

{Xideral}

10/05/2024

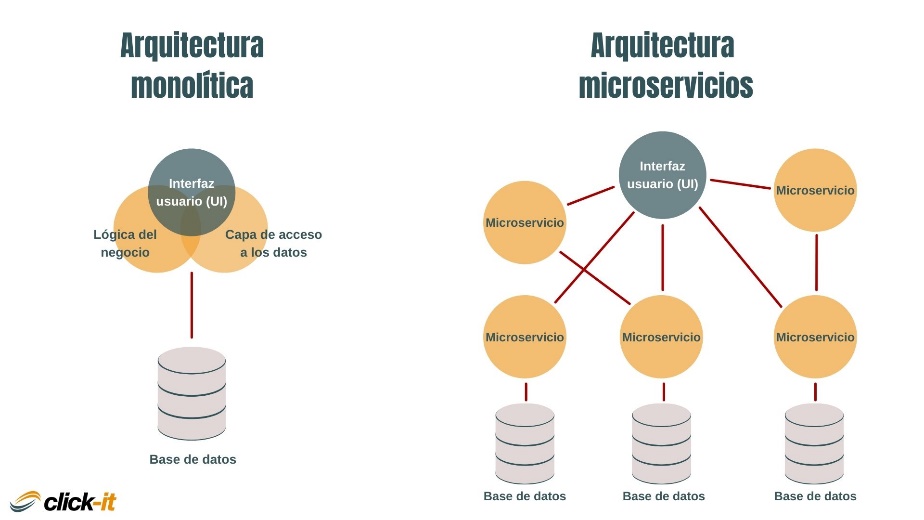
**MICROSERVICIOS**

**Luis Alberto García Reyes**

# Introduccion

## definición sobre la arquitectura de microservicios.

Los microservicios son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí.

A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de esos elementos o procesos es un microservicio. Este enfoque de desarrollo de software valora el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Es un elemento fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

## Importancia de los microservicios

Los microservicios mejoran las rutinas e impulsan a sus equipos por medio del **desarrollo distribuido**. También puede desarrollar múltiples microservicios de forma simultánea. Gracias a ello, más desarrolladores pueden trabajar en la misma aplicación simultáneamente para reducir el tiempo invertido en el desarrollo.

Algunas de los beneficios de los microsevicios son:

* Agilidad
* Mayor Accesibilidad
* Libertad del desarrollador
* Implementacion simplificada
* Alta escalabilidad
* Resiliencia sin paragon
* Reutilizacion
* Reduccion del tiempo de comercilizacion

# CAracteristicas de los microservicios

**Capacidad de prueba**

Como los microservicios están diseñados como una colección de servicios individuales e independientes, se pueden probar fácilmente como componentes independientes. Los problemas en los componentes se pueden aislar rápidamente, en lugar de tener que probar sistemas y aplicaciones completos para luego invertir mucho tiempo en tratar de aislar los fallos concretos.

**Poca conexión**

Para operar simultáneamente, los microservicios deben mantener la comunicación entre sí. Dicho esto, lo cierto es que entre ellos hay poca conexión, ya que los cambios que se implementan en un servicio no afectan directamente a los demás.

**Descentralización**

En lugar de compartir los almacenes de datos entre servicios, cada componente mantiene el suyo propio. Esto ayuda a evitar el acoplamiento accidental de distintos servicios y garantiza que los cambios no afecten involuntariamente a otros servicios independientes.

**Implementación independiente**

Los servicios individuales se cambian e implementan en el entorno de producción sin tener que emplear otros. Todas las implementaciones en el sistema se gestionan de esta forma, lo que hace que los microservicios puedan mejorarse muy rápidamente.

**Enfoque empresarial**

Los microservicios emplean equipos multifuncionales que se organizan en torno a un único propósito empresarial. Estos equipos suelen formarlos desarrolladores, ingenieros de bases de datos, evaluadores, ingenieros de infraestructura y otros, con el objetivo de desarrollar productos específicos basados en múltiples servicios independientes.

**Redirección simple**

En el caso de los microservicios, cada servicio independiente puede recibir, procesar y responder solicitudes. Esto supone una gran simplificación en comparación con muchos de los sistemas más tradicionales, donde las complejas capas de aplicación de redirección y reglas empresariales pueden ralentizar el proceso.

**Resistencia a fallos**

Para que un sistema de microservicios falle por completo, todos los servicios independientes deben fallar al mismo tiempo. Al depender de servicios interdependientes con poca conexión, un sistema puede seguir funcionando casi a la capacidad óptima incluso en caso de fallo en uno de sus servicios. Puesto que los servicios están descentralizados, la pérdida de uno tendrá poco o ningún impacto en los demás.

**Adaptabilidad y escalabilidad**

Como los microservicios son de naturaleza modular, es relativamente sencillo añadir nuevos servicios cuando es necesario. Esto posibilita que las organizaciones adapten los sistemas actuales a nuevos usos y escalen o disminuyan los sistemas para satisfacer la demanda cambiante.

**Pila de tecnología intercambiable**

Al desarrollar un nuevo servicio, las organizaciones tienen la libertad de escoger entre varias pilas de tecnología. A la vez, pueden emplear otras nuevas cuando realizan cambios en los servicios existentes.

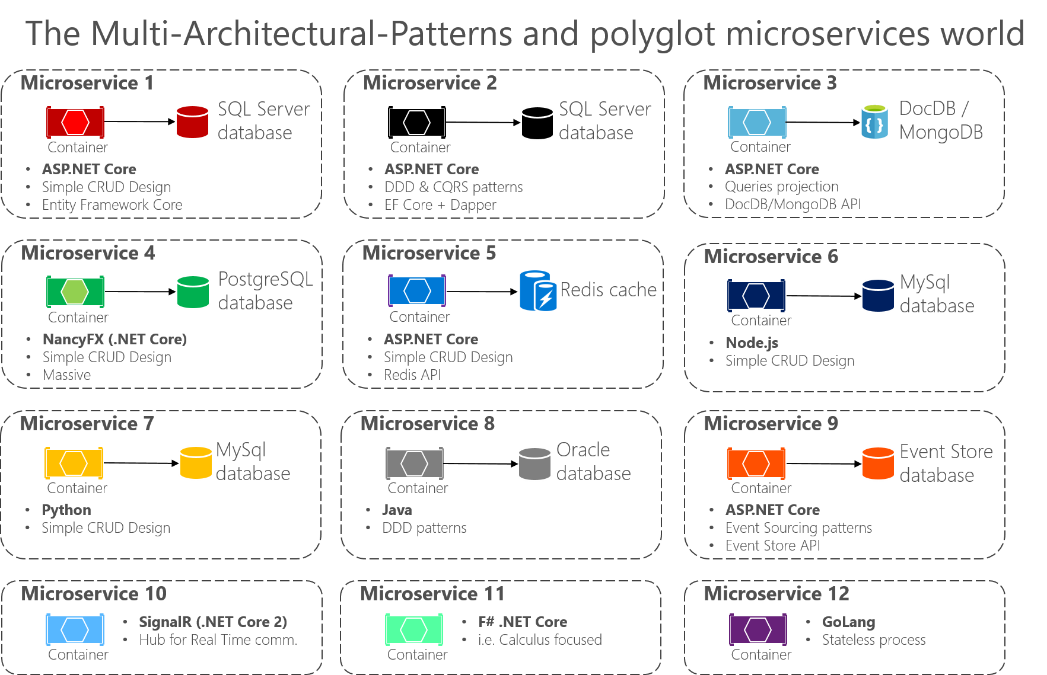
# Tecnologías y herramientas comunes

## Frameworks

Los desarrolladores y arquitectos de software usan muchos modelos arquitectónicos. Los siguientes son algunos de ellos (se combinan estilos y modelos arquitectónicos):

* CRUD simple, de un nivel y una capa.
* Tradicional de N capas.
* Diseño controlado por dominios de N capas.
* Arquitectura limpia (como se usa con eShopOnWeb)
* Segregación de responsabilidades de consultas de comandos (CQRS).
* Arquitectura controlada por eventos (EDA).

Lo importante es que ningún modelo o estilo arquitectónico determinado, ni ninguna tecnología concreta, es adecuado para todas las situaciones.



## Contenedores y orquestadores

### Contenedores

Un contenedor es una unidad de software ligera que incluye todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas y las dependencias. Los contenedores proporcionan aislamiento, lo que significa que cada aplicación se ejecuta en su propio entorno independiente, sin interferir con otras aplicaciones en el mismo sistema. Por ejemplo:

**Docker** es la plataforma de contenedores más popular y ampliamente utilizada. Permite empaquetar y distribuir aplicaciones en contenedores de forma eficiente y portátil.

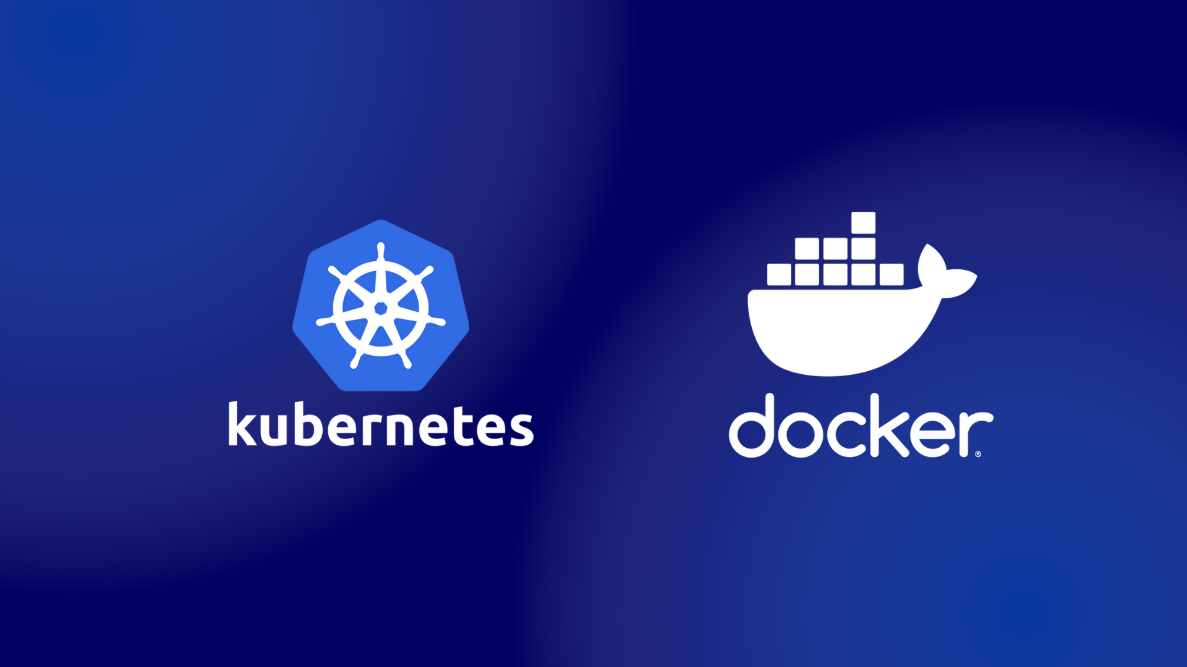
### Orquestadores

Los orquestadores son herramientas que automatizan la implementación, el escalado, la gestión y el monitoreo de contenedores en un entorno de producción.

**Kubernetes** es el orquestador de contenedores más popular y ampliamente adoptado en la actualidad. Permite gestionar clústeres de contenedores a gran escala de manera eficiente y escalable. Dado que los microservicios son aplicaciones divididas en servicios pequeños e independientes, cada uno empaquetado y desplegado en su propio contenedor.

Los contenedores proporcionan un entorno ligero y portátil para ejecutar microservicios, lo que facilita su implementación y gestión en cualquier infraestructura. En este caso los orquestadores como Kubernetes permiten gestionar de forma dinámica y eficiente un gran número de contenedores, lo que es fundamental para la implementación y operación de microservicios a escala.

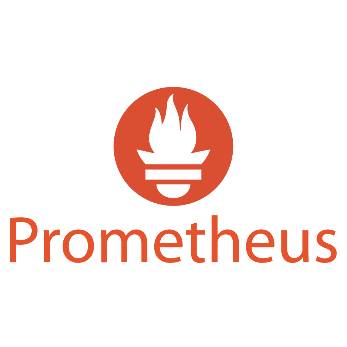
Kubernetes facilita la automatización de tareas como el despliegue, la escalabilidad, el balanceo de carga, la recuperación ante fallos y la gestión de la red, lo que simplifica la gestión de un entorno de microservicios complejo.



## Herramientas de monitoreo y gestión

Las herramientas de monitoreo y gestión son fundamentales para garantizar el rendimiento, la disponibilidad y la fiabilidad de los microservicios en un entorno de producción. Aquí te presento algunas de las herramientas más comunes utilizadas para monitorear y gestionar microservicios:

### Prometheus

* Prometheus es un sistema de monitoreo y alerta de código abierto diseñado especialmente para entornos de contenedores y microservicios.
* Permite recopilar métricas de diversos servicios y componentes de la infraestructura, como contenedores, orquestadores, aplicaciones y sistemas operativos.
* Proporciona consultas flexibles y expresivas para analizar y visualizar datos de monitoreo en tiempo real.
* También ofrece capacidades de alerta que permiten definir umbrales y condiciones para enviar notificaciones cuando se detectan problemas.

### Grafana

* Grafana es una plataforma de visualización de datos de código abierto que se utiliza comúnmente junto con Prometheus para crear paneles de monitoreo y visualizar métricas.
* Permite crear gráficos interactivos, tablas y diagramas de manera fácil y flexible.
* Es altamente personalizable y admite una amplia variedad de fuentes de datos, incluyendo Prometheus, bases de datos de series temporales y sistemas de registro.

### ELK Stack (Elasticsearch, Logstash, Kibana):

* ELK Stack es una combinación de tres herramientas de código abierto utilizadas para la recopilación, el almacenamiento, el análisis y la visualización de registros y datos de registro.
* Elasticsearch se utiliza como motor de búsqueda y análisis en tiempo real para almacenar y consultar datos de registros.
* Logstash se utiliza para recopilar, filtrar y enrutar datos de registros desde diversas fuentes hacia Elasticsearch.
* Kibana se utiliza para visualizar y analizar datos de registros mediante la creación de tableros interactivos y gráficos.

### Jaeger:

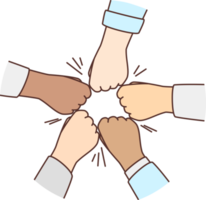
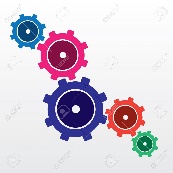
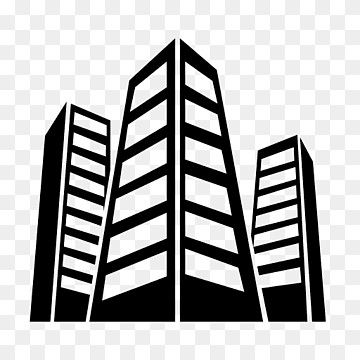
* Jaeger es una plataforma de trazabilidad de código abierto que se utiliza para el seguimiento distribuido de solicitudes a través de microservicios.
* Permite rastrear y visualizar el flujo de solicitudes a través de los distintos servicios de una aplicación, lo que facilita la detección y el diagnóstico de problemas de rendimiento y latencia.
* Proporciona herramientas para la generación de perfiles, la resolución de problemas y el análisis de tendencias en el rendimiento de las aplicaciones distribuidas.

### New Relic:

* New Relic es una plataforma de monitoreo de aplicaciones en la nube que proporciona visibilidad en tiempo real sobre el rendimiento y la salud de las aplicaciones y los servicios.
* Ofrece herramientas para el monitoreo de infraestructura, el seguimiento de transacciones, el análisis de errores y el diagnóstico de problemas de rendimiento.
* También proporciona alertas personalizables y tableros de control para visualizar métricas clave y tendencias de rendimiento.

# Funcionamiento de los microservicios

Para comprender el funcionamiento y uso de los microservicios podemos comenzar mencionando que un microservicio es una pequeña aplicación independiente que se enfoca en realizar una tarea específica dentro de un sistema más grande. El funcionamiento de un microservicio se basa en varios principios y prácticas:

* Descomposición de la aplicación: Un sistema monolítico se divide en varios microservicios, cada uno responsable de una función o tarea específica. Esta descomposición permite escalar y actualizar partes del sistema de forma independiente.
* Independencia y autonomía: Cada microservicio es una entidad autónoma con su propia lógica de negocio, datos y capacidad para realizar operaciones de manera independiente. Esto permite un desarrollo, despliegue y mantenimiento más ágil y flexible.
* Comunicación: Los microservicios se comunican entre sí a través de protocolos de red, como HTTP, gRPC o mensajes asincrónicos. Esta comunicación puede ser síncrona (a través de llamadas de red directas) o asíncrona (a través de colas de mensajes).
* Gestión de datos: Cada microservicio puede tener su propia base de datos o almacenamiento de datos, lo que permite aislamiento y independencia en el acceso y gestión de la información. Sin embargo, también se pueden compartir bases de datos o sistemas de almacenamiento si es necesario.
* Despliegue y escalabilidad: Los microservicios se pueden desplegar y escalar de forma independiente, lo que permite optimizar los recursos según la demanda de cada servicio. Esto significa que se pueden asignar más recursos a los microservicios que requieren mayor capacidad de procesamiento o almacenamiento.
* Tolerancia a fallos: Los microservicios están diseñados para ser tolerantes a fallos. Si un microservicio falla, no afecta necesariamente al funcionamiento de otros microservicios en el sistema. Además, se pueden implementar estrategias de recuperación, como la replicación y el balanceo de carga, para garantizar la disponibilidad y la fiabilidad del sistema en su conjunto.

# COMUNICACION

La comunicación efectiva entre microservicios es fundamental para su funcionamiento conjunto. Es crucial comprender los diferentes enfoques de comunicación disponibles, ya que la elección adecuada puede influir en la eficiencia, confiabilidad y rendimiento del sistema.

## La importancia de la comunicacion

A diferencia de las aplicaciones monolíticas, donde los componentes interactúan dentro del mismo contexto, los microservicios se ejecutan de manera independiente y se comunican a través de la red.

Esto introduce desafíos adicionales en términos de ***latencia, fiabilidad y seguridad***. La selección de protocolos de comunicación, la gestión de errores, la sincronización de datos y la garantía de la disponibilidad de los servicios pueden volverse tareas complejas que requieren una planificación cuidadosa y una implementación sólida.

Además, la diversidad tecnológica entre microservicios puede añadir complejidad adicional.

Cada servicio puede estar desarrollado en un lenguaje de programación diferente, utilizar bases de datos distintas o seguir paradigmas de diseño variados. La comunicación efectiva entre estos componentes puede requerir la implementación de patrones y prácticas específicas, como la integración de bus de mensajes, la gestión de versiones de API y una documentación exhaustiva.

## Riesgos

Es importante considerar que, debido a las diferencias entre proyectos, incluyendo limitaciones de presupuesto, plazos de desarrollo y complejidad, no todos los proyectos son adecuados para adoptar un enfoque orientado a microservicios. Aunque este enfoque ofrece ventajas como agilidad y escalabilidad, puede implicar un aumento sustancial en los costos de desarrollo y mantenimiento que no todos los proyectos pueden soportar. Por lo tanto, si bien los microservicios ofrecen beneficios significativos, la complejidad en su comunicación es un factor crítico que debe abordarse de manera cuidadosa, estratégica y temprana en la arquitectura de una aplicación.

## Comunicación por receptores

Un segundo criterio de clasificación de los protocolos de comunicación entre microservicios es en función del número de receptores, distinguiéndose entre un único receptor o varios.

1. **Receptor único**. En este caso, cada solicitud ha de ser procesada por un receptor o un servicio. Un ejemplo de este tipo de comunicación es el patrón Command.
2. **Varios receptores.** Un ejemplo de este tipo es la arquitectura de microservicios controlada por eventos, la cual se basa en un agente de mensajes o interfaz de bus de eventos que se dedica a propagar las actualizaciones de datos entre diferentes microservicios a través de eventos. Por lo general, este interfaz se implementa a través de un bus de servicio o algún dispositivo similar, como Azure Service Bus, mediante temas y suscripciones.

## comunicación sincrona

* HTTP: En este modelo, un microservicio actúa como cliente y envía una solicitud HTTP a otro microservicio que actúa como servidor. Esta solicitud puede llevar parámetros y datos necesarios para la acción solicitada. El microservicio receptor procesa la solicitud y devuelve una respuesta.
* REST (Representational State Transfer): Define estándares sobre cómo diseñar y estructurar servicios web. Utiliza HTTP como protocolo de comunicación y permite realizar operaciones sobre recursos identificados a través de URLs. Los microservicios exponen endpoints RESTful que permiten a otros microservicios realizar operaciones específicas.
* gRPC (Google Remote Procedure Call): Es un sistema de llamada a procedimientos remotos desarrollado por Google. gRPC permite a los microservicios comunicarse de manera eficiente y segura utilizando interfaces definidas por IDL (Interface Definition Language). Utiliza HTTP/2 para la transferencia de datos, lo que mejora la velocidad y eficiencia.

En todos estos escenarios, la comunicación entre microservicios es secuencial, y un microservicio que solicita información a otro queda a la espera de una respuesta inmediata, ya sea de éxito o de error.

Diagrama

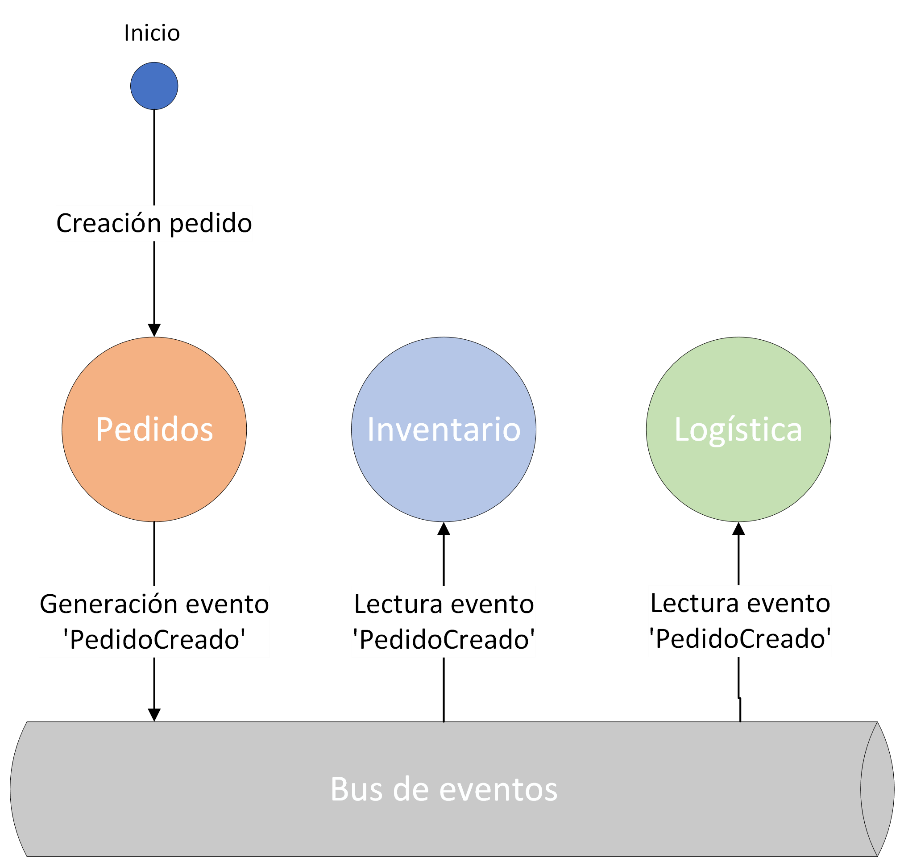
Descripción generada automáticamente

## Comunicación asíncrona

La comunicación asíncrona entre los microservicios es una estrategia fundamental para mejorar la flexibilidad, escalabilidad y eficiencia en el intercambio de información. A diferencia de la comunicación síncrona, donde un microservicio espera una respuesta inmediata, en la comunicación asíncrona los microservicios pueden enviar y recibir información sin depender de una respuesta inmediata, lo que les permite iniciar nuevas tareas de manera más eficiente. Algunas estrategias comunes para este tipo de comunicaciones son:

* Sistemas de Mensajería: Plataformas como Apache Kafka, RabbitMQ y AWS SQS ofrecen sistemas de mensajería que permiten a los microservicios enviar y recibir mensajes a través de colas y canales temporales antes de ser procesados por los microservicios destinatarios.
* Publicación y Suscripción: En esta estrategia, los microservicios emisores publican mensajes en canales específicos, y los microservicios receptores se suscriben a estos canales para recibirlos. Esto permite que varios microservicios respondan a un mismo evento simultáneamente y que se desencadenen diferentes acciones en paralelo.
* Arquitectura Orientada a Eventos (EDA): En esta arquitectura, los microservicios generan eventos en respuesta a cambios o acciones en el sistema. Otros microservicios pueden reaccionar a estos eventos y actuar en consecuencia, lo que facilita una comunicación asíncrona.

Este tipo de comunicación promueve la independencia entre los microservicios, lo que resulta en un bajo acoplamiento y una mayor autonomía. Sin embargo, puede aumentar la complejidad de la lógica necesaria para garantizar el correcto funcionamiento, ya que se requieren más controles para asegurar que los microservicios se comuniquen correctamente debido a la ausencia de respuestas inmediatas.



# Seguridad en microservicios

## Autenticación y Autorización

La autenticación es el proceso de verificar la identidad de un usuario o un servicio que solicita acceso a un recurso. La autorización es el proceso de determinar qué acciones u operaciones puede realizar un usuario o un servicio en un recurso. Tanto la autenticación como la autorización se basan en el principio de privilegios mínimos, lo que significa otorgar solo los permisos mínimos necesarios para realizar una tarea.

## Gestión de Tokens

Se utilizan tokens de seguridad, como JSON Web Tokens (JWT), para autenticar y autorizar las solicitudes entre microservicios. Estos tokens contienen información codificada sobre el usuario o servicio y sus permisos.

Expiración de Tokens: Establecer una política de expiración para los tokens JWT es fundamental para garantizar la seguridad de la aplicación. Los tokens deben tener un tiempo de vida limitado después del cual se vuelven inválidos y no se pueden utilizar para acceder a los recursos protegidos. Esto ayuda a mitigar el riesgo de que un token sea comprometido y utilizado por un atacante malintencionado.

* Renovación de Tokens: La renovación de tokens permite a los usuarios autenticados obtener un nuevo token válido después de que el token actual haya expirado. Esto se puede lograr implementando un mecanismo de renovación automática o proporcionando una función de renovación explícita a través de la cual los usuarios pueden solicitar un nuevo token antes de que el actual expire.
* Políticas de Renovación: Es importante establecer políticas claras para la renovación de tokens, incluyendo la frecuencia máxima permitida para renovar un token y los requisitos de autenticación adicionales que pueden ser necesarios para renovar un token, como la verificación de la identidad del usuario.
* Seguridad de las Claves de Firma: Los tokens JWT utilizan claves de firma para verificar su autenticidad. Es fundamental proteger adecuadamente estas claves para evitar su compromiso y el acceso no autorizado a los recursos protegidos. Esto puede implicar el uso de almacenamiento seguro de claves y la rotación periódica de las claves para mitigar el riesgo de compromiso.

# Desarrollo y despliegue de microservicios

Los principales proveedores de servicios en línea, como Google, Amazon y Netflix, que operan con arquitecturas distribuidas a hiperescala, demuestran que todos los equipos de DevOps deben abordar ciertos principios y cuestiones para prepararse para un despliegue de microservicios.

Estos son 6 pasos que han aprendido con la experiencia:

## 1.- Utilizar servicios en la nube para la infraestructura de producción

Una arquitectura de microservicios implica aplicaciones compuestas por componentes independientes y escalables, aprovechando los servicios en la nube para desagregar funciones de aplicaciones en componentes separados. Esto se traduce en una mayor rentabilidad y conveniencia al ejecutar funciones de aplicaciones como servicios bajo demanda. Los contenedores y las funciones sin servidor son opciones preferidas sobre instancias de VM para implementar microservicios, gestionadas idealmente a través de servicios Kubernetes en la nube como Azure Kubernetes Service, AWS Elastic Kubernetes Service o Google Kubernetes Engine. Al evaluar la viabilidad, costos y rendimiento de servicios en la nube para microservicios, es crucial no basarse en experiencias previas con tecnologías heredadas y realizar investigaciones sobre las diferencias financieras y de rendimiento entre unidades de estado sólido y discos duros, lo que podría justificar la inversión adicional en SSD.

## 2.- Diseñar para el fracaso

Una arquitectura de microservicios es distribuida y desacoplada, con componentes individuales que funcionan independientemente. Esto permite actualizaciones y escalabilidad sin depender de otros microservicios. La infraestructura de microservicios no se basa en un solo servidor, evitando un diseño monolítico. Es crucial diseñar para tolerancia a fallos, tratando servidores como entidades sin estado. Las aplicaciones deben recuperarse de fallos a nivel de componente. Herramientas como Gremlin y Chaos Monkey prueban la resiliencia de la infraestructura de microservicios.

## 3.- Descentralizar la gestión de datos

En los centros de datos empresariales, la consolidación de bases de datos y almacenamiento simplifica la gestión y reduce costos, pero es incompatible con el diseño de microservicios. Compartir una base de datos entre varios equipos de microservicios puede causar problemas al actualizarla. En cambio, los desarrolladores necesitan libertad para elegir la mejor base de datos, facilitado por opciones en la nube como AWS o Azure. También necesitan flexibilidad para cambiar esquemas y procedimientos sin riesgo de romper el código existente. Esto permite adaptarse a las necesidades cambiantes y mantener la agilidad en el desarrollo de software.

Gráfico

Descripción generada automáticamente

## 4.- Distribuir la gobernanza

Las organizaciones de TI tradicionales operan en silos funcionales, obstaculizando la agilidad y la innovación. Los microservicios reemplazan estos silos con equipos interfuncionales basados en DevOps y prácticas ágiles. Estos equipos se centran en productos específicos en lugar de funciones laborales, con un gestor responsable de su desarrollo y entrega. Para desplegar microservicios con éxito, se necesita una cultura que equilibre procesos mínimos con responsabilidad para solucionar problemas. La velocidad es prioritaria sobre la eficiencia, y los equipos de infraestructura son pequeños debido al soporte de proveedores de servicios en la nube. Los equipos centrados en productos fomentan la agilidad y la comunicación estrecha, facilitando la creación de interfaces API para la automatización y la integración de microservicios.

## 5.- Automatizar el despliegue de la infraestructura, adoptar procesos CI/CD

Los microservicios prosperan en un entorno ágil de DevOps, donde la automatización de procesos desempeña un papel crucial. Al automatizar tareas como la construcción y las pruebas de software en respuesta a actualizaciones de código, se acelera el ciclo de desarrollo y se fomenta un diseño evolutivo, aspectos fundamentales para los microservicios. Mantener la infraestructura inmutable, donde se crea un nuevo microservicio para agregar funciones o corregir errores mientras el antiguo sigue operativo, garantiza la continuidad del servicio. Además, se recomienda utilizar compilaciones separadas para cada microservicio para gestionar eficazmente las dependencias y evitar versiones obsoletas en el código. Esta automatización permite adaptarse rápidamente a cambios y minimiza la fricción en el proceso de desarrollo, contribuyendo al éxito de los microservicios en entornos dinámicos.

## 6.- Supervisar, registrar y solucionar problemas desde el principio

La construcción de aplicaciones a partir de un conjunto de microservicios al estilo Legos complica considerablemente la forma de supervisar y solucionar los problemas de los sistemas y su rendimiento. Varios microservicios suelen desencadenar una cascada de eventos que conducen a un fallo de la aplicación. Para minimizar los fallos –que no son un tal vez, sino una realidad– incorpore la supervisión y la resolución de problemas en el diseño de los microservicios.

# Pruebas y mantenimiento

Las pruebas son fundamentales en los microservicios para garantizar su funcionamiento correcto, confiabilidad y seguridad. Los equipos de desarrollo y operaciones (DevOps) son los principales usuarios de estas pruebas, ya que están involucrados en el desarrollo, implementación y mantenimiento de los microservicios.

Entre los tipos de pruebas comunes en microservicios se encuentran:

* **Pruebas unitarias**: Verifican el correcto funcionamiento de componentes individuales del microservicio.
* **Pruebas de integración**: Validan la interoperabilidad entre diferentes microservicios y sus dependencias.
* **Pruebas de aceptación del usuario (UAT)**: Se realizan para garantizar que el microservicio cumpla con los requisitos y expectativas del usuario final.
* **Pruebas de carga y rendimiento**: Evalúan la capacidad del microservicio para manejar la carga de trabajo esperada y su rendimiento en condiciones de producción.
* **Pruebas de seguridad**: Identifican posibles vulnerabilidades y riesgos de seguridad en el microservicio.

Empresas de diferentes industrias, desde tecnología hasta finanzas y comercio electrónico, utilizan estas pruebas en sus implementaciones de microservicios. Ejemplos de empresas que emplean pruebas en microservicios incluyen Netflix, Amazon, Spotify, eBay y muchas más. Estas compañías reconocen la importancia de garantizar la calidad y confiabilidad de sus microservicios para ofrecer servicios y productos de alta calidad a sus clientes