

INICIO

Desarrollo de Microservicios con Spring Cloud Netflix OSS

ISC. Ivan Venor García Baños





+

NETFLIX
OSS

Agenda

1. Presentación
2. Objetivos
3. Contenido
4. Despedida



Microservices



3. Contenido

- i. Arquitectura de sistemas monolíticos
- ii. Introducción a la Arquitectura Orientada a Servicios
- iii. Fundamentos Spring Boot 2.x
- iv. Arquitectura de Microservicios
- v. Implementación de Microservicios con Spring Boot
- vi. Microservicios con Spring Cloud y Spring Netflix OSS



3. Contenido

- i. Arquitectura de sistemas monolíticos
- ii. Introducción a la Arquitectura Orientada a Servicios
- iii. Fundamentos Spring Boot 2.x
- iv. Arquitectura de Microservicios
- v. Implementación de Microservicios con Spring Boot
- vi. Microservicios con Spring Cloud y Spring Netflix OSS



+

NETFLIX
OSS

i. Arquitectura de sistemas monolíticos

Microservices



+

NETFLIX
OSS

i. Arquitectura de sistemas monolíticos

- i.i ¿Qué son los sistemas monolíticos?
- i.ii Escalabilidad de sistemas monolíticos
- i.iii Protocolos de integración



Microservices



+

NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos?



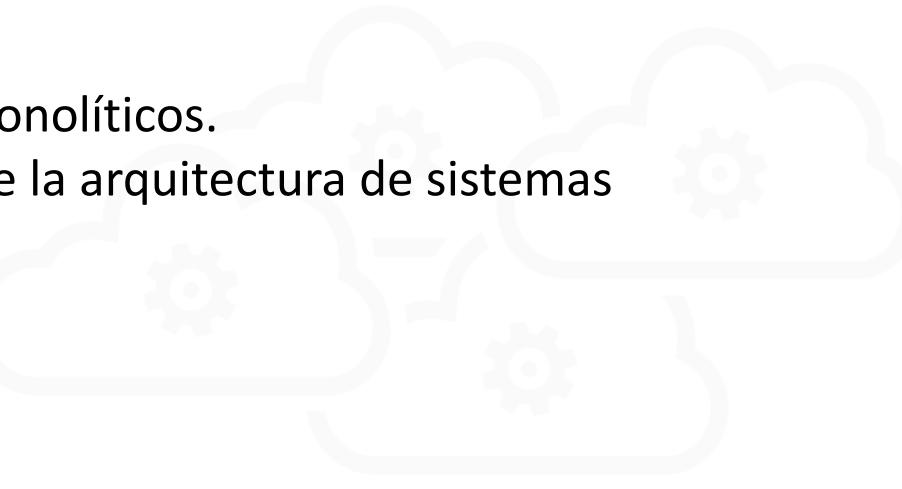
+

NETFLIX
OSS

Objetivos de la lección

i.i ¿Qué son los sistemas monolíticos?

- Comprender que son los sistemas monolíticos.
- Analizar las ventajas y desventajas de la arquitectura de sistemas monolíticos.



Microservices

i.i ¿Qué son los sistemas monolíticos? (a)

- El término monolito o monolítico (monolith) se refiere a un estilo de arquitectura o a un patrón (o anti-patrón) de desarrollo de software.
- Diferentes estilos de arquitectura o patrones de desarrollo de software se clasifican en diferentes tipos de vista o “viewtypes” (un “viewtype” es un conjunto o categoría de cómo se visualiza una arquitectura de software).

i.i ¿Qué son los sistemas monolíticos? (b)

- El estilo de arquitectura monolítica se visualiza principalmente en tres tipos de vista (viewtypes) los cuales son:
 - Módulos (modules)
 - Asignación (allocation) y,
 - Ejecución (runtime)





+

NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos? (c)

- Módulos monolíticos.
- Se refiere a que todo el código de un sistema se encuentra en un único código base o código fuente que es compilado y produce un único artefacto o pieza de software.
- El código fuente puede estar bien estructurado, mediante clases y paquetes, y puede estar escrito implementando las mejores prácticas de desarrollo sin embargo, no se encuentra dividido en distintos módulos para su mantenimiento, compilación, despliegue y ejecución.



+

NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos? (d)

- Módulos monolíticos.
- En el estricto sentido, el diseño de un módulo no-monolítico mantiene el código fuente del sistema dividido o separado en múltiples módulos o librerías los cuales pueden ser compilados, mantenidos, desplegados y ejecutados de manera separada.
- A su vez, el diseño de un módulo no-monolítico puede estar almacenado en diferentes códigos base y diferentes repositorios, lo que permite que puedan ser referenciados o modificados de forma independiente cuando sea necesario.



+

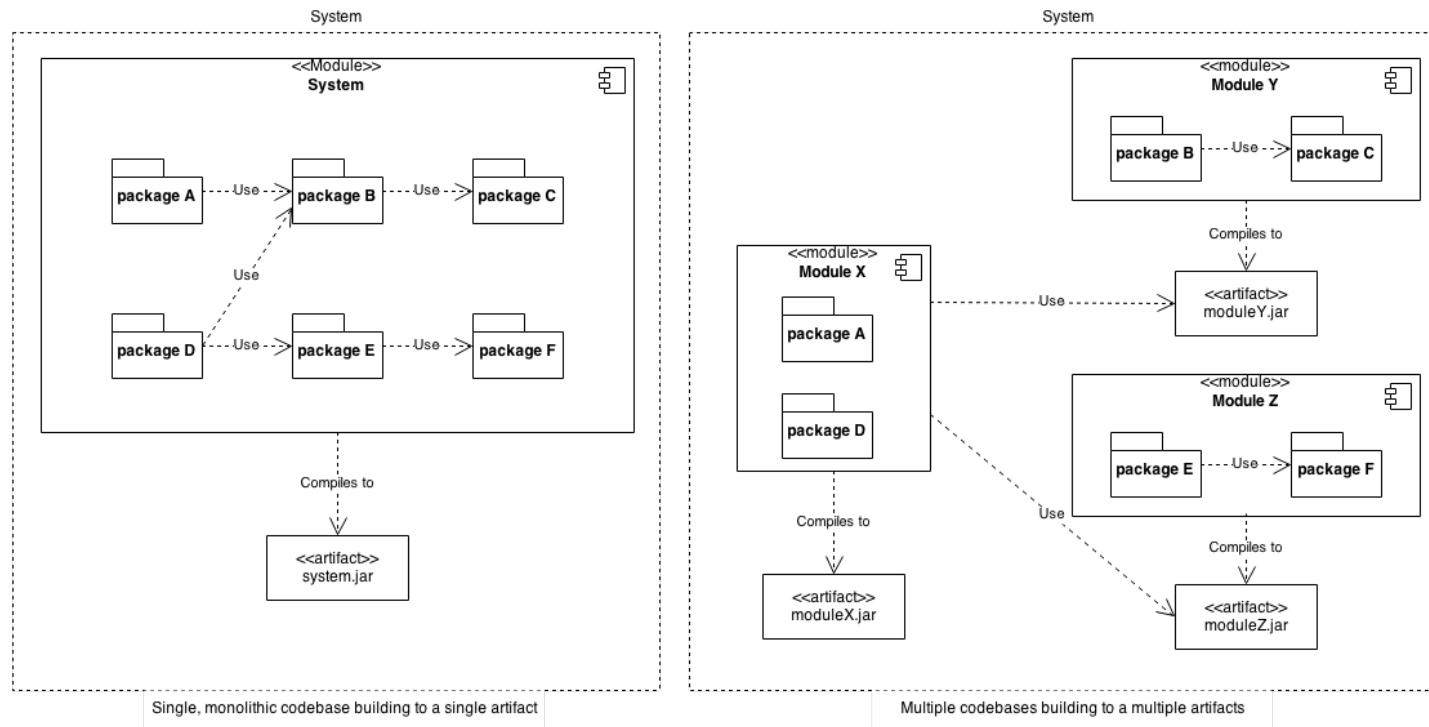
NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos? (e)

- Módulos monolíticos.
- Existen ventajas y desventajas referentes a módulos monolíticos y no-monolíticos sin embargo, únicamente estamos referenciando cómo el código fuente, de ambas estrategias, es utilizado.

i.i ¿Qué son los sistemas monolíticos? (f)

- Módulos monolíticos vs no-monolíticos.





i.i ¿Qué son los sistemas monolíticos? (g)

- Asignación (allocation) monolítica.
- Se refiere a que todo el código fuente del módulo o sistema monolítico ha sido empaquetado y desplegado en el mismo momento, al mismo tiempo.
- La asignación monolítica, se refiere a que, una vez que el código fuente ha sido compilado y esta listo para ser liberado, existe una única versión del empaquetado el cuál contiene todos los componentes necesarios para su ejecución.



+

i.i ¿Qué son los sistemas monolíticos? (h)

- Asignación (allocation) monolítica.
- Todos los componentes en ejecución tienen la misma versión del software que se ejecuta en un momento determinado.
- La asignación (allocation) monolítica es independiente de si la estructura (a nivel código) del módulo es un monolito o no, debido a una de las dos posibles asignaciones (allocations) siguientes:
 - Despliegue de un único compilado, de un único código fuente o,
 - Despliegue simultáneo de múltiples compilados, de distintos códigos fuente.



+

i.i ¿Qué son los sistemas monolíticos? (i)

- Asignación (allocation) monolítica.
- Despliegue de un único compilado, de un único código fuente.
 - Es posible que se haya compilado todo el código base a la vez antes de su despliegue.



i.i ¿Qué son los sistemas monolíticos? (j)

- Asignación (allocation) monolítica.
- Despliegue simultaneo de múltiples compilados de distintos códigos fuente.
 - Es posible que se hayan compilado un conjunto de artefactos diferentes, desde repositorios de código diferentes lo cual origino diferentes empaquetados si, todos sus componentes han sido desplegados a la vez, al mismo tiempo, en el mismo despliegue y ello a deteniendo la operación de todo el sistema debido al despliegue del sistema el cuál ha sido, después, reiniciado.

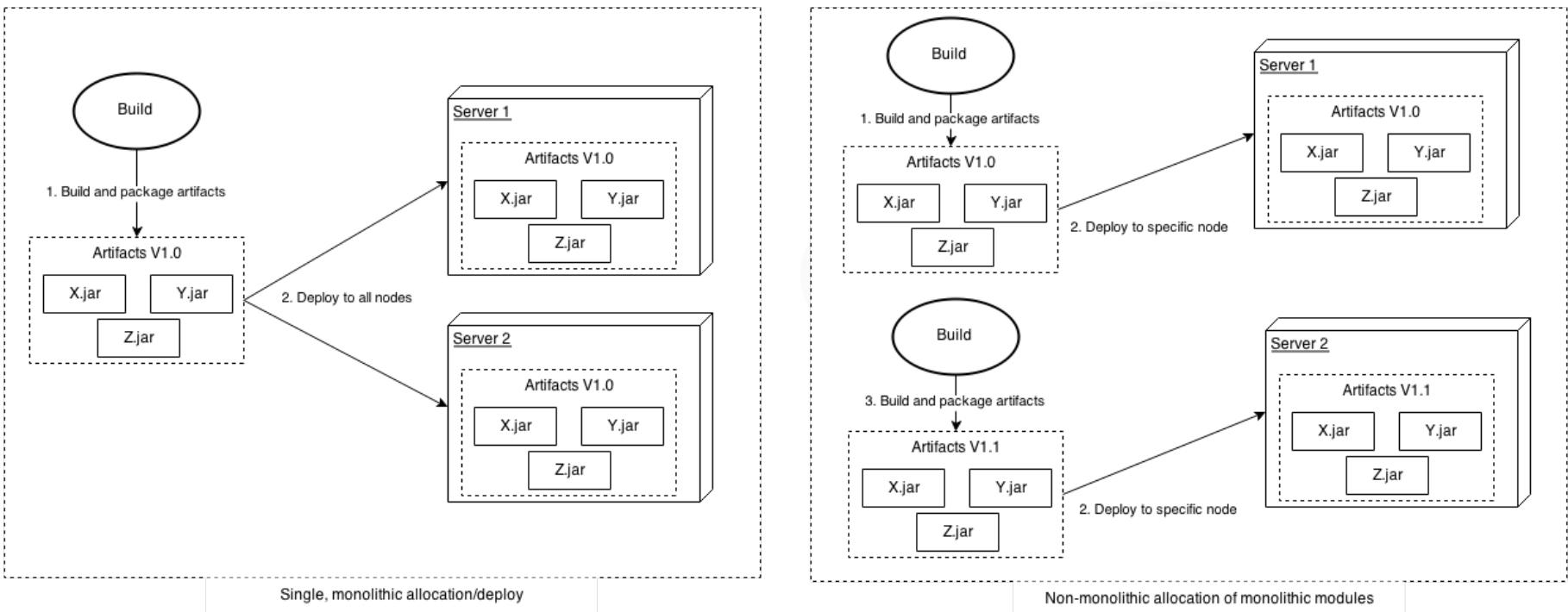


i.i ¿Qué son los sistemas monolíticos? (k)

- Asignación (allocation) monolítica.
- La asignación (allocation) no-monolítica implicaría desplegar diferentes compilados (con diferentes versiones de código o no), en diferentes nodos o equipos, en diferentes momentos y que la afectación del reinicio del sistema, sólo afecta al nodo donde se está realizando el despliegue.
- De nueva cuenta, la asignación monolítica o no-monolítica es independiente de la estructura de código del módulo o sistema ya que, diferentes versiones de un módulo monolítico pueden desplegarse de forma independiente.

i.i ¿Qué son los sistemas monolíticos? (I)

- Asignación (allocation) monolítica vs no-monolítica.





+

NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos? (m)

- Ejecución (runtime) monolítica.
- Un monolito en tiempo de ejecución tiene un solo proceso que ejecuta el trabajo del sistema (task).
- Desde los inicios de la informática, muchos sistemas “tradicionales” se han escrito de forma monolítica y se ejecutan en un único proceso.
- La ejecución monolítica es independiente de si la estructura del código fuente del módulo (o sistema) es monolítica o no-monolítica.



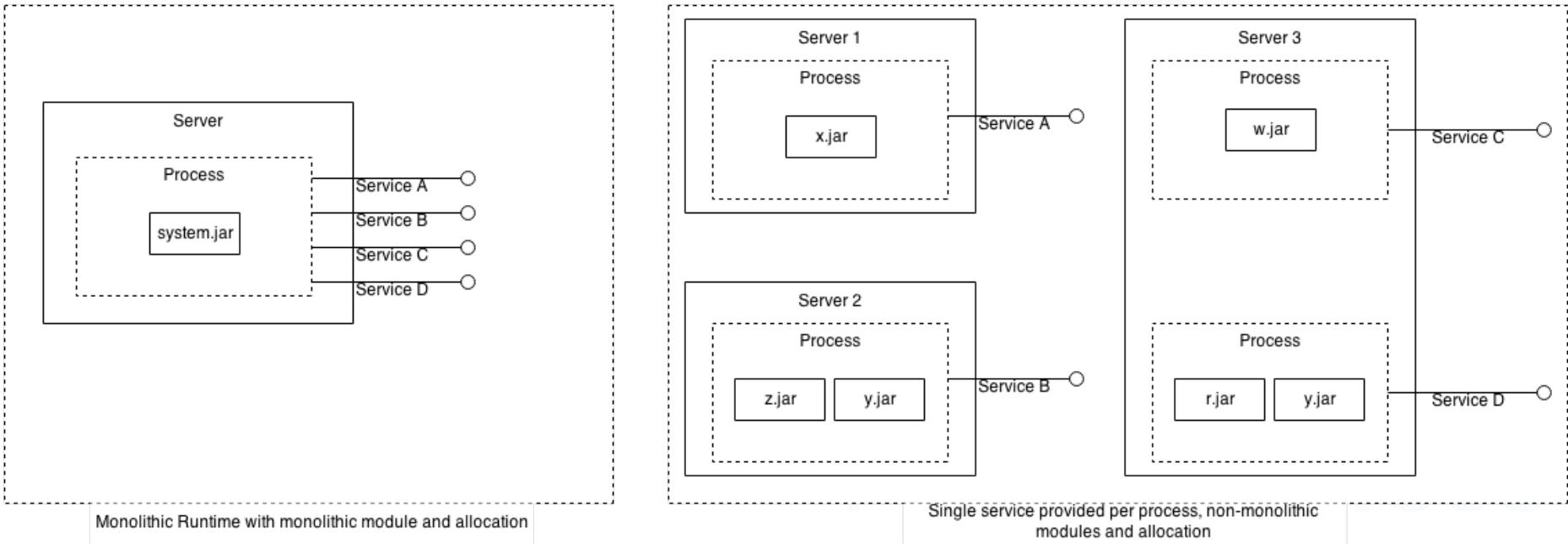
+

i.i ¿Qué son los sistemas monolíticos? (n)

- Ejecución (runtime) monolítica.
- Un monolito de tiempo de ejecución a menudo implica un monolito de asignación (asignación monolítica) si solo se despliega en un único nodo o equipo como componente principal.

i.i ¿Qué son los sistemas monolíticos? (ñ)

- Ejecución (runtime) monolítica.





+

i.i ¿Qué son los sistemas monolíticos? (o)

- Ventajas de los sistemas monolíticos (a):
 - Facilidad para desarrollar en entornos no colaborativos.
 - Facilidad de “debug & test”.
 - Performance, acceder a datos compartidos en memoria es más rápido que comunicación entre procesos (IPC) y aún más rápido que el “overhead” que ocasiona comunicación entre procesos en diferentes nodos mediante protocolos de comunicación como HTTP.
 - Sencillos de construir (compilar).
 - Simples de desplegar.
 - Facilidad de escalar horizontalmente.

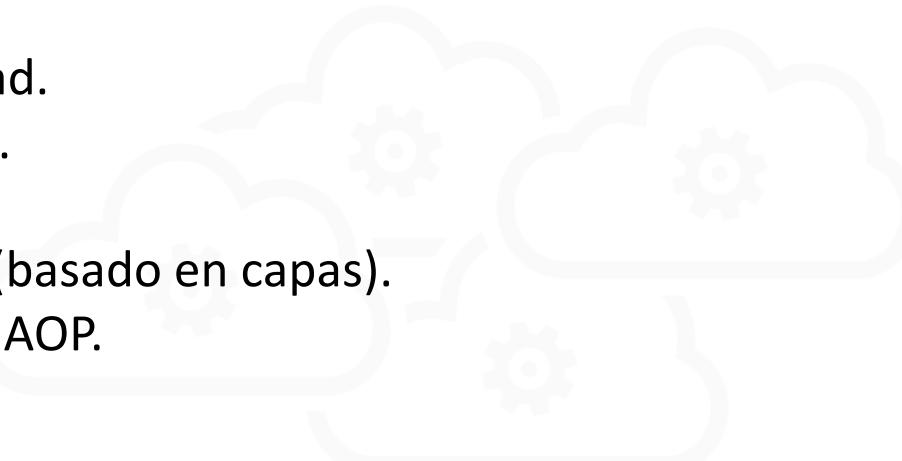


+

NETFLIX
OSS

i.i ¿Qué son los sistemas monolíticos? (p)

- Ventajas de los sistemas monolíticos (b):
 - Más fáciles de implementar seguridad.
 - Facilidad de monitoreo, operaciones.
 - Facilidad de planificación agile.
 - Fáciles de diseñar tradicionalmente (basado en capas).
 - Facilidad para la implementación de AOP.





+

i.i ¿Qué son los sistemas monolíticos? (q)

- Desventajas de los sistemas monolíticos (a):
 - Difíciles de mantener, difíciles de entender.
 - Difíciles de evolucionar, difíciles de entender.
 - Difíciles de desarrollar en entornos colaborativos.
 - Detienen la operativa total del sistema dado un error en el mismo.
 - Mantienen un alto acoplamiento entre sus componentes.
 - Componentes difíciles de reutilizar.
 - Costoso de escalar horizontal y verticalmente.
 - Existe un punto en el que no podrá escalar horizontal ni verticalmente debido a limitaciones técnicas.

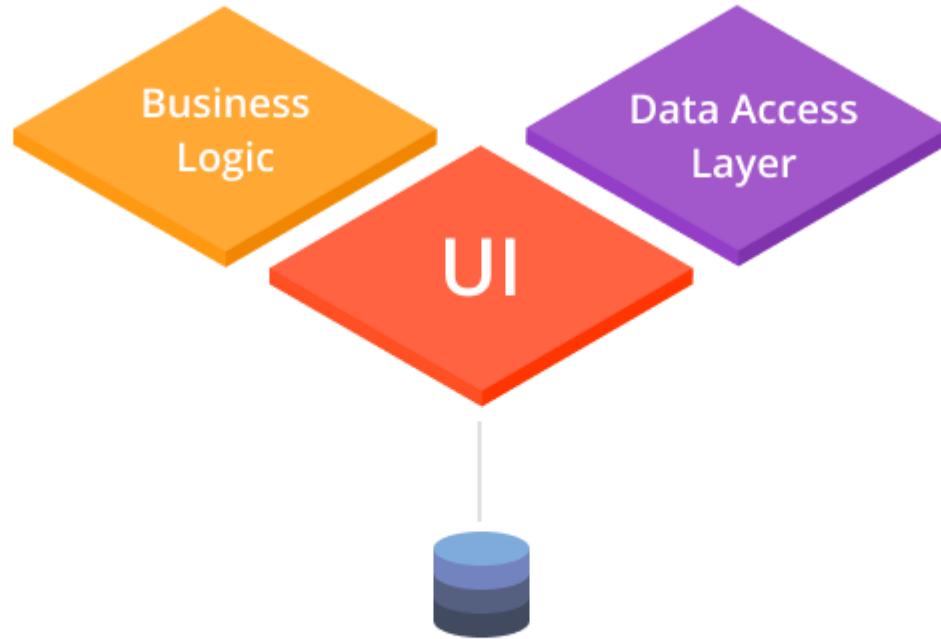


i.i ¿Qué son los sistemas monolíticos? (r)

- Desventajas de los sistemas monolíticos (b):
 - Al crecer ampliamente el código base de un aplicativo:
 - Mayor sobrecarga para IDEs de desarrollo.
 - Mayor sobre carga de memoria RAM para el entorno de desarrollo.
 - Mayor sobrecarga de contenedores de aplicaciones (contenedores web).
 - Difíciles de desplegar.
 - Dificultad para implementar CI / CD.
 - Requiere un fuerte compromiso con el stack tecnológico utilizado para su desarrollo.
 - entre otros ...



i.i ¿Qué son los sistemas monolíticos? (s)



Monolithic Architecture



+

NETFLIX
OSS

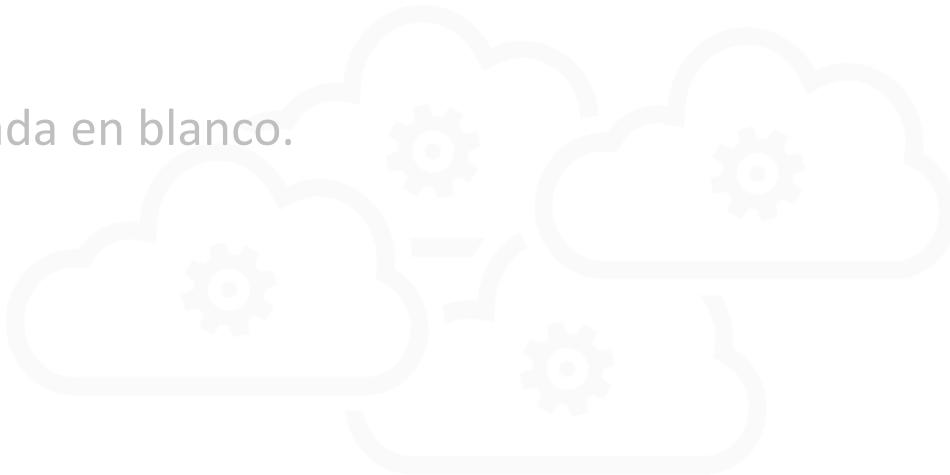
Resumen de la lección

i.i ¿Qué son los sistemas monolíticos?

- Comprendemos las características a nivel de código, despliegue y ejecución de sistemas monolíticos y no-monolíticos.
- Analizamos que un sistema monolítico implica el despliegue de un único compilado, independientemente de si es modular o no la estructura de su código.
- Comprendemos que un sistema monolítico se ejecuta en un único proceso.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

i. Arquitectura de sistemas monolíticos

- i.i ¿Qué son los sistemas monolíticos?
- i.ii Escalabilidad de sistemas monolíticos
- i.iii Protocolos de integración





+

NETFLIX
OSS

i.ii Escalabilidad de sistemas monolíticos



+

NETFLIX
OSS

Objetivos de la lección

i.ii Escalabilidad de sistemas monolíticos

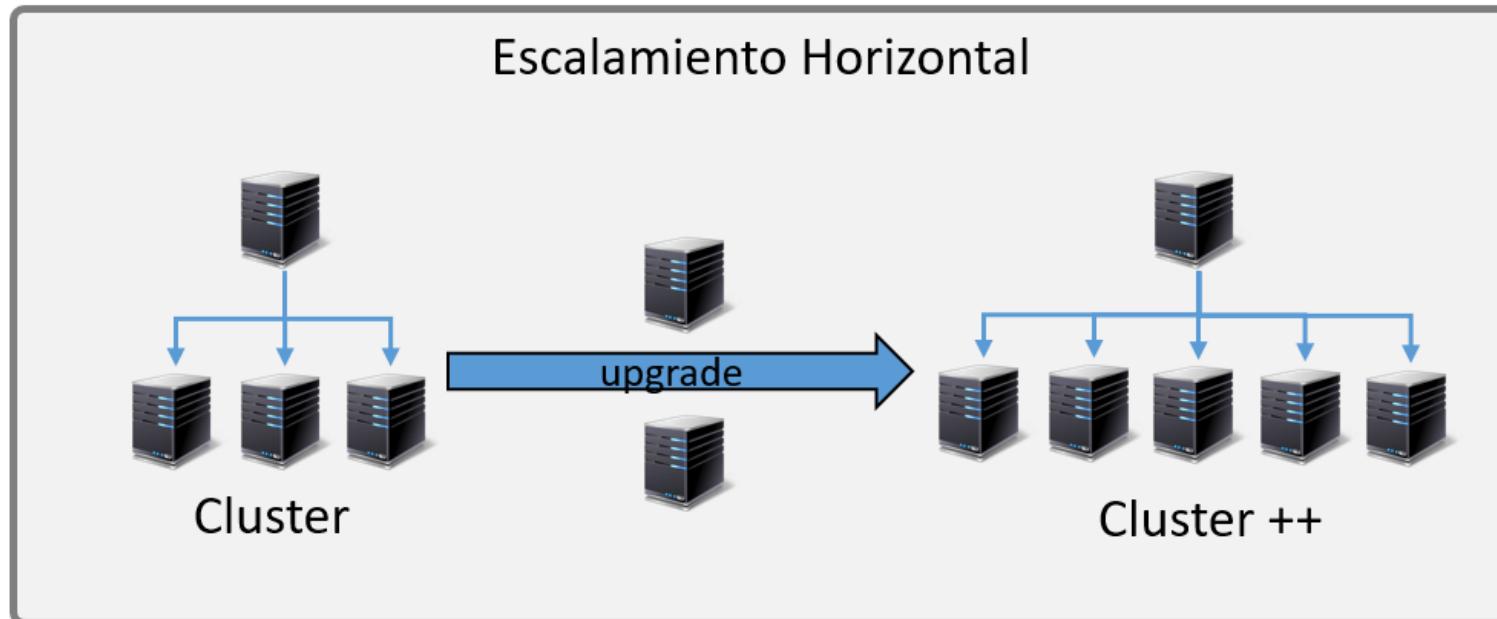
- Revisar los diferentes tipos de escalabilidad de los sistemas monolíticos.
- Analizar las ventajas y desventajas de los diferentes tipos de escalabilidad de los sistemas monolíticos.

i.ii Escalabilidad de sistemas monolíticos (a)

- Los sistemas monolíticos pueden escalar en dos dimensiones:
 - Escalamiento horizontal.
 - Escalamiento vertical.
- Sin embargo, el sistema monolítico, escala por completo, es decir, todos sus componentes escalan en la misma proporción y no sólo los componentes o servicios que se requiera.

i.ii Escalabilidad de sistemas monolíticos (b)

- Escalamiento horizontal.





+

i.ii Escalabilidad de sistemas monolíticos (c)

- Ventajas escalamiento horizontal.
- Amplio espectro de escalabilidad, debido a que se podrían agregar tantos servidores como sean necesarios.
- Se puede combinar con el escalamiento vertical.
- Permite la alta disponibilidad del servicio.
- Soporta balanceo de carga.
- Facilidad de escalamiento si se cuenta con el conocimiento requerido.



+

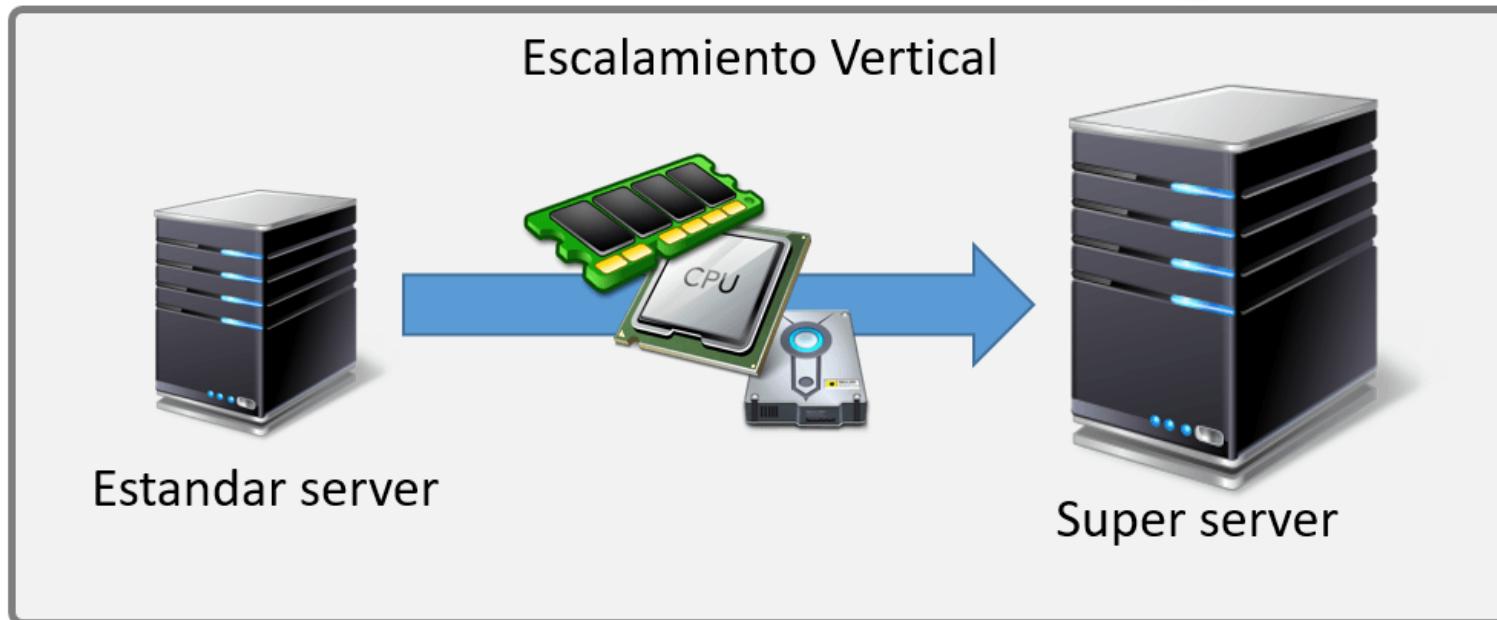
NETFLIX
OSS

i.ii Escalabilidad de sistemas monolíticos (d)

- Desventajas escalamiento horizontal.
- Costos medianamente aceptables aunque puede crecer exponencialmente.
- Requiere demasiado mantenimiento.
- El exceso en el mantenimiento requerido aumenta los costos operativos.
- Requiere una infraestructura más compleja.
- La aplicación requiere estar diseñada para trabajar en cluster.
- Dificultad de implementación y configuración si no se cuenta con el conocimiento requerido.

i.ii Escalabilidad de sistemas monolíticos (e)

- Escalamiento vertical.



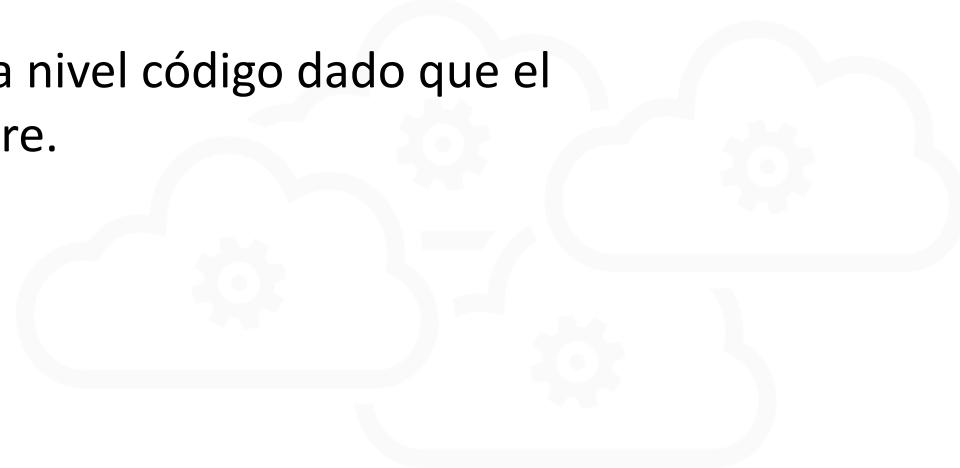


+

NETFLIX
OSS

i.ii Escalabilidad de sistemas monolíticos (f)

- Ventajas escalamiento vertical.
- No implica cambios en el sistema a nivel código dado que el escalamiento es a nivel de hardware.
- Fácilidad de implementación.
- Rapidez.





+

NETFLIX
OSS

i.ii Escalabilidad de sistemas monolíticos (g)

- Desventajas escalamiento vertical.
- El crecimiento está limitado por el hardware.
- Fallas a nivel hardware debido a su actualización pueden resultar catastróficas.
- El escalamiento vertical no proporciona alta disponibilidad del servicio.
- Elevados costos para la compra de hardware (disco, RAM y procesadores) más reciente y potente.
- No todos los equipos pueden escalar verticalmente, existe un límite.
- Posiblemente se llegue a un punto donde el hardware ya no puede escalar y será requerido adquirir otro equipo.



+

NETFLIX
OSS

Resumen de la lección

i.ii Escalabilidad de sistemas monolíticos

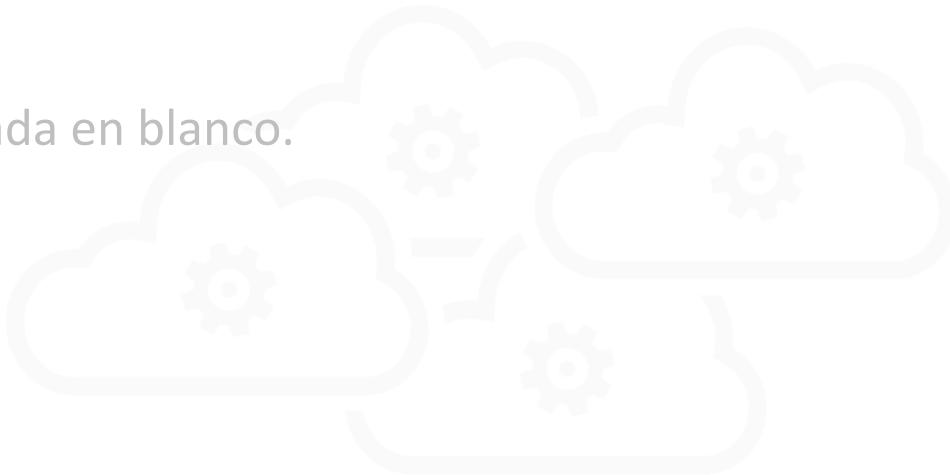
- Comprendemos los diferentes tipos de escalabilidad aplicables a sistemas monolíticos.
- Analizamos las diferencias, ventajas y desventajas del escalamiento horizontal y vertical.
- Comprendemos que para aplicar escalabilidad horizontal, la aplicación debe estar preparada para ello a nivel implementación de código.
- Analizamos que escalar verticalmente un producto de software no proporciona alta disponibilidad del servicio.



+

NETFLIX
OSS

Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

i. Arquitectura de sistemas monolíticos

- i.i ¿Qué son los sistemas monolíticos?
- i.ii Escalabilidad de sistemas monolíticos
- i.iii Protocolos de integración



Microservices



+

NETFLIX
OSS

i.iii Protocolos de integración

Objetivos de la lección

i.iii Protocolos de integración

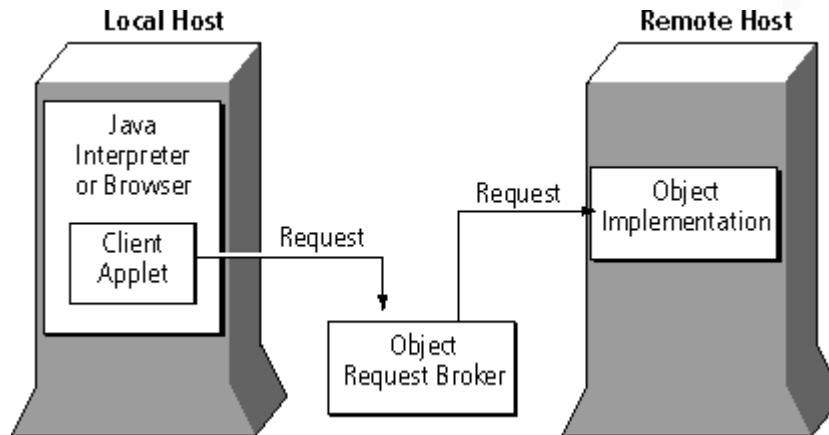
- Comentar brevemente los diferentes protocolos de integración utilizados más populares para sistemas monolíticos y/o distribuidos.

i.iii Protocolos de integración (a)

- CORBA
- Common Object Request Broker.
- Es un estandar, definido por la OMG (Object Management Group) que permite a los objetos realizar solicitudes y recibir respuestas entre sistemas distribuidos heterogeneos de forma transparente.
- Permite la interoperabilidad entre sistemas:
 - En diferentes maquinas, nodos o servidores.
 - Heterogeneos en ambientes distribuidos a través de la red.

i.iii Protocolos de integración (b)

- CORBA

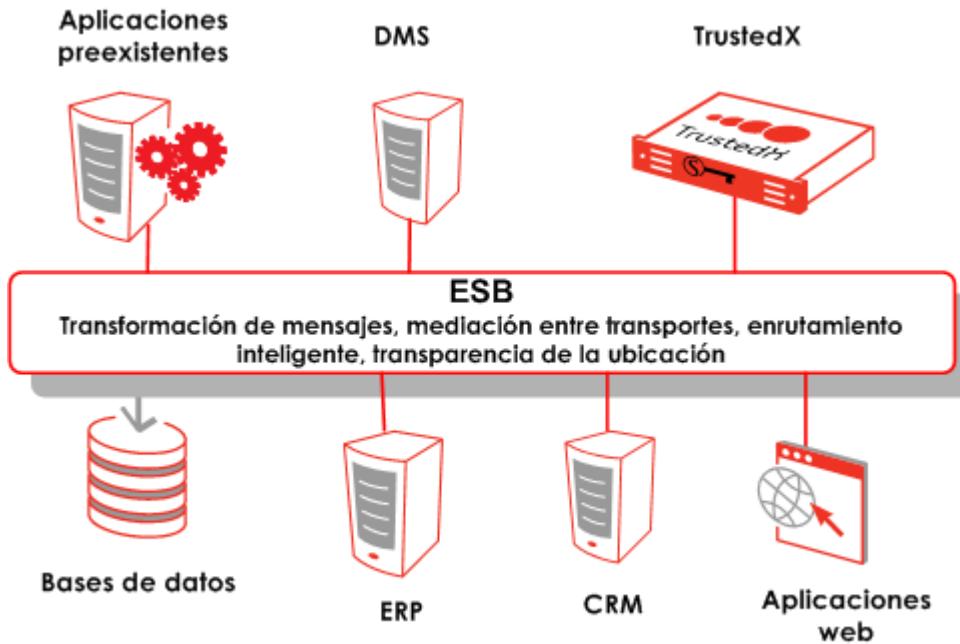


i.iii Protocolos de integración (c)

- EAI
- Enterprise Application Integration.
- Es un framework compuesto por una colección de tecnologías y servicios en forma de middleware que permite la integración de sistemas y aplicaciones heterogéneas en las organizaciones.
- Colección de patrones, para interconectar sistemas de forma eficiente, para evitar los problemas comunes de integración.

i.iii Protocolos de integración (d)

- EAI



i.iii Protocolos de integración (e)

- EAI





+

NETFLIX
OSS

i.iii Protocolos de integración (f)

- Web-services
- Los Web-services, o servicios web, son un método de comunicación entre dos componentes de software a través de una red.
- La implementación de un web-service utiliza una colección de protocolos abiertos y estándares, requeridos para intercambiar datos entre aplicaciones o sistemas heterogéneos, las cuales pueden estar escritas en diversos lenguajes de programación.
- La interoperatividad entre sistemas heterogéneos, por ejemplo entre Java y Python o Windows y Linux se debe al uso de estándares abiertos.



+

NETFLIX
OSS

i.iii Protocolos de integración (g)

- Web-services
- Los Web-services utilizan XML como formato de intercambio de mensajes de forma estandarizada.
- XML-RPC es el protocolo más sencillo de implementar para el intercambio de datos entre sistemas utilizado para llevar a cabo llamadas a procedimientos remotos, RPCs.
- Los “Remote Procedure Call” son un protocolo de red que permite a un programa a ejecutar código en una máquina remota.



+

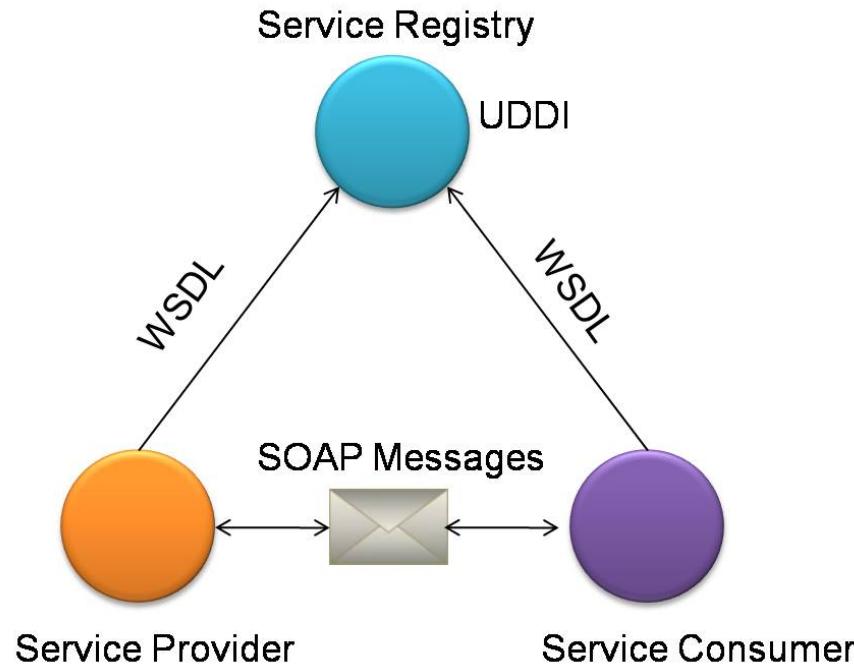
NETFLIX
OSS

i.iii Protocolos de integración (h)

- Web-services
- Las solicitudes “XML-RPC” son una combinación entre contenido XML y encabezados HTTP; su simpleza hizo que el estándar evolucionara a SOAP, siendo éste el protocolo básico en la implementación de Web-services.

i.iii Protocolos de integración (i)

- Web-services





i.iii Protocolos de integración (j)

- REST
- REpresentational State Transfer, o transferencia de estado representacional, es un conjunto de restricciones con las que se define un estilo de arquitectura de software para la implementación de web-services respetando el protocolo HTTP.
- REST permite la interoperabilidad de sistemas distribuidos mediante hipermédia.



+

NETFLIX
OSS

i.iii Protocolos de integración (k)

- REST
- Las restricciones que definen a un sistema basado en REST son (a):
 - **Cliente-servidor:** Esta restricción mantiene al cliente y al servidor débilmente acoplados.
 - **Stateless:** El servidor no mantiene ninguna información con respecto a las peticiones del cliente, es decir, no guarda estado. No implementa sesiones.



i.iii Protocolos de integración (I)

- REST
- Las restricciones que definen a un sistema basado en REST son (b):
 - **Cacheable:** Debe admitir un sistema de almacenamiento en cache que permita evitar repetir una misma ejecución de una solicitud de un cliente al servidor para recuperar un mismo recurso.
 - **Interfaz uniforme:** Define una interfaz uniforme para administrar cada interacción que se produzca entre el cliente y el servidor. Esta restricción indica que cada recurso que manipula el sistema basado en REST es representado por una única URI, manteniendo un único identificador.



i.iii Protocolos de integración (m)

- REST
- Las restricciones que definen a un sistema basado en REST son (c):
 - **Sistema en capas:** El servidor puede disponer de diversas capas en su implementación lo cual ayuda a mejorar su escalabilidad, rendimiento y seguridad.
 - **Código bajo demanda (opcional):** Esta restricción permite que el cliente pueda solicitar código bajo demanda para que el servidor lo retorne y el cliente pueda ejecutarlo.



+

NETFLIX
OSS

i.iii Protocolos de integración (n)

- REST
- REST esta dirigido por recursos los cuales son cualquier tipo de objeto a los que un cliente puede acceder.
- En el diseño de sistemas basados en REST predomina la correcta implementación del protocolo HTTP para interactuar con los servicios que exponen los recursos.
- El mayor grado de madurez en una arquitectura de servicios basado en REST sugiere la implementación de HATEOAS.



+

i.iii Protocolos de integración (ñ)

- REST
- HATEOAS (Hypermedia As The Engine Of Application State) o Hipermédia como motor del estado de la aplicación, define que cada que un cliente solicita una operación sobre un recurso, el servidor devolverá, en formato de hipervínculo, la navegación asociada a los recursos u operaciones disponibles para el recurso solicitado.
- HATEOAS permite el descubrimiento de servicios mediante los hipervínculos que permiten la navegabilidad de un cliente REST a través de las URLs de recursos relacionados al recurso solicitado.



+

i.iii Protocolos de integración (o)

- REST
- HATEOAS desacopla el cliente del servidor permitiendo que el servidor evolucione de manera independiente y forza a que el cliente sea lo suficientemente inteligente para poder implementar la navegabilidad de recursos y descubrir nuevos recursos de manera eficaz.



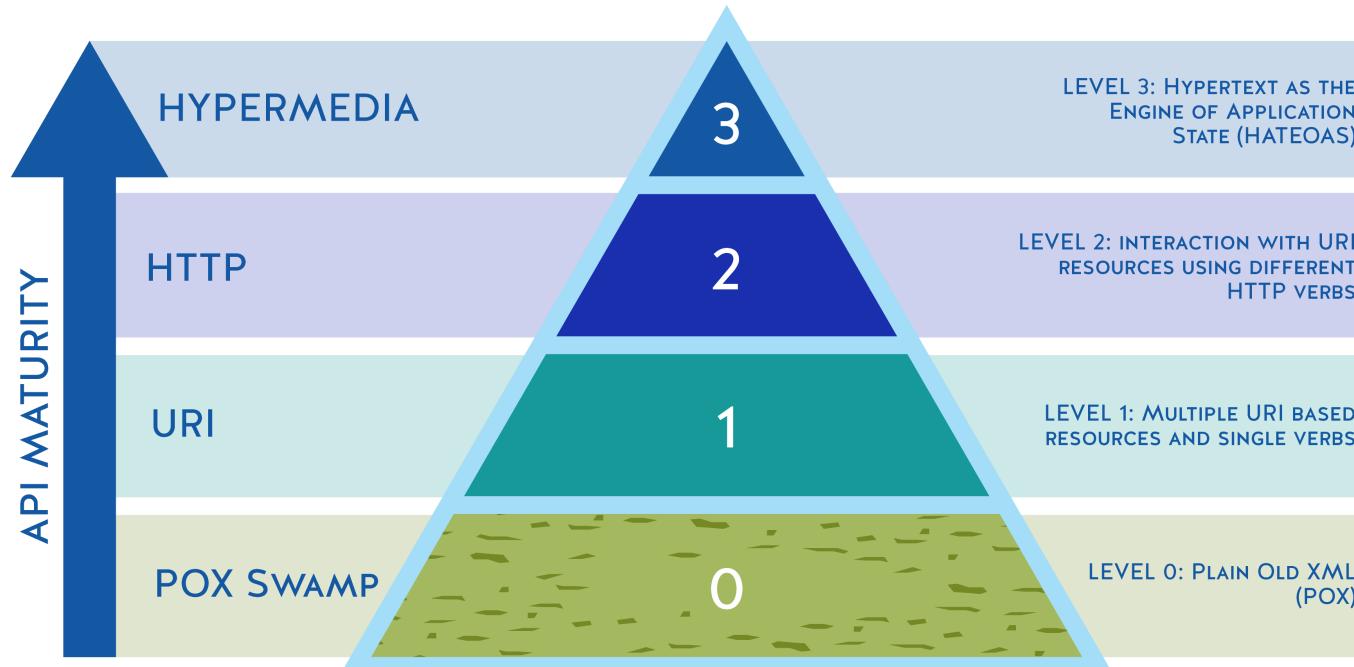
+

NETFLIX
OSS

i.iii Protocolos de integración (p)

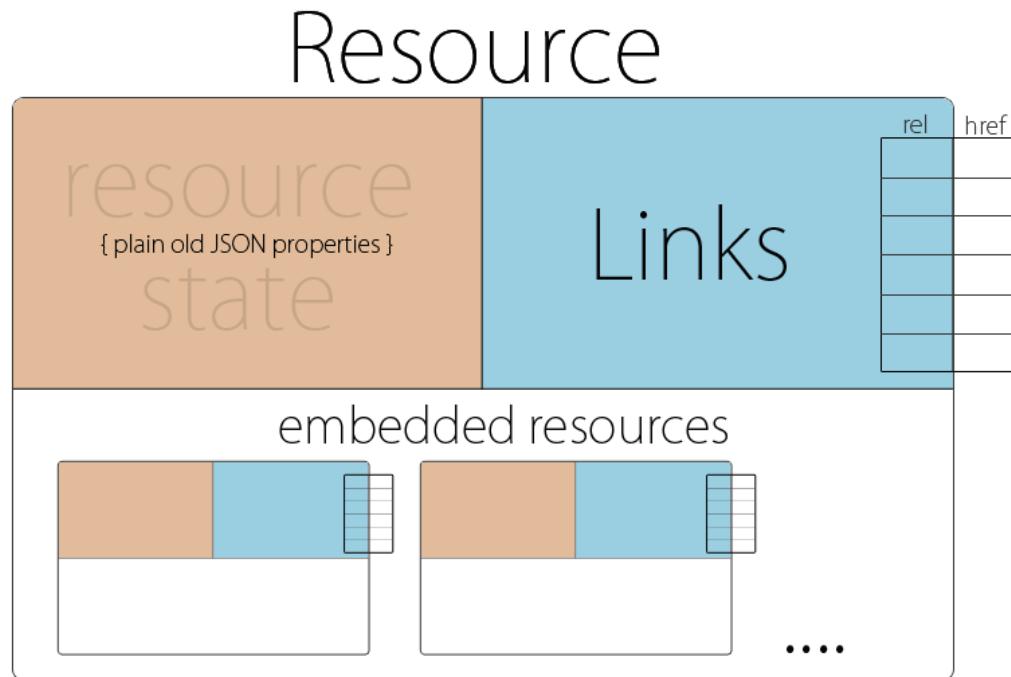
- REST

THE RICHARDSON MATURITY MODEL



i.iii Protocolos de integración (q)

- REST





+

NETFLIX
OSS

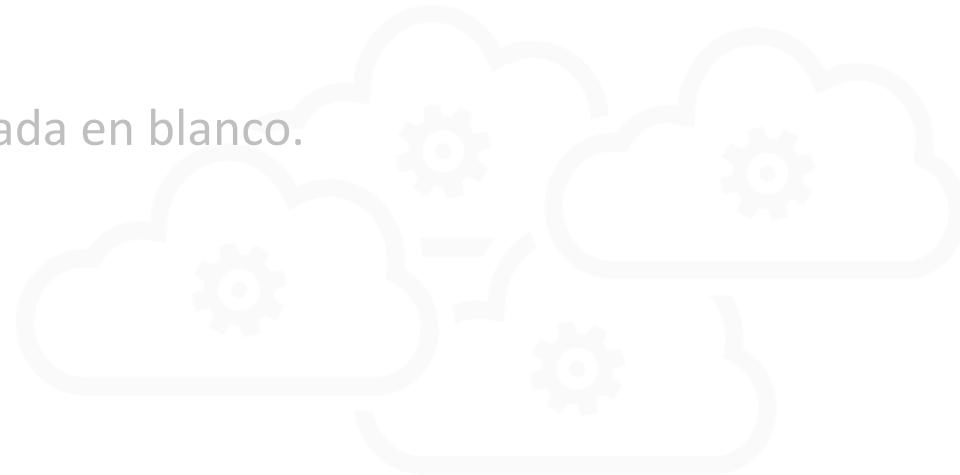
Resumen de la lección

i.iii Protocolos de integración

- Analizamos a grandes rasgos algunos de los estilos y protocolos que facilitan la interoperabilidad entre sistemas.
- Comprendemos la diferencia entre dichos protocolos y analizamos su aplicabilidad en sistemas distribuidos.
- Conocemos la principal diferencia entre web-services basados en SOAP y servicios REST o (web-services REST).



Esta página fue intencionalmente dejada en blanco.



Microservices



+

3. Contenido

- i. Arquitectura de sistemas monolíticos
- ii. **Introducción a la Arquitectura Orientada a Servicios**
- iii. Fundamentos Spring Boot 2.x
- iv. Arquitectura de Microservicios
- v. Implementación de Microservicios con Spring Boot
- vi. Microservicios con Spring Cloud y Spring Netflix OSS



+

NETFLIX
OSS

ii. Introducción a la Arquitectura Orientada a Servicios



Microservices



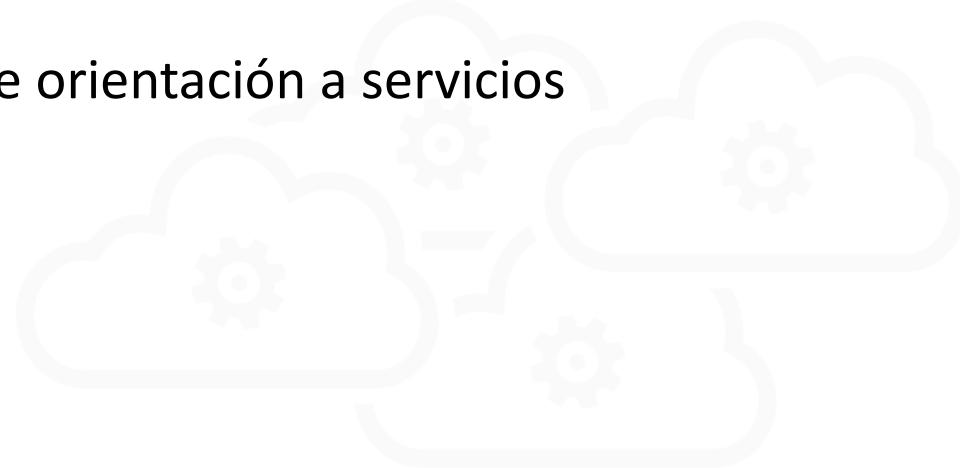
+

NETFLIX
OSS

ii. Introducción a la Arquitectura Orientada a Servicios

ii.i ¿Qué es SOA?

ii.ii Principios de diseño de orientación a servicios



Microservices



+

NETFLIX
OSS

ii.i ¿Qué es SOA?





+

Objetivos de la lección

ii.i ¿Qué es SOA?

- Analizar lo que es la Arquitectura Orientada a Servicios o Service Oriented Architecture.
- Comprender SOA a nivel macro y analizar sus posibles implementaciones.



ii.i ¿Qué es SOA? (a)

- SOA
- La Arquitectura Orientada a Servicios (SOA) es un framework conceptual que permite a las organizaciones unir los objetivos de negocio con la infraestructura de TI integrando los datos y la lógica de negocio de sus sistemas separados.
- SOA es un enfoque de desarrollo de aplicaciones de software empresarial, en el cual los procesos del software se descomponen en servicios, que después se hacen disponibles y visibles en una red.



ii.i ¿Qué es SOA? (b)

- SOA es una representación de una arquitectura abierta, extensible y federada basada en composición, que promueve la orientación a los servicios interoperables e independientes de los proveedores, los cuales pueden ser identificados en catálogos con gran potencial de reutilización e implementados como servicios Web.
- Cada servicio expuesto provee funcionalidades para poder ser adecuado a las necesidades de la empresa, mientras esconde los detalles inherentes de implementación.

ii.i ¿Qué es SOA? (c)

- SOA aborda la complejidad, inflexibilidad y debilidades de los enfoques tradicionales para la interoperabilidad de aplicaciones separadas, tales como compartición de archivos, compartición de base de datos, conversión y adaptación de estructuras de I/O para la intercomunicación de sistemas.

ii.i ¿Qué es SOA? (d)

- SOA provee la infraestructura de tecnologías de la información que permite a diferentes aplicaciones intercambiar datos y participar en los procesos de negocio, independientemente del sistema operativo o de los lenguajes de programación con los cuales los servicios y los sistemas interconectados han sido desarrollados.



+

NETFLIX
OSS

ii.i ¿Qué es SOA? (e)

- Existen diversas definiciones de SOA que incluyen el término “servicios web” o “web-services” sin embargo, es necesario hacer la distinción de estos conceptos y aclarar que SOA no es lo mismo que Servicios Web.
- La Arquitectura Orientadas a Servicios, a diferencia de los Servicios Web, define y trata un paradigma de orientación a servicios, en tanto que los Servicios Web son sólo una forma posible de implementar la infraestructura SOA utilizando una estrategia de implementación tecnológica específica.



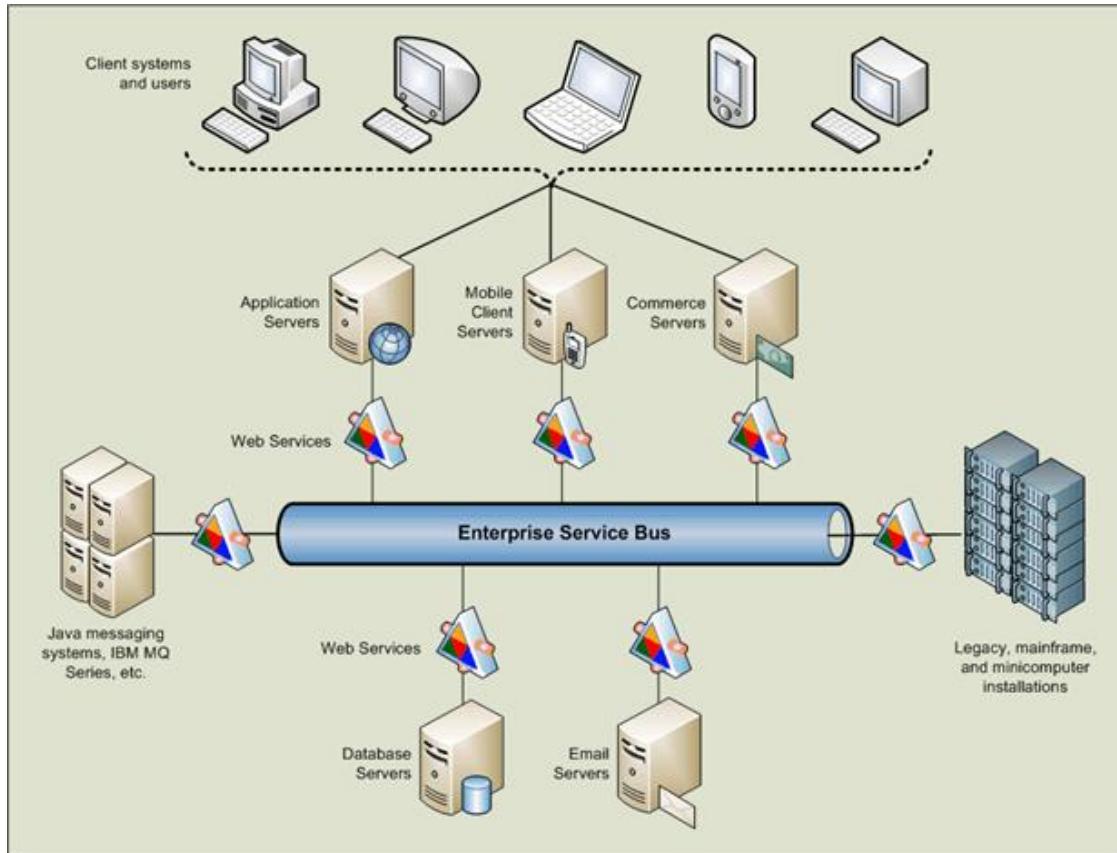
+

NETFLIX
OSS

ii.i ¿Qué es SOA? (f)

- Podemos resumir que SOA es un paradigma arquitectónico que permite el tratamiento de procesos de negocio distribuidos de sistemas heterogéneos, que se encuentran bajo el control o responsabilidad de diferentes propietarios donde sus conceptos clave son:
 - Disponibilizar un activo de negocio como un servicio.
 - La interoperabilidad entre diversos lenguajes y aplicaciones, y
 - El bajo acoplamiento entre los componentes.
- Los principales actores en una SOA son la infraestructura, la arquitectura y los procesos.

ii.i ¿Qué es SOA? (g)





+

NETFLIX
OSS

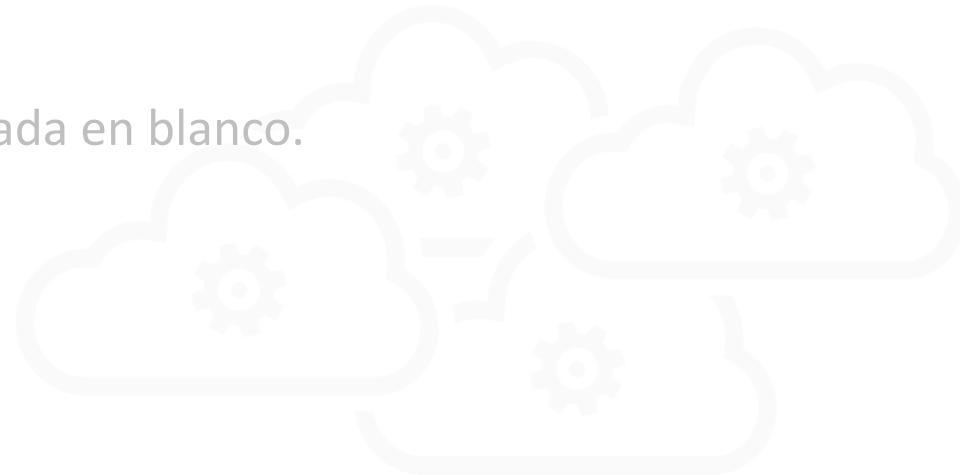
Resumen de la lección

ii.i ¿Qué es SOA?

- Analizamos a grandes rasgos la Arquitectura Orientada a Servicios.
- Comprendemos de que trata las Arquitecturas SOA.
- Debatimos referente a las diversas implementaciones SOA en la industria.
- Analizamos el marco de trabajo SOA como un conjunto de prácticas que convierten los activos funcionales de una empresa a servicios para que sean reutilizados a través de sus sistemas separados.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

ii. Introducción a la Arquitectura Orientada a Servicios

ii.i ¿Qué es SOA?

ii.ii Principios de diseño de orientación a servicios



Microservices



+

NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios

Microservices



+

NETFLIX
OSS

Objetivos de la lección

ii.ii Principios de diseño de orientación a servicios

- Verificar los principios de diseño de orientación a servicios.
- Analizar los diferentes principios de diseño de orientación a servicios para tener en cuenta al momento de especificar servicios-web ya sea basados en arquitectura SOA o en arquitectura basada en microservicios.
- Distinguir las diferencias entre los distintos principios de diseño de orientación a servicios.



+

NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (a)

- Principios de diseño de orientación a servicios:
 - Contratos de servicios estandarizados.
 - Bajo acoplamiento.
 - Abstracción.
 - Reusabilidad.
 - Autonomía.
 - Sin estado.
 - Descubrimiento.
 - Composición.





+

ii.ii Principios de diseño de orientación a servicios (b)

- Contrato de servicios estandarizados (a).
- Cuando un servicio se implementa como un Servicio Web, se debe adherir un contrato o interfaz de comunicaciones explícitamente declarado y definido, colectivamente, por uno o más descriptores del servicio, en el cual debe figurar especificaciones como:
 - Nombre del Servicio.
 - Forma de acceso.
 - Funcionalidades que ofrece.
 - Descripción de los datos de entrada de cada funcionalidad que ofrece.
 - Descripción de los datos de salida de cada funcionalidad que ofrece.

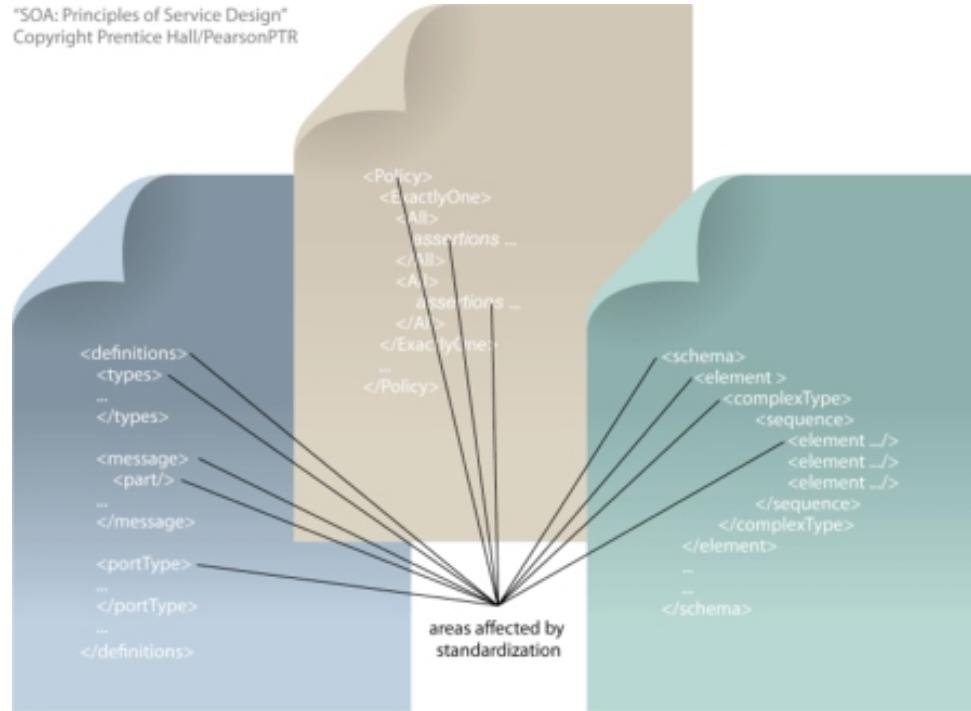


ii.ii Principios de diseño de orientación a servicios (c)

- Contrato de servicios estandarizados (b).
- Mediante contratos de servicios estandarizados, todo consumidor de servicios accederá a éste mediante su contrato definido, logrando la independencia entre el consumidor y la implementación del servicio.
- De esta forma, se evita el manejo incorrecto de los datos, se evita trabajo innecesario al momento de la invocación del servicio y también se pone de manifiesto de la existencia de un modelo de datos a nivel de servicio, siendo esta una de las primeras necesidades que se debe tener en cuenta al momento de definir servicios.

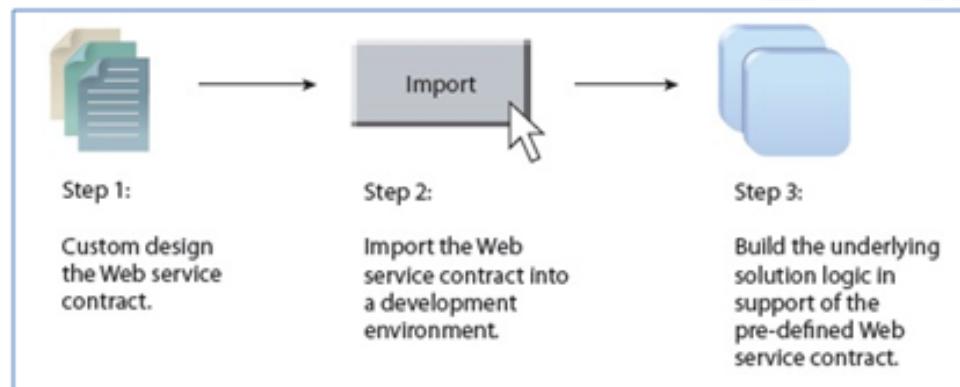
ii.ii Principios de diseño de orientación a servicios (d)

- Contrato de servicios estandarizados (c).



ii.ii Principios de diseño de orientación a servicios (e)

- Contrato de servicios estandarizados (c).





+

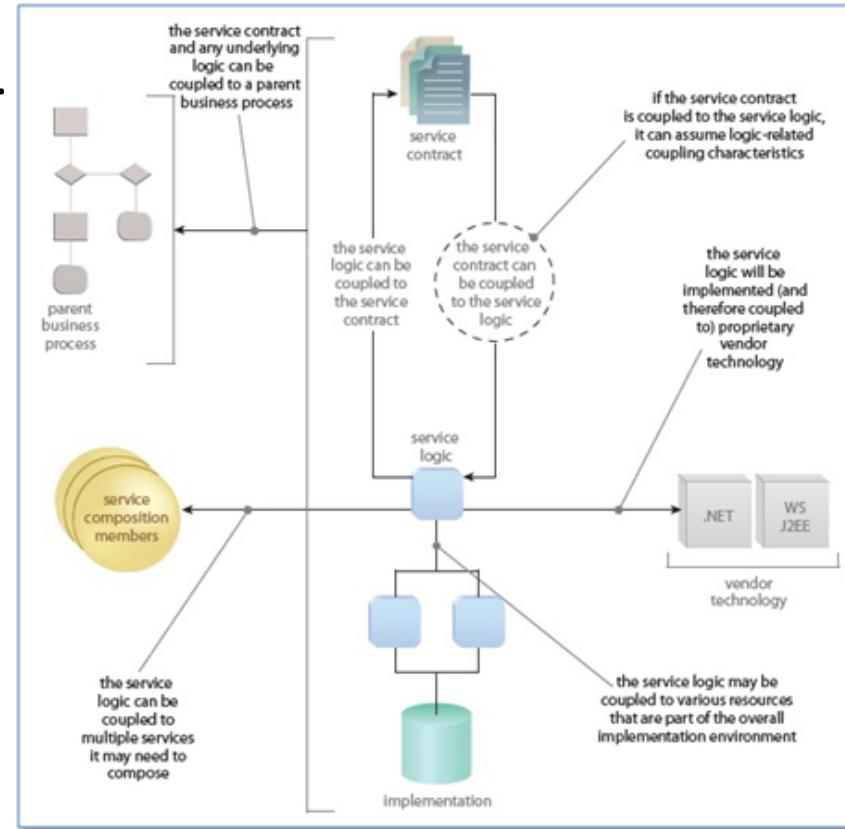
NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (f)

- Bajo acoplamiento (a).
- Este principio hace referencia a que los servicios tienen que ser independientes los unos de los otros.
- Para lograr el bajo acoplamiento entre los servicios, se define que el único medio de acceso a los servicios es el contrato o interfaz, logrando así la independencia entre el servicio que se va a ejecutar y el que lo llama.
- El bajo acoplamiento permite que los servicios sean totalmente reutilizables.

ii.ii Principios de diseño de orientación a servicios (g)

- Bajo acoplamiento (b).





+

NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (h)

- Abstracción (a):
- A nivel fundamental, la abstracción permite ocultar los detalles de implementación de un servicio, tanto como sea posible.
- El principio de abstracción permite encapsular el servicio como una “caja negra”, del cuál no se saben los detalles y únicamente está definido por su contrato, habilitando así el bajo acoplamiento, que a su vez, la definición y especificación del contrato es el mínimo acoplamiento posible entre el servicio y un consumidor.



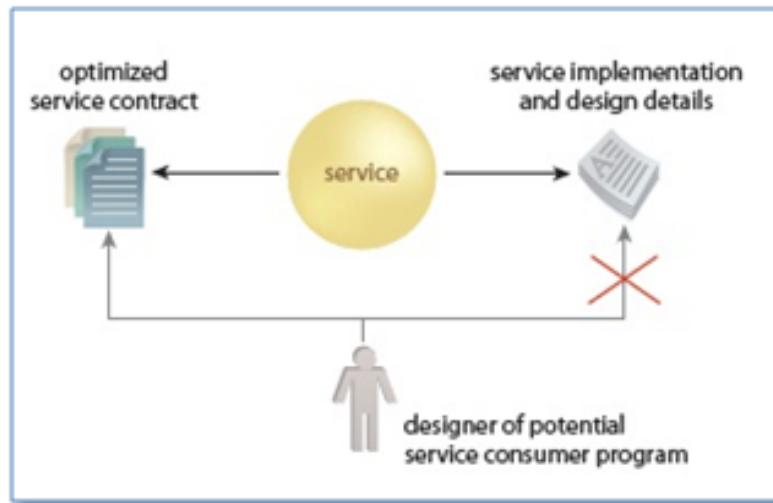
+

ii.ii Principios de diseño de orientación a servicios (i)

- Abstracción (b):
- El uso de estándares permite definir interfaces uniformes que esconden la lógica del servicio respecto del entorno que le rodea (sistemas proveedores y consumidores).
- Este nivel de abstracción permite centrarse exclusivamente en la especificación del servicio, sin incluir información tecnológica ni de ninguna otra naturaleza, más allá de la propia especificación estándar del servicio, desde el punto de vista del negocio al que sirve, la cual queda definida en su contrato.

ii.ii Principios de diseño de orientación a servicios (j)

- Abstracción (c):





+

NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (k)

- Reusabilidad (a):
- La reusabilidad (reutilización) es el principal principio de la arquitectura SOA, cada servicio debe ser analizado, diseñado y construido de manera que su uso pueda ser explotado al máximo por otros sistemas consumidores del servicio.
- Los servicios deben de ser definidos de tal forma que puedan utilizarse en diferentes contextos y satisfacer distintos objetivos de negocio, únicamente centrandose en las capacidades del negocio y no en alguna tecnología específica.



+

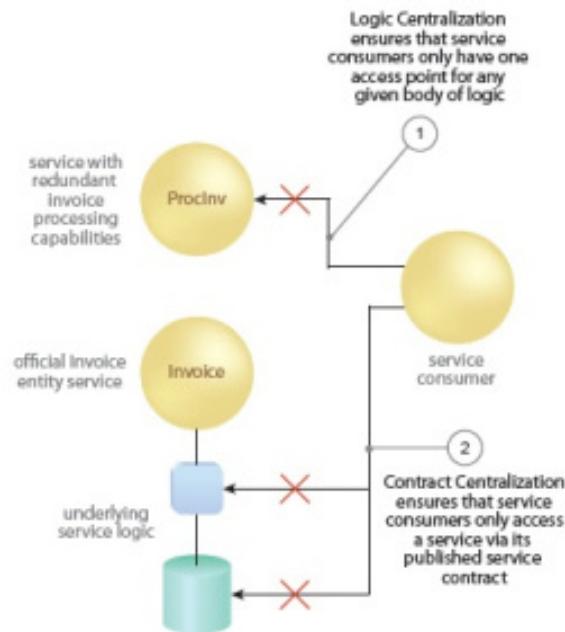
NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (I)

- Reusabilidad (b):
- De esta manera, los servicios pueden ser reusables dentro de la misma aplicación, dentro del dominio de aplicaciones de la empresa o incluso dentro del dominio público.
- Por otro lado, estos servicios reutilizables deben estar diseñados de manera tal que su solución lógica sea independiente de cualquier proceso de negocio o tecnología en particular.

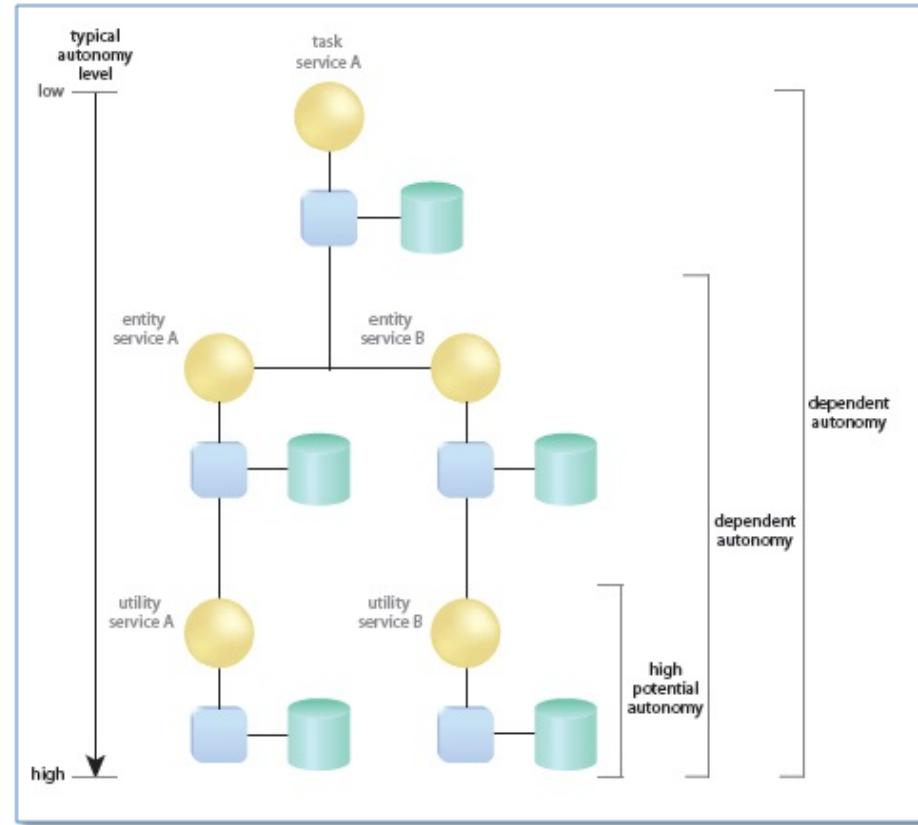
ii.ii Principios de diseño de orientación a servicios (m)

- Reusabilidad (c):
- Con este principio se busca reducir las posibilidades de duplicación de lógica.



ii.ii Principios de diseño de orientación a servicios (n)

- Autonomía:
- Este principio refiere a que todo servicio debe tener su propio entorno de ejecución, logrando que dicho servicio sea totalmente independiente.
- De esta forma se asegura la reusabilidad del servicio desde el punto de vista de la plataforma de ejecución.





+

NETFLIX
OSS

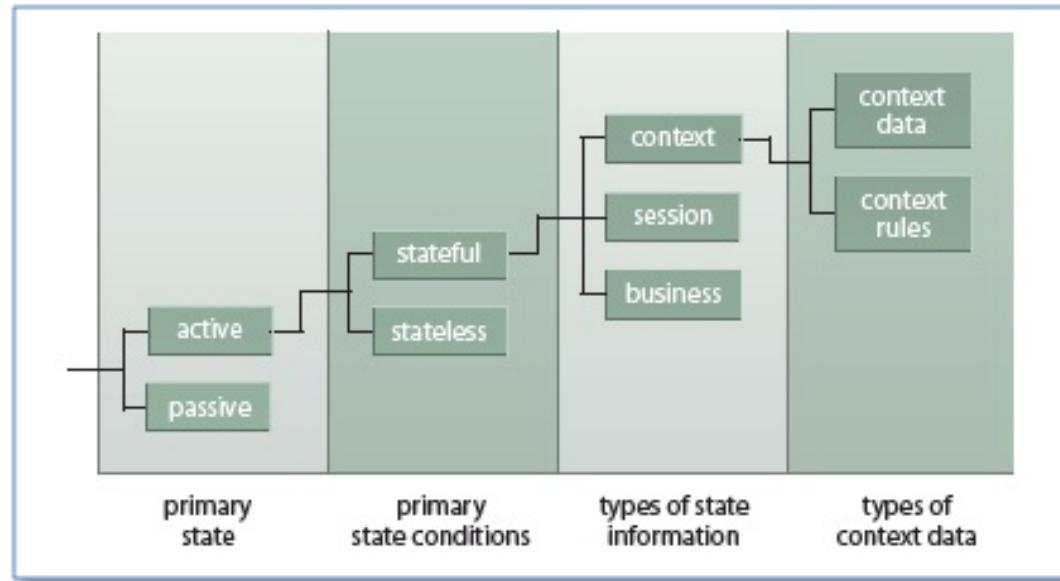
ii.ii Principios de diseño de orientación a servicios (ñ)

- Sin estado (a):
- Los servicios no deben de almacenar algún tipo de información con referencia a los consumidores del mismo.
- La gestión de exesiva carga de información minimiza y deteriora la escalabilidad del servicio lo cual afecta gravemente la disponibilidad del servicio.
- De forma ideal, todos los datos que necesita un servicio para su ejecución deben provenir directamente de los parámetros de entrada que provee el consumidor del servicio.



ii.ii Principios de diseño de orientación a servicios (o)

- Sin estado (b):



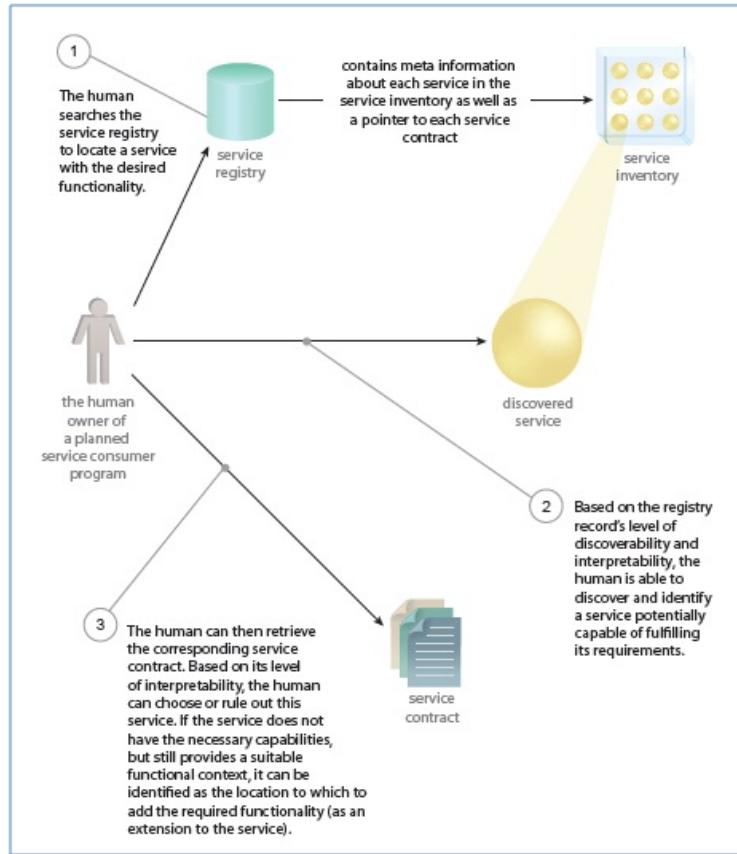


ii.ii Principios de diseño de orientación a servicios (p)

- Descubrimiento (a):
- Todo servicio expuesto debe poder ser descubierto de alguna forma para que pueda ser utilizado.
- El descubrimiento de servicios evita la ducplicidad accidental de servicios que proporcionen una misma funcionalidad.
- Para el caso de aplicar descubrimiento de servicios en una arquitectura SOA basada en Servicios Web, se deberá contar con la publicación de los servicio a traves de un registro UDDI (Universal Description, Discovery and Integration).

ii.ii Principios de diseño de orientación a servicios (q)

- Descubrimiento (b):





+

NETFLIX
OSS

ii.ii Principios de diseño de orientación a servicios (r)

- Composición (a):
- El principio de composición asegura la habilidad efectiva de un servicio para ser utilizado para componer otros servicios más complejos.
- La composición es uno de los requisitos más críticos para lograr una arquitectura orientada a servicios.
- El diseño de composición de servicios más complejos, basados en servicios de menor nivel (más atómicos) deben de ser visualizados con anticipación para evitar un doble esfuerzo.

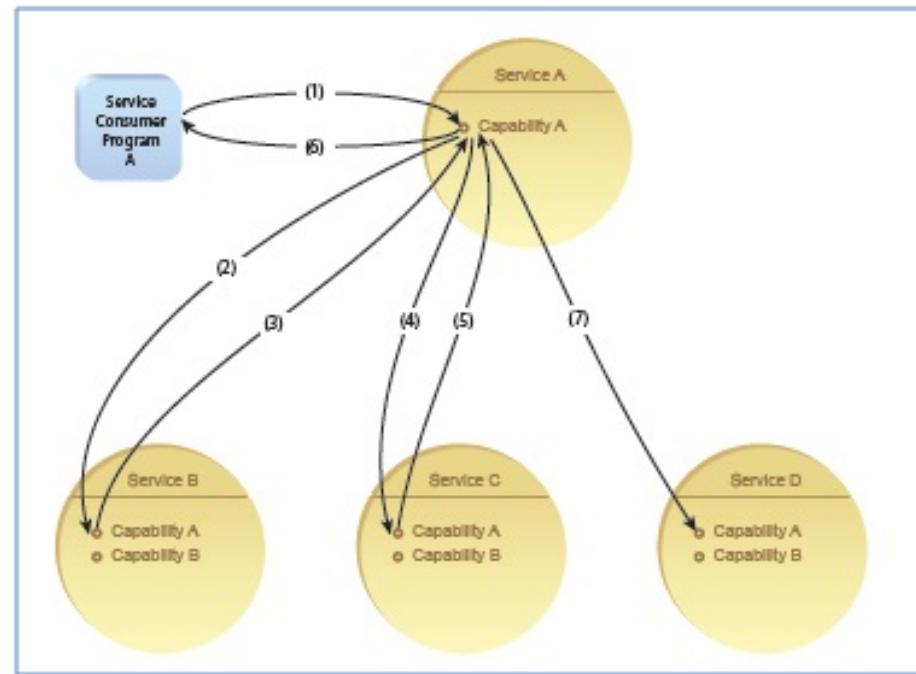


ii.ii Principios de diseño de orientación a servicios (s)

- Composición (b):
- El concepto de desarrollo de software a partir de componentes existentes de forma independiente, fomenta el concepto de composición.
- Dada la composición de servicios es que es posible automatizar un proceso de negocio debido a que, dicho proceso, se ejecuta mediante la combinación y composición de múltiples servicios.
- SOA dispone de su agilidad de implementación de servicios mediante la composición de servicios.

ii.ii Principios de diseño de orientación a servicios (t)

- Composición (c):
- Dada la implementación de Servicios Web como, pilar fundamental en una arquitectura SOA, WS-BPEL y WS-CDL permiten la composición de servicios mediante orquestación y coreografía.





+

NETFLIX
OSS

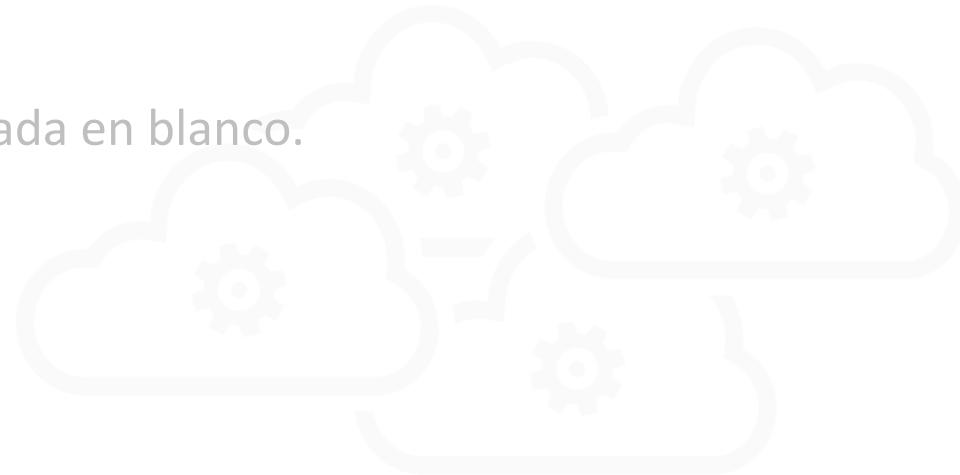
Resumen de la lección

ii.ii Principios de diseño de orientación a servicios

- Analizamos a grandes rasgos la Arquitectura Orientada a Servicios.
- Comprendemos de que trata las Arquitecturas SOA.
- Debatimos referente a las diversas implementaciones SOA en la industria.
- Analizamos el marco de trabajo SOA como un conjunto de prácticas que convierten los activos funcionales de una empresa a servicios para que sean reutilizados a través de sus sistemas separados.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

3. Contenido

- i. Arquitectura de sistemas monolíticos
- ii. Introducción a la Arquitectura Orientada a Servicios
- iii. **Fundamentos Spring Boot 2.x**
- iv. Arquitectura de Microservicios
- v. Implementación de Microservicios con Spring Boot
- vi. Microservicios con Spring Cloud y Spring Netflix OSS



+

NETFLIX
OSS

iii. Fundamentos Spring Boot 2.x





+

NETFLIX
OSS

iii. Fundamentos Spring Boot 2.x

iii.i Introducción a Spring Boot

iii.ii Configuración de propiedades en Spring Boot

iii.iii Perfiles

iii.iv Spring MVC

iii.v Spring Data JPA

iii.vi Spring Data REST

iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

iii.i Introducción a Spring Boot



Microservices



+

NETFLIX
OSS

Objetivos de la lección

iii.i Introducción a Spring Boot

- Aprender como iniciar un proyecto desde cero con Spring Boot.
- Conocer las principales herramientas para el rápido “up-and-running” de una aplicación con Spring Boot.
- Conocer que son los “Bill-of-materials” (BOM) y los “Starters” que manejan las dependencias en Spring Boot.
- Iniciar un aplicativo simple con Spring Boot.



+

iii.i Introducción a Spring Boot (a)

- Spring vs Spring Boot.
- Spring Framework es el Framework defacto para el desarrollo de aplicaciones Java empresariales, sin embargo requiere:
 - Amplia experiencia en el manejo de dependencias del proyecto de Spring e integración de proveedores externos como Hibernate, Quartz, Jackson, JavaMail, etc.
 - Configuración desde cero. Es necesario configurar todos los beans de infraestructura (boilerplate code).
 - Spring Framework es un conjunto de librerías, modulos, buenas prácticas, clases, etc, sin embargo no define como deben de ser configuradas dada su alta flexibilidad.

iii.i Introducción a Spring Boot (b)

- Spring Framework:
 - No exige un estandar de desarrollo y ni de configuración.
 - Es altamente flexible.
- Integrar otros frameworks y sub-proyectos de Spring cada vez se hace más complicado.





+

NETFLIX
OSS

iii.i Introducción a Spring Boot (c)

- Spring Boot es un proyecto “umbrella” que esta construido sobre Spring Framework.
- Ha sido desarrollado por la misma comunidad de Spring Framework en el cuál colaboran los más exitosos desarrolladores Java open-source de la industria aplicando las mejores prácticas de desarrollo.
- Spring Boot es:
 - Opinionado debido a que está basado en un estándar de desarrollo dirigido por sus colaboradores.
 - Hacer “pair-programming” con el equipo de Spring Boot.

iii.i Introducción a Spring Boot (d)

- Spring Boot:



A screenshot of a Twitter post from user @starbuxman. The post features a profile picture of a person with purple hair, the name "Josh Long (龙之春, जोश)" with a blue verified badge, and the handle "@starbuxman". To the right of the profile picture is a blue "Following" button. The tweet text is: "a reminder: @SpringBoot lets you pair-program with the #Spring team." Below the tweet is the timestamp "7:28 PM - 21 Sep 2017 from San Antonio, TX". The background of the slide features a faint watermark of the word "Microservices" and some decorative icons like clouds and gears.



+

NETFLIX
OSS

iii.i Introducción a Spring Boot (e)

- El principal objetivo de Spring Boot es:
 - Permitirle a los desarrolladores crear aplicaciones productivas basadas en Spring Framework con el mínimo esfuerzo requerido, para su configuración, basandose en la experiencia de los colaboradores del proyecto Spring Boot.



+

NETFLIX
OSS

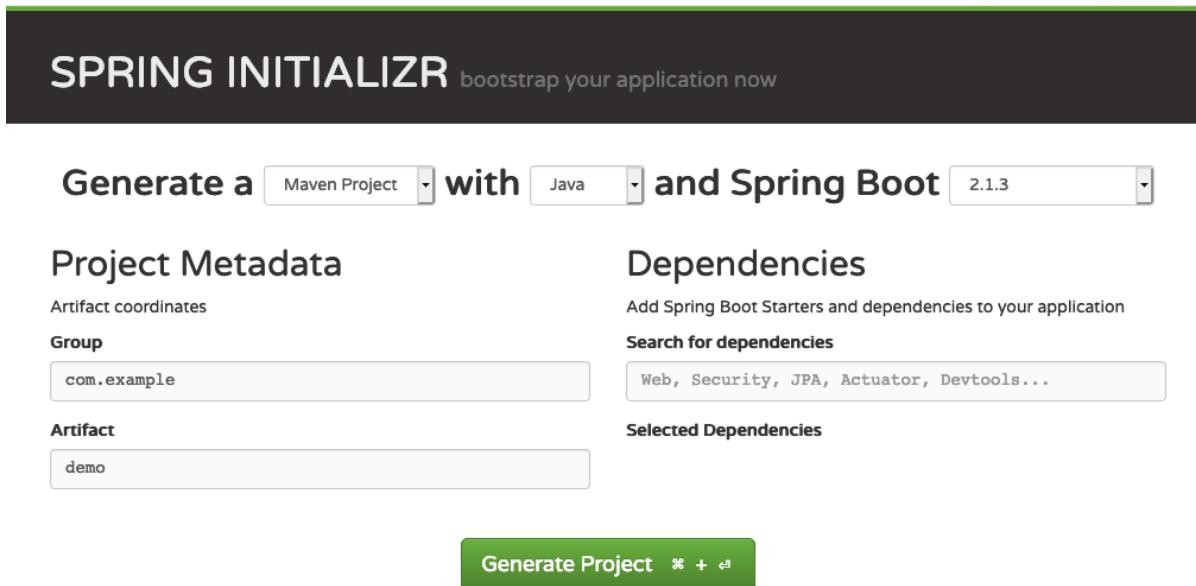
iii.i Introducción a Spring Boot (f)

- Spring boot.



iii.i Introducción a Spring Boot (g)

- Spring Initializr: Es una interface web para inicializar un proyecto con Spring Boot:



The screenshot shows the Spring Initializr interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below that, there's a main heading "Generate a with and Spring Boot .

Project Metadata

Artifact coordinates
Group: com.example
Artifact: demo

Dependencies

Add Spring Boot Starters and dependencies to your application
Search for dependencies: Web, Security, JPA, Actuator, Devtools...
Selected Dependencies

Generate Project *

Don't know what to look for? Want more options? [Switch to the full version.](#)

<https://start.spring.io/>



+

NETFLIX
OSS

iii.i Introducción a Spring Boot (h)

- **Práctica 1. Spring Initializr**
- Inicializar un proyecto Spring Boot mediante Spring Initializr.
- Generar un proyecto con las siguientes características:
 - Maven Project con Java y usando Spring Boot 2.1.3 (o la más actual, NO SNAPSHOT)
 - Group: **com.consulting.mgt.springboot**
 - Artifact: **1-Spring-Initializr**
 - Dependencias: **Web**
 - Package Name: **com.consulting.mgt.springboot.practica1**
 - Generar proyecto e importar en eclipse / STS.
- Implementar un **@RestController** simple, ejecutar el proyecto y abrir navegador <http://localhost:8080>



iii.i Introducción a Spring Boot (i)

- Spring Boot CLI.
- Herramienta de línea de comandos para crear un prototipo de proyecto basado en Spring Boot “production-ready”; es similar a Spring Initializr.
- Requiere instalación:
 - <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html#getting-started-installing-the-cli>



iii.i Introducción a Spring Boot (j)

- Spring Boot con Spring Boot CLI



Rob Winch
@rob_winch

```
@Controller
class ThisWillActuallyRun {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        "Hello World!"
    }
}
```



15:12 - 6 ago. 2013

https://twitter.com/rob_winch/status/364871658483351552?lang=es



iii.i Introducción a Spring Boot (k)

- BOM – Bill of Materials (Lista de materiales)
 - ¿Qué es un POM? (Maven)
 - Project Object Model.
 - Archivo XML que contiene información y configuración del proyecto, utilizado por Maven, para importar sus dependencias.
 - ¿Qué es un BOM?
 - Es un tipo especial de POM que es utilizado para controlar las versiones de las dependencias de un proyecto de forma centralizada.
 - Provee flexibilidad para agregar una dependencia evitando la preocupación referente a la versión compatible de la misma con el resto de dependencias.



iii.i Introducción a Spring Boot (I)

- BOM – Bill of Materials (Lista de materiales)
 - **¿Cómo utilizar un BOM en nuestro proyecto?**
 - Hacer que nuestro proyecto herede de un POM padre (BOM).
 - Un proyecto puede heredar de un solo proyecto padre, no es eficiente en definiciones de proyectos amplios y complejos (por ejemplo: Requiere Spring Cloud).
 - Importar uno o varios BOMs tal como sea requerido.



iii.i Introducción a Spring Boot (m)

- Heredar de BOM padre (a):

```
<project ... >
...
<parent>
    <groupId>org.springframework</groupId>
    <artifactId>spring-framework-bom</artifactId>
    <version>5.1.5.RELEASE</version>
    <type>pom</type>
    <scope>import</scope>
</parent>
...

```

Heredar de POM (BOM)
padre



+

NETFLIX
OSS

iii.i Introducción a Spring Boot (n)

- Heredar de BOM padre (b):

...

```
<dependencies>
```

```
    <dependency>
```

```
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-context</artifactId>
```

```
    </dependency>
```

```
    <dependency>...</dependency>
```

```
  </dependencies>
```

```
</project>
```

No requiere
especificar versión.



+

NETFLIX
OSS

iii.i Introducción a Spring Boot (ñ)

- Importación de BOM (a):

```
<project ... >
...
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
    <dependency>...</dependency>
</dependencies>
...
```

No requiere
especificar versión.



+

iii.i Introducción a Spring Boot (o)

- Importación de BOM (b):

...

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>5.1.5.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
</project>
```





+

iii.i Introducción a Spring Boot (p)

- Starters.
- Son un conjunto de descriptores de dependencias que se incluirán en el proyecto si dicho “starter” es incluido en el mismo.
- Los “starters” evitan que el desarrollador tenga que preocuparse por incluir todas las dependencias relacionadas a una funcionalidad, implementada con Spring en particular.
- Si el proyecto requiere Spring Data JPA para implementar acceso a datos, únicamente es necesario incluir la dependencia “spring-boot-starter-data-jpa” al proyecto.



iii.i Introducción a Spring Boot (q)

- Existen múltiples “starters” proveidos por la comunidad de Spring Boot listos para ser incluidos en un proyecto Java empresarial con Spring Boot:
 - <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#spring-boot-starter-tomcat>
 - spring-boot-starter
 - spring-boot-starter-activemq
 - spring-boot-starter-amqp
 - spring-boot-starter-aop
 - spring-boot-starter-data-elasticsearch
 - spring-boot-starter-data-jdbc
 - spring-boot-starter-data-jpa
 - spring-boot-starter-data-mongodb
 - spring-boot-starter-data-redis
 - spring-boot-starter-web
 - spring-boot-starter-webflux
 - spring-boot-starter-data-rest
 - spring-boot-starter-oauth2-client



iii.i Introducción a Spring Boot (r)

- Creando un jar ejecutable.
- Spring Boot genera aplicaciones autocontenidoas que incluyen todo lo necesario para poder ejecutarse, también llamadas aplicaciones “fat jar” o “uber jar” las cuales, por default, son empaquetadas como jar.
- Existen 3 modos de generar aplicaciones Spring Boot autocontenidoas:
 - Spring CLI mediante comando “**spring jar <nombre-de-jar> <archivos>**”
 - Maven mediante **spring-boot-maven-plugin** starter y comando: “**mvn package**”
 - Gradle mediante “**spring-boot**” plugin y comando “**gradle build**” (fuera de alcance).

iii.i Introducción a Spring Boot (s)

- Ventajas de crear una aplicación autocontenido:
 - No requiere más dependencias para ejecutarse.
 - Se ejecuta mediante “java –jar”.
 - Se ejecuta donde sea que contenga una JVM.
 - No requiere servidor de aplicaciones.
 - Facilita el diseño de arquitecturas distribuidas.
 - Ideal para arquitecturas de Microservicios.
 - Ideal para empaquetar y desplegar en la nube. (PaaS y/o IaaS).





+

NETFLIX
OSS

iii.i Introducción a Spring Boot (t)

- **Práctica 2. Creando un jar ejecutable**
 - Inicializar un proyecto Spring Boot mediante Spring CLI.
 - Instala Spring CLI.
 - Crea el fichero: **{tu-workspace}/2-Spring-CLI/AlumnosController.java**
 - Implementar el controlador AlumnosController, ejecuta el comando:
“spring run AlumnosController.java” y,
abrir navegador: <http://localhost:8080>
 - Ingresar a la ruta: **{tu-workspace}/2-Spring-CLI**
 - Ejecuta los comandos:
“spring jar miApp.jar AlumnosController.java” después,
“java -jar miApp.jar” y,
abrir navegador: <http://localhost:8080>



+

NETFLIX
OSS

iii.i Introducción a Spring Boot (u)

- Configuración Spring Boot.
- Spring Boot habilita auto- configuración mediante el escaneo automático de dependencias que han sido agregadas al proyecto, es decir, genera los beans necesarios para el proyecto en base a las dependencias existentes en el classpath.
 - Ejemplo: Si la librería (dependencia) H2 se encuentra en el classpath del proyecto y no se ha definido ninguna configuración de beans relacionada a un DataSource, Spring Boot **auto- configurará** una base de datos en memoria utilizando el motor H2.



iii.i Introducción a Spring Boot (v)

- Configuración Spring Boot.
- La auto-configuración que aplica Spring Boot es no es invasiva, es decir, es posible definir una configuración propia del proyecto simplemente definiendo los beans correspondientes:
 - Ejemplo: Si se define un bean DataSource, la **auto-configuración** de Spring Boot para definir un DataSource en memoria (embebida) no surtirá efecto.



+

NETFLIX
OSS

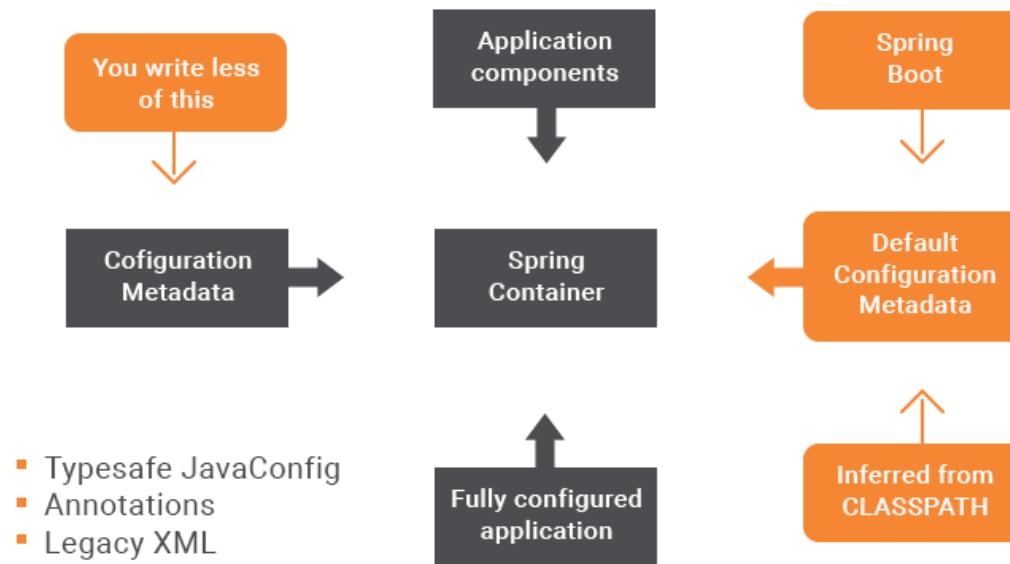
iii.i Introducción a Spring Boot (w)

- Configuración Spring Boot.
- Para habilitar la **auto-configuración**, de Spring Boot, únicamente es necesario agregar las anotaciones **@EnableAutoConfiguration** o **@SpringBootApplication** a la clase de configuración primaria del proyecto (definir **@EnableAutoConfiguration** o **@SpringBootApplication** una sola vez).

iii.i Introducción a Spring Boot (x)

- Configuración Spring Boot.

Spring Boot Simplifies Configuration





+

NETFLIX
OSS

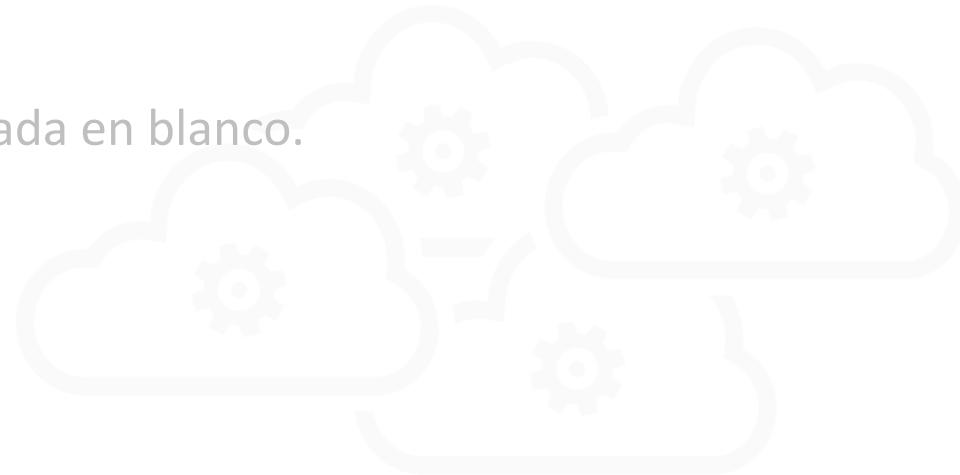
Resumen de la lección

iii.i Introducción a Spring Boot

- Comprendemos las diferencias entre Spring y Spring Boot.
- Aprendemos a utilizar las herramientas Spring Initializr y Spring CLI para crear proyectos Spring Boot desde cero.
- Comprendemos que el objetivo de Spring Boot es inicializar un aplicativo Java Empresarial sin la complejidad de su configuración.
- Revisamos que es el BOM y los Starters.
- Creamos un aplicativo Spring Boot autocontenido en un jar, mediante la aplicación de las herramientas Spring Initializr y Spring CLI.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles
- iii.iv Spring MVC
- iii.v Spring Data JPA
- iii.vi Spring Data REST
- iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

iii.ii Configuración de propiedades en Spring Boot



+

Objetivos de la lección

iii.ii Configuración de propiedades en Spring Boot

- Comprender como se configura a nivel propiedades, una aplicación Spring Boot.
- Analizar los diferentes mecanismos de configuración mediante archivo de propiedades y YAML.
- Exponer configuración externalizada.



+

iii.ii Configuración de propiedades en Spring Boot (a)

- application.properties y configuración YAML.
- Spring Boot carga sus propiedades de múltiples formas sin embargo, el archivo “**application.properties**” es el archivo de propiedades por default de Spring Boot.
- Spring Boot carga las propiedades, del archivo “**application.properties**” y las agrega al objeto **Environment** de Spring.



+

iii.ii Configuración de propiedades en Spring Boot (b)

- application.properties y configuración YAML.
- La ubicación del archivo “**application.properties**” se define por la siguiente precedencia:
 - Subdirectorio /config del directorio actual.
 - El directorio actual
 - En el paquete /config del classpath
 - En el directorio raíz del classpath
- Las propiedades definidas en ubicaciones superiores sobre-escriben aquellas definidas en ubicaciones inferiores.



+

iii.ii Configuración de propiedades en Spring Boot (c)

- Acceso a propiedades por línea de comando.
- Por default Spring Boot convierte cualquier argumento de línea de comandos (iniciando con --) como una propiedad y la agrega al objeto Environment: Ejemplo: **--server.port=9090**
- Es posible cambiar el nombre del archivo “**application.properties**” utilizando la propiedad **spring.config.name** pasada como parámetro al ejecutar la aplicación Spring Boot.
 - **java -jar myproject.jar --spring.config.name=myproject**
 - El comando anterior buscará un archivo de propiedades llamado “**myproject.properties**” en el orden de precedencia por default.



iii.ii Configuración de propiedades en Spring Boot (d)

- application.properties
- Spring Boot define un conjunto de propiedades básicas para habilitar auto- configuración:

Spring Config

- spring.config.location

- spring.config.name

Email

- spring.mail.default-encoding

- spring.mail.host

- spring.mail.password

Profiles

- spring.profiles.active

- spring.profiles.include

Embedded Server Configuration

- server.port

- server.servlet.context-path

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>



+

NETFLIX
OSS

iii.ii Configuración de propiedades en Spring Boot (e)

- Configuración YAML
- Es posible utilizar el formato YAML en lugar de utilizar “**properties**” siempre y cuando la dependencia SnakeYAML se encuentre en el classpath.
 - **spring-boot-starter** agrega la dependencia SnakeYAML.
- YAML es un subconjunto de JSON y es preferido al especificar configuración de propiedades de forma jerárquica (opinonado).
- YAML es más “humanamente leíble” sin embargo, es opinionado.



iii.ii Configuración de propiedades en Spring Boot (f)

- Configuración YAML
- Ejemplo de application.yml:

```
myapp:  
  name: Ivan  
  age: 31
```
- Como archivo application.properties:

```
myapp.name=ivan  
myapp.age=31
```
- Propiedades definidas por archivos “**properties**” tiene mayor precedencia a las definidas por archivos YAML.



+

iii.ii Configuración de propiedades en Spring Boot (g)

- Spring Boot permite externalizar la configuración del aplicativo de modo que sea posible ejecutar el mismo código en distintos ambientes:
 - Configuración de conexión a Base de Datos por diferentes ambientes.
 - Configuración de servidor SFTP por diferentes ambientes, etc.
- La configuración puede definirse de diferentes modos:
 - Archivos de propiedades.
 - Archivos YAML.
 - Variables de entorno.
 - Argumentos de línea de comandos.

iii.ii Configuración de propiedades en Spring Boot (h)

- Los valores de las propiedades pueden ser inyectadas directamente sobre las propiedades de los Beans mediante la anotación **@Value** o accediendo a ellas a través del objeto **Environment** de Spring (`env.getProperty(key)`).



iii.ii Configuración de propiedades en Spring Boot (i)

- Precedencia en el que se busca la configuración externalizada (a):
 1. Configuración global Devtools: En el directorio raíz del sistema (`~/.spring-boot-devtools.properties`), sólo si DevTools está activo.
 2. Anotación `@TestPropertySource` sobre clases de Test.
 3. Atributos “`properties`” sobre tests, disponibles mediante `@SpringBootTest`.
 4. Argumentos de línea de comandos (`--argument=value`). ✓
 5. Propiedades desde variable de entorno `SPRING_APPLICATION_JSON` (JSON embebido en línea como variable de entorno o propiedad del sistema). ✓
 6. Parámetros de inicialización de configuración de Servlet (`ServletConfig init parameters`).
 7. Parámetros de inicialización de contexto de Servlet (`ServletContext init parameters`).



+

iii.ii Configuración de propiedades en Spring Boot (j)

- Precedencia en el que se busca la configuración externalizada (b):
 8. Atributos JNDI desde: **java:comp/env**.
 9. Propiedades Java del sistema: **System.getProperties()**.
 10. Variables de entorno del Sistema Operativo (OS environment variables).
 11. Propiedades Random (**random.***).
 12. application.properties específicas de Perfil desde fuera del empaquetado del aplicativo/jar (**application-{profile}.properties incluye variantes YAML**).
 13. application.properties específicas de Perfil desde dentro del empaquetado del aplicativo/jar (**application-{profile}.properties incluye variantes YAML**).



+

iii.ii Configuración de propiedades en Spring Boot (k)

- Precedencia en el que se busca la configuración externalizada (c):
 - ✓ 14. application.properties desde fuera del empaquetado del aplicativo/jar (**application.properties incluye variantes YAML**).
 - ✓ 15. application.properties desde dentro del empaquetado del aplicativo/jar (**application.properties incluye variantes YAML**).
 - 16. Anotaciones **@PropertySource** sobre clases de configuración **@Configuration**.
 - 17. Propiedades default especificadas por **SpringApplication.setDefaultProperties**.

- Nota: No se revisarán todas las formas de precedencia de configuración externalizada, sólo las más comunes.

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html#boot-features-external-config>



+

NETFLIX
OSS

iii.ii Configuración de propiedades en Spring Boot (I)

- **Práctica 3. Configuración Externalizada**
- Instala lombok en tu IDE.
- Ingresar a la ruta: **{tu-workspace}/3-Spring-Configuracion-Externalizada**
- Ejecutar el proyecto ejecutando la clase principal y abrir navegador en:
<http://localhost:8080>
- Implementar diferentes métodos de configuración externalizada de propiedades:
 - controller.message
 - server.port



+

iii.ii Configuración de propiedades en Spring Boot (m)

- **Práctica 3. Configuración Externalizada**
 1. Analizar la clase principal Application.java, ejecutar el proyecto.
 2. Probar <http://localhost:8080>
 3. Crear archivo “**application.yml**” y definir propiedades **server.port: 8081** y **controller.message: Mi mensaje desde application.yml**. Ejecutar el proyecto.
 4. Probar <http://localhost:8081> (nótese que el puerto 8080 ya no responde).
 5. Definir propiedades, en archivo “**application.properties**”, **server.port=8085** y **controller.message=Mi mensaje desde application.properties**. Ejecutar el proyecto.
 6. Probar <http://localhost:8081> y <http://localhost:8085>



+

NETFLIX
OSS

iii.ii Configuración de propiedades en Spring Boot (n)

- **Práctica 3. Configuración Externalizada**
- 7. Definir la variable de sistema (variable de entorno) **SERVER_PORT=9090** y **CONTROLLER_MESSAGE='Mi mensaje desde variable del sistema'** dependiendo del sistema operativo host.
 - Ejemplo, en Mac:
 - ✓ Editar `~/.bashrc`, agregar variables del sistema:
 - ✓ `export SERVER_PORT=9090`
 - ✓ `export CONTROLLER_MESSAGE='Mi mensaje desde variable del sistema'`
 - ✓ Recargar el archivo `~/.bash_profile` (source `~/.bash_profile`)
- 8. Compilar proyecto mediante **mvn clean package** y ejecutar mediante comando:
java -jar target/3-Spring-Configuracion-Externalizada-0.0.1-SNAPSHOT.jar
- 9. Probar <http://localhost:9090>



+

iii.ii Configuración de propiedades en Spring Boot (ñ)

- **Práctica 3. Configuración Externalizada**
- 7. Definir la variable de sistema (variable de entorno) **SERVER_PORT=9090** y **CONTROLLER_MESSAGE='Mi mensaje desde variable del sistema'** dependiendo del sistema operativo host.
 - Ejemplo, en Windows:
 - ✓ Editar variables del sistema:
 - ✓ Variable: SERVER_PORT Valor: 9090
 - ✓ Variable: CONTROLLER_MESSAGE Valor: Mi mensaje desde variable del sistema.
- 8. Compilar proyecto mediante **mvn clean package** y ejecutar mediante comando:
java -jar target/3-Spring-Configuracion-Externalizada-0.0.1-SNAPSHOT.jar
- 9. Probar <http://localhost:9090>



iii.ii Configuración de propiedades en Spring Boot (o)

- **Práctica 3. Configuración Externalizada**

10. Definir la variable de sistema (variable de entorno)

SPRING_APPLICATION_JSON='{"controller":{"message":"Mi mensaje desde SPRING_APPLICATION_JSON"}, "server":{"port":9099}}' dependiendo del sistema operativo host.

11. Compilar proyecto mediante **mvn clean package** y ejecutar mediante comando:

java -jar target/3-Spring-Configuracion-Externalizada-0.0.1-SNAPSHOT.jar

12. Probar <http://localhost:9099>



+

NETFLIX
OSS

iii.ii Configuración de propiedades en Spring Boot (p)

- Práctica 3. Configuración Externalizada

13. Por último, para comprobar la precedencia de la configuración externalizada, compilar proyecto mediante **mvn clean package**.
14. Ejecutar el proyecto mediante comando:
`java -jar target/3-Spring-Configuracion-Externalizada-0.0.1-SNAPSHOT.jar
--controller.message="Mi mensaje desde linea de comando."
--server.port=9001`
15. Probar <http://localhost:9001>



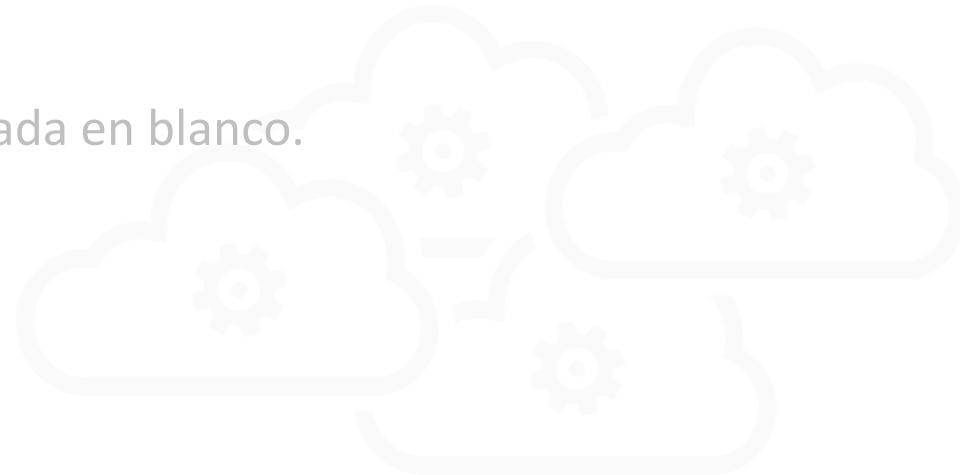
Resumen de la lección

iii.ii Configuración de propiedades en Spring Boot

- Comprendemos las diferencias entre configuración mediante archivos “properties” y YAML.
- Revisamos la precedencia de los archivos de configuración para la implementación de configuración externalizada.
- Analizamos algunas propiedades de Spring Boot que facilitan la configuración de la aplicación.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles**
- iii.iv Spring MVC
- iii.v Spring Data JPA
- iii.vi Spring Data REST
- iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

iii.iii Perfiles



Microservices



Objetivos de la lección

iii.iii Perfiles

- Comprender para que se utilizan los perfiles o “profiles” en Spring.
- Aprender como activar perfiles dependiendo el contexto de ejecución del aplicativo.



+

NETFLIX
OSS

iii.iii Perfiles (a)

- Los perfiles o “**profiles**” es un mecanismo que permite al contenedor de beans de Spring habilitar el registro de diferentes beans de un mismo tipo para diferentes ambientes de ejecución, es decir, los perfiles permiten agrupar beans de una misma naturaleza o, beans de un mismo ambiente o entorno de ejecución.
- Un perfil es un identificador que permite agrupar definiciones de beans para ser registrados en el contenedor sólo si dicho perfil se encuentra activo.



+

NETFLIX
OSS

iii.iii Perfiles (b)

- Ejemplo: Es posible que se defina un bean de tipo DataSource con una base de datos en memoria H2 para un perfil de desarrollo o “**development**” y, definir, otro bean de tipo DataSource el cual adquiera su referencia a través de JNDI para un perfil de producción o “**production**”.



+

NETFLIX
OSS

iii.iii Perfiles (c)

- Mediante la anotación **@Profile** es posible indicar para qué perfil corresponde un bean componente.
- El contenedor de beans de Spring sólo levantará los beans registrados en el perfil default (sin perfil) y aquellos registrados en los perfiles activos.
- Se pueden definir expresiones complejas a evaluar por los perfiles a través de los operadores lógicos “!” (not), “&” (and) y “|” (or).



+

iii.iii Perfiles (d)

- Es posible agrupar operadores lógicos únicamente utilizando paréntesis.
 - La expresión = “**production & us-east | eu-central**” no es válida.
 - La expresión = “**production & (us-east | eu-central)**” es válida.
- Es posible utilizar la anotación @Profile como una meta-anotación para crear anotaciones personalizadas y “type-safe”.

```
@Retention(RetentionPolicy.RUNTIME)
@Profile("production")
public @interface Production { }
```

@Production

@Profile(“production”)



+

NETFLIX
OSS

iii.iii Perfiles (e)

- @Profile

```
@Configuration
@Profile("dev")
public class StandaloneDataConfig {

    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder().setType(EmbeddedDatabaseType.HSQL)
            .addScript("classpath:com/bank/config/sql/schema.sql")
            .addScript("classpath:com/bank/config/sql/test-data.sql") .build();
    }
}
```



+

NETFLIX
OSS

iii.iii Perfiles (f)

- @Profile

```
@Configuration
```

```
@Production
```

```
public class JndiDataConfig {
```

```
    @Bean(destroyMethod="close")
```

```
    public DataSource dataSource() throws Exception {
```

```
        Context ctx = new InitialContext();
```

```
        return (DataSource) ctx.lookup("java:comp/env/jdbc/datasource");
```

```
}
```

```
}
```



+

NETFLIX
OSS

iii.iii Perfiles (g)

- Se utiliza la propiedad **spring.profiles.active** para habilitar un perfil o perfiles separados por coma (separados por coma implica “or”).
- También es posible, y más recomendable, pasar el perfil por línea de comando (**--spring.profiles.active**) o mediante variable de entorno del sistema.
- Definir la misma propiedad en múltiples archivos de configuración, ya sea por medio de **application.properties** o pasando el valor por **línea de comandos**, sigue la misma precedencia de como se cargan las propiedades.



+

NETFLIX
OSS

iii.iii Perfiles (h)

- Archivos de configuración específicos por Perfil.
- Spring Boot permite la configuración de propiedades a través de archivos “**properties**” o archivos YAML.
- Estos archivos de configuración pueden tener configuración particular para un perfil específico.
- La convención para definir archivos “**properties**” o YAML específicos por perfil es **application-{profile}.properties** o **application-{profile}.yml**, dichos archivos se buscarán con la misma precedencia referente a la configuración externalizada.



+

iii.iii Perfiles (i)

- Archivos de configuración YAML multi-perfil.
- Es posible especificar multiple configuración de perfiles en un único archivo YAML. Para ello se utiliza la propiedad **spring.profiles**.

```
server:  
  address: 192.168.1.100 } Configuración  
                        default  
---  
spring:  
  profiles: development } Configuración  
  server:  
    address: 127.0.0.1   perfil “development”  
---  
spring:  
  profiles: production & eu-central } Configuración  
  server:  
    address: 192.168.1.120  perfil “production & eu-central”
```

iii.iii Perfiles (j)

- Archivos de configuración YAML multi-perfil.
- Al utilizar configuración específica por perfil mediante YAML, sólo utilizar uno de los dos métodos expuestos para definir propiedades multi-perfil.
- En la práctica, Spring Boot sugiere utilizar configuración multi-perfil en un solo archivo YAML, debido a que es “humanamente leíble”.
- En la experiencia, se sugiere utilizar configuración mediante múltiples archivos específicos por perfil mediante “properties” (`application-{profile}.properties`).



iii.iii Perfiles (k)

- **Práctica 4. Perfiles y configuración multi-perfil**
- Ingresar a la ruta: **{tu-workspace}/4-Spring-Configuracion-Multi-Perfil**
- Analizar la clase ConnectionDataBase. ¿Qué bean DummyDataSource se va a inyectar? ¿valor de connectionURL se va a inyectar?
- Define la propiedad: **myapp.connection.url** con los valores requeridos para los perfiles dev, qa, staging y production.
- Define la clase ProfilesConfig y describe los 4 beans requeridos, de tipo DummyDataSource, para los ambientes dev, qa, staging y production.
- Poner en práctica los distintos modos de activación de perfiles.



+

NETFLIX
OSS

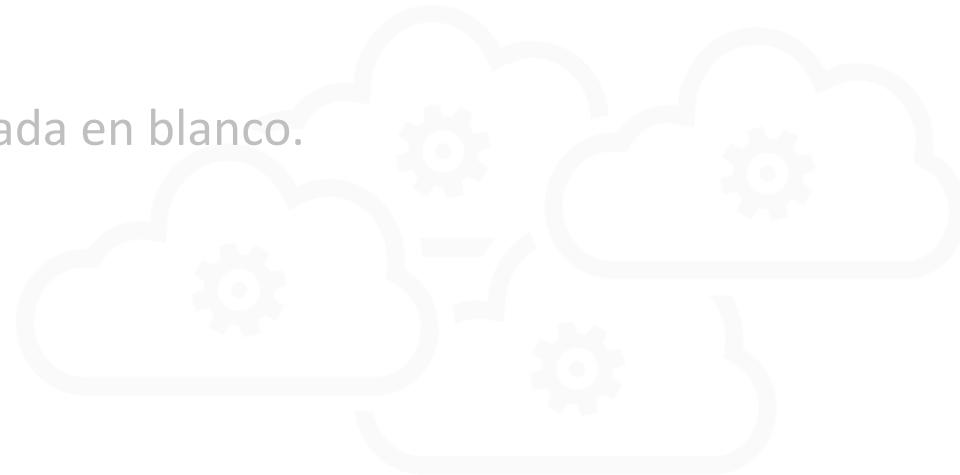
Resumen de la lección

iii.iii Perfiles

- Comprendemos cómo implementar perfiles para el despliegue de aplicaciones Spring Boot en diferentes ambientes.
- Aprendimos como utilizar la anotación **@Profile** para definir beans de diferentes perfiles.
- Revisamos como activar perfiles.
- Verificamos como asociar perfiles a archivos de propiedades específicos mediante “**properties**” y YAML.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles
- iii.iv **Spring MVC**
- iii.v Spring Data JPA
- iii.vi Spring Data REST
- iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

iii.iv Spring MVC





+

Objetivos de la lección

iii.iv Spring MVC

- Conocer la auto-configuración de Spring MVC mediante Spring Boot.
- Revisar a grandes rasgos las características que habilita Spring Boot a las aplicaciones Spring MVC.
- Conocer la convención sobre configuración principal de aplicaciones Spring MVC auto-configuradas con Spring Boot.
- Comprender como se implementa la negociación de contenido en aplicaciones Spring MVC con Spring Boot.
- Comprender cómo se integra Spring HATEOAS y cómo se habilita CORS a aplicaciones Spring MVC con Spring Boot.



+

NETFLIX
OSS

iii.iv Spring MVC (a)

- Spring Boot habilita el desarrollo rápido de aplicaciones web mediante el módulo "**starter**" **spring-boot-starter-web**.
- Spring Boot permite crear aplicaciones web auto-contenidas con un servidor embebido siendo Tomcat, Jetty, Netty o Undertow, permitiendo empaquetar la aplicación en un jar y ejecutarlo en cualquier JVM, no necesita servidor web o servidor de aplicaciones.
- No es necesario configurar algún bean en particular para iniciar el desarrollo de una aplicación web, Spring Boot auto-configura los beans necesarios por default.



+

NETFLIX
OSS

iii.iv Spring MVC (b)

- La auto-configuración de Spring Boot mediante el módulo “**starter**” **spring-boot-starter-web** incluye:
 - Inclusión de los beans **ContentNegotiatingViewResolver** y **BeanNameViewResolver**.
 - Soporte para servir contenido estático, así como recursos estáticos desde WebJars.
 - Registro automático de beans **Converter**, **GenericConverter** y **Formatter**.
 - Soporte de **HttpMessageConverters**.
 - Registro automático de **MessageCodesResolver**.
 - Soporte estático de templates.
 - Soporte de Favicon personalizado.
 - Configuración automática mediante bean **ConfigurableWebBindingInitializer**.



iii.iv Spring MVC (c)

- Mediante la inclusión del módulo “starter” **spring-boot-starter-web** es posible utilizar las anotaciones comunes al framework Spring Web MVC de forma tradicional y desarrollar una aplicación web.
 - @Controller
 - @RestController
 - @RequestMapping
 - @GetMapping
 - @PostMapping
 - @PutMapping
 - @DeleteMapping
 - @PatchMapping
 - @ResponseStatus
 - @ResponseBody
 - @RequestBody
 - @PathVariable
 - @RequestParam
 - @RequestHeader
 - @CookieValue
 - @RequestPart
 - @ModelAttribute
 - @SessionAttributes
 - @ControllerAdvice
 - @ExceptionHandler



iii.iv Spring MVC (d)

- Por default Spring Boot sirve contenido estático desde las ubicaciones **/static, /public, /resource, /META-INF/resources o /templates** del classpath o de la raíz del ServletContext.
- No se recomienda el uso de la ubicación standard **/src/main/webapp** cuando se trabaja con Spring Boot empaquetado como jar. La ubicación **/src/main/webapp** únicamente es considerada cuando la aplicación es empaquetada como war.
- Spring Boot por default muestra el archivo estático **index.html** como “welcome page”, el cual busca desde las ubicaciones mencionadas.



iii.iv Spring MVC (e)

- Content negotiation
- Spring Boot, como buena práctica, deshabilita la negociación del contenido de URL mediante sufijos y sugiere la utilización correcta del Header “Accept”.
 - La petición **GET /myproject/person.json** no se mapeará al handler mapping **@GetMapping("/myproject/person")**
- Por default Spring MVC, mapeaba la negociación del contenido mediante sufijo (file extension) en automático, dada la imposibilidad de los navegadores de manipular correctamente el Header “Accept”.



iii.iv Spring MVC (f)

- Content negotiation
- De no ser posible manipular correctamente el Header “Accept” por parte del cliente para negociar el contenido de respuesta, Spring Boot sugiere utilizar “query parameters” para la negociación del contenido.
 - La petición **GET /myproject/person?format=json** se mapeará al handler mapping **@GetMapping("/myproject/person")** si se habilita la propiedad:
spring.mvc.contentnegotiation.favor-parameter=true
 - Es posible cambiar el parámetro de negociación de contenido mediante la propiedad:
spring.mvc.contentnegotiation.parameter-name=myparam



iii.iv Spring MVC (g)

- Templates
- Spring Boot habilita el uso y auto-configuración para la integración de templates para Spring MVC tales como: FreeMarker, Groovy, Thymeleaf y Mustache.
- JSP debe ser evitado debido a que se tiene prueba de limitaciones técnicas cuando se trabaja con contenedores web embebidos.
 - Con Jetty y Tomcat, JSP funciona con empaquetado war y ejecutando la aplicación mediante comando java -jar.
 - Undertow no soporta JSPs.
 - La creación de un pagina de error customizada “error.jsp” no sobre-escribe la vista por default de manejo de errores (/error).



+

NETFLIX
OSS

iii.iv Spring MVC (h)

- HATEOAS
- Spring Boot utiliza auto-configuración mediante la inclusión del módulo “starter” **spring-boot-starter-hateoas** sin mayor configuración, no necesita ser habilitado mediante `@EnableHypermediaSupport`.
- Habilita la auto-configuración del bean **ObjectMapper** de forma automática mediante propiedades **spring.jackson.***.
- Es posible crear la definición manual del bean **Jackson2ObjectMapperBuilder** para crear el bean **ObjectMapper**.



iii.iv Spring MVC (i)

- Soporte CORS
- Spring MVC permite el Cross-origin Resource Sharing o CORS mediante la anotación **@CrossOrigin**.
- La anotación **@CrossOrigin** puede utilizarse a nivel de método (handler methods) o a nivel de clase **@Controller** o **@RestController**.
- Spring Boot no requiere configuración adicional para habilitar CORS.
- **@CrossOrigin** habilita: todos los orígenes, todos los headers y todos los métodos HTTP que tenga mapeados el controlador.



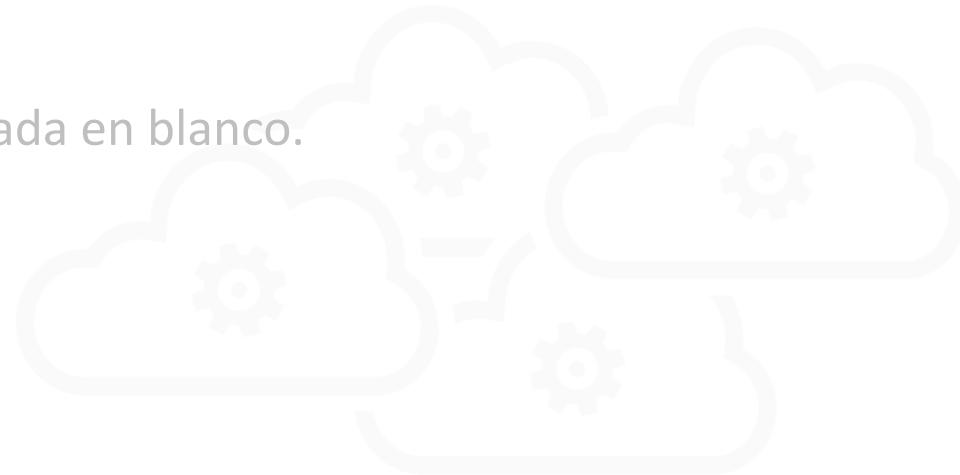
Resumen de la lección

iii.iv Spring MVC

- Comprendemos como habilitar auto-configuration de:
 - Spring MVC
 - Spring HATEOAS
- Comprendemos el mecanismo de negociación de contenido.
- Comprendemos cuales son las rutas pre-configuradas de Spring Boot para Spring MVC.
- Analizamos las debilidades de JSPs en aplicaciones Spring MVC auto-contenidas con Spring Boot.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles
- iii.iv Spring MVC
- iii.v Spring Data JPA
- iii.vi Spring Data REST
- iii.vii Spring Boot Actuator



Microservices



iii.v Spring Data JPA



Microservices



Objetivos de la lección

iii.v Spring Data JPA

- Conocer la auto-configuración de Spring Data JPA mediante Spring Boot.
- Revisar a grandes rasgos las características que habilita Spring Boot a las aplicaciones Spring Data JPA.
- Conocer la convención sobre configuración principal de aplicaciones Spring Data JPA auto-configurationadas con Spring Boot.
- Comprender como implementar Repositorios mediante Spring Data JPA.
- Comprender como implementar “**query methods**” en Spring Data JPA.



iii.v Spring Data JPA (a)

- Spring Framework provee extensivo soporte para trabajar con bases de datos SQL (y no SQL) mediante el acceso a través de JdbcTemplate (spring-jdbc) o integración completa mediante un ORM tal como Hibernate (spring-orm y hibernate).
- Spring Data es un proyecto construido sobre Spring Framework que provee un nivel más de abstracción sobre repositorios, el cual nos permite definir repositorios a partir de interfaces.
- Con Spring Data no es necesario implementar repositorios (o DAOs) una y otra vez conforme los requerimientos de las aplicaciones.



+

NETFLIX
OSS

iii.v Spring Data JPA (b)

- Spring Data JPA trabaja sobre bases de datos relacionales o SQL, mediante el estándar JPA e implementación a través de Hibernate.
- Las aplicaciones Spring Data JPA auto-configuradas con Spring Boot 2.x utilizan Hibernate 5 como implementación JPA 2.0.
- Hibernate 3 y 4 han sido depreciadas en Spring Framework 5.0 y Spring Boot 2.x utiliza como código base Spring Framework 5.0.



iii.v Spring Data JPA (c)

- Spring Boot habilita el desarrollo rápido de aplicaciones con Spring Data JPA mediante el módulo "**starter**" **spring-boot-starter-data-jpa** o **spring-boot-starter-jdbc** para el uso simple de Spring JDBC.
- Por default, Spring Boot habilita soporte para la creación de base de datos embebida (en memoria), si ningún bean de tipo **DataSource** ha sido configurado.
- Los motores de base de datos soportadas para implementar "**in-memory databases**" son: **HSQL**, **H2** y **Derby**. Únicamente es necesario agregar la dependencia correspondiente, a la base de datos requerida, en el classpath.

iii.v Spring Data JPA (d)

- Spring Boot permite definir al desarrollador el DataSource requerido para el aplicativo y tomar todo el control referente al mismo.
- Definir nuestro propio bean **DataSource**, evitará la auto-configuración de Spring Data JPA mediante Spring Boot.



+

NETFLIX
OSS

iii.v Spring Data JPA (e)

- Conectividad a base de datos productiva.
- Spring Boot 2.x confía en la implementación de **DataSource** mediante “**pool**” de conexiones, lo cual habilita auto-configuración de **DataSource** productivos mediante los siguientes criterios:
 - Spring Boot seleccionará, en primer lugar, el driver **HikariCP** para crear pool de conexiones productivo, si la dependencia se encuentra en el classpath.
 - De no encontrarse **HikariCP**, utilizará el **JDBC Connection Pool** de **Tomcat**, si la dependencia se encuentra en el classpath.
 - Si no se encuentra **HikariCP** ni **Tomcat JDBC Connection Pool**, Spring Boot utilizará **Commons DBCP2**, para crear pool de conexiones, si se encuentra en el classpath.
 - No existen otras implementaciones de pool de conexiones que Spring Boot auto-configure.



+

NETFLIX
OSS

iii.v Spring Data JPA (f)

- Conectividad a base de datos productiva.
- Si se utilizan los módulos “starter” **spring-boot-starter-jdbc** o **spring-boot-starter-data-jpa**, la dependencia **HikariCP** es agregada automáticamente.
- Es posible saltar el algoritmo de selección de **DataSource** por default especificando la propiedad **spring.datasource.type**, debido a que si la aplicación se ejecuta en un contenedor **Tomcat**, el tipo de **DataSource** (con pool de conexiones) será **tomcat-jdbc** por default.
- Es posible configurar otros **DataSource** manualmente.



+

NETFLIX
OSS

iii.v Spring Data JPA (g)

- La auto-configuration del **DataSource** se da mediante configuración externalizada mediante las propiedades **spring.datasource.*** las cuales pueden definirse en el application.properties.
 - **spring.datasource.url=jdbc:mysql://localhost/test**
 - **spring.datasource.username=dbuser**
 - **spring.datasource.password=dbpass**
 - **spring.datasource.driverclass-name=com.mysql.jdbc.Driver**
- Al menos es necesario especificar la URL de conexión, de no definirse, Spring Boot tratará de auto-configurar una base de datos en memoria.
- Definir la propiedad **driverclass-name** es opcional.



iii.v Spring Data JPA (h)

- Spring Boot provee auto-configuration sobre **HikariCP**, **Tomcat JDBC Connection Pool** y **Commons DBCP2** mediante las propiedades **spring.datasource.hikari.***, **spring.datasource.tomcat.*** y **spring.datasource.dbcp2.*** correspondientemente.
 - Ejemplo:
 - # Number of ms to wait before throwing an exception if no connection is available.
 - **spring.datasource.tomcat.max-wait=10000**
 - # Maximum number of active connections that can be allocated from this pool at the same time.
 - **spring.datasource.tomcat.max-active=50**
 - # Validate the connection before borrowing it from the pool.
 - **spring.datasource.tomcat.test-on-borrow=true**



iii.v Spring Data JPA (i)

- **spring-boot-starter-data-jpa** provee las siguientes dependencias para trabajar con Spring Data JPA:
 - Hibernate: La implementación JPA más popular de la industria.
 - Spring Data JPA: Proyecto que implementa Repositorios y "querys" personalizados, basado en interfaces.
 - Spring ORM: Clases core adaptadoras para la integración de frameworks ORM.



+

NETFLIX
OSS

iii.v Spring Data JPA (j)

- Configuración Spring Data JPA con Spring Boot.
- Spring Boot no requiere del archivo "**persistence.xml**" para especificar las entidades JPA, en automático, Spring Boot escaneará, las entidades anotadas con **@Entity**, **@Embeddable** o **@MappedSuperclass**, desde el paquete base de la aplicación anotada con **@EnableAutoConfiguration** o **@SpringBootApplication**.
- Es posible mejorar el performance de la aplicación al hacer el escaneo de entidades mediante **@EntityScan**. Así mismo, si las entidades no están definidas bajo un sub-paquete del paquete base, es necesario especificar dicho paquete mediante **@EntityScan**.



+

NETFLIX
OSS

iii.v Spring Data JPA (k)

- Definición de Spring Data JPA Repositories.
- Es posible definir un Repositorio genérico, agnóstico de la tecnología de persistencia mediante definir una interfaz que herede de la interfaz **Repository** o una de sus sub-interfaces.
- La interface **Repository** es una “**marker-interface**” y no define ningún método.
- Si se define un repositorio heredando de la interface **Repository**, es necesario definir los “**query methods**” a implementar por Spring Data JPA al vuelo.



+

NETFLIX
OSS

iii.v Spring Data JPA (I)

- Definición de Spring Data JPA Repositories.

@Entity

```
public class City implements Serializable {  
    @Id  
    @GeneratedValue  
    private Long id;  
  
    @Column(nullable = false)  
    private String name;  
  
    @Column(nullable = false)  
    private String state;  
  
    ...  
}
```

```
public interface CityRepository extends Repository<City, Long> {  
    Page<City> findAll(Pageable pageable);  
  
    City findByNameAndStateAllIgnoringCase(  
        String name, String state);  
}
```



+

NETFLIX
OSS

iii.v Spring Data JPA (m)

- Definición de Spring Data JPA Repositories.
- Spring Data JPA, provee interfaces **Repository** específicas de la tecnología de persistencia a utilizar, tal como **JpaRepository** y **MongoRepository**.
- Dichas interfaces heredan de **CrudRepository** o **PagingAndSortingRepository**, las cuales ofrecen capacidades superiores tales como implementación de operaciones CRUD o paginación de resultados “**out-of-the-box**”.



+

NETFLIX
OSS

iii.v Spring Data JPA (n)

- Configurando propiedades JPA.
- Por default, Spring Boot crea bases de datos únicamente si, la base de datos, es en memoria (H2, HSQL o Derby).
- Para crear y eliminar una base de datos mediante hibernate es posible especificarlo mediante la propiedad: **spring.jpa.hibernate.ddl-auto=create-drop**
- Internamente Hibernate tiene una propiedad para crear y eliminar la base de datos **hibernate.hbm2ddl.auto**, sin embargo, para utilizarla



+

iii.v Spring Data JPA (ñ)

- Configurando propiedades JPA.
- Internamente Hibernate tiene una propiedad para crear y eliminar la base de datos **hibernate.hbm2ddl.auto**, sin embargo, para utilizar propiedades específicas de Hibernate es necesario hacerlo mediante el prefijo **spring.jpa.properties.***, ejemplo:
spring.jpa.properties.hibernate.hbm2ddl.auto
- Se recomienda siempre utilizar la propiedad **spring.jpa.hibernate.ddl-auto** para configurar cómo se creará la base de datos debido a que, dicha propiedad, tiene distintos valores default dependiendo de las condiciones en la ejecución de la aplicación.



+

NETFLIX
OSS

iii.v Spring Data JPA (o)

- Configurando propiedades JPA.
- Si la base de datos de la aplciación es embebida, el valor de la propiedad **spring.jpa.hibernate.ddl-auto** será “**create-drop**”, para todos los demás caso es “**none**”.
- Atención de no eliminar accidentalmente la base de datos en producción.
- Para más opciones de auto-configuration de Spring Data JPA mediante Spring Boot:

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>



+

iii.v Spring Data JPA (p)

- Configurando el patrón “Open EntityManager in View”.
- Si se está desarrollando una aplicación web, por default, Spring Boot registra el bean **OpenEntityManagerInViewInterceptor** el cual aplica el patrón “Open EntityManager in View” el cuál habilita la extracción de referencias **Lazy** sobre vistas web.
- Si no se requiere dicho comportamiento, es posible deshabilitarlo mediante la propiedad **spring.jpa.open-in-view=false** en el **application.properties**.



iii.v Spring Data JPA (q)

- Consola web para H2.
- La base de datos en memoria H2, provee una consola basada en web. Spring Boot auto-configura la consola si ocurren las siguientes condiciones:
 - Se está desarrollando una aplicación web.
 - La base de datos en memoria es H2 y el paquete com.h2database:h2 está en el classpath.
 - Se está utilizando Spring Boot DevTools.



+

NETFLIX
OSS

iii.v Spring Data JPA (r)

- Consola web para H2.
- Si no se está utilizando Spring Boot DevTools, es posible habilitar la consola web H2 mediante la propiedad:
spring.h2.console.enabled=true.
- Por default, Spring Boot habilita la consola web H2 sobre la URL de la aplicación web **/h2-console**. Es posible personalizar el path de la consola web H2 mediante la propiedad **spring.h2.console.path**.



+

iii.v Spring Data JPA (s)

- Práctica 5. Spring MVC y Spring Data JPA
- Ingresar a la ruta: {tu-workspace}/5-Spring-MVC-and-Spring-Data-JPA
- Define la entidad **User** en el paquete **com.consulting.mgt.springboot.practica5.entities**.
- Define el repositorio **UserRepository** heredando de **JpaRepository**. Define un método que busque un listado de entidades **User** a partir del nombre.
- En el paquete **com.consulting.mgt.springboot.practica5.services.impl**, implementa la interface **IUserService**.



+

iii.v Spring Data JPA (t)

- Práctica 5. Spring MVC y Spring Data JPA
- Revisa los “**templates**” de Thymeleaf, de la ubicación **/src/main/resources/templates** y analiza qué objetos son requeridos para alimentar dichas vistas.
- Revisa también el archivo **data.sql** que está sobre raíz del classpath.
- Analiza dónde se encuentra el **favicon.ico** personalizado.
- En el paquete **com.consulting.mgt.springboot.practica5.controller** implementa el controlador **UserController**, tratando de implementar las rutas y objetos requeridos para las vistas correspondientes.



+

NETFLIX
OSS

iii.v Spring Data JPA (u)

- **Práctica 5. Spring MVC y Spring Data JPA**
- No es necesario realizar cambios a las vistas/templates HTML de Thymeleaf.
- Habilita la consola web de H2 sobre el path **/h2**.
- Ejecuta la clase principal, anotada con **@SpringBootApplication** y prueba tu trabajo en el navegador: <http://localhost:8080>



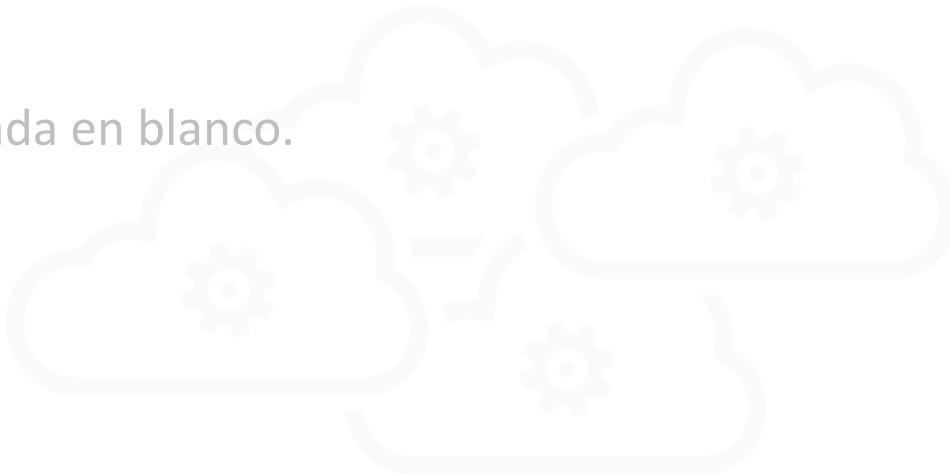
Resumen de la lección

iii.v Spring Data JPA

- Comprendemos como Spring Boot habilita la auto-configuración en aplicaciones Spring Data JPA.
- Comprendemos varios puntos importantes a considerar al utilizar la convención sobre configuración que implementa Spring Boot.
- Revisamos que no es necesario configurar ningún bean para implementar acceso a datos mediante Spring Data JPA y base de datos en memoria, útil para ambientes de desarrollo.
- Practicamos la experiencia adquirida en implementar una aplicación CRUD básica mediante Spring MVC y Spring Data JPA, mediante Spring Boot.



Esta página fue intencionalmente dejada en blanco.



Microservices



+

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles
- iii.iv Spring MVC
- iii.v Spring Data JPA
- iii.vi **Spring Data REST**
- iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

iii.vi Spring Data REST



Microservices



+

NETFLIX
OSS

Objetivos de la lección

iii.vi Spring Data REST

- Conocer la auto-configuración de Spring Data REST mediante Spring Boot.
- Revisar a grandes rasgos las características que habilita Spring Boot a las aplicaciones Spring Data REST.
- Conocer como Spring Data REST implementa HATEOAS mediante HAL.
- Exponer repositorios Spring Data JPA como servicios REST mediante Spring Data REST sin mayor esfuerzo.
- Utilizar Swagger como plataforma de documentación de servicios e interface para probarlos.



iii.vi Spring Data REST (a)

- Spring Data REST expone los repositorios definidos mediante la interfaz **Repository**, de Spring Data, o cualquiera de sus sub-interfaces, como REST endpoints listos para ser consumidos por un cliente REST.
- Spring Boot habilita el desarrollo rápido REST endpoints con Spring Data REST mediante el módulo "**starter**" **spring-boot-starter-data-rest**, para su uso en conjunto con algún módulo soportado de Spring Data.
- Spring Data REST es soportado oficialmente por los módulos:
 - Spring Data JPA
 - Spring Data MongoDB
 - Spring Data Neo4j
 - Spring Data GemFire
 - Spring Data Cassandra



+

NETFLIX
OSS

iii.vi Spring Data REST (b)

- Spring Data REST se construye encima de repositorios definidos mediante Spring Data y los exporta como REST endpoints.
- Spring Data REST expone como APIs REST, todos los repositorios **Repository** definidos o de alguna de sus sub-interfaces.
- Spring Boot expone una serie de propiedades útiles para configurar los repositorios REST expuestos como endpoints mediante el conjunto de propiedades: **spring.data.rest.***.
- No es requerida configuración adicional sobre Spring Boot.



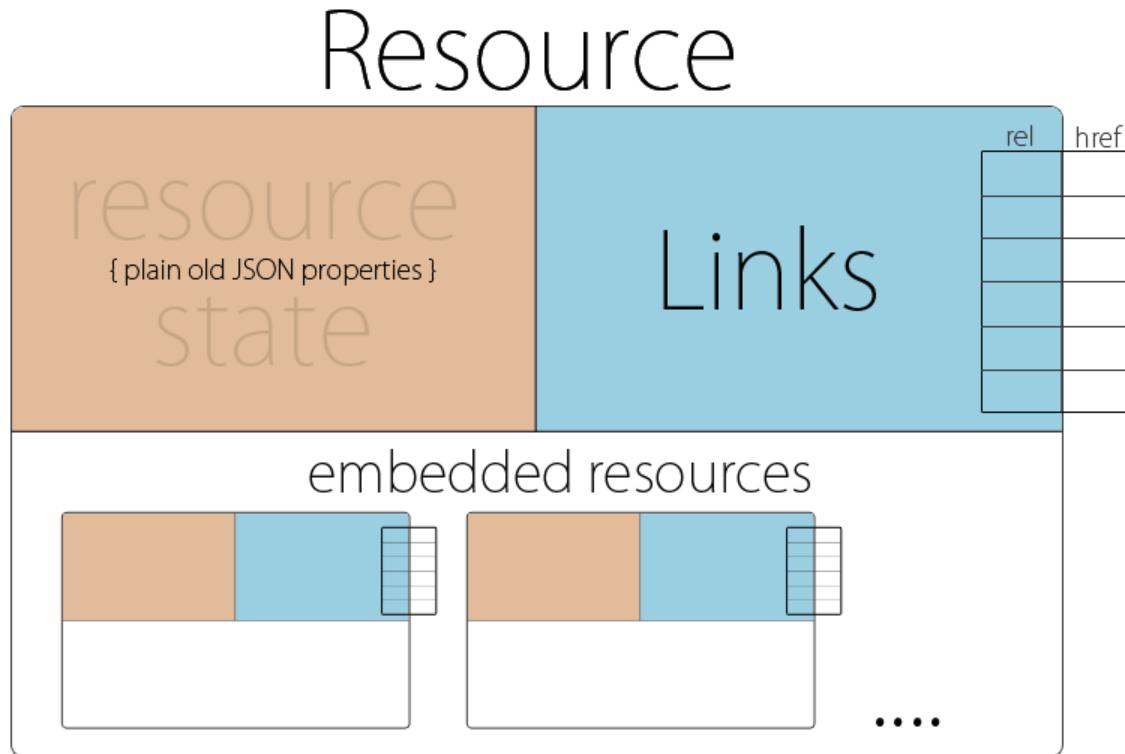
iii.vi Spring Data REST (c)

- Spring Data REST se apalanza mediante el uso correcto de HATEOAS, lo cual permite a los clientes REST, conocer las funcionalidades expuestas por las APIs REST que Spring Data REST expone.
- El principio básico de HATEOAS es que, los clientes puedan descubrir los servicios, es decir, que los servicios sean detectables a través de links.
- Existen algunos mecanismos de-facto, para representar links en formato JSON, Spring Data REST utiliza HAL para representar las respuestas que devuelven los servicios expuestos.



iii.vi Spring Data REST (d)

- HAL



<https://tools.ietf.org/html/draft-kelly-json-hal-08>

iii.vi Spring Data REST (e)

- Ejemplo representación HAL:

GET /orders/523 HTTP/1.1

Host: example.org

Accept: application/hal+json

HTTP/1.1 200 OK

Content-Type: application/hal+json

```
{  
    "_links": {  
        "self": { "href": "/orders/523" },  
        "warehouse": { "href": "/warehouse/56" },  
        "invoice": { "href": "/invoices/873" }  
    },  
    "currency": "USD",  
    "status": "shipped",  
    "total": 10.20  
}
```

GET /orders HTTP/1.1

Host: example.org

Accept: application/hal+json

HTTP/1.1 200 OK

Content-Type: application/hal+json

```
{  
    "_links": {  
        "self": { "href": "/orders" },  
        "next": { "href": "/orders?page=2" },  
        "find": { "href": "/orders{id}" }, "templated": true  
    },  
    "_embedded": {  
        "orders": [  
            {  
                "_links": {  
                    "self": { "href": "/orders/123" },  
                    "basket": { "href": "/baskets/98712" },  
                    "customer": { "href": "/customers/7809" }  
                },  
                "total": 30.00,  
                "currency": "USD",  
                "status": "shipped"  
            }  
        ],  
        "currentlyProcessing": 14,  
        "shippedToday": 20  
    }  
}
```

iii.vi Spring Data REST (f)

- Por default Spring Boot expone los REST endpoints, APIs REST, sobre la raíz de la aplicación web, o “**root**”.
- A partir de Spring Boot 1.2 es posible definir el path base desde donde se expondrán las APIs REST mediante la propiedad:
spring.data.rest.basePath.
- Es posible definir la configuración de un recurso en particular mediante las anotaciones **@RepositoryRestResource** y **@RestResource**.
- La anotación **@CrossOrigin** sobre los repositorios expuestos habilitan el Cross-Domain.



+

NETFLIX
OSS

iii.vi Spring Data REST (g)

- Ejemplo:

```
@CrossOrigin
@RepositoryRestResource(path = "usuarios",
                        collectionResourceRel = "amigos",
                        itemResourceRel = "my_self")
public interface UserRepository extends JpaRepository<User, Long> {

    @RestResource(exported = true, rel = "find", path = "find_by_name")
    List<User> findByName(String name);
}
```

```
{
  "_embedded" : {
    "amigos" : [ {
      "name" : "Claudia",
      "email" : "claudia@boot.com",
      "id" : 1,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api-rest/usuarios/1"
        },
        "my_self" : {
          "href" : "http://localhost:8080/api-rest/usuarios/1"
        }
      }
    },
    {
      "name" : "Fernanda",
      "email" : "fernanda@boot.com",
      "id" : 2,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/api-rest/usuarios/2"
        },
        "my_self" : {
          "href" : "http://localhost:8080/api-rest/usuarios/2"
        }
      }
    },
    { ... }, ... , { ... }],
    ...
  }
}
```

iii.vi Spring Data REST (h)

```
...
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/api-rest/usuarios{&sort}",
      "templated" : true
    },
    "profile" : {
      "href" : "http://localhost:8080/api-rest/profile/usuarios"
    },
    "search" : {
      "href" : "http://localhost:8080/api-rest/usuarios/search"
    }
  },
  "page" : {
    "size" : 100,
    "totalElements" : 11,
    "totalPages" : 1,
    "number" : 0
  }
}
```

GET /api-rest/usuarios HTTP/1.1
 Host: localhost:8080
 Accept: application/hal+json



+

NETFLIX
OSS

iii.vi Spring Data REST (i)

GET /api-rest/usuarios/search HTTP/1.1

Host: localhost:8080

Accept: application/hal+json

HTTP/1.1 200 OK

Content-Type: application/hal+json

```
{  
  "_links" : {  
    "find" : {  
      "href" : "http://localhost:8080/api-rest/usuarios/search/find_by_name{?name}",  
      "templated" : true  
    },  
    "self" : {  
      "href" : "http://localhost:8080/api-rest/usuarios/search"  
    }  
  }  
}
```

GET /api-rest/usuarios/search/find_by_name?name=Fernanda HTTP/1.1

Host: localhost:8080

Accept: application/hal+json

HTTP/1.1 200 OK

Content-Type: application/hal+json

```
{  
    "_embedded" : {  
        "amigos" : [ {  
            "name" : "Fernanda",  
            "email" : "fernanda@boot.com",  
            "id" : 2,  
            "_links" : {  
                "self" : {  
                    "href" : "http://localhost:8080/api-rest/usuarios/2"  
                },  
                "my_self" : {  
                    "href" : "http://localhost:8080/api-rest/usuarios/2"  
                }  
            }  
        }]  
    },  
    "_links" : {  
        "self" : {  
            "href" : "http://localhost:8080/api-rest/usuarios/search/find_by_name?name=Fernanda"  
        }  
    }  
}
```

iii.vi Spring Data REST (j)



+

NETFLIX
OSS

iii.vi Spring Data REST (k)

- Práctica 6. Spring Data REST
- Ingresar a la ruta: {tu-workspace}/6-Spring-Data-REST
- La práctica **6-Spring-Data-REST** parte de la práctica anterior, **5-Spring-MVC-and-Spring-Data-JPA**. Se reutiliza la entidad **User** y se desechan el controlador **UserController**, la interface **IUserService** y su implementación **UserService**.
- A su vez, se elimina la dependencia **Thymeleaf**, debido a que la presente práctica expondrá el repositorio **UserRepository** como API REST mediante Spring Data REST.



iii.vi Spring Data REST (I)

- Práctica 6. Spring Data REST
- Agregar la dependencia **spring-boot-starter-data-rest** al proyecto **6-Spring-Data-REST**.
- Analizar que se han agregado las dependencias **springfox-swagger2**, **springfox-data-rest**, **springfox-beanValidators** y **springfox-swagger-ui**, para la generación y auto-documentación de las APIs creadas por Spring Data REST con **swagger** (opcional).



+

iii.vi Spring Data REST (m)

- Práctica 6. Spring Data REST
- Agregar la configuración necesaria para cumplir con las siguientes condiciones:
 - Que el servidor embebido se ejecute en el puerto **8080**.
 - Que la consola H2 este **habilitada**.
 - Que el path de la consola H2 sea **/h2**.
 - Que el path-base de los repositorios Spring Data REST estén definidos por **/api-rest**.
 - Que el tamaño de página por default de los repositorios expuestos por Spring Data REST sea de **5 elementos** embebidos.
- Se provee la configuración de Swagger en la clase **Swagger2Config** del paquete **com.consulting.mgt.springboot.practica6.swaggerconfig**.



+

NETFLIX
OSS

iii.vi Spring Data REST (n)

- Práctica 6. Spring Data REST
- Analizar que se han eliminado las dependencias de Thymeleaf en los archivos **index.html**, **add-user-form.html** y **update-user-form.html**. Dichos archivos contienen HTML plano.
- Ejecuta la clase principal, anotada con **@SpringBootApplication** y prueba tu trabajo en el navegador: <http://localhost:8080>
- Swagger UI se muestra en la URL: <http://localhost:8080/swagger-ui.html>
- Analiza la funcionalidad implementada mediante JavaScript y los REST endpoints expuestos por Spring Data REST.



iii.vi Spring Data REST (ñ)

- Práctica 6. Spring Data REST Client
- Ingresar a la ruta: {tu-workspace}/6-Spring-Data-REST-Client
- La práctica **6-Spring-Data-REST-Client** parte de la práctica anterior, **6-Spring-Data-REST**. Se reutilizan las vistas HTML del proyecto anterior y se desecha todo lo demás.
- El proyecto **6-Spring-Data-REST-Client** se desplegará en el puerto 9090 y consumirá las APIs expuestas por el proyecto **6-Spring-Data-REST** desplegado en el puerto 8080.



iii.vi Spring Data REST (o)

- Práctica 6. Spring Data REST Client
- Si el proyecto **6-Spring-Data-REST** ha sido desplegado en un puerto diferente al 8080, revisar y asignar correctamente la variable **USER_API_HOST** de los archivos **index.js**, **add-user.js** y **update-user.js** del proyecto **6-Spring-Data-REST-Client**.
- Confirmar que la configuración del proyecto **6-Spring-Data-REST-Client** defina que se desplegará en el puerto 9090.



iii.vi Spring Data REST (p)

- **Práctica 6. Spring Data REST Client**
- Ejecuta la clase principal, de ambos proyectos, anotada con **@SpringBootApplication** y prueba tu trabajo en el navegador:
<http://localhost:8080> (**6-Spring-Data-REST**) y
<http://localhost:9090> (**6-Spring-Data-REST-Client**).
- Analiza la funcionalidad implementada mediante JavaScript en el proyecto **6-Spring-Data-REST-Client**, el cual consume los REST endpoints expuestos por el proyecto **6-Spring-Data-REST**.
- ¿Por qué no funciona la aplicación **6-Spring-Data-REST-Client**? ¿Qué comenta la consola JS? Realiza las correcciones necesarias al proyecto **6-Spring-Data-REST**.



Resumen de la lección

iii.vi Spring Data REST

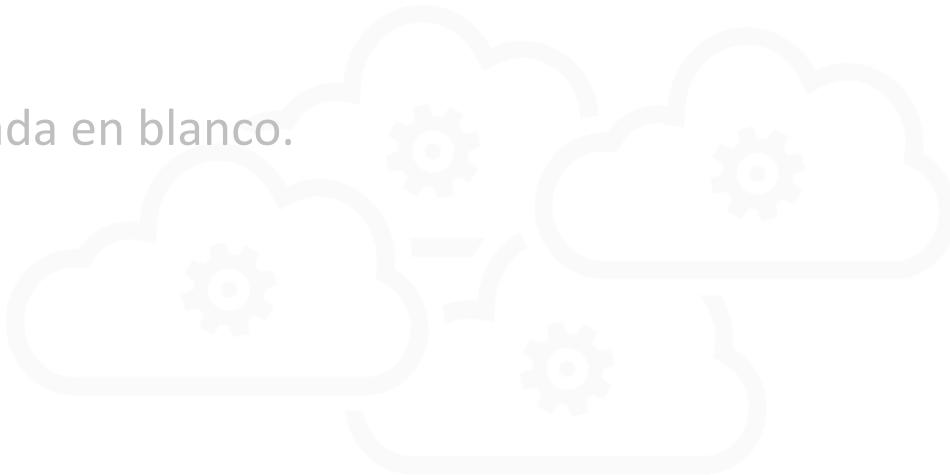
- Comprendemos el mecanismo de auto-configuración de Spring Data REST mediante Spring Boot.
- Implementamos servicios REST a partir de la definición de repositorios Spring Data JPA.
- Implementamos una aplicación simple que consuma dichos servicios REST a través de JavaScript, sin utilizar ninguna tecnología de presentación Java web del lado del servidor (no JSPs, no Thymeleaf).
- Implementamos una aplicación cliente que consuma los servicios REST expuestos bajo otro puerto mediante cross-domain.



+

NETFLIX
OSS

Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

iii. Fundamentos Spring Boot 2.x

- iii.i Introducción a Spring Boot
- iii.ii Configuración de propiedades en Spring Boot
- iii.iii Perfiles
- iii.iv Spring MVC
- iii.v Spring Data JPA
- iii.vi Spring Data REST
- iii.vii **Spring Boot Actuator**



Microservices



iii.vii Spring Boot Actuator



Microservices



+

NETFLIX
OSS

Objetivos de la lección

iii.vii Spring Boot Actuator

- Conocer la auto-configuración de Spring Boot Actuator.
- Conocer las funcionalidades “**out-of-the-box**” que ofrece Spring Boot Actuator.
- Implementar métricas mediante Spring Boot Actuator.
- Comprender que son los actuadores o “**actuators**” e implementar un “actuator” personalizado.



iii.vii Spring Boot Actuator (a)

- Entre todas las opciones de auto-configuración que provee Spring Boot en conjunción con los múltiples proyectos que soporta, Spring Boot ofrece “**Actuators**”.
- Spring Boot ofrece “**out-of-the-box**” funcionalidades adicionales que apoyan, a los desarrolladores y personal de operaciones, a monitorear y administrar la aplicación al momento de estar en producción.
- Es posible administrar y monitorear la aplicación mediante “**endpoints**” HTTP o a través de JMX.



iii.vii Spring Boot Actuator (b)

- Spring Boot Actuator habilita la administración y monitorización de las aplicaciones mediante el módulo “starter” **spring-boot-starter-actuator**.
- Un “**actuator**” o actuador, es un término proveniente de la industria manufacturera que refiere a un dispositivo mecánico para mover o controlar algo. Los actuadores pueden generar una gran cantidad de movimiento a partir de un ligero cambio.
- Los actuadores o “**actuators**” se disponibilizan a través de “**endpoints**” HTTP o mediante JMX.



+

NETFLIX
OSS

iii.vii Spring Boot Actuator (c)

- Spring Boot Actuator incluye varios "actuators" de entre los cuales destacan:
 - beans
 - env
 - health
 - httptrace
 - info
 - metrics
 - mappings
 - shutdown, entre otros.

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready-endpoints>



+

iii.vii Spring Boot Actuator (d)

- Por default, todos los endpoints “**actuators**” están habilitados por default, excepto el endpoint “**shutdown**”.
- Se utiliza la propiedad **management.endpoint.<endpoint>.enabled** para habilitar un endpoint.
- Si se desea, es posible deshabilitar todos los endpoints por default, y habilitar sólo aquellos requeridos para un momento determinado por la aplicación mediante la anotación:
management.endpoints.enabled-by-default

iii.vii Spring Boot Actuator (e)

- Los endpoints se configuran, uno a uno mediante la propiedad **management.endpoint.<nombre-del-endpoint>**.
- Por default, Spring Boot Actuator, habilita los endpoints a través de hipermedia, mediante un endpoint especial para el auto-descubrimiento de actuators bajo el path base **/actuator**. Es posible configurar éste path base mediante la propiedad **management.endpoints.web.base-path**.



+

NETFLIX
OSS

iii.vii Spring Boot Actuator (f)

- Los endpoints configurados con Spring Boot Actuator, están deshabilitados para el uso cross-domain de los mismos; para habilitarlos es necesario configurar las siguientes propiedades:
 - `management.endpoints.web.cors.allowed-origins`
 - `management.endpoints.web.cors.allowed-methods`



+

NETFLIX
OSS

iii.vii Spring Boot Actuator (g)

- Actuadores personalizados.
- Para definir actuadores personalizados utilizamos la anotación **@Endpoint**, la cuál expone la clase anotada como un endpoint la cual deberá responder a solicitudes **GET, POST y/o DELETE** para manipular la información que el endpoint requiera.
- Las anotaciones **@ReadOperation**, **@WriteOperation** y **@DeleteOperation**, sobre métodos de una clase anotada con **@Endpoint** expondrán dicha funcionalidad mediante el verbo correspondiente HTTP.



iii.vii Spring Boot Actuator (h)

- Recibir parámetros sobre actuadores personalizados.
- El paso de parámetros a los métodos “**operation**”, aquellos anotados con **@ReadOperation**, **@WriteOperation** y **@DeleteOperation**, se da mediante el nombre de sus parámetros y el nombre de los atributos de las llaves de “**query-parameters**” o el nombre de las llaves del objeto JSON de entrada en la petición HTTP.

iii.vii Spring Boot Actuator (i)

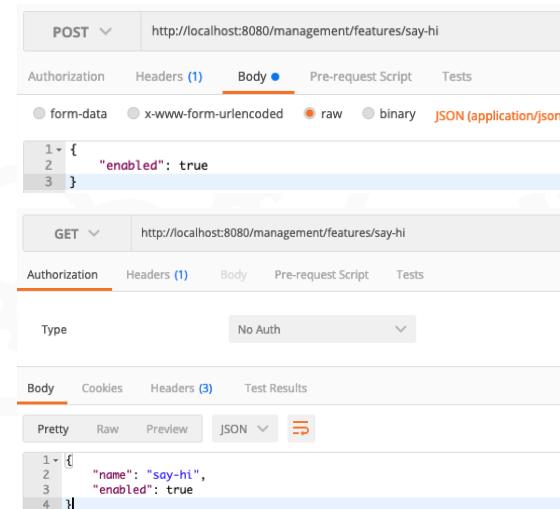
- Recibir parámetros sobre actuadores personalizados.

```
@Component
@Endpoint(id = "features")
public class FeaturesEndpoint {

    @Autowired
    private Map<String, Feature> features;

    @ReadOperation
    public Feature feature(@Selector String name) {
        return features.get(name);
    }

    @WriteOperation
    public void configureFeature(@Selector String name, boolean enabled) {
        features.put(name, Feature.builder().name(name).enabled(enabled).build());
    }
    ...
}
```



The screenshot shows two requests in Postman:

- POST** to `http://localhost:8080/management/features/say-hi` with a JSON body:

```
1 - {
2   "enabled": true
3 }
```

- GET** to `http://localhost:8080/management/features/say-hi` with a JSON response:

```
1 - {
2   "name": "say-hi",
3   "enabled": true
4 }
```

iii.vii Spring Boot Actuator (j)

- De forma análoga, mediante las anotaciones **@ControllerEndpoint** y **@RestControllerEndpoint**, es posible definir endpoints actuadores, definidos mediante anotaciones de Spring MVC o Spring WebFlux.
- No se revisa Spring Boot Actuator en su extensión debido a que no es objetivo del curso.



iii.vii Spring Boot Actuator (k)

- Práctica 7. Spring Boot Actuator
- Ingresar a la ruta: {tu-workspace}/7-Spring-Boot-Actuator
- Habilita la exposición web, es decir, vía HTTP de los siguientes “**actuators**”: **shutdown**, **health**, **info**, **beans**, **env**, **metrics** y **features**, mediante la propiedad **management.endpoints.web.exposure.include**.
- Habilita el “**actuator**” **shutdown**, para apagar la aplicación web correctamente.
- Cambia el path base de Spring Boot Actuator de: **/actuator** a **/management**.



+

NETFLIX
OSS

iii.vii Spring Boot Actuator (I)

- **Práctica 7. Spring Boot Actuator**
- Analiza la definición del bean **features** en la clase principal del proyecto.
- Analiza la clase **Feature**.
- Analiza la clase de configuración **CustomFeaturesConfig**.
- Opcional. Analiza las clases del paquete
com.consulting.mgt.springboot.practica7.spel.
- Analiza la clase **HelloController**.



+

iii.vii Spring Boot Actuator (m)

- Práctica 7. Spring Boot Actuator
- Implementa la clase **@Endpoint FeaturesEndpoint** para definir métodos de lectura (dos, uno que resuelva todas las features y otro que resuelva por nombre de feature), escritura (para definir un nuevo feature) y de borrado (para eliminar una feature).
- Ejecuta la clase principal del proyecto, anotada con **@SpringBootApplication** y prueba tu trabajo en el navegador: <http://localhost:8080>. Accede al path **/management** y manipula los diversos endpoints expuestos por Spring Boot Actuator.



+

NETFLIX
OSS

iii.vii Spring Boot Actuator (n)

- **Práctica 7. Spring Boot Actuator**
- Prueba el endpoint “/management/health” ¿Qué se visualiza? Extiende la visualización del endpoint “health” mediante la propiedad **management.endpoint.health.show-details**.
- Prueba el endpoint “/management/info” ¿Qué se visualiza? Extiende la visualización del endpoint “info” mediante definir propiedades **info.***.
- Agrega la dependencia Spring Data REST HAL Browser:
org.springframework.data:spring-data-rest-hal-browser, vuelve a lanzar la clase principal y accede a: <http://localhost:8080/browser/index.html>



+

NETFLIX
OSS

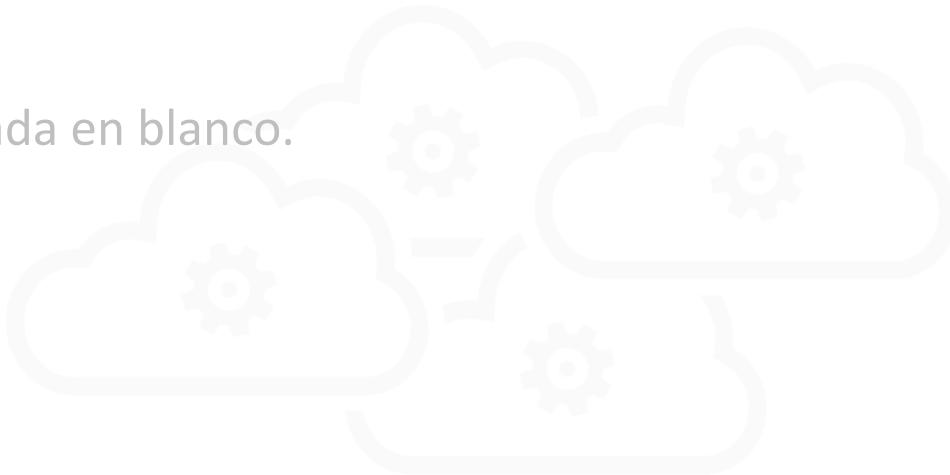
Resumen de la lección

iii.vii Spring Boot Actuator

- Comprendemos algunas de las funcionalidades “out-of-the-box” que ofrece Spring Boot Actuator.
- Aprendimos que es un “Actuator” o actuador y como beneficia a nuestra aplicación para la administración y monitoreo
- Implementamos métricas y actuadores personalizados.
- Analizamos algunos de los “Actuator” endpoints más utilizados por default en aplicaciones Spring Boot.



Esta página fue intencionalmente dejada en blanco.



Microservices



3. Contenido

- i. Arquitectura de sistemas monolíticos
- ii. Introducción a la Arquitectura Orientada a Servicios
- iii. Fundamentos Spring Boot 2.x
- iv. **Arquitectura de Microservicios**
- v. Implementación de Microservicios con Spring Boot
- vi. Microservicios con Spring Cloud y Spring Netflix OSS



+

NETFLIX
OSS

iv. Arquitectura de Microservicios



Microservices



+

NETFLIX
OSS

iv. Arquitectura de Microservicios

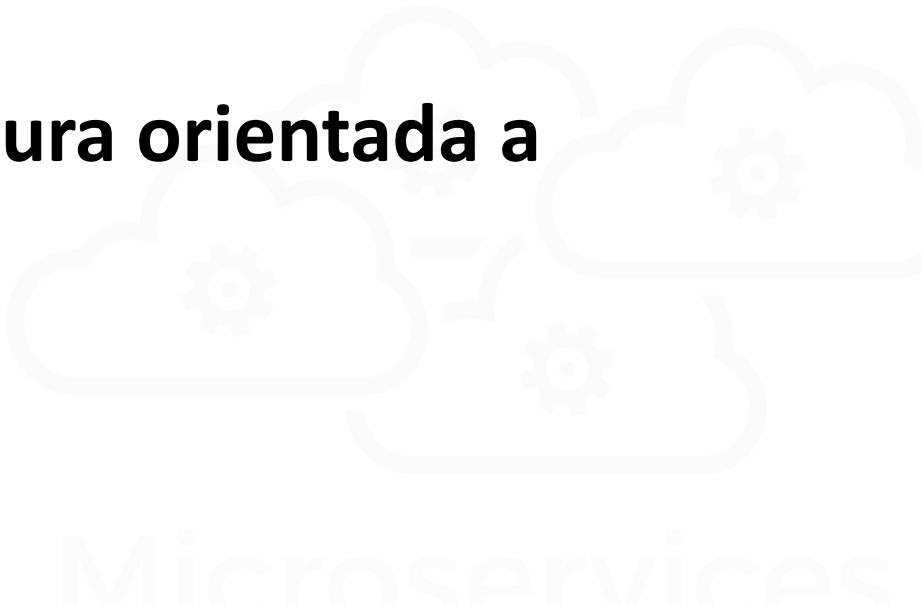
- iv.i ¿Qué es la arquitectura orientada a microservicios?
- iv.ii Descomponiendo aplicaciones monolíticas.
- iv.iii Protocolos ligeros de comunicación para microservicios.
- iv.iv Aplicaciones “cloud-native”.
- iv.v Twelve-Factor Apps.
- iv.vi Orquestación vs Coreografía.
- iv.vii Patrones de diseño para la nube.
- iv.viii Gestión de Transacciones ACID vs BASE.
- iv.ix Principios de diseño para aplicaciones “cloud-native”.
- iv.x SOA vs APIs vs Microservicios.
- iv.xi API Layer.



+

NETFLIX
OSS

iv.i ¿Qué es la arquitectura orientada a microservicios?



Microservices



+

NETFLIX
OSS

Objetivos de la lección

iv.i ¿Qué es la arquitectura orientada a microservicios?

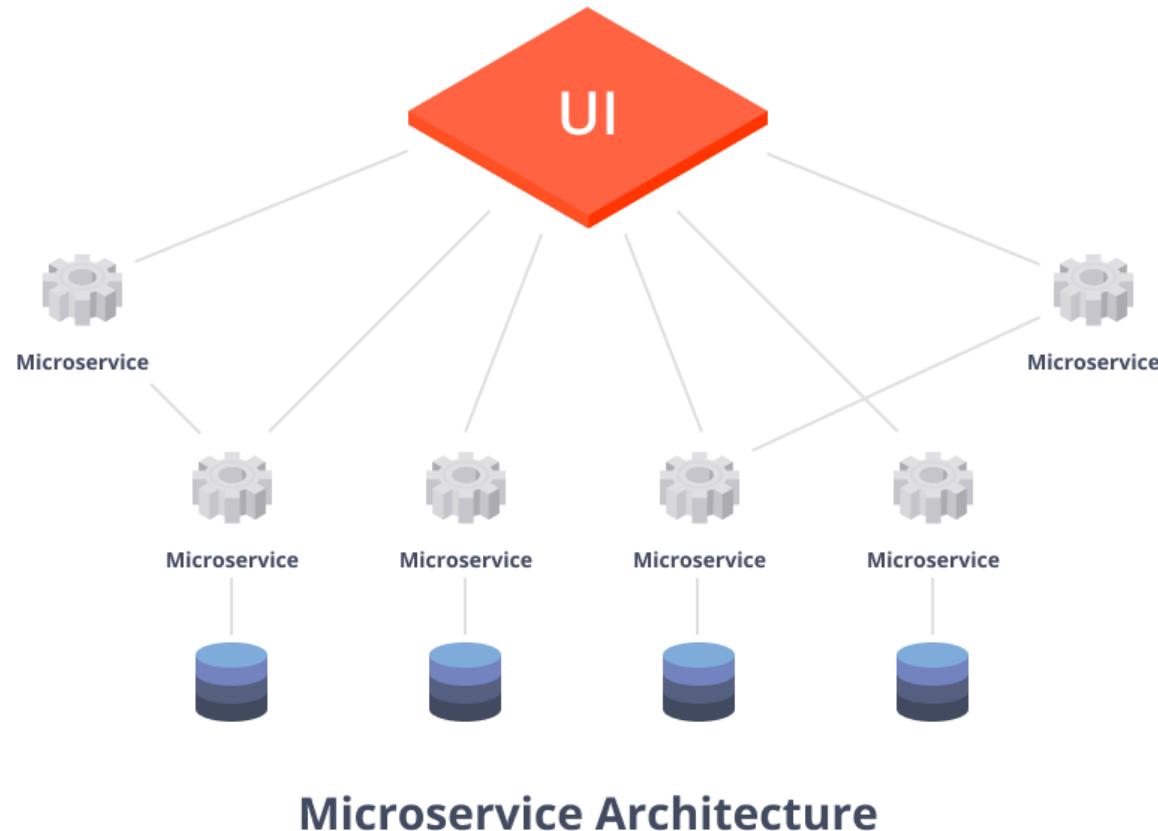
- Comprender qué es una arquitectura orientada a microservicios.
- Contrastar una arquitectura tradicional o monolítica contra una arquitectura orientada a microservicios.
- Comprender las características de las arquitecturas orientadas a microservicios.
- Analizar ventajas y desventajas de las arquitecturas orientadas a microservicios.



iv.i ¿Qué es la arquitectura orientada a microservicios? (a)

- Los microservicios, o la arquitectura orientada a microservicios, es un tipo de arquitectura, que sirve para diseñar aplicaciones donde las funciones o funcionalidades del sistema están desplegadas de forma independiente a diferencia de los sistemas basados en arquitecturas tradicionales o monolíticas.
- Cada función se denomina servicio y se puede diseñar, codificar e implementar de forma independiente, permitiendo que funcionen por separado, de forma aislada, y que también fallen por separado sin afectar a los demás servicios.

iv.i ¿Qué es la arquitectura orientada a microservicios? (b)





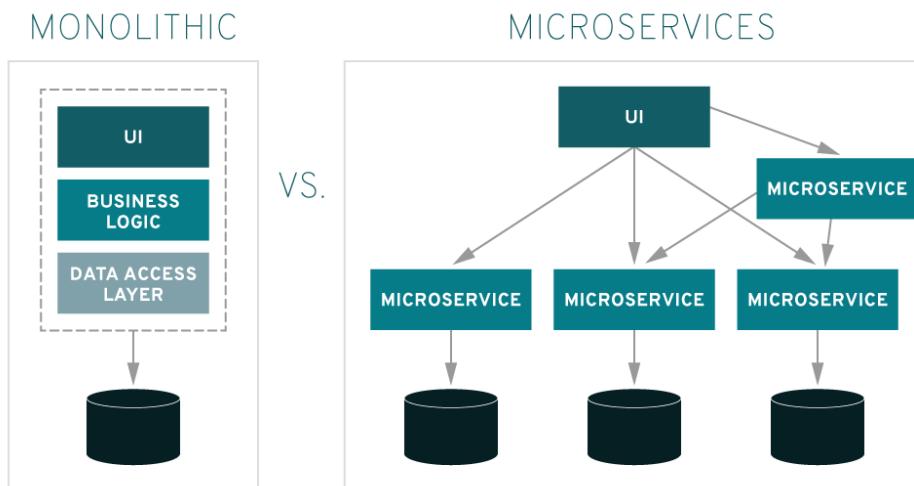
iv.i ¿Qué es la arquitectura orientada a microservicios? (c)

- Una arquitectura orientada a microservicios consta de un conjunto de servicios pequeños que hacen una sola cosa bien.
- Los microservicios son autónomos, es decir, son independientes entre sí y cada uno debe de implementar una funcionalidad de negocio individual.
- Desde un punto de vista técnico, los microservicios son la evolución natural de las arquitecturas orientadas a servicios.



iv.i ¿Qué es la arquitectura orientada a microservicios? (d)

- Las arquitecturas de microservicios consisten en desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno con su propio proceso y cada uno independiente de los demás, lo cual los hace más flexibles al cambio, escalables y robustos.





iv.i ¿Qué es la arquitectura orientada a microservicios? (e)

- Características que definen un microservicio (a):
 - Los microservicios son pequeños e independientes y se mantienen desacoplados mediante interfaces bien definidas (o acoplados de forma flexible).
 - Cada microservicio tiene un código base independiente, que puede administrarse por un equipo de desarrollo pequeño (two-pizza-rule).
 - Los microservicios pueden implantarse de manera independiente, es decir, un equipo puede actualizar un servicio existente sin tener que volver a compilar y desplegar toda la aplicación.



iv.i ¿Qué es la arquitectura orientada a microservicios? (f)

- Características que definen un microservicio (b):
 - Los microservicios son responsables de conservar sus propios datos o estado externo. Esto difiere del modelo tradicional, donde una capa de datos independiente controla la persistencia de los datos.
 - Los microservicios se comunican entre sí mediante protocolos conocidos, simples y APIs bien definidas, ocultando los detalles de la implementación interna de cada microservicio frente a otros microservicios.
 - No es necesario que los microservicios compartan el mismo “stack” tecnológico, bibliotecas o frameworks. Los microservicios pueden ser políglotas.



iv.i ¿Qué es la arquitectura orientada a microservicios? (g)

- Ventajas:
 - Desarrollo independiente.
 - Equipos pequeños y centrados, metodologías ágiles.
 - Despliegue independiente
 - Escalabilidad independiente.
 - Reusabilidad.
 - Aislamiento de errores
 - Stack tecnológico mixto (políglotas).
 - Facilidad de mantenimiento.
 - Facilidad de ejecución de pruebas unitarias.
 - Tolerancia a fallos, alta disponibilidad y replicación.
 - Distribución de carga, concurrencia y tiempos de respuesta.



iv.i ¿Qué es la arquitectura orientada a microservicios? (h)

- ¿Cuándo implementar una arquitectura orientada a microservicios?:
 - Aplicaciones grandes que requieran una **alta velocidad** de publicación de nuevos "**releases**" o funcionalidades.
 - Aplicaciones complejas que requieren de gran **escalabilidad y elasticidad**.
 - Aplicaciones que den soporte a negocios complejos donde se definan **múltiples dominios o sub-dominios** de negocio.
 - Organizaciones que dispongan de **pequeños equipos de trabajo**.
 - Organizaciones que dispongan de **equipos de trabajo distribuidos**.
- La arquitectura orientada a microservicios no es una "bala de plata".



iv.i ¿Qué es la arquitectura orientada a microservicios? (i)

- Desventajas (a):
 - **Complejidad:** Una aplicación de microservicios tiene más partes en movimiento que la aplicación monolítica equivalente. Cada servicio es más sencillo, pero el sistema como un todo es más complejo.
 - **Dependencias para desarrollo y pruebas:** El rápido desarrollo de aplicaciones orientadas a microservicios suponen un mayor esfuerzo en el desarrollo (y pruebas) de microservicios dependientes de otros. La refactorización en las interfaces y en los límites del servicio puede resultar catastróficos. Debido a lo anterior, el versionado de los componentes es muy importante.



iv.i ¿Qué es la arquitectura orientada a microservicios? (j)

- Desventajas (b):
 - **Falta de Gobierno:** Debido a la falta de gobernabilidad, una arquitectura basada en microservicios puede acabar con tantos lenguajes y frameworks diferentes causando que la aplicación sea difícil de mantener.
 - **Congestión y latencia de red:** Uno de los mayores problemas de las arquitecturas basadas en microservicios son las **falacias de la computación distribuida**. La comunicación entre muchos microservicios pequeños y detallados dar lugar a una mayor congestión y latencia en la red.



iv.i ¿Qué es la arquitectura orientada a microservicios? (k)

- Desventajas (c):
 - **Integridad de los datos:** Cada microservicio es responsable de la conservación de sus propios datos, como consecuencia, la coherencia de los datos puede suponer un problema (eventual consistencia).
 - **Administración:** Para tener éxito con los microservicios se necesita una cultura de DevOps consolidada debido a que el registro correlacionado entre microservicios puede resultar un desafío.



iv.i ¿Qué es la arquitectura orientada a microservicios? (I)

- Desventajas (d):
 - **Control de versiones:** Las actualizaciones de un servicio no deben interrumpir servicios que dependen del mismo. Es posible que varios microservicios se actualicen en cualquier momento, por lo tanto, sin un cuidadoso diseño entre sus interfaces y sin un adecuado versionamiento, podrían surgir problemas con la compatibilidad con versiones anteriores y/o posteriores del software.
 - **Conjunto de habilidades:** Los microservicios son sistemas muy distribuidos. Es necesario evaluar si el equipo de desarrollo tiene los conocimientos y la experiencia para desenvolverse correctamente en el desarrollo de sistemas basados en arquitectura de microservicios.



iv.i ¿Qué es la arquitectura orientada a microservicios? (m)

- Desventajas (e):
 - **Mayor complejidad para los operadores:** Para los operadores, o equipos de monitoreo, se origina una explosión de mayores procesos a administrar y monitorear debido a que pueden desplegarse decenas, cientos o miles de microservicios en ejecución donde cada uno de ellos, se ejecuta en un proceso independiente.
 - **Conjunto de habilidades:** Los microservicios son sistemas muy distribuidos. Es necesario evaluar si el equipo de desarrollo tiene los conocimientos y la experiencia para desenvolverse correctamente en el desarrollo de sistemas basados en arquitectura de microservicios.



iv.i ¿Qué es la arquitectura orientada a microservicios? (n)

- Desventajas (f):
 - **Mayor complejidad en la delimitación de los microservicios:** La complejidad del negocio puede aparentar que, sobre el papel, los microservicios estén bien delimitados, sin embargo, mientras se va desarrollando e implementando posibles caminos alternos, se descubre que los microservicios no son tan independientes entre. Ejemplo, compartición de los mismos datos.
 - **Obviar la complejidad del estado entre microservicios:** El manejo de microservicios sin estado es efectivo, es decir que los servicios son “stateless”. El manejo de estado dificulta la escalabilidad. Los microservicios deberían de recibir como entrada todos los datos requeridos para operar.



iv.i ¿Qué es la arquitectura orientada a microservicios? (ñ)

- Desventajas (g):
 - **Transaccionabilidad:** Amplia dificultad para implementar operaciones transaccionales entre llamadas a microservicios. Supone un gran esfuerzo y ello conyeva a aumentar los tiempos de respuesta de los microservicios, habilitando “cuellos de botella”. No se recomienda implementar transaccionabilidad entre microservicios, para ello se recomienda implementar servicios idempotentes o habilitar “eventual consistencia”.
 - **Monolítos disfrazados, microservicios o nanoservicios:** Delimitar los microservicios es crucial. ¿Qué tan micro es un microservicio?



iv.i ¿Qué es la arquitectura orientada a microservicios? (o)

- Desventajas (h):
 - **Dificultad para el despliegue:** Dado el alto número de microservicios que puede suponer un sistema en su totalidad, la administración para el despliegue de los microservicios supone un reto. Requiere automatización y una cultura DevOps madura.
 - Falacias de la computación distribuida, entre otras ...



+

NETFLIX
OSS

iv.i ¿Qué es la arquitectura orientada a microservicios? (p)

- Falacias de la computación distribuida
 - La red es confiable.
 - La latencia es cero.
 - El ancho de banda es infinito.
 - La red es segura.
 - La topología no cambia.
 - Hay uno y sólo un administrador.
 - El costo de transporte es cero.
 - La red es homogénea





Resumen de la lección

iv.i ¿Qué es la arquitectura orientada a microservicios?

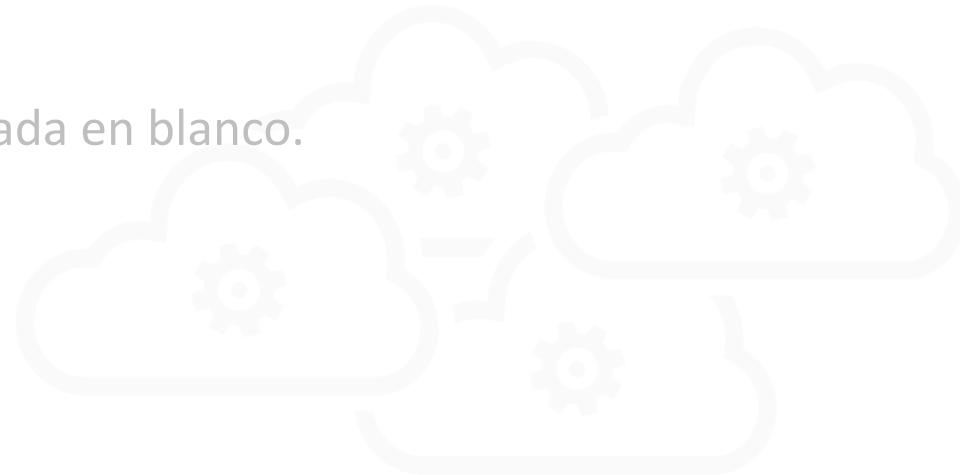
- Asasd.a.sd.as.da.sd



Microservices



Esta página fue intencionalmente dejada en blanco.



Microservices



+

iv. Arquitectura de Microservicios

- iv.i ¿Qué es la arquitectura orientada a microservicios?
- iv.ii Descomponiendo aplicaciones monolíticas.
- iv.iii Protocolos ligeros de comunicación para microservicios.
- iv.iv Aplicaciones “cloud-native”.
- iv.v Twelve-Factor Apps.
- iv.vi Orquestación vs Coreografía.
- iv.vii Patrones de diseño para la nube.
- iv.viii Gestión de Transacciones ACID vs BASE.
- iv.ix Principios de diseño para aplicaciones “cloud-native”.
- iv.x SOA vs APIs vs Microservicios.
- iv.xi API Layer.



+

NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas.

Microservices



Objetivos de la lección

iv.ii Descomponiendo aplicaciones monolíticas

- Analizar algunos puntos a considerar para saber si es conveniente o no implementar una aplicación mediante una arquitectura orientada a microservicios.
- Comprender como implementar la migración de un sistema monolítico a una arquitectura orientada a microservicios.
- Aprender ciertas recomendaciones a la hora de descomponer aplicaciones monolíticas.
- Analizar lo que es el Domain-Driven Design y los "Bounded-context".

iv.ii Descomponiendo aplicaciones monolíticas (a)

- Hoy en día, las organizaciones ya cuentan con múltiples sistemas, monolíticos o no, escritos en diversos lenguajes de programación o normados y desarrollados bajo uno o más lenguajes de programación.
- Por lo general, las organizaciones han invertido mucho dinero en el desarrollo de sus sistemas y no pueden darse el lujo de tirar todo y volver a empezar desarrollando, nuevamente, sus sistemas orientados a microservicios.
- Sin embargo refactorizar una aplicación monolítica a una arquitectura de microservicios, puede resultar una buena idea, para disfrutar de sus beneficios y afrontar sus desventajas.

iv.ii Descomponiendo aplicaciones monolíticas (b)

- ¿Es conveniente adoptar una arquitectura orientada a microservicios en mi empresa o proyecto? Sí, si...
 - Es una aplicación grande que requiera una **alta velocidad** de publicación de nuevos "**releases**" o funcionalidades.
 - Es una aplicación compleja que requiere de gran **escalabilidad y elasticidad**.
 - Es una aplicación que da soporte a un negocio complejos donde se definen **múltiples dominios o sub-dominios** de negocio.
 - La organización dispone de **pequeños equipos de trabajo**.
 - La organización dispone de **equipos de trabajo distribuidos**.
 - La organización esta dispuesta a afrontar los desafios inherentes.



+

NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (c)

- La refactorización de un código fuente o aplicación sugiere: “Introducir una transformación de código preservando el comportamiento existente”.
- Durante la refactorización de un monolito a microservicios se resume a mantener iguales sus APIs externas mientras se cambia la manera en la que el sistema se compila, empaqueta, despliega y opera internamente.
- No se añade nueva funcionalidad mientras se refactoriza a microservicios.

iv.ii Descomponiendo aplicaciones monolíticas (d)

- Para evaluar si una aplicación es apta para ser refactorizada a microservicios es necesario considerar:
 - ¿Cómo esta empaquetada la aplicación?
 - ¿Cómo funciona el código de la aplicación?
 - ¿Con qué tipo de arquitectura está implementada la aplicación?
 - ¿Se cuenta con la arquitectura, “blueprints”, de la aplicación?
 - ¿Cómo están definidos los orígenes de datos de la aplicación?
 - ¿Cómo estan estructurados los datos persistidos en la aplicación?



+

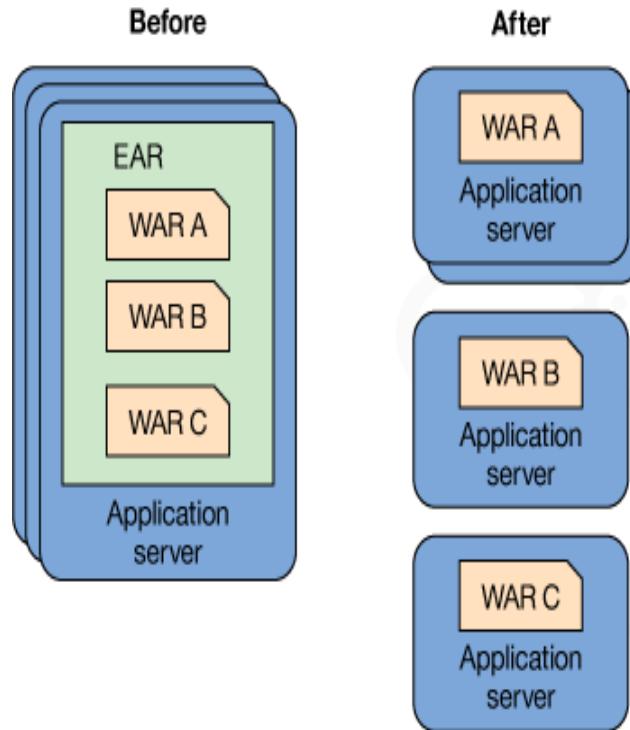
NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (e)

- ¿Cómo re-empaquetar la aplicación? (aplicaciones Java)
- **Analizar su diagrama de despliegue.**
- **Revisar la estructura de la aplicación.**
- **Dividir archivos EAR:** En lugar de empaquetar en un EAR todos los WAR y/o JARs requeridos, dividir en archivos WAR independientes, lo cual causaría cambios en el código.
- **Implementar contenedores por servicio:** Desplegar cada WAR en un contenedor de Servlets independiente.
- **Crear, desplegar y gestionar de forma independiente.**

iv.ii Descomponiendo aplicaciones monolíticas (f)

- Re-empaquetado y re-despliegue de la aplicación.

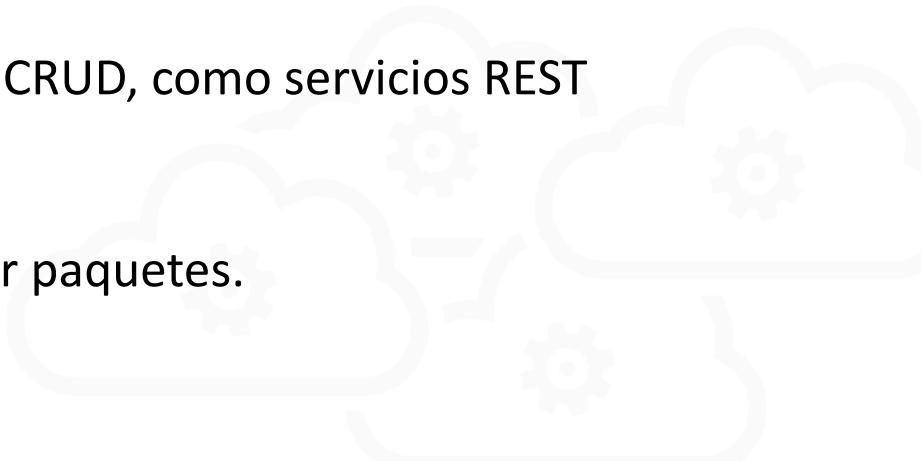


iv.ii Descomponiendo aplicaciones monolíticas (g)

- Refactorizar código.
- Debido a que el aplicativo ha sido desplegado en diferentes archivos WAR y en diferentes contenedores, las interfaces de comunicación entre ellos deberán refactorizarse también.
- Posiblemente muchas llamadas entre servicios se hacían en forma de procesos, es decir, mediante una simple llamada desde un objeto “caller” a un “worker”, ahora deberá hacerlas mediante protocolos de comunicación síncronos o asíncronos entre distintos microservicios.

iv.ii Descomponiendo aplicaciones monolíticas (h)

- Refactorizar código.
- Exponer operaciones simples, tipo CRUD, como servicios REST orientados a dominio.
- Distribución de funcionalidades por paquetes.



Microservices



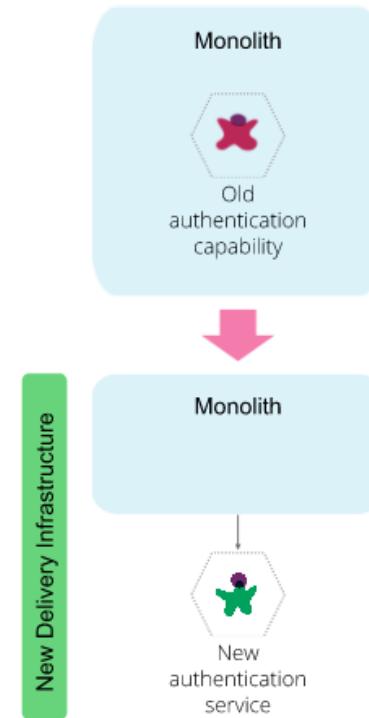
+

iv.ii Descomponiendo aplicaciones monolíticas (i)

- Refactorizar los datos y su persistencia.
- Analizar como refactorizar los datos.
- Verificar como se realizan las búsquedas en la aplicación:
 - Si se realizan mediante claves primarias, posiblemente migrar a una Base de Datos NoSQL, resulta bien y con mejor performance a una Base de Datos relacional.
 - Si se realizan búsquedas complejas en base a múltiples uniones entre tablas, una Base de Datos SQL puede seguir resultando bien.
- Comprender apliamente como se distribuyen los datos entre tablas y/o colecciones, dado que serán particionadas.

iv.ii Descomponiendo aplicaciones monolíticas (j)

- Recomendaciones.
- Iniciar con una capacidad del sistema simple y bastante desacoplada.





+

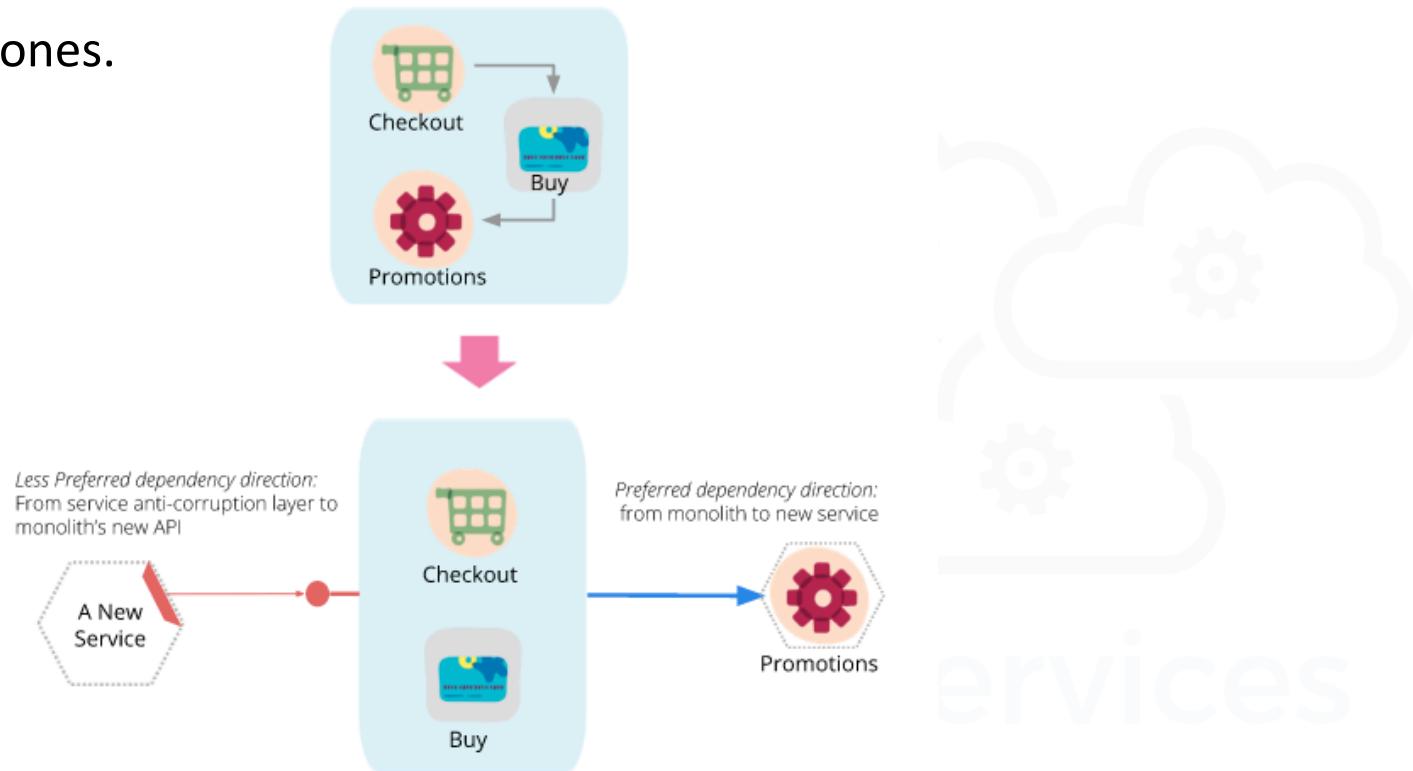
NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (k)

- Recomendaciones.
- Minimizar la dependencia de los microservicios hacia el monolito y tratar de aumentar la dependencia del monolito hacia los microservicios, ello ocasionará continuar desacoplando los componentes dependientes hacia los microservicios, creando nuevos microservicios.
- Desacoplar las funcionalidades acopladas mediante interfaces.

iv.ii Descomponiendo aplicaciones monolíticas (I)

- Recomendaciones.



iv.ii Descomponiendo aplicaciones monolíticas (m)

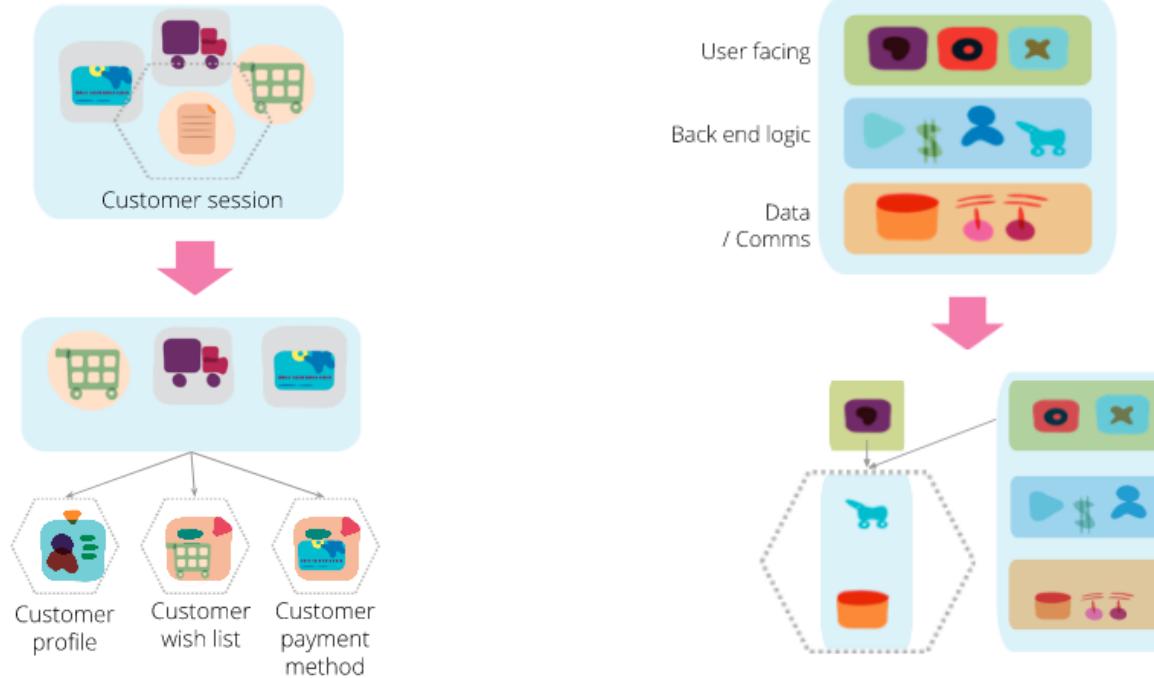
- Recomendaciones.
- Implementar cache distribuido en lugar de sesiones (web).
- Desacoplar grupos de funcionalidades por contexto de negocio (bounded-context).
- Desacoplar funcionalidades sin re-escribir código, es viable refactorizar utilizando interfaces, pero sin re-implementar lógica de negocio actualmente implementadas.
- Mantener clases cohesivas.

iv.ii Descomponiendo aplicaciones monolíticas (n)

- Recomendaciones.
- Refactorizar funcionalidad a nivel macro y luego a micro, empezando primero con pocos microservicios y después segregando a más microservicios definidos por un contexto delimitado (bounded-context).
- Refactorizar a pasos evolutivos e iterativos, es decir, refactorice la aplicación a nivel macro en su totalidad y luego a micro en su totalidad.

iv.ii Descomponiendo aplicaciones monolíticas (ñ)

- Recomendaciones.



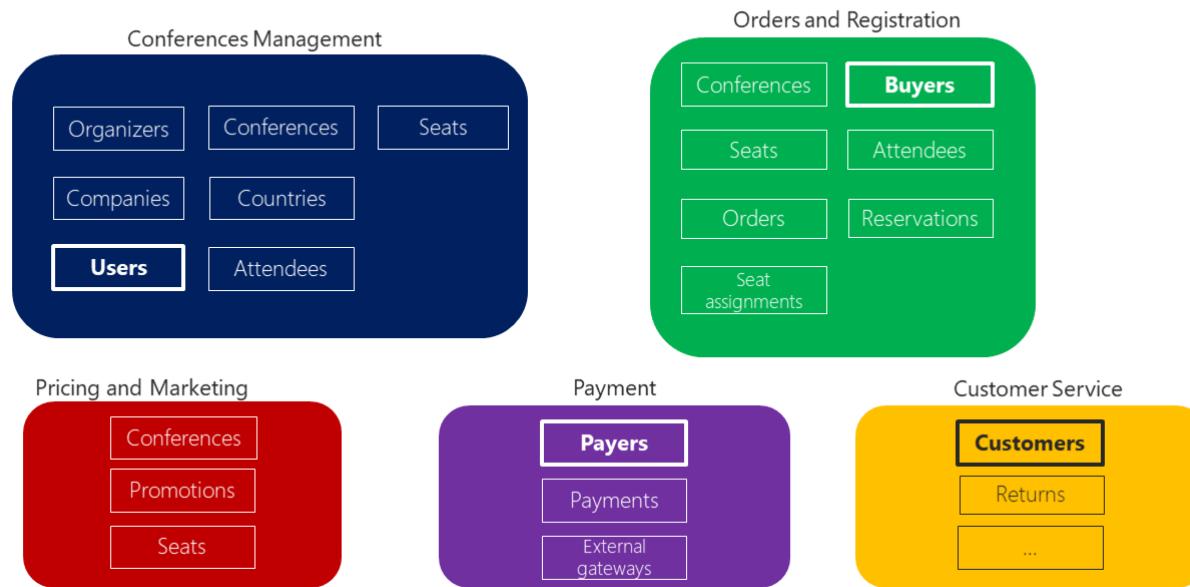
iv.ii Descomponiendo aplicaciones monolíticas (o)

- Domain Driven Design (Diseño Guiado por el Dominio).
- El DDD propone un modelado de objetos basado en la realidad de negocio con relación a sus casos de uso.
- En el contexto del desarrollo de aplicaciones, DDD hace referencia a los problemas como dominios y describe áreas del negocio independientes como contextos delimitados (cada contexto delimitado está correlacionado con un microservicio) resultando en un lenguaje común para hablar del negocio y de las clases o implementaciones que dan soporte tecnológico al negocio.

iv.ii Descomponiendo aplicaciones monolíticas (p)

- Domain Driven Design (Diseño Guiado por el Dominio).

Identifying a Domain Model per Microservice or Bounded Context





+

NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (q)

- Domain Driven Design (Diseño Guiado por el Dominio).
- Por lo regular los patrones y técnicas del DDD se perciben como obstáculos al delimitar contextos para la implementación de los microservicios, pero los patrones y las técnicas no son lo importante, sino organizar el código para que esté en línea con los problemas de negocio y utilizar los mismos términos empresariales a nivel de código (lenguaje ubicuo).
- Lenguaje ubicuo: Lenguaje común entre programadores/técnicos y usuarios o negocio.

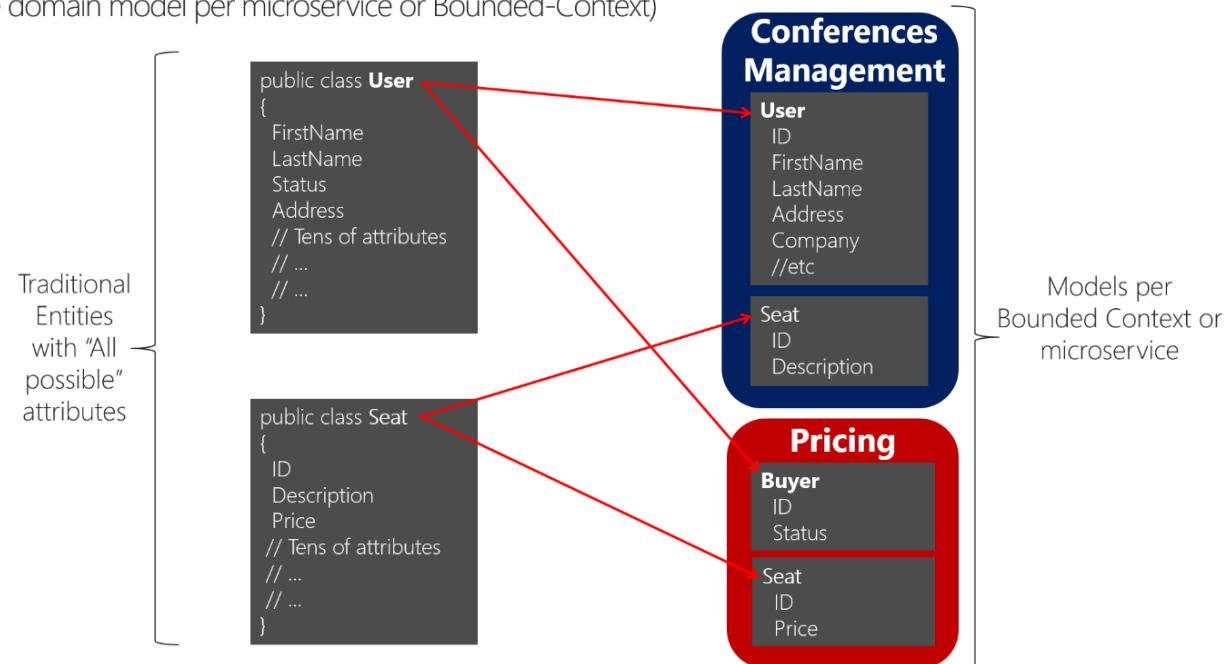
iv.ii Descomponiendo aplicaciones monolíticas (r)

- Domain Driven Design (Diseño Guiado por el Dominio).
- La clave para una correcta delimitación de contextos para la implementación de microservicios esta en situar los límites en el contexto de negocio y trasladar ese contexto y sus problemas a resolver a un microservicio.
- El DDD afecta a los límites en el contexto de negocio y por tanto afecta a los límites en la implementación de microservicios, es muy importante delimitar los contextos.

iv.ii Descomponiendo aplicaciones monolíticas (s)

- Domain Driven Design (Diseño Guiado por el Dominio).

Decomposing a traditional data model into multiple domain models
(One domain model per microservice or Bounded-Context)





+

NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (t)

- “**Bounded-context**” estrechos. Delimitar contextos relativamente estrechos.
- Determinar dónde colocar los límites entre contextos delimitados contrapone dos objetivos.
 1. Es importante delimitar los microservicios lo más pequeños posibles aunque el principal objetivo no es que sean pequeños, sino que sean delimitados alrededor de un contexto de negocio y que sean coherentes.
 2. Evitar un alto volumen de intercomunicación entre microservicios, lo cual ocasionará:
 - Saturación en la red.
 - Dado un nula comunicación entre microservicios, puede originar microservicios-monolíticos.



+

NETFLIX
OSS

iv.ii Descomponiendo aplicaciones monolíticas (u)

- “**Bounded-context**” estrechos. Delimitar contextos relativamente estrechos.
- La cohesión, no la sencillez, ni pequeñez, ni la atomicidad del microservicio, es la única clave para delimitar un contexto de negocio.
- Si dos microservicios requieren colaborar mucho entre si, son altamente dependientes, posiblemente ambos tengan que ser un mismo microservicio.
- Si un microservicio debe depender de otro servicio para satisfacer directamente una solicitud, entonces no es realmente autónomo.



+

iv.ii Descomponiendo aplicaciones monolíticas (v)

- Práctica 8. Descomponiendo ACME HR System
- Analiza la aplicación Acme-HR-System.
- Descompón la aplicación Acme-HR-System en dos microservicios: **8-Employee-Acme-HR-Microservice** y **8-Workstation-Acme-HR-Microservice** cumpliendo con las mismas (casi) interfaces REST de la aplicación Acme-HR-System.



+

NETFLIX
OSS

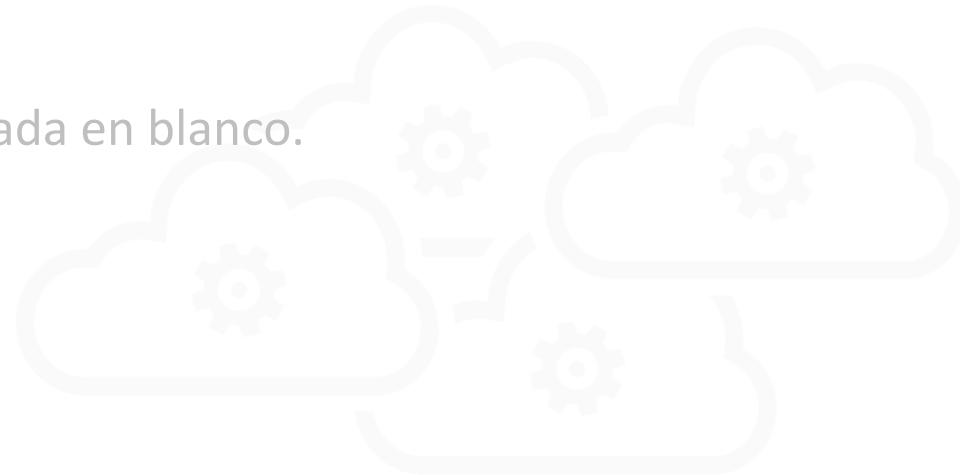
Resumen de la lección

iv.ii Descomponiendo aplicaciones monolíticas

- Comprendemos los puntos a observar para migrar una aplicación monolítica a una orientada a microservicios.
- Aprendimos las recomendaciones a la hora de descomponer aplicaciones monolíticas.
- Verificamos lo que es el Domain-Driven Design y los "Bounded-context".



Esta página fue intencionalmente dejada en blanco.



Microservices



+

NETFLIX
OSS

iv. Arquitectura de Microservicios

- iv.i ¿Qué es la arquitectura orientada a microservicios?
- iv.ii Descomponiendo aplicaciones monolíticas.
- iv.iii **Protocolos ligeros de comunicación para microservicios.**
- iv.iv Aplicaciones “cloud-native”.
- iv.v Twelve-Factor Apps.
- iv.vi Orquestación vs Coreografía.
- iv.vii Patrones de diseño para la nube.
- iv.viii Gestión de Transacciones ACID vs BASE.
- iv.ix Principios de diseño para aplicaciones “cloud-native”.
- iv.x SOA vs APIs vs Microservicios.
- iv.xi API Layer.



+

NETFLIX
OSS

iv.iii Protocolos ligeros de comunicación para microservicios.



Objetivos de la lección

iv.iii Protocolos ligeros de comunicación para microservicios

- Analizaremos los distintos mecanismos de comunicación entre microservicios.
- Comprenderemos las diferencias entre comunicación síncrona y asíncrona.
- Comprenderemos las ventajas y desventajas de cada protocolo de comunicación.
- Implementaremos comunicación asíncrona mediante broker de mensajes ActiveMQ.

iv.iii Protocolos ligeros de comunicación para microservicios (a)

- En una aplicación monolítica, las comunicaciones entre componentes se ejecutan en un único proceso, no necesariamente en un mismo hilo de ejecución, donde los componentes se invocan entre sí mediante llamadas de funciones (o llamadas a métodos) a nivel de lenguaje de forma acoplada o desacoplada.
- Lo más complicado al refactorizar una aplicación monolítica a microservicios es cambiar el mecanismo de comunicación entre los componentes mediante comunicación entre procesos (Inter Process Communication, IPC).



iv.iii Protocolos ligeros de comunicación para microservicios (b)

- En una arquitectura de microservicios, es preferible disminuir la cantidad de comunicaciones que tendrá un microservicio con respecto de los demás.
- No existe una única solución, para evitar las comunicaciones, una posible solución puede ser delimitar los contextos de negocio lo más aislados posibles, de tal grado que no exista comunicación entre microservicios.
- Una aplicación basada en microservicios es un sistema distribuido el cual se ejecuta en varios procesos o servicios e incluso en diferentes servidores, físicos o virtuales donde, cada instancia de un microservicio es un proceso independiente.

iv.iii Protocolos ligeros de comunicación para microservicios (c)

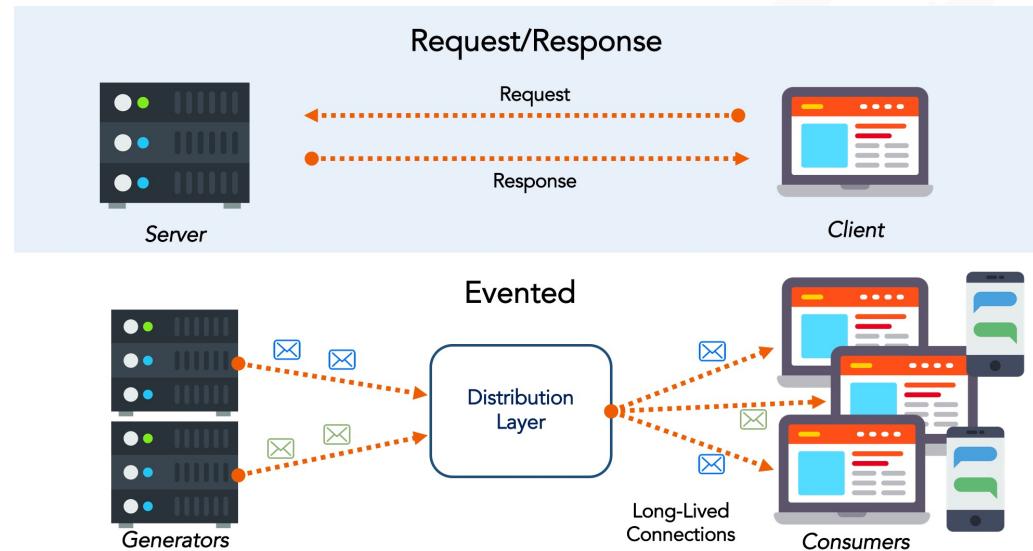
- Dado que cada instancia de un microservicio es un proceso independiente, debe implementar un protocolo de comunicación entre procesos como son HTTP, AMQP, gRPC o TCP/IP, dependiendo de la función de cada microservicio.
- Una sana implementación de microservicios promueve que la intercomunicación entre microservicios sea mediante “**smart endpoints and dumb pipes**” (Puntos de conexión inteligentes y tuberías tontas) fomentando así un diseño desacoplado entre microservicios.

iv.iii Protocolos ligeros de comunicación para microservicios (d)

- En una correcta implementación de microservicios, cada microservicio es responsable de sus propios datos y de implementar su lógica de negocio dado su delimitación de contexto sin embargo, las aplicaciones basadas en microservicios, partiendo de un caso de uso “**end-to-end**”, establecen comunicaciones entre microservicios y prevalecen los protocolos ligeros tales como:
 - HTTP mediante REST en lugar de protocolos complejos como WS-* o SOAP.
 - JMS, AMQP o comunicaciones guiadas por eventos en lugar de orquestadores de procesos de negocio centralizados como ESB.
 - gRPC o “Google open-source remote procedure call”, para llamadas internas entre microservicios.

iv.iii Protocolos ligeros de comunicación para microservicios (e)

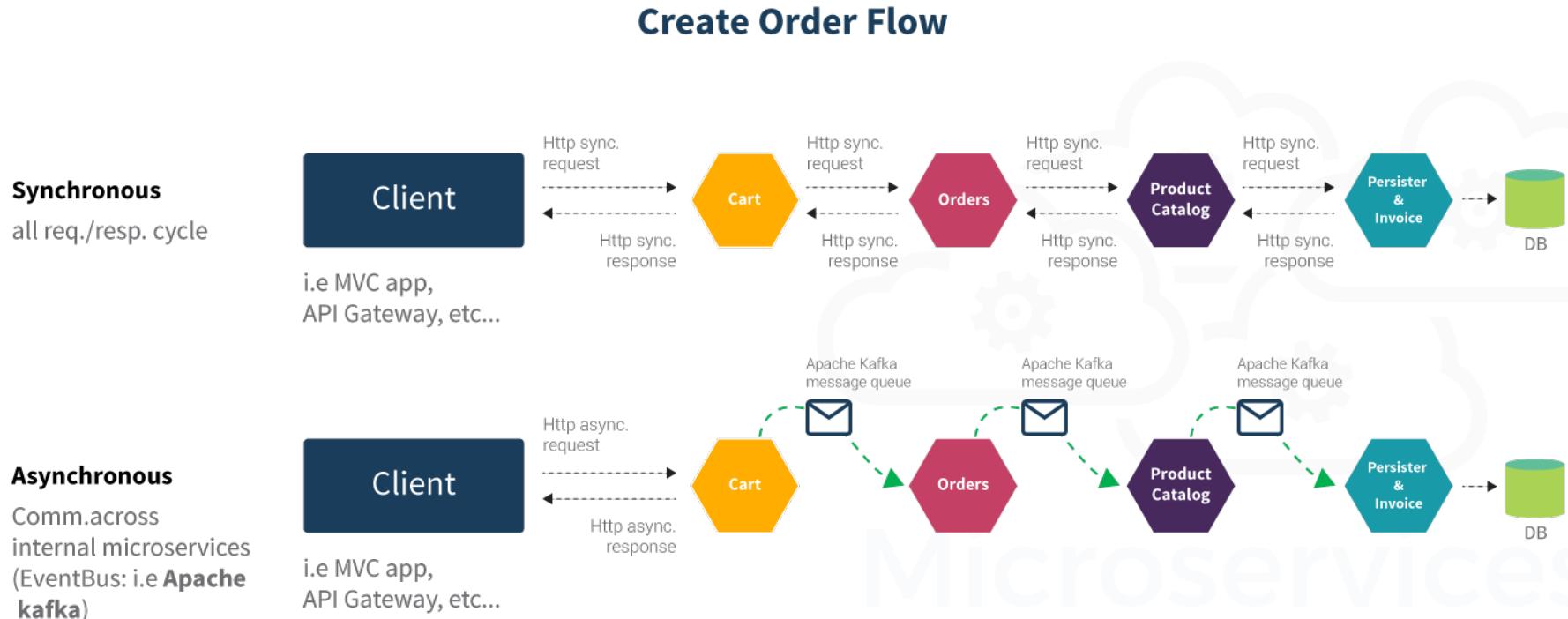
- Habitualmente se sugieren el protocolo HTTP para situaciones “**request-response**” mediante APIs REST y mensajería asíncrona ligera para comunicación de tipo “**fire-and-forget**”.



iv.iii Protocolos ligeros de comunicación para microservicios (f)

- La comunicación entre microservicios se divide en dos ejes principales:
 1. Comunicación síncrona o asíncrona.
 - **Protocolos síncronos:** HTTP / HTTPS.
 - **Protocolos asíncronos:** JMS (TCP/IP), AMQP, entre otros.
 2. Comunicación a un único receptor o a varios receptores.
 - **Único receptor:** Colas de mensajes o “**queues**” que implementan el patrón “**Producer/Consumer**” y balanceo de carga (es posible que existan múltiples “**listeners**” de la “**queue**” pero sólo uno recibe el mensaje).
 - **Varios receptores:** La comunicación entre varios receptores debe ser asíncrona, Topicos o “**topics**” que implementan el patrón “**Publish/Subscribe**” o arquitectura controlada por eventos.

iv.iii Protocolos ligeros de comunicación para microservicios (g)



iv.iii Protocolos ligeros de comunicación para microservicios (h)

- Una aplicación basada en arquitectura orientada a servicios acostumbra utilizar una combinación entre los tipos de comunicación síncrona y asíncrona.
- Lo más común en la comunicación desde la aplicación cliente, consumidora de un microservicio es mediante protocolo síncrono como HTTP/HTTPS, mientras que, internamente, los microservicios se comunican entre sí de forma asíncrona, lo cual habilita su independencia, obligando su autonomía, facilidad de escalabilidad, reusabilidad, tolerancia a fallos y disponibilidad.



+

iv.iii Protocolos ligeros de comunicación para microservicios (i)

- Ventajas de comunicación síncrona:
 - Baja complejidad en el diseño.
 - Facilidad para el manejo de errores.
 - Recepción de respuestas en tiempo real (on-the-go).
- Desventajas de comunicación síncrona:
 - El servicio debe estar disponible todo el tiempo, si el servicio no está disponible, el hilo “caller” puede bloquearse por un tiempo, hasta que ocurra un error por “time-out”, causando problemas de performance.
 - Posibilidad de propagación no controlada de errores de comunicación.
 - Respuestas lentas debido a que los servicios deben esperar a que termine de recibir la respuesta de los demás servicios involucrados.
 - Necesidad de un protocolo orientado a conexión (TCP).

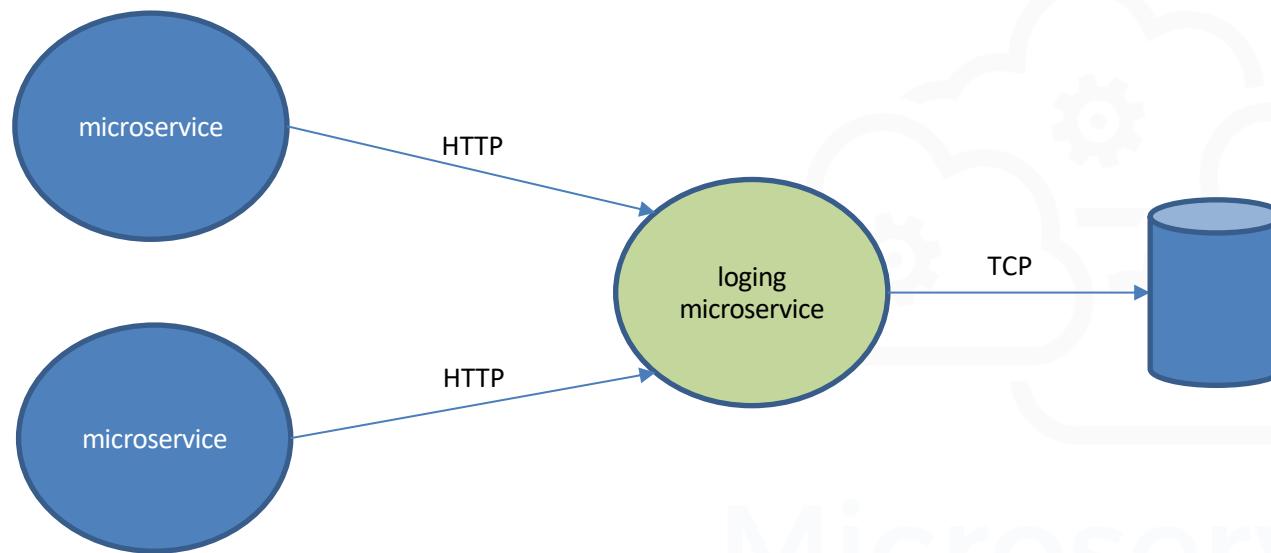


iv.iii Protocolos ligeros de comunicación para microservicios (j)

- Autonomía de microservicios (a).
- Tal como se ha mencionado, el punto más crítico de una aplicación basada en microservicios es como integrar/comunicar los microservicios entre sí.
- Idealmente es preferible evitar las comunicaciones entre si, utilizar código reusable (librerías) en lugar de depender de llamadas a microservicios de uso general.

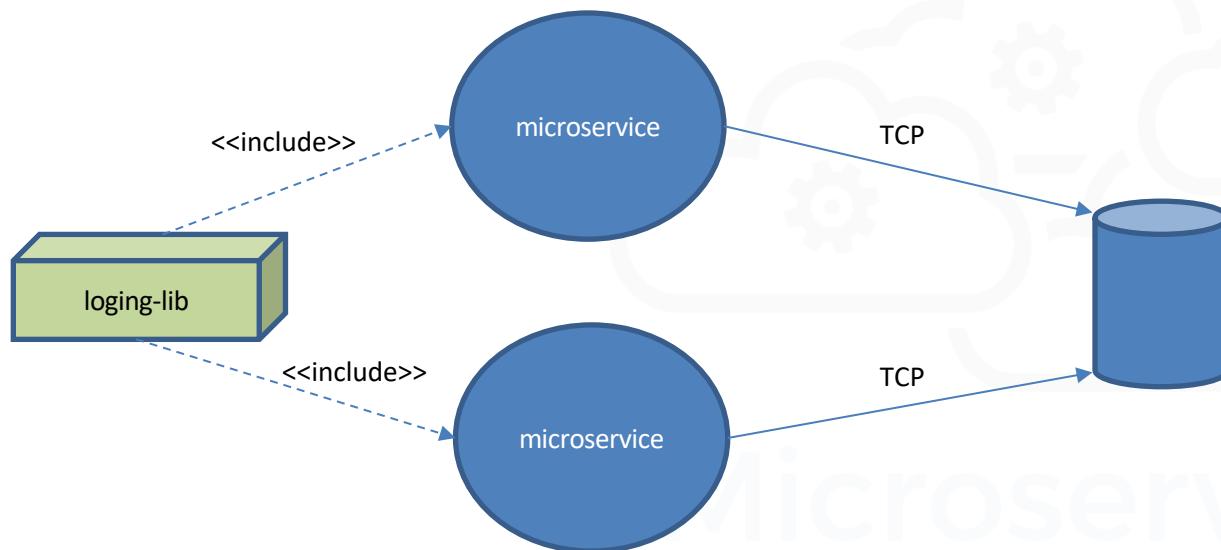
iv.iii Protocolos ligeros de comunicación para microservicios (k)

- Autonomía de microservicios (b).



iv.iii Protocolos ligeros de comunicación para microservicios (I)

- Autonomía de microservicios (c).



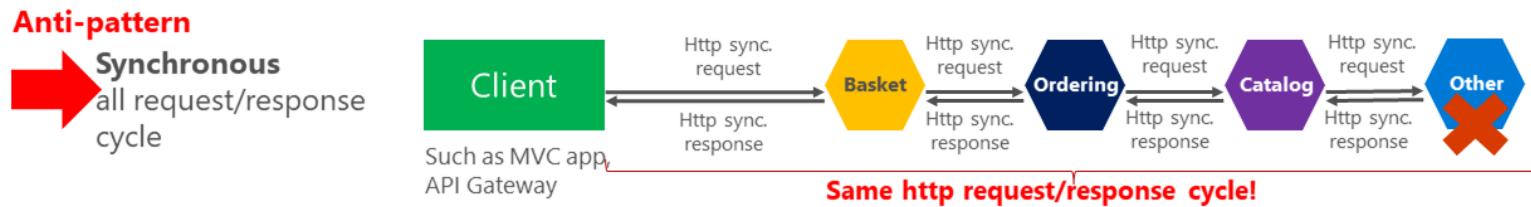
iv.iii Protocolos ligeros de comunicación para microservicios (m)

- Autonomía de microservicios (d).
- Cuando sea necesaria la comunicación entre microservicios, dicha comunicación, como regla fundamental debe ser asíncrona. Siendo posible jamás depender de una comunicación síncrona.
- La comunicación síncrona entre microservicios promueve una alta dependencia entre microservicios, lo cual evita su independencia y autonomía, indistintamente de que el despliegue de los microservicios sea autónomo e independiente.

iv.iii Protocolos ligeros de comunicación para microservicios (n)

- Autonomía de microservicios (e).

Comunicación síncrona vs asíncrona entre microservicios

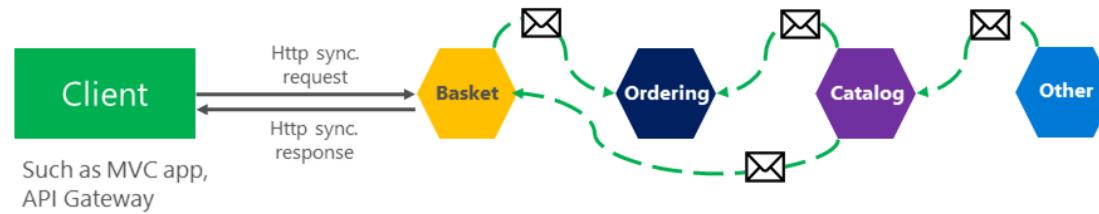


iv.iii Protocolos ligeros de comunicación para microservicios (ñ)

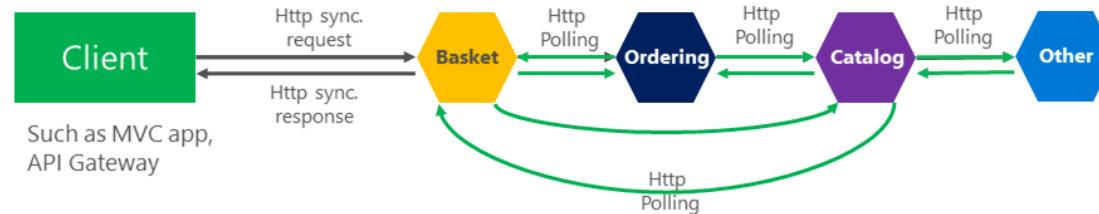
- Autonomía de microservicios (f).

Comunicación síncrona vs asíncrona entre microservicios

Asynchronous
Comm. across internal
microservices
(EventBus: like **AMQP**)



"Asynchronous"
Comm. across
internal microservices
(Polling: **Http**)

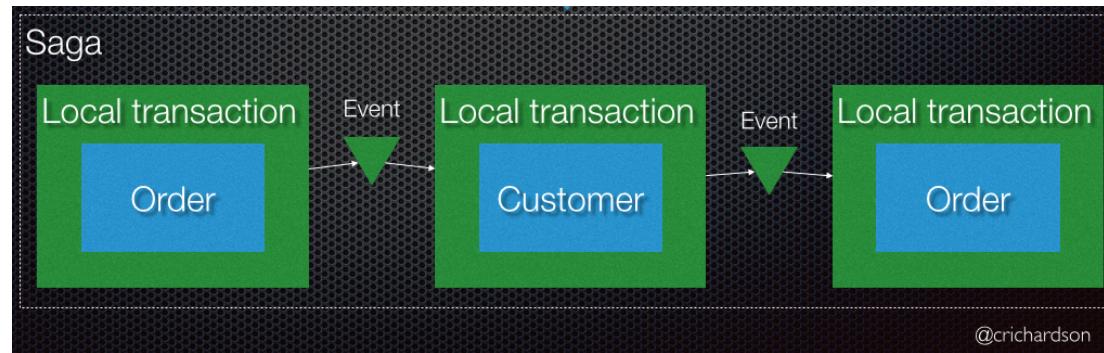




iv.iii Protocolos ligeros de comunicación para microservicios (o)

- Autonomía de microservicios (g).
- En caso que un microservicio deba accionar alguna operación en otro microservicio, como por ejemplo, una actualización de datos, ejecute dicha acción orientada a eventos (o implementando Saga Pattern), es decir, de forma asíncrona.

iv.iii Protocolos ligeros de comunicación para microservicios (p)





iv.iii Protocolos ligeros de comunicación para microservicios (q)

- Ventajas de comunicación asíncrona:
 - Sin necesidad de protocolos orientados a conexión ya que la información viaja mediante “brokers” de mensajes.
 - Sin necesidad de esperar una respuesta del lado del consumidor, no hay problemas de performance.
 - El productor (quien envía el mensaje) no necesita saber quien o cuantos son los consumidores (quien recibe el mensaje) del mensaje. Existe un bajo acoplamiento entre servicios.
 - No es necesaria una alta disponibilidad del servicio consumidor.
 - La comunicación asíncrona mejora la tolerancia a fallos, aísla los fallos.



iv.iii Protocolos ligeros de comunicación para microservicios (r)

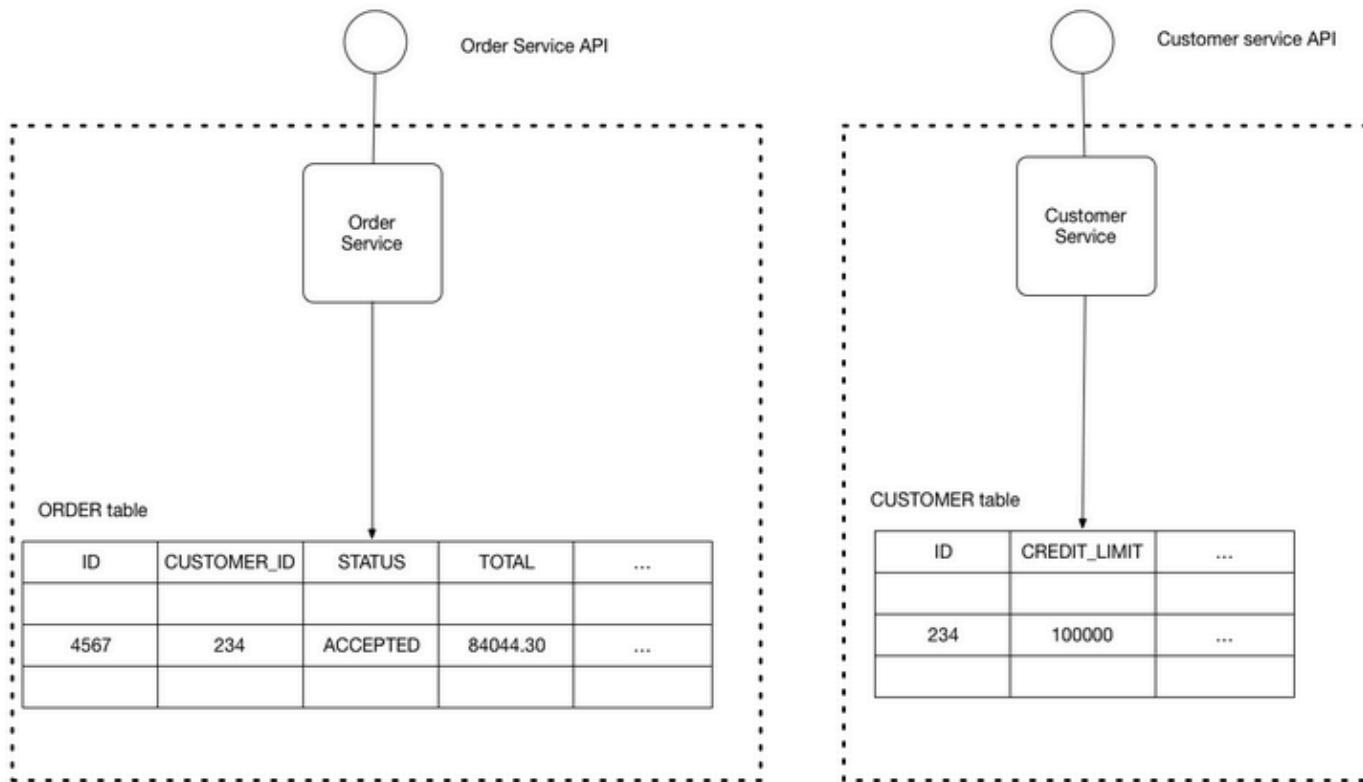
- Desventajas de comunicación síncrona:
 - Mayor complejidad en el diseño de sistemas distribuidos mediante comunicación asíncrona.
 - Dificultad de manejar errores en componentes con comunicación asíncrona.
 - Alto acoplamiento con el "broker" de mensajes.
 - Alto costo de latencia si la bandeja (cola) de mensajes alcanza su límite.
 - Alto costo para el monitoreo y "debug" de la infraestructura de mensajes.



iv.iii Protocolos ligeros de comunicación para microservicios (s)

- Autonomía de microservicios (i).
- Cuando un microservicio requiera datos para operar, cuyo propietario de los datos es otro microservicio, no dependa de la solicitud de dichos datos a ese otro microservicio. Replique los datos en ambos microservicios (Database per Service Pattern).

iv.iii Protocolos ligeros de comunicación para microservicios (t)





iv.iii Protocolos ligeros de comunicación para microservicios (u)

- Práctica 9. Comunicación asíncrona
- Analiza la aplicación **9-Embedded-Broker-Service**, **9-Order-Producer-Microservice** y **9-Order-Consumer-Microservice**.
- Ingresar a la ruta: **{tu-workspace}/9-Embedded-Broker-Service**
- Importar el proyecto **9-Embedded-Broker-Service** en STS.
- Define el Bean **BrokerService** para exponer el broker ActiveMQ embebido como un broker de mensajes JMS en local a través del conector “**tcp://localhost:61616**”.



iv.iii Protocolos ligeros de comunicación para microservicios (v)

- **Práctica 9. Comunicación asíncrona**
- Ejecuta la clase principal del proyecto, anotada con **@SpringBootApplication**, se deberá visualizar una salida en consola similar a lo siguiente:

```
No active profile set, falling back to default profiles: default
Using Persistence Adapter: KahaDBPersistenceAdapter[/Users/xvhx/mgt-ws/ws-curso-micros
JMX consoles can connect to service:jmx:rmi://jndi/rmi://localhost:1099/jmxrmi
KahaDB is version 6
PListStore: [/Users/xvhx/mgt-ws/ws-curso-microservicios-spring-cloud-netflix-profesor/9-
Apache ActiveMQ 5.15.8 (localhost, ID:trial-62922-1558910531705-0:2) is starting
Listening for connections at: tcp://localhost:61616
Connector tcp://localhost:61616 started
Apache ActiveMQ 5.15.8 (localhost, ID:trial-62922-1558910531705-0:2) started
For help or more information please see: http://activemq.apache.org|
LiveReload server is running on port 35729
Started EmbeddedBrokerServiceMicroservice in 0.162 seconds (JVM running for 29.44)
Condition evaluation unchanged
```



+

iv.iii Protocolos ligeros de comunicación para microservicios (w)

- Práctica 9. Comunicación asíncrona
- Ingresar a la ruta: **{tu-workspace}/9-Order-Consumer-Microservice**
- Importar el proyecto **9-Order-Consumer-Microservice** en STS.
- Analiza la clase **Order** del paquete
com.consulting.mgt.springboot.practica9.embedded.broker.service.model.
- Analiza la clase de configuración **ActiveMQConfig** del paquete
com.consulting.mgt.springboot.practica9.embedded.broker.service._config; habilita mensajería JMS mediante la anotación **@EnableJms**.



+

iv.iii Protocolos ligeros de comunicación para microservicios (x)

- Práctica 9. Comunicación asíncrona
- Define una constante String **ORDER_QUEUE** con el valor “**order-queue**” en la clase **ActiveMQConfig**.
- Define el Bean **OrderConsumer**, mediante configuración por anotaciones, directamente sobre la clase **OrderConsumer** en el paquete **com.consulting.mgt.springboot.practica9.embedded.broker.service.consumer**.
- Define un listener JMS, mediante la anotación **@JmsListener** que escuche mensajes directamente de la “**queue**” destino definida por la constante **ORDER_QUEUE**. Procesa el mensaje **Order** mediante la anotación **@Payload** y loggea el cuerpo del mensaje.



iv.iii Protocolos ligeros de comunicación para microservicios (y)

- **Práctica 9. Comunicación asíncrona**
- Ingresa en una consola o terminal a la ubicación **{tu-workspace}/9-Order-Consumer-Microservice** y, compila y empaqueta la aplicación mediante el comando: “**mvn clean package**”.
- Posteriormente ejecuta la aplicación mediante el comando “**java -jar target/9-Order-Consumer-Microservice-0.0.1-SNAPSHOT.jar**”.
- Es posible iniciar uno o más instancias del servicio **9-Order-Consumer-Microservice**, lo cual mejoraría la alta disponibilidad del servicio.



+

iv.iii Protocolos ligeros de comunicación para microservicios (z)

- Práctica 9. Comunicación asíncrona
- Ingresar a la ruta: **{tu-workspace}/9-Order-Producer-Microservice**
- Importar el proyecto **9-Order-Producer-Microservice** en STS.
- Analiza la clase **Order** del paquete
com.consulting.mgt.springboot.practica9.embedded.broker.service.model.
- Analiza la clase de configuración **ActiveMQConfig** del paquete
com.consulting.mgt.springboot.practica9.embedded.broker.service._config; habilita mensajería JMS mediante la anotación **@EnableJms**.



+

NETFLIX
OSS

iv.iii Protocolos ligeros de comunicación para microservicios (a')

- Práctica 9. Comunicación asíncrona
- Define el Bean **OrderProducer**, mediante configuración por anotaciones, directamente sobre la clase **OrderProducer** en el paquete **com.consulting.mgt.springboot.practica9.embedded.broker.service.producer**.
- Define inyección de dependencias de un objeto **JmsTemplate** sobre el Bean **OrderProducer**.
- Define un método para enviar un mensaje de tipo **Order** mediante el template **JmsTemplate** definido.



+

NETFLIX
OSS

iv.iii Protocolos ligeros de comunicación para microservicios (b')

- Práctica 9. Comunicación asíncrona
- Define el Bean controlador REST sobre la clase **OrderController**, mediante configuración por anotaciones, directamente sobre la clase **OrderController** en el paquete **com.consulting.mgt.springboot.practica9.embedded.broker.service.restcontroller**.
- Define inyección de dependencias del objeto **OrderProducer** sobre el Bean **OrderController**.
- Define un “**handler method**” que reciba las peticiones HTTP entrantes mediante la URI **/place-order** a través del método **GET** y envía un mensaje de tipo **Order** utilizando la dependencia **OrderProducer**.



iv.iii Protocolos ligeros de comunicación para microservicios (c')

- **Práctica 9. Comunicación asíncrona**
- Ingresa en una consola o terminal a la ubicación **{tu-workspace}/9-Order-Producer-Microservice** y, compila y empaqueta la aplicación mediante el comando: “**mvn clean package**”.
- Posteriormente ejecuta la aplicación mediante el comando “**java -jar target/9-Order-Producer-Microservice-0.0.1-SNAPSHOT.jar**”.



iv.iii Protocolos ligeros de comunicación para microservicios (d')

- **Práctica 9. Comunicación asíncrona**
- Desde un cliente HTTP, ejecuta una petición GET al servicio expuesto por el servicio **/place-order**, mediante la URL <http://localhost:8080/place-order>.
- Como ejemplo, mediante consola, ejecuta el comando “**http localhost:8080/place-order**” (requiere **httpie** instalado). Es posible utilizar **cURL** mediante el comando “**curl -i -X GET <http://localhost:8080/place-order>**”.
- Analiza el comportamiento de los servicios y la comunicación asíncrona.



Resumen de la lección

iv.iii Protocolos ligeros de comunicación para microservicios

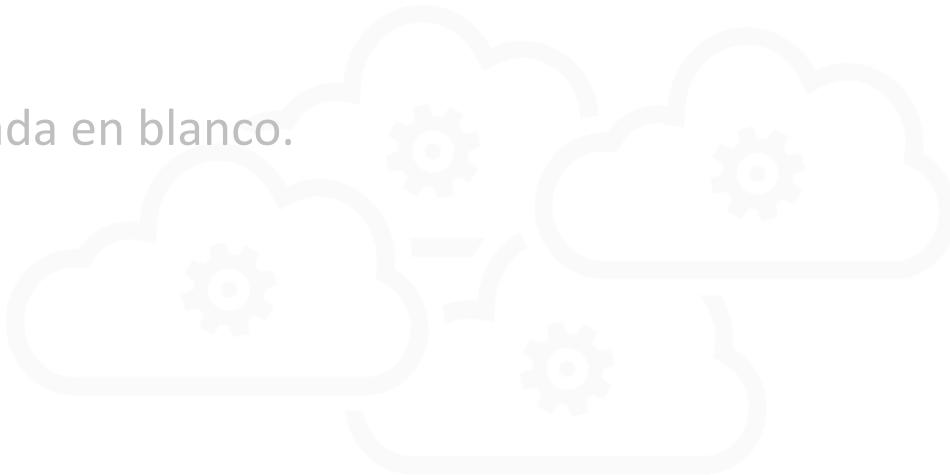
- Comprendemos los distintos mecanismos de comunicación entre microservicios diferenciando entre síncronos y asíncronos.
- Analizamos y discutimos las diferencias entre comunicación síncrona y asíncrona y cuál es la recomendación para los distintos casos de uso.
- Aplicamos comunicación asíncrona mediante broker de mensajes ActiveMQ entre dos microservicios.



+

NETFLIX
OSS

Esta página fue intencionalmente dejada en blanco.



Microservices