

**Carrera:**

Ingeniería de Sistemas

**Investigación 1:**

“Arquitectura de Software”

**Nombre:**

Dilan Elmer Poma Mamani

**Fecha:**

29-09-2023

**Materia:**

Lenguaje Audiovisual

**Docente:**

ING. Luis Jimenez Peña

**Arquitecturas de Software**

La arquitectura de software se refiere a la estructura fundamental y organización de un sistema de software. Es el diseño de alto nivel que define la manera en que los componentes del software interactúan entre sí, cómo se gestionan los datos y la lógica de negocio, y cómo se logran los objetivos del sistema. La arquitectura de software proporciona una visión global de la aplicación, delineando su estructura, componentes, módulos y las relaciones entre ellos.

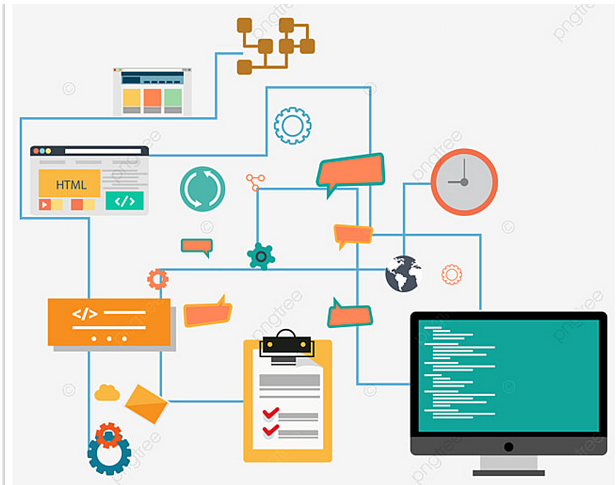
Debido a la cantidad de datos que se generan por segundo, es necesaria una serie de mecanismos que faciliten la administración de la información. Entre ellas se encuentra la Arquitectura de Software. Sin embargo, este es un tema muy extenso, puesto que la arquitectura de sistema abarca una gran variedad de herramientas, procesos, modelos y mecanismos para llevar a cabo su tarea en la gestión de datos.

Por ello, llevar a cabo una arquitectura de sistema, a nivel general, requiere de una serie de conocimientos Big Data junto a una preparación en guías y métodos de estructuración de la información. En este post, te introducimos a lo que es una arquitectura de Software y algunos términos clave dentro de una arquitectura.

**¿Qué es la arquitectura de software?**

**La arquitectura de Software hace referencia a la estructura y la relación entre las diferentes partes de un software y sus propiedades visibles externas. En suma, una arquitectura de Software está compuesta por más arquitecturas de datos articuladas entre sí. Esta es la razón por la que la arquitectura de sistema abarca tantos elementos y herramientas para llevarse a cabo.**

**El principal objetivo de la arquitectura y diseño de software radica en ofrecer cierta calidad al sistema de administración de datos, a partir de su desempeño, ahorro de tiempo, su disponibilidad y usabilidad, la capacidad de modificarse y adecuarse a las nuevas necesidades del sistema, entre otros atributos de calidad. Implantar una arquitectura de software capacitada para los intereses de una compañía deriva en la reducción de costos, mejora en la toma de decisiones, efectividad de rutas de acción y proyección acertada.**



Por último, ten en cuenta que una **arquitectura de software** o arquitectura y diseño de software requiere de un determinado ciclo de etapas que constituyen sistemas: **requisitos, diseño, documentación y evaluación de su funcionamiento.**

Una vez cumplas con este ciclo, será mucho más probable que se **mantenga la calidad de datos y su confiabilidad.** Como arquitecto de software deberás **priorizar la información y poseer un amplio conocimiento de tecnología.**

**Algunos términos de Arquitectura**

Ahora que sabes qué es la arquitectura de software, hablemos de los términos que se relacionan. A continuación, te compartimos siete términos imprescindibles a la hora de desarrollar una arquitectura de datos o de una arquitectura y diseño de software, a nivel general.

1. **Disponibilidad:** se refiere a la posibilidad de acceder a la base de datos de las arquitecturas de software. Esta debería estar disponible todo el tiempo.
2. **Latencia:** se refiere a la distancia que puede existir entre los datos. La arquitectura de datos debe apuntar a una unificación.
3. **Topología/conectividad/geografía:** de igual manera, consta de la relación unificada de los datos entre sí.
4. **Redundancia:** un arquitecto de software necesita saber duplicar los servidores, porque si uno se cae tendrá que tener otro.
5. **Gobierno**: es el factor que administra la segmentación de las arquitecturas de software. Se debe tener en cuenta su influencia en el desarrollo de una arquitectura de datos.
6. **Seguridad:** la arquitectura de datos debe proteger la información en su desarrollo.
7. **Elasticidad:** un arquitecto de software debe poder gestionar desde poca cantidad de datos hasta un gran volumen.

Asegúrate de considerar cada una de estas características cuando vayas a desarrollar una arquitectura de datos o arquitecturas de software. Cada uno de estos términos se encarga de tareas fundamentales para crear una estructura de la información efectiva, confiable y duradera, así como saber qué es la arquitectura de software.

**Tipos de Arquitectura de Software**

En la actualidad, existen varias arquitecturas de software utilizadas en diferentes contextos y aplicaciones. La elección de la arquitectura de software depende de las necesidades y requisitos específicos de un proyecto. Algunas de las arquitecturas de software más utilizadas en la actualidad incluyen:

1. **Arquitectura de Microservicios:** En esta arquitectura, una aplicación se divide en múltiples servicios independientes y autónomos que se comunican entre sí a través de API. Esto permite una mayor escalabilidad y flexibilidad, ya que cada servicio puede desarrollarse, implementarse y escalar por separado.
2. **Arquitectura de Monolito:** Es el enfoque tradicional donde una aplicación se desarrolla como una sola unidad. Aunque puede ser menos flexible que la arquitectura de microservicios, sigue siendo común en aplicaciones más pequeñas y proyectos donde la escalabilidad no es una preocupación inmediata.
3. **Arquitectura Orientada a Eventos:** Esta arquitectura se basa en la generación y el procesamiento de eventos. Los componentes del sistema se comunican mediante la emisión y escucha de eventos, lo que facilita la integración y la escalabilidad.
4. **Arquitectura de Capas:** Divide una aplicación en capas lógicas, como la capa de presentación, la capa de lógica de negocios y la capa de acceso a datos. Cada capa tiene una responsabilidad específica, lo que facilita el mantenimiento y la escalabilidad.
5. **Arquitectura de Contenedores:** Utiliza contenedores (como Docker) para empaquetar aplicaciones y todas sus dependencias en una unidad única y portátil. Esto simplifica la implementación y la gestión de aplicaciones en diferentes entornos.
6. **Arquitectura Serverless:** En este enfoque, las aplicaciones se ejecutan en entornos sin servidor, lo que significa que no es necesario administrar la infraestructura subyacente. Se paga solo por el tiempo de ejecución y los recursos utilizados.
7. **Arquitectura de Cómputo sin Estado:** Cada solicitud o proceso se maneja de forma independiente, sin estado. Esto facilita la escalabilidad y la distribución de la carga de trabajo.
8. **Arquitectura de Microfrontend:** Similar a la arquitectura de microservicios, pero se aplica al frontend de una aplicación. Permite desarrollar y escalar las partes del frontend de manera independiente.
9. **Arquitectura Hexagonal (Puertos y Adaptadores):** Define una arquitectura centrada en el dominio del problema, donde los puertos representan las interfaces y los adaptadores son implementaciones concretas. Esto facilita la sustitución de componentes y la integración de sistemas.
10. **Arquitectura sin Código (No-Code/Low-Code):** Estas arquitecturas permiten desarrollar aplicaciones con un esfuerzo mínimo de codificación. Son ideales para aplicaciones empresariales y flujos de trabajo automatizados.

La elección de la arquitectura adecuada dependerá de factores como los requisitos del proyecto, la escalabilidad, el rendimiento, la facilidad de mantenimiento y otros aspectos específicos del contexto de desarrollo. Además, es común que se utilicen enfoques híbridos o adaptaciones personalizadas para satisfacer las necesidades particulares de un proyecto.

**Arquitectura de Microservicios:**

No hay una única arquitectura de software que se considere como la más utilizada en todos los contextos, ya que la elección de la arquitectura depende de los requisitos y necesidades específicos de cada proyecto. En la actualidad, las arquitecturas más utilizadas varían según el tipo de aplicación y el entorno en el que se desarrolla el software.

La en aplicaciones empresariales complejas y altamente escalables, la arquitectura de microservicios se ha vuelto muy popular debido a su capacidad para dividir la aplicación en componentes independientes que pueden escalarse y desplegarse de forma individual. Sin embargo, en proyectos más pequeños o aplicaciones monolíticas, la arquitectura de monolito sigue siendo común.

Las arquitecturas de microservicios proporcionan un sistema moderno altamente escalable y distribuido

Modernizar las aplicaciones en el mundo actual a menudo conlleva migrar a aplicaciones nativas de la nube creadas como microservicios. A continuación, se implementan mediante tecnologías de contenedores como Docker y Kubernetes. Netflix y Atlassian lo hicieron, así como infinidad de otras organizaciones. El motivo: una arquitectura de microservicios mejora la escalabilidad, la velocidad del desarrollo y la iteración de servicios.

Una arquitectura de microservicios divide una aplicación en una serie de servicios implementables de forma independiente que se comunican a través de API. Este enfoque permite implementar y escalar cada servicio individual de forma independiente, así como la entrega rápida y frecuente de aplicaciones grandes y complejas. A diferencia de una aplicación monolítica, una arquitectura de microservicios permite a los equipos implementar nuevas funciones y hacer cambios más rápido, sin tener que volver a escribir una gran parte del código existente.

**Estas son algunas de las características fundamentales de las arquitecturas de microservicios:**

**Varios servicios de componentes**

Los microservicios se componen de servicios de componentes individuales y poco vinculados que se pueden desarrollar, implementar, operar, cambiar y volver a implementar sin afectar al funcionamiento de otros servicios o a la integridad de una aplicación. Esto permite una implementación rápida y fácil de cada una de las funciones de una aplicación.

**Muy fáciles de mantener y de probar**

Los microservicios permiten a los equipos experimentar con nuevas funciones y revertirlas si no funcionan. Esto facilita la actualización del código y acelera el tiempo de salida al mercado de nuevas funciones. Además, simplifica el proceso de aislamiento y corrección de fallos y errores en los servicios individuales.

**Pertenecen a equipos pequeños**

Los equipos pequeños e independientes suelen crear un servicio dentro de microservicios, lo que los anima a adoptar prácticas de metodología ágil y de DevOps. Los equipos pueden trabajar de forma independiente y moverse rápidamente, lo que acorta el ciclo de desarrollo.

**Se organizan en torno a capacidades empresariales**

Un enfoque de microservicios permite organizar los servicios en torno a las capacidades empresariales. Los equipos son multifuncionales, disponen de la gama completa de habilidades necesarias para el desarrollo y trabajan para crear una funcionalidad concreta.

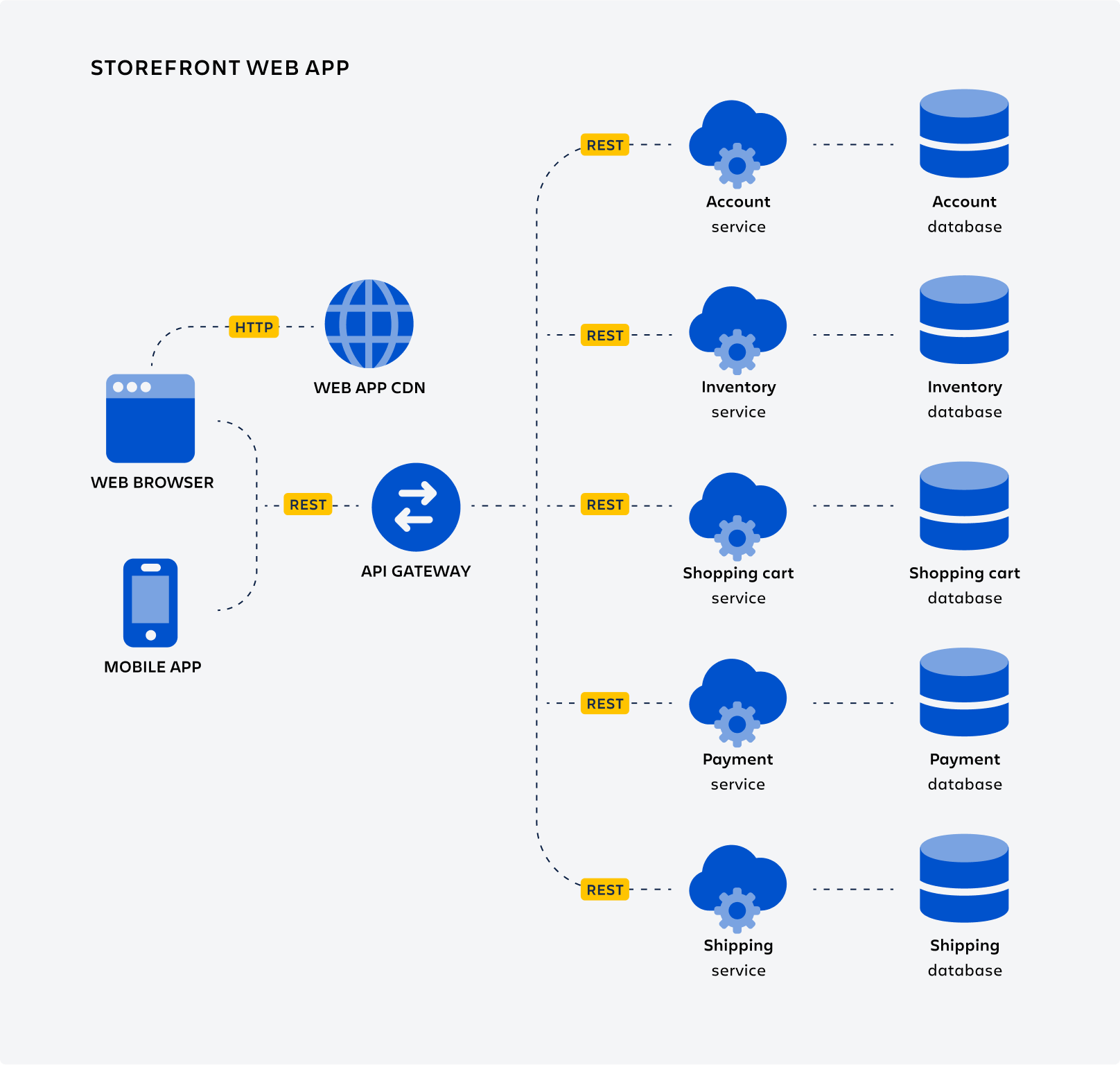
**Infraestructura automatizada**

Los equipos que crean y mantienen microservicios suelen utilizar prácticas de automatización de infraestructuras, como la integración continua (CI), la entrega continua (CD) y la implementación continua (también CD). Esto permite a los equipos crear e implementar cada servicio de forma independiente sin afectar a los demás equipos, así como implementar una nueva versión de un servicio en paralelo con la versión anterior.

**Ejemplo de arquitectura de microservicios**

Pongamos como ejemplo un hipotético proyecto de software de comercio electrónico. El siguiente diagrama es de un sitio de comercio electrónico con una aplicación web y una aplicación móvil que interactúan con varios microservicios, cada uno de los cuales proporciona funciones específicas para un dominio.

Las aplicaciones web modernas se ejecutan en navegadores y, a menudo, se sirven desde una red de distribución de contenido (CDN). Las CDN proporcionan la ventaja de poder distribuir aplicaciones web a servidores de todo el mundo, para que los navegadores web las puedan descargar rápidamente. También se utilizan para ofrecer recursos multimedia, como imágenes, audio y vídeo. Por ejemplo, en este sistema las imágenes y los vídeos de los productos a la venta se sirven desde la CDN.



**Los microservicios de este gráfico son los siguientes:**

**Servicio de cuentas** El servicio de cuentas proporciona información sobre la cuenta del cliente, como la dirección y la información de pago.

**Servicio de inventario** Ofrece información de inventario actualizada sobre los bienes que el cliente puede comprar.

**Servicio de carrito de la compra** Los clientes lo usan para seleccionar los productos del inventario que quieren comprar.

**Servicio de pago** Los clientes lo usan para pagar por los productos que han añadido al carrito de la compra.

**Servicio de envío** Programa el embalaje y la entrega de los bienes adquiridos.

Las aplicaciones interactúan con los microservicios a través de las API de REST que publica cada uno de los microservicios. Una puerta de enlace de API permite que las aplicaciones se basen en las API proporcionadas por los microservicios y permite que se intercambien unos microservicios por otros con la misma API.

Cada microservicio se compone de un servicio y una base de datos. Los servicios gestionan la API de REST, implementan la lógica empresarial y almacenan datos en una base de datos. Los recursos de los distintos microservicios, como bases de datos y colas, se aíslan siguiendo el contrato de 12 Factor App

**Cómo crear microservicios**

Muchas organizaciones empiezan con una arquitectura monolítica. Luego, hay que dividir una base de código en varios servicios, implementar los patrones correctos para fallar de forma limpia y recuperarte de las incidencias en la red, lidiar con la coherencia de los datos, supervisar la carga de servicio, etc. Y esto teniendo en cuenta solo la parte técnica. También hay que reorganizar los equipos y, muy probablemente, adoptar una cultura de DevOps.

Luego viene la parte difícil: descomponer el monolito en microservicios. La refactorización de un esquema de base de datos monolítico puede ser una operación delicada. Es importante determinar con claridad qué conjuntos de datos necesita cada servicio y las superposiciones. La entrega continua ayuda a reducir los riesgos de fallos de publicación, así como a conseguir que el equipo se centre en crear y ejecutar la aplicación, en lugar de quedarse atascado en implementarla.

**Comparación entre la arquitectura monolítica y la arquitectura de microservicios**

Una arquitectura monolítica es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. Una arquitectura de microservicios es el concepto opuesto al de la arquitectura monolítica, ya que es un método que se basa en una serie de servicios que se pueden implementar de forma independiente. La arquitectura monolítica puede resultar práctica al principio de un proyecto para aliviar la sobrecarga cognitiva de la gestión de código, así como la implementación. Pero una vez que una aplicación monolítica se vuelve grande y compleja, resulta difícil escalarla, la implementación continua pasa a ser un desafío y las actualizaciones pueden resultar complicadas.

Si bien una aplicación monolítica se crea como una sola unidad indivisible, los microservicios dividen esa unidad en una colección de unidades independientes que contribuyen a un todo más amplio. Una aplicación se construye como una serie de servicios que se pueden implementar de forma independiente, están descentralizados y se desarrollan de forma autónoma.