Desarrollo Basado en Componente y Servicios

Introducción al Laboratorio - Conceptos Previos

1. Introducción

En este documento vamos a exponer el trabajo a realizar en el laboratorio de la asignatura Desarrollo Basado en Componentes y Servicios (DBCS). También indicaremos y describiremos las tecnologías a usar. La instalación del software necesario lo iremos planteando según vayamos avanzando en las prácticas.

2. Contenido del laboratorio

Vamos a construir una aplicación basada en microservicios, tecnología muy utilizada actualmente. Esta aplicación sigue una arquitectura basada en cliente (consumidor de los servicios o frontend) – servidor (los que proporcionan los servicios o backend). El esquema general de la aplicación se muestra en la Figura 1.

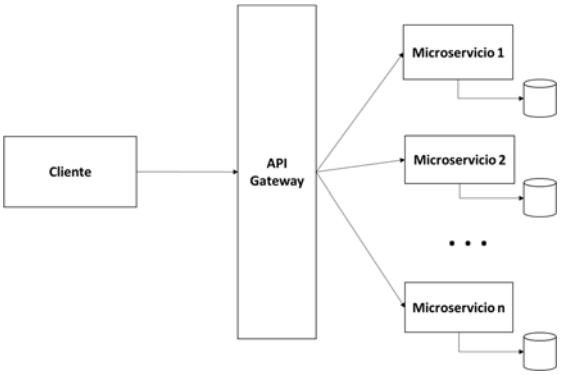

Figura 1. Esquema general de la aplicación a construir

Vamos a describir brevemente cada parte de la aplicación.

1.1. Microservicio

Un microservicio o, mejor, una arquitectura basada en microservicios es una forma de estructurar una aplicación como una colección de servicios, en vez de como un sistema monolítico. Se puede considerar como una evolución de la arquitectura basada en servicio (SOA). Una descripción de las diferencias entre ambos excede el objetivo de este tutorial, pero es fácil de encontrar en internet; se deja para el alumno.

El objetivo (y, por lo tanto, las ventajas) de los microservicios es que sean o puedan ser:

- Poco o nada acoplados.
- Desplegados de manera independiente.
- Creados con diferentes tecnologías.
- Basados en el modelo de negocio (bussines capabilities), es decir, cada servicio desarrolla funcionalidades diferentes. La aplicación se descompone en sus funciones principales. Se especializan, por lo tanto, en resolver un problema concreto.

- Sencillos de desarrollar. En contra de lo necesario para crear una aplicación única, su desarrollo se puede realizar con equipos de personas pequeños. También esto hace que su desarrollo sea más ágil y rápido.
- Mantenibles. Al desarrollarse, probarse y desplegarse por separado, los errores y, con ello, el mantenimiento de la aplicación se limita al servicio que falle; el resto puede seguir funcionando sin problema.
- Escalables. Permiten desarrollar aplicaciones más fácilmente adaptables a una necesidad concreta.
- Reutilizables. Un servicio desarrollado para una determinada aplicación puede ser usado para otra que necesite los mismos servicios. En este sentido, se pueden considerar como componentes, que pueden ser reutilizados en su totalidad o con ligeras modificaciones.

Frente a estas ventajas, el uso de microservicios presenta una serie de retos o dificultades, muchos derivados de que el sistema final es más complejo que el de una aplicación monolítica:

- Diseño. Dividir una aplicación en partes, requiere un esfuerzo previo importante para que esas partes sean las correctas. No solo se dividen las funcionalidades, sino todo lo referente a datos. También hay que identificar las dependencias entre servicios.
- Integración. Las pruebas de integración se vuelven más complejas. De igual modo la depuración de la aplicación completa.
- Comunicación. Hay que proveer, por un lado, de una forma de comunicación entre servicios. Por otra, una forma de comunicar el cliente con servicios que han sido creados y desplegados de manera independiente y con distintas tecnologías. También hay que tener en cuanto su eficiencia, ya que el aumento de microservicios puede llevar consigo un gran incremento de esta comunicación.
- Consistencia de los datos, al estar también distribuidos entre componentes.
- Recursos necesarios. El incremento de servicios lleva consigo un incremento de los recursos necesarios.
- Coordinación y gestión. Hay que proporcionar servicios para la coordinación, gestión y monitorización de los diferentes servicios.

En estas prácticas vamos a ver algunas soluciones tecnológicas para solucionar estos retos. No hay que olvidar que nuestro objetivo está centrado en el aprendizaje de una forma moderna de estructurar aplicaciones, como es la arquitectura basada en microservicios, no describir tecnologías. Por eso, se han escogido, dentro de las utilizadas de forma real en el mundo empresarial, aquellas que mejor se adapten a nuestro objetivo y los límites temporales para realizar el trabajo práctico.

En estas prácticas desarrollaremos microservicios desplegados como Servicios Web de tipo REST (Representational State Transfer). Por lo tanto, accederemos a ellos mediante llamadas HTTP. Estos servicios serán incluidos en un Contenedor (Container) que encapsula las funcionalidades y elementos del servicio, haciéndoles más fácilmente accesibles y desplegables. Un poco más adelante en este documento concretaremos los detalles tanto del servicio web, como de su “contenerización”, así como las tecnologías que usaremos.

1.2. API Gateway

En este laboratorio vamos a seguir el patrón de puerta de enlace de API o API Gateway.

Es el encargado de comunicar al cliente con los microservicios. Es decir, es el único punto de entrada (punto de conexión o URL) desde el que los clientes pueden acceder a las funcionalidades

proporcionadas por los distintos microservicios. Soluciona, pues, el problema de comunicación de un cliente con servicios desarrollados con distintas tecnologías y desplegados de manera independiente. Hace que esta comunicación sea más sencilla y eficiente que si conectáramos directamente el cliente con los microservicios.

Aquí vamos a usar un único punto de entrada, es decir, un único Gateway, pero es un sistema real puede haber varios, cada uno adaptado a las distintas características de distintos tipos de clientes, como, por ejemplo, si la conexión es desde un ordenador o desde un móvil.

Su principal función es *redirigir y enrutar las solicitudes a los puntos de conexión de los microservicios*. Sin embargo, aunque las funcionalidades son dependientes de la tecnología usada, entre algunas adicionales que pueden tener podemos destacar que puede ser el encargado de:

- Agregar las solitudes de funcionalidades de distintos microservicios que vienen de una única petición del cliente.
- Descargar de algunas operaciones a los microservicios, como, por ejemplo, la autenticación.
- Cuestiones transversales como cachés, equilibrios de carga, registros y seguimiento de peticiones, etc.
- Cifrado de datos.
- Políticas de uso
- …

Dentro de las distintas alternativas que hay, aquí vamos a usar **Kong** (https://konghq.com/kong/). Lo describiremos brevemente un poco más adelante.

1.3. Cliente

Será el encargado de consumir los microservicios. Dada la naturaleza de estos, proporcionados como servicios web de tipo REST, cualquier tecnología capaz de usar ese tipo de servicios podía ser usada para esta parte.

Dentro de las múltiples disponibles, aquí vamos a usar **Angular** (https://angular.io/). Es una de las tecnologías basadas en Javascript más populares para crear “front-ends”, es decir, la parte del cliente de una aplicación web.

3. Desarrollo del laboratorio

Lo descrito en el apartado anterior será la aplicación final que construiremos, pero se llegará a ella de manera gradual. El laboratorio de divide en las siguientes tres partes:

- **Parte I: Servicio Web REST y Cliente**. Se construirá un servicio web de tipo REST (API REST) y un cliente para acceder a él. Para construir el servicio web se usará **Spring Boot** (https://spring.io/projects/spring-boot), basado en Java. Este framework simplifica mucho el desarrollo y prueba de este tipo de servicios. Para el cliente, como se ha comentado, usaremos **Angular**.
- **Parte II: Contenedores**. El servicio web se “alojará” en un Contenedor. Para ello usaremos **Docker** (https://www.docker.com/), uno de los gestores de contenedores software más usados actualmente. El contenedor se desplegará en un servidor y será accedido por el cliente a través de un API Gateway.

- **Parte III: Aplicación Final.** Al sistema de la Parte II se añadirán nuevos servicios, actualizando adecuadamente el cliente y el API Gateway. La idea es desarrollar algún nuevo microservicio, usando, para ello, una tecnología diferente a Spring Boot (Java). El objetivo es ver como podemos desarrollar, desplegar y acceder a distintos servicios, creados con distintas tecnologías de manera totalmente transparente.

4. Descripción de las principales tecnologías

Para tener una mejor visión global de las prácticas vamos a realizar una breve descripción de las principales tecnologías que usaremos a lo largo de las prácticas. En el momento de usarlas, profundizaremos en esta descripción.

4.1. Spring Boot (https://spring.io/projects/spring-boot)

Es una de las tecnologías dentro del ecosistema Spring (https://spring.io/), que es un framework de código abierto que facilita el desarrollo de aplicaciones Java; quizás uno de los usos más extendidos de Spring es el apoyo al desarrollo mediante JEE (Java Enterprise Edition), facilitando el desarrollo de aplicaciones empresariales y el uso de los componentes EJB (Enterprise Jaba Beans).

Spring Boot facilita el trabajo de configurar las aplicaciones basadas en Spring. El desarrollo de una aplicación con Spring tiene tres pasos fundamentales (https://www.arquitecturajava.com/que-es-spring-boot/):

1. Seleccionar los jars con Maven
2. Crear la aplicación
3. Desplegar en el servidor

Pues bien, Spring Boot nace con la idea de simplificar los pasos 1 y 3 y que, como desarrolladores, nos podamos centrar en el paso 2, que es el que interesa.

En nuestro caso, veremos que nos permite crear y configurar (https://start.spring.io/) de manera muy sencilla y rápida el esqueleto de un servicio web de tipo REST con Spring, al tiempo que resuelve todas las bibliotecas y dependencias necesarias (paso 1). Además, incorpora un servidor de aplicaciones, que nos va a permitir desplegar y probar (paso 3) el servicio web creado (paso 2) de manera simple, sin necesidad de tener que utilizar servidores externos. El servidor integrado permite la ejecución de nuestro servicio web empaquetado como *.jar*, sin necesidad de tener que empaquetarlo en un *.war* (necesario si tuviéramos que desplegarlo en un servidor externo), proceso mucho más complejo. Esto también va a facilitar el alojamiento de nuestro servicio web en un contenedor tipo Docker, para poder ser usado como microservicio, siguiendo la arquitectura mostrada en la Figura 1.

Por supuesto, la configuración inicial de la aplicación podrá ser modificada posteriormente para, por ejemplo, añadir nuevas dependencias o cambiar el puerto en el que la aplicación escucha peticiones; Spring Boot se encargará de manera automática, en esos ejemplos, de resolver las nuevas dependencias o modificar el puerto en el servidor.

4.2. Angular (https://angular.io/)

Angular es una plataforma que permite el desarrollo de aplicaciones web dinámicas en la parte del cliente usando HTML y javascript. Hay dos enfoques extremos a la hora de crear páginas web dinámicas (https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps):

- **“Round-Trip Model” (RTP)**: la creación de las páginas recae enteramente en el servidor. Cualquier petición (request) del usuario implica una llamada al servidor, que genera y devuelve una nueva página al navegador. Es la forma en que se ha operado al crear aplicaciones web basadas en JEE. Es también la forma de trabajar más tradicionalmente usada.
- **“Single-Page Application” (SPA)**: el servidor sirve la primera página y a partir de ahí todo el procesamiento dinámico (creación de nuevas vistas) se realiza en el cliente.

La mayoría de las aplicaciones web se mueven en enfoques intermedios, con la inclusión de pequeños programas JavaScript. Angular está pensado para proyectos complejos basados en la opción SPA.

Está basado en el uso de componentes. Esto permite encapsular mejor la funcionalidad, facilitando la construcción y el mantenimiento de las aplicaciones. Los componentes Angular son las “piezas” básicas mediante las que construimos la aplicación y que se pueden comunicar entre sí. El objetivo de cada componente es encapsular una determinada funcionalidad de la aplicación, incluso, podríamos decir que, en general, encapsulan todo lo relacionado con una determinada vista de nuestra aplicación; desde este punto de vista, podría ser reutilizados, pero, como veremos, no es el objetivo principal de los componentes Angular.

Otra característica importante de Angular es que favorece el uso del patrón MVC (Modelo-Vista-Controlador). Las vistas se implementarán mediante ficheros html, llamados en lenguaje Angular “templates” o plantillas, (https://howtodoinjava.com/angular/angular-templates-and-views/) y css; aunque lo normal es que la vista se implemente en un fichero html, es conveniente indicar que también se pueden incluir templates (código html) dentro de los controladores (“Inline template”, https://howtodoinjava.com/angular/angular-templates-and-views/). Los controladores serán ficheros Typescript (https://www.typescriptlang.org/) y contendrán la lógica del componente. Aunque Angular usa typescript para el desarrollo, cuando la aplicación sea construida (compilada), este código será traducido a JavaScript.

En nuestro caso lo vamos a usar para construir una aplicación web cliente de los servicios, lo que se conoce con el anglicismo “front-end”.

4.3. Contenedores Software – Docker (https://www.docker.com/)

Para entender mejor que es un contenedor software vamos a partir de un problema habitual en el desarrollo de aplicaciones. Cada aplicación que se desarrolla está creada usando una determinada tecnología y con unas dependencias en cuanto a las bibliotecas usadas. Pues bien, algo habitual es que una vez desarrollada, modificaciones en la tecnología usada o en las dependencias (ej. actualizaciones) hacen que la aplicación deje de funcionar y haya que rehacerla. Otro problema es cuando se quiere distribuir esa aplicación, al movernos de entorno (máquina), la aplicación puede dejar de funcionar por las diferencias en la configuración, sistema operativo o hardware.

Pues bien, el contenedor software proporciona la solución a ese problema, ya que es un empaquetamiento que aloja todas las dependencias que la aplicación necesita para ser ejecutada: el propio código, las librerías del sistema, el entorno de ejecución o cualquier tipo de configuración. Este empaquetamiento independiza la aplicación tanto del entorno hardware donde se ejecuta, como de la configuración software de éste.

En palabras de Solomon Hykes, creador de Docker: “utilizando contenedores para ejecutar tu código solventamos el típico quebradero de cabeza de que estés usando una versión de Python 2.7 diferente en local o en el entorno de pruebas, pero en producción la versión sea totalmente distinta como Python 3 u otras dependencias propias del entorno de ejecución, incluso el sistema operativo. Todo lo que necesitas está dentro del propio contenedor y es invariable”.

Para entenderlo mejor, y de ahí su nombre, se puede usar el símil de los contendores donde se transportan productos. No importa su contenido, como cumple con unos estándares, puede ser

transportado por distintos medios completamente diferentes entre sí (camión, barco, etc.). Además, puede ser transportado de uno en uno o apilando varios a la vez (símil de los microservicios).

Así como las máquinas virtuales son una virtualización de los sistemas hardware, los contendores son una virtualización de los sistemas software.

Dentro de las distintas alternativas para crear contenedores software, Docker fue una de las primeras y es actualmente una de las más utilizadas. Además, se puede integrar con diferentes herramientas de infraestructura, entre las cuales están las más importantes de computación en la nube como son Amazon Web Services, Google Cloud Computing o Microsoft Azure.

Es un proyecto de código abierto desarrollado para permitir la automatización del proceso de creación, diseño y mantenimiento de contenedores, así como su despliegue sobre un sistema host. La tecnología Docker está basada en el kernel de Linux y características que permiten aislar recursos de éste, como Cgroups y namespaces. Construida inicialmente sobre Linux Containers (LXC), se fue progresivamente separando de esta tecnología para mejorar la experiencia del usuario, es decir, facilitar las operaciones relacionadas con la creación y despliegue de contenedores.

4.4. Kong API Gateway (https://konghq.com/kong/)

Es una de las alternativas más populares para crear un API Gateway. Es un proyecto de código abierto creado sobre el lenguaje Lua. Sus principales características son:

- Fácil escalado.
- Modular. Se puede ampliar agregando nuevos complementos, que se configuran fácilmente. Incluso podremos crear nuestros propios complementos
- Se puede ejecutar tanto en remoto (cloud) como en local. No necesita un servidor
- Buen rendimiento.

Entre los complementos que podemos usar, y que proporciona servicios adicionales, podemos destacar:

- Autenticación
- Control de tráfico
- Análisis y monitorización de operaciones
- Transformación o adaptación de protocolos

5. IDE

Para facilitar la creación del código vamos a necesitar el uso de un IDE. Aunque no son los únicos (se podría usar Eclipse), desde la asignatura aconsejamos el uso de los dos siguientes:

- **Visual Studio Code, VSC** (https://code.visualstudio.com/). IDE ligero y de uso gratuito, con multitud de extensiones que facilitan el desarrollo del software que vamos a realizar durante las prácticas.
- **IntelliJ Idea** (https://www.jetbrains.com/es-es/idea/)**.** Pertenece a la empresa Jet Brains. En principio es de pago, pero tiene versiones gratuitas (Community Edition). Aunque soporta distintos lenguajes, está diseñado para trabajar con Java, siendo uno de los más potentes para trabajar con esta tecnología. No necesita plugins para desarrollar en este lenguaje, a diferencia de VSC.

Ambos han sido usados para el desarrollo de las prácticas que se plantearán. Sin embargo, para los tutoriales, las figuras serán imágenes de VSC. Hacer lo mismo en IntelliJ es sencillo.

6. Despliegue Final de la Aplicación

Se intentará, si hay tiempo, desplegar los diferentes microservicios en un servidor real. Para ello, se utilizará Okteto (https://okteto.com/). Se trata de una herramienta que se encarga de todo el despliegue de una aplicación contenerizada. Por otro lado, permite recibir y visualizar feedback de nuestra aplicación mediante logs, métricas, tratamiento de errores, etc.

La herramienta es gratuita para los desarrolladores utilizando la versión en el Cloud de Okteto y, además, permite compartir los namespaces con los compañeros.

Para poder acceder e iniciar sesión en Okteto es necesaria una cuenta de GitHub (https://github.com/), por lo que se recomienda crearla si todavía no se dispone de ella.

Por otro lado, está la versión para estudiantes que, al registrarse con la cuenta de la Universidad permite acceder al pack con un gran número de herramientas orientadas al desarrollo de manera gratuita: https://education.github.com/pack