



Evolución de las arquitecturas de software

de los monolitos a los servicios

Evolución de las arquitecturas de software

Contenidos

- Evolución
- Arquitectura monolítica
- Arquitectura Cliente / Servidor
- Arquitectura N-capas
- Arquitectura Orientada a Servicios
- Arquitecturas basadas en microservicios

Evolución

THE EVOLUTION OF SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED
ARCHITECTURE
(aka Copy & Paste)



2000's

LASAGNA-ORIENTED
ARCHITECTURE
(aka Layered Monolith)



2010's

RAVIOLI-ORIENTED
ARCHITECTURE
(aka Microservices)



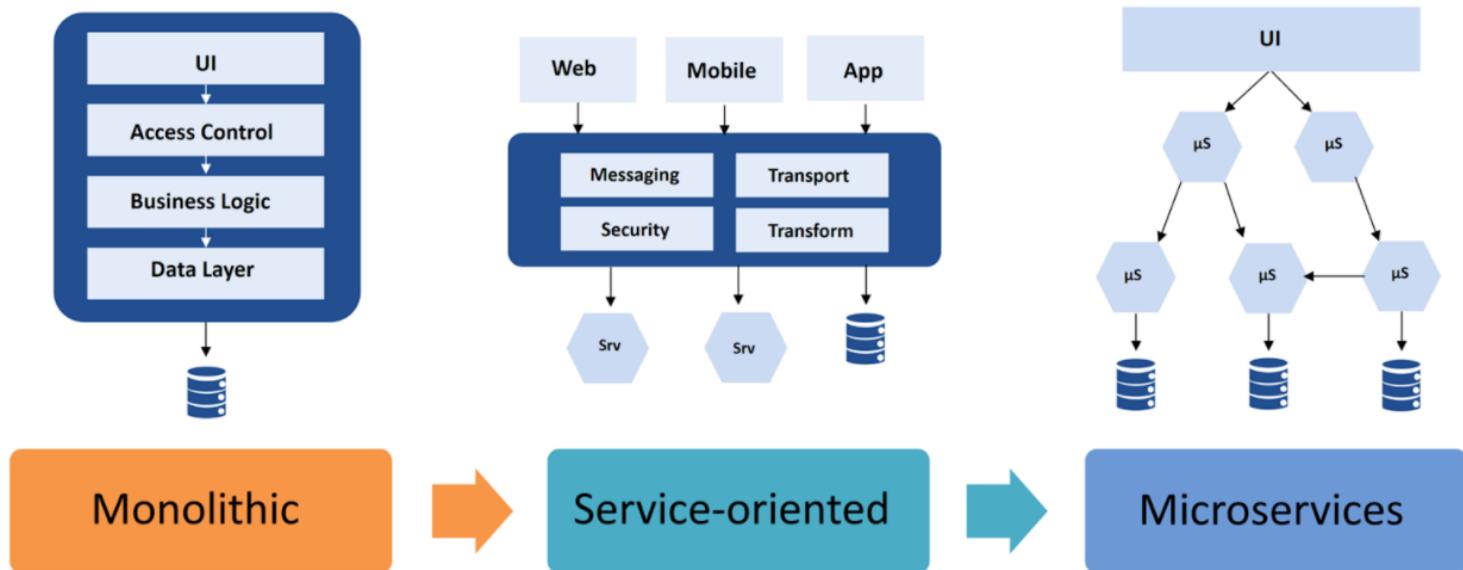
WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE

By @benorama

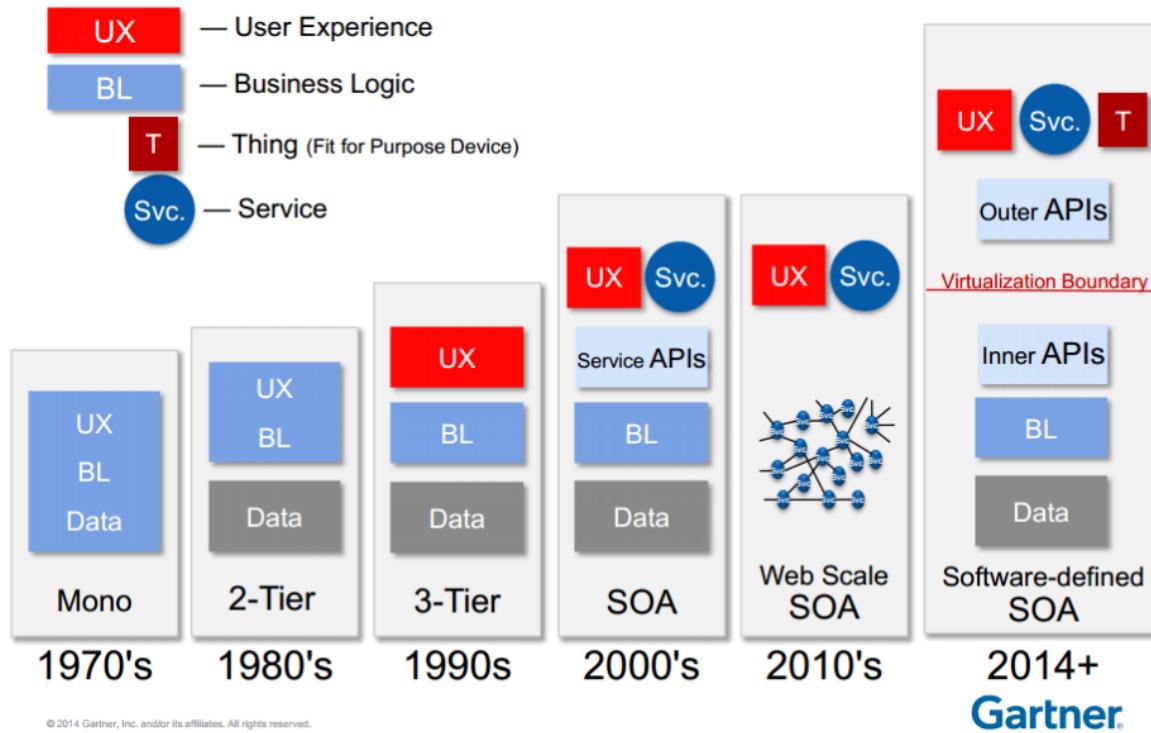
Evolución

Evolution of Software Architectures



Evolución

Software-defined Applications on the Application Architecture Road Map



© 2014 Gartner, Inc. and/or its affiliates. All rights reserved.

Arquitectura monolítica



Monolítico:

- › adj. Perteneciente o relativo al monolito.
- › adj. Que está hecho de una sola piedra.
- › adj. De una pieza, sin fisuras.
- › adj. Inconmovible, rígido, inflexible

Arquitectura monolítica

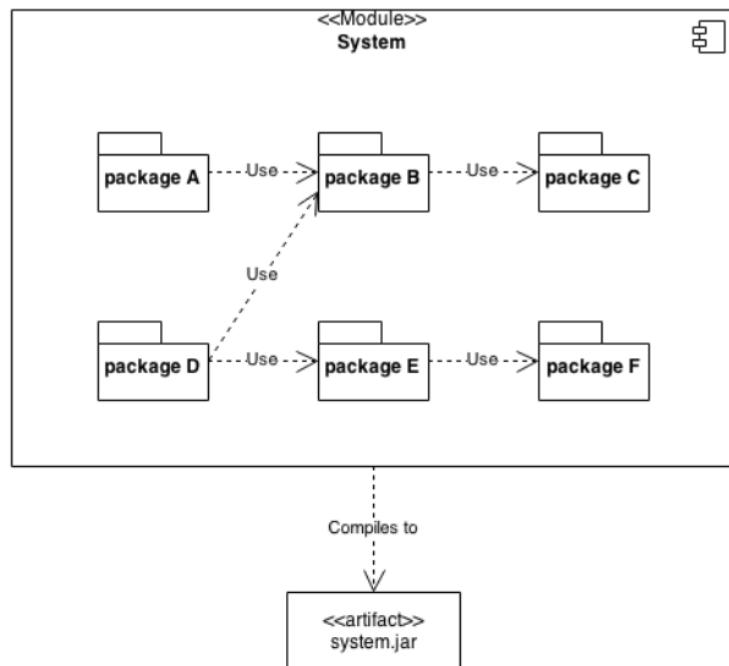
- Monolítico, en el contexto de software, significa compuesto todo de una pieza.
- El software monolítico está diseñado para ser autónomo; los componentes del programa están interconectados e interdependientes en lugar de estar débilmente acoplados, como es el caso de los programas de software modulares.
- Dado su alto nivel de acoplamiento, cada componente y sus componentes asociados deben estar presentes para que el código sea ejecutado o compilado.
- Si se debe actualizar cualquier componente del programa, se debe reescribir toda la aplicación.
- Los sistemas monolíticos no están divididos en partes independientes, corren en una sola unidad de procesamiento
- En su momento fueron la única elección ya que los computadores no hablaban mucho entre ellos.

Arquitectura monolítica

- La ausencia de las comunicaciones complejas requería altos niveles de procesamiento y solo permitía escalabilidad vertical.
- Estructura interna indefinida y niveles de funcionalidad no separados.
- Típica e Históricamente, los sistemas monolíticos hablaban con clientes brutos. (terminales).
- Un solo programa compuesto de un conjunto de rutinas entrelazadas a través del ligador con un alto acoplamiento.
- Carecen de protecciones y privilegios al entrar a rutinas que manejan diferentes aspectos de los recursos como memoria, disco, etc.
- No hay distribución, tanto a nivel físico ni a nivel lógico. Los programas monolíticos suelen tener un mejor rendimiento que los enfoques modulares y pueden ser más fáciles de probar y depurar porque, con menos elementos, hay menos variables que entran en juego.
- Un monolito puede considerarse un estilo arquitectónico o un patrón de desarrollo de software (o anti-patrón visto de forma negativa).

Arquitectura monolítica

Módulo monolítico:

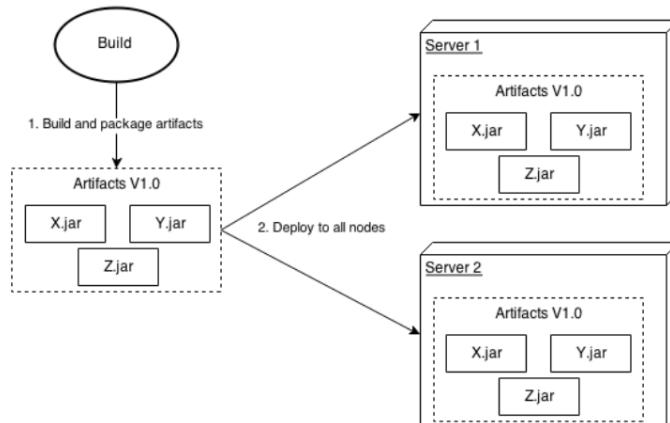


- Todo el código de un sistema se encuentra en una única base de código compilada conjuntamente y que produce un solo artefacto.
- El código aún puede estar bien estructurado (clases y paquetes coherentes y desacoplados) pero no se divide en módulos separados para la compilación.
- Existen ventajas y desventajas se hace principalmente para la administración del desarrollo.

Arquitectura monolítica

Monolito de asignación:

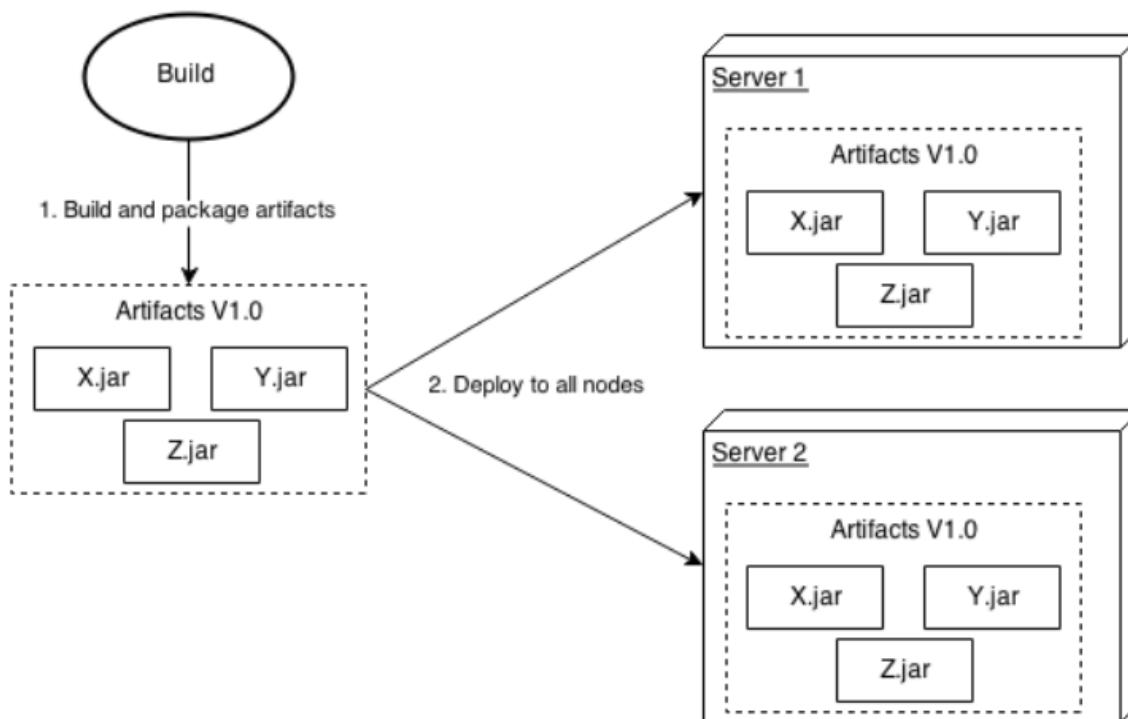
- Todo el código se envía / implementa al mismo tiempo. En otras palabras, una vez que el código compilado está “listo para su lanzamiento”, se envía una única versión a todos los nodos.
- Todos los componentes en ejecución tienen la misma versión del software ejecutándose en cualquier momento.



- Esto es independiente de si la estructura del módulo es un monolito.*
- De cualquier forma, esta versión se implementa en todas partes a la vez (requiere detener todo el sistema, desplegar el software y luego reiniciar).*

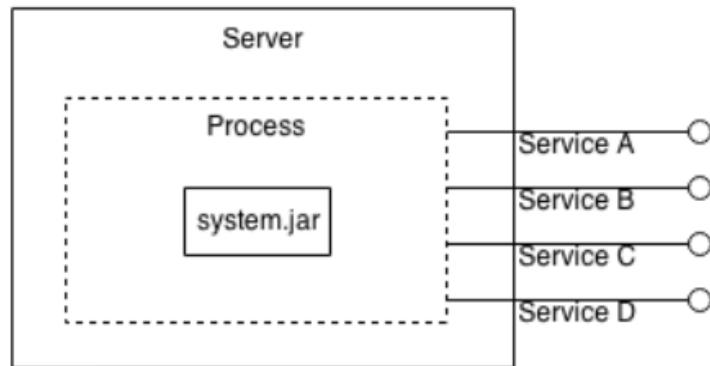
Arquitectura monolítica

Monolito de asignación:



Arquitectura monolítica

Monolito en tiempo de ejecución:

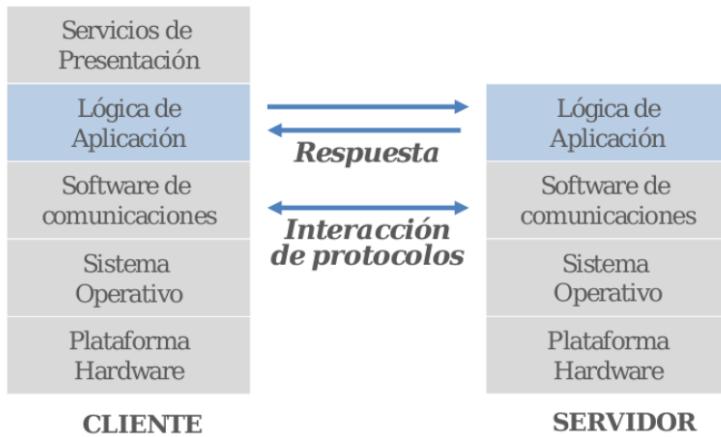


- Tienen una única aplicación o proceso que realice el trabajo para el sistema.
- Si el tiempo de ejecución es un monolito es independiente de si el código del sistema es un monolito de módulo o no.
- Un monolito de tiempo de ejecución a menudo implica un monolito de asignación si solo se implementa un nodo / componente principal

Arquitectura cliente / servidor

- Consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) quien da respuesta.
- Aunque se puede aplicar a programas que se ejecutan sobre una sola computadora ofrece más ventajas en un sistema operativo multiusuario distribuido a través de una red de computadoras.
- La capacidad de proceso está repartida entre los clientes y los servidores.
- Ofrece ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades.
- Facilita y clarifica el diseño del sistema.
- La separación entre cliente y servidor es una separación de tipo lógico.
- El servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

Arquitectura cliente / servidor



- Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc.
- Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

Arquitectura cliente / servidor

Cliente:

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación (dispositivo maestro o amo).
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.

Arquitectura cliente / servidor

Servidor:

- Al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número de clientes (en ciertos casos el número máximo de peticiones puede estar limitado).
- No es frecuente que interactúen directamente con los usuarios finales.

Arquitectura cliente / servidor

Clasificación arquitectura cliente / servidor

- Sistema de clasificación según tamaño de componentes.
 - *Fat Client (Thin Server).*
 - *Fat Server (Thin Client).*
- Sistema de clasificación según planos o capas (Tier).
 - *Planos a niveles de software.*
 - *Planos a niveles de hardware.*
- Clasificación según la naturaleza del servicio.
 - *Servidores de bases de datos.*

Arquitectura N-capas

Capas:

- Hace referencia a la forma como una solución es segmentada desde el punto de vista lógico, las capas son el mecanismo de estructuración lógica para los elementos que componen la solución del software.

Niveles:

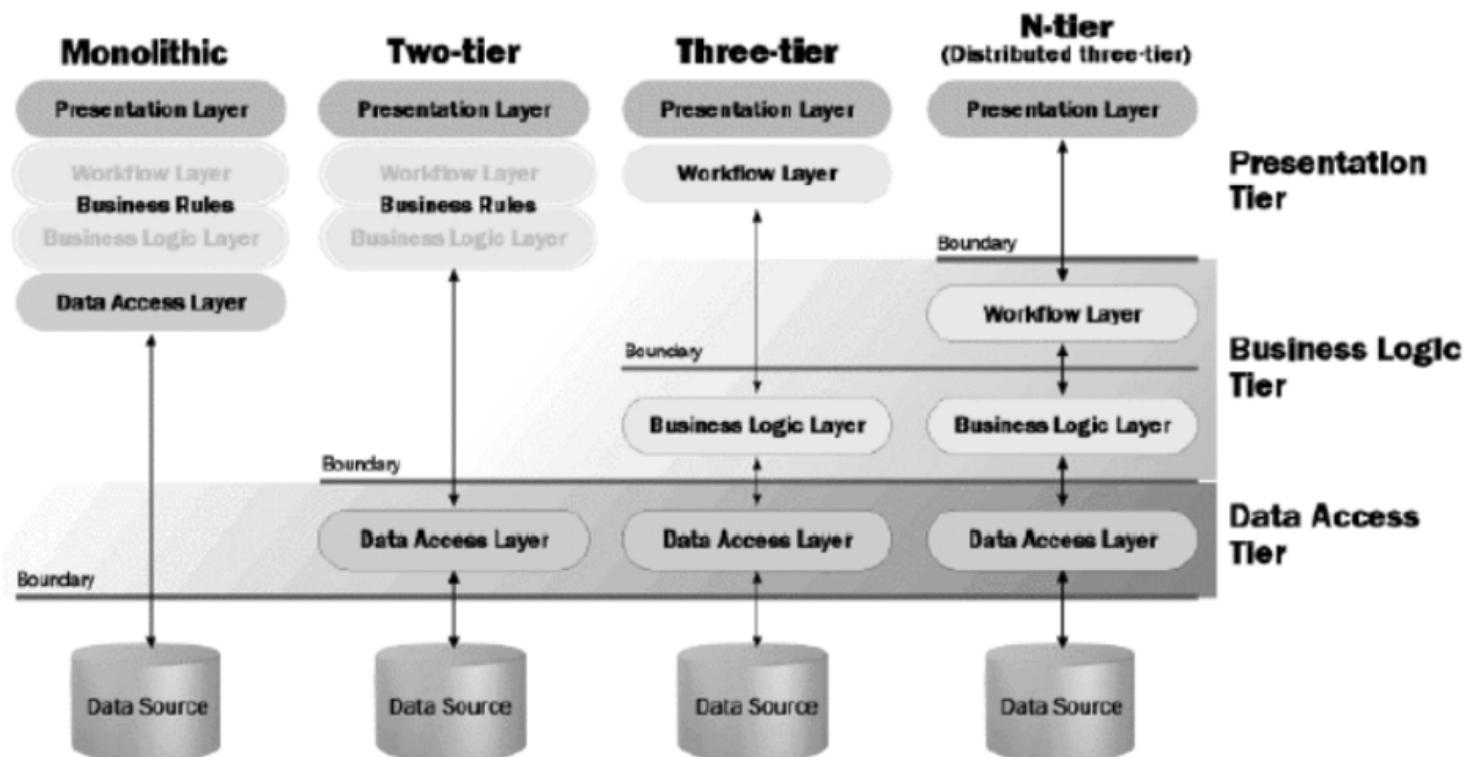
- Corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Es un mecanismo para la estructuración de la infraestructura del sistema.

Arquitectura N-capas

Niveles y capas:

- Una solución de tres capas (presentación, lógica, datos) que residen en un solo ordenador (Presentación + lógica + datos). Se dice que la arquitectura de la solución es de ***tres capas y un nivel***.
- Una solución de tres capas (presentación, lógica, datos) que residen en dos ordenadores (presentación + lógica, lógica + datos). Se dice que la arquitectura de la solución es de ***tres capas y dos niveles***.
- Una solución de tres capas (presentación, lógica, datos) que residen en tres ordenadores (presentación, lógica, datos). La arquitectura que la define es: solución de ***tres capas y tres niveles***.

Arquitectura N-capas



Arquitectura N-capas

Descripción:

- La N significa cualquier cantidad de capas, no hay límites.
- Ayuda a simplificar, al enfrentamos a mezcla de aplicaciones, interfaces y las redes multiplataformas.
- La cantidad de capas de software depende de las necesidades del negocio
- Las capas se pueden distribuir en el hardware que sea necesario.
- Presentan organizaciones Jerárquicas.
- Cada capa resuelve una parte del problema.
- Las capas se comunican mediante servicios.
- Cada capa puede cambiar su implementación independiente de las otras capas
- Los sistemas en capas más conocidos son las pilas de protocolos para comunicaciones y los sistemas operativos.

Arquitectura N-capas

Mecanismos, protocolos y estándares para la comunicación entre capas:

- **CORBA** (Common Object Request Broker Architecture) Arquitectura Común de Intermediarios en Peticiones a Objetos: es un estándar que facilita la invocación de métodos remotos bajo un paradigma orientado a objetos.
- **DCOM** (Distributed Component Object Model) Modelo de Objetos de Componentes Distribuidos, es una tecnología propietaria de Microsoft para desarrollar componentes software distribuidos.
- **RMI** (Java Remote Method Invocation) es un mecanismo ofrecido en Java para invocar un método remotamente.
- **SOAP** (Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML
- **Web Services**: es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Arquitectura N-capas

Ventajas:

- Administrabilidad
- Reutilización de capas
- Facilidad en la estandarización
- Las dependencias se limitan a intra-capa
- Contención de cambios a una o pocas capas
- Centralización del control
- Escalabilidad
- Fácil mantenimiento
- Existencia de tecnologías con soporte a desarrollo de n-capas
- Extensibilidad
- Seguridad

Arquitectura N-capas

Desventajas:

- › Posible pérdida en la contención del cambio
- › Pérdida de eficiencia (redundancia)
- › Dificultad de diseñar correctamente la granularidad de las capas.
- › Aumento de la complejidad.
- › Problemas de comunicaciones (congestión del tráfico).
- › Costos de Mantenimiento.

Arquitectura Orientada a Servicios

Definiciones:

- **W3C:** “Conjunto de componentes que pueden ser invocados, cuyas descripciones de interfaces se pueden publicar y descubrir”
- **SEI:** “Estilo resultante de políticas, prácticas y frameworks que permiten que la funcionalidad de una aplicación se pueda proveer y consumir como conjuntos de servicios, con una granularidad relevante para el consumidor. Los servicios pueden invocarse, publicarse y descubrirse y están abstraídos de su implementación utilizando una sola forma estándar de interface”.
- **IBM:** “Una arquitectura de aplicación en la cual todas las funciones se definen como servicios independientes con interfaces invocables bien definidas, que pueden ser llamadas en secuencias definidas para formar procesos de negocios”.

Arquitectura Orientada a Servicios

SOA es un estilo de arquitectura que promueve descomponer la lógica funcional de una aplicación en unidades autónomas denominadas servicios

- **Operación:** Unidad de trabajo o procesamiento en una arquitectura SOA.
- **Servicio:** Contenedor de lógica. Compuesto por un conjunto de operaciones, las cuales ofrecerá a sus usuarios.
- **Mensaje:** Encargados de encapsular los datos de entrada y de salida.
- **Proceso de negocio:** Conjunto de operaciones ejecutada en una determinada secuencia con el objetivo de realizar una determinada tarea.

Arquitectura Orientada a Servicios

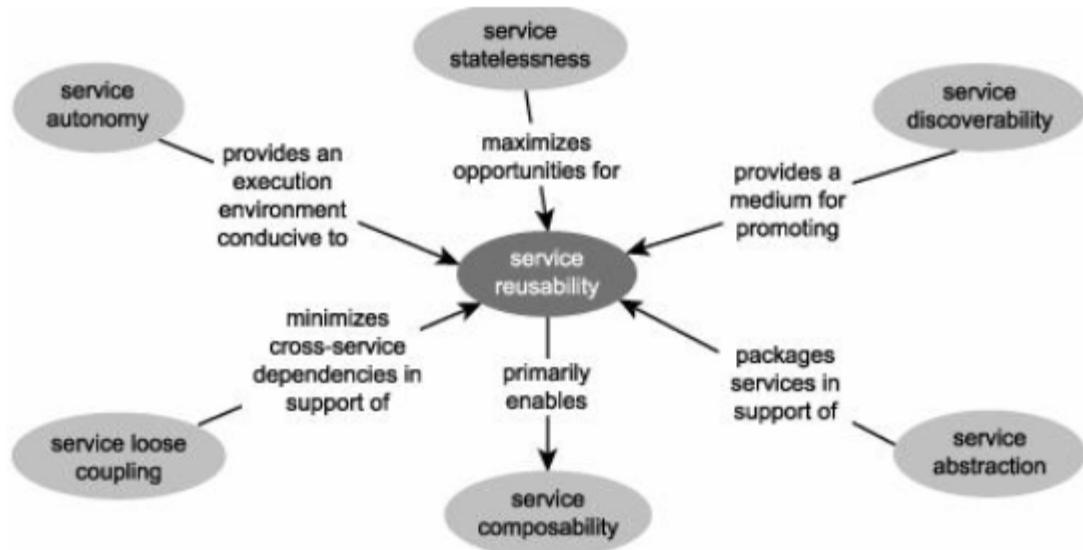
Servicios:

“Una aplicación SOA está formada por un conjunto de servicios interconectados cuyo objetivo es automatizar uno o varios procesos de negocio”.



Arquitectura Orientada a Servicios

Principios:



Arquitectura Orientada a Servicios

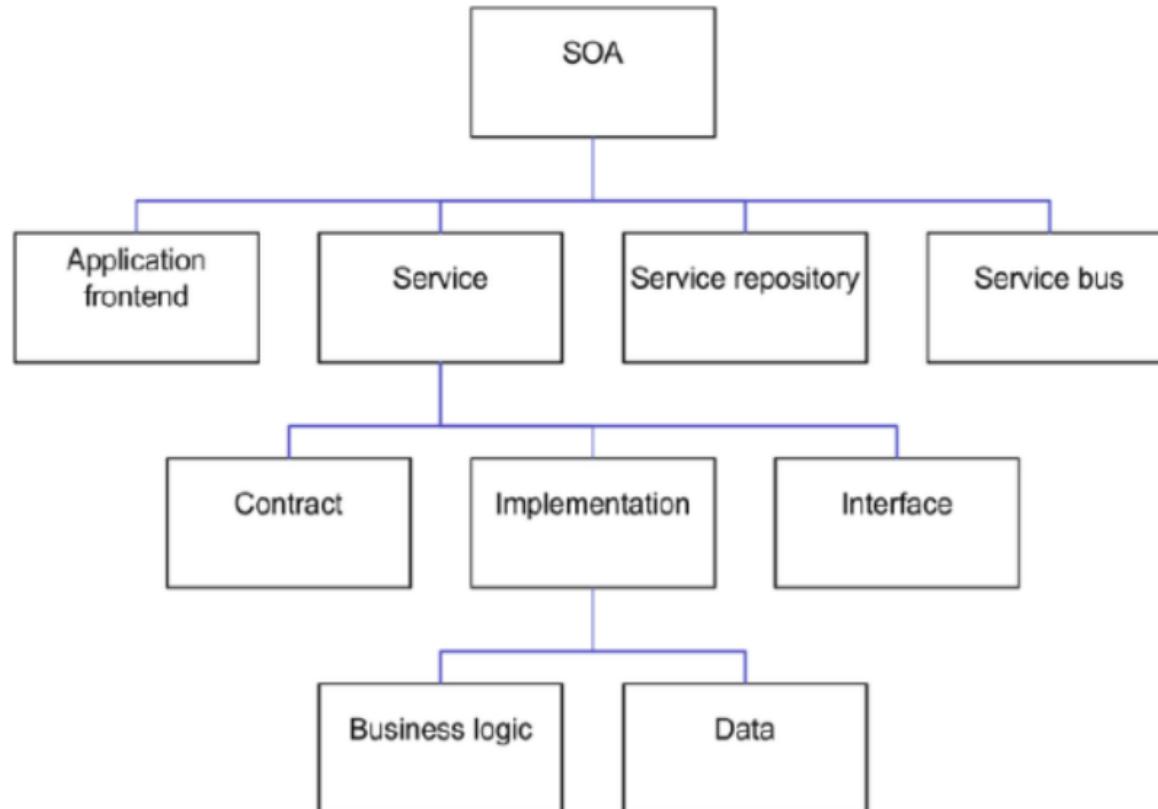
- SOA es un estilo arquitectónico para la construcción de aplicaciones de software en base a servicios disponibles.
- Es un paradigma para organizar y utilizar capacidades distribuidas y bajo el control de diferentes propietarios y dominios.
- Topología de software, que consiste de servicios y consumidores de servicios en una relación débilmente acoplada.
- Las capacidades del Negocio y los Procesos del negocio, serán modelados como servicios.
- Provee una manera uniforme de ofrecer, descubrir, interactuar y usar dichas capacidades para producir los efectos deseados de manera consistente y medible.
- Arquitectura conceptual.

Arquitectura Orientada a Servicios

- Organiza funciones de negocio como servicio interoperables.
- Permite reutilización de servicios para satisfacer necesidades de negocio.
- Es basado en estándares.
- Estrategia de tecnologías de información a nivel empresarial.
- Esconde detalles internos (Plataforma, Lenguaje de programación, lógica que ejecuta los procesos, almacenamiento).
- Los servicios encapsulan procesos de negocios.
- Esta en capacidad de involucrar diferentes tecnologías y representa mejor la integración de las mismas.
- Todos los servicios son independientes, operan como cajas negras. Los componentes externos no conocen ni les interesa como desempeñan su función, solo que entreguen el resultado esperado.

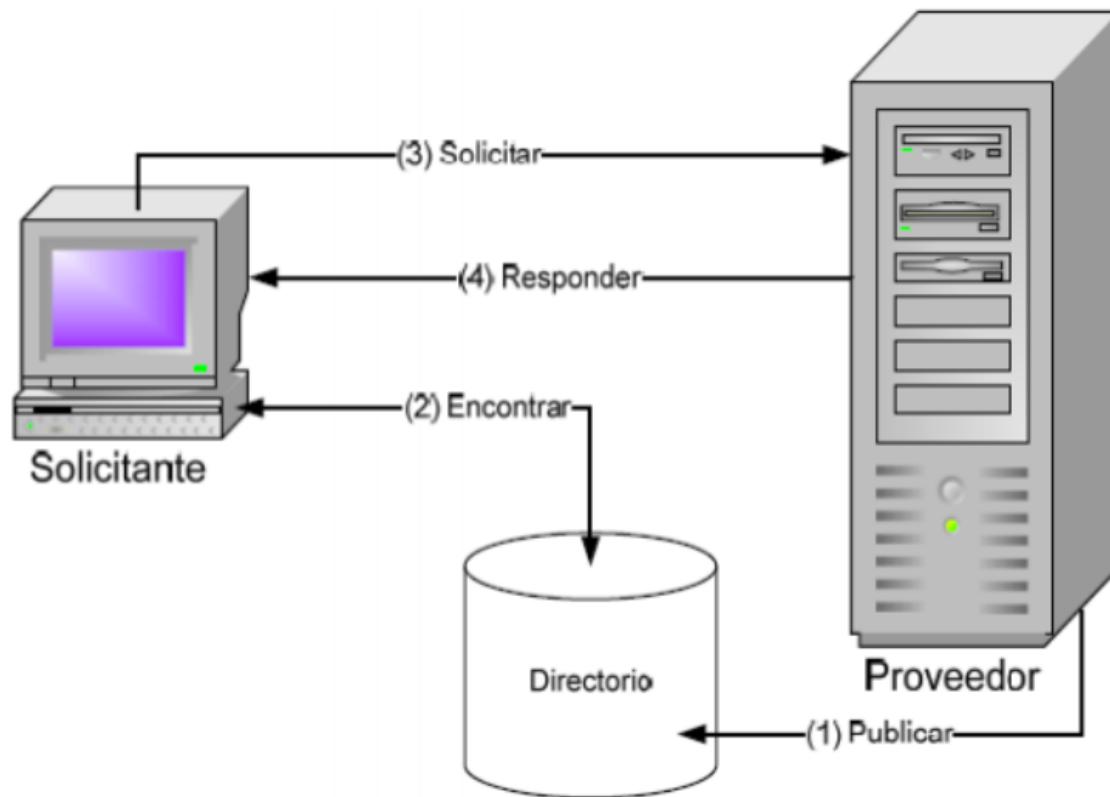
Arquitectura Orientada a Servicios

Componentes:



Arquitectura Orientada a Servicios

Dinámica:



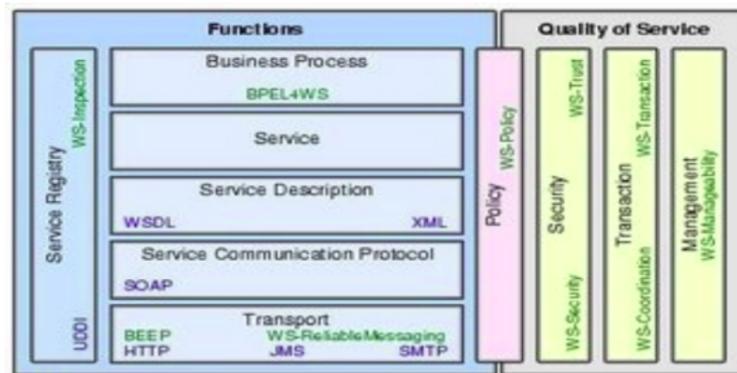
Arquitectura Orientada a Servicios

Niveles de servicio:

- **Servicios de Aplicación** (Application Services): Sirven como base para la creación de servicios de negocio.
- **Servicios de Negocio** (Business Services): Expresan la lógica del negocio.
- **Servicios de Orquestación** (Orchestration Services): Consisten en servicios de proceso compuestos por business services y application services coordinados mediante reglas de negocio y secuencias de ejecución.

Arquitectura Orientada a Servicios

Estándares SOA:



- **XML:** Extensible Markup Language
- **SOAP:** Protocolo de Mensajería.
- **WSDL:** Definición de la interfaz.
- **UDDI y ebXML:** Capacidades de localización.
- **WS-Security, WS-BPEL**
- **HTTP:** Mecanismo de transporte.

Arquitectura Orientada a Servicios

Ventajas:

- Ayuda a mejorar la agilidad y flexibilidad de las organizaciones.
- Permite simplificación del desarrollo de soluciones mediante la utilización de estándares de la industria y capacidades comunes de industrialización.
- Permite alinear y acercar la áreas de tecnología y de negocio.
- Permite aislar mejor a los sistemas frente a los cambios generados por otras partes de la organización.

Desventajas:

- SOA requiere un cambio en las organizaciones, un alto esfuerzo
- Elimina progresos que la historia de los paradigmas suelen resolver paso a paso, tales como el rendimiento o la integridad de la información.

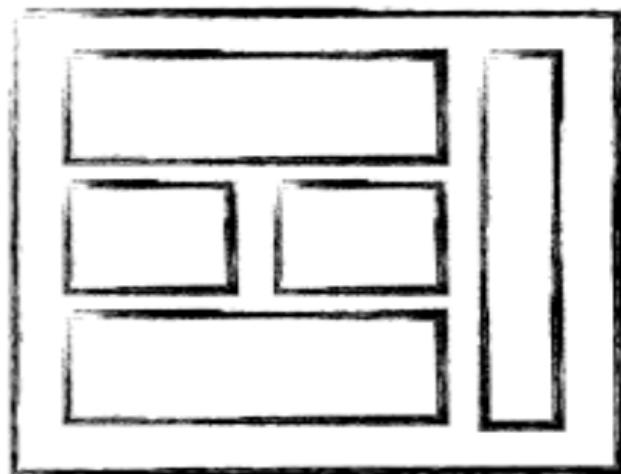
Arquitecturas basadas en microservicios

¿Qué son los microservicios?

- La arquitectura orientada a microservicios es un estilo arquitectónico en el cual una aplicación compleja está compuesta de pequeños servicios independientes que se comunican entre sí usando APIs agnósticas.
- Estos servicios son pequeños, altamente desacoplados y enfocados en pequeñas tareas.
- La arquitectura de microservicios permite la entrega / despliegue continuo de aplicaciones grandes y complejas.

Arquitecturas basadas en microservicios

Monolítico vs Microservicios



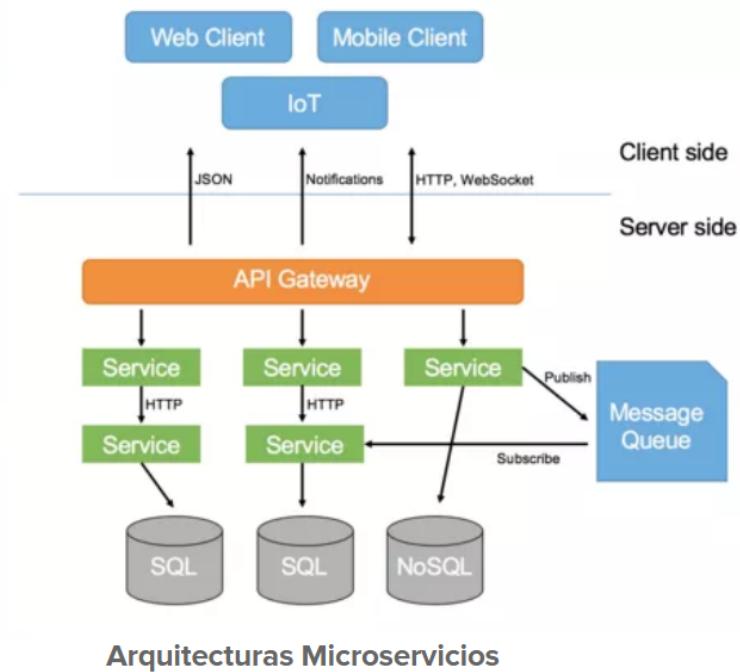
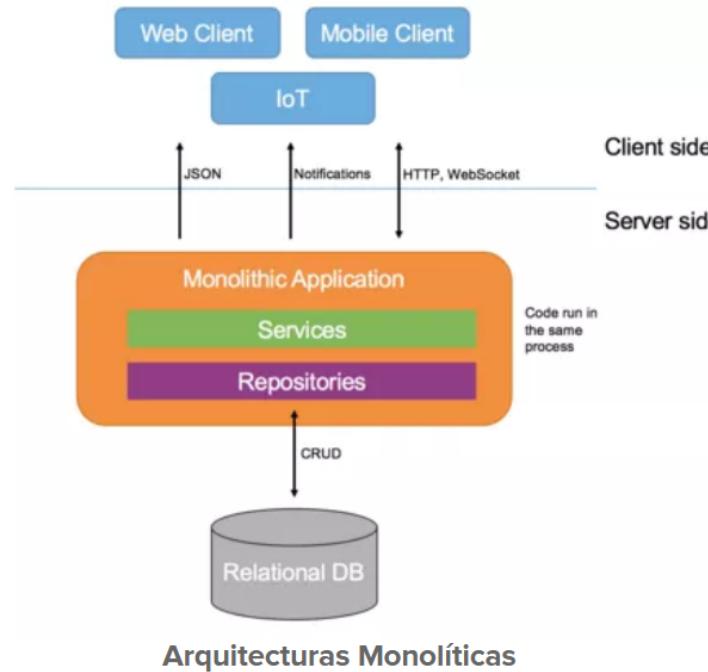
MONOLITHIC/LAYERED



MICRO SERVICES

Arquitecturas basadas en microservicios

Monolítico vs Microservicios



Arquitecturas basadas en microservicios

Monolítico vs Microservicios



Arquitecturas Monolíticas:

cualquier cambio afecta el producto completo



Arquitecturas SOA:

Hay más independencia, pero debe existir coordinación para encajar en el conjunto.



Arquitecturas microservicios:

- Desarrollo independiente.
- Despliegue independiente.
- Políglotas.
- Escalado eficiente.
- Menor Time-to-market

Arquitecturas basadas en microservicios

Beneficios de los microservicios

Responsabilidad única: una única tarea por microservicio



Arquitecturas Monolíticas



Arquitecturas Microservicios

Arquitecturas basadas en microservicios

Beneficios de los microservicios

Desarrollo eficiente: microservicios pequeños y especializados



Arquitecturas Monolíticas



Arquitecturas Microservicios

Arquitecturas basadas en microservicios

Beneficios de los microservicios

Escalado eficiente, elástico y horizontal: basado en la demanda



Arquitecturas Monolíticas



Arquitecturas Microservicios

Arquitecturas basadas en microservicios

Beneficios de los microservicios

Políglota: los microservicios puede estar desarrollado en plataformas diferentes



Arquitecturas Monolíticas



Arquitecturas Microservicios

Arquitecturas basadas en microservicios

Beneficios de los microservicios

Despliegue independiente: los microservicios se despliegan de forma independiente



Arquitecturas Monolíticas



Arquitecturas Microservicios

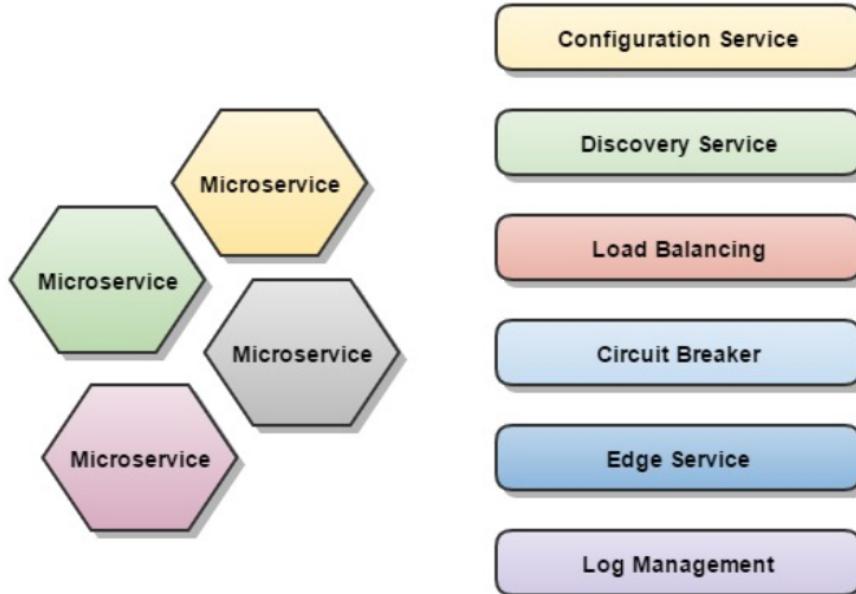
Arquitecturas basadas en microservicios

Características de la arquitectura

- Los componentes son servicios.
- Organizada en torno a las funcionalidades del negocio.
- Productos no proyectos.
- Extremos inteligentes tuberías bobas.
- Tener gobierno descentralizado permite usar tecnologías que se adapten mejor a cada funcionalidad.
- Gestión de datos descentralizada.
- Diseño pensado en las fallas.
- Automatización de la infraestructura.
- Diseño evolutivo.

Arquitecturas basadas en microservicios

¿Qué necesito para microservicios?



Arquitecturas basadas en microservicios

¿Qué necesito para microservicios?

Servidor de configuración central

- Se encargará de centralizar y proveer remotamente la configuración a cada microservicio. Esta configuración se mantiene convencionalmente en un repositorio Git, lo que nos permitirá gestionar su propio ciclo de vida y versionado.

Servicio de registro / descubrimiento

- Será el encargado de proveer los endpoints de los servicios para su consumo. Todo microservicio se registrará automáticamente en él en tiempo de bootstrap.

Arquitecturas basadas en microservicios

¿Qué necesito para microservicios?

Balanceo de carga (Load balancer)

- Permite el balanceo entre distintas instancias de forma transparente a la hora de consumir un servicio.

Tolerancia a fallos (Circuit breaker)

- Permite que cuando se produzca un fallo, este no se propague en cascada por todo el pipe de llamadas, y poder gestionar el error de forma controlada a nivel local del servicio donde se produjo.

Servidor perimetral/exposición de servicios (Edge server)

- Será un gateway en el que se expondrán los servicios a consumir.

Arquitecturas basadas en microservicios

¿Qué necesito para microservicios?

Centralización de logs

- Se hace necesario un mecanismo para centralizar la gestión de logs. Sería inviable la consulta de cada log individual de cada uno de los microservicios.

Servidor de Autorización

- Para implementar la capa de seguridad (recomendable en la capa de servicios API)

Monitorización

- Es interesante el poder disponer de mecanismos y algún dashboard para monitorizar aspectos de los nodos como, salud, carga de trabajo.

Arquitecturas basadas en microservicios

Stacks tecnológicos reconocidos

- Docker – <http://www.docker.io>
- Amazon Web Services (AWS) – <http://amazon.com>
- Eureka (Servicio de descubrimiento) – <https://github.com/Netflix/eureka>
- Ribbon (Balanceador de carga en cliente) – <https://github.com/Netflix/ribbon>
- Hystrix (Circuit Breaker y Monitorización) – <https://github.com/Netflix/Hystrix>
- Zuul (Edge/Proxy Server , con enrutado dinámico, monitorización, capacidad de recuperación, seguridad, etc...) – <https://github.com/Netflix/zuul>
- Spring Cloud Config (Configuración distribuida) – <http://cloud.spring.io/spring-cloud-config/>
- Archaius (Configuración distribuida con cambios en caliente) – <https://github.com/Netflix/archaius>

Arquitecturas basadas en microservicios

Buenas prácticas relacionadas

- Evitar congestiones de tráfico haciendo hincapié en el cacheo de las peticiones a microservicios de alto uso.
- Implementar y adherirse a las prácticas culturales de DevOps. Los recursos de desarrollo necesitarán su propio camino hasta producción para sus cambios.
- Pasar los datos vía http headers cuando sea posible para reducir congestiones de tráfico entre microservicios.
- Implementar el patrón “Circuit Breaker” para evitar fallos en cadena y hacer al sistema más tolerante a fallos.
- Usar un servicio de mensajería basado en colas tal como RabbitMQ, OpenMQ, etc… cuando sea posible.

Arquitecturas basadas en microservicios

Buenas prácticas relacionadas

- ➊ Des-sincronizar todo lo posible para incrementar el rendimiento.
- ➋ Implementar protocolos de restricción de acceso para reducir la disponibilidad de los microservicios a aquellos que no deben llamarlos.
- ➌ Usar soluciones de descubrimiento de servicios para que la arquitectura pueda dinámicamente expandirse o contraerse sin la intervención de un agente de gobierno.
- ➍ Esperar lo inesperado.