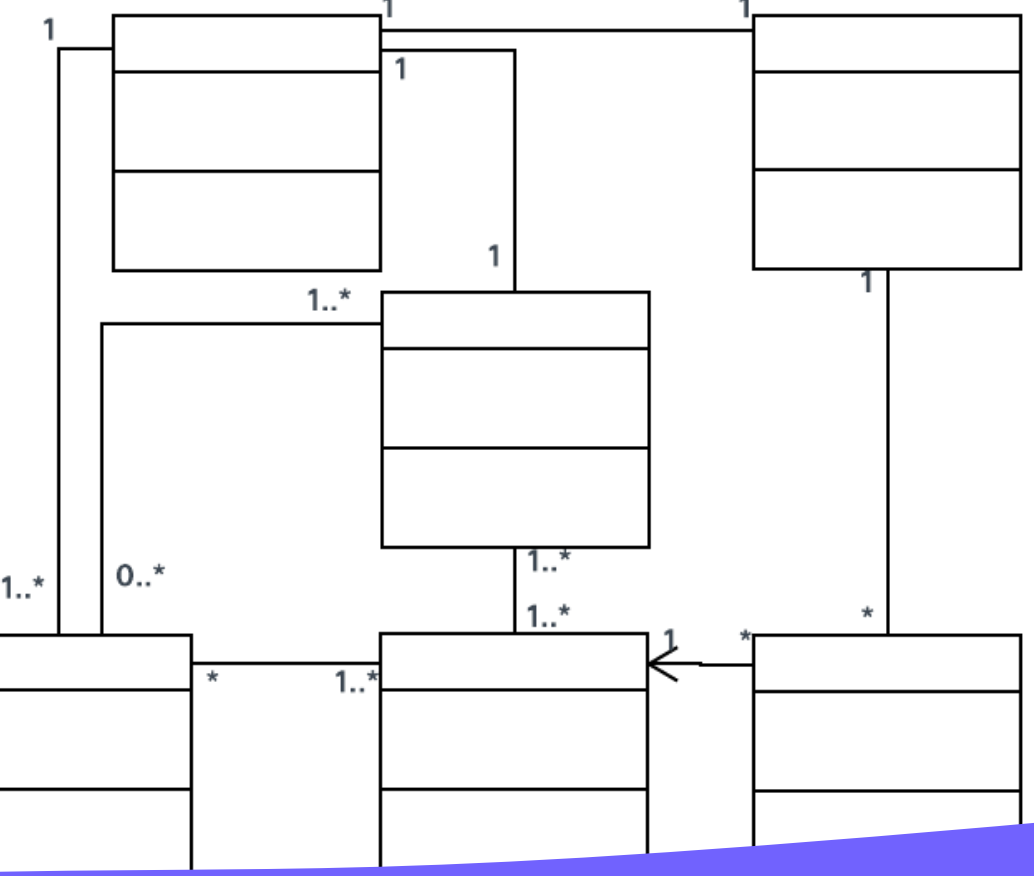
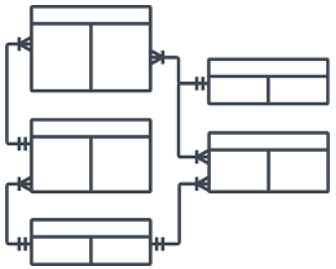




Diseño de Sistemas

UTN.BA
DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Agenda

- Aplicaciones Monolíticas y No Monolíticas
- SOA y Microservicios
- Solicitud – Respuesta
- Subscripción de eventos
- Patrón Publicación – Subscripción
- Patrón Broker
- Pipe & Filters
- P2P

Arquitectura de Software

Aplicaciones Monolíticas y No Monolíticas

Aplicación Monolítica

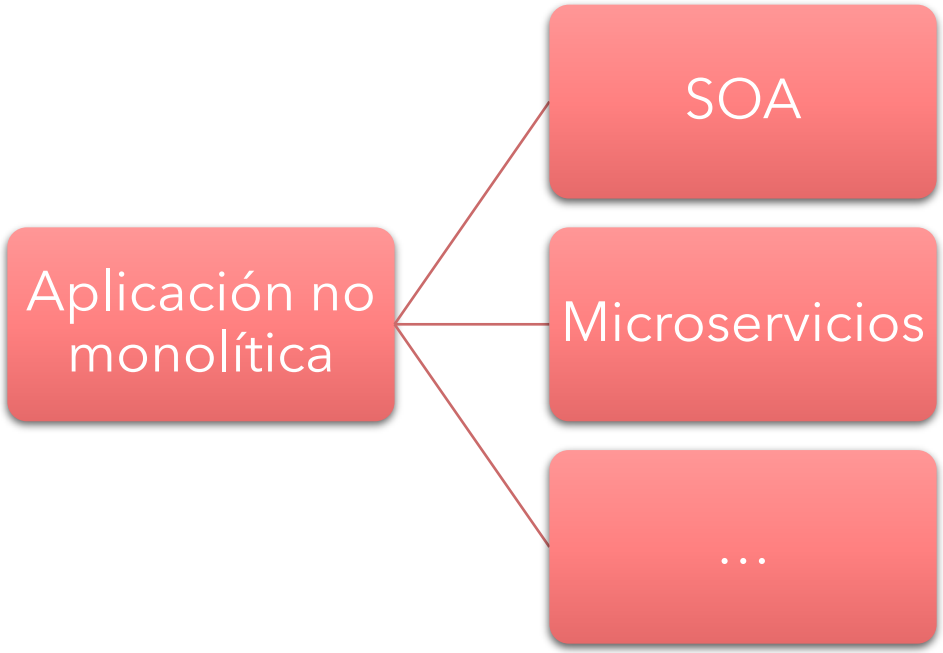
- Las aplicaciones monolíticas o sistemas monolíticos se caracterizan por tener toda la lógica de negocio en un único proyecto de código fuente.
- Las aplicaciones Desktop tradicionales siguen este tipo de arquitectura.
- Si pensamos en un Sistema bajo una Arquitectura Web Monolítica, con capa de presentación visual y un cliente liviano, podemos notar que todos sus componentes están en un único código fuente ejecutándose del lado del Servidor.

Aplicación Monolítica

- Aunque son “fáciles” de desarrollar, no siempre es la mejor opción. Por ejemplo, si existe un gran número de personas trabajando sobre el mismo código fuente, podría volverse inviable.

- Si se realiza algún cambio en el código fuente, en cualquiera de los componentes, todo el aplicativo debe volverse a desplegar por completo.

Aplicación no monolítica



SOA – Service Oriented Architecture

- Es un estilo de arquitectura que promueve descomponer la lógica funcional de una aplicación en unidades autónomas denominadas servicios.

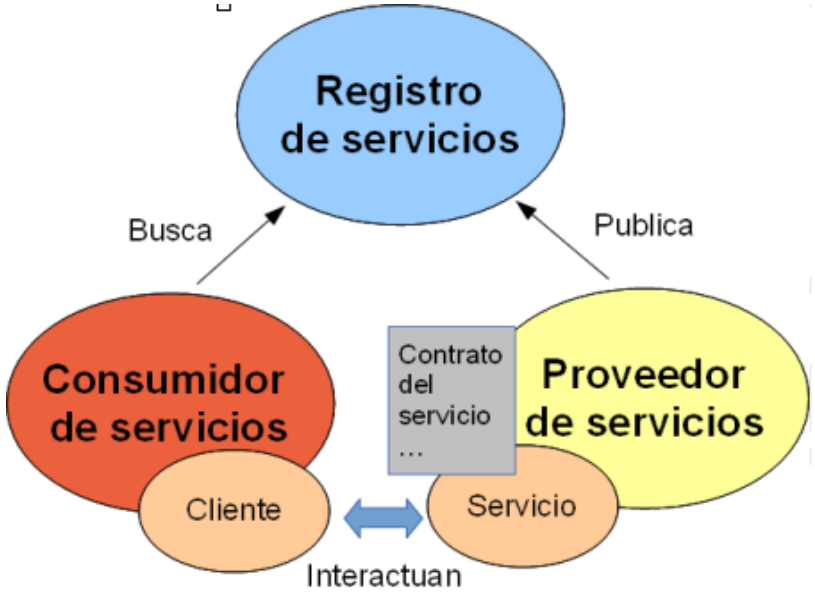
SOA – Service Oriented Architecture

Principios

- El Valor del Negocio por encima de la estrategia técnica.
- Las Metas Estratégicas por encima de los beneficios específicos de los proyectos.
- La Interoperabilidad Intrínseca por encima de la integración personalizada.
- Los Servicios Compartidos por encima de las implementaciones de propósito específico.
- La Flexibilidad por encima de la optimización.
- El Refinamiento Evolutivo encima de la búsqueda de la perfección inicial.

SOA – Service Oriented Architecture

Actores



SOA – Service Oriented Architecture

Componentes

- ***Consumidor de servicios***: Es una aplicación, un módulo de software u otro servicio que demanda la funcionalidad proporcionada por un servicio.

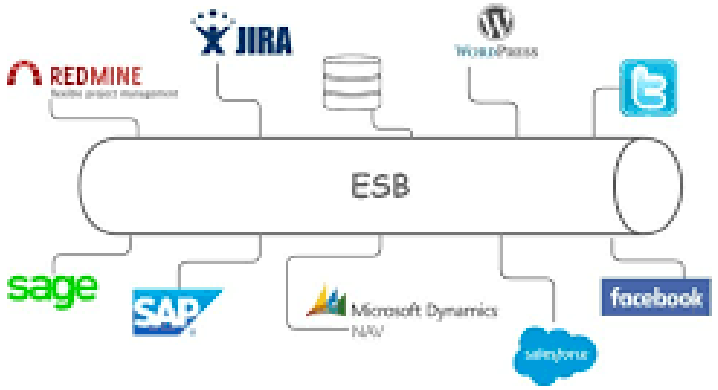
- ***Proveedor de servicios***: Es una entidad accesible a través de la red que acepta y ejecuta consultas de consumidores y publica sus servicios y su contrato de interfaces en el registro de servicios para que el consumidor pueda descubrir y acceder al servicio.

- ***Registro de servicios***: Es un repositorio de servicios disponibles. Permite visualizar las interfaces de los proveedores de servicios a los consumidores interesados (de dichos servicios)

SOA – Service Oriented Architecture

Sumemos un Componente… ¿Que es un ESB?

Un Bus de Servicio Empresarial (ESB por sus siglas en inglés) es un componente de arquitectura de software que gestiona la comunicación entre múltiples servicios web. Se enfoca en resolver el problema que surge cuando los servicios web dentro de una organización se multiplican lo que hace necesario desarrollar conectores que permitan comunicar las diferentes aplicaciones.



SOA – Service Oriented Architecture

ESB

- Los servicios no interactúan directamente, sino que la comunicación es a través de un conector. El ESB proporciona la virtualización de los servicios.

- ***Ubicación e Identidad***: El ESB identifica y establece las rutas de los mensajes entre los servicios, de manera que éstos no tienen porque conocer la ubicación o la identidad de otros participantes en la comunicación.

- ***Protocolo de comunicación***: El ESB permite el flujo de mensajes a través de diferentes protocolos de transporte o los estilos de interacción (HTTP, FTP, SMTP).

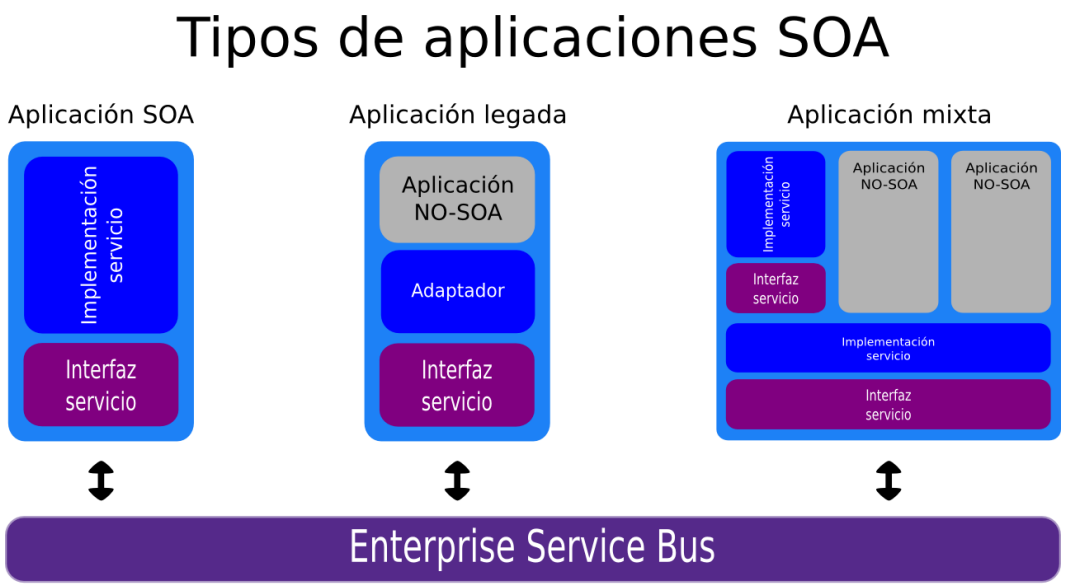
SOA – Service Oriented Architecture

ESB – Algunas implementaciones

- OpenESB
- Oracle ESB
- Azure Service Bus
- IBM WebSphere ESB
- IBM WebSphere Integration Bus (IBM WebSphere Message Broker)
- JBoss Fuse
- Spring Integration
- Apache ServiceMix

SOA – Service Oriented Architecture

Tipos de Aplicaciones



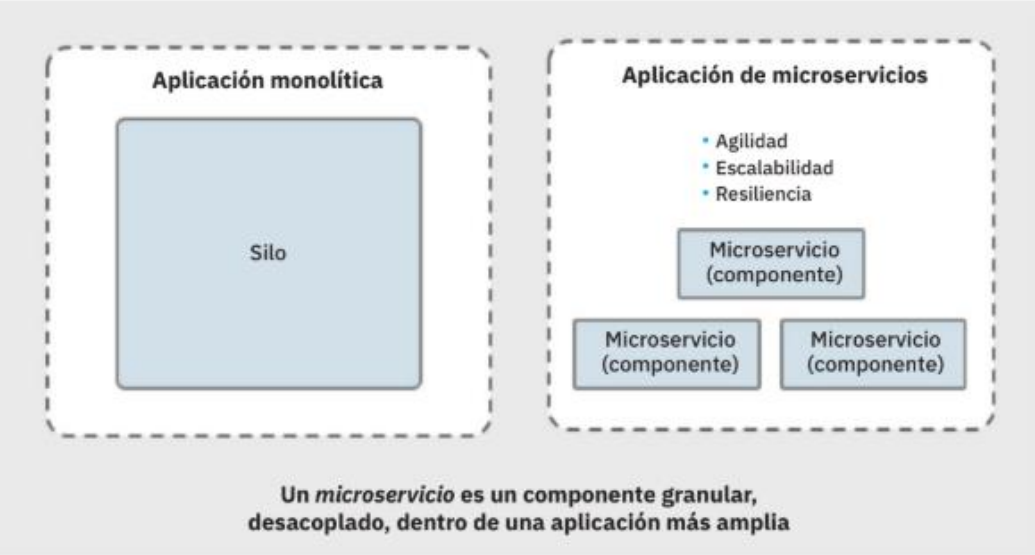
Microservicios

Es un **estilo arquitectónico** con un enfoque para desarrollar **una única aplicación** como un **conjunto de pequeños servicios**, cada uno de ellos ejecutándose en su **propio proceso** y **comunicándose** con **mecanismos ligeros**, a menudo una **API REST**.

Estos **servicios** se construyen alrededor de **capacidades empresariales** y para ser **desplegados independientemente** por herramientas de despliegue totalmente **automatizadas**.

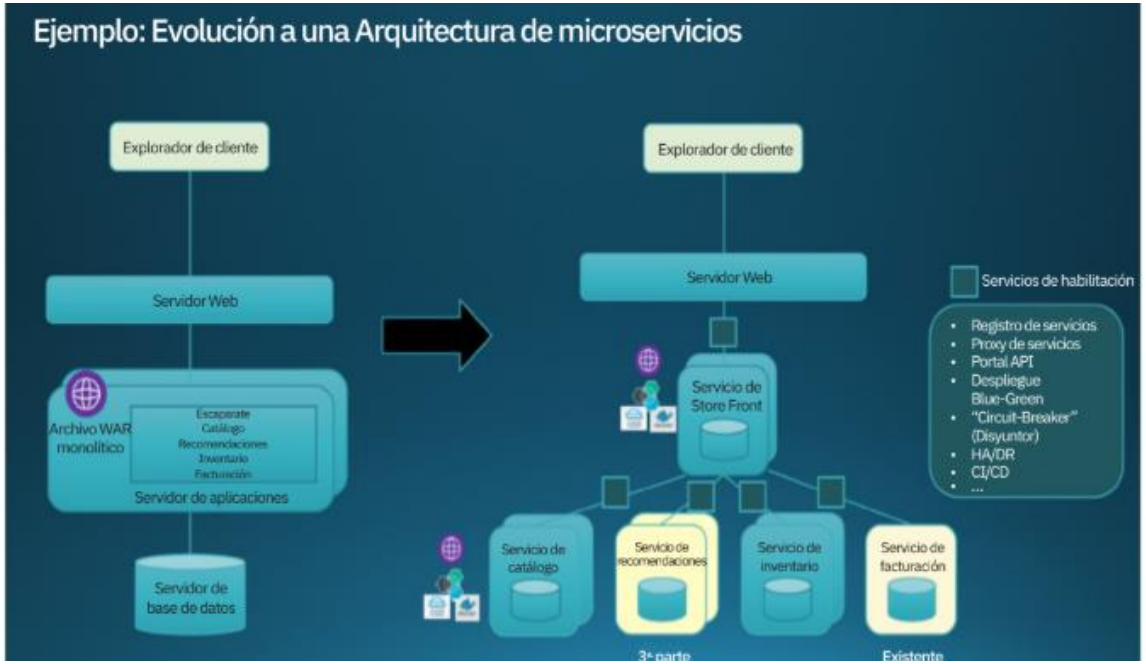
Hay un **mínimo** absoluto de **gestión centralizada** de estos servicios, que **pueden** ser **escritos** en **diferentes lenguajes de programación** y usar **diferentes tecnologías de almacenamiento de datos**.

Microservicios vs Monolítica

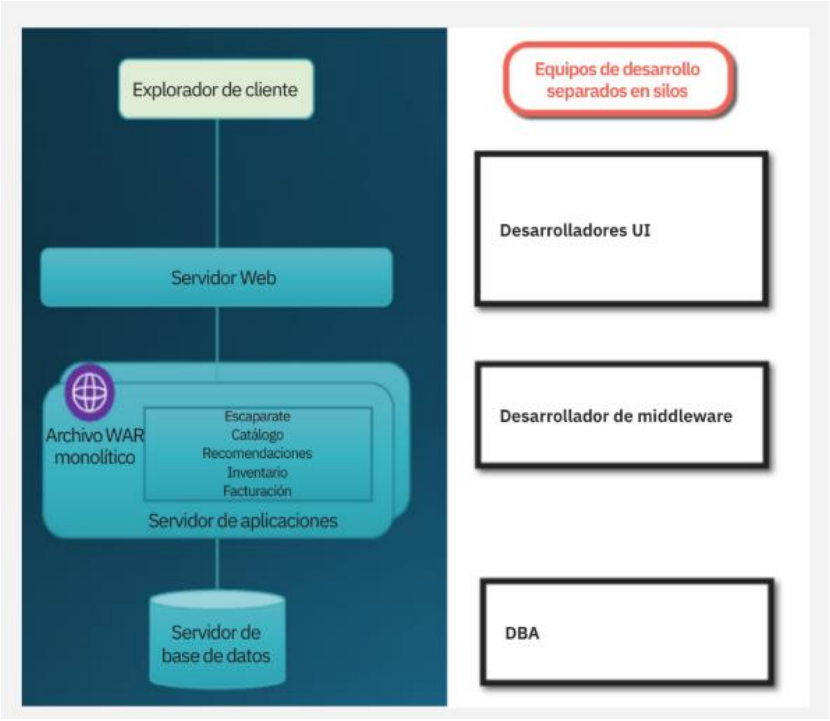


Un *microservicio* es un componente granular, desacoplado, dentro de una aplicación más amplia

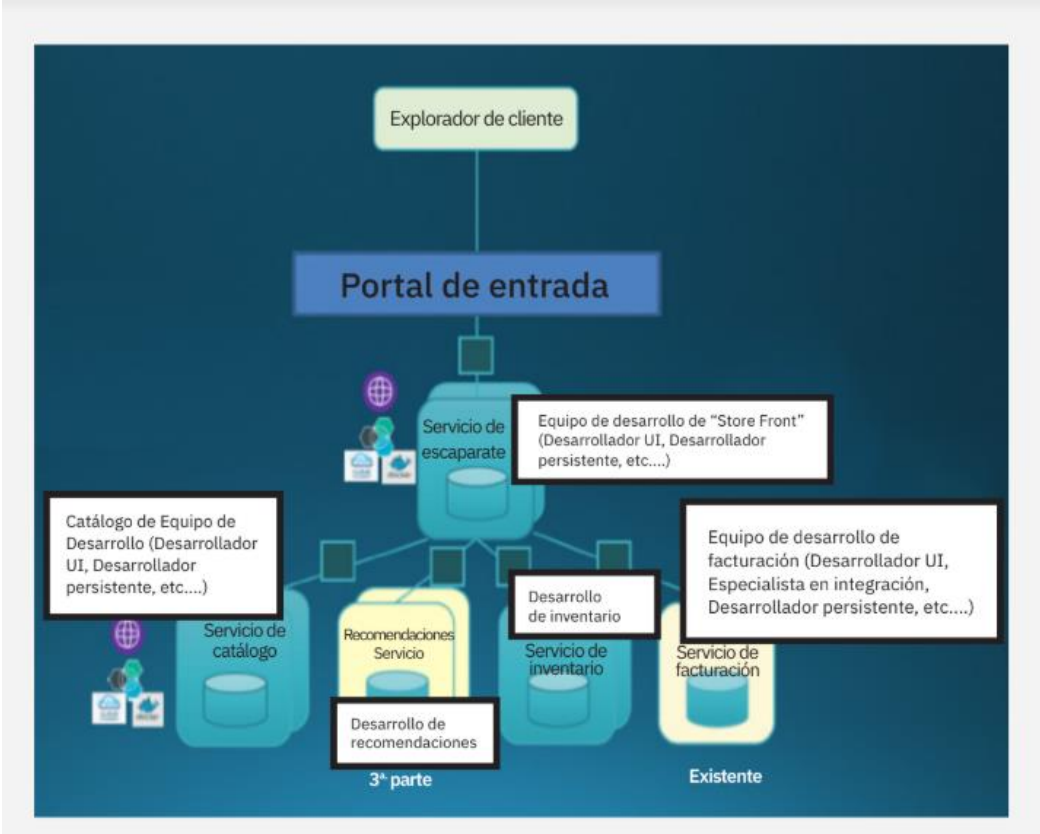
Microservicios vs Monolítica



Microservicios vs Monolítica



Microservicios vs Monolítica



Microservicios - Características

- En una **arquitectura de microservicios**, los **servicios** son **pequeño**s e **independientes** y están **débilmente acoplados**.
- Cada **servicio** es un **código base independiente**, que puede **administrarse** por un **equipo** de **desarrollo pequeño**.
- Los **servicios** pueden implementarse de **manera independiente**. Un equipo puede **actualizar** un **servicio existente** sin tener que **volver** a "compilar" y desplegar **toda la aplicación**.
- Los **servicios** son los **responsables** de **conservar** sus **propios datos** o **estado externo.**
- Los **servicios** se **comunican** entre sí mediante **API bien definidas**.
- **No** es **necesario** que los **servicios compartan** la misma **pila de tecnología.**

Microservicios – Consideraciones para el desarrollo

Separar grandes monolitos en muchos servicios pequeños
Un solo servicio accesible en la red es la unidad a desplegar más pequeña para una aplicación de microservicios. Cada servicio ejecuta su propio proceso. Esta regla se denomina un servicio por contenedor.

Optimizar los servicios para una función única
Una sola función empresarial por servicio. Esto hace que cada servicio sea más pequeño y simple para escribir y mantener. Esto se denomina como el Principio de responsabilidad única (SRP).

Comunicarse a través de las API REST y los Buses de Mensajes
Los microservicios tienden a evitar el estrecho acoplamiento introducido por la comunicación implícita a través de una base de datos. Toda la comunicación entre un servicio y otro debe hacerse a través de la API del servicio o componentes intermedios con los Buses de Mensajes.

Microservicios – Consideraciones para el desarrollo

Aplicar CI/CD por servicio

En una **gran aplicación** compuesta por muchos servicios, los diferentes servicios **evolucionan** a **velocidades diferentes**.

Cada servicio tiene una única y continua tubería de permisos de integración/entrega que avanza a un ritmo natural.

Esto no es posible con el enfoque monolítico, donde cada aspecto del sistema es liberado forzosamente a la velocidad de la parte del sistema que se mueve más lentamente.

Aplicar decisiones de alta disponibilidad (HA) / "clustering" por servicio

En el enfoque monolítico se escalan todos los servicios del monolito al mismo nivel y esto hace que se usen en exceso los recursos. La realidad es que en un sistema grande, no todos los servicios necesitan ser ampliados, muchos pueden ser desplegados en un número mínimo de servidores para conservar los recursos. Otros requieren ser ampliados a cantidades muy grandes.

Microservicios – Componentes necesarios

Administración. El componente de administración es responsable de la colocación de servicios en los nodos, la identificación de errores, el reequilibrio de servicios entre nodos, etc.

Detección de servicios. Mantiene una lista de servicios y los nodos en que se encuentran. Permite la búsqueda de servicios para localizar el punto de conexión de un servicio.

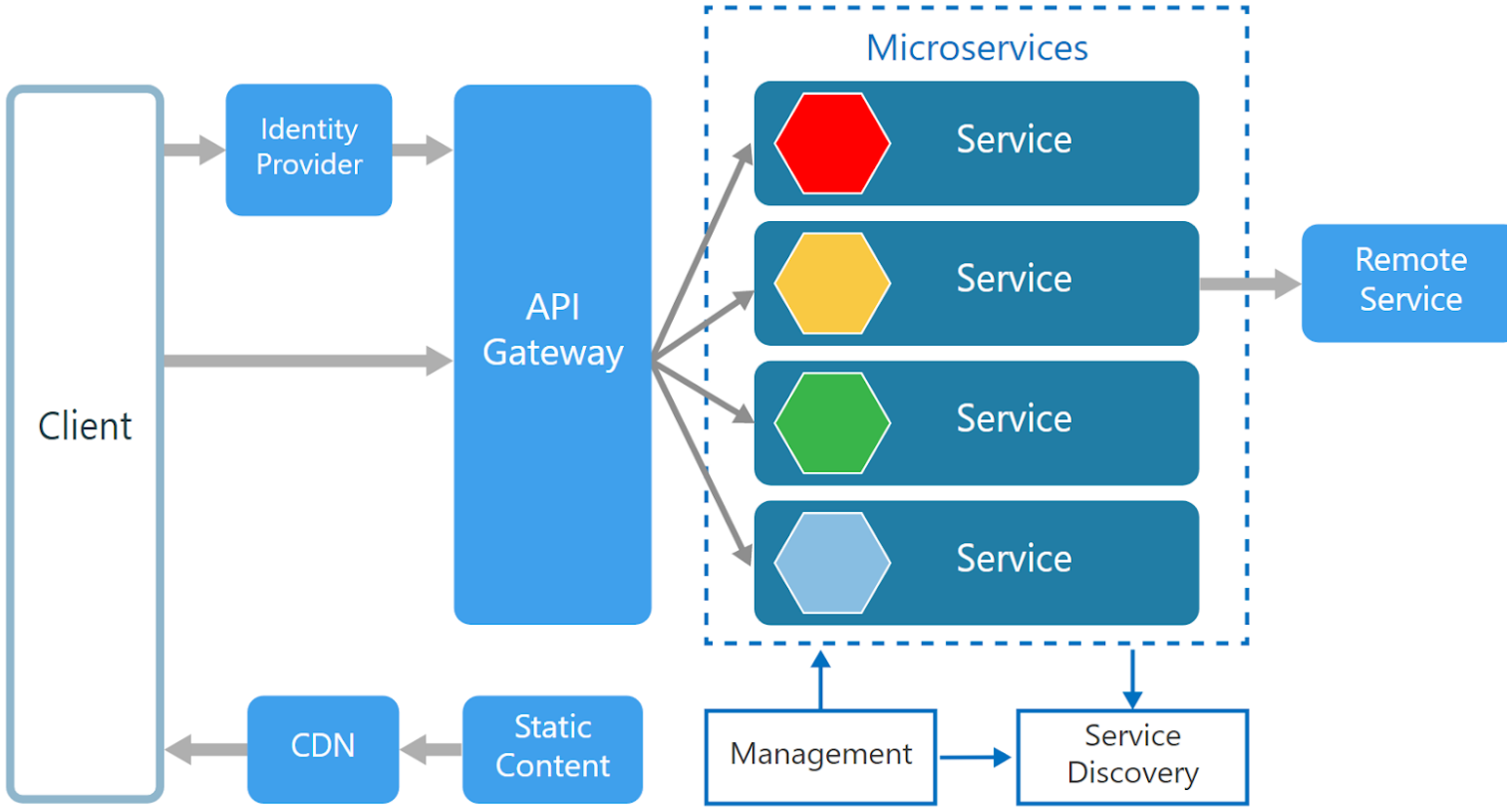
Puerta de enlace de API es el punto de entrada para los clientes. Los clientes no llaman directamente a los servicios. En su lugar, llaman a la puerta de enlace de API, que reenvía la llamada a los servicios apropiados en el back-end. La puerta de enlace de API podría agregar las respuestas de varios servicios y devolver la respuesta agregada. La puerta de enlace de API puede realizar otras funciones transversales como la autenticación, el registro, la terminación SSL y el equilibrio de carga.

Microservicios – Opciones Despliegue

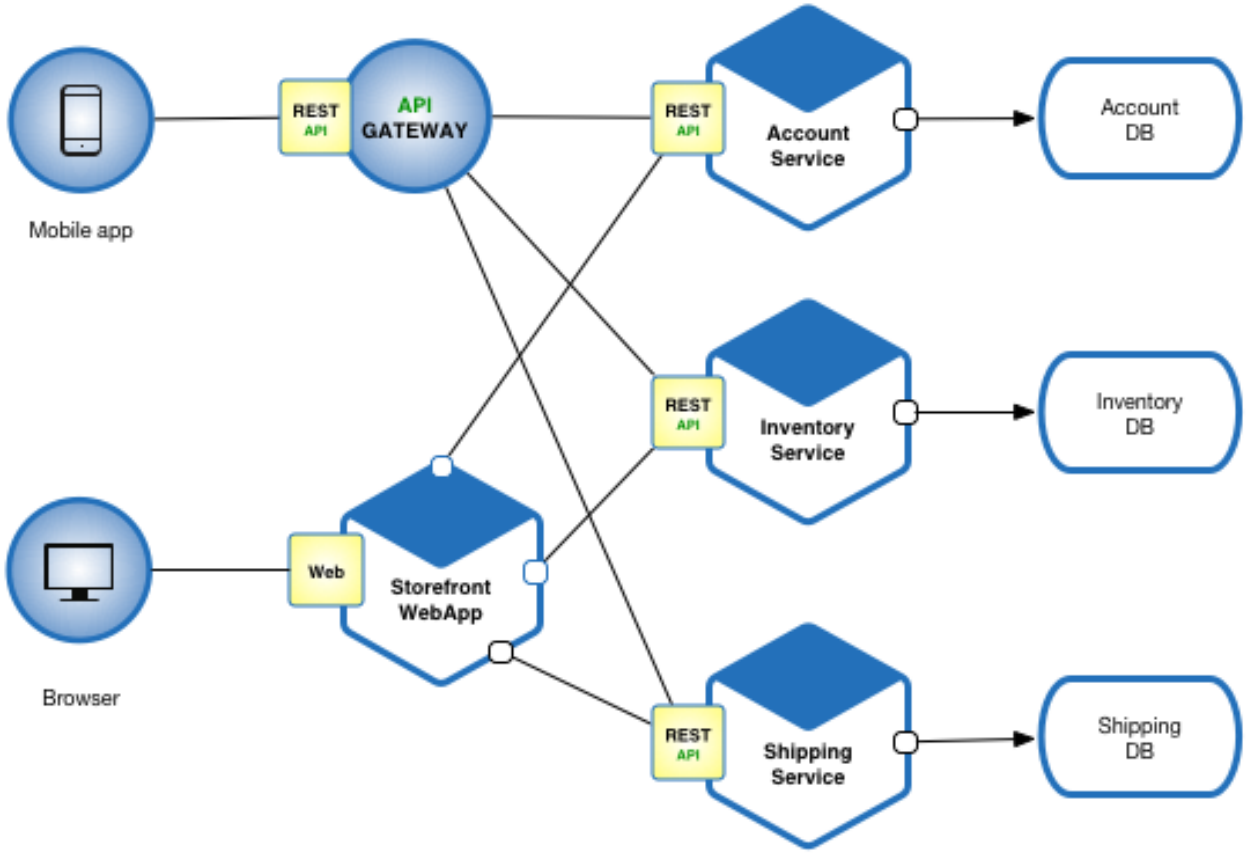
Microservicios – ¿Cuándo utilizarlos?

- Aplicaciones grandes que requieren una alta velocidad de publicación.
- Aplicaciones complejas que necesitan gran escalabilidad.
- Aplicaciones que requieren escalar de forma dinámica y es difícil predecir cuando sucederá el evento que lo requiere.
- Aplicaciones requieren crecer rápidamente a nivel funcional y estos cambios son sostenidos en el tiempo.
- Aplicaciones con dominios complejos o muchos subdominios.
- Una organización que disponga de pequeños equipos de desarrollo.

Microservicios – Arquitectura



Microservicios – Arquitectura



Microservicios – Algunos Patrones

El **Patrón fachada** define una interfaz a través de una **API externa** específica para un **sistema o subsistema**.

Los **Patrones de Entidad (Entity)** y **Agregado (Aggregate)** son útiles para identificar **conceptos específicos** de la **empresa** que se relacionan directamente con **microservicio.**

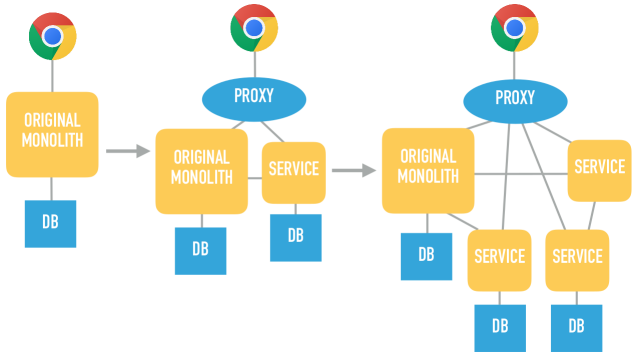
El **Patrón de Servicios** ofrece una manera de **mapear operaciones** que **no corresponden a una sola entidad o agregado** en el enfoque basado en entidad que se requiere para los microservicios.

https://martinfowler.com/bliki/DDD_Aggregate.html

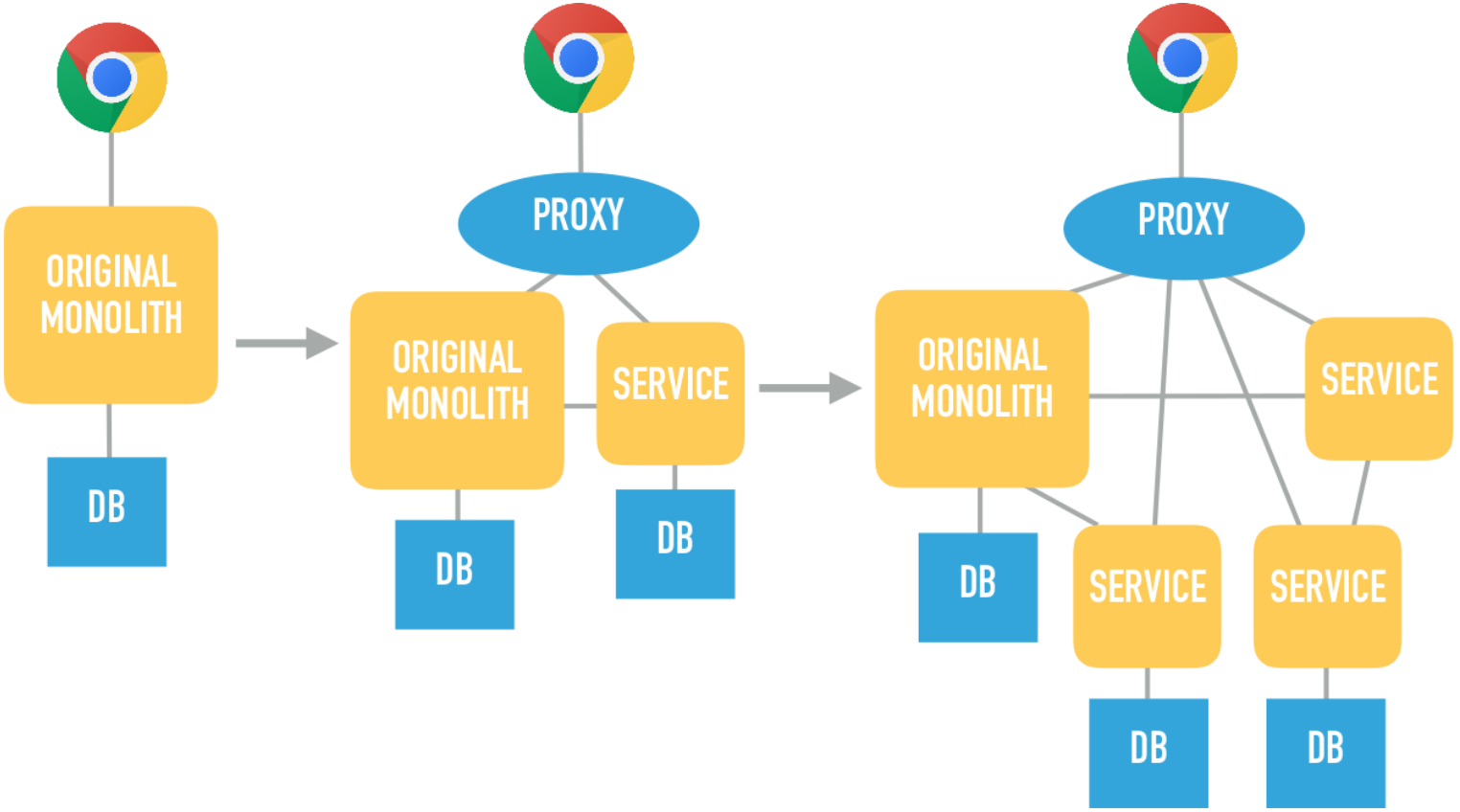
Microservicios – Algunos Patrones

El **Patrón Adaptador de Microservicios** (Microservicios adaptadores) adapta entre dos API **diferentes** (conceptualmente). Se **adapta** una **API heredada o servicio tradicional** SOAP a una **API RESTful** o técnicas de mensajería ligeras.

El **Patrón Strangler (estrangulador)** aborda el hecho de realizar una **reingeniería** de una **aplicación monolítica** y orientarla a **microservicios**. El patrón brinda un enfoque para **gestionar** la **refactorización** y avanzar la **migración**.



Microservicios – Patrón Strangler



Microservicios – Patrones de Operaciones

El **Patrón "Service Registry"** (Registro de servicio) está vinculado a **identificar servicios** y poder **intercambiarlos** de manera ágil.

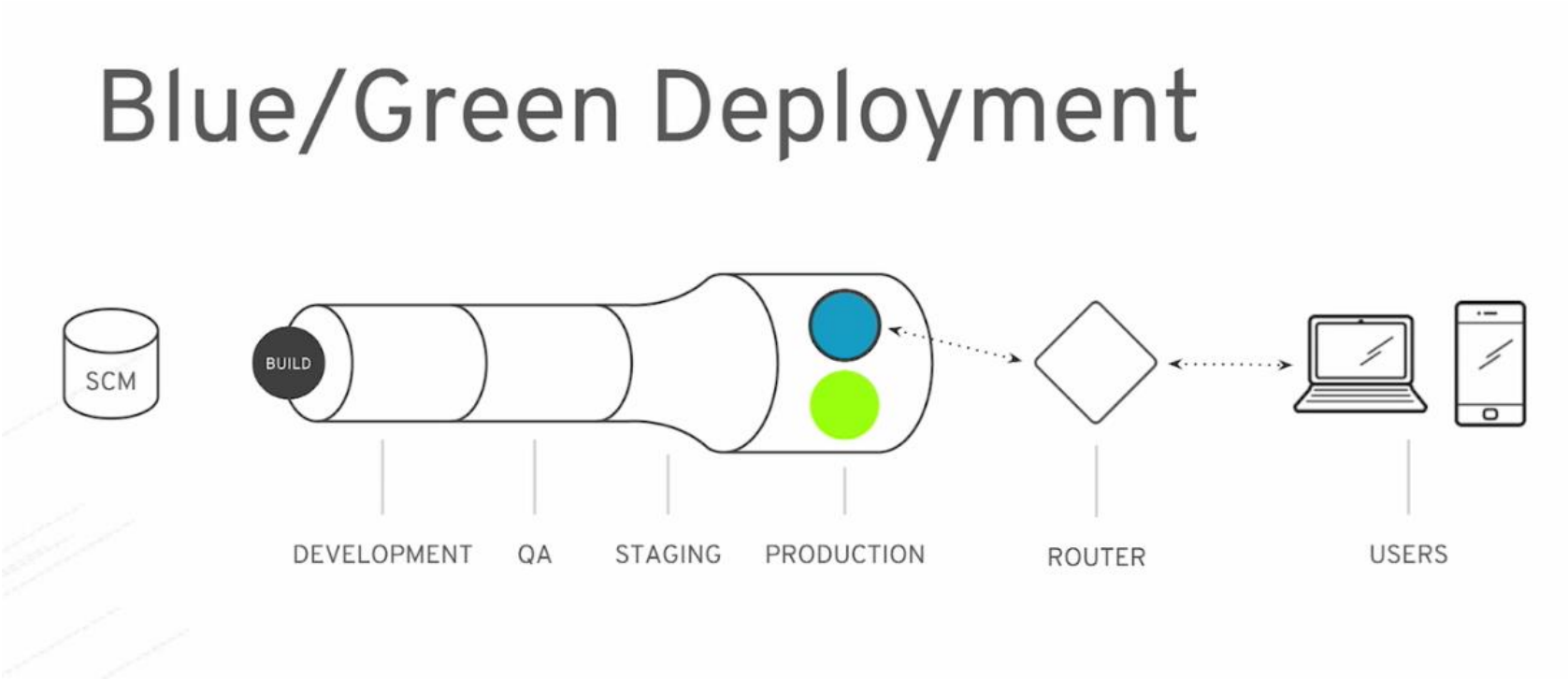
•

Los **Patrones "Correlation ID"** y ***"Log Aggregator"*** *resuelve problemas de* **trazabilidad, descubrimiento y seguimiento de errores**.

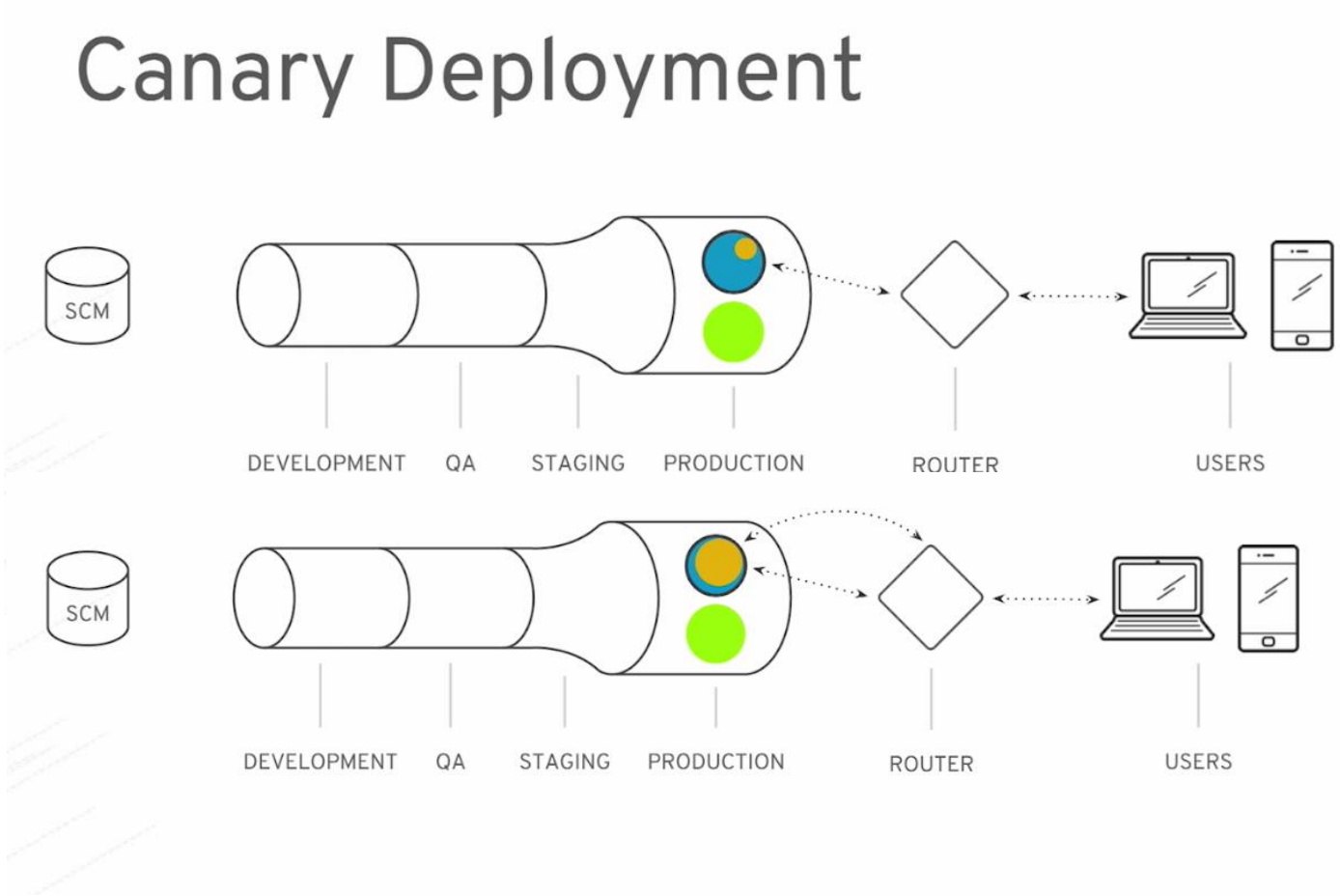
•

El **Patrón "Circuit Breaker"** está vinculado a **detecta**r y **cortar** el llamado a **servicios** que están **funcionando** de manera **incorrecta**.

Microservicios – Patrones



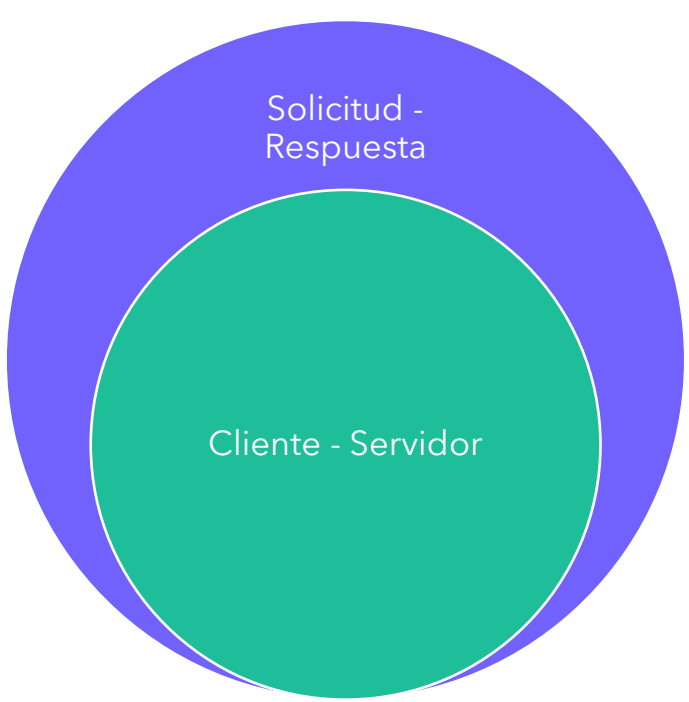
Microservicios – Patrones

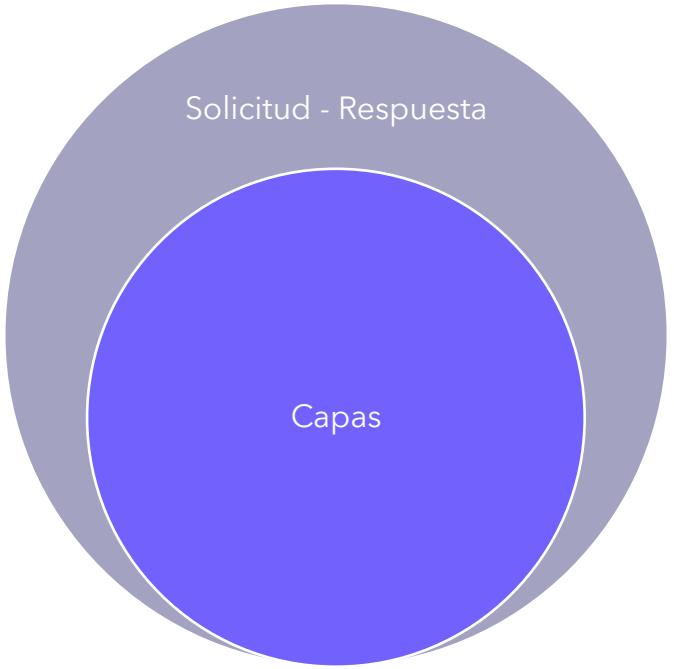


Arquitectura de Software

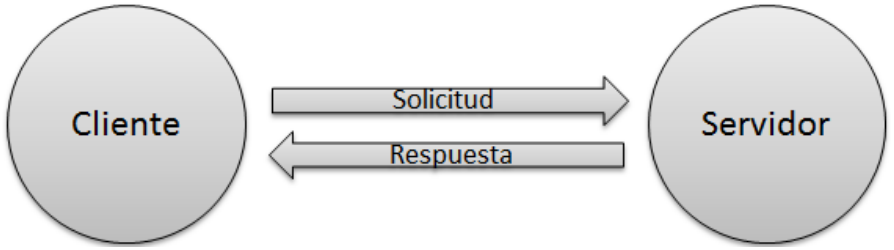
Patrones arquitectónicos tipo "Call & Return"

Estilo Solicitud – Respuesta (Call & Return)

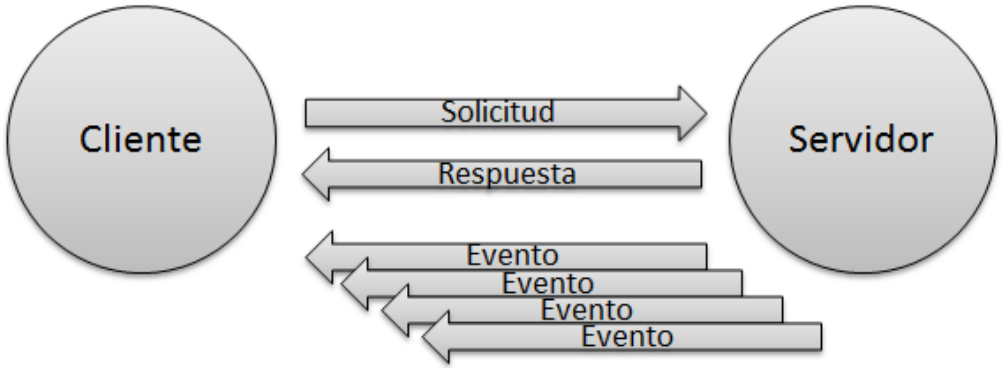




Solicitud - Respuesta

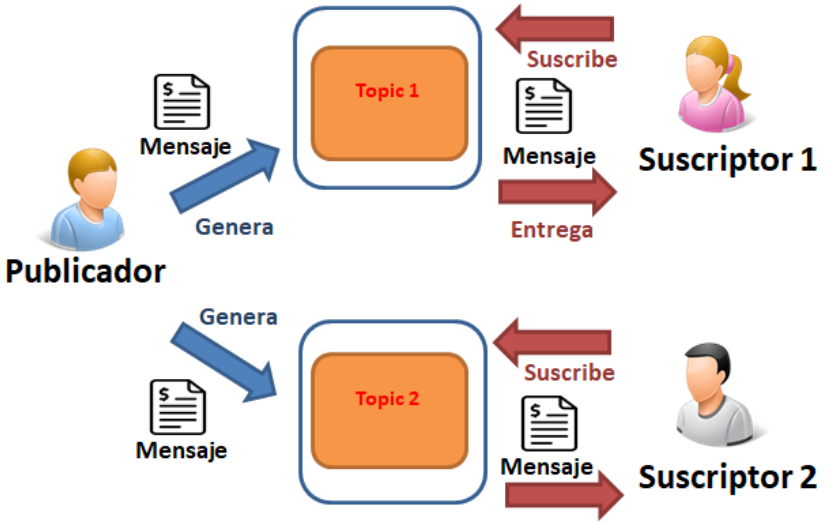


Patrón Subscripción de Eventos



Patrón Publicación - Subscripción

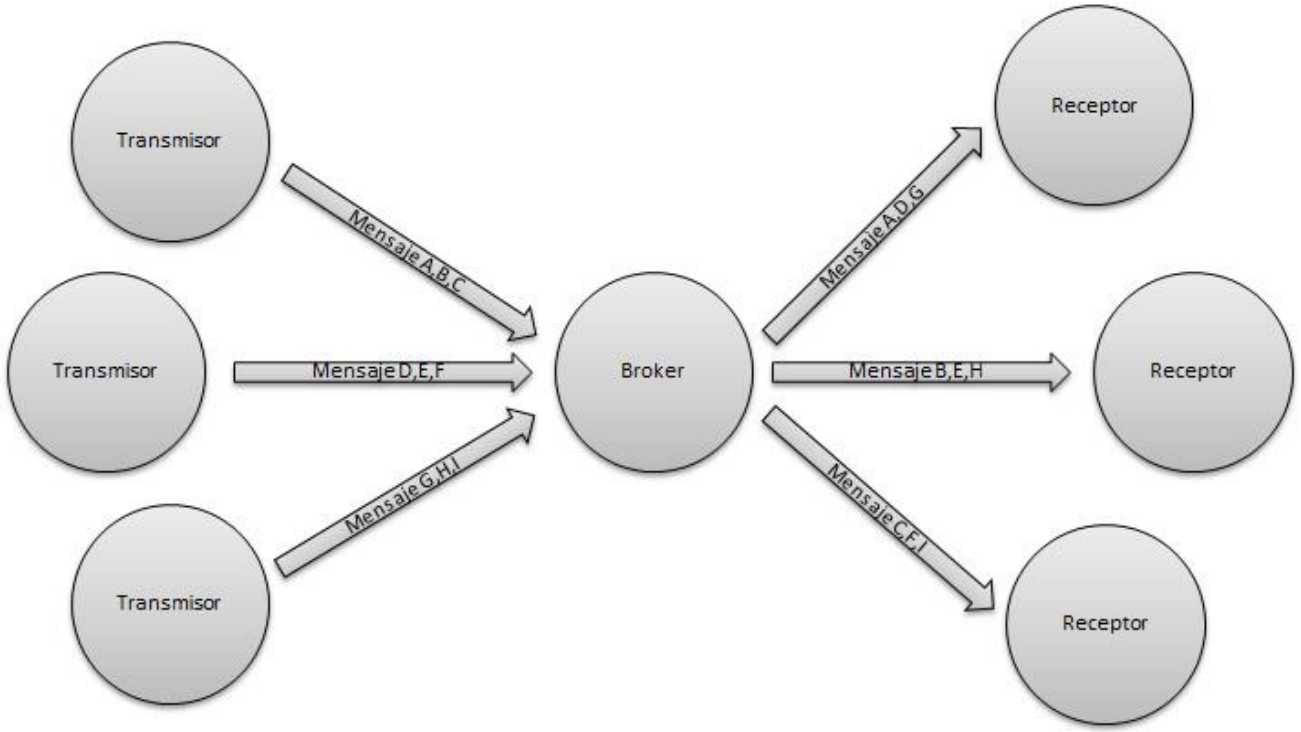
- Es un Patrón de Mensajería
- Define un Subscriptor y un Publicador
- Define el concepto de Topic (tema)
- Minimiza el Acoplamiento (Cliente - Servidor)



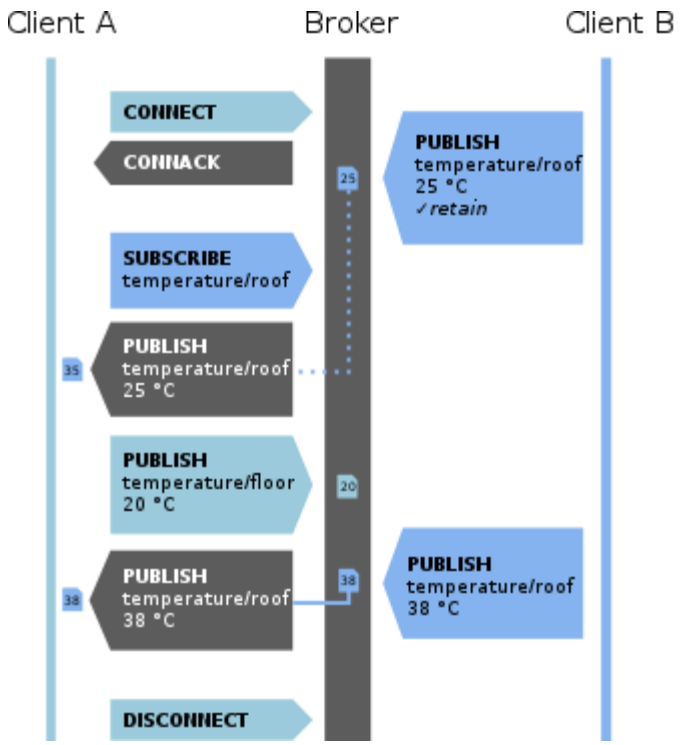
Patrón Broker

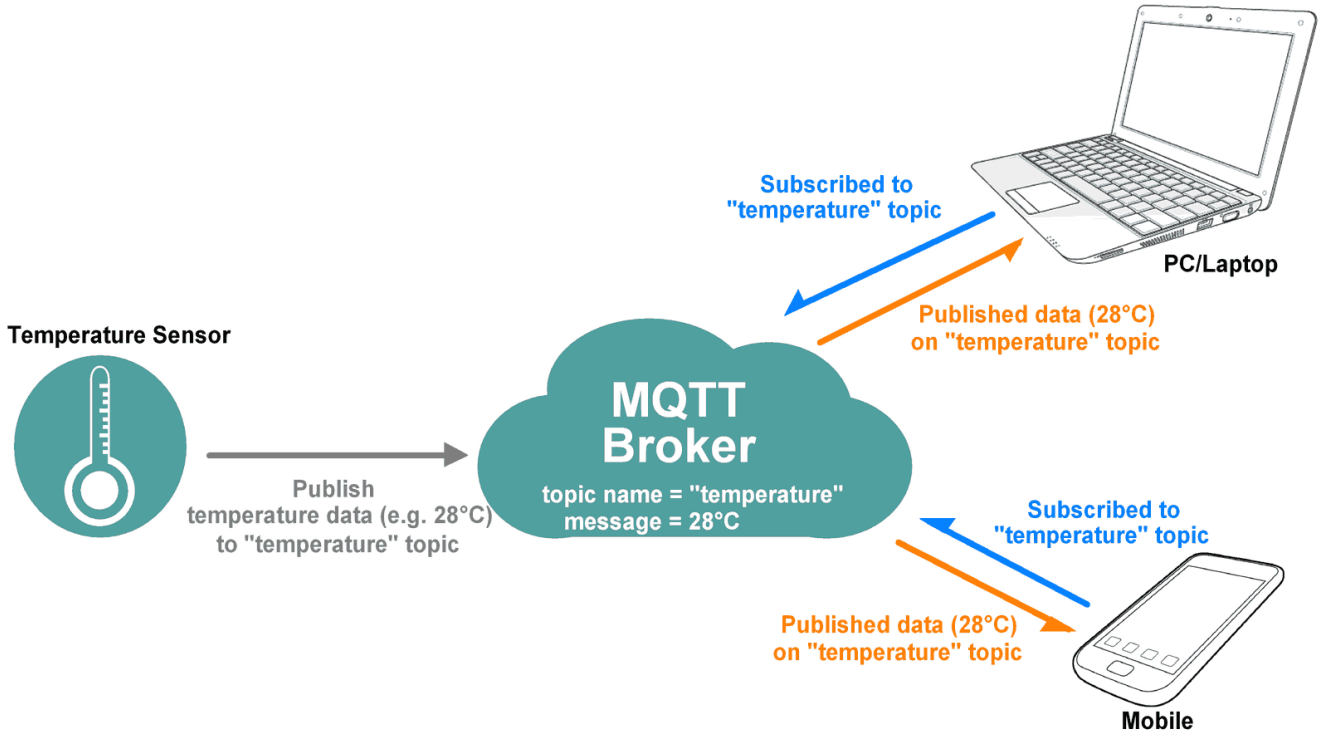
- Intermediario
- Implementa Publicación / Subscripción
- Iguala los Participantes
- Resuelve problemas de Firewall
- Servicios de Autenticación

Patrón Broker



Patrón Broker - Ejemplo





Patrón Broker - Ejemplo

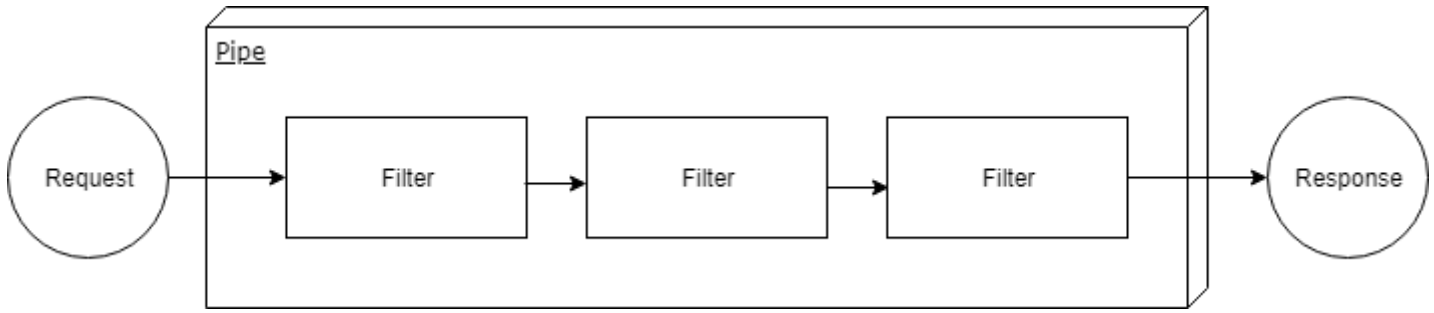
1. Descargar Aplicación MQTT Dash o Similar
2. Conectarse al Broker.
    1. Dirección del Broker: broker.hivemq.com:1883
3. Subscribirse al topic:
    1. dds2023/temperatura
    2. dds2023/humedad

Arquitectura de Software

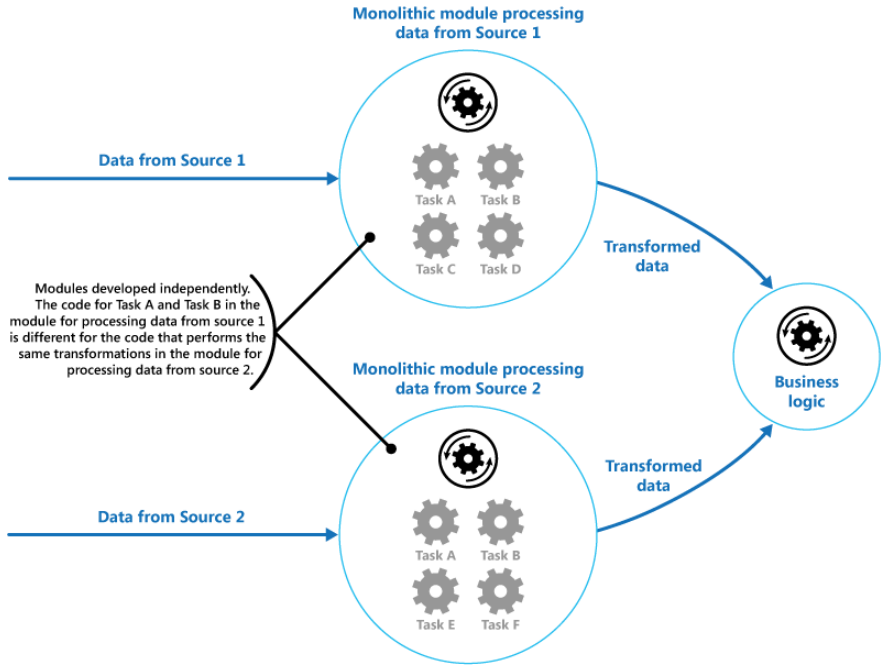
Patrones arquitectónicos tipo “Flujo de Datos”

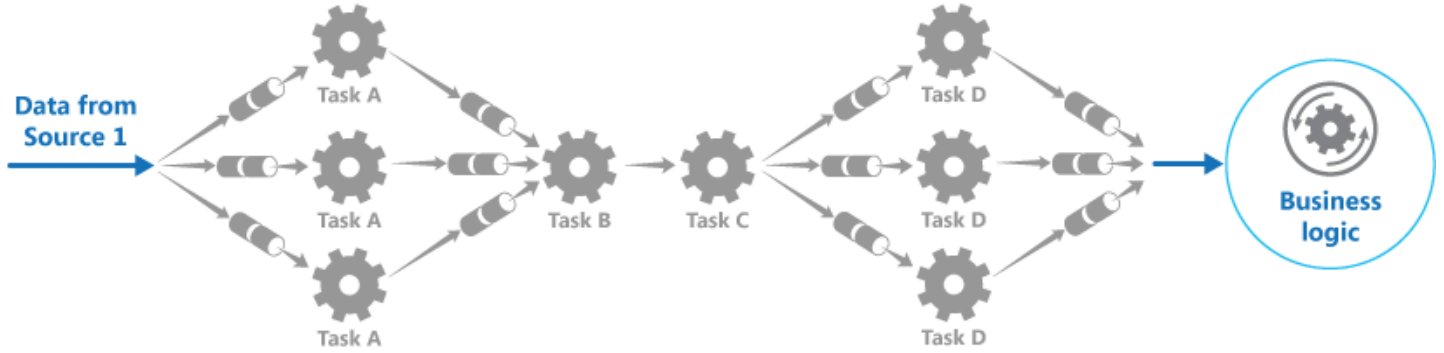
Pipe & Filters

Descompone una tarea que realiza un procesamiento complejo en una serie de elementos independientes que se pueden volver a utilizar. Este patrón puede mejorar el rendimiento, la escalabilidad y la capacidad de reutilización al permitir que los elementos de tarea que realizan el procesamiento se implementen y escalen por separado.

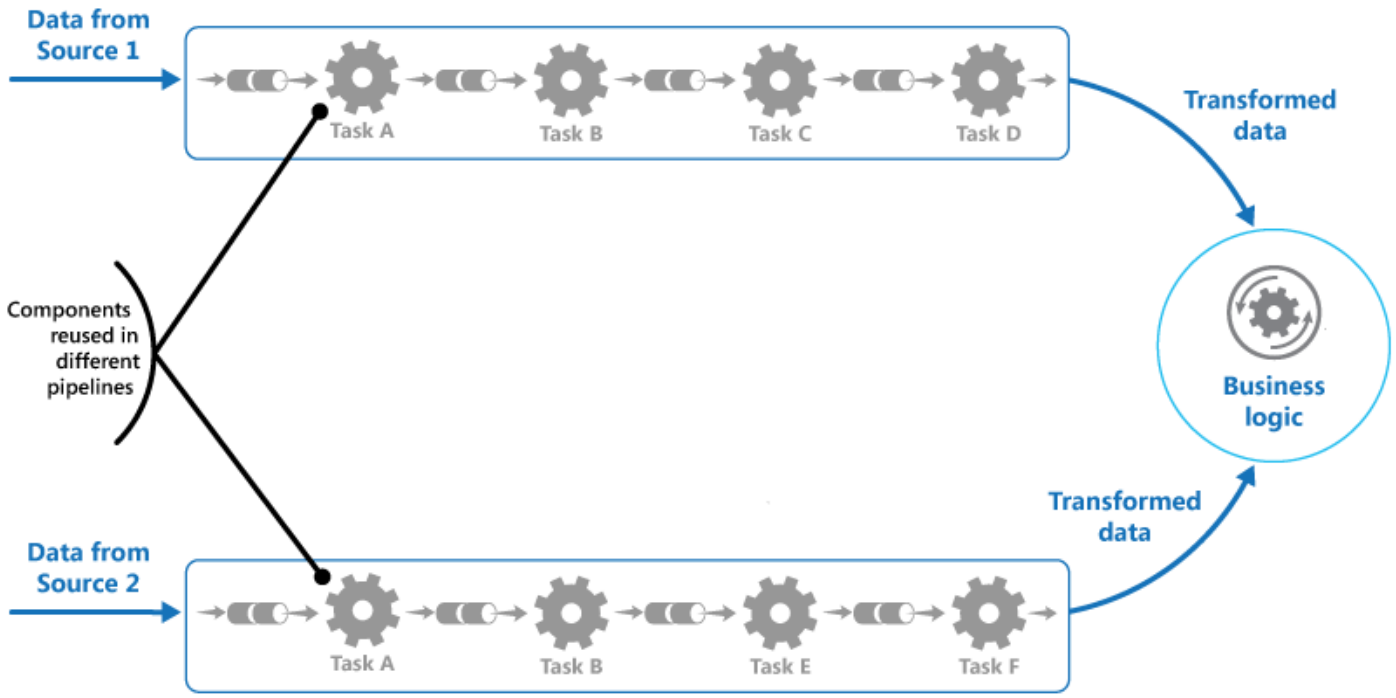


Pipe & Filters





Pipe & Filters



P2P – Peer to Peer

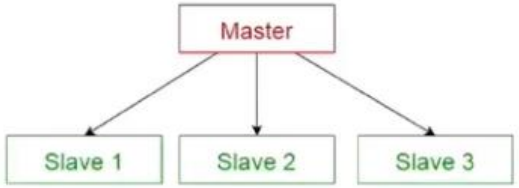
- Cualquier par (peer) es libre de comunicarse directamente con cualquier otro par, sin necesidad de utilizar un servidor central.
- Los “pares” o compañeros suelen localizarse entre sí mediante el intercambio de listas de forma automática entre compañeros conocidos.
- Cada par es capaz de actuar como tanto el cliente (al hacer solicitudes) o como servidor (cuando atienden solicitudes), siendo habitualmente ambos al mismo tiempo.

Ejemplos de P2P

- Apache Cassandra
- ….

- Es distribuida
- Escala linealmente

- No sigue un patrón maestro servidor. Es P2P



- Escalabilidad horizontal

Scale Up- Vertical Scaling

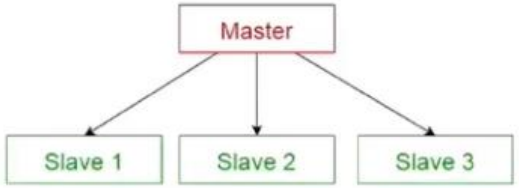
Scale Out- Horizontal Scaling

Ejemplos de P2P

- Apache Cassandra
- Blockchain

- Es distribuida
- Escala linealmente

- No sigue un patrón maestro servidor. Es P2P



- Escalabilidad horizontal

Scale Up- Vertical Scaling

Scale Out- Horizontal Scaling

Ejemplos de P2P

Descentralización. No hay un nodo central que maneje nodos secundarios, todos tienen el mismo rol dentro de un clúster y pueden dar respuesta a cualquier solicitud.

Replicación. Diseñada para el despliegue en gran cantidad de nodos, permite incluso la replicación en diversos centros de datos.

Tolerancia a fallos. cuando un nodo no funciona la base de datos sigue trabajando normalmente con los restantes, se pueden añadir y quitar nodos sin detener el servicio.

Escalabilidad. Es una de las bases de datos más escalables y para conseguirlo únicamente se deben instanciar más nodos.

Gracias