

PROCESO DIRECCIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUÍA 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)

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

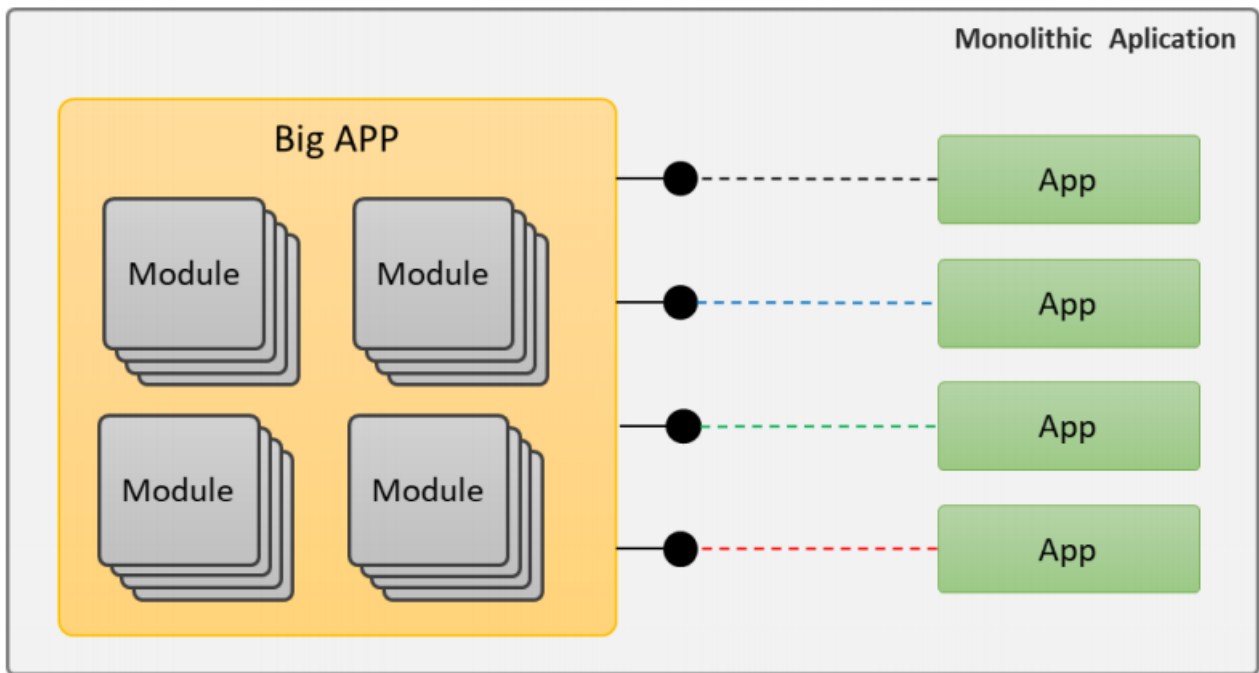
hace bien y son totalmente autosuficientes, lo que les permite evolucionar de forma totalmente independiente del resto de componentes.

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.

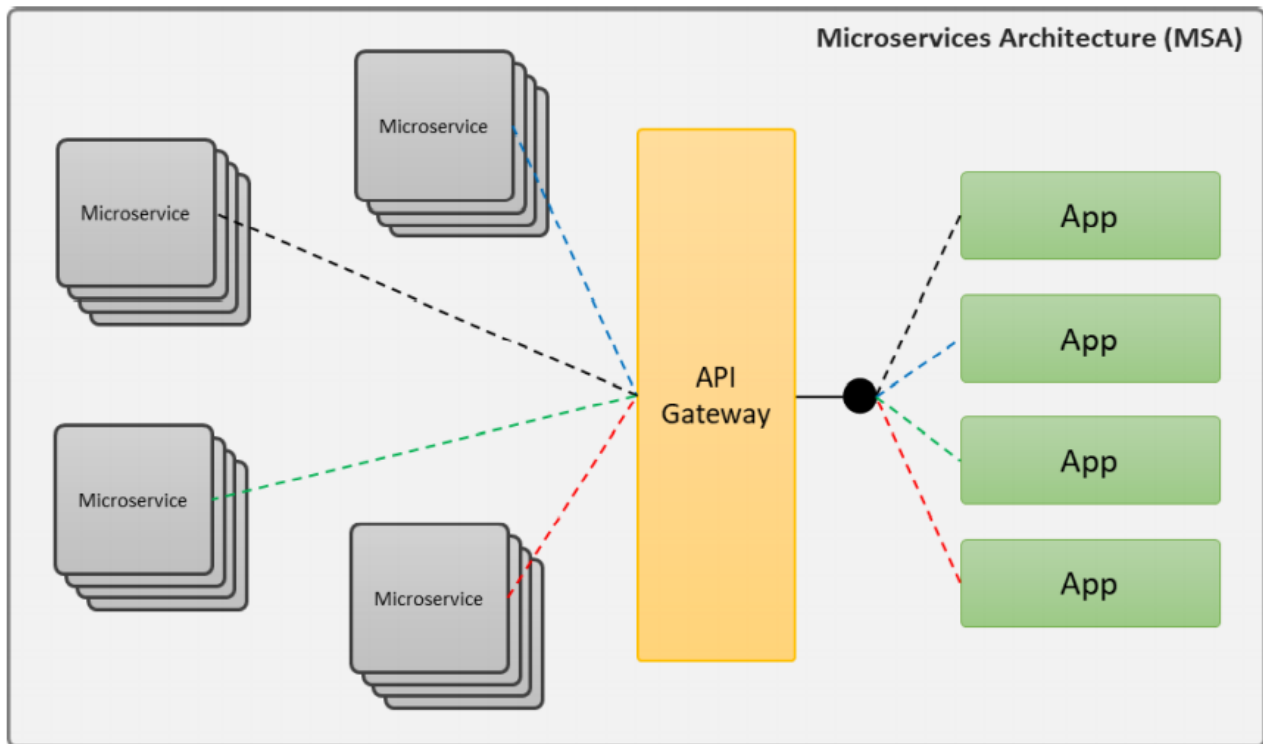
Como se estructura un Microservicios

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

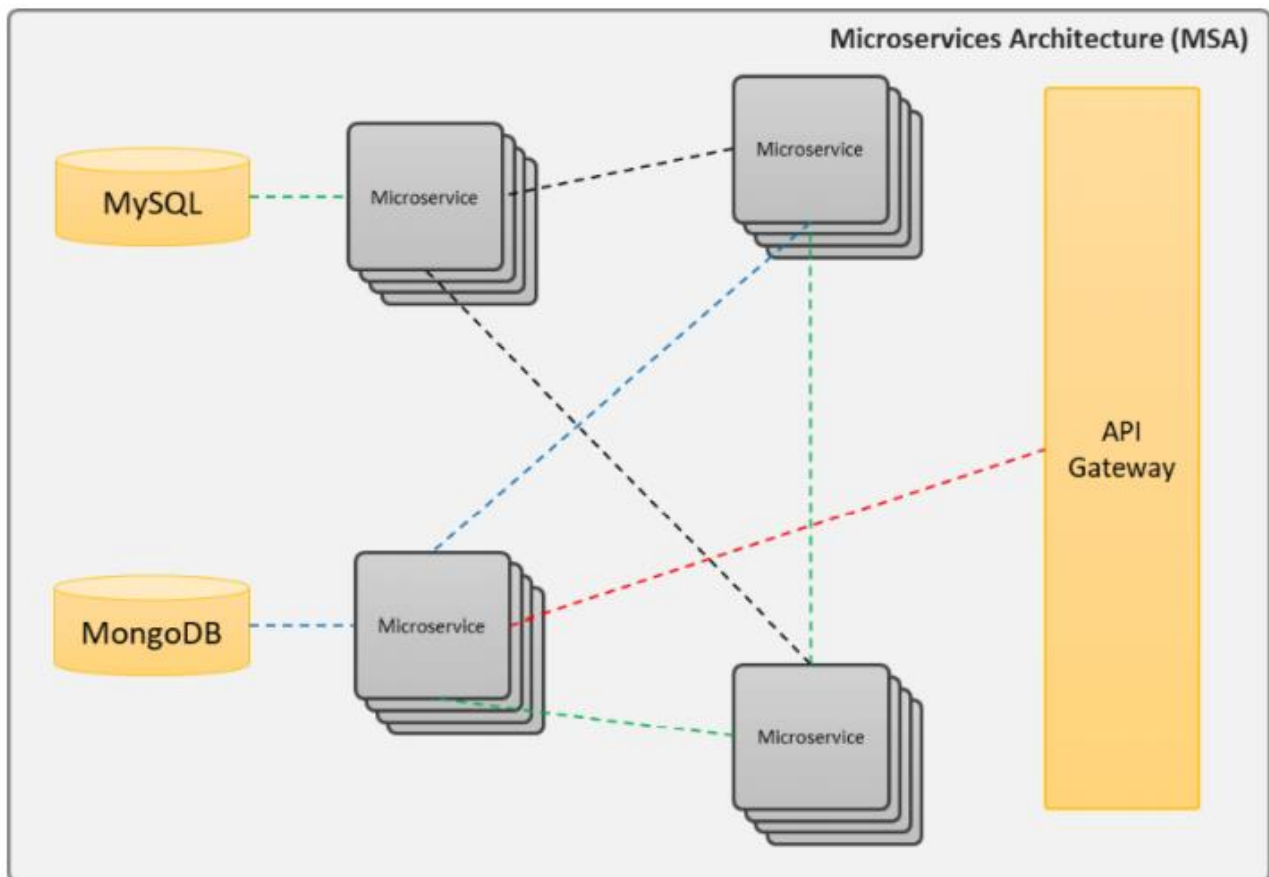
Una aplicación Monolítica tiene todos los módulos y funcionalidad necesario dentro de ella, lo que lo hace una aplicación muy grande y pesada, además, en una arquitectura Monolítica, es Todo o Nada, es decir, si la aplicación está arriba, tenemos toda la funcionalidad disponible, pero si está abajo,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

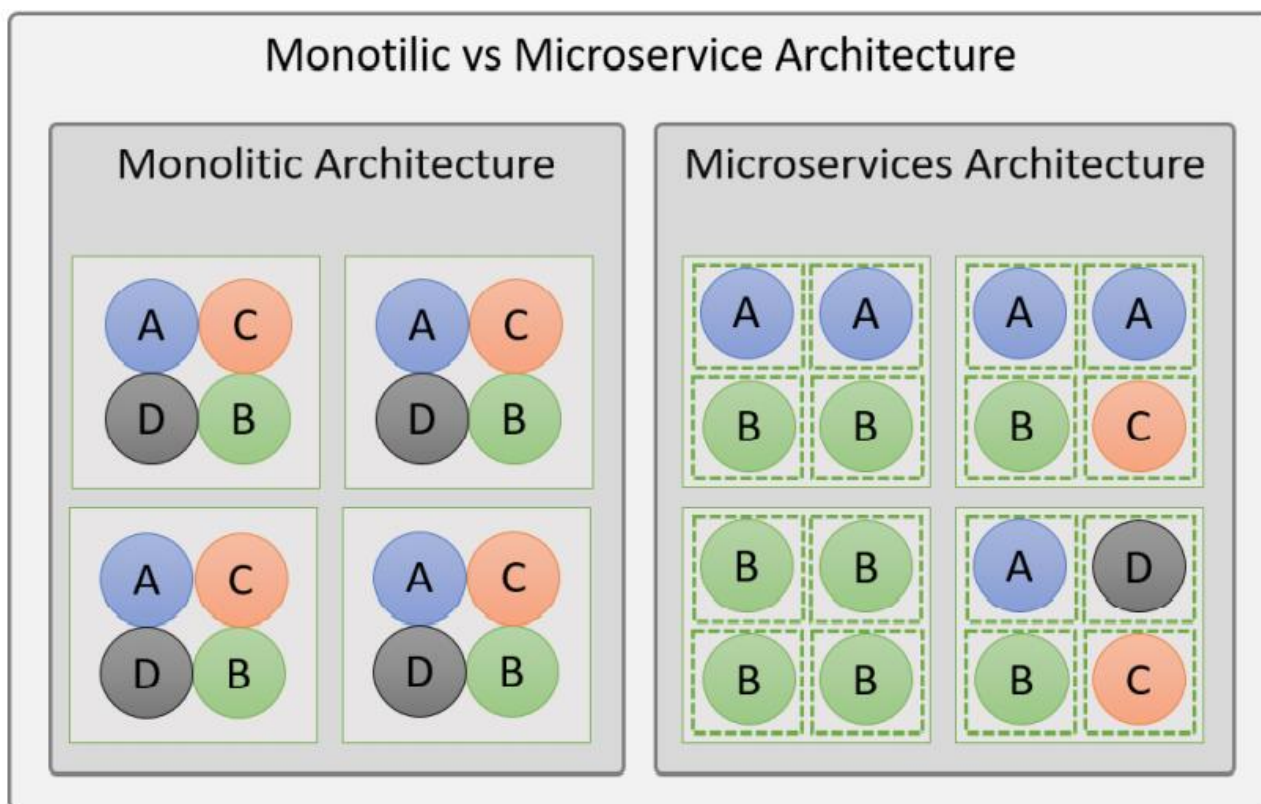
Algo a tomar en cuenta es que los Microservicios trabajan en una arquitectura distribuida, lo que significa todos los Microservicios son desplegados de forma independiente y están desacoplados entre sí. Además, deben ser accedidos por medio de protocolos de acceso remoto, como colas de mensajes, SOAP, 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 tanta 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 nuestro 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:

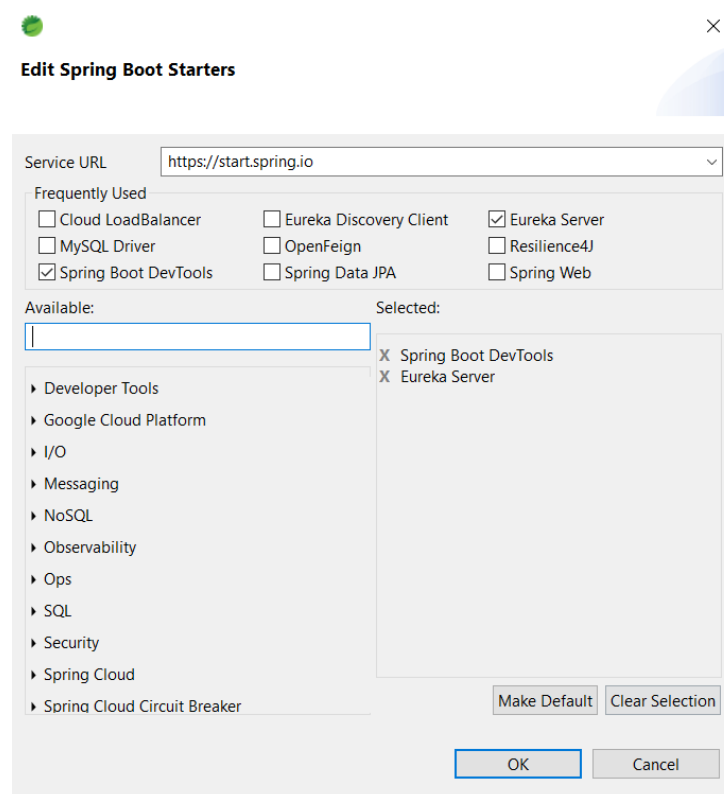
- Aplicaciones pensadas para tener un gran escalamiento
- Aplicaciones grandes que se complica ser desarrolladas por un solo equipo de trabajo, lo que hace complicado dividir las tareas si entrar en constante conflicto.
- Donde se busca agilidad en el desarrollo y que cada componente pueda ser desplegado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

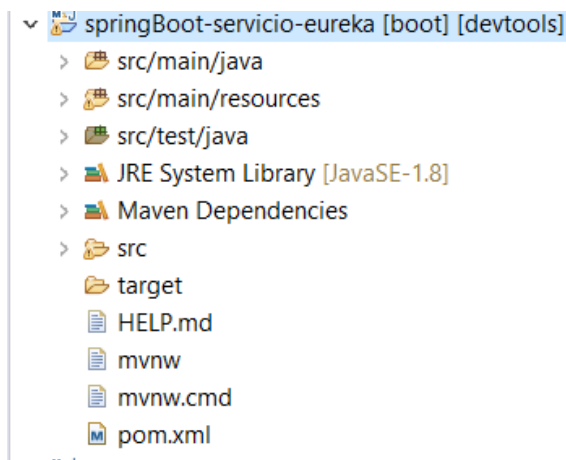
EJERCICIO

Configuración de eruka server y eureka client en el ejercicio.

Cree un proyecto nuevo para colocar el service Discovery en el servidor de eureka con las siguientes características:



Este será el servidor eureka y tendrá la siguiente estructura:



Ingresa al proyecto del servidor de eureka, ubique el application properties y configure el servidor:

```

1 spring.application.name=servicio-eureka-server
2
3 server.port=8761
4
5 eureka.client.register-with-eureka=false
6 eureka.client.fetch-registry=false
7 eureka.renewalPercentThreshold=0.85
8

```

Ubique el archivo de SpringBootServicioEurekaApplication del proyecto y habilite eureka:

```

1 package com.app.items;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @EnableEurekaServer
8 @SpringBootApplication
9 public class SpringBootServicioEurekaApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(SpringBootServicioEurekaApplication.class, args);
13     }
14
15 }
16

```

Una vez realizados los pasos anteriores, corra el servicio de eureka y revise que quede funcionando:



The screenshot shows the Spring Eureka web interface. At the top, there's a navigation bar with 'HOME' and 'LAST 1000 SINCE STARTUP'. Below this, the 'System Status' section contains two tables. The first table lists 'Environment' and 'Data center' as 'N/A'. The second table lists 'Current time' as '2021-02-13T02:01:26 -0500', 'Uptime' as '00:00', 'Lease expiration enabled' as 'false', 'Renews threshold' as '1', and 'Renews (last min)' as '0'. Below this is the 'DS Replicas' section, which states 'Instances currently registered with Eureka' and shows a table with columns 'Application', 'AMIs', 'Availability Zones', and 'Status'. A message 'No instances available' is displayed. The 'General Info' section shows a table with columns 'Name' and 'Value', listing various system metrics like 'total-avail-memory', 'num-of-cpus', 'current-memory-usage', 'server-upptime', 'registered-replicas', and 'unavailable-replicas'.

Ya quedo configurado el servidor de eureka, ahora procedemos a configurar los dos clientes que son los dos servicios web creados previamente.

En el proyecto servicio item agregue la dependencia de eureka client en el pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>
    spring-cloud-starter-netflix-eureka-client
  </artifactId>
</dependency>
```

Se configura el application properties de ese proyecto para que quede con la ubicación del servidor de eureka (Ya no vamos a asignar los puertos físicos, eso lo realiza eureka)

The screenshot shows an IDE with several tabs open. The active tab is 'application...' and it displays the following configuration in 'application.properties':

```
1 spring.application.name=servicio-items
2
3 #Port
4 server.port=8083
5
6 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
7
```

Ubique el archivo de SpringBootServicioItemApplication del proyecto y habilite el cliente eureka:

```

1 package com.app.items;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6 import org.springframework.cloud.openfeign.EnableFeignClients;
7
8 @EnableEurekaClient
9 @EnableFeignClients
10 @SpringBootApplication
11 public class SpringBootServicioItemApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(SpringBootServicioItemApplication.class, args);
15     }
16
17 }
18

```

En el proyecto servicio productos agregue la dependencia de eureka client en el pom.xml

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>
        spring-cloud-starter-netflix-eureka-client
    </artifactId>
</dependency>

```

Se configura el application properties de ese proyecto para que quede con la ubicación del servidor de eureka (Ya no vamos a asignar los puertos físicos, eso lo realiza eureka), el server.port lo va a tomar del id de la instancia que se genera de forma aleatoria.



```
SpringBootSe... application... SpringBootSe... springBoot-... springBoot-... ItemRestCon... AppConfig.java *applicatio... »4
1 spring.application.name=servicio-productos
2
3 #Data source
4 #Indica el driver/lib para conectar java a mysql
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6
7 #Url donde esta el servicio de tu mysql y el nombre de la base de datos
8 spring.datasource.url=jdbc:mysql://localhost:3306/proyecto_microservicios?useUnicode=true&useJDBCCompliantTimezon
9
10 #Usuario y contraseña para tu base de datos descrita en la linea anterior
11 spring.datasource.username=root
12 spring.datasource.password=1234
13
14 #Port
15 server.port=${PORT:0}
16
17 #Asigan un puerto aleatorio en el servidor de eureka
18 eureka.instance.instance-id=${spring.application.name}:${spring.application.instance_id:${random.value}}
19
20 spring.jpa.hibernate.ddl-auto=update
21
22 #[opcional]Imprime en tu consola las instrucciones hechas en tu base de datos.
23 spring.jpa.show-sql = true
24
25 #Eureka client
26 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
27
```

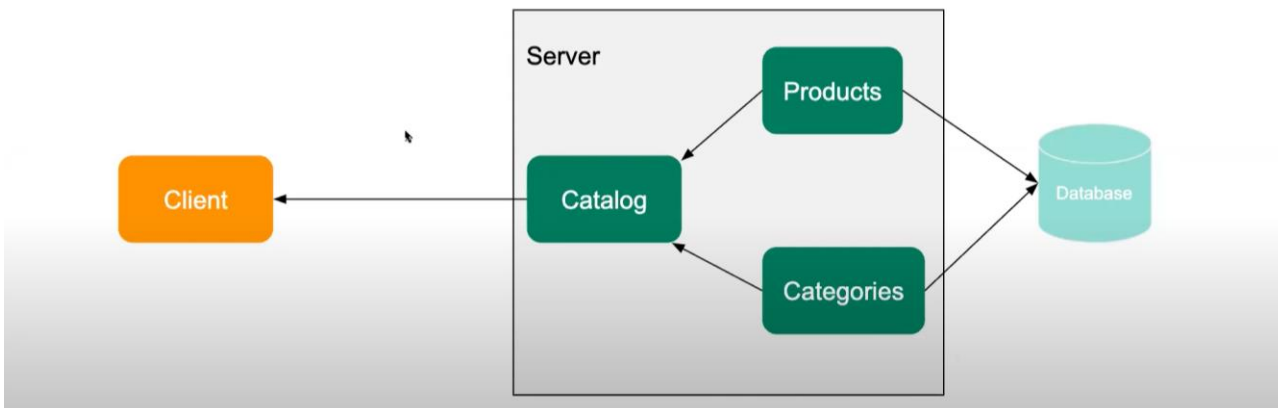
Ubique el archivo de SpringBootServicioProductosApplication del proyecto y habilite el cliente eureka:

```
1 package com.app.productos;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @EnableEurekaClient
8 @SpringBootApplication
9 public class SpringBootServicioProductosApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(SpringBootServicioProductosApplication.class, args);
13     }
14
15 }
```

Suba los dos servicios y pruébelo. En el servicio productos genere TRES instancias, realice el consumo del servicio de esas instancias y el de item.

Caso de estudio:

Tenemos una aplicación monolítica y necesitamos escalarla a microservicios en java.



Proponga el diseño según las especificaciones dadas y realice el código en java o lenguaje que considere en el que registre los servicios y el despliegue lo realice de forma automática el numero de instancias requerido.

Evidencia de Conocimiento: Describa como funciona el patrón de service Discovery y el de service register en el ejercicio realizado.

Evidencia de Desempeño: Realiza la **presentación** del código de eureka (Esta guía) y la **presentación** del caso de estudio – Describiendo que realiza cada uno

Evidencia de Producto: Entrega de los documentos requeridos en las evidencias de conocimiento y el repositorio de github con cada proyecto presentado.

| Evidencias de Aprendizaje | Criterios de Evaluación | Técnicas e Instrumentos de Evaluación |
|-----------------------------|---|---|
| Evidencias de Conocimiento: | Reconoce los principales patrones de la arquitectura por microservicios | Redacción |
| Evidencias de Desempeño: | Interpreta el uso de una arquitectura microservicios básica 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 | FEBRERO-2020 |

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

| | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|------------|--------|-------|-------------|-------|------------------|
| Autor (es) | | | | | |