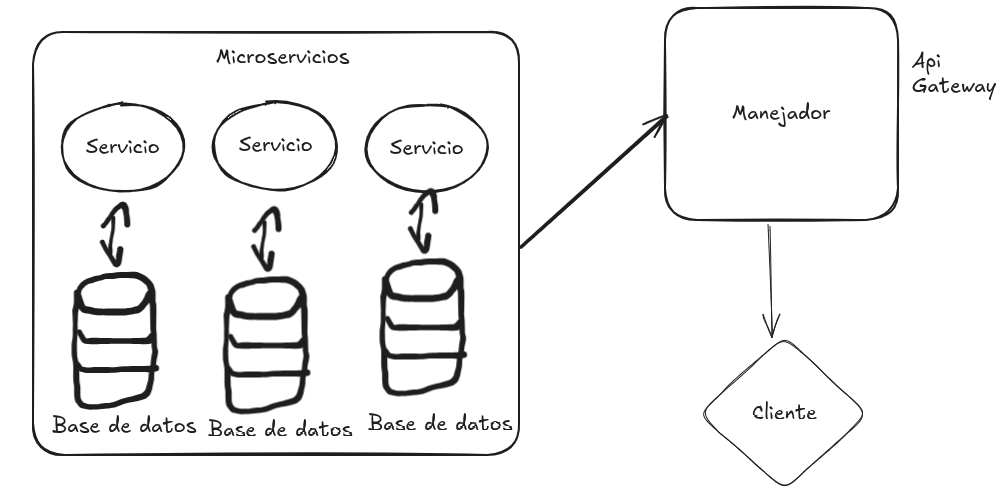
Articulos

# 1.Arquitectura de microservicios

La **arquitectura de microservicios** surge como alternativa al estilo **monolítico**, donde las aplicaciones empresariales se construyen como una unión de tres partes: cliente, manejo de datos y servidor con lógica de dominio. Aunque este enfoque puede ser exitoso, los constantes cambios en el modelo de negocio hacen que la aplicación crezca, se complique su mantenimiento y su escalabilidad incremente la capacidad de toda la aplicación en lugar del segmento que lo requiere.

Para enfrentar este problema, la arquitectura de microservicios propone construir una aplicación como un conjunto de **servicios independientes**, incluso en diferentes lenguajes y equipos, lo que permite escalar solo el servicio que lo necesita. Ejemplos como Amazon, eBay y Netflix adoptaron este enfoque. Cada servicio tiene su propia **arquitectura hexagonal** y base de datos independiente, lo que asegura bajo acoplamiento y diversidad tecnológica.



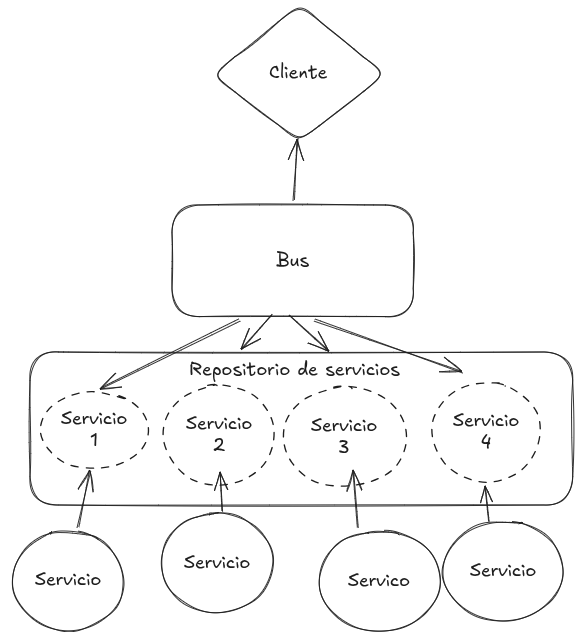
Reflexión:

La arquitectura de microservicios no solo representa una solución efectiva para proyectos de gran escala, sino que, gracias a la independencia y flexibilidad entre servicios, permite reutilizar microservicios de un proyecto en otro, lo que ayuda a reducir el tiempo y los costos de desarrollo. Además, fomenta la creación de soluciones dinámicas y adaptables a los cambios en cualquier tipo de proyecto.

# Bibliografía:

Contreras, D. A. B. (2018). Arquitectura de microservicios. *Tecnología Investigación y Academia*, *6*(1), 36-46.

# 2.Arquitectura orientada a servicios

La arquitectura de software es una parte crucial de la ingeniería de software, encargada de estudiar, analizar y dibujar el mapa completo de un sistema. Se concentra mucho, en las conexiones entre sus piezas principales.  
En este ámbito, aparece la Arquitectura Orientada a Servicios (SOA) como un enfoque que une la TI con las exigencias de la empresa. SOA permite fusionar sistemas diversos, utilizando servicios independientes, estos se publican, encuentran y activan gracias a interfaces normales, sin importar el lenguaje o la plataforma empleada.  
El corazón de SOA late gracias a servicios autónomos que hablan a través de un bus de servicios y un repositorio. Facilita la reutilización, la integración y la gran flexibilidad. Hay servicios básicos, de negocio, de procesos o públicos, usando, frecuentemente, servicios web que usan estándares como XML, SOAP y WSDL.  
De esta manera, la productividad recibe un gran empujón, además de la seguridad y la flexibilidad organizacional frente a las transformaciones.

# Reflexión:

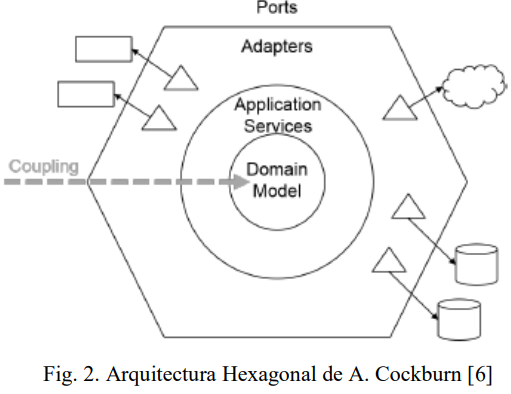
La arquitectura orientada a servicios puede reflejar un trabajo en equipo, cada persona representa un servicio el cual trabaja y funciona independientemente, pero uniendo 2 o más, se puede construir o hacer funcionar algo. El bus puede ser como un líder, el cual es el que solicita o requiere esos servicios para unificarlo o manejarlo para que se pueda usar.

# Bibliografía:

Martín, Y. E. (2012). Arquitectura de software. Arquitectura orientada a servicios. *Serie Científica de la Universidad de las Ciencias Informáticas*, *5*(1), 1-10.

# 3.Arquitectura hexagonal

La arquitectura hexagonal es un patrón estructural que establece entradas y salidas en los bordes del diseño, permitiendo intercambiar manejadores sin cambiar el código del núcleo. De esta forma, la aplicación puede ser manejada por usuarios, programas, pruebas automatizadas o scripts por lotes, y desarrollada de manera aislada de dispositivos y bases de datos en tiempo de ejecución.  
A diferencia de la arquitectura en capas, la arquitectura hexagonal permite que la interfaz de usuario también se pueda intercambiar. El modelo resultante se compone del modelo de dominio, los servicios de aplicación, los puertos y los adaptadores. Cada lado del hexágono representa un puerto concreto, aunque en la práctica puede haber más puertos con su correspondiente adaptador.  
El modelo de dominio, como núcleo, es independiente de los servicios de aplicación y adaptadores. Esto facilita cambiar frameworks o herramientas escribiendo un nuevo adaptador. El interior del hexágono se mantiene libre de cuestiones tecnológicas, y la comunicación se da mediante interfaces llamadas puertos.



Reflexión:

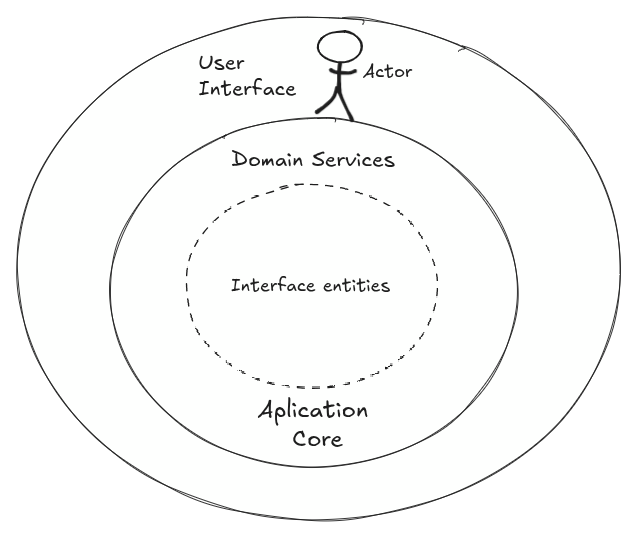
La arquitectura hexagonal propone que en lugar de construir sistemas dependientes de frameworks, base de datos o interfaces graficas, planeta que el “corazón debe ser el dominio”, esto quiere decir las reglas y lógica de negocio que realmente dan valor,

Bibliografía:

Cambarieri, M., Difabio, F., & García Martínez, N. (2020). Implementación de una Arquitectura de Software guiada por el Dominio. In *XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIO 49 (Modalidad virtual)*.

4.Arquitectura onion

La arquitectura onion combina las capas de Domain Driven Desing (DDD), que permite colocar en el centro la aplicación. En ese sentido, cada microservicio diseñado consta de este principio, donde la capada de interfaz de usuario se encarga de la presentación de la información al usuario, lo que permite la menor exposición del usurario. En la capa de dominio, se establece la lógica general del desarrollo de la plataforma web, permitiendo establecer la capa que contiene los datos y lógica de manipulación de los mismo. En la capa de aplicación, se establecen los casos de uso, que permiten la activación por medio de la application core mediante interfaz de usuario. La capa de datos queda la interacción de las bases de datos empeladas en el desarrollo de la plataforma web.



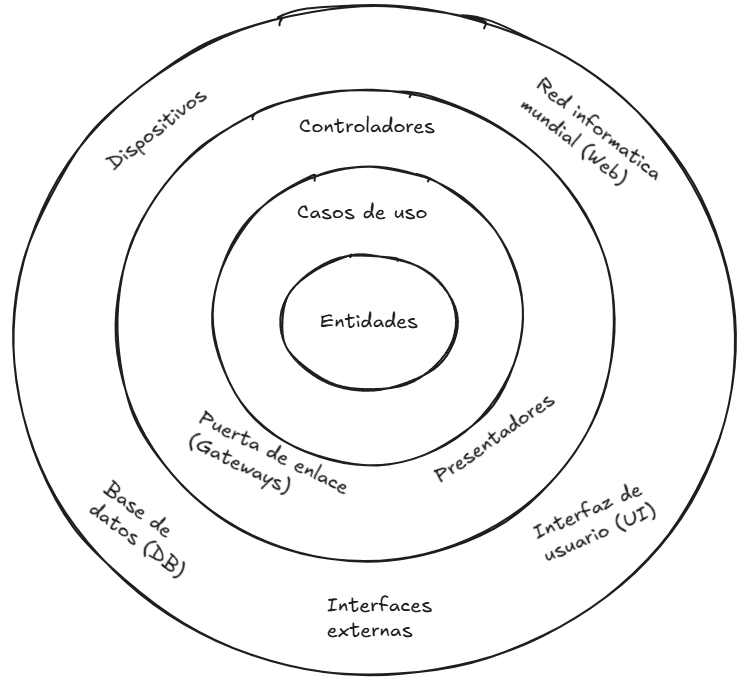
Reflexión:

Bibliografía:

Velásquez Meriño, H. D., & Viloria, J. (2025). Plataforma web con inteligencia artificial para la evaluación de habilidades de pensamiento crítico en estudiantes de la Corporación Universitaria del Caribe CECAR utilizando los TEST PC-B y TEST PC-A.

5. Arquitectura limpia

La arquitectura limpia se puede entender como la combinación de diferentes arquitecturas que buscan separar las preocupaciones a través de una capa de negocio y otra de interfaces. Esta estructura permite que las aplicaciones evolucionen de manera continua y se modifiquen sin afectar otras partes del sistema.  
  
Su funcionamiento se organiza en capas: la capa de presentación, que se encarga de interactuar con la interfaz de usuario utilizando el patrón de diseño MVP; la capa de dominio, que alberga la lógica de negocio y los casos de uso; y la capa de datos, que se ocupa del acceso y almacenamiento de la información. Cada capa se comunica hacia adentro, según la regla de la dependencia.  
  
Así, la lógica de negocio se mantiene independiente de detalles externos como frameworks, servicios o bases de datos. Si se necesita un cambio de tecnología, simplemente se pueden reemplazar los componentes externos sin tocar el núcleo de la aplicación.



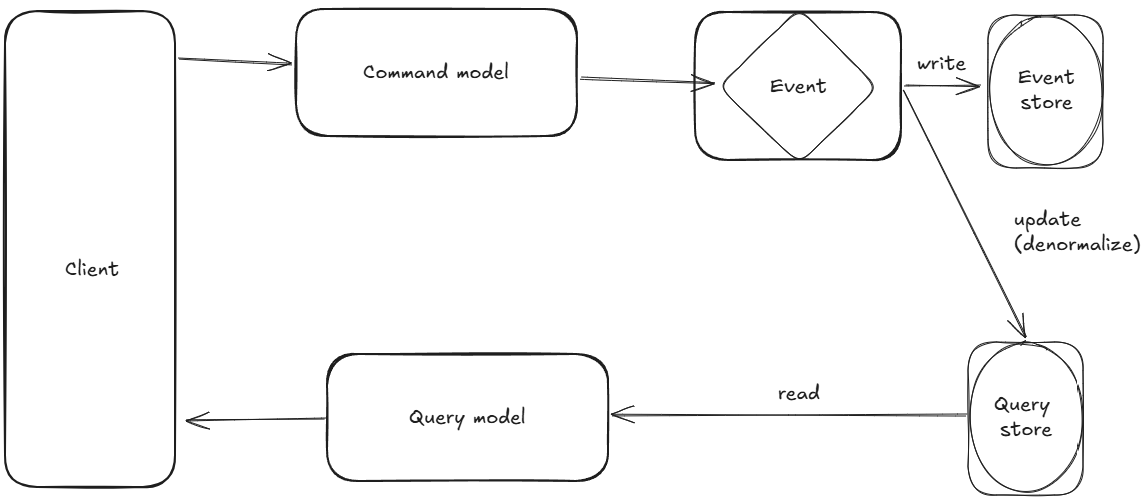
Reflexión:

Bibliografía:

Arias-Orezano, J. F., Reyna-Barreto, B. D., & Mamani-Apaza, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. TecnoLógicas, 24(52), 226-241

6. Arquitectura CQRS

La arquitectura CQRS se fundamenta en el principio de Command Query Separation, donde las operaciones se dividen en dos categorías: los comandos, que modifican el estado de la aplicación, y las consultas, que recuperan información. En este modelo, los comandos y las consultas se implementan como servicios separados dentro de una arquitectura de microservicios y la comunicación entre ellos se realiza a través de mensajes en un entorno orientado a eventos.  
El funcionamiento se organiza alrededor de tres tipos de mensajes: comandos, eventos y consultas. Los comandos expresan la intención de cambiar el estado y son manejados por command handlers que publican eventos. Los eventos notifican que una acción ocurrió y son capturados por event handlers y event sourcing handlers, los cuales actualizan el estado de los agregados y las proyecciones. Finalmente, las consultas solicitan datos y se gestionan con query handlers que acceden a los modelos de lectura.  
De esta manera, el núcleo del sistema se organiza en agregados, que representan entidades funcionales procesadas en un límite consistente. Los cambios se registran como eventos que mantienen sincronizados el almacén de escritura y el de lectura.



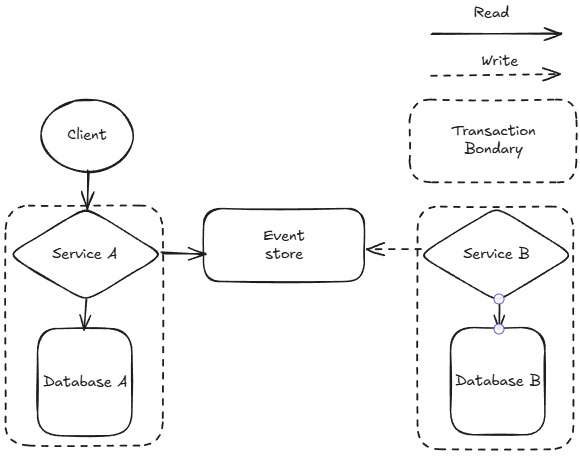
Reflexion:

Bibliografia:

Kindson, M., & Péter, M. (2023). A simplified approach to distributed message handling in a CQRS architecture. *Acta Polytechnica Hungarica*, *20*(4).

7.Arquitectura Event Sourcing

Los comandos son objetos que representan las tareas o acciones básicas de nuestro sistema. Cuando se lanza un comando, este es capturado por un CommandHandler, el cual a su vez lanza el evento o los eventos derivados de esta acción.  
Estos eventos son objetos que codifican la información del evento producido por la acción que acaba de ocurrir. En el subsistema de escritura, los eventos son capturados en el propio agregado por los EventSourcingHandlers los cuales se encargan de persistir dichos eventos en el EventStore.  
Los agregados o aggregates son entidades que representan el estado de la información ante cada evento que las modifique. Cada evento que provoca un cambio sobre un agregado crea una especie de snapshot del estado del agregado en ese momento, por lo que podemos ver todos los eventos en orden cronológico que han modificado el objeto desde su creación hasta el momento actual.



Reflexion:

Bibliografia:

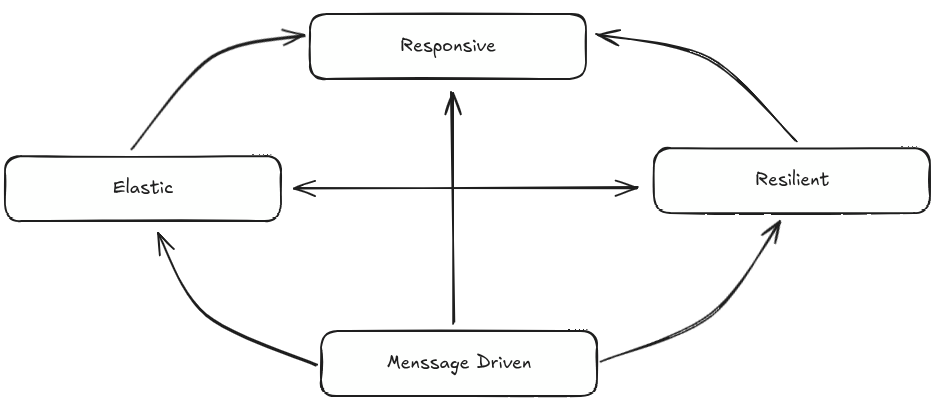
Ropero Morales, F. J. (2021). Aplicación de los patrones CQRS y Event/Sourcing en el desarrollo de un portal inmobiliario.

8.Arquitectura reactiva

La arquitectura reactiva se basa en cuatro principios fundamentales: Responsividad, Resiliencia, Elasticidad y Orientación a Mensajes, los cuales son esenciales para garantizar sistemas que se adapten a demandas de alta carga y baja latencia. Estos principios permiten que un sistema sea reactivo, respondiendo eficientemente a eventos y ajustándose a condiciones cambiantes, como la sobrecarga o la pérdida de disponibilidad.

En una arquitectura reactiva, el flujo de datos se maneja a través de flujos reactivos (como *Mono* y *Flux* en el caso de Java con Reactor), lo que permite que las operaciones se realicen de manera asíncrona, no bloqueante, y sin la necesidad de esperar respuestas de procesos anteriores. Esto mejora la resiliencia y el rendimiento general del sistema, al permitir que los recursos se gestionen de forma más eficiente.

Además, los sistemas reactivos están diseñados para ser distribuidos, con un enfoque de comunicación asíncrona y basada en mensajes, donde los componentes interactúan mediante eventos y comandos. Esto reduce el acoplamiento entre componentes y permite que el sistema sea más flexible y escalable, además de mejorar su tolerancia a fallos mediante el manejo adecuado del flujo de datos y el control de carga (backpressure).

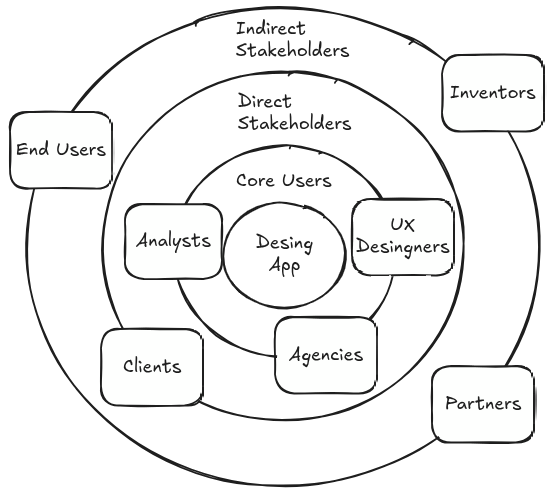


Reflexion:

Bibliografia:

|  |
| --- |
| Zapata Zapata, J. A. (2019). Arquitectura reactiva aplicada al desarrollo de software. |

9.Arquitectura basada en el modelo de actores

Los actores extienden los objetos secuenciales encapsulando un hilo de control junto con procedimientos y datos en la misma entidad. De esta manera, proveen una unidad de abstracción y distribución cuando se usa concurrencia. Los actores se comunican por paso de mensajes asincrónicos. A menos que se fuercen maneras de coordinación específicas, los mensajes se reciben en un orden arbitrario que puede diferir del orden de envío. Una implementación de este modelo normalmente provee para los mensajes una caja de correo local, y no hay garantía de que los mensajes sean procesados en el mismo orden en el cual son recibidos. Los actores son intrínsecamente independientes, concurrentes y autónomos, lo cual permite eficiencia en la ejecución paralela.

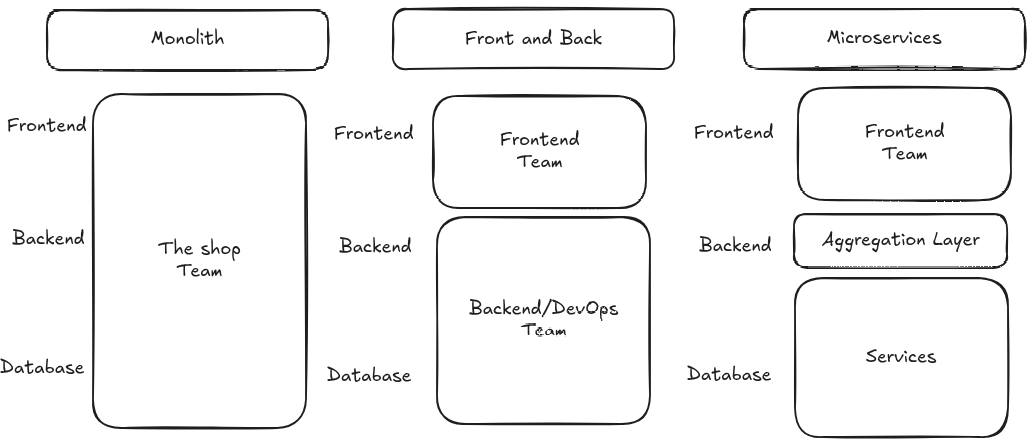
Reflexion:

Bilbiografia:

Varela Guevara, L., & Guerrero Serrano, J. M. (2004). Modelo de actores y la implementaión del portal de servicios para sistemas móviles en el lenguaje orientado a actores, SALSA

10.Arquitectura de micro-frontends

La arquitectura de microfrontends ha emergido como un enfoque que busca descomponer el frontend de una aplicación en partes más manejables, de manera similar a cómo los microservicios dividen el backend. Cada equipo se encarga de un microfrontend completo, tomando decisiones sobre las tecnologías, metodologías y plazos de entrega. Esto permite que el frontend se estructure en módulos independientes que se integran en un sistema más amplio.  
Su funcionamiento se basa en diversas formas de implementación. Una de ellas es a través de un frontend principal utilizando Single SPA, que funciona como un contenedor y orquesta los microfrontends desarrollados con diferentes tecnologías como React, Angular o Vue. También se pueden integrar microfrontends por ruta, aprovechando la infraestructura de un CDN como CloudFront, o mediante subdominios, donde cada proyecto se despliega de forma aislada y puede comunicarse con el dominio principal a través de mecanismos como cross-origin.  
La comunicación entre microfrontends se lleva a cabo mediante medios del cliente como local storage, session storage o cookies, que permiten almacenar y compartir información. Así, aunque cada microfrontend sea independiente y utilice distintas tecnologías, se puede mantener la interacción entre ellos dentro de un mismo sistema web.



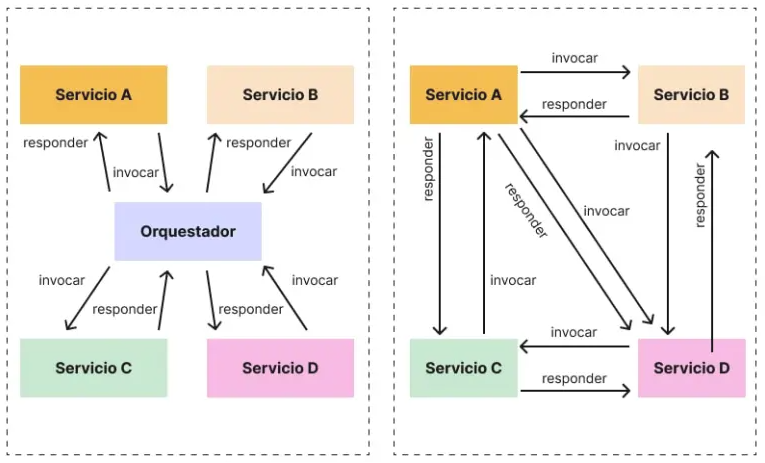
Reflexion:

Bibliografia:

Arango Amaya, L. D. (2021). Arquitecturas de micro Frontends, componentes web y librerías de componentes.

11.Arquitectura de orquestaciòn y coreografia de servicios.

Hablamos de orquestación y coreografía cuando se habla de la manera de acoplar  
varios microservicios juntos.  
Por un lado la orquestación hace referencia a una instancia, llamada director u  
orquestador, que tiene control del resto de microservicios de forma centralizada. Por otro  
lado la coreografía hace referencia a que cada microservicio se gestiona de manera propia y  
la aplicación resultante aparece con la suma de todos los microservicios.  
Un sistema basado en coreografía implica un mayor grado de libertad a la hora de  
gestionar los microservicios. Si aparece un evento determinado, cada microservicio actúa  
por su parte. Otra ventaja es que al agregar un microservicio nuevo, no es necesario  
acoplarlo al orquestador (como en la orquestación), simplemente responde si al escuchar el  
evento, es necesario. Como desventaja, al no existir un director, es difícil rastrear y conocer  
si todas las acciones requeridas se realizan con éxito. Aunque esto se podría solucionar con  
un microservicio adicional que compruebe si se activan los microservicios que deben  
ejecutarse al saltar un evento concreto.



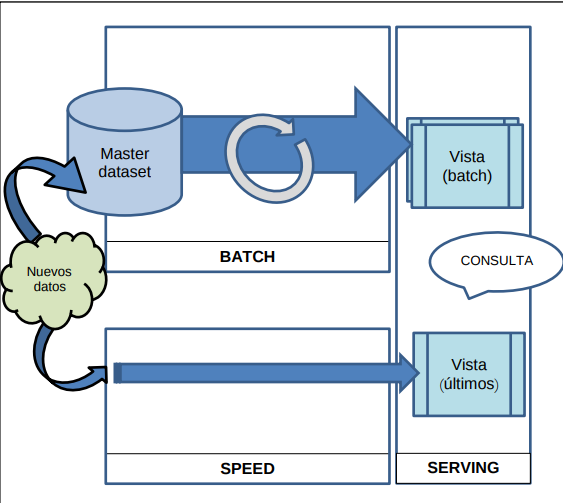
Reflexion:

Bibliografia:

Soler Castaño, J. F. (2021). Soluciones de integración en la WoT basadas en microservicios.

12.Arquitectura Lambda

La arquitectura Lambda pretende dar respuesta a los desafíos que plantea el Big Data mediante la organización en tres capas que interactúan entre sí.La capa Batch gestiona un conjunto de datos maestro que es inmutable y de por sí va creciendo con el tiempo.Su función principal es procesar esos datos de forma periódica, con el objeto de poder obtener vistas que representen resultados consolidados, aunque no en tiempo real, ya que se requieren varias horas para que se propaguen.  
Para suplir esa carencia, la capa Speed es la que se encarga de procesar los datos entrantes de manera inmediata.Esta arquitecta utiliza algoritmos incrementales que permiten obtener vistas rápidas con información reciente, pero con una calidad provisional, ya que después serán corregidas con el ciclo de la capa Batch.  
Por último, la capa Serving fusiona los resultados generados por ambas capas, la Batch y la Speed, para ser capaces de exponerlos para consultas de solo lectura.



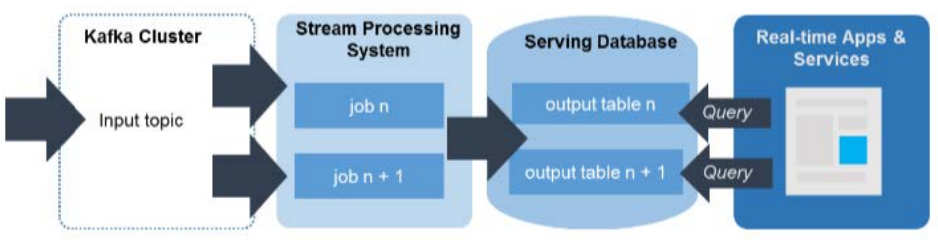
Reflexion:

Bibliografia:

Vallejo Martínez, A. (2015). Arquitectura lambda aplicada a clustering de documentos en contextos Big Data.

13.Arquitectura Kappa

La arquitectura Kappa ofrece una alternativa a Lambda para simplificar el procesamiento de datos; su funcionamiento consiste simplemente en eliminar la capa Batch y tener un solo sistema de procesamiento en tiempo real. De este modo, todos los datos son procesados en línea desde el momento en el que se introducen al sistema, evitando en consecuencia tener que mantener dos sistemas distintos y por lo tanto, simplificando la solución.De esta forma, la capa de real-time permitirá realizar las diversas actividades de procesamiento en línea así como aplicar las diversas actualizaciones de código. Esto va a permitir que al cambiar el código, los datos puedan ser reprocesados y sus resultados actualizados automáticamente, garantizando que la información recoja en consecuencia el cambio sin depender de procesos por lotes. Donde es necesario reprocesar, la arquitectura ejecuta una segunda instancia de la misma tarea que lleva a cabo el procesamiento en línea. Esta nueva instancia lleva a cabo el procesamiento a partir de la información que está almacenada en la base de datos, de modo que se actualizan los resultados de forma armónica sin duplicar arquitecturas ni separar responsabilidades de tareas entre diferentes capas.

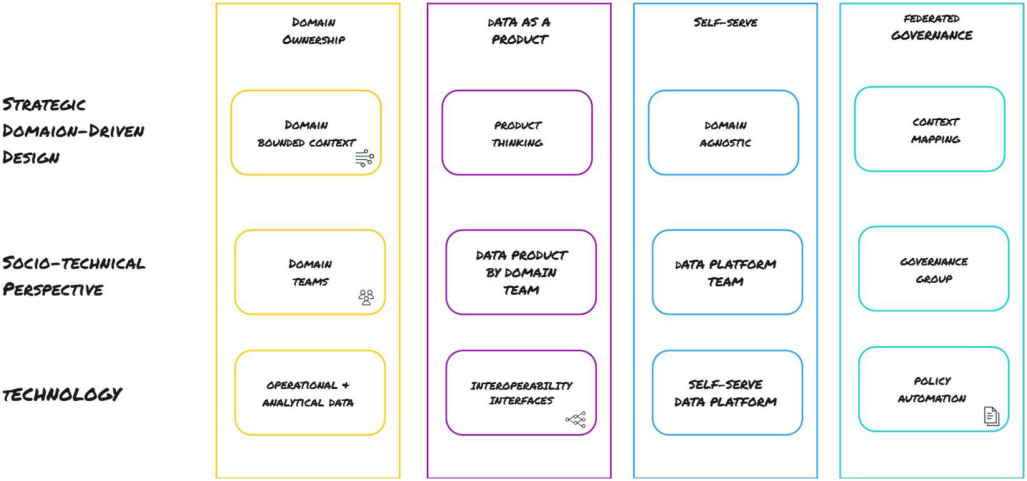
****

Reflexion:

Bibliografia:

Cantero Casares, A. (2019). Arquitectura Big Data para el procesamiento de señales en industria 4.0.

14.Arquitectura Data mesh

La **arquitectura Data Mesh** se fundamenta en el principio de **propiedad del dominio**, que establece que los equipos de cada dominio deben asumir la responsabilidad de sus propios datos. Los datos analíticos se estructuran alrededor de los dominios, alineándose con los límites del sistema, lo que transfiere la propiedad desde un equipo central de datos hacia los equipos de dominio dentro de una arquitectura distribuida.Otro principio clave es el de **datos como producto**, que aplica el pensamiento orientado a productos a los datos analíticos. Esto significa que los equipos de dominio no solo generan datos para sí mismos, sino que también deben satisfacer las necesidades de otros dominios, ofreciendo datos de alta calidad como si fueran una API pública que pueda ser consumida más allá de sus fronteras.El funcionamiento se completa con la **infraestructura de autoservicio** y la **gobernanza federada**. La primera proporciona herramientas y sistemas para que los equipos de dominio construyan y consuman productos de datos de manera independiente, mientras que la gobernanza federada asegura la interoperabilidad mediante estandarización. Así, se crea un ecosistema de datos que cumple reglas organizacionales y regulaciones de la industria, manteniendo autonomía y cohesión en la gestión de la información.

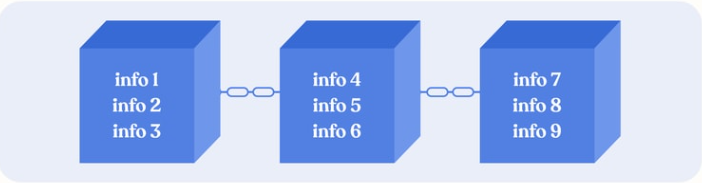
Reflexion

Bibliografia:

JAUMOT PEÑA, F. R. A. N. C. E. S. C. (2025). DESCENTRALIZACIÓN DE LA GESTIÓN DE LA CADENA DE SUMINISTRO MEDIANTE DATA MESH.

15.Arquitectura basada en blockchain

Es una estructura de almacenamiento que permanece encadenada entre cada bloque. Los datos se mantienen guardados en una copia completa sobre una estructura llamada “libro de transacción mayor” (ledger) que es algo similar a una base de datos, donde cada nodo tiene un fragmento de la información. Existe un bloque que es un tipo especial en la cadena de bloques, se llama nodo minero que se genera a través de algoritmos de consenso. El nodo minero empaqueta los datos o transacciones dentro de la red blockchain en bloques y las envía hacia el resto de los nodos, mientras tanto los otros nodos adicionan la transacción a la red generada a su copia. Blockchain es una cadena de bloques que se mantiene descentralizada mediante un mecanismo de consenso. En la red, una institución centralizada no mantiene el libro de transacciones maestro (ledger), sino que cada nodo participante mantiene su libro de transacciones (ledger), de modo que el verdadero estado del libro de transacciones puede estar garantizado contra la falsificación, siempre y cuando que menos de la mitad de los nodos participantes no sean nodos maliciosos (Hashim & Harous, 2021).



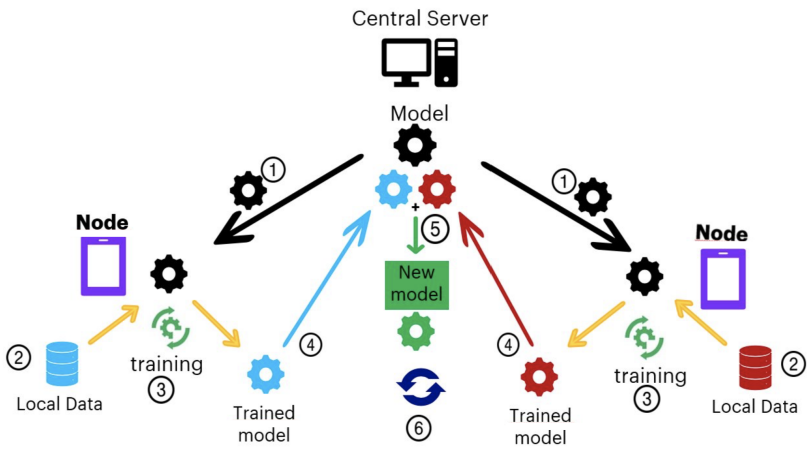
Reflexion:

Bibliografia:

Lara Moran, R. I. (2024). *Arquitectura de información para asegurar la trazabilidad en la distribución de medicina en el Ministerio de Salud Pública basado en Blockchain* (Bachelor's thesis).

16. Arquitectura de aprendizaje federado

El Aprendizaje Federado (FL) es un enfoque de entrenamiento distribuido que permite construir modelos de inteligencia artificial sin necesidad de centralizar los datos. En esta arquitectura, existe un servidor central que orquesta el proceso y múltiples clientes que poseen los datos localmente. El procedimiento se desarrolla en rondas: primero, el servidor envía los parámetros del modelo a los clientes; luego, cada cliente entrena el modelo con sus propios datos en varias épocas locales; al finalizar, devuelve los parámetros ajustados. Finalmente, el servidor combina esas actualizaciones mediante algoritmos como Federated Averaging, generando un modelo global que se redistribuye en la siguiente iteración. Este esquema resuelve el desafío de entrenar modelos sin acceso directo a todos los datos, garantizando mayor privacidad y eficiencia. En el caso de redes neuronales, se deben considerar tanto parámetros propios del entrenamiento local (batch-size, épocas) como parámetros federados (número de rondas, cantidad de clientes seleccionados y distribución de datos). De esta manera, el Aprendizaje Federado se presenta como una arquitectura clave para escenarios donde la privacidad y la descentralización son esenciales.



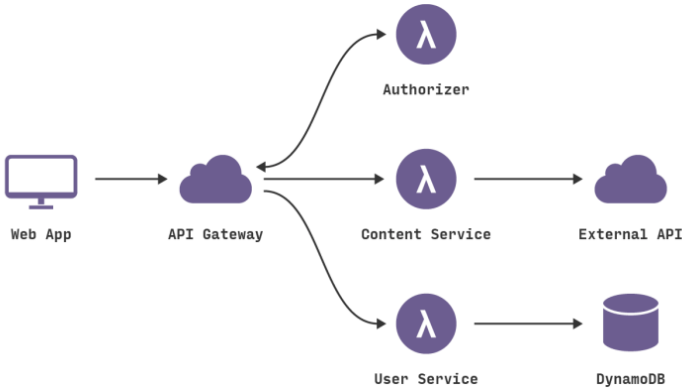
Reflexion:

Bibliografia:

Bermolen, P., Capdheourat, G., Etcheverry, L., Fachola, C., Fariello, M. I., & Tornaría, A. (2022). FLEA: Aprendizaje Federado aplicado a Analíticas de Aprendizaje.

17.Arquitectura Serverless

La arquitectura Serverless es un modelo de ejecución en el que el usuario puede correr su código sin necesidad de gestionar hardware ni sistemas operativos. Generalmente se implementa en la nube, lo que elimina la necesidad de almacenamiento físico y reduce los costes asociados a la infraestructura. En este esquema, el acceso a los recursos computacionales se da bajo demanda, a través de servicios independientes que se pagan únicamente cuando se utilizan, lo que lo hace un modelo dinámico y eficiente dentro del cloud computing. El funcionamiento se basa en el uso de funciones que se ejecutan de forma automática cuando se desencadena un evento. Por ejemplo, en Amazon Web Services (AWS) esto se logra mediante AWS Lambda, que permite ejecutar código empaquetado con librerías y procesar grandes volúmenes de datos de manera escalable. Cada vez que la función se activa, se crean contenedores temporales con un tiempo de vida limitado, donde se almacenan y procesan los resultados. A diferencia de los servidores tradicionales, donde se requiere hardware dedicado y costes fijos, Serverless ofrece flexibilidad, menor coste y escalabilidad, ya que solo se paga por el tiempo de ejecución y los recursos efectivamente consumidos.



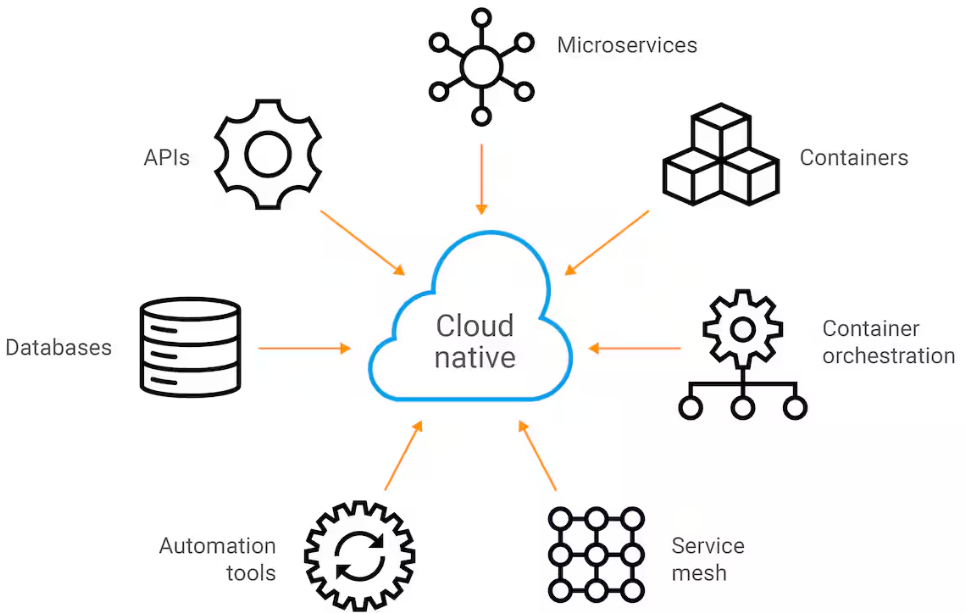
Reflexion:

Bibliografia:

Pacios Izquierdo, D. (2022). Arquitectura serverless para el procesado de datos y detección de anomalías en el instrumento MARSIS.

18. Arquitectura nativa en la nube

La arquitectura nativa en la nube es un enfoque de diseño que aprovecha las capacidades únicas del cloud frente a la infraestructura tradicional. Su objetivo es crear sistemas resilientes, escalables y fáciles de mantener, donde los recursos se consumen bajo demanda y se automatizan procesos de despliegue, recuperación y actualización continua (CI/CD). Su funcionamiento se apoya en varios principios. Uno es la automatización, que permite crear, escalar y reparar sistemas sin intervención manual. Otro es el diseño de componentes sin estado (stateless), lo que facilita balanceo de carga, recuperación ante fallos y escalado horizontal. También destaca el uso de servicios gestionados que reducen costos y complejidad, ya que el proveedor cloud administra la infraestructura. Además, aplica seguridad en profundidad, protegiendo cada componente de manera individual en lugar de depender de un perímetro fijo. Finalmente, se trata de una arquitectura en constante evolución, que se ajusta a nuevas necesidades y cambios del proveedor, garantizando flexibilidad, eficiencia y resiliencia.



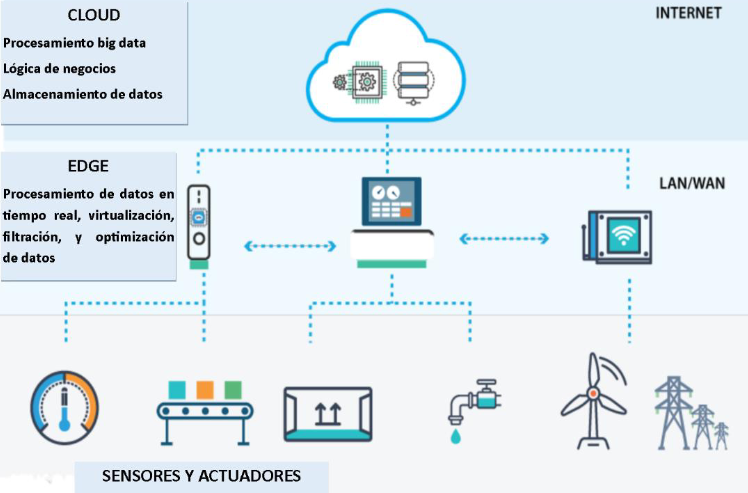
Reflexion

Bibliografia:

Grey, T. (2019, June 19). 5 principles for cloud-native architecture—what it is and how to master it. Google Cloud.

19. Arquitectura Edge Computing

La arquitectura de Edge Computing surge como respuesta al incremento de datos y a las limitaciones del modelo centralizado de la nube. Su principio consiste en ubicar los recursos de procesamiento y almacenamiento en el borde de la red, lo más cerca posible de las fuentes de datos y de los usuarios finales. De esta forma, se reducen los cuellos de botella en el ancho de banda, la latencia en las comunicaciones y los costos asociados al envío masivo de información hacia la nube. Esta arquitectura se organiza en capas. La primera está formada por los dispositivos finales como sensores, controladores o equipos IoT, responsables de la captura e ingestión de datos. Luego, los nodos edge se encargan del procesamiento, enrutamiento y operaciones informáticas intermedias. Finalmente, los servicios cloud realizan tareas de análisis avanzado, inteligencia artificial, aprendizaje automático y visualización, complementando el trabajo local con funciones más complejas.



Reflexion:

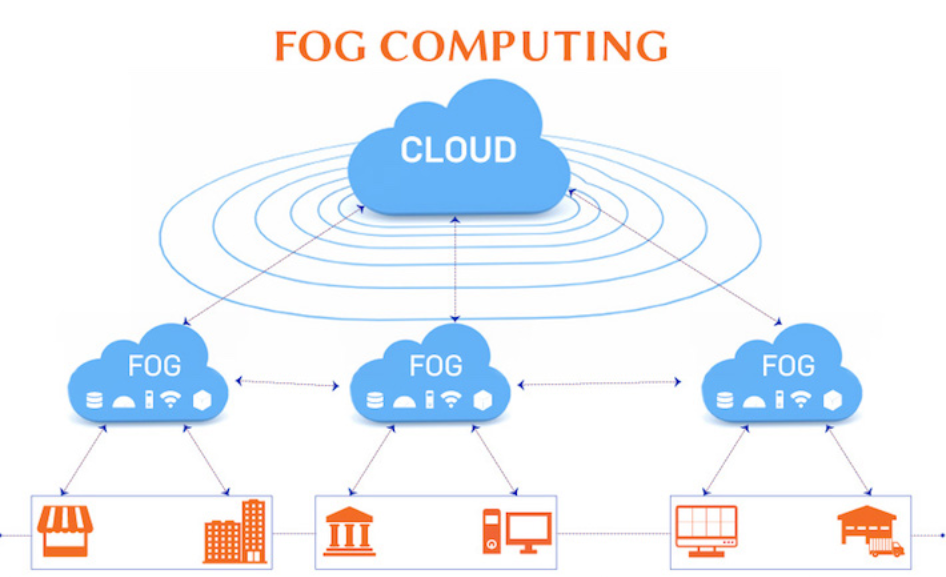
Bibliografia

Mela, J. L., Cedeño, G. D., & Herrera, E. C. (2021). Edge Computing: Aplicaciones y desafíos actuales. *Visión Antataura*, *5*(1), 75-91

20. Arquitectura Fog computing

La **Fog Computing** se define, según el Open Fog Consortium, como una arquitectura que **distribuye funciones de computación, almacenamiento, control e intercomunicación más cerca de los usuarios**, ubicándolas en el camino que conecta los dispositivos con la nube. A diferencia del modelo tradicional de **Cloud Computing**, donde el procesamiento ocurre en centros de datos centralizados, la niebla propone trasladar capacidades de análisis y decisión hacia los propios dispositivos o hacia nodos intermedios en el borde de la red.

Este enfoque permite que los datos generados masivamente por dispositivos IoT sean procesados **localmente**, enviando hacia niveles superiores solo la información ya **agregada y filtrada**. De esta manera, se reducen los grandes volúmenes de tráfico hacia la nube, se mejora la eficiencia en el uso de la red y se responde a las limitaciones de latencia. Así, la arquitectura de Fog Computing resulta especialmente útil en aplicaciones con **requisitos críticos en tiempo real**, donde depender exclusivamente de la nube generaría retrasos y sobrecarga en el sistema.



Reflexion:

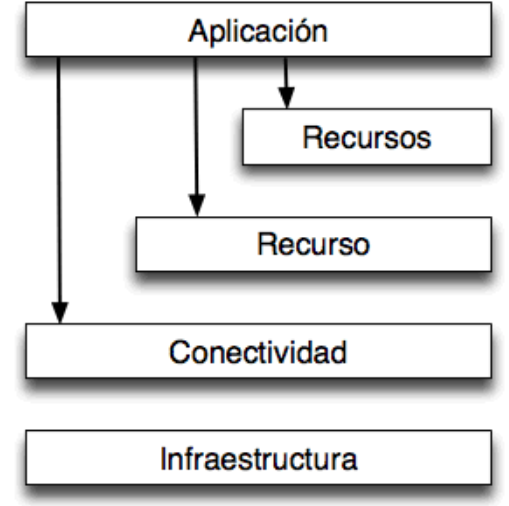
Bibliografia:

Lara Aragonés, D. (2017). Agentes inteligentes para Fog Computing.

21. Arquitectura Grid computing

La **tecnología Grid Computing** surge como respuesta a la necesidad de compartir y coordinar recursos de cómputo **geográficamente distribuidos**, principalmente en los ámbitos científico y de ingeniería. Según Foster, un **Grid** es un sistema que integra recursos no sujetos a un control centralizado, mediante el uso de **protocolos e interfaces abiertas y estándar**, con el fin de ofrecer servicios de calidad avanzada. Esta infraestructura permite la colaboración distribuida, el análisis de grandes volúmenes de datos, la ejecución de trabajos computacionales complejos y el acceso remoto a almacenamiento e instrumentación.

Entre sus **ventajas** destacan la reducción de costos en hardware, la disponibilidad de recursos bajo demanda y la gestión segura de operaciones. Actualmente, el **Open Grid Forum (OGF)** lidera su estandarización a través del **Web Services Resource Framework (WSRF)**, base para la **Open Grid Service Architecture (OGSA)**. Esta última, inspirada en la **arquitectura orientada a servicios (SOA)**, define la gestión de tareas, recursos y seguridad, apoyándose en tecnologías como **SOAP** y **WSDL** para la interoperabilidad.



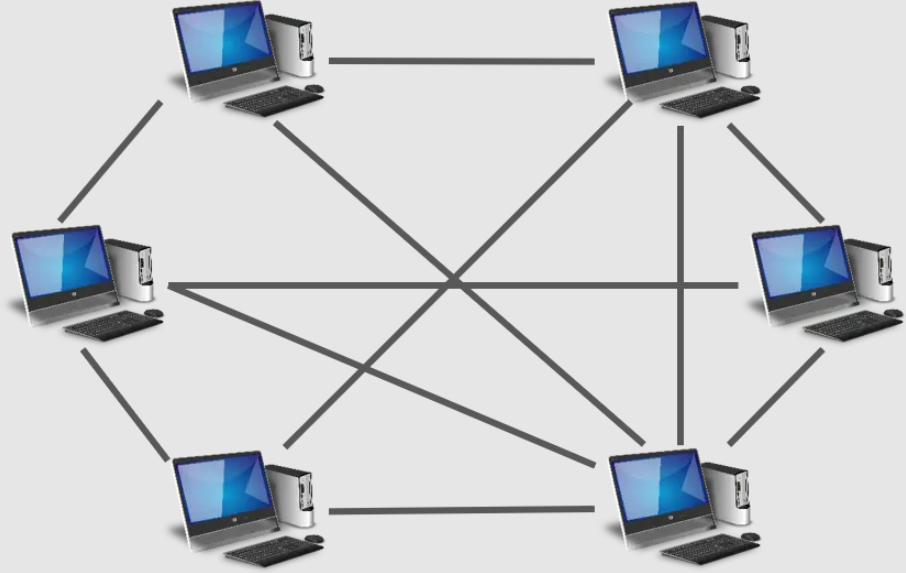
Reflexion:

Bibliografia:

Fernández-Rivas, J., & Siabato Vaca, W. L. (2010). Arquitectura GRID Computing como medio para la democratización e integración de datos LiDAR

22. Arquitectura Peer-to-Peer

Peer-to-peer es una arquitectura de software distribuida. Está compuesta por diferentes nodos o pares que se comportan como iguales entre sí, formando la denominada red de nodos. En esta arquitectura no será necesario un elemento central que administre la red, los nodos se comunicarán entre sí auto-organizando la misma. De este modo, los pares son tanto proveedores como consumidores de recursos, contrastando con el modelo de arquitectura tradicional cliente-servidor. La alta escalabilidad y la tolerancia a fallos son las dos grandes ventajas de esta arquitectura. El principal inconveniente reside en la complejidad de la administración de la red. La descentralización implica la generación de bastante tráfico y la existencia de mecanismos complejos para la gestión de la red. A pesar de que la arquitectura p2p busca la descentralización, existen variantes que renuncian parcialmente a esta para mitigar las desventajas asociadas a la complicada gestión de los nodos. Particularmente, en este trabajo solo serán tratadas las redes p2p puras, pues garantizan una alta descentralización y escalabilidad.



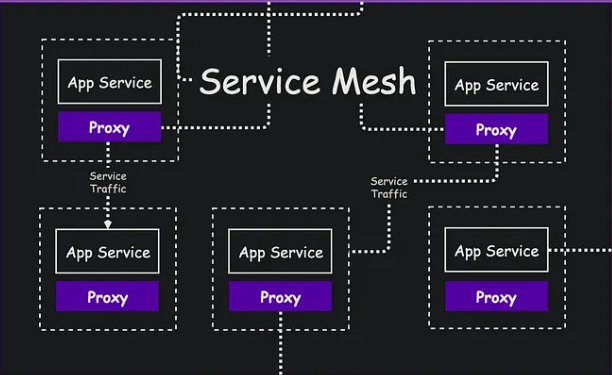
Reflexion:

Bibliografia:

Guijas Bravo, P. (2022). Desarrollo de una librería para el aprendizaje federado bajo una arquitectura peer-to-peer

23. Arquitectura Service Mesh

La arquitectura Service Mesh surge como un modelo que permite gestionar la comunicación entre distintos servicios en entornos distribuidos, más allá del esquema tradicional cliente-servidor. En este enfoque, los servicios pueden actuar tanto como clientes como servidores, apoyándose mutuamente para entregar al usuario la información requerida. De esta manera, se facilita el diseño de arquitecturas basadas en microservicios, donde la interacción entre componentes es transparente y flexible. Service Mesh ofrece soluciones en tres áreas principales: la observación, que permite recolectar telemetría del comportamiento de la plataforma; la gestión del tráfico, que configura y controla el flujo de datos; y la seguridad, que establece medidas apoyadas en la telemetría y el control del tráfico. Estas funciones, que antes recaían en las propias aplicaciones o en elementos de red, pasan a estar centralizadas en la malla de servicios. El Service Mesh separa la lógica de negocio de la lógica de control y seguridad, implementando ambas en capas diferenciadas dentro de la misma arquitectura. Esta segmentación proporciona beneficios sustanciales en términos de seguridad, observabilidad y gestión eficiente del tráfico.



Relflexion:

Bigliografia:

García Velasco, J. (2022). Implementación de protocolos seguros en arquitecturas Service Mesh con Istio.

24. Arquitectura basada en Contenedores

La arquitectura de microservicios basada en contenedores emerge como un paradigma transformador que permite desacoplar funcionalidades complejas en servicios autónomos y ligeros. La contenerización proporciona un mecanismo estandarizado para empaquetar aplicaciones junto con todas sus dependencias, garantizando portabilidad y consistencia; cada aplicación se ejecuta en su propio espacio aislado con virtualización ligera, reduciendo la sobrecarga computacional. La orquestación de contenedores gestiona de manera automatizada el despliegue, escalado y operación de múltiples contenedores; Kubernetes asegura disponibilidad, rendimiento y resiliencia. La escalabilidad horizontal expande capacidades mediante la adición progresiva de recursos y la redistribución dinámica de la carga. El marco referencial resalta desacoplamiento de servicios, comunicación asíncrona y tolerancia a fallos. Complementan la arquitectura el aislamiento y segmentación de servicios (políticas de red, mínimos privilegios) y el monitoreo y observabilidad distribuida con Prometheus, Grafana y distributed tracing para visibilidad integral y optimización continua.

Reflexion:

Bibliografia:

Decimavilla-Alarcón, D. C., & Marcillo-Franco, P. F. (2025). Arquitectura de microservicios basada en contenedores para despliegue ágil de aplicaciones IoT en la nube. *Revista Científica Episteme & Praxis*, *3*(1), 35-49.

25.Arquitectura basada en Microkernel

La arquitectura de Microkernel, también llamada de plugins, separa una aplicación en un Core mínimo y múltiples plugins/módulos. El Core concentra funciones esenciales; los plugins amplían sin modificar el núcleo, siguiendo el principio Open-Closed. Para habilitar la extensibilidad, el Core publica un API independiente (interfaces/clases, p. ej., “Plugin”) y exige descriptores para instalación e integración; así, equipos distintos pueden desarrollar módulos en paralelo. Ejemplos: IDEs como Eclipse, NetBeans, Visual Studio/VS Code y suites ofimáticas (Word, Excel), que añaden conectores y nuevas capacidades. Ventajas: alta testabilidad al aislar core y plugins; buen rendimiento una vez instalados (ejecución “monolítica”); despliegue sencillo e incluso en caliente; activación bajo demanda según perfiles; fuerte modularidad y reutilización/comercialización de plugins. Desventajas: alta complejidad técnica y de análisis para preservar la esencia del Core y definir límites de extensibilidad; escalabilidad limitada en escenarios típicamente standalone, con algunas excepciones. En síntesis, ofrece gran personalización y ecosistema de terceros a cambio de mayor dificultad y posibles límites de escala.

Gráfico en cascada

El contenido generado por IA puede ser incorrecto.

Reflexion:

Bibliografia:

Arquitectura de Microkernel. (s. f.). Reactive Programming. Recuperado el 20 de septiembre de 2025, de <https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/microkernel>

26.Arquitectura basada en componentes

Diagrama

El contenido generado por IA puede ser incorrecto.La arquitectura basada en componentes divide el software en unidades modulares, portátiles y reemplazables con interfaces bien definidas (métodos, eventos, propiedades). Su objetivo central es maximizar la reutilización: cada componente encapsula funcionalidad y puede integrarse de forma autónoma, favoreciendo mantenibilidad y escalabilidad. Se apoya en tecnologías como COM/DCOM, JavaBeans, EJB, CORBA, .NET y servicios web, comunes tanto en GUIs (botones, grids) como en componentes “invisibles” empresariales. Se entiende desde tres vistas: orientada a objetos (conjuntos de clases e interfaces), convencional (módulos con lógica y datos internos) y relacionada con procesos (ensamblaje desde bibliotecas existentes). Sus rasgos clave incluyen reutilización, reemplazo, independencia del contexto, extensibilidad, encapsulamiento e independencia. Principios y guías: definir interfaces claras (puertos requeridos/proporcionados), conectar vía conectores e interacciones (invocaciones sincrónicas/asíncronas, mensajes, flujo de datos), modelar dependencias mediante interfaces, usar nomenclatura del dominio y considerar puntos de extensión tipo plug-in. La realización incluye identificar clases de dominio e infraestructura, diseñar interfaces, datos persistentes y comportamientos.

Reflexion:

Bibliografia:

Tutoriales.edu.lat. (s. f.). Arquitectura basada en componentes. Recuperado el 20 de septiembre de 2025, de [Arquitectura basada en componentes](https://tutoriales.edu.lat/pub/software-architecture-design/component-based-architecture/arquitectura-basada-en-componentes)