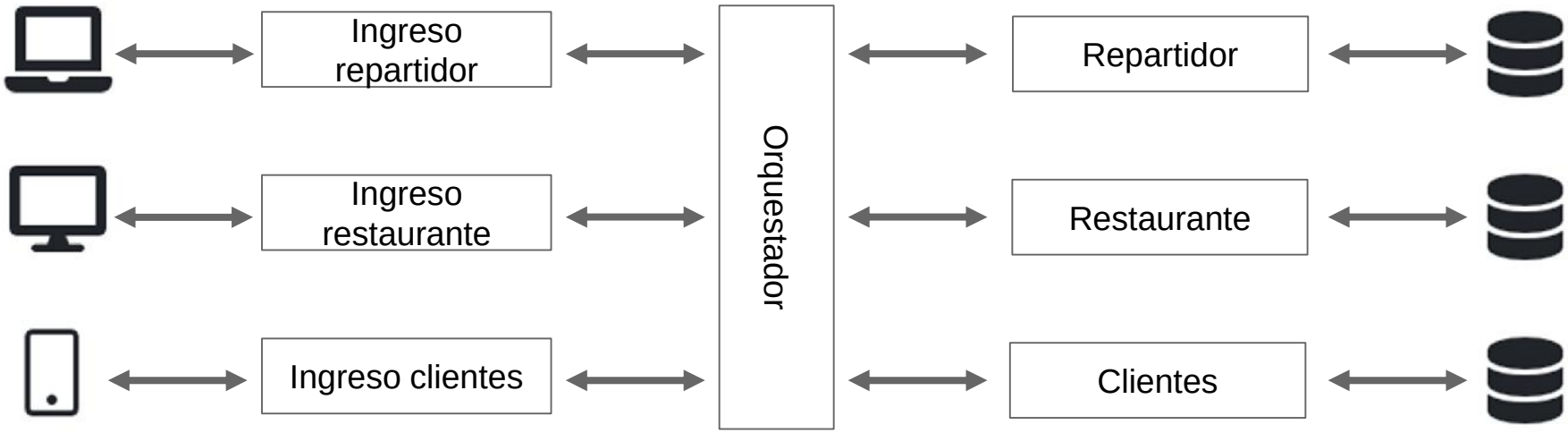
Ejemplo de coreografía



Orquestación de servicios

- Utiliza un punto central de control de los servicios.
- Cada servicio es autónomo y desconoce a los demás.
- Se usa para composición de servicios

Ejemplo de orquestación



Orquestación

La orquestación es el enfoque más comúnmente usado en la composición de servicios y procesos de negocio.

Con la orquestación, defines la secuencia de pasos en un proceso, incluyendo condiciones excepciones, y luego crear un controlador central para implementar la secuencia. En una SOA, los pasos individuales de la secuencia son implementados por operaciones sobre servicios.

La secuencia puede ser implementada por una variedad de técnicas diferentes. Frecuentemente, de forma relativa las composiciones de servicios simples son orquestados en código, tales como Java o C#, que reside dentro del servicio compuesto. Pero para operaciones más complejas, frecuentemente usarás una herramienta para crear un modelo visual de la secuencia, y luego para generar el código que ejecute la secuencia, típicamente en un entorno de ejecución dedicado.

Orquestación

- Define un solo control maestro de todos los aspectos de un proceso (enfoque top-down). Soporta una vista gráfica de la secuencia.
- Se mapea fácilmente a SOA.
- Generalmente es más simple con el cual iniciar; pero frecuentemente es más difícil de escalar a procesos más complejos.
- Es manejado por el modelo de secuencia gráfica, por ejemplo, a las funciones le siguen formularios.
- Representa el estado de la práctica, y es soportado por la mayoría de las herramientas de esta forma, la popularidad de este enfoque es comprensible. Pero no soporta todos los tipos de procesos igualmente bien.

Coreografia

La coreografía proporciona un enfoque diferente que está obteniendo aceptación en escenarios que tienen procesos complejos con algunas partes interactivas, sistemas basados en agentes y basados en eventos. En un enfoque de coreografía, se crean las reglas que determinan el comportamiento de cada participante individual en el proceso. El comportamiento general del proceso basado en la interacción de las piezas individuales, cada una de forma autónomia siguiendo sus propias reglas.

Coreografia

La coreografía proporciona un enfoque diferente, actualmente hay dos enfoques principales para coreografía, basado en mensaje, y basado en componente de trabajo.

El enfoque de mensaje está basado en examinar los mensajes entre participantes en un proceso general. Con este enfoque, defines comportamientos exhaustivamente capturando los contratos de mensajes entre las partes colaboradoras. Este es el mecanismo soportado por el estándar WS-CDL (Lenguaje de Definición de Coreografía de Web Services) y es utilizado frecuentemente para aplicaciones B2B.

El enfoque de coreografía basado en mensaje es atractivo debido a que sólo necesitas especificar las definiciones de intercambio de mensajes, tanto de sintaxis, semántica y de comportamiento.

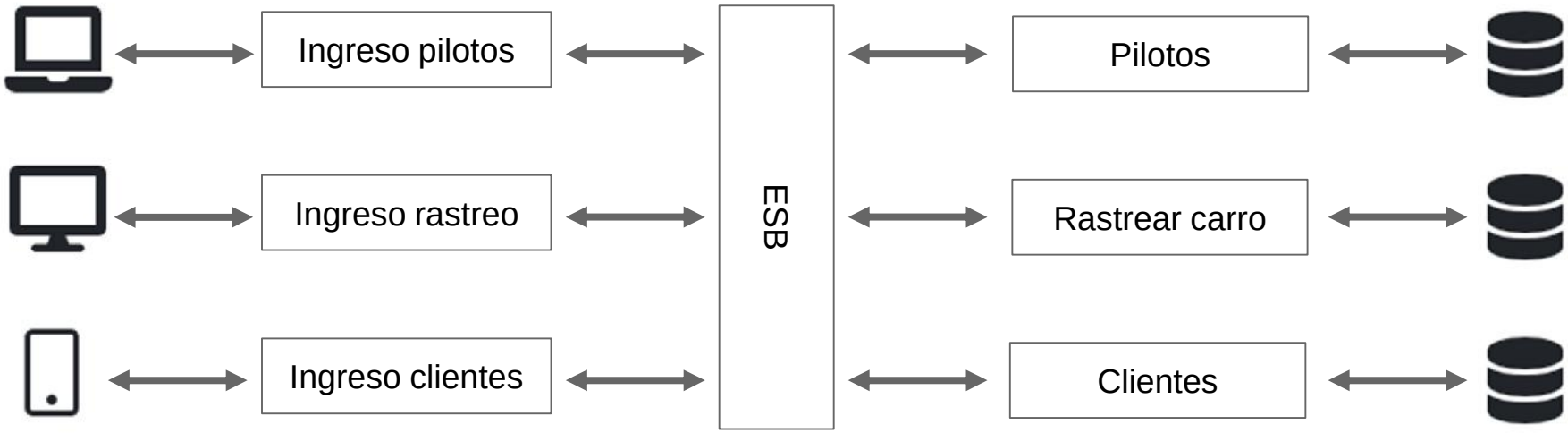
Enterprise Service Bus

ESB

ESB: Enterprise Service Bus

- Es una implementación de una orquestación de servicios
- El orquestador (bus) es broker de servicios y orquestador (cliente y servicio a la vez)
- Comunica cada uno de los servicios

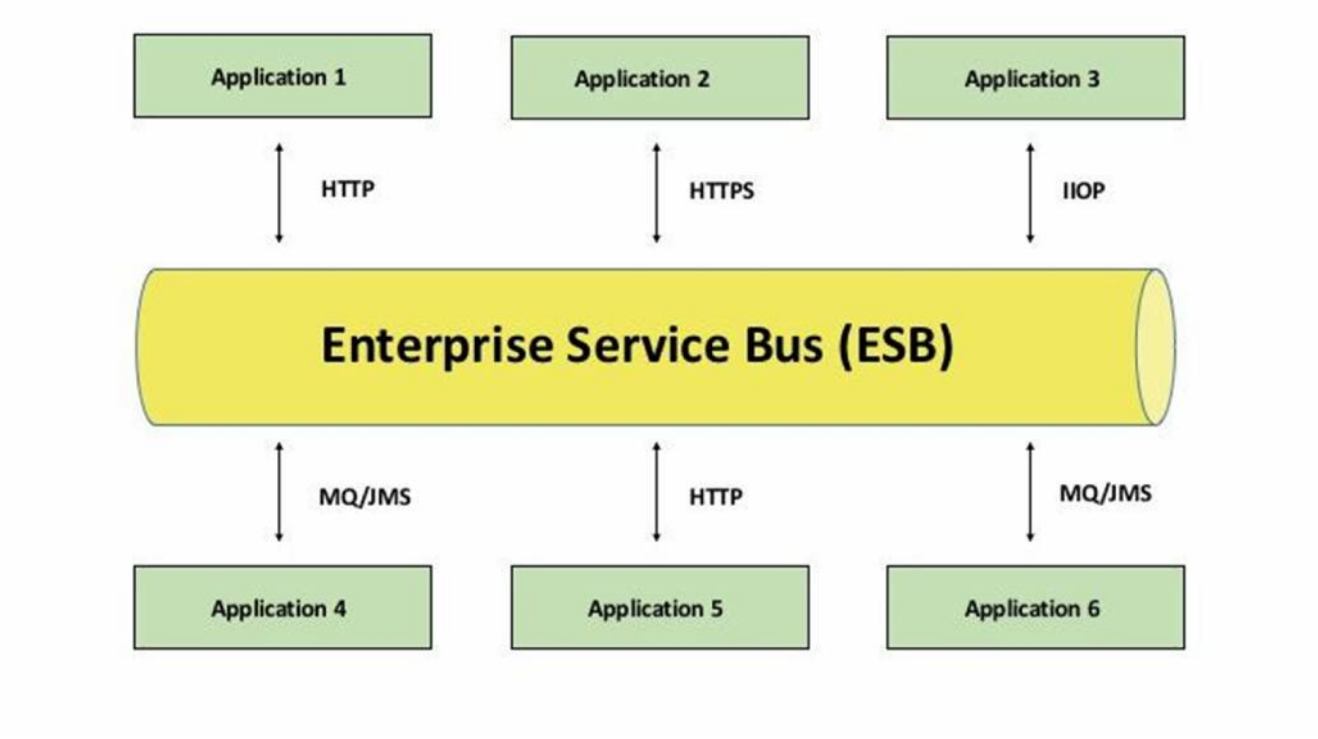
Ejemplo de ESB



¿Qué tiene de especial respecto a un orquestador?

- El ESB contiene reglas de negocio que facilitan la comunicación y reduce las interacciones entre los servicios.
- Un ESB puede contener varios nodos para distribución geográfica o de unidades de negocio.

ESB


Application 1
Application 2
Application 3
HTTP
HTTPS
IIOP
Enterprise Service Bus (ESB)
MQ/JMS
HTTP
MQ/JMS
Application 4
Application 5
Application 6

Topologías de ESB

Conceptos básicos

ESB / Enterprise Service Bus

- Es una implementación de una orquestación de servicios
- El orquestador (bus) es broker de servicios y orquestador (cliente y servicio a la vez)
- Comunica cada uno de los servicios
- Provee abstracción a los clientes, quienes conocen al ESB (o uno de sus nodos) aunque desconocen detalles de su implementación interna (la cual puede ser cambiante).

Patrones de modelo de negocio

Única compañía global

Única compañía global.

Desde una ubicación se trata con proveedores locales, legislaciones, impuestos, almacenamientos, productos.

Todo centralizado.

Las empresas aspiran a esto pero no es óptimo.

Múltiples geografías

Cada compañía local trata con proveedores locales, impuestos y legislaciones.

Cada compañía local ofrece servicios a clientes.

Las compañías interactúan con la compañía global para cumplir requerimientos globales e interacción, reportes, etc.

Múltiples divisiones de negocio

Cada división de negocio se maneja globalmente y se distribuye a las diferentes ramas de la empresa.

Es un modelo común para bancos: servicios bancarios, tarjetas de crédito, préstamos, etc.

Red de franquicias

Se proveen servicios globales que se distribuyen a todas las franquicias.

Cada franquicia puede tener personalizaciones a sus servicios.

Modelo jerárquico

Cada división provee sus propios servicios, haciendo uso (probablemente parcial) de los servicios ofrecidos por la central.

Puede centralizarse en divisiones locales, que a su vez puede tener personalizaciones en las ramas finales.

Patrones de gobernanza de SOA

Gobernanza única

La interacción se realiza solamente en el área de negocio que provee la gobernanza del mismo.

Por ejemplo, la responsabilidad de una planta de manufactura puede estar dentro de cada país y no hay provisiones especiales para esta gobernanza desde una central.

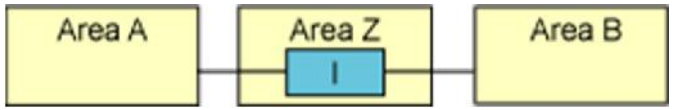
Gobernanza local

Existen interfaces globales para poder publicar una interfaz. El acceso sigue siendo local pero siguiendo reglas globales para proveer el servicio.

Gobernanza Intermedia

La interacción ocurre con un área especial de negocio que facilita y provee interfaces para proveer cierto servicio.

Por ejemplo, finanzas y administración pueden proveer servicios para ciertas otras áreas.

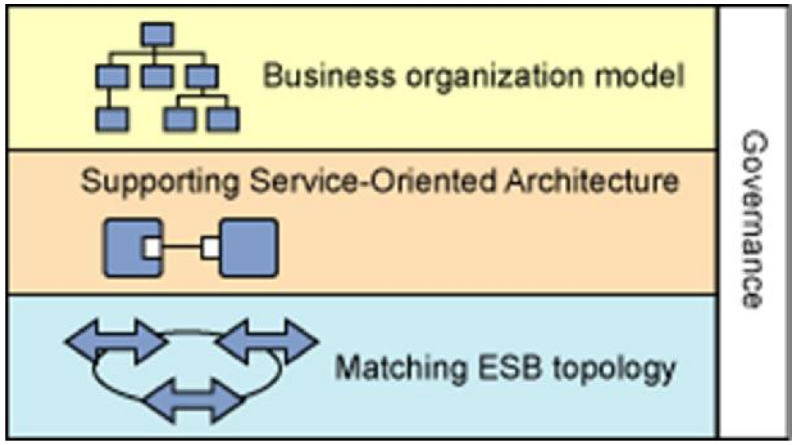


Gobernanza federada

Cada división gobierna sus propias interacciones y posee sus propias interfaces de servicios.

Por ejemplo: una división acepta órdenes y una fábrica realiza la fabricación de las mismas.

Patrones de topologías ESB



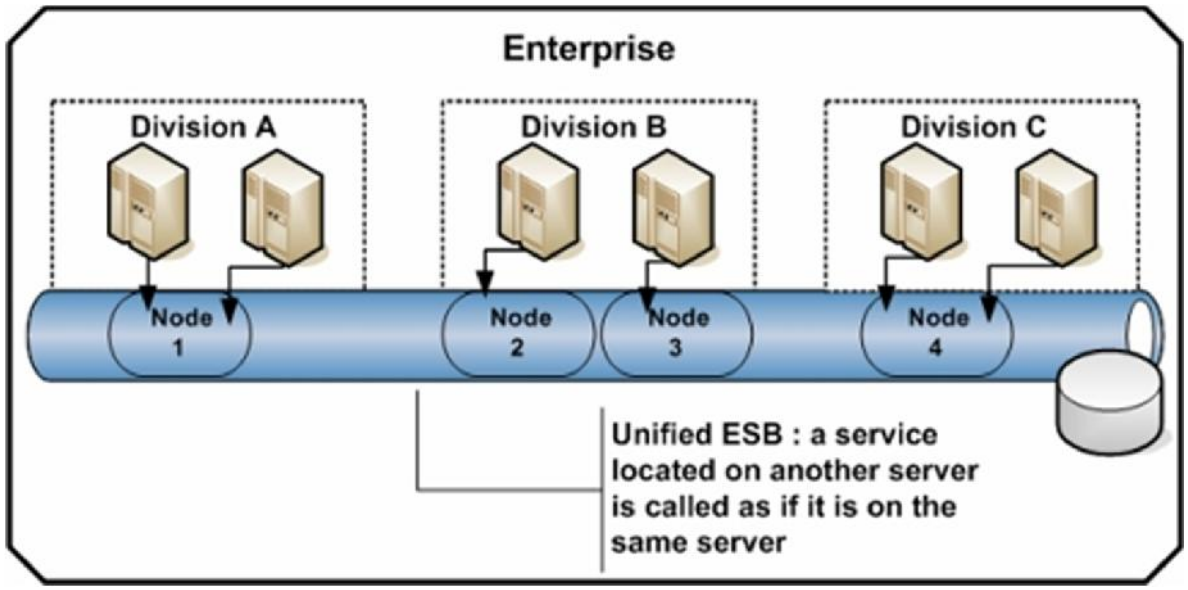
Combinación de patrones según negocio y arquitectura SOA

Único ESB lógico

Compañías que quieren proyectar una presencia única global.

Puede tener uno o más nodos de ESB (para distribución geográfica y alta disponibilidad).

Las instancias de ESB deben enrutar las solicitudes de servicio a cada proveedor, transparentemente.



Único ESB lógico

ESB de conexión directa con registro único

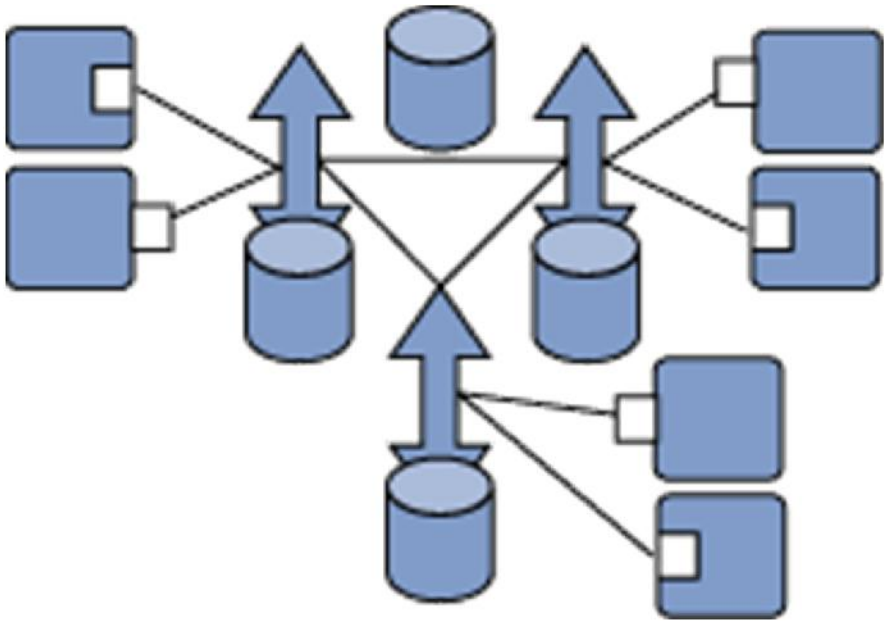
Aplicado en situaciones de múltiples geografías y divisiones.

Los patrones de gobernanza local y federado pueden usarse para tener control de la integraciones.

Cada nodo de ESB hace uso de otro nodo para ciertos requerimientos.

Todos los nodos comparten un único registro.

Cada servicio implica a dos nodos ESB al menos



ESB de conexión directa con registro único

ESB de conexión directa con múltiples registros

Es una variación del anterior, sin un registro central.

Cada unidad de negocio debe definir sus propias reglas para conexión hacia las otras divisiones.

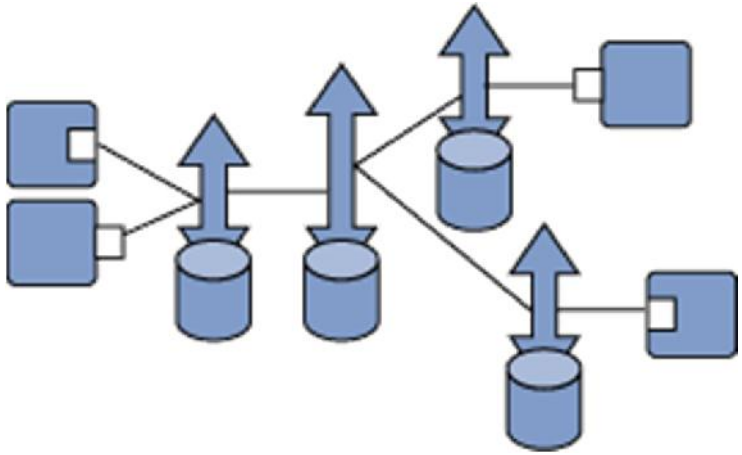
No hay reglas comunes acerca de la sincronización, replicación, partición y transformación de la metadata.

ESB negociado (brokered)

Parte del patrón anterior, pero en vez de acceder directamente de un ESB al otro, se hace a través de negociadores (brokers) que saben a dónde enrutar la información.

Se aplica en patrones de gobernanza intermedia.

Este patrón logra un desacoplamiento en donde se reflejan los cambios de negocio más fácilmente.



ESB negociado (brokered)

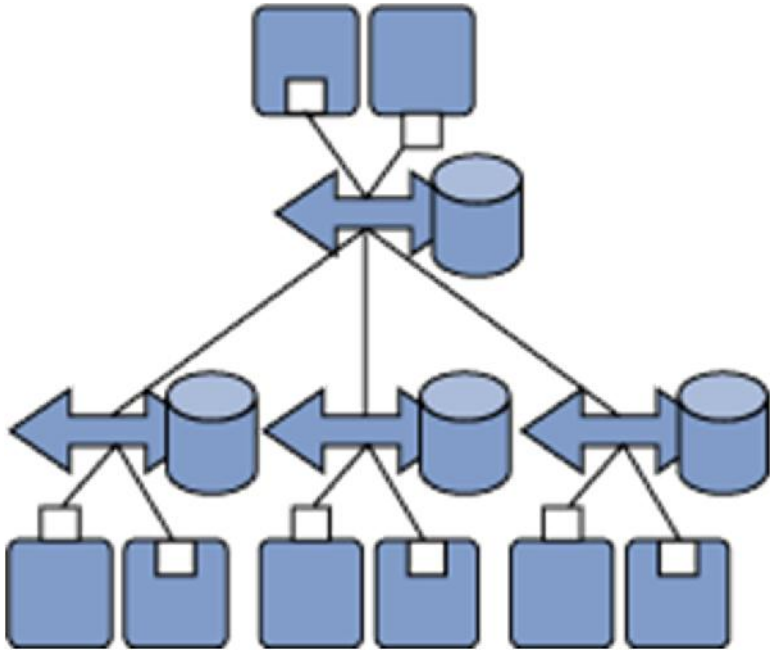
Nodos de ESB (Hub and Spokes ESB)

Se utiliza en negocios con modelos de franquicia, con gobernanza centralizada sobre procesos de negocio.

Cada franquicia puede tener funciones específicas (locales) que debe ejecutar.

Existe en este caso una central ESB que interconecta a las locales para proveer centralización de servicios.

En caso de desconexión, los servicios locales pueden seguirse ofreciendo.



Hub and Spokes ES B