PROCESO DIRECCIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION.
- Código del Programa de Formación: 228106
- Nombre del Proyecto:
- Fase del Proyecto: PLANEACIÓN
- Actividad de Proyecto: DISEÑAR LA ESTRUCTURA TECNOLÓGICA DEL SISTEMA INTEGRAL
- Competencia: PARTICIPAR EN EL PROCESO DE NEGOCIACIÓN DE TECNOLOGÍA INFORMÁTICA PARA PERMITIR LA IMPLEMENTACIÓN DEL SISTEMA DE INFORMACIÓN.
- Resultados de Aprendizaje: DEFINIR ESTRATEGIAS PARA LA ELABORACIÓN DE TÉRMINOS DE REFERENCIA Y PROCESOS DE EVALUACIÓN DE PROVEEDORES, EN LA ADQUISICIÓN DE TECNOLOGÍA, SEGÚN PROTOCOLOS ESTABLECIDOS.
- Duración de la Guía

2. PRESENTACIÓN

Para muchos, dominar la arquitectura de software es uno de los objetivos que han buscado durante algún tiempo, sin embargo, no siempre es claro el camino para dominarla, ya que la arquitectura de software es un concepto abstracto que cada persona lo interpreta de una forma diferente, dificultando con ello comprender y diseñar con éxito una arquitectura de software.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- Materiales:
  Git, Visual code
  Portatil ó Computador de Escritorio
- Ambiente Requerido
- Descripción de la(s) Actividad(es)

EVOLUCIÓN DEL DESARROLLO DE SOFTWARE

En la imagen 1 se visualizan los grandes hitos de la evolución del desarrollo de software, las necesidades emergentes de cada uno de esos momentos y los avances tecnológicos que surgieron como solución. Se pone énfasis en la necesidad presente desde el inicio de contar con estrategias y técnicas que permitieran el desarrollo de productos confiables en el marco de proyectos predecibles. Esta necesidad fue la que motivó la creación de la Ingeniería de Software

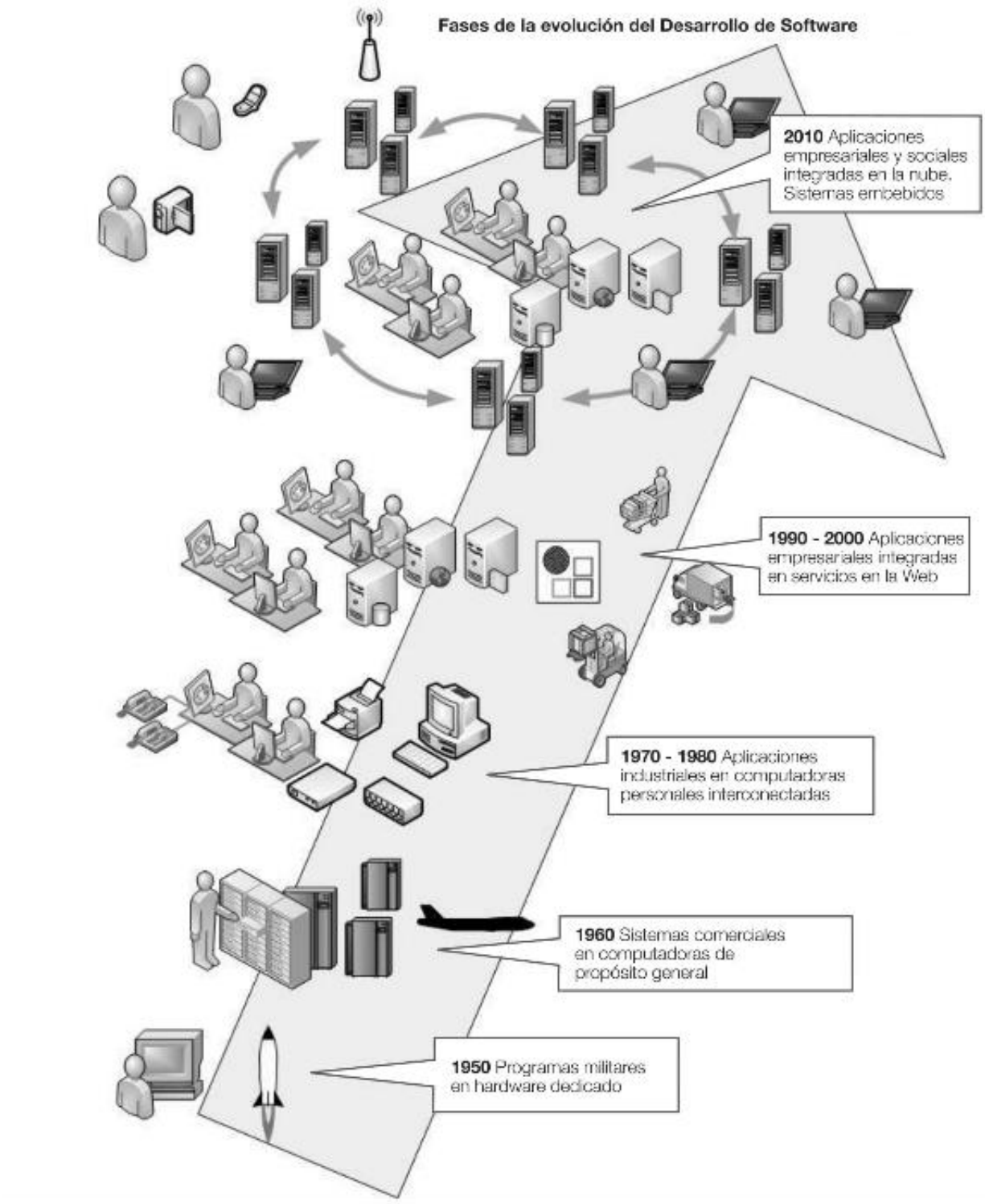


Imagen 1 – Evolución del desarrollo de software

ARQUITECTURA DE SOFTWARE

Antes de comenzar revisemos algunas definiciones de arquitectura de software:

“La arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema “

— "An introduction to Software Architecture" de David Garlan y Mary Shaw

“La Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. “

— Software Engineering Institute (SEI)

“El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos, y las propiedades de ambos. “

— Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al, AddisonWesley, 2010

“La arquitectura de software de un programa o sistema informático es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos. “

— Software Architecture in Practice (2nd edition), Bass, Clements, Kazman; AddisonWesley 2003

¿Que tienen en común las anteriores definiciones?

Todas coinciden en que la arquitectura **se centra en la estructura del sistema, los componentes que lo conforman y la relación que existe entre ellos**.

PATRONES DE DISEÑO

Los patrones de diseño tienen un impacto relativo con respecto a un componente, esto quiere decir que tiene un impacto menor sobre todo el componente. Dicho de otra forma, si quisiéramos quitar o remplazar el patrón de diseño, solo afectaría a las clases que están directamente relacionadas con él, y un impacto imperceptible para el resto de componentes que conforman la arquitectura.

A principios de la década de 1990 fue cuando los patrones de diseño tuvieron su gran debut en el mundo de la informática a partir de la publicación del libro Design Patterns, escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogían 23 patrones de diseño comunes que ya se utilizaban sin ser reconocidos como patrones de diseño.

Es importante mencionar que la utilización de patrones de diseño demuestra la madurez de un programador de software ya que utiliza soluciones probadas para problemas concretos que ya han sido probados en el pasado. Toma en cuenta que el dominio de los patrones de diseño es una práctica que se tiene que

perfeccionar y practicar, es necesario conocer las ventajas y desventajas que ofrece cada uno de ellos, pero sobre todo requiere de experiencia para identificar dónde se deben de utilizar.

Lo más importante de utilizar los patrones de diseño es que evita tener que reinventar la rueda, ya que son escenarios identificados y su solución está documentada y probada por lo que no es necesario comprobar su efectividad. Los patrones de diseño se basan en las mejores prácticas de programación.

Los patrones de diseño pretenden:

 Proporcionar un catálogo de soluciones probadas de diseño para problemas comunes conocidos.

 Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.

 Crear un lenguaje estándar entre los desarrolladores. Facilitar el aprendizaje a nuevas generaciones de programadores.

Asimismo, no pretenden:

 Imponer ciertas alternativas de diseño frente a otras.

 Imponer la solución definitiva a un problema de diseño.

 Eliminar la creatividad inherente al proceso de diseño.

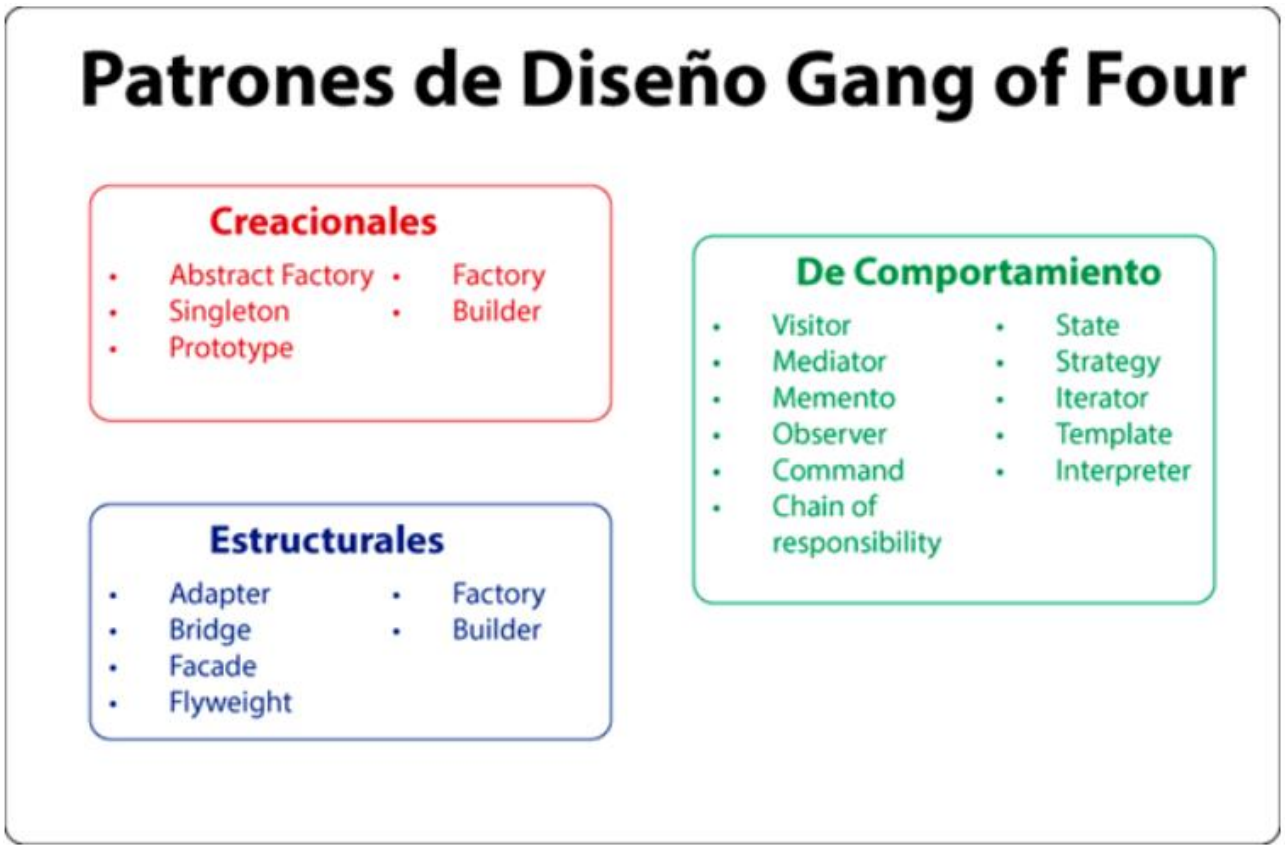


Imagen 2 – Lista de los patrones de diseño

PATRONES ARQUITECTONICOS

Los patrones arquitectónicos tienen un gran impacto sobre el componente, lo que quiere decir que cualquier cambio que se realice una vez construido el componente podría tener un impacto mayor.

A continuación, veremos las principales definiciones de los patrones arquitectónicos.

“Expresa una organización estructural fundamental o esquema para sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos. “

— TOGAF

“Los patrones de una arquitectura expresan una organización estructural fundamental o esquema para sistemas complejos. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades únicas e incluye las reglas y pautas que permiten la toma de decisiones para organizar las relaciones entre ellos. El patrón de arquitectura para un sistema de software ilustra la estructura de nivel macro para toda la solución de software. Un patrón arquitectónico es un conjunto de principios y un patrón de grano grueso que proporciona un marco abstracto para una familia de sistemas. Un patrón arquitectónico mejora la partición y promueve la reutilización del diseño al proporcionar soluciones a problemas recurrentes con frecuencia. Precisamente hablando, un patrón arquitectónico comprende un conjunto de principios que dan forma a una aplicación.”

— Achitectural Patterns - Pethuru Raj, Anupama Raman, Harihara Subramanian

“Los patrones de arquitectura ayudan a definir las características básicas y el comportamiento de una aplicación.”

— Software Architecture Patterns - Mark Richards

Tanto los patrones de diseño cómo los patrones arquitectónicos pueden resultar similares ante un arquitecto inexperto, pero por ninguna razón debemos confundirlos, ya son cosas diferentes.

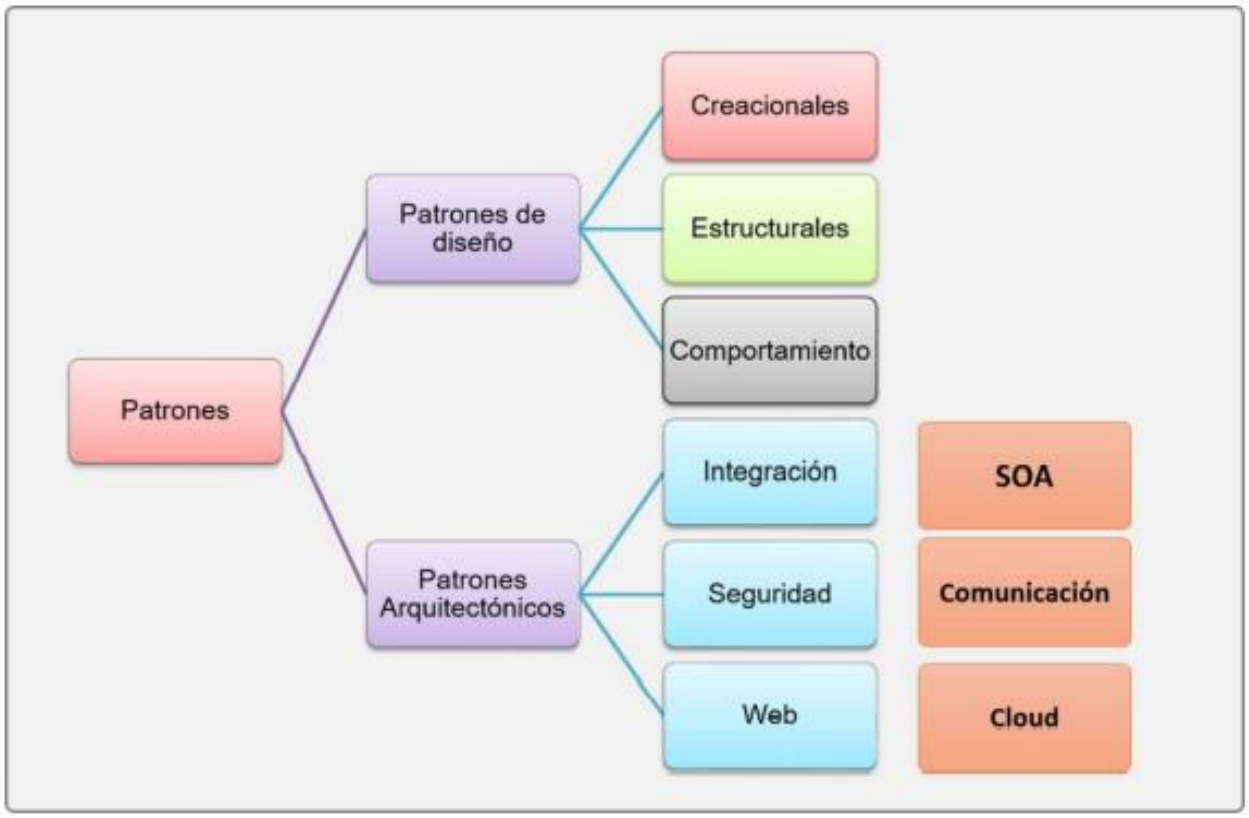


Imagen 3 – Consolidado de patrones

¿CÓMO DIFERENCIAR UN PATRÓN ARQUITECTÓNICO?

Los patrones arquitectónicos son fáciles de reconocer debido a que tiene un impacto global sobre la aplicación, e incluso, el patrón rige la forma de trabajar o comunicarse con otros componentes, es por ello que a cualquier cambio que se realice sobre ellos tendrá un impacto directo sobre el componente, e incluso, podría tener afectaciones con los componentes relacionados.

Un ejemplo típico de un patrón arquitectónico es el denominado “Arquitectura en 3 capas”, el cual consiste en separar la aplicación 3 capas diferentes, las cuales corresponden a la capa de presentación, capa de negocio y capa de datos:

Imagen 4 – Arquitectura de capas

ESTILOS ARQUITECTÓNICOS

Un estilo arquitectónico, es necesario regresarnos un poco a la arquitectura tradicional (construcción), para ellos, un estilo arquitectónico es un método específico de construcción, caracterizado por las características que lo hacen notable y se distingue por las características que hacen que un edificio u otra estructura sea notable o históricamente identificable.

En el software aplica exactamente igual, pues un estilo arquitectónico determina las características que debe tener un componente que utilice ese estilo, lo cual hace que sea fácilmente reconocible. De la misma forma que podemos determinar a qué periodo de la historia pertenece una construcción al observar sus características físicas, materiales o método de construcción, en el software podemos determinar que estilo de arquitectura sigue un componente al observar sus características.

Algunas definiciones de estilos arquitectónicos:

"Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural; Un vocabulario de componentes y conectores, con restricciones sobre cómo se pueden combinar. "

— M. Shaw and D. Garlan, Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996.

"Un estilo de arquitectura es una asignación de elementos y relaciones entre ellos, junto con un conjunto de reglas y restricciones sobre la forma de usarlos"

—Clements et al., 2003

"Un estilo arquitectónico es una colección con nombre de decisiones de diseño arquitectónico que son aplicables en un contexto de desarrollo dado, restringen decisiones de diseño arquitectónico que son específicas de un sistema particular dentro de ese contexto, y obtienen cualidades beneficiosas en cada uno sistema resultante."

—R. N. Taylor, N. Medvidović and E. M. Dashofy, Software architecture: Foundations, Theory and Practice. Wiley, 2009

LA RELACIÓN ENTRE PATRONES DE DISEÑO, ARQUITECTÓNICOS Y ESTILOS ARQUITECTÓNICOS

Para una gran mayoría de los arquitectos, incluso experimentados, les es complicado diferenciar con exactitud que es un patrón de diseño, un patrón arquitectónico y un estilo arquitectónico, debido principalmente a que, como ya vimos, no existe definiciones concretas de cada uno, además, no existe una línea muy delgada que separa a estos tres conceptos.

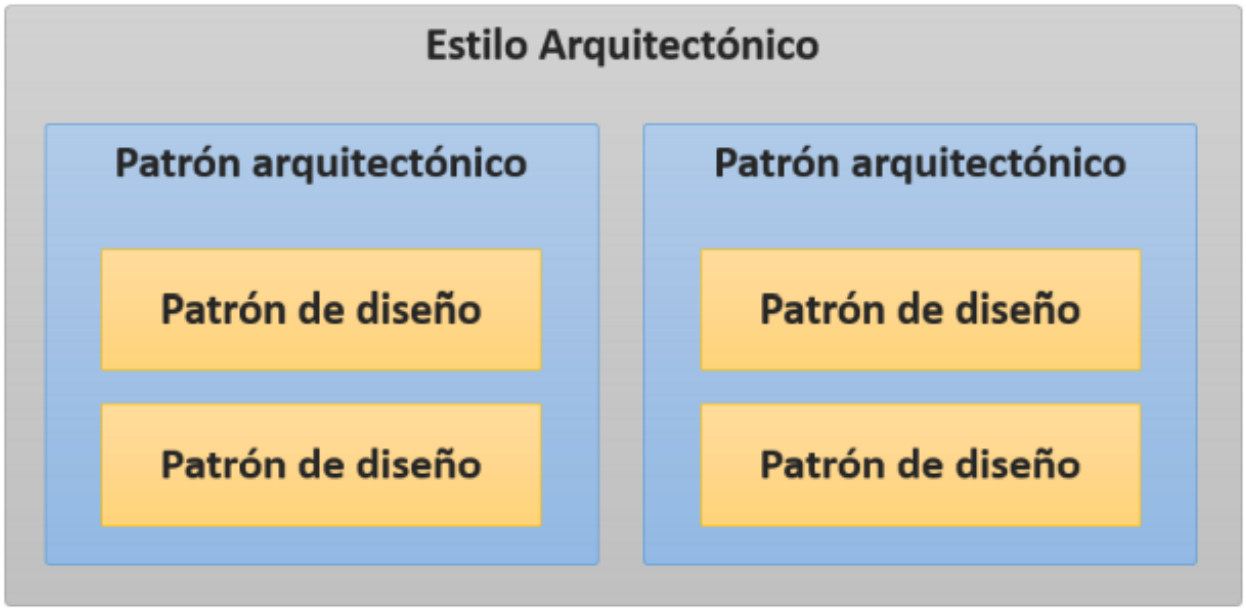


Imagen 5 – Relación entre estilo arquitectónico, patrón arquitectónico y patrón de diseño

ESTILOS ARQUITECTONICOS

Un estilo arquitectónico establece un marco de referencia a partir del cual es posible construir aplicaciones que comparten un conjunto de atributos y características mediante el cual es posible identificarlos y clasificarlos.

ARQUITECTURA MICROSERVICIOS

El estilo arquitectónico de Microservicios se ha convertido en el más popular de los últimos años, y es que no importa la conferencia o eventos de software te dirijas, en todos están hablando de los Microservicios, lo que la hace el estilo arquitectónico más relevante de la actualidad. El estilo de Microservicios consiste en crear pequeños componentes de software que solo hacen una tarea, la hace bien y son totalmente autosuficientes, lo que les permite evolucionar de forma totalmente independiente del resto de componentes.

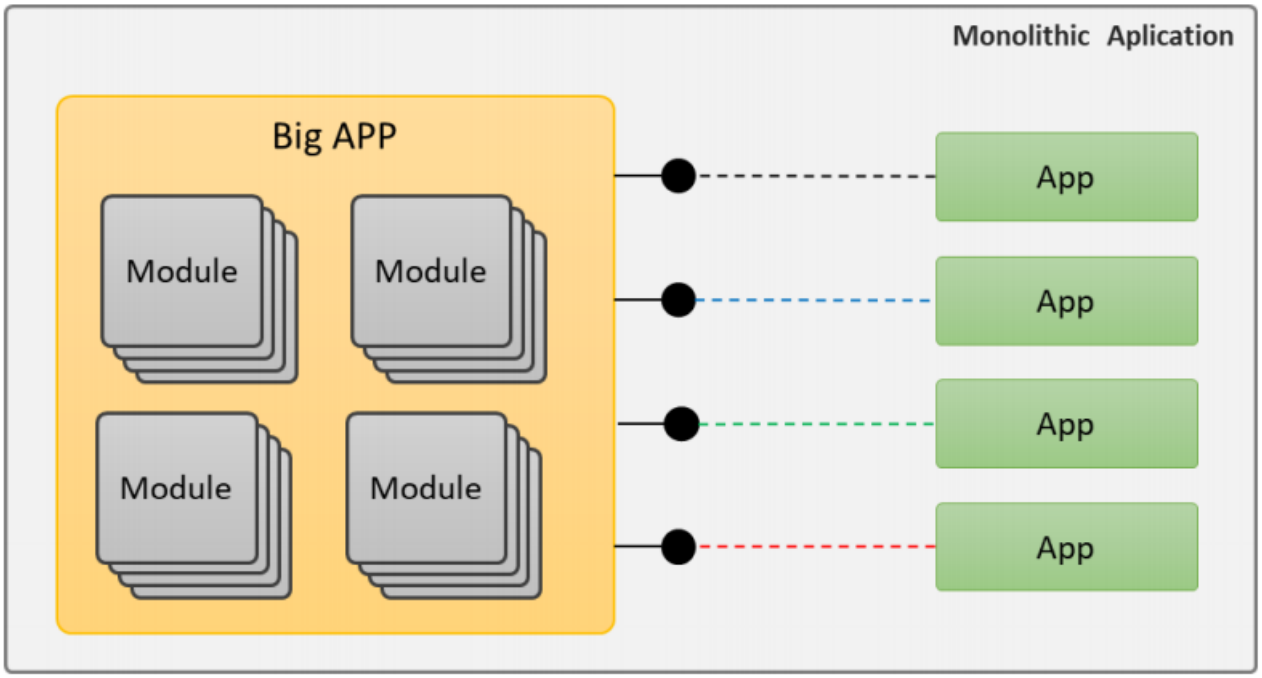
El nombre de “Microservicios” se puede mal interpretar pensando que se trata de un servicio reducido o pequeño, sin embargo, ese es un concepto equivocado, los Microservicios son en realidad pequeñas aplicaciones que brindan un servicio, y observa que he dicho “brinda un servicio” en lugar de “exponen un servicio”, la diferencia radica en que los componentes que implementan un estilo de Microservicios no tiene por qué ser necesariamente un Web Services o REST, y en su lugar, puede ser un procesos que se ejecuta de forma programada, procesos que mueva archivos de una carpeta a otra, componentes que responde a eventos, etc. En este sentido, un Microservicios no tiene porqué exponer necesariamente un servicio, sino más bien, proporciona un servicio para resolver una determinada tarea del negocio. Un Microservicios es un pequeño programa que se

especializa en realizar una pequeña tarea y se enfoca únicamente en eso, por ello, decimos que los Microservicios son Altamente Cohesivos, pues toda las operaciones o funcionalidad que tiene dentro está extremadamente relacionadas para resolver un único problema.

En este sentido, podemos decir que los Microservicios son todo lo contrario a las aplicaciones Monolíticas, pues en una arquitectura de Microservicios se busca desmenuzar una gran aplicación en muchos y pequeños componentes que realizar de forma independiente una pequeña tarea de la problemática general. Una de las grandes ventajas de los Microservicios es que son componentes totalmente encapsulados, lo que quiere decir que la implementación interna no es de interés para los demás componentes, lo que permite que estos evolucionen a la velocidad necesaria, además, cada Microservicios puede ser desarrollado con tecnologías totalmente diferentes, incluso, es normal que utilicen diferentes bases de datos.

Como se estructura un Microservicios

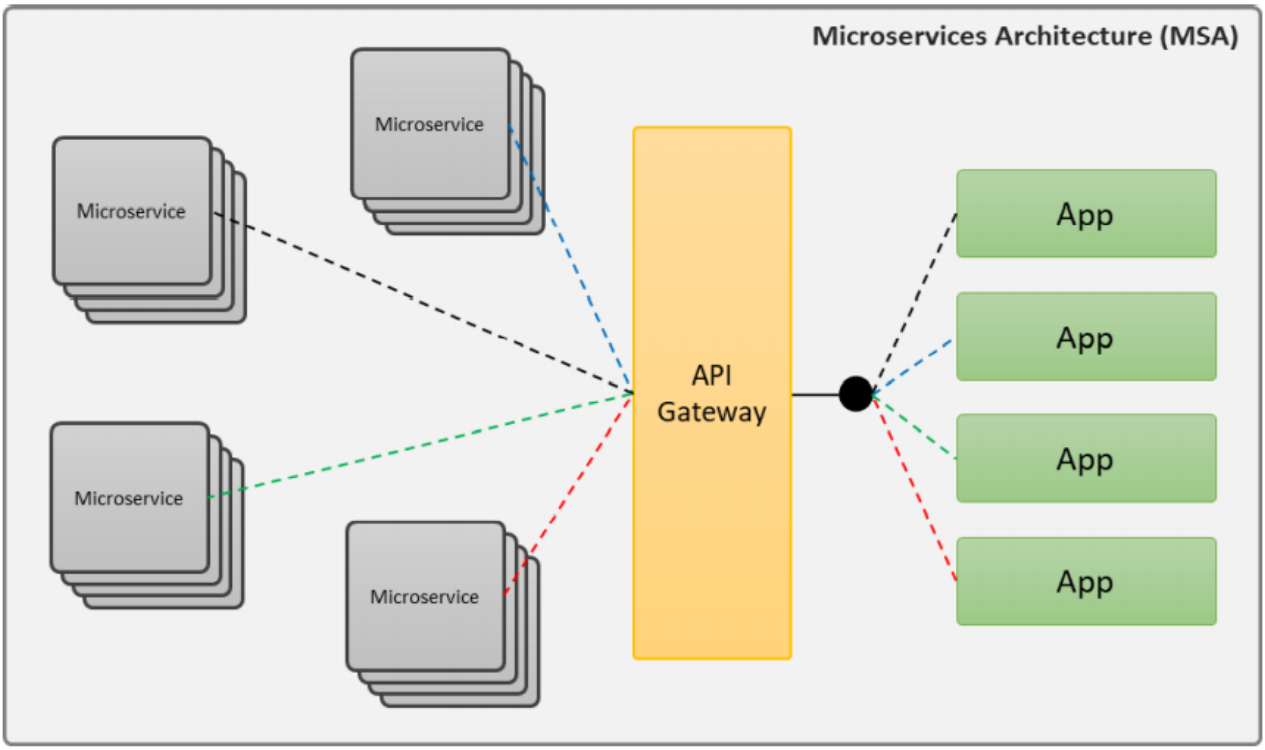
Para comprender los Microservicios es necesario regresar a la arquitectura Monolítica, donde una solo aplicación tiene toda la funcionalidad para realizar una tarea de punta a punta, además, una arquitectura Monolítica puede exponer servicios, tal como lo vemos en la siguiente imagen:



Arquitectura monolítica

Una aplicación Monolítica tiene todos los módulos y funcionalidad necesario dentro de ella, lo que lo hace una aplicación muy grande y pesada, además, en una arquitectura Monolítica, es Todo o Nada, es decir, si la aplicación está arriba, tenemos toda la funcionalidad disponible, pero si está abajo, toda la funcionalidad está inoperable. La arquitectura de Microservicios busca exactamente

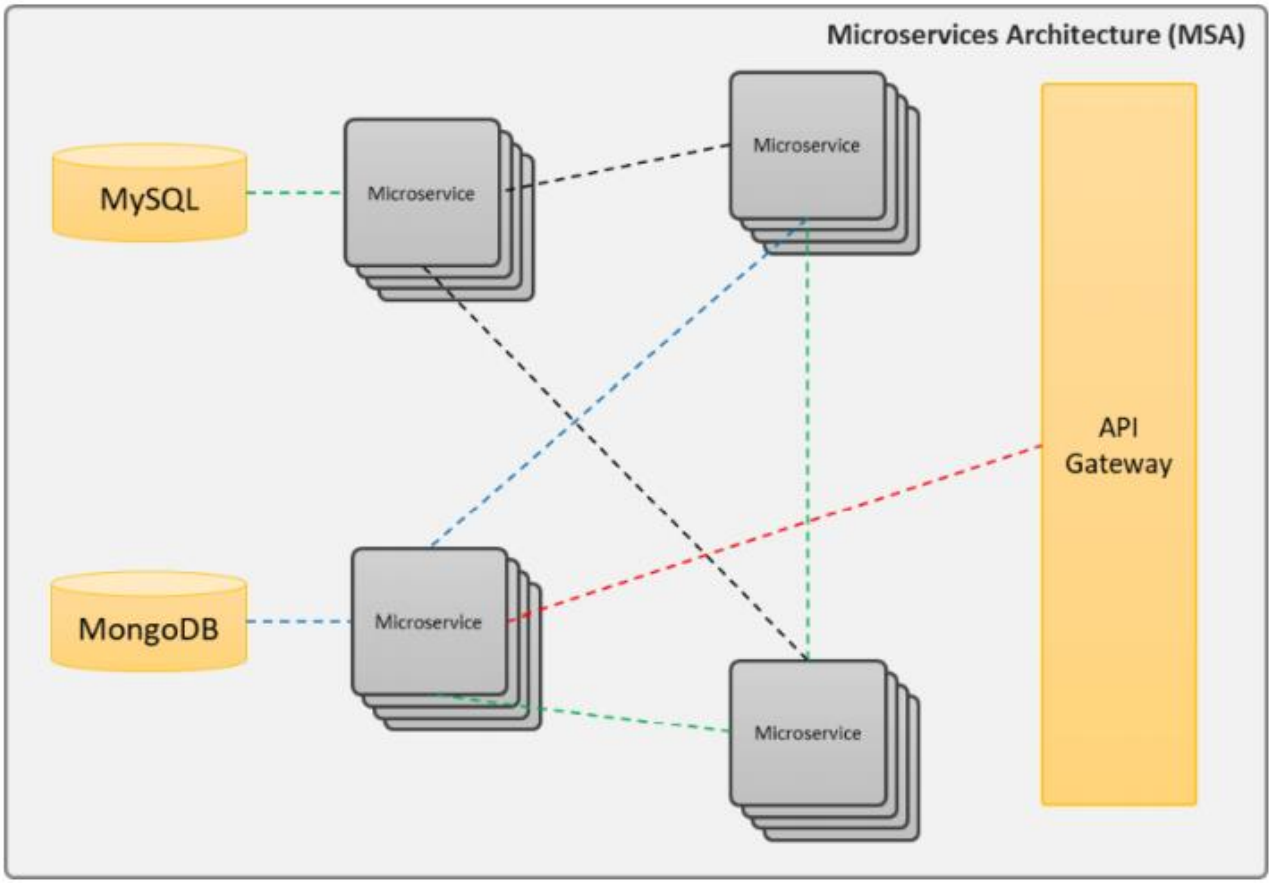
lo contrario, dividiendo toda la funcionalidad en pequeños componentes autosuficientes e independientes del resto de componentes, tal y como lo puedes ver en la imagen anterior.



Arquitectura microservicios

En la arquitectura de Microservicios es común ver algo llamado API Gateway, el cual es un componente que se posiciona de cara a los microservicios para servir como puerta de entrada a los Microservicios, controlando el acceso y la funcionalidad que deberá ser expuesta a una red pública.

Debido a que los Microservicios solo realizan una tarea, es imposible que por sí solos no puedan realizar una tarea de negocio completa, por lo que es común que los Microservicios se comuniquen con otros Microservicios para delegar ciertas tareas, de esta forma, podemos ver que todos los Microservicios crean una red de comunicación entre ellos mismos, incluso, podemos apreciar que diferentes Microservicios funcionan con diferentes bases de datos.



Comunicación entre microservicios

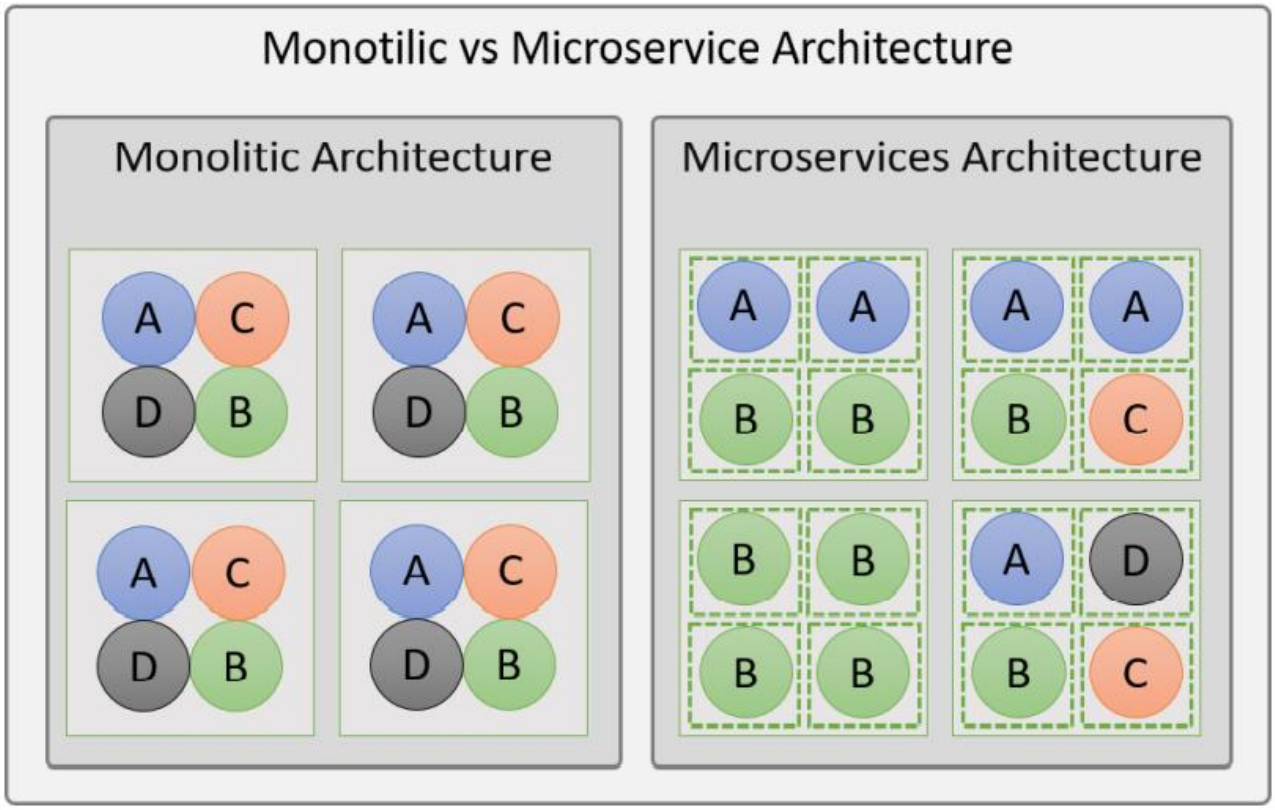
Algo a tomar en cuenta es que los Microservicios trabajan en una arquitectura distribuida, lo que significa todos los Microservicios son desplegados de forma independiente y están desacoplados entre sí. Además, deben ser accedidos por medio de protocolos de acceso remoto, como colas de mensajes, SOAP, REST, RPC, etc. Esto permite que cada Microservicio funcione o pueda ser desplegado independientemente de si los demás están funcionando.

Escalabilidad Monolítica vs escalabilidad de Microservicios

Una de las grandes problemáticas cuando trabajamos con arquitecturas Monolíticas es la escalabilidad, ya que son aplicaciones muy grandes que tiene mucha funcionalidad, la cual consume muchos recursos, se use o no, estos recursos tienen un costo financiero pues hay que tener el hardware suficiente para levantar todo, ya que recordemos que una arquitectura Monolítica es TODO o NADA, pero hay algo más, el problema se agudiza cuando necesitamos escalar la aplicación Monolítica, lo que requiere levantar un nuevo servidor con una réplica de la aplicación completa.

En una arquitectura Monolítica no podemos decidir qué funcionalidad desplegar y que no, pues la aplicación viene como un TODO, lo que nos obliga a escalar de forma Monolítica, es decir,

escalamos todos los componentes de la aplicación, incluso si es funcionalidad que casi no se utiliza o que no se utiliza para nada.



Escalamiento monolotico vs microservicios

Del lado izquierdo de la imagen anterior podemos ver como una aplicación Monolítica es escalada en 4 nodos, lo que implica que la funcionalidad A, B, C y D sean desplegadas por igual en los 4 nodos. Obviamente esto es un problema si hay funcionalidad que no se requiere, pero en un Monolítico no tenemos otra opción.

Del lado derecho podemos ver cómo es desplegado los Microservicios. Podemos ver que tenemos exactamente las mismas funcionalidades A, B, C y D, con la diferencia de que esta vez, cada funcionalidad es un Microservicio, lo que permite que cada funcionalidad sea desplegada de forma independiente, lo que permite que decidamos que componentes necesitan más instancias y cuales menos. Por ejemplo, pude que la funcionalidad B sea la más crítica, que es donde realizamos las ventas, entonces creamos 8 instancias de ese Microservicio, por otro lado, tenemos el componente que envía Mail, el cual no tiene tanda demanda, así que solo dejamos una instancia.

Como podemos ver, los Microservicios permite que controlemos de una forma más fina el nivel de escalamiento de los componentes, en lugar de obligarnos a escalar toda la aplicación de forma Monolítica.

Características de un Microservicio

En esta sección analizaremos las características que distinguen al estilo de Microservicios del resto. Las características no son ventajas o desventajas, si no que más bien, nos sirven para identificar cuando una aplicación sigue un determinado estilo arquitectónico.

Las características se pueden convertir en ventajas o desventajas solo cuando analizamos la problemática que vamos a resolver, mientras tanto, son solo características:

- Son componentes altamente cohesivos que se enfocan en realizar una tarea muy específica.
- Son componentes autosuficientes y no requieren de ninguna otra dependencia para funcionar.
- Son componentes distribuidos que se despliegan de forma totalmente independiente de otras aplicaciones o Microservicios.
- Utilizan estándares abiertos ligeros para comunicarse entre sí.
- Es normal que existan múltiples instancias del Microservicios funcionando al mismo tiempo para aumentar la disponibilidad.
- Los Microservicios son capaces de funcionar en hardware limitado, pues no requieren de muchos recursos para funcionar.
- Es común que una arquitectura completa requiere de varios Microservicios para ofrecer una funcionalidad de negocio completa.
- Es común ver que los Microservicios están desarrollados en tecnologías diferentes, incluido el uso de bases de datos distintas.

Ventajas

- Alta escalabilidad: Los Microservicios es un estilo arquitectónico diseñado para ser escalable, pues permite montar numerosas instancias del mismo componente y balancear la carga entre todas las instancias.
- Agilidad: Debido a que cada Microservicios es un proyecto independiente, permite que el componente tenga ciclo de desarrollo diferente del resto, lo que permite que se puedan hacer despliegues rápidos a producción sin afectar al resto de componentes.
- Facilidad de despliegue: Las aplicaciones desarrolladas como Microservicios encapsulan todo su entorno de ejecución, lo que les permite ser desplegadas sin necesidad de dependencias externas o requerimientos específicos de Hardware.
- Testabilidad: Los Microservicios son especialmente fáciles de probar, pues su funcionalidad es tan reducida que no requiere mucho esfuerzo, además, su naturaleza de exponer o brindar servicios hace que sea más fácil de crear casos específicos para probar esos servicios.
- Fácil de desarrollar: Debido a que los Microservicios tiene un alcance muy corto, es fácil para un programador comprender el alcance del componente, además, cada Microservicios puede ser desarrollado por una sola persona o un equipo de trabajo muy reducido.
- Reusabilidad: La reusabilidad es la médula espinal de la arquitectura de Microservicios, pues se basa en la creación de pequeños componentes que realice una única tarea, lo que hace que sea muy fácil de reutilizar por otras aplicaciones o Microservicios.

- Interoperabilidad: Debido a que los Microservicios utilizan estándares abiertos y ligeros para comunicarse, hace que cualquier aplicación o componente pueda comunicarse con ellos, sin importar en que tecnología está desarrollado.

Desventajas

- Performance: La naturaleza distribuida de los Microservicios agrega una latencia significativa que puede ser un impedimento para aplicaciones donde el performance es lo más importante, por otra parte, la comunicación por la red puede llegar a ser incluso más tardado que el proceso en sí.
- Múltiples puntos de falla: La arquitectura distribuida de los Microservicios hace que los puntos de falla de una aplicación se multipliquen, pues cada comunicación entre Microservicios tiene una posibilidad de fallar, lo cual hay que gestionar adecuadamente.
- Trazabilidad: La naturaleza distribuida de los Microservicios complica recuperar y realizar una traza completa de la ejecución de un proceso, pues cada Microservicio arroja de forma separa su traza o logs que luego deben de ser recopilados y unificados para tener una traza completa.
- Madurez del equipo de desarrollo: Una arquitectura de Microservicios debe ser implementada por un equipo maduro de desarrollo y con un tamaño adecuado, pues los Microservicios agregan muchos componentes que deben ser administrados, lo que puede ser muy complicado para equipo poco maduros.

Cuando debo de utilizar un estilo de Microservicios

Debido a que los Microservicios se han puesto muy moda en estos años, es normal ver que en todas partes nos incentivan a utilizarlos, sin embargo, su gran popularidad ha llegado a segar a muchos, pues alientan y justifican la implementación de microservicios para casi cualquier cosa y ante cualquier condición, muy parecido al anti-patrón Golden Hammer.

Los Microservicios son sin duda una excelente arquitectura, pero debemos tener en claro que no sirve para resolver cualquier problema, sobre todos en los que el performance es el objetivo principal.

Un error común es pensar que una aplicación debe de nacer desde el inicio utilizando Microservicios, lo cual es un error que lleva por lo general al fracaso de la aplicación, tal como lo comenta Martin Fowler en uno de sus artículos. Lo que él recomienda es que una aplicación debe nacer como un Monolítico hasta que llegue el momento que el Monolítico sea demasiado complicado de administrar y todos los procesos de negocio están muy maduros, en ese momento es cuando Martin Fowler recomienda que debemos de empezar a partir muestro Monolítico en Microservicios.

Puede resultar estúpido crear un Monolítico para luego partirlo en Microservicios, sin embargo, tiene mucho sentido. Cuando una aplicación nace, no se tiene bien definido el alcance y los procesos de negocio no son maduros, por lo que comenzar a fraccionar la lógica en Microservicios desde el inicio puede llevar a un verdadero caos, pues no se sabe a ciencia cierta qué dominio debe atacar cada

Microservicio, además, Empresas como Netflix, que es una de las que más promueve el uso de Microservicios también concuerda con esto.

Dicho lo anterior y analizar cuando NO debemos utilizar Microservicios, pasemos a analizar los escenarios donde es factible utilizar los Microservicios. De forma general, los Microservicios se prestan más para ser utilizadas en aplicaciones que:

- Aplicaciones pensadas para tener un gran escalamiento
- Aplicaciones grandes que se complica ser desarrolladas por un solo equipo de trabajo, lo que hace complicado dividir las tareas si entrar en constante conflicto.
- Donde se busca agilidad en el desarrollo y que cada componente pueda ser desplegado de forma independiente.

Hoy en día es común escuchar que las empresas más grandes del mundo utilizan los Microservicios como base para crear sus aplicaciones de uso masivo, como es el caso claro de Netflix, Google, Amazon, Apple, etc.

EJERCICIO

1. Opción según indicación.
2. Según el artículo https://www.xataka.com/servicios/netflix-no-es-solo-video-bajo-demanda-es-uno-de-los-grandes-monstruos-tecnologicos-de-la-historia Realice un resúmen del articulo y cuál sería la opinión del grupo con respecto a lo realizado por Netflix?
3. ¿Cuál es la razón por la que Netflix tomo la decisión de moverse de un datacenter monolitico a una nube basada en arquitectura microservicios?
4. Realice un glosario de términos con las palabras desconocidas para el grupo.

CASO DE ESTUDIO

Según el caso de estudio, como crearía el grupo el diagrama de componentes y el diagrama de despliegue utilizando microservicios y que permita gestionar el proceso de negocio. Adicional cree la estructura del proyecto en github.

Suponga las indicaciones dadas por el instructor.

4. ACTIVIDADES DE EVALUACIÓN

Evidencia de Conocimiento: Realizar una infografía digital sobre la arquitectura de microservicios realice una presentación sobre las 4 preguntas y el caso de estudio.

Evidencia de Desempeño: Realiza la sustentación de cada evidencia de conocimiento.

Evidencia de Producto: Entrega de los documentos requeridos en las evidencias de conocimiento.

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento:	**Reconoce las principales características de la arquitectura por capas**	Redacción
Evidencias de Desempeño:	**Interpreta el uso de una aplicación arquitectura por capas en un entorno empresarial**	Presentación
Evidencias de Producto	**Realiza el entregable con la documentación completa**	Entrega de la guía con todas las evidencias requeridas.

1. **GLOSARIO DE TÉRMINOS**

   De acuerdo a la práctica realizar su propio glosario de términos.

6. REFERENTES BIBLIOGRÁFICOS

Construya o cite documentos de apoyo para el desarrollo de la guía, según lo establecido en la guía de desarrollo curricular

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	**Néstor Rodríguez**	**Instructor**	**Teleinformática**	**NOVIEMBRE-2020**

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					