

Trabajo Fin de Grado:

Arquitecturas basadas en microservicios.

Autor: Manuel Pérez-Herrera Cuadrillero

Tutor: Santiago Pavón Gómez

Tribunal:

Presidente: Santiago Pavón Gómez

Vocal: Joaquín Luciano Salvachúa Rodríguez

Secretario: Gabriel Huecas Fernández-Toribio

Suplente: Juan Quemada Vives

Fecha de lectura:

Calificación:

Grado en Ingeniería de Tecnologías y Servicios de
Telecomunicación

Trabajo Fin de Grado:

**ARQUITECTURAS
BASADAS EN
MICROSERVICIOS**

Manuel Pérez-Herrera Cuadrillero

Resumen

En este Trabajo Fin de Grado se ha realizado un estudio sobre una nueva tendencia emergente en el desarrollo de aplicaciones web. Esta tendencia se basa en un nuevo modelo de arquitectura conocido como microservicios.

La utilización de arquitecturas basadas en microservicios, supone un nuevo punto de vista en el desarrollo de aplicaciones web completamente diferente al que se ha estado desarrollando en los últimos años, durante los cuales las aplicaciones web se han desplegado generalmente siguiendo arquitecturas monolíticas.

Utilizando arquitecturas monolíticas el desarrollador descompone la aplicación web en tres capas: la interfaz de usuario, la lógica de la aplicación y el sistema de gestión de datos. Estas capas son ejecutadas habitualmente sobre una misma máquina (normalmente externalizando las bases de datos). Esto provoca que la flexibilidad sea mínima, que un pequeño fallo en cualquier punto tenga un gran impacto, que todas las funcionalidades estén escritas sobre un mismo lenguaje y que el escalado se haga horizontalmente de una manera ineficiente.

Las arquitecturas basadas en microservicios proponen sin embargo una arquitectura en la que cada funcionalidad quede dividida en un nuevo servicio web lo más independiente posible, lo cual va a mejorar en gran medida los puntos débiles de las aplicaciones monolíticas.

Para poner en práctica y entender en mayor medida los conocimientos sobre este tipo de arquitecturas, se ha desarrollado un ejemplo de aplicación web basada en microservicios. Por último, se ha analizado su funcionamiento, llegando a una serie de conclusiones.

Abstract

In this Final Bachelor Degree Project, a study has been carried out on a new emerging trend in web application development. This trend is based on a new architecture model known as microservices.

The use of architectures based on microservices, represents a new point of view in the development of web applications substantially different from the one that has been in use in the last years, during which web applications have been deployed generally following monolithic architectures.

When using monolithic architectures the developer decomposes the web application in three layers: the user interface, the application logic and the data management system. These three layers are usually running in the same machine (mostly outsourcing the databases), causing a number of design failures that are difficult to avoid: minimal flexibility, chain-reaction from a single error through the application, impossibility to write functionalities in other programming language, and an inefficient horizontal scalability.

Microservices-based architectures however propose an structure in which each functionality is divided into a new web service as independent as possible. Therefore greatly improving the weak points of monolithic applications.

In order to have a practical and better understanding of this type of architecture, an example application based in microservices has been developed. Finally, its performance has been analyzed and conclusions have been drawn.

Palabras Clave

Microservicios, aplicaciones web, Docker, Node JS, SOA, escalabilidad.

Índice

1.- Introducción	6
1.1 - Motivación personal	7
1.2 - Objetivos del trabajo fin de grado	7
2.- ¿Qué son las “Arquitecturas basadas en microservicios”?	8
2.1 - Origen de las arquitecturas basadas en Microservicios: SOA.....	9
2.2 - Estado actual del desarrollo de microservicios a nivel empresarial.....	10
3.- Características de las Arquitecturas basadas en Microservicios a través del Caso de Estudio realizado.....	11
3.1 - Visión global de la aplicación web desarrollada.....	11
3.1.1 - Estilo orquestado y estilo coreográfico	11
3.2 - Funcionalidad de los Microservicios desplegados	13
3.2.1 - Microservicio de gestión de usuarios y peticiones.....	14
3.2.2 - Microservicio de búsqueda.....	15
3.2.3 - Microservicio de ordenación del catálogo.....	16
3.2.4 - Microservicio carrito de la compra	17
3.3 - Interfaz de usuario	18
3.4 - Justificación de las tecnologías utilizadas.....	22
3.4.1 - Node JS	22
3.4.2 - Docker.....	23
3.4.3 - Bases de datos utilizadas: MondoDB y Redis	24
3.4.4 - Comunicación entre microservicios	25
3.5 - Características generales de los microservicios	26
3.5.1 - Independencia.....	26
3.5.2 - Automatización en el despliegue	26
3.5.3 - Descentralización en el manejo de las bases de datos	27
3.6 - Ventajas que aportan las arquitecturas basadas en microservicios	28
3.6.1 - Disminución del impacto provocado por fallos	28
3.6.2 - Heterogeneidad de sistemas	29
3.6.3 - Escalabilidad eficiente	30

3.6.4 - Flexibilidad y facilidad en la actualización	33
3.6.5 - Capacidad organizativa del equipo de trabajo	34
3.6.6 - Eficiencia	35
3.7 - Puntos débiles de las arquitecturas basadas en microservicios	36
3.7.1 - Complejidad de la aplicación.....	36
3.7.2 - Difícil migración desde aplicaciones monolíticas.....	36
3.7.3 - Comunicación entre microservicios	37
3.7.4 - Seguridad en las comunicaciones.....	38
4.- Conclusiones	39
5.- Líneas de investigación futuras.	40
5.1 - Comparación del rendimiento de Tecnologías	40
5.2 - Comparación de eficiencia.....	40
5.3 - Comparación de estilos	40
5.4 - Automatización del escalado.....	41
7.- Bibliografía y Referencias	42
ANEXO 1: Instalación y configuración de Docker	43
Instalación de Docker sobre Linux Ubuntu	44
Instalación de Docker sobre Mac OS	45
ANEXO 2: Despliegue local del ejemplo de app web	46
Descarga de la aplicación	47
Despliegue de la aplicación en local.....	47
Añadir productos a la base de datos de MongoDB.....	50

1.- Introducción

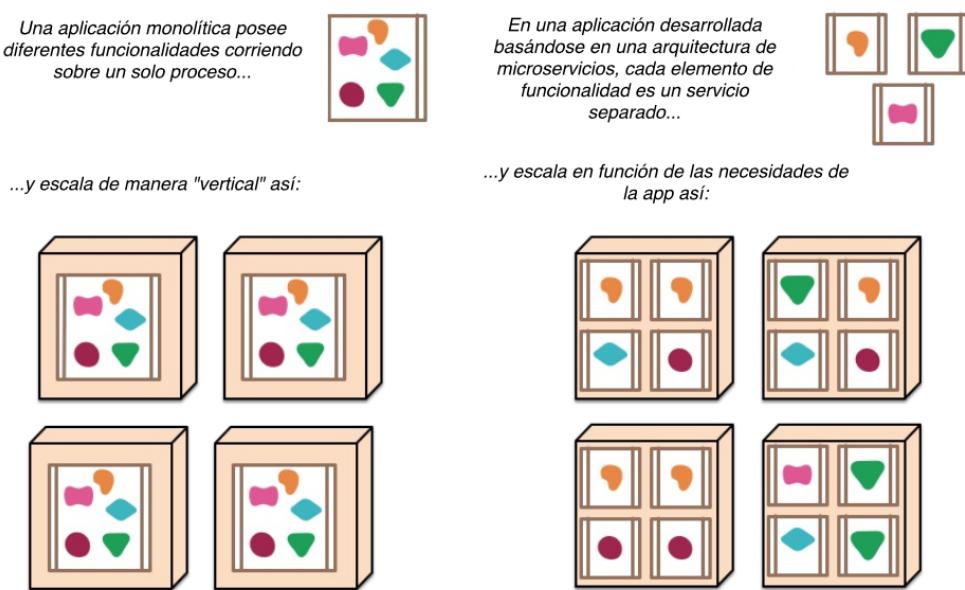
Hoy en día la mayoría de aplicaciones web están desarrolladas siguiendo una arquitectura conocida como “monolítica”. Normalmente este tipo de aplicaciones pueden dividirse en tres grandes partes, la interfaz de usuario del lado cliente, el lado servidor y el conjunto de bases datos. El lado servidor de las aplicaciones que siguen la estructura clásica o monolítica está compuesto por múltiples funciones y procesos escritos unos tras otros, en muchas ocasiones dependientes entre sí con el fin de dar a la aplicación la funcionalidad requerida por el cliente.

Este tipo de arquitectura acarrea muchos problemas para el equipo desarrollador cuando es necesario realizar un cambio en el lado servidor, ya sea por motivos de escalabilidad (puede ser muy costoso replicar esta parte entera) o por la necesidad de introducir algún cambio en la funcionalidad. Esto se debe a que un pequeño cambio en el código de un proceso puede provocar una reacción en cadena de errores en otros procesos que también deben ser corregidos para que la operatividad se mantenga tal y como estaba en el estado inicial, teniendo por tanto que realizar probablemente una reconstrucción y despliegue de la aplicación entera.

En muchas ocasiones, la mejoría de ciertas aplicaciones web se ve ralentizada por alguno de estos motivos, ya que el tener que reconstruir gran parte de la aplicación por pequeño que sea el cambio deseado, puede suponer un gran gasto tanto económico como temporal entre los miembros del equipo de desarrollo. Esto provoca que se opte por la filosofía de “cambiar lo menos posible” o “si funciona es mejor no tocarlo” por miedo a que la innovación provoque más desventajas que ventajas en la aplicación ya desarrollada.

Como consecuencia a este tipo de problemas, surge la necesidad de crear una nueva arquitectura que permita la escalabilidad de algunos recursos requeridos por la aplicación de manera independiente, y que además proporcione a la misma mucha más flexibilidad que la estructura monolítica. Esta arquitectura es la basada en microservicios, que como veremos posteriormente mejora en gran medida las características mencionadas, además de conceder grandes ventajas en otros ámbitos a tener en cuenta a la hora del desarrollo de aplicaciones web.

En la siguiente imagen se puede apreciar una comparación entre el desarrollo basándose en una arquitectura monolítica y en una arquitectura basada en microservicios:



- Fuente de la imagen: <http://www.martinfowler.com/articles/microservices.html> -

1.1 - Motivación personal

Personalmente me he decidido a realizar este Trabajo Fin de Grado porque me resulta un tema muy interesante y novedoso además de tener a priori una gran proyección en un futuro muy próximo.

La realización del mismo me va a permitir consolidar los conocimientos que he adquirido durante el transcurso del Grado sobre tecnologías web. Además, me va a permitir apreciar en primera persona las dificultades y el esfuerzo que conlleva el desarrollo de una aplicación web basada en microservicios, así como el valorar la apreciación de los múltiples detalles que se deben de tener en cuenta para que todo funcione de la manera deseada en un aplicación de este estilo.

La realización de un ejemplo práctico durante el desarrollo del Trabajo Fin de Grado me va a aportar una experiencia bastante significativa en el uso y desarrollo de algunas de las tecnologías más utilizadas hoy en día en el desarrollo de aplicaciones web.

1.2 - Objetivos del trabajo fin de grado

Este trabajo fin de grado tiene objetivo principal realizar un estudio exhaustivo de las características que poseen las arquitecturas basadas en microservicios utilizadas en el desarrollo de aplicaciones web, así como las ventajas que algunas de sus propiedades pueden ofrecer a los desarrolladores que utilicen este tipo de estructura.

Para ello se va a estudiar toda la documentación disponible a nuestro alcance, lo que se prevé complejo ya que existe poca información al respecto debido a lo novedoso que el tema abarcado resulta.

Como parte del trabajo de estudio, también se van a valorar distintas tecnologías que se pueden utilizar para implantar una arquitectura basada en microservicios, con el objetivo de aclarar al lector la elección a la hora de desarrollar una aplicación de este tipo, a través de mostrar las diferencias entre ellas y las ventajas que pudieran aportar.

Este TFG no pretende limitarse únicamente a un estudio teórico que sintetice las diferentes ventajas y desventajas que pueden ofrecer las arquitecturas basadas en microservicios. Como hilo conductor, durante gran parte del documento, se pretende realizar un ejemplo de aplicación lo más real posible que este basado en este tipo de arquitectura. Mediante este ejemplo se desea que el trabajo fin de grado pueda servir como “tutorial” teórico-práctico de aprendizaje para todos aquellos desarrolladores interesados en este modelo de desarrollo.

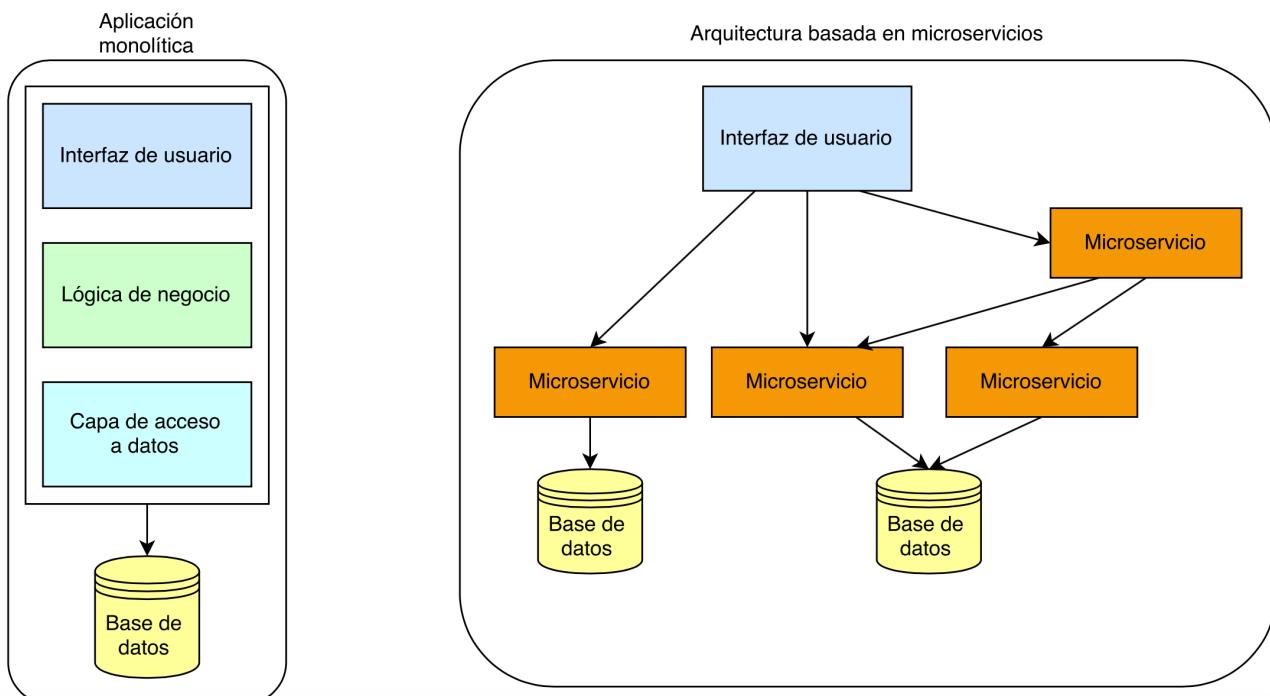
A través esta aplicación se pretende ejemplificar las diferentes partes teóricas que se van a tratar en el TFG con el fin de esclarecer los conceptos que se van a definir. El código de esta aplicación estará disponible en la plataforma Github de manera pública para que cualquier lector pueda acceder a el de manera sencilla.

Se espera que este TFG resulte de gran utilidad a desarrolladores que pretendan instaurar o probar este nueva metodología de realización de aplicaciones, así como que sirva de motivación entre los que tienen más dudas sobre la misma.

2.- ¿Qué son las “Arquitecturas basadas en microservicios”?

Es necesario antes de entrar a enumerar las características de este modelo de arquitectura, conocer una breve definición de “servicios web” (de ahora en adelante servicios). Conocemos como servicios un conjunto de aplicaciones o tecnologías con capacidad de intercambiar datos entre ellos independientemente de las propiedades de cada aplicación o de las plataformas sobre las que estas se encuentran instaladas, normalmente utilizando el protocolo HTTP.

El modelo de arquitectura basado en microservicios tiene como idea principal dividir el software de la aplicación en servicios que sean lo más independientes posible entre ellos (cabe la posibilidad en función de la aplicación desarrollada que algunos servicios deban estar relacionados entre sí de manera irrefutable) y que sean fácilmente reemplazables y actualizables. No obstante, sigue siendo necesario que las aplicaciones que sigan esta estructura posean un lado cliente tal y como lo conocemos ahora en el que de algún modo confluyan los resultados de los procesos llevados a cabo en el lado servidor y muestren al usuario final la funcionalidad de la aplicación de manera correcta.



- Fuente de la imagen: <https://developer.ibm.com/> -

A pesar de que todos los expertos en el desarrollo de tecnologías web están generalmente de acuerdo en cuales son las características principales de las arquitecturas basadas en microservicios, existe bastante controversia a la hora de definir de manera unívoca tanto la palabra “Microservicios” como “arquitecturas basadas en microservicios”.

La definición más general es la dada por Martín Fowler y James Lewis, dos “pioneros” en este tipo de arquitectura en su artículo “Microservices” [1]. En este artículo se definen microservicios como un “estilo arquitectural en el que múltiples servicios, cada uno corriendo de manera individual y desplegados de la forma más automatizada posible, se comunican entre sí mediante mecanismos ligeros, generalmente un recurso API basado en HTTP”.

Otra de las preguntas que surgen al profundizar en el tema estudiado es: ¿como de pequeño debe ser un servicio web para ser considerado microservicio?. Al igual que ocurría con la definición de microservicios, no existe una respuesta exacta o estricta a esta pregunta, ya que en el desarrollo de software, no existe una unidad de medida como tal más allá del peso en Mb que puede ocupar una aplicación o del número de líneas de código que la constituyen, lo cual no es relevante más allá del dimensionamiento de espacio. El término microservicio surge cuando se da un paso adelante a la hora de descomponer en los fragmentos más pequeños posibles una aplicación monolítica. El autor Sam Newman en su libro “Building Microservices” [2] se aferra a la definición dada por Jon Eaves que opina que “se puede considerar microservicio todo servicio web funcional que pueda ser reescrito por una persona en menos de dos semanas”, definición que si puede tener más sentido si lo que estamos buscando es algo que se pueda medir.

Lo que es rigurosamente claro a la hora de considerar un microservicio como tal es que este debe cumplir ciertas características: debe desplegarse de manera individual, tiene que ser lo más independiente posible con respecto al resto de microservicios que compongan la aplicación, debe ser capaz de comunicarse con el resto y debe ser escalable de manera individual.

En cualquier caso, aunque el término pueda sonar novedoso, la idea de microservicios tiene su origen en el inicio del uso de la palabra SOA (Services Oriented Architecture). De hecho, una primera lectura sobre el tema puede dar a entender que son lo mismo o prácticamente iguales. La realidad es que de manera general entre los desarrolladores se entienden las arquitecturas basadas en microservicios como una evolución natural de SOA. En el siguiente apartado, se van a comentar los fundamentos que comparten SOA y microservicios así como alguna de sus diferencias.

2.1 - Origen de las arquitecturas basadas en Microservicios: SOA

Si en el apartado anterior comentábamos que existe cierta controversia a la hora de definir microservicios, esta controversia es aún mayor entre expertos desarrolladores si el tema que consideramos es la similitud entre Microservicios y SOA. Algunos opinan incluso que estos dos modelos son iguales, lo que ha instaurado cierto temor a la hora de avanzar en el desarrollo de aplicaciones basadas en microservicios, ya que la imagen de SOA quedó empañada hace un tiempo cuando debido a su desconocimiento y falta de tecnologías para instaurarse, se produjeron múltiples implementaciones fallidas.

En el manifiesto SOA [4] se declaran las características generales que siguen las arquitecturas de este tipo. En general las definiciones que en este documento se realizan, son en alguno de los casos muy abiertas por lo que nuestro concepto de arquitecturas basadas en microservicios encajan en muchos de sus puntos.

Si se puede apreciar, que existen dos diferencias importantes entre ambas definiciones. La primera, según el manifiesto en SOA los servicios se exponen a través de contratos, sin embargo en microservicios la forma natural de exponer los servicios es mediante RESTful API. La segunda es que en microservicios, los servicios web se despliegan de manera automatizada y totalmente independiente, no ocurre así en SOA donde a priori los servicios se desplegarán manualmente.

Mas allá de entrar en opiniones personales, lo que es evidente es que los fundamentos que han dado pie a la aparición de las arquitecturas basadas en microservicios son los mismos en los que se basa SOA.

2.2 - Estado actual del desarrollo de microservicios a nivel empresarial

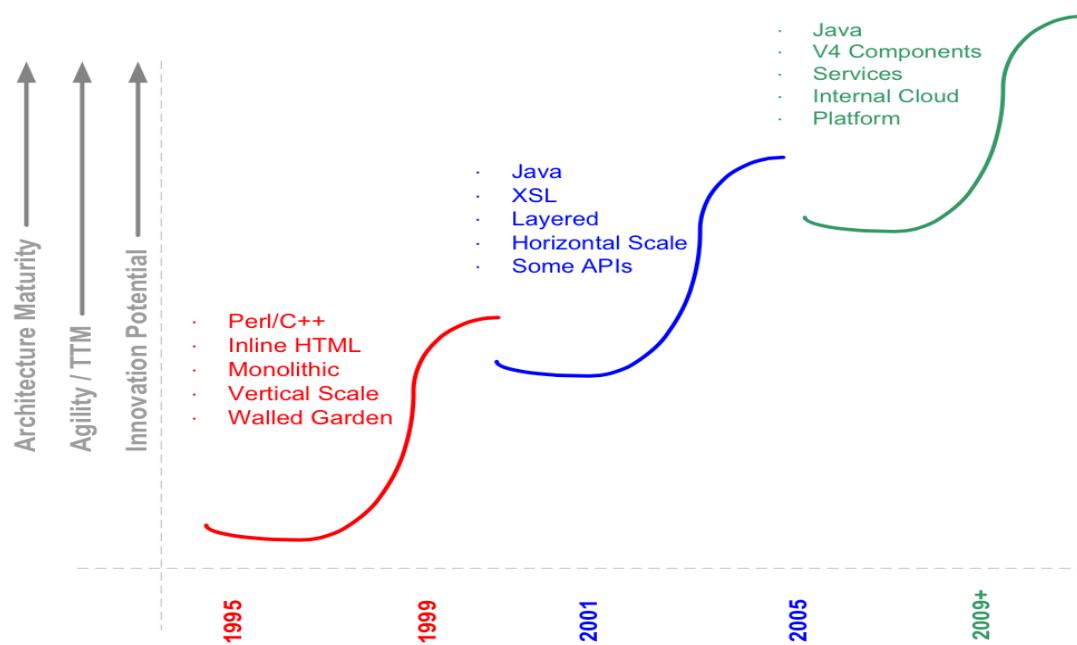
Aunque en apartados anteriores se ha comentado que el término microservicios es bastante novedoso, son múltiples las empresas que están empezando a migrar sus aplicaciones para el desarrollo sobre este tipo de arquitectura.

Las circunstancias que han motivado el cambio en estas empresas son muy variadas, ya que como veremos en apartados posteriores, el uso de microservicios puede aportar grandes ventajas en diferentes apartados importantes de las aplicaciones web.

Como vamos a ver posteriormente el uso de este tipo de arquitecturas en aplicaciones web de pequeño alcance, puede carecer de sentido. Esto se debe a que las ventajas residen principalmente en campos como la escalabilidad, la eficiencia, la protección ante fallos etc.. Campos que se ven potenciados cuando el número de usuarios que hacen uso de la aplicación es muy grande o cuando se acrecienta de manera inesperada.

Un ejemplo real de cambio y evolución constante en su arquitectura es la empresa eBay. Desde su fundación cercana a 2002, han realizado cinco cambios muy importantes en su infraestructura, a destacar principalmente dos de ellos, a través de estos cambios se puede observar la evolución de los diferentes modelos utilizados para el desarrollo de aplicaciones web. eBay comenzó desarrollando aplicaciones monolíticas en Perl y C++, a continuación evolucionó al desarrollo de servicios a través de Java en el año 2007, y debido a su gran crecimiento en 2011 comenzó a utilizar microservicios como base en sus desarrollos.

En la siguiente imagen se aprecia la evolución realizada llevada a cabo por eBay hasta 2011:



- Fuente de la imagen: <http://www.slideshare.net/tcng3716/ebay-architecture> -

Otros ejemplos que están comenzando a migrar sus infraestructuras son Google, Twitter, Netflix o Amazon que siguen una evolución similar a eBay ya que también se han visto obligados a escalar muy rápido en poco tiempo.

3.- Características de las Arquitecturas basadas en Microservicios a través del Caso de Estudio realizado

Como se ha comentado en el apartado destinado a describir los objetivos del trabajo, la exposición de las características de arquitecturas basadas en microservicios, así como las ventajas y desventajas de su implantación, no sólo se van a definir de manera teórica sino que se van a ejemplificar a través de la aplicación web desarrollada con el fin de esclarecer algunos de los conceptos abordados en este apartado.

Por este motivo, antes de comentar dichas características se va a presentar el ejemplo de aplicación web desarrollada, ya que en muchos puntos del Trabajo Fin de Grado nos vamos a apoyar sobre ella para ejemplificar algunas de estas definiciones.

3.1 - Visión global de la aplicación web desarrollada

Esta aplicación pretende simular el servicio de compra online que ofrecen numerosas tiendas hoy en día, simplificando alguno de los servicios que en este TFG no son objetos de estudio.

Las arquitecturas basadas en microservicios se pueden reagrupar según la disposición de elementos de la aplicación y la estructura que sigue la comunicación entre los microservicios en dos grandes estilos, “estilo orquestado” y “estilo coreográfico”.

3.1.1 - Estilo orquestado y estilo coreográfico

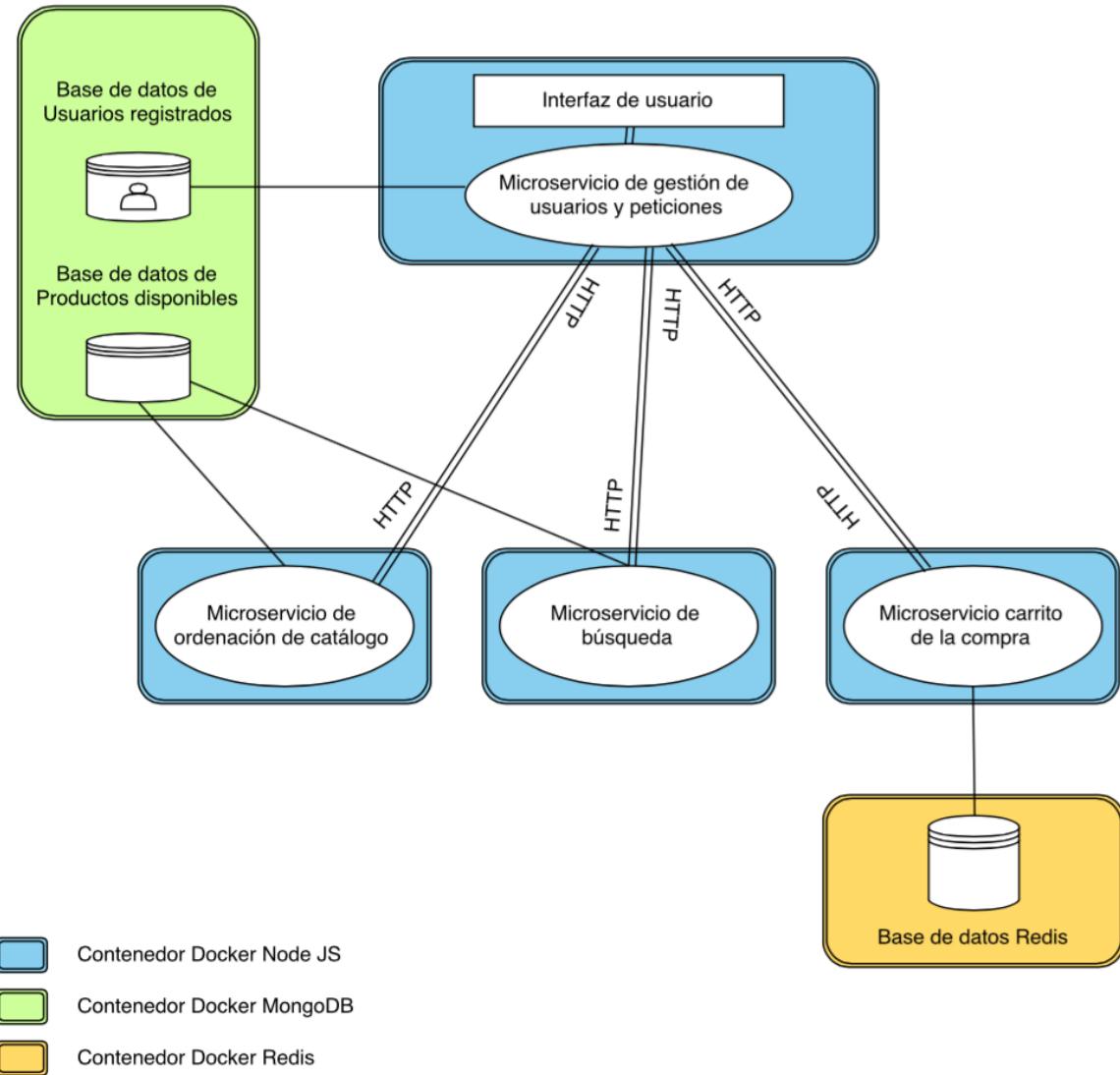
El estilo orquestado se caracteriza por tener un elemento que actúa como director de orquesta con respecto al resto de servicios. Cuando se produce un evento, este elemento es el encargado de dirigir y manejar las peticiones hacia el/los microservicios que deben tomar parte en la funcionalidad requerida, y posteriormente se encarga de ofrecer resultados a la parte cliente de la aplicación.

Por su parte, en el estilo coreográfico, a través de un primer elemento encargado de recibir las peticiones, se informa a todos los microservicios que dependen de este elemento de la llegada de la petición. Cada uno de los diferentes servicios se encarga de gestionar la petición (o descartarla si fuera necesario) tal y como ha sido programado.

Si comparamos los dos tipos de arquitectura es evidente que el despliegue de un estilo coreográfico es ligeramente más complejo que en el caso del despliegue de un estilo orquestado. Esto es debido a que mientras que al utilizar un estilo orquestado la mayor complejidad reside en un desarrollo óptimo del elemento principal de la aplicación, si queremos llevar a cabo un estilo coreográfico hay que desarrollar en detalle un sistema de notificación de eventos entre los servicios para que todo funcione de manera fluida y ágil.

En nuestro caso se ha decidido desarrollar la aplicación bajo un estilo orquestado ya que las características de este estilo permiten realizar una descripción más clara de las características que vamos a definir.

A continuación se muestra un esquema de la arquitectura lógica que posee la aplicación, donde se pueden observar los diferentes microservicios desarrollados así como la implementación de diferentes bases de datos que van ligadas a ellos. La función de estos microservicios así como las tecnologías utilizadas para su desarrollo se explicarán posteriormente.



Como se puede ver en el diagrama, y como se ha comentado en la página anterior, el estilo que se ha seguido (más allá de consistir en una arquitectura basada en microservicios) es el estilo orquestado.

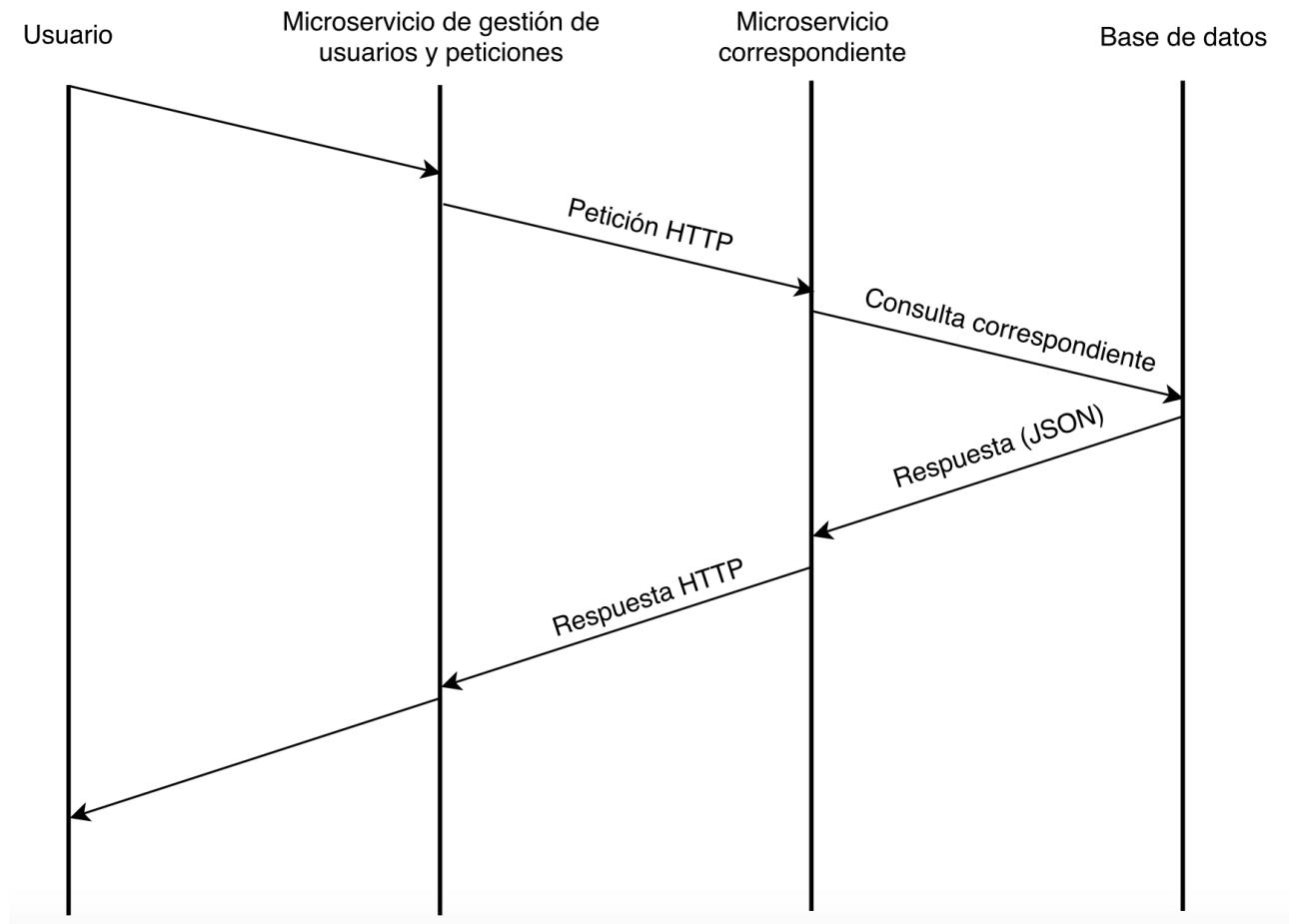
El elemento principal, o director de orquesta en este caso se encuentra en el contenedor Docker aparece en la parte superior de la imagen. En la aplicación aquí desarrollada, se dirigirán las rutas hacia el resto de los microservicios y se mostrarán los resultados a través de la interfaz de usuario, desplegada en este mismo contenedor.

El código que compone todos los elementos de esta aplicación está ubicado de forma pública en: <https://github.com/ManuPH/Microservices>.

3.2 - Funcionalidad de los Microservicios desplegados

Antes de definir las funcionalidades de cada uno de los microservicios desarrollados en el ejemplo de aplicación web realizada, se debe aclarar que a pesar de que en el esquema aparece un elemento denominado “Servicio de pago externo”, ese servicio no se ha implementado como tal sino que se ha simulado.

Todos los microservicios funcionan de manera externa de forma similar, en el siguiente esquema se reflejan los pasos que se da en el interior de la aplicación cuando un usuario realiza una petición a través de la interfaz:



En este esquema se muestra el camino que recorre la información cuando se requiere de la funcionalidad de alguno de los microservicios. Como es obvio, en función del recurso requerido por el usuario de la aplicación, la información que viajará entre microservicios será diferente (el tipo de petición HTTP, el contenido del archivo JSON...).

En los siguientes apartados se va a profundizar en como varía la información en función de los microservicios que participan en la petición, y se va a entrar en detalle en la descripción de la funcionalidad de dichos microservicios.

3.2.1 - Microservicio de gestión de usuarios y peticiones

Este microservicio se puede dividir en dos partes, la gestión de usuarios y la gestión de peticiones.

La parte de gestión de usuarios es la encargada de realizar las funciones de login y registro de usuarios. Para ello se ha utilizado el middleware “Passport” desarrollado para Node Js.

Esta primera parte se encarga principalmente de tres funciones. Por un lado, cuando un usuario decide registrarse en la página web, este microservicio guarda los datos introducidos por este en una base de datos escrita en MongoDB (en apartados posteriores se realizará una justificación de las tecnologías utilizadas), de tal manera que queden accesibles para la realización de consultas sobre los mismos desde el resto de funciones. La segunda función es la conocida como “login”. En el diseño de la aplicación se ha decidido que todo aquel que quiera “navegar” por la aplicación debe haber hecho login antes, por lo tanto, en el momento en el que un usuario decide acceder a la aplicación a través de sus credenciales, se comprueba que el usuario y la contraseña coinciden con los almacenados en la base de datos de usuarios y en caso afirmativo se habilita a dicho usuario a realizar las acciones que deseé.

Además, cada vez que se utiliza alguno de los otros microservicios, se comprueba a través del sistema de gestión de usuarios, que el usuario que está utilizando la aplicación está logueado. Este es el motivo principal por el que este microservicio está desplegado en el mismo contenedor Docker que la interfaz de usuario.

La segunda parte es la que se ha denominado como “gestión de peticiones”. Es el elemento que siguiendo la nomenclatura correspondiente al estilo orquestado, denominaríamos “director de orquesta”. Cuando un usuario quiere acceder a un recurso, esta parte del microservicio es la encargada de redirigir las peticiones necesarias hacia el microservicio correspondiente, y una vez obtenidos los recursos requeridos, ofrecerlos a la interfaz de usuario para que se muestren correctamente.

Este microservicio es vital, y todos los usuarios van a pasar por él de manera continua, por este motivo, y como veremos más adelante, es uno de los microservicios “candidato” para ser replicado en el caso de que fuera necesario proporcionar servicios a más usuarios.

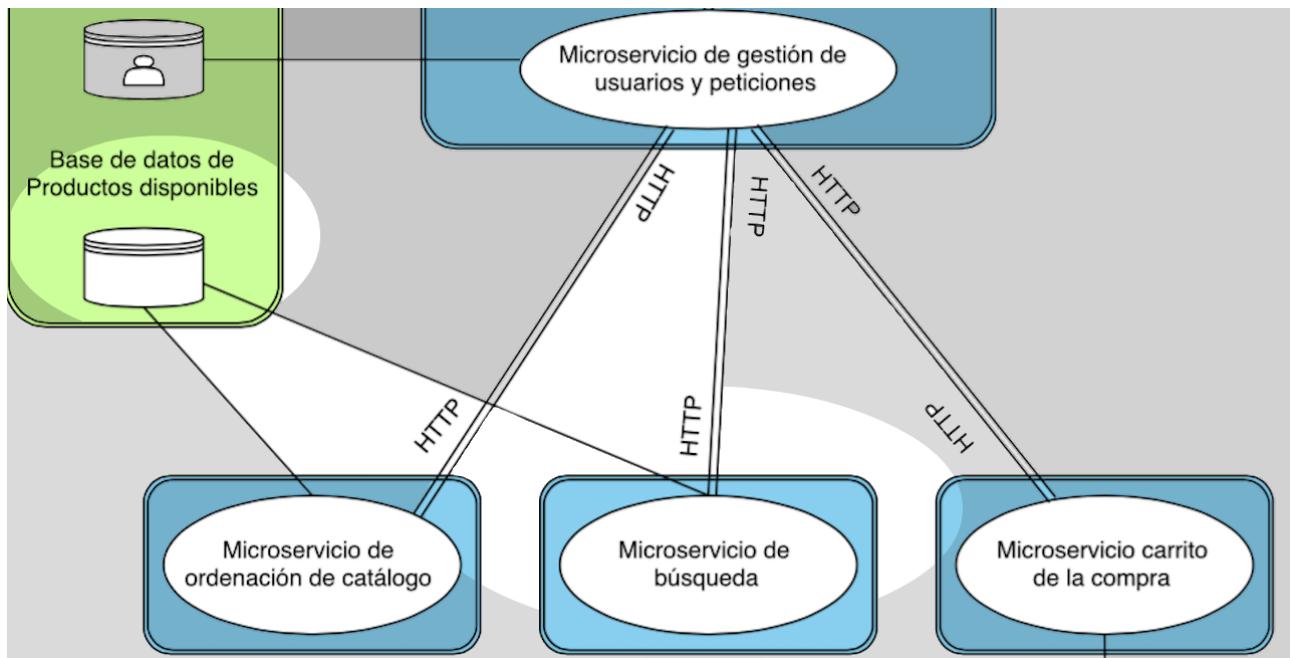
Como veremos posteriormente en el apartado destinado a comentar los puntos débiles de las arquitecturas basadas en microservicios, si la aplicación desarrollada fuera una aplicación comercializada, habría que prestar mucha atención a la seguridad instaurada en esta parte de la aplicación. Desde el microservicio de gestión de usuarios, se van a manejar datos personales cuya confidencialidad es muy importante, y además en el caso de que se realizara de forma real el pago de los productos seleccionados, los datos necesarios para realizar dicha acción también se transmitirían desde el microservicio de gestión de usuarios y peticiones.

El código del microservicio de gestión de usuarios y peticiones se encuentra junto con la interfaz en el directorio denominado “src” que se encuentra alojado en Github.

Para descargar tanto este microservicio como el resto de la aplicación, se puede consultar el Anexo 2, donde se explica el procedimiento.

3.2.2 - Microservicio de búsqueda

Este es el microservicio más sencillo de los 4 microservicios que se han implementado en esta aplicación. En la siguiente imagen se destaca la parte de la arquitectura global en la que interviene el microservicio de búsqueda:



El usuario utilizará este servicio cuando en el buscador de la interfaz de usuario (podrá verse gráficamente en el apartado posterior destinado a esta parte de la aplicación) escriba el modelo del producto que desea buscar. A partir de ese punto, se realiza una petición HTTP tipo POST a este microservicio en la que se envía el modelo buscado por el usuario. Este microservicio se encarga de consultar en la base de datos que existe el modelo buscado por el usuario y en caso afirmativo, devuelve de nuevo a través de una petición HTTP el resultado buscado. Este resultado consiste en un archivo tipo JSON que incluirá los datos correspondientes al producto almacenado en la base de datos.

Cabe destacar que este microservicio concurre con otro distinto, el cual se detallará a continuación, denominado “microservicio de ordenación de catálogo” en el acceso a la base de datos de producto, ya que de este modo solo se debe realizar una única actualización de la base de datos para que los dos servicios queden “notificados”.

A pesar de ser un servicio simple, se puede prever que en una aplicación real sería posiblemente el más utilizado por los usuarios. Por este motivo, del mismo modo que ocurre con el microservicio de gestión de usuarios y peticiones que se ha visto en el apartado anterior sería buena idea “ligerizarlo” para que su escalado sea eficiente (las ventajas que puede aportar el uso de microservicios se verán más adelante).

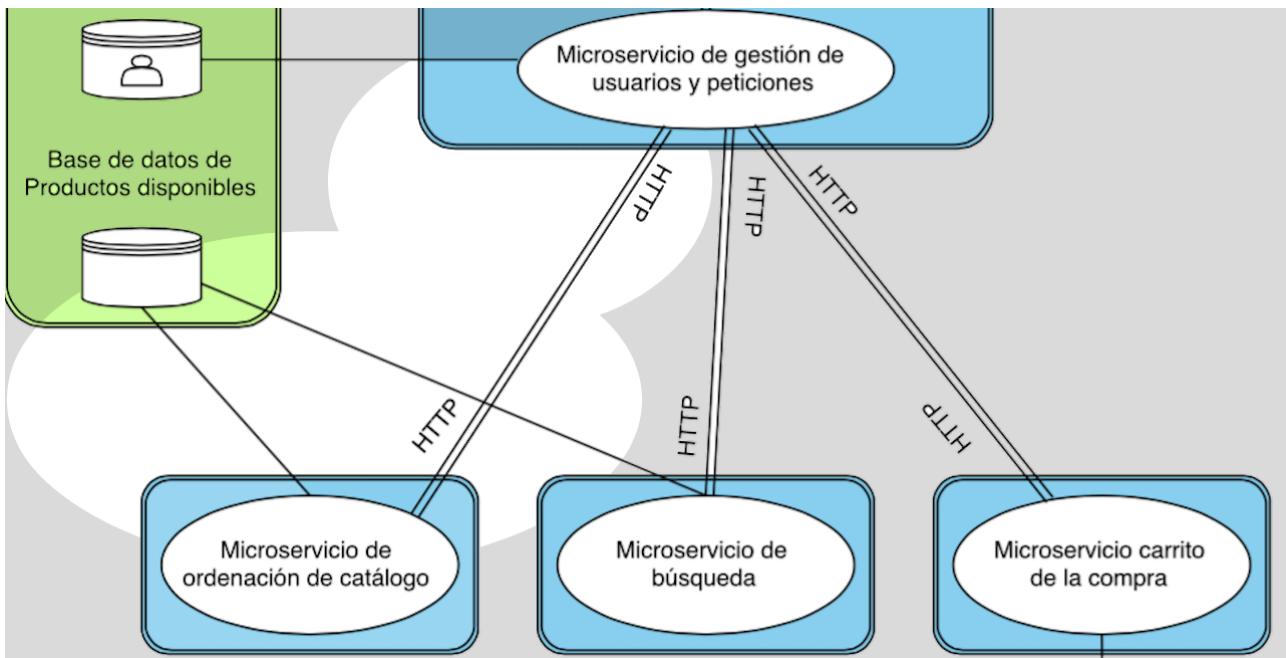
Este microservicio se encuentra en el directorio “búsqueda” alojado en el link de Github

3.2.3 - Microservicio de ordenación del catálogo

Del mismo modo que funcionaba el microservicio de búsqueda, este servicio es el encargado de ofrecer al usuario una lista con los modelos de los que dispone la “tienda”.

Es una funcionalidad que también es muy común entre la mayoría de tiendas que tienen aplicaciones web para la realización del servicio de venta online. Para adecuarse a las necesidades de los clientes se ofrece la posibilidad de ordenar el catálogo de productos disponibles en función de unas determinadas características. Se ha implementado la posibilidad de ordenar el catálogo en función de su precio (tanto de manera ascendente como descendente) y en función de la talla disponible de los modelos.

En la siguiente imagen se destaca la parte de la arquitectura global en la que interviene el microservicio de ordenación de catálogo:



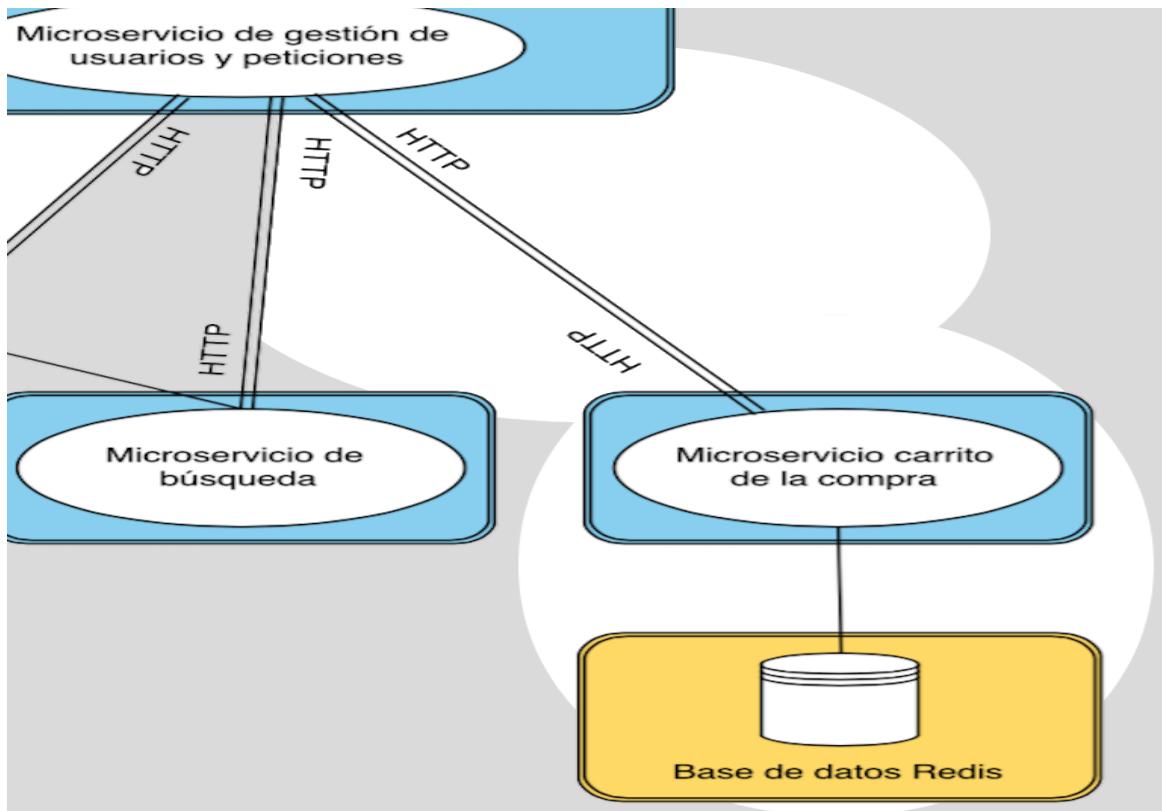
En este caso, al realizar la petición se pasa como parámetro al microservicio la forma en la que el usuario a decidido ordenar el catálogo. Esta petición se realiza de nuevo mediante una petición HTTP pero en este caso de tipo GET, ya que se ha entendido que los parámetros transmitidos no son de gran importancia (por lo que la seguridad no es esencial) y además puede ser orientativo para el cliente que pueda observar en la barra de navegación la manera que ha seleccionado de ordenar el catálogo.

La petición se traslada a la base de datos, que devuelve todos los modelos que en ella se encuentra pero ya ordenados de la manera elegida por lo que la interfaz de usuario no tiene que “conocer” que se quiere ordenar el catálogo, sino simplemente representar los objetos que llegan a la misma.

Este microservicio se encuentra en el directorio “catálogo” alojado en el link de Github.

3.2.4 - Microservicio carrito de la compra

Si el microservicio destinado a la búsqueda de un producto podíamos considerarlo el más sencillo, en este caso por contra, podemos decir que se trata del más complejo. En la siguiente imagen se destaca la parte de la arquitectura global en la que interviene el microservicio carrito de la compra:



De nuevo se trata de una funcionalidad que se ve en la mayoría de tiendas online. Dicha funcionalidad consiste en almacenar los objetos seleccionados por el cliente en una base de datos temporal, de tal manera que luego puedan visualizarse y comprar de manera conjunta. Esto evita que, en el caso de que se quieran comprar múltiples productos, allá que realizar el pago cada vez que se selecciona uno de ellos.

La dificultad reside en que los productos seleccionados por una persona tiene que ir asociados a sus credenciales de usuario, y además esos productos deben de expirar pasado cierto tiempo, ya que en caso contrario, en muchas ocasiones se estaría ocupando un espacio importante de forma totalmente innecesaria. Estas características nos han hecho escoger Redis para su implementación, cuya elección justificaremos posteriormente en más detalle en el apartado destinado a tal propósito.

Como hemos comentado en el párrafo anterior, hay que asociar cada carrito a cada usuario, por tanto además de enviar a través de una petición HTTP el producto seleccionado, también se transmite el usuario que está realizando la petición en ese momento. El nombre de usuario se utiliza como “clave” en la base de datos Redis y de este modo los productos quedan asociados a su sesión.

Este microservicio se encuentra en el directorio “carrito” alojado en el link de Github.

3.3 - Interfaz de usuario

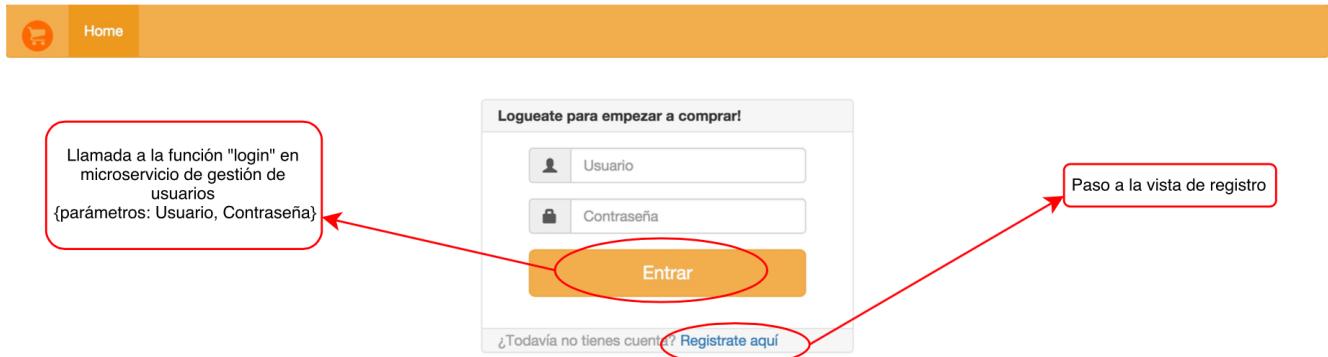
En el ejemplo de aplicación web desarrollada, la interfaz de usuario consiste en una presentación sencilla, de tal manera que se puedan observar los resultados obtenidos del desarrollo realizado en el lado servidor, pero sin entrar en demasiados ornamentos, ya que no es el objeto de estudio principal de este TFG.

Para el desarrollo de la interfaz se han utilizado los lenguajes HTML5, CSS y JavaScript, ya que combinan muy bien con Node JS, lo que nos permite realizar todas las funciones deseadas.

Para dar una apariencia más vistosa a la web se ha utilizado “Twitter Bootstrap”. Se trata de un Framework desarrollado por Twitter que ofrece numerosos elementos predefinidos pero personalizables que pueden incluirse en cualquier pagina web. Además, permite que el diseño se adapte en función del tamaño del dispositivo utilizado para la visualización, una gran ventaja y ahorro de tiempo para los desarrolladores.

Se puede considerar que la interfaz desarrollada está dividida en cuatro vistas diferentes, con una barra de navegación que hace de nexo entre todas ellas, las cuatro partes son: vista inicial o de “login”, la vista destinada al registro, la vista de presentación del catálogo, y la vista de los productos añadidos al carrito de la compra, estas vistas junto con una explicación de la barra de navegación se detalla a continuación.

Vista inicial o “login”:



Como se aprecia en la imagen se trata una vista simple. En apartados anteriores, donde se ha comentado la funcionalidad de los microservicios, se ha expuesto que a la hora de navegar por la aplicación web se exige que el usuario esté logueado. Esta es la función básica de la interfaz inicial, permitir al usuario a hacer login, aunque también da la posibilidad de crear una nueva cuenta de usuario a aquel que aún no esta logueado a través del enlace “Regístrate aquí”.

Como se ha indicado en la imagen, la acción de hacer click en “entrar” hace una llamada al microservicio de gestión de usuarios que realizará la función comentada en el apartado anterior destinado a este microservicio.

La opción de hacer click en “Regístrate aquí” te lleva a la siguiente parte de la interfaz que se comenta a continuación.

Vista destinada al registro:

The screenshot shows a registration form with the following fields:

- Nombre de usuario (User name) - input field
- Nombre (Name) - input field
- Apellido (Last name) - input field
- Email - input field
- Password - input field
- Dirección (Address) - input field
- Número de cuenta (Account number) - input field

On the right side, there is a section titled "Terms and Conditions" with the following text:

- By clicking on "Register" you agree to The Company's Terms and Conditions
- While rare, prices are subject to change based on exchange rate fluctuations - should such a fluctuation happen, we may request an additional payment. You have the option to request a full refund or to pay the new price. (Paragraph 13.5.8)
- Should there be an error in the description or pricing of a product, we will provide you with a full refund (Paragraph 13.5.6)
- Acceptance of an order by us is dependent on our suppliers ability to provide the product. (Paragraph 13.5.6)

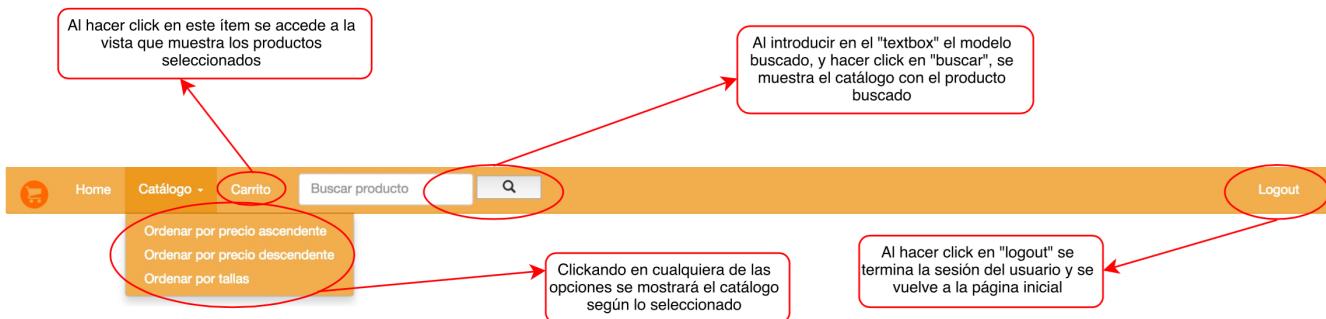
A red circle highlights the "Confirmar registro" (Confirm registration) button. A callout box points to it with the text: "Llamada a la función \"SignUp\" en el microservicio de gestión de usuarios {parámetros: todos los del formulario de registro}" (Call to the "SignUp" function in the user management microservice {parameters: all from the registration form}).

En la imagen se pueden ver los diferentes datos que se van a guardar de cada usuario. Además a través de una opción que ofrece HTML5, se requiere que el usuario que se vaya a registrar complete todos ellos.

Como se ha indicado en la captura, al hacer click en “Confirmar registro” se hace una llamada a la función SignUp del microservicio de gestión de usuarios, que se encargará de almacenar en la base de datos destinada a este fin los parámetros registrados del nuevo usuario.

Una vez completados los datos y formalizado el registro de manera satisfactoria, se redirige al usuario a la página de login para que pueda acceder a la aplicación con sus credenciales y empezar a utilizar la aplicación.

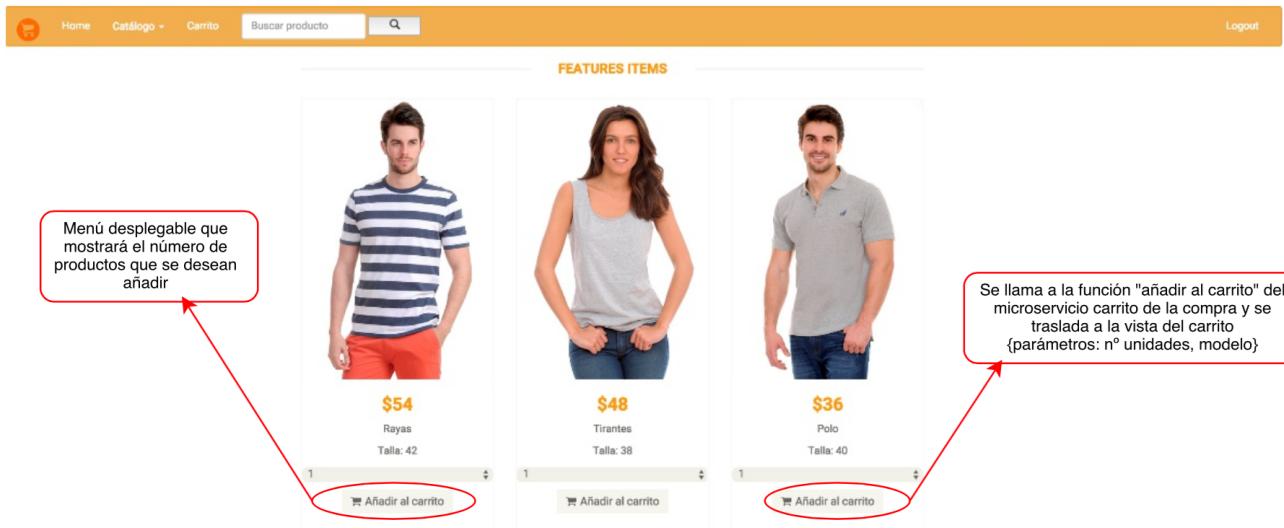
Barra de navegación una vez hecho login:



Esta es la barra de navegación que se muestra en todas las páginas que se visitan una vez hecho login.

En la imagen se expone la funcionalidad que tiene cada ítem que la compone, la idea es que a través de la barra de navegación los usuarios pueden acceder a todos los servicios de los que dispone la aplicación, excluyendo el añadir productos al carrito de la compra virtual, función que como veremos en la página posterior se hace desde la vista catálogo.

Vista del catálogo:



Se trata de la vista principal de la interfaz de usuario. En ella el usuario tiene la posibilidad de navegar a través de la interfaz para encontrar el producto “que va a comprar” y seleccionarlo.

En esta página se muestran los productos de la web, bien todos si se ha decidido ordenar el catálogo de una determinada manera, o bien un único ítem en el caso de que se haya buscado a través del buscador y se tenga éxito en la búsqueda.

El producto o productos que aparecen en la galería de prendas disponibles tienen cuatro características fijas y una seleccionable. Por un lado aparece el precio de cada producto, el modelo y la talla del mismo, estos son parámetros almacenados en la base de datos denominada “productos” a la que se puede acceder desde los microservicios de búsqueda y de ordenación del catálogo y que se devuelven en un formato JSON cuando son requeridos por el usuario a través del protocolo HTTP. La cuarta característica fija es la imagen, que para descargar de espacio la base de datos, y como se puede ver en la carpeta “public” dentro de “src” en el código alojado en GitHub se encuentra en el bloque que se ha denominado como “director de orquesta”.

La parte seleccionable consiste en el número de productos iguales que se desean comprar. Para que pueda haber correcciones sobre los productos seleccionados, la función “carrito de la compra” se ha programado para que en el caso de seleccionar un mismo producto en dos ocasiones el número de elementos que “se queda” en el carrito sea el último.

Como se puede ver en la vista “catalogo” de la carpeta “/src/views”, este formato de lista se ha programado a través de un bucle for sobre HTML que permite que la vista sea siempre la misma, pero que se acople la interfaz al número de elementos devueltos por el microservicio utilizado en cada caso concreto.

Una vez hecho click sobre añadir al carrito tras haber seleccionado el número de productos elegidos, dichos productos quedan añadidos al carrito de la compra y se redirecciona al usuario a la última de las vistas, donde se listan los productos seleccionados.

Vista de productos añadidos al carrito de la compra:

UNIDADES	PREnda	PRECIO
1	Polo talla 40	\$ 54
4	Rayas talla 42	\$ 152
2	Tirantes talla 38	\$ 98

Comprar

Se simula la llamada al servicio externo de pago

Esa vista, como se puede ver en la imagen, simplemente muestra el total de los productos seleccionados antes de confirmar la compra.

La lista que se muestra al usuario tiene tres columnas. La primera de ellas muestra el número de unidades añadidas al carrito de la compra por parte del usuario de cada producto. En la siguiente, aparece el modelo y talla de los productos seleccionados. Por último, se realiza un cálculo para obtener el precio total de coste de la suma de unidades seleccionadas de cada uno de los productos añadidos, y se muestra en la tercera columna.

También se puede apreciar al final de la lista un botón denominado “Comprar”. Como se ha comentado anteriormente, en la descripción global de la aplicación, se ha simulado el servicio correspondiente a la confirmación y abono del pago. Por lo tanto una vez hecho click en comprar, aparecerá la pantalla de confirmación que se puede ver en la siguiente imagen.

Al hacer click en "OK", se borran los productos añadidos al carrito del usuario. Se da por finalizada la compra

Su pago se ha recibido con éxito

Muchas gracias por su compra!

OK

Además de esta pantalla de confirmación, se borra el carrito asociado al usuario que ha iniciado sesión, ya que se da por finalizado el proceso de compra.

3.4 - Justificación de las tecnologías utilizadas

En esta sección se van a enumerar las tecnologías que se han utilizado para desarrollar la aplicación web de ejemplo descrita en el apartado anterior. A pesar de que se van a dar razones por las que se han elegido estas tecnologías, las arquitecturas basadas en microservicios pueden establecerse sobre múltiples tecnologías diferentes, muchas de las cuales aquí no se contemplan. En función de las características de la aplicación a desarrollar es posible que utilizando herramientas diferentes se optimicen los beneficios que ofrecen este tipo de arquitecturas.

Las dos principales tecnologías sobre las que vamos a desarrollar nuestra aplicación son Docker y Node JS. Se trata de tecnologías relativamente nuevas y que están teniendo gran aceptación entre los desarrolladores. Además, para el uso de microservicios, se compaginan muy bien y facilitan muchas de las funcionalidades requeridas.



3.4.1 - Node JS

Para el desarrollo de los diferentes servicios que intervienen en el ejemplo se ha utilizado la tecnología Node JS. Node permite la programación del lado servidor apoyándose en JavaScript y ofrece una configuración del apartado entrada/salida del servidor orientado a eventos, lo que facilita mucho el despliegue global de nuestra aplicación.

Además, Node JS permite la construcción, el despliegue y la escalabilidad de servicios web de manera sencilla, lo que como se ha visto en apartados anteriores, son características imprescindibles en el desarrollo de arquitecturas basadas en microservicios.

Otra ventaja que nos ofrece el uso de este lenguaje es la facilidad para la comunicación entre servicios a través de su módulo HTTP, en muy pocas líneas de código se pueden desarrollar tanto el envío como la recepción de una petición HTTP de cualquier tipo.

Node JS “conjunta” muy bien con la tecnología de virtualización ligera Docker sobre la que se profundizará en el siguiente apartado.

3.4.2 - Docker

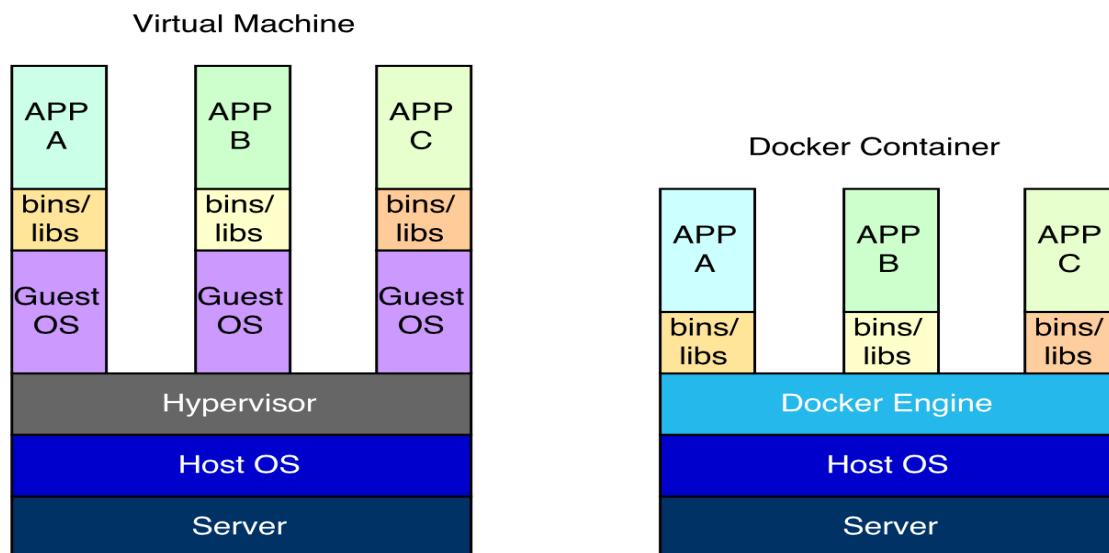
Además de Node JS, para el despliegue tanto de los servicios como de las bases de datos de diferentes tipos hemos utilizado la tecnología Docker.

Docker es una plataforma “Open Source” destinada a la construcción y el despliegue de aplicaciones en sistemas distribuidos de un modo en gran parte automatizado. Docker parte de un sistema operativo Linux para ofrecer al desarrollador tantas máquinas virtualizadas que pueden correr el mismo tiempo, como desee emplear este en su sistema.

A priori puede parecer que no hay gran diferencia entre el uso de Docker y el despliegue “normal” de máquinas virtuales Linux en un único sistema, pero las diferencias en cuanto a rendimiento, y eficiencia de recursos son mayúsculas.

Docker utiliza un sistema de virtualización ligera para la puesta en escena de sus máquinas virtuales (denominadas contenedores). Esto significa, que mientras una máquina virtual tiene un sistema operativo completo, una memoria propia y emula los recursos del sistema operativo original, múltiples contenedores Docker pueden correr a la vez apoyándose sobre un único “motor Docker” sin necesidad de virtualizar un sistema operativo nuevo, con el ahorro de recursos que esto supone.

La comparación entre el uso de recursos de máquinas virtuales y contenedores Docker se puede observar gráficamente en la siguiente imagen.



- Fuente de la imagen: <https://www.docker.com/whatisdocker> -

Tal y como se ha comentado en apartados anteriores, este TFG pretende ser en parte un tutorial teórico-práctico, por este motivo en el anexo 1 se va a incluir un manual de instalación de Docker sobre sistemas operativos que soporten Linux o SO diferentes, en estos últimos se deberá instalar la máquina virtual “boot2docker” para su correcto funcionamiento. Además, también se va a incluir en el anexo 2, una descripción detallada de como se debe desplegar la aplicación de ejemplo a través de Docker de manera local, para que poder observar el funcionamiento de la misma.

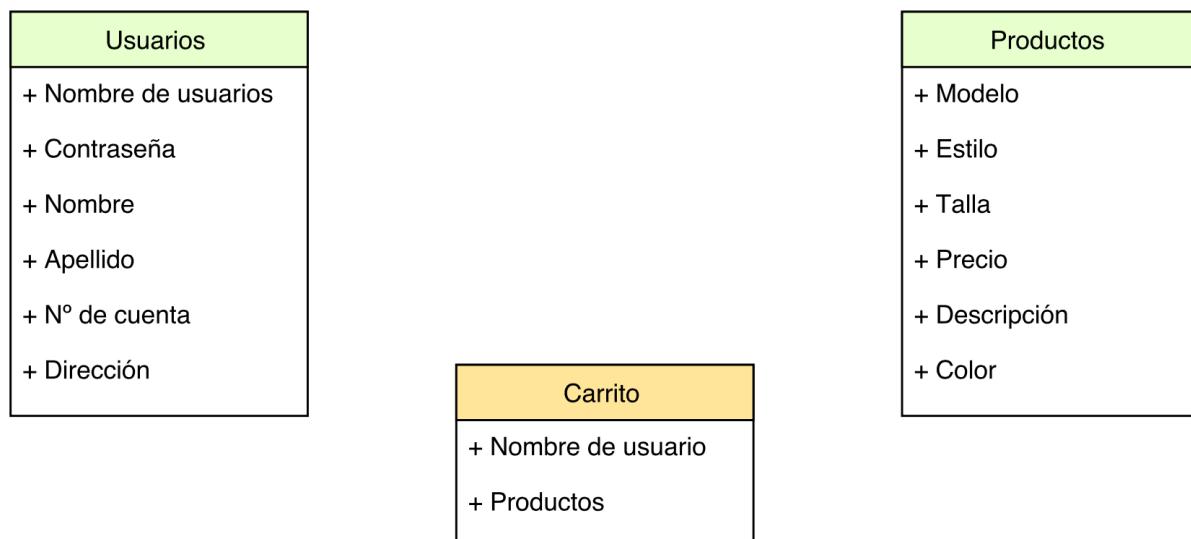
3.4.3 - Bases de datos utilizadas: MongoDB y Redis

Para la realización del ejemplo se han implementado tres bases de datos distintas, la primera almacena los datos sobre los usuarios registrados (nombre de usuario, correo electrónico, contraseña etc.), la segunda guarda información sobre los productos disponibles en stock (modelo, talla, color, precio etc.) y por último la tercera, es la encargada de almacenar los productos seleccionados por el usuario durante la sesión (función carrito de la compra).

Tanto la base de datos de usuarios como la de productos se han desarrollado utilizando MongoDB. Se trata de una base de datos NoSQL escrita en código abierto, en la que se guardan los documentos en objetos tipo JSON (en formato BSON) los cuales son muy fáciles de transmitir a través de HTTP. Además Este tipo de base de datos no solo ofrece gran disponibilidad y escalabilidad sino que además encaja perfectamente con Node JS por lo que es la ideal en esta aplicación ejemplo.

Por otro lado, la tercera es una base de datos basada en Redis. Esta base de datos también es NoSQL y de tipo clave-valor (lo que nos permite asociar productos a usuarios). Además ofrece posibilidades de expiración de datos. Esto es exactamente lo que buscamos a la hora de implementar la función “carrito de la compra” tal y como se ha detallado anteriormente en el apartado destinado a la descripción de los microservicios, ya que no parece lógico que la selección de un producto por parte de un usuario se almacene indefinidamente ocupando espacio de manera innecesaria.

Los datos que vamos a almacenar en las tres bases de datos sobre las que se apoyan los microservicios aparecen en la siguiente imagen:



3.4.4 - Comunicación entre microservicios

Como se ha comentado entre las ventajas que aporta la elección de Node JS para desarrollar la aplicación se ha decidido utilizar el protocolo HTTP (HyperText Transfer Protocol) para la comunicación entre microservicios.

En este ejemplo de aplicación la comunicación entre microservicios es bastante sencilla. Este protocolo ofrece todas las características necesarias para el buen funcionamiento de la aplicación global. En apartados anteriores donde se describe más en detalle la funcionalidad de cada microservicio web, en algunas ocasiones las peticiones HTTP serán de tipo GET o de tipo POST (será de tipo POST en aquellos microservicios donde se requiere enviar ciertos parámetros que pueden ser delicados para recibir un determinado recurso o actualizarlo). Se ha tomado esta elección ya que, a pesar de que no se trata de una aplicación que se vaya a comercializar, en cuestiones de seguridad y protección de los datos enviados, es más seguro el envío de datos a través de peticiones POST (donde los datos permanecen “ocultos”) que a través de peticiones de tipo GET (donde los datos se transmiten a través de la URL).

En cualquier caso las peticiones HTTP se configuran en el microservicio de gestión de usuarios y peticiones. Esta configuración sigue siempre el mismo esquema salvando el cuerpo de la petición que variara en función del microservicio hacia al que va destinada la petición.

Los campos que van a componer la petición son los siguientes:

- Tipo: como ya se ha explicado anteriormente, en función de la necesidad del microservicio, la petición variará su tipo entre POST y GET.
- Uri: se trata del destino de la petición. Como se puede ver en el Anexo 2, en el código está predefinido el destino de cada petición. Dentro de cada uno de los microservicios, se gestionarán las peticiones recibidas de manera diferente, y se redirigirán al módulo concreto encargado de realizar la función correspondiente.
- Form: se trata del cuerpo de la petición, generalmente se incluirán parámetros con un formato clave-valor. Que seguirán un esquema previamente “acordado” con el microservicio al que irán dirigidas.

El envío de datos a través del protocolo HTTP (cuando se produce una respuesta a la petición) se realizará sobre un formato de tipo JSON, ya que es sencillo el análisis de este tipo de archivos por parte de los módulos que componen los microservicios.

Es muy importante que la comunicación entre microservicios funcione de manera perfecta, ya que como veremos en apartados posteriores, la realización de múltiples peticiones HTTP insta a que se produzcan más fallos que en el caso de aplicaciones con arquitectura monolítica donde la comunicación interna es más fiable. Es por tanto, este punto uno de los puntos considerados como crucial a la hora de la implementación de arquitecturas basadas en microservicios.

3.5 - Características generales de los microservicios

Para entender las características de las arquitecturas basadas en microservicios y de los microservicios como tal, se debe dar un salto conceptual sobre los conocimientos acerca del desarrollo de aplicaciones web que se tiene hasta la fecha.

Anteriormente se ha visto una descripción general de este tipo de arquitecturas, ahora se va a entrar en detalle en las características más importantes y equipararlas o ejemplificarlas a través de la aplicación web desarrollada.

3.5.1 - Independencia

Una de las características más importantes que se debe cumplir de manera estricta para que se puedan optimizar al máximo las ventajas que ofrece este modelo arquitectónico, es la independencia entre microservicios.

Como vamos a detallar posteriormente, esta característica va ofrecer un rango inmenso de ventajas (escalabilidad, flexibilidad, protección ante fallos...). Los microservicios deben ser desplegados de manera totalmente independiente unos de otros. También es importante conocer el límite entre ellos, ya que en ocasiones, un concepto difuso de este límite puede provocar que el desarrollo que estemos realizando sea incorrecto. Un buen conocimiento del límite va a facilitar la realización de las comunicaciones entre microservicios así como una rápida localización y corrección de errores en caso de fallo.

En nuestro ejemplo, cada uno de los microservicios (más allá de las bases de datos) es desplegado de manera individual dentro de cada contenedor Docker, que aunque algunos de ellos estén conectados entre sí también corren de manera individual. La realización del despliegue de las microservicios y de las bases de datos sobre contenedores Docker se explica en detalle en los anexos. Además, si se observa el código subido en Github se puede ver que las servicios están escritos como aplicaciones individuales, que pueden funcionar de manera independiente, aunque la funcionalidad global de la aplicación depende de todos ellos.

3.5.2 - Automatización en el despliegue

Para que la arquitectura sea tan ágil y flexible como deseamos, esta característica también toma mucha importancia, en nuestro ejemplo, nos hemos apoyado en Docker para “cumplir” con este requisito.

La tecnología Docker es imprescindible para que se cumpla esta característica. Este framework ofrece de manera implícita una automatización en el despliegue de aplicaciones sobre contenedores y en la ejecución de los mismos.

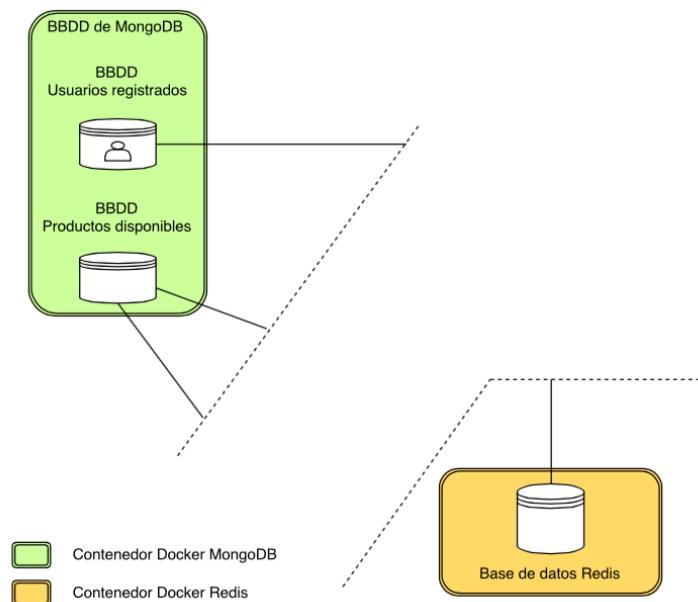
Esta automatización se realiza a través del desarrollo de un documento de configuración que permite que con dos líneas de código a través del terminal el microservicio funcione de la manera deseada. No obstante como ya se ha comentado previamente, en el anexo 1 se puede ver en detalle el despliegue de aplicaciones Node Js, poniendo como ejemplo la desarrollada, y bases de datos sobre contenedores, y la posterior ejecución de estos.

3.5.3 - Descentralización en el manejo de las bases de datos

Esta es de nuevo una característica clave en las arquitecturas basadas en microservicios, ya que en ella se basa (junto con la independencia de cada microservicio) la que probablemente sea la ventaja más importante que ofrece este modelo, la escalabilidad.

En las aplicaciones monolíticas normalmente se accede desde una única capa de acceso a datos a las bases de datos, las cuales normalmente son todas del mismo tipo, o incluso en muchas ocasiones se trata de una única base de datos de gran tamaño que guarda múltiples tablas que no tienen porque tener relación entre sí.

En nuestro ejemplo se puede ver la descentralización en las siguientes partes del esquema:



Se aprecia que, los microservicios acceden a tres bases de datos completamente independientes entre sí (salvo por el hecho de que dos de ellas se encuentran en el mismo contenedor desplegadas). Esto ofrece muchas ventajas ya que se pueden replicar las bases de datos que necesiten mayor nivel de disponibilidad en función de las necesidades del sistema.

Además, también ofrece la ventaja de poder implementar diferentes tipos bases de datos que se pueden ajustar de nuevo a las funcionalidades de cada microservicio, lo que aportará una mejora en la eficiencia global del sistema. Esto proporciona una gran ventaja para el desarrollador, que en ocasiones tiene que realizar “malabarismos” para que un único tipo base de datos encaje en todas las funciones por diferentes que sean.

Además, como también se puede ver en el esquema de la aplicación, la única ventaja que puede ofrecer un sistema de gestión de datos único, que reside en que los cambios producidos por un servicio se mantengan para que resto de servicios que accedan a la base de datos “puedan ver” esos cambios (clásico problema de concurrencia en bases de datos que utilizan sistemas de gestión de ficheros por ejemplo), se puede seguir manteniendo en el caso de utilización de microservicios. En la aplicación desarrollada, tanto el microservicio de búsqueda, como el de ordenación de catálogo, concurren en el acceso a la base de datos. Esto permite que una única actualización de la base de datos, influya en ambos microservicios.

3.6 - Ventajas que aportan las arquitecturas basadas en microservicios

En el apartado anterior hemos visto las características más importantes que deben poseer los microservicios y las arquitecturas basadas en ellos para que sean considerados como tal. Evidentemente se podrían enumerar un sinfín de características hasta completar la definición de este tipo de arquitecturas, pero muchas de ellas pueden ser “omitidas” o relegadas a un segundo plano en función de las necesidades que requiere la aplicación web en concreto que se desea desarrollar. No obstante, las anteriormente expuestas son las que se consideran de mayor relevancia.

Estas características generan multitud de ventajas con respecto a las aplicaciones que siguen una arquitectura clásica o monolítica. En este apartado se pretende profundizar en las ventajas más importantes, y de nuevo llevando un paralelismo con la aplicación web de ejemplo desarrollada, mostrar en qué partes afectan o donde se aprecian los cambios más significativos.

3.6.1 - Disminución del impacto provocado por fallos

La disminución del impacto ante la ocurrencia de un fallo es una gran ventaja con respecto al gran impacto que puede provocar un fallo, por pequeño que sea, en una aplicación web que sigue el modelo clásico.

Es prácticamente inevitable, que en algún momento durante la exposición de la aplicación web a todos los usuarios se produzca ningún fallo. Estos fallos se pueden producir por muy diversos motivos, una saturación a la hora de pedir recursos desde los usuarios, la caída de una aplicación externa de la que depende alguna parte la nuestra, o simplemente un error en el desarrollo pasado por alto por el equipo encargado de realizar los test y las pruebas pertinentes, entre muchos otros. Por este motivo, una vez aceptado que la ocurrencia de un fallo es altamente probable, hay que intentar que cuando ocurra dicho fallo, la funcionalidad de la web permanezca lo más intacta posible y el impacto sobre el usuario sea mínimo.

Mediante la implantación de microservicios, la reducción del impacto se puede hacer de manera relativamente sencilla. Al tratarse de servicios prácticamente independientes entre sí, el fallo de alguno de ellos se puede aislar en la mayoría de los casos para que solo quede afectado el servicio concreto en el que se ha producido.

Además, otra ventaja relacionada con la repercusión que puede llegar a tener un fallo, es la agilidad y rapidez a la hora de detectar donde se ha producido y poder corregirlo con la mayor brevedad posible. En ocasiones, en una aplicación monolítica, un fallo puede provocar una reacción en cadena, lo que puede llevar a cierta confusión a la hora de saber cual ha sido el desencadenante principal del fallo. En Arquitecturas basadas en microservicios sin embargo, al provocar un fallo la inhabilitación de un servicio, y al ser este servicio de un tamaño reducido, la corrección del error será muy rápida y eficaz.

En la aplicación desarrollada (como se puede ver en el código), se ha pretendido que un fallo, bien en uno de los microservicios o bien en el envío o recepción de una petición HTTP, inhabilite ese servicio mostrando una página de error que permita seguir utilizando el resto. No obstante, este sistema de fallos es posible mejorarla y “afinarla” mucho más de tal modo que prácticamente no se aprecien algunos de los fallos menos importantes.

3.6.2 - Heterogeneidad de sistemas

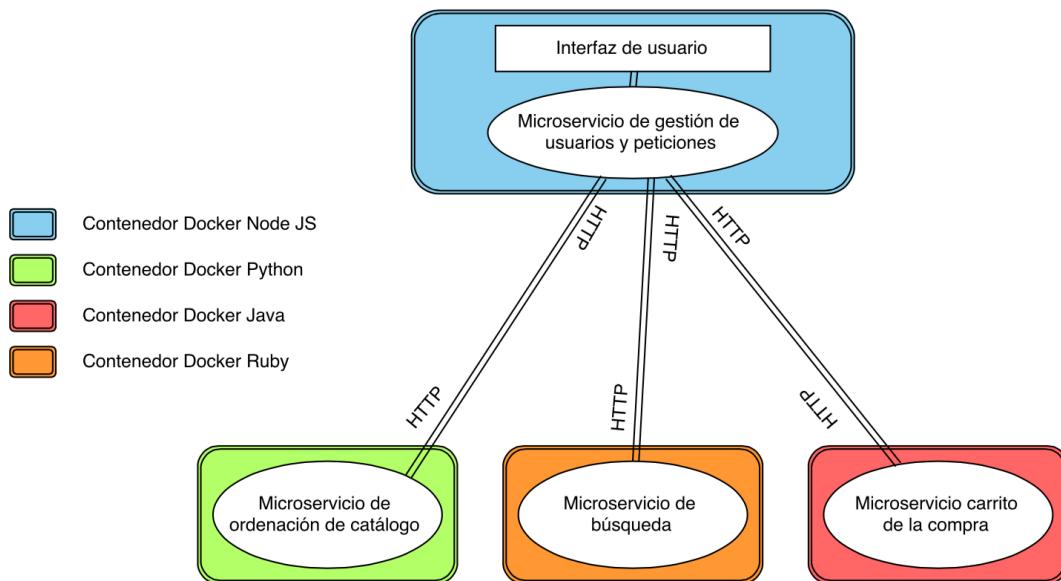
En una aplicación que sigue la arquitectura clásica o monolítica, la totalidad de servicios que se describen en el sistema deben de estar escritos en el mismo lenguaje y empaquetados y compilados por el mismo constructor, ya que en caso contrario sería imposible el despliegue en conjunto de la aplicación.

La característica de independencia y despliegue individualizado de los microservicios, nos permite que cada uno de ellos estén desarrollados y desplegados sobre el lenguaje y la tecnología que más se ajuste a la funcionalidad que pretenden proporcionar.

La heterogeneidad en los sistemas dentro de cada microservicio es posible gracias a que solo tienen una funcionalidad en común, la comunicación entre ellos. Por lo tanto cualquier tipo de lenguaje puede ser utilizado siempre y cuando sea capaz de converger en el sistema de comunicación utilizado (normalmente HTTP) con el resto de microservicios.

En nuestro ejemplo hemos utilizado en todos los microservicios Node JS y han sido desplegados todos ellos sobre contenedores Docker. La heterogeneidad en este caso se puede observar más en el acceso a los diferentes tipos de bases de datos utilizadas. No se entrará en este apartado a comentar dicho tema ya que se ha detallado en profundidad en el punto 3.3.3 “Descentralización en el manejo de bases de datos”.

No obstante, y tal y como se va a comentar en uno de los últimos apartados de este TFG, puede ser una buena línea de investigación futura el desarrollo de la misma aplicación en diferentes lenguajes y tecnologías, y la comparación en cuestiones de eficiencia y escalabilidad con el ejemplo base ya desplegado. Una posibilidad podría ser algo así:



Esta heterogeneidad junto con la brevedad de los microservicios va a permitir “perder el miedo” a la actualización hacia nuevas tecnologías y lenguajes, ya que siempre tendremos la posibilidad de actualizar una parte de nuestra aplicación que no sea demasiado importante sin perder la posibilidad de volver a la aplicación origen con poco esfuerzo.

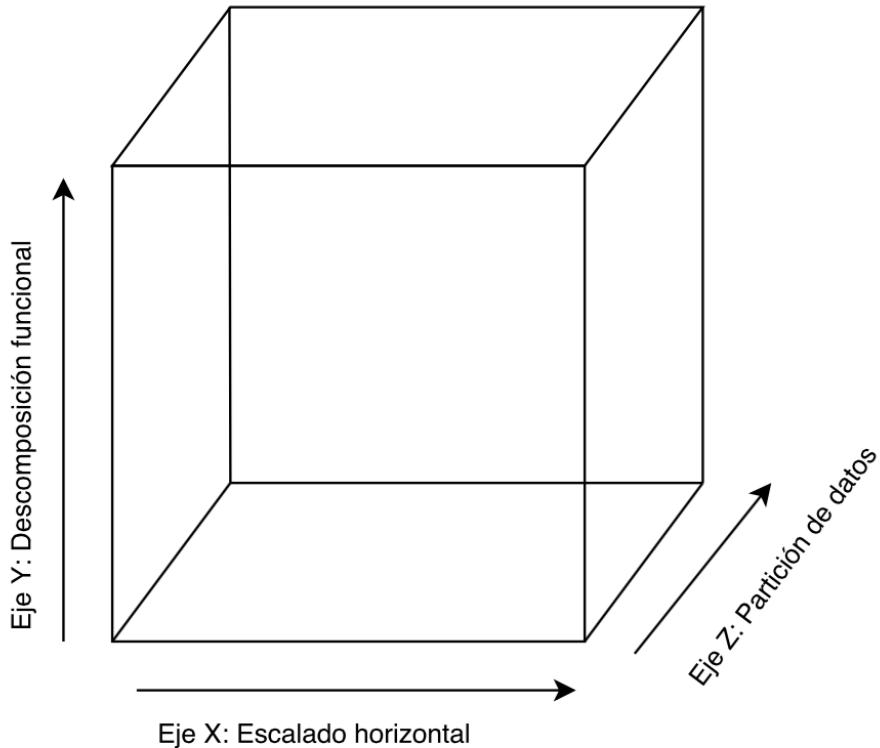
3.6.3 - Escalabilidad eficiente

En el momento en el que se mueve el sector hoy en día, donde tanto el número de usuarios que navegan por internet como los recursos que requieren las nuevas aplicaciones aumenta a pasos agigantados, la escalabilidad eficiente se vuelve vital.

Se trata de una de las grandes motivaciones que han provocado el emerger de los microservicios, ya que es uno de los grandes defectos de las aplicaciones monolíticas. Normalmente, si se ha desarrollado una aplicación con estructura monolítica, cuando es necesario replicar los recursos para que un número mayor de usuarios puedan tener acceso a los mismos manteniéndose el rendimiento, se tiene que replicar la aplicación en su totalidad, ya que se trata de un bloque compacto y generalmente indivisible.

Sin embargo, en arquitecturas basadas en microservicios, se puede replicar cualquiera de los microservicios de manera totalmente individual. Esto va a permitir optimizar la eficiencia del escalado en gran medida, ya que solo será necesario reproducir aquellos microservicios que sean o bien los más utilizados por el usuario, o bien los que por sus características tengan que tener una disponibilidad muy alta para evitar fallos que afecten en gran medida al usuario.

Para definir este tipo de escalabilidad y compararla con la escalabilidad que se realiza sobre aplicaciones monolíticas de manera gráfica, Martin L. Abbot y Michael T. Fisher, en su libro “The art of scalability” [5], equiparan el escalado de una aplicación a las dimensiones de un cubo, dicho cubo se puede ver en la siguiente imagen:



- Fuente de la imagen: <http://microservices.io/articles/scalercube.html> -

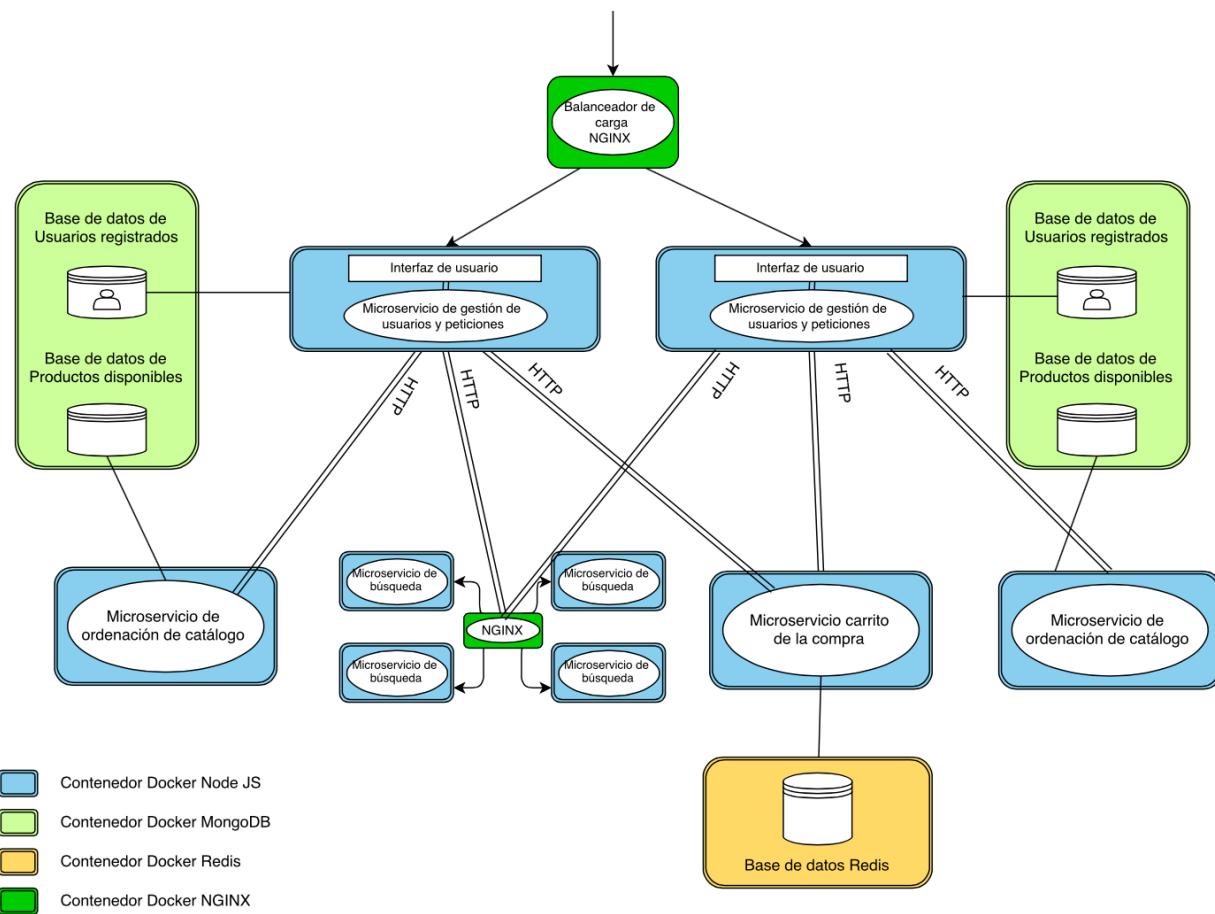
En el cubo se representa a través de las dimensiones del mismo, las diferentes posibilidades a la hora de escalar una aplicación web. Mediante el uso de arquitecturas basadas en microservicios se pueden aprovechar al máximo las posibilidades que nos ofrece el escalado. A continuación se expone una explicación sobre cada uno de los ejes representados:

- Eje X: es la opción de escalado que nos ofrecen las aplicaciones clásicas o monolíticas. Se denomina escalado horizontal y consiste en la simple replicación de la aplicación web en su totalidad. Para dividir el tráfico hacia los distintos servidores en los que corre cada copia de la aplicación, se utiliza un balanceador de carga que redirecciona las peticiones de los nuevos usuarios hacia el servidor más descargado. El problema que tiene el escalar de esta manera ya se aprecia a simple vista, la eficiencia del escalado. Un escalamiento horizontal de una aplicación monolítica, obliga a repetir la aplicación al completo en diferentes servidores, lo que puede suponer en muchas ocasiones un gran desperdicio de espacio, ya que no siempre es necesario replicar todas las funcionalidades de una aplicación para poder dar servicio a un número mayor de usuarios.
- Eje Z: se trata de un escalado que posee similitudes al que hemos visto en la descripción del eje X. También se realiza una copia del código de la aplicación que se despliega en tantos servidores como se deseé para cumplir con los requisitos. La diferencia es que en este caso cada servidor tiene acceso a una partición de los datos diferente. En alguna parte del sistema debe haber un componente que hace de encaminador para que los clientes sean redirigidos a un servidor determinado. Esto permite aumentar la disponibilidad de los datos alojados en las diferentes bases de datos, y también la posibilidad de hacer una distinción en la calidad de acceso de los clientes, ya que en función de unos parámetros determinados podrían ser dirigidos a diferentes servidores. También posee ciertas desventajas, como el aumento de la complejidad de la aplicación.
- Eje Y: el escalado a través de este eje es el salto en la optimización de la escalabilidad que nos ofrecen las arquitecturas basadas en microservicios. A diferencia de el escalado sobre el eje Z o sobre el Eje Y este tipo no consiste en la replicación total de la aplicación sobre diferentes servidores, sino que solamente se repiten los microservicios que más “demanda” tienen por parte de los usuarios. La complejidad de este escalado reside en dos puntos clave, el conocer cuales deben ser los límites de cada servicio, y el ser capaz de mantener las comunicaciones entre los mismos a pesar de que se hayan replicado de manera individual.

El despliegue de una aplicación web sobre arquitectura basada en microservicios, normalmente se va realizar sobre aplicaciones en las que se espera un gran aumento en el número de usuarios que la utilicen, por lo que una escalabilidad eficiente puede suponer un ahorro en términos de costes de mantenimiento muy elevado.

En nuestro caso, el ejemplo realizado no es comercializable, y no se ha realizado un escalado bajo ninguno de los tres ejes diferentes estudiados, no obstante en la página siguiente se va a proponer un esquema con un posible escalado. También en el apartado destinado líneas de investigación futura se va a proponer la realización de un estudio y la comparación del rendimiento al realizar un escalado utilizando los tres ejes.

En el siguiente esquema se muestra un posible escalado* para los microservicios desplegados en el ejemplo de aplicación desarrollada.



*Las conexiones entre los microservicios de búsqueda y las bases de datos de productos disponibles no se muestran en el esquema ya que harían prácticamente ilegible el dibujo.

Esta propuesta de escalado se ha hecho en un supuesto caso de que el microservicio de búsqueda sea el más utilizado, y el servicio de carrito de la compra el menos demandado por los usuarios, algo que parece lógico a priori en una aplicación web destinada a la venta online. No obstante no deja de ser un escalado “irreal” ante un incremento ficticio del número de usuarios que acceden a la aplicación, debería hacerse un estudio a través de las peticiones de los usuarios que permitiera conocer exactamente cuales son los microservicios más utilizados.

Tampoco se muestra en el esquema el montaje del sistema de las bases de datos, que tendrían que estar coordinadas para que en el caso de que cambien los datos sobre alguna de ellas ese cambio de produjera también en el resto de bases de datos destinadas a alimentar los mismos microservicios.

Se ha propuesto utilizar un balanceador de carga Nginx ya que existen diferentes ejemplos públicos en internet donde se ha utilizado sobre Docker con buenos resultados, no obstante también debería realizarse un estudio y comparación de diferentes tecnologías destinadas a este propósito.

3.6.4 - Flexibilidad y facilidad en la actualización

Más allá de los cambios que se deban realizar por un rápido incremento de usuarios, lo que provoca un problema de escalabilidad tal y como se ha tratado en el apartado anterior, son múltiples los posibles motivos que pueden obligar al desarrollador a realizar un cambio.

Estos cambios pueden ser por una evolución en las tecnologías (puede ocurrir que la migración a un nuevo lenguaje o tecnología mejore alguna funcionalidad de la aplicación en gran medida), o simplemente porque las necesidades del cliente sobre la aplicación que se ha realizado hayan cambiado, entre muchos otros motivos. Si se da alguna de estas circunstancias, el desarrollador se va a ver obligado a realizar una modificación en el código y un “redespliegue” de la aplicación.

El proceso de actualización de una aplicación puede tornarse muy complicado en el caso de aplicaciones con arquitecturas monolíticas, ya que por pequeño que sea el cambio, hay que asegurarse de que la totalidad de la aplicación siga funcionando correctamente lo que puede hacer muy complejo el sistema de pruebas. Además es probable que durante la actualización, el sistema global no esté funcionando normalmente, lo que puede afectar al usuario de manera directa.

Sin embargo a través del uso de arquitecturas basadas en microservicios, la actualización se va a realizar solamente sobre el microservicio que sea necesario por lo que el resto de la aplicación no debe verse afectada al realizar el cambio.

Esta capacidad para realizar cambios de manera ágil, va a permitir que las aplicaciones desarrolladas utilizando microservicios estén siempre a la vanguardia en el uso de nuevas tecnologías. Esto es debido a que tal y como se ha detallado en el apartado 3.4.1 “disminución del impacto provocado por fallos”, a pesar de que una actualización o prueba en uno de los microservicios provoque un fallo, este fallo debe ser fácilmente detectable y corregible, y en ningún caso debe afectar al total de la aplicación.

También se van a producir actualizaciones más constantes y sobre los microservicios que más lo necesiten, ya que, como veremos en un apartado posterior dedicado a profundizar en las ventajas en la organización del equipo de desarrollo, el equipo al completo tiene la posibilidad de centrarse en la mejora de un único microservicio (el más prioritario), de este modo se va mejorar la usabilidad de la aplicación al usuario invirtiendo poco tiempo.

Como se ha podido ver en las descripciones de las diferentes posibilidades que ofrece la flexibilidad en este tipo de aplicaciones, esta ventaja se vuelve a apoyar en los principios básicos de los microservicios, la independencia, la heterogeneidad de sistemas (que es la cualidad que nos va a permitir el migrar solo un servicio a una nueva tecnología) y el despliegue independiente (permitirá la actualización de un único servicio).

Es difícil de conocer hasta qué punto puede ser aprovechable la flexibilidad en las arquitecturas basadas en microservicios, ya que va a depender en gran parte del tipo de aplicación web realizada y de como cambian las necesidades de la empresa para la que se realice a lo largo del tiempo. No obstante, en comparación con las aplicaciones que siguen la arquitectura clásica, las ventajas en ningún caso van a ser menores.

En nuestro ejemplo, esta gran ventaja de nuevo carece de sentido ya que se va aprovechar principalmente en proyectos a largo plazo, en los que en cualquier momento pueda surgir la necesidad de un cambio.

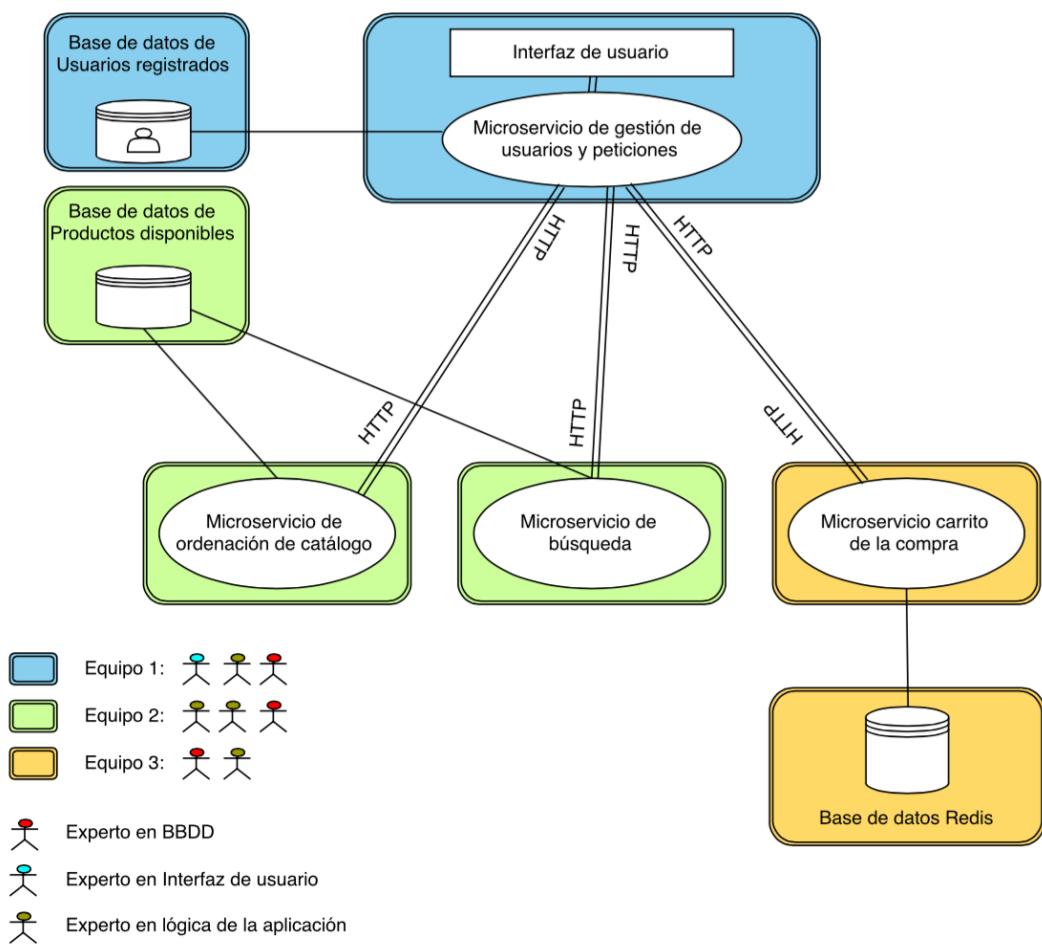
3.6.5 - Capacidad organizativa del equipo de trabajo

La forma de organizar un equipo de trabajo en un proyecto de desarrollo software, va a afectar de manera directa tanto al rendimiento del equipo, como al resultado final, visible en forma de aplicación web.

Normalmente, cuando se realiza una aplicación con una arquitectura monolítica el equipo de trabajo se divide para desarrollarla en las tres partes principales de esta, la interfaz de usuario, la lógica de la aplicación y el sistema de almacenamiento de datos. Este esquema favorece el aprovechamiento de expertos en alguna de las tres capas, ya que exclusivamente se van a dedicar a su parte del proyecto. Sin embargo, este sistema organizativo puede provocar muchas pérdidas de tiempo en reuniones de coordinación a la hora de hacer converger las tres capas hacia la aplicación final.

El uso de microservicios, nos permite realizar una organización que puede ser más eficiente en equipos de desarrollo grandes. En un principio el equipo de trabajo se va a dividir en grupos que tendrán como objetivo la creación de un microservicio. En este nuevo esquema organizativo, los grupos en lugar de dedicarse al desarrollo de una única capa, serán multifuncionales por lo que estarán compuestos por expertos en diferentes aspectos. Al hacer la división en pequeños grupos, la coordinación va a ser mucho más ágil por lo que el rendimiento global del equipo de trabajo aumentará.

A continuación se muestra de manera gráfica una comparación entre como se dividiría la aplicación desarrollada si fuera una aplicación monolítica y como se propone un posible sistema organizativo sobre el ejemplo de aplicación desarrollado:



3.6.6 - Eficiencia

Actualmente, rara es la aplicación corporativa que posee sus propias infraestructuras para almacenamiento y procesado de datos en lugar de utilizar lo que hoy conocemos como “computación en la nube”. Esta nueva forma de ofrecer servicios, permite externalizar las infraestructuras y acceder a ellas de una manera flexible en función de las necesidades de cada momento, lo que supone un ahorro en el mantenimiento de los servicios desplegados inimaginable hace solo algunos años.

A través del uso de microservicios esta eficiencia crece exponencialmente ya que el despliegue “en la nube” se puede hacer de una manera ultraflexible. Aunque puede parecer una utopía, se prevé que esta unión de microservicios y el “cloud computing” o “computación en red”, permita el mantener algunos de los servicios en un estado de “suspensión” o “espera”, y sea la propia lógica de la aplicación la que decida cuando debe activarse esa máquina virtual en la que se aloja el servicio para proporcionar la funcionalidad requerida en cada momento.

Más allá del ahorro energético o en costes de mantenimiento que nos facilita el uso en conjunto de arquitecturas basadas en microservicios y el cloud computing, también es evidente que para un equipo de desarrolladores, el uso de microservicios puede permitirles un ahorro en inversión y tiempo incalculable.

Si una empresa se dedica a la creación, despliegue y mantenimiento de aplicaciones web, puede aprovecharse mucho del uso de una arquitectura basada en microservicios. Normalmente este tipo de empresas gestionan multitud de aplicaciones web, y muchas de ellas tienen partes en común o incluso pueden llegar a ser prácticamente idénticas (este sería el caso por ejemplo en el que una empresa haya desarrollado varias aplicaciones e-commerce como la exemplificada para diferentes tiendas). En este caso, el uso de microservicios va a permitir a la empresa la reutilización de los microservicios ya desarrollados.

Como se ha comentado en apartados anteriores, los microservicios solamente van a tener como punto en común la comunicación entre ellos, por lo que si una empresa dedicada al desarrollo sigue siempre la misma “filosofía” a la hora de programar sus aplicaciones, esta reutilización de microservicios se va a poder realizar de manera sencilla.

Estas son las principales ventajas que ofrecen las arquitecturas basadas en microservicios, no obstante, el uso de esta arquitectura también posee algún inconveniente. En el siguiente apartado se van a comentar los principales puntos débiles del uso de microservicios frente a arquitecturas monolíticas.

A pesar de los posibles inconvenientes que se van a comentar, si pusiéramos sobre una balanza ambas partes, se puede ver que el uso de microservicios es bastante conveniente en la mayoría de aplicaciones.

3.7 - Puntos débiles de las arquitecturas basadas en microservicios

En el apartado anterior se han mostrado las ventajas más importantes que podemos obtener del uso de microservicios, no obstante, el uso de este tipo de arquitectura para el desarrollo de aplicaciones web, como cualquier otro, posee ciertos puntos débiles.

En este apartado se van a tratar los puntos débiles más importantes de las arquitecturas basadas en microservicios, con el fin de hacer ver al lector que no todo son ventajas, pero que estos tipos de inconvenientes se pueden reducir en gran medida a través de una buena organización e integración de los microservicios.

3.7.1 - Complejidad de la aplicación

Se trata del principal inconveniente en la implementación de aplicaciones sobre arquitecturas basadas en microservicios.

Con respecto a arquitecturas monolíticas, si se quieren desarrollar microservicios hay que dar un salto conceptual. Este nuevo modelo rompe con muchas de las reglas básicas sobre las que se apoya el modelo clásico. Para algunos desarrolladores o equipos de desarrollo acostumbrados a seguir un esquema disciplinario para la elaboración de proyectos software de este tipo, puede ser complejo el acoplarse a la nueva metodología que se propone a través del uso de microservicios.

Es evidente que el desarrollo de aplicaciones sobre las arquitecturas estudiadas en este Trabajo Fin de Grado va a proporcionar al desarrollador nuevos quebraderos de cabeza a la hora de programar. En este nuevo estilo se tienen que abordar nuevos problemas, como la comunicación entre microservicios y la integración de todos ellos para conformar una única aplicación, problemas que no aparecían en las arquitecturas monolíticas.

Por estos motivos resulta lógico que alguien que empieza a conocer el mundo de las aplicaciones web, se inicie en el desarrollo aprendiendo sobre aplicaciones monolíticas, y si en algún momento se plantea la realización de una aplicación que va a necesitar grandes recursos y sobre la que se prevé un aumento a corto/largo plazo de los usuarios que la van a utilizar, tenga en cuenta la posibilidad de implementar microservicios.

3.7.2 - Difícil migración desde aplicaciones monolíticas

Uno de los motivos por los que algunas de las principales empresas no dan el salto a microservicios es esta dificultad.

En muchas ocasiones puede resultar extraordinariamente complejo la migración de una aplicación con una arquitectura clásica a una arquitectura basada en microservicios. Son numerosas las aplicaciones en las que la mayoría de funcionalidades dependen unas de otras, es en estos casos cuando la división en microservicios de las funcionalidades de una aplicación ya desplegada puede resultar muy complejo.

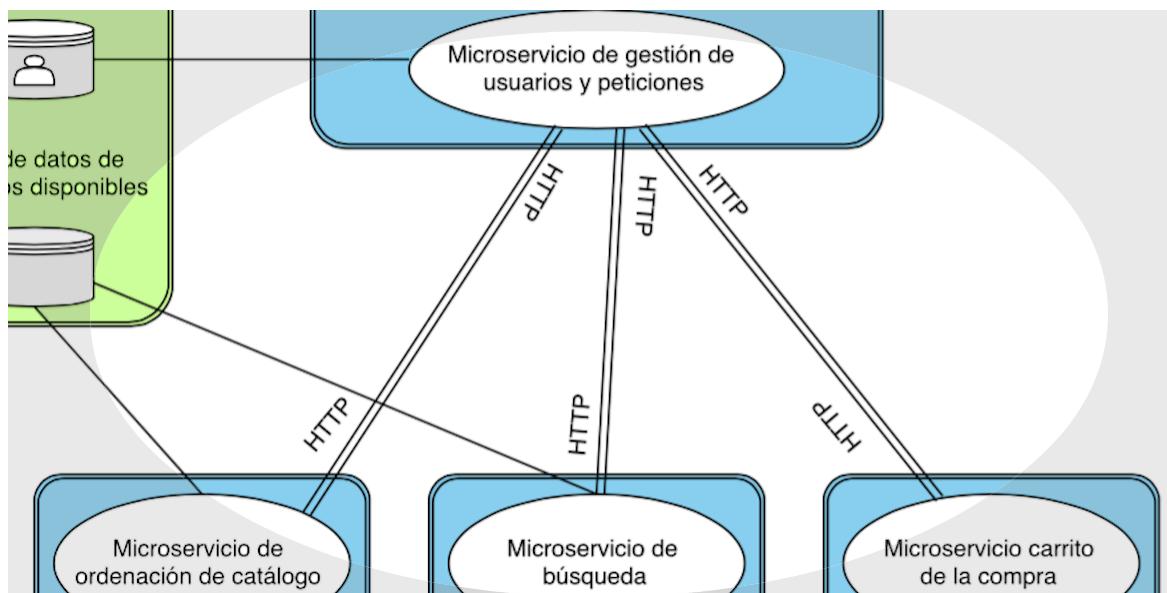
Por eso, muchos de los desarrolladores que quieren migrar su aplicación a microservicios, eligen reescribirla de nuevo para darle un enfoque diferente, esta opción siempre va a funcionar, y además puede orientarse el nuevo enfoque de tal forma que se optimicen los recursos mucho más.

3.7.3 - Comunicación entre microservicios

Como ya se ha comentado en otros apartados, una buena implementación de la comunicación entre microservicios es crucial para que la aplicación funcione de manera fluida y el cliente no aprecie negativamente que esta usando una aplicación basada en este tipo de arquitectura.

También se ha comentado anteriormente que para el ejemplo realizado se ha utilizado el protocolo HTTP. Dependiendo del motivo por el que este desplegada la aplicación este protocolo puede no cumplir con todas las características buscadas. Bien por cuestiones de seguridad, como veremos en el siguiente apartado o de sincronismo, varios protocolos van a tener que “convivir” en una misma aplicación, viéndonos obligados en ocasiones a programar “traductores” para compaginar las diferentes formas de comunicación entre microservicios.

En la aplicación desarrollada este punto débil se refleja en la siguiente imagen, donde se pueden ver las principales comunicaciones (obviando las que se realizan entre microservicios y bases de datos) en las que intervienen los microservicios.



En este caso se trata de una aplicación sencilla por lo que no hay demasiados puntos de comunicación entre microservicios.

Muchos de los fallos que se producen en aplicaciones web, vienen provocados por la caída de un enlace o por un fallo en los datos enviados, en aplicaciones complejas, estos enlaces se multiplican exponencialmente, y con ello las posibilidades de que uno falle en algún momento.

Aun así, tal y como hemos visto en apartados anteriores, el impacto de fallo va a ser mucho menos que en el caso de que se produjera sobre una aplicación que sigue la arquitectura clásica, por lo que este defecto puede paliarse en gran medida si el sistema de prevención de fallos ha sido diseñado correctamente.

3.7.4 - Seguridad en las comunicaciones

Este punto esta directamente relacionado con el anterior, pero se ha querido hacer una mención especial ya que en muchas aplicaciones la seguridad de los datos que se transportan (entidades bancarias, aplicaciones en las que se realice algún tipo de pago, o aplicaciones que trabajen sobre informes médicos entre otras) son de gran importancia y requieren de la máxima confidencialidad posible.

Son múltiples los ataques informáticos que se pueden realizar sobre cualquier tipo de aplicación web. El estudio de este tipo de ataques así como sus consecuencias no es objeto de estudio en este TFG por lo que no se van a enumerar. No obstante si merece la pena mencionar que una mala implementación en este tipo de arquitectura puede favorecer que los atacantes tengan un más fácil acceso a algunos de los datos que se manejan en la comunicación.

Al utilizar una arquitectura basada en microservicios, tal y como se ha visto en el apartado anterior, las comunicaciones entre partes de la aplicación aumentan considerablemente. Estos pueden ser puntos en los que los atacadores intenten hacerse con los datos transmitidos. Evidentemente las posibilidades de éxito aumentan si no se presta suficiente atención a los protocolos de comunicación empleados.

Una buena práctica puede ser el cifrado de estos datos antes de ser transmitidos, para que en el caso de que un ataque tenga éxito el impacto sea mínimo. Sin embargo, se debe encontrar el equilibrio entre que datos cifrar/descifrar y que datos no. En caso de decidir cifrar alguno de los datos que se transmiten, hay que estudiar encarecidamente que sistema utilizar para ello, ya que como hemos comentado las comunicaciones pueden llegar a ser muy numerosas, y un cifrado pesado en alguna de ellas puede provocar retardos que se van acumulando unos tras otros haciendo que la usabilidad y manejabilidad de la aplicación se vean muy reducidas.

Otro motivo por el que preocuparse en lo que a seguridad se refiere es la descentralización de los servicios. Normalmente este aspecto cuando se realiza una aplicación web que sigue una arquitectura monolítica es mucho más sencillo que en el uso de microservicios. Como se ha comentado en apartados anteriores, los microservicios van a estar ejecutándose muy probablemente en máquinas completamente independientes. Es por este motivo por el que hay que ser muy exhaustivo en la seguridad de manera individualizada de todas las máquinas en las que corran los servicios desplegados, un punto de acceso en cualquiera de ellos puede provocar que el atacante arruine la aplicación por completo.

4.- Conclusiones

Tras la realización del Trabajo Fin de Grado se puede concluir que se han conseguido los objetivos marcados al inicio del mismo.

A través del estudio y la lectura de múltiples artículos, tanto científicos como de opinión, que definen lo que se conoce como arquitecturas basadas en microservicios, se ha realizado un estudio teórico en el que se ha podido observar las características que poseen estas arquitecturas. Entrando en mayor profundidad en las que se han considerado más importantes o en su caso, las mas diferenciales con respecto a las arquitecturas monolíticas.

Además a través de estas características, se han expuesto las principales ventajas que pueden ofrecer al desarrollador la implantación de este tipo de estructura. También se ha realizado un estudio sobre los posibles puntos débiles que se puede encontrar el equipo de desarrollo a la hora de utilizar microservicios, lo que debe permitir poner más atención en la resolución de estos puntos, ya que en caso contrario pueden generar inconvenientes.

Por otro lado, se ha desarrollado una aplicación web basada en microservicios que ha hecho de hilo conductor aclaratorio a la hora de comentar alguno de los aspectos teóricos de este modelo arquitectónico.

A través de los anexos, se consigue que el lector despliegue la aplicación web desarrollada. De este modo se puede observar de manera gráfica alguno de los conceptos, y además se permite observar que el resultado final a la vista del usuario es el mismo que en el uso de aplicaciones monolíticas.

Personalmente estoy muy satisfecho con la elección y realización de este Trabajo Fin de Grado, ya que me ha permitido adquirir ciertos conocimientos que no había obtenido durante el desarrollo de mi formación, además de fortalecer las bases sobre ciertos aspectos sobre los que si había tenido cierta experiencia, como puede ser el desarrollo en el lado servidor, la configuración de base de datos etc..

En concreto me ha permitido conocer un modelo arquitectónico para la realización de aplicaciones web que se espera que va a tener mucho futuro, reforzar mi conocimiento sobre el desarrollo de este tipo de aplicaciones que había visto durante diferentes asignaturas en la carrera, y descubrir y aprender sobre una tecnología tan puntera como novedosa, clave en la virtualización de servicios como es Docker.

Creo y espero que este trabajo fin de grado sea de gran utilidad a todas aquellas personas interesadas en la implantación de esta arquitectura.

5.- Líneas de investigación futuras.

En este Trabajo Fin de Grado, tal y como se ha comentado en apartados anteriores, se ha estudiado un modelo arquitectónico de desarrollo de aplicaciones web muy novedoso, por lo que muchas de sus características, aplicaciones o ventajas todavía se encuentran bajo debate entre los desarrolladores. Además sobre este TFG simplemente se han expuesto las bases de microservicios, lo que abre múltiples líneas de desarrollo futuras. En este apartado se van a proponer alguna de ellas, no obstante las posibilidades son innumerables.

5.1 - Comparación del rendimiento de Tecnologías

Las dos principales tecnologías que se están utilizando para implementar microservicios a nivel de programación en el lado servidor son Java adicionalmente coordinando y construyendo los microservicios a través de Spring y Maven o Node JS con Express.

Una buena línea de investigación puede ser comparar el rendimiento de Java y Node JS, o incluso el de estas con nuevas tecnologías emergentes. Es evidente que dependiendo de la aplicación puede ser más conveniente el uso de una u otra y que no hay un lenguaje “definitivo”, sin embargo este análisis puede ser muy útil para desarrolladores que tengan que decidirse por una u otra, o incluso hacer una mezcla de ambas.

5.2 - Comparación de eficiencia

En este TFG se ha podido observar las diferentes ventajas de las arquitecturas basadas en microservicios, sin embargo, no se ha estudiado en qué medida, o bajo qué circunstancias estas ventajas son óptimas.

Una buena línea puede ser la investigación sobre en qué porcentaje se mejora la eficiencia en el escalado cuando se ha producido un incremento en los usuarios. Para ello debería simularse un gran número de peticiones y medirse parámetros como retardo de las peticiones, tamaño ocupado por las bases de datos o por la aplicación en sí, o eficiencia de los balanceadores de carga instalados. Y realizar estas peticiones sobre una petición monolítica y sobre la misma aplicación migrada a microservicios.

5.3 - Comparación de estilos

Como se ha visto en el apartado 3.1.1, existen dos vertientes dentro de las arquitecturas basadas en microservicios, el estilo orquestado y el estilo coreográfico.

Dentro de la comparación del rendimiento de tecnologías, podría ampliarse la comparación para realizarla sobre estos dos estilos. Si bien en este TFG se ha desarrollado un ejemplo con un estilo orquestado, puede ser muy interesante el desarrollar el mismo tipo de aplicación siguiendo un estilo coreográfico.

5.4 - Automatización del escalado

Son muchos los intereses económicos en este apartado, y múltiples las empresas que trabajan por refinar su sistema de escalado día a día.

Una monitorización de las peticiones que se hacen tanto sobre los microservicios, como la aplicación global, y un estudio sobre las mismas, es clave para saber qué módulos se tienen que escalar y en qué medida porque son los más utilizados, o porque son los que “no pueden fallar” bajo ninguna circunstancia.

A día de hoy es prácticamente una utopía la realización de una aplicación capaz de estudiar independientemente de la arquitectura este tipo de parámetros, y que tenga la inteligencia suficiente como para comunicar al administrador de la aplicación web qué microservicio o qué base de datos debe escalar.

7.- Bibliografía y Referencias

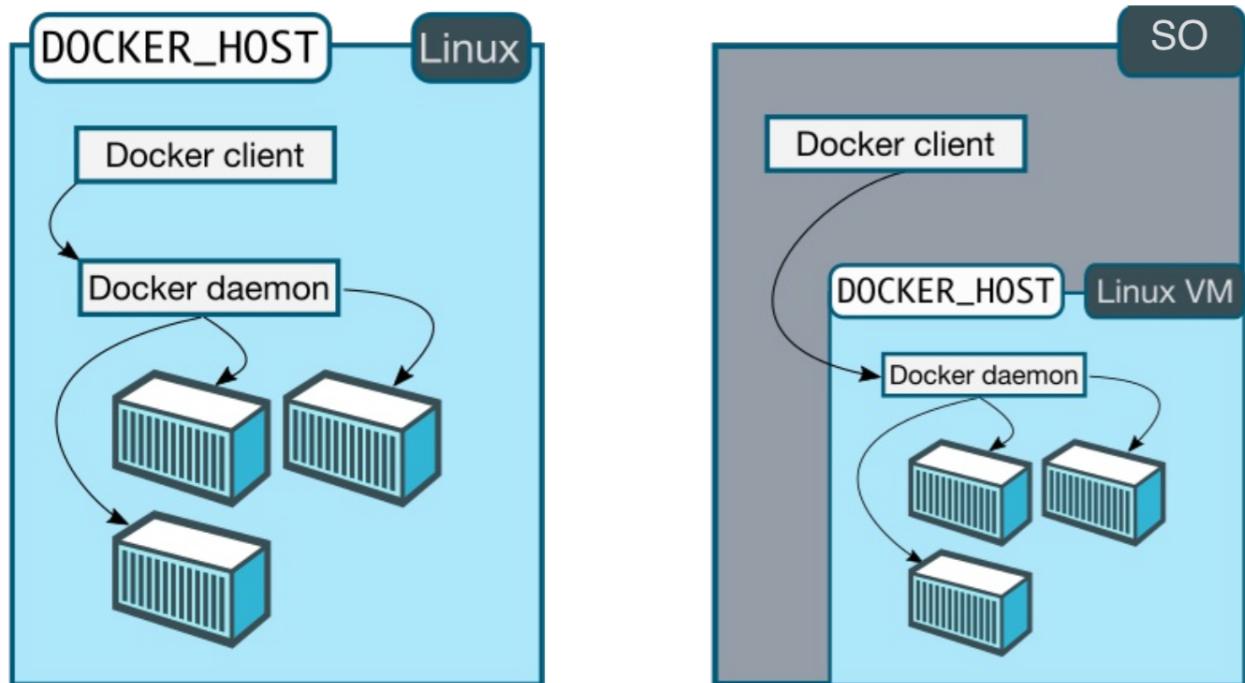
1. Artículo de Martin Fowler sobre microservicios:
[“http://www.martinfowler.com/articles/microservices.html”](http://www.martinfowler.com/articles/microservices.html)
2. Newman, Sam. (2015). “Building Microservices, designing fine-grained systems”. Sebastopol, CA: O'Reilly Media.
3. Artículo en “Computer weekly”, “Microservices, how to prepare next generation cloud applications”
[“http://www.computerweekly.com/feature/Microservices-How-to-prepare-next-generation-cloud-applications”](http://www.computerweekly.com/feature/Microservices-How-to-prepare-next-generation-cloud-applications)
4. Manifiesto SOA :
[“http://www.soa-manifesto.org/default_spanish.html”](http://www.soa-manifesto.org/default_spanish.html)
5. Martin L. Abbot, Michael T. Fisher. (2015). “The art of scalability”. Adisson-Wesley.
6. Artículo de Chris Richardson sobre microservicios:
[“http://microservices.io/articles/scalercube.html”](http://microservices.io/articles/scalercube.html)
7. Web de instalación Docker:
[“https://docs.docker.com/installation/”](https://docs.docker.com/installation/)
8. Paquete instalación boot2Docker para Mac OS:
[“https://github.com/boot2docker/osx-installer/releases/tag/v1.7.0”](https://github.com/boot2docker/osx-installer/releases/tag/v1.7.0)
9. Web Docker que describe el documento constructor (DockerFile):
[“https://docs.docker.com/reference/builder/”](https://docs.docker.com/reference/builder/)

ANEXO 1: Instalación y configuración de Docker

En este anexo se va a orientar al lector sobre como instalar y configurar Docker para el despliegue de microservicios. No obstante, esta guía de instalación es válida para la utilización de Docker para cualquier otro propósito. La instalación de Docker es bastante sencilla por lo que cualquier persona con mínimas nociones sobre utilización del terminal en cualquier sistema operativo y sobre el uso de máquinas virtuales no debería de tener ningún problema para la correcta realización de la misma.

Antes de comenzar a comentar a seguir los pasos necesarios se recomienda leer el punto 3.3.2 “Docker” de este documento donde se exponen las características de esta tecnología de virtualización ligera sobre contenedores. También es necesario conocer como “va a funcionar” Docker en nuestro sistema operativo una vez que lo tengamos instalado.

Docker opera sobre un sistema operativo Linux, es por esto que para otro tipo de sistemas operativos vamos a tener que instalar una máquina virtual (llamada “boot2docker”) que ya está perfectamente configurada y nos va a permitir el uso de Docker como si nuestro SO fuera Linux salvo pequeñas discrepancias que comentaremos posteriormente. En la siguiente imagen se puede observar claramente este concepto:



- Fuente de la imagen: <https://docs.docker.com/installation/> -

Se puede observar que sobre sistemas operativos diferentes a Linux se trabaja sobre una máquina virtual, por lo que va a ser necesario tener un gestor de máquinas virtuales (desde Docker se recomienda VirtualBox). No obstante a través de la web de instalación de boot2docker se comentará en la página posterior nos proporcionan una descarga directa a dicho aplicación.

Se va a detallar la instalación tanto para Linux Ubuntu como para Mac OS, ya que se cree que partiendo de la instalación en estos SOs se puede extender a muchos otros. La instalación en cualquiera de los casos es muy sencilla, de cualquier modo la información que aquí se presenta ha sido obtenida de la web de instalación de Docker [7], por lo que en caso de fallo en la instalación o que no quede del todo claro que pasos seguir para un sistema operativo que aquí no se contempla, el lector debe consultar dicha referencia.

Todas las instrucciones que aparecen en este anexo se deben dar desde el terminal del sistema operativo correspondiente.

Instalación de Docker sobre Linux Ubuntu

Antes de empezar se debe instalar el paquete que nos va a permitir que se virtualicen los contenedores Docker, este paso aparece junto al resto de pasos como paso 0:

0. Actualización del gestor de paquetes e instalación de los paquetes requeridos:

```
$ sudo apt-get update  
$ sudo apt-get install linux-image-generic-lts-trusty  
$ sudo reboot
```

1. Otorgar privilegios de super usuario a la sesión.

2. Instalar si no se ha instalado previamente wget:

```
$ sudo apt-get install wget
```

3. Descargar el último paquete de Docker:

```
$ wget -qO- https://get.docker.com/ | sh
```

4. Comprobación de que Docker se ha instalado correctamente:

```
$ docker run hello-world
```

Habiendo seguido estos pasos ya tendríamos que tener Docker instalado y listo para usar. Si todo ha salido correctamente, el último paso tendría que dar como resultado la siguiente captura del terminal:

```
Hello from Docker.  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(Assuming it was not already locally available.)  
3. The Docker daemon created a new container from that image which runs the  
executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
to your terminal.
```

Instalación de Docker sobre Mac OS

Como se ha comentado anteriormente, la instalación para sistemas operativos diferentes a Linux es un poco más compleja, en este caso se va a explicar como hacerlo para sistemas operativos MAC OS.

1. El primer paso es instalar boot2docker, para ello nos tenemos que descargar el paquete alojado en la plataforma Github de manera pública [8]. Este paquete va a contener tanto lo necesario para instalar boot2docker como para instalar VirtualBox en el caso de que no se hubiera instalado previamente.

2. Descargado nuestro paquete tipo “.pkg”, al hacer doble click en el mismo nos aparecerán las opciones de instalar boot2docker y VirtualBox como hemos hecho con el resto de programas instalados en nuestro equipo.

3. Para este punto que podría llamarse configuración existen dos posibilidades:

1. Ejecutar las siguientes instrucciones desde nuestro terminal:

```
$ boot2docker init  
$ boot2docker start  
$ boot2docker shellinit  
$ eval "$(boot2docker shellinit)"
```

Estas instrucciones inician la máquina virtual “boot2docker” y “nos dejan” en posición de utilizar docker de manera normal desde el terminal como si de un sistema operativo Linux se tratase.

2. A parte de la configuración “manual” utilizada en el punto anterior, es posible realizar esta configuración de una manera más automatizada. Si abrimos la carpeta de aplicaciones y ejecutamos boot2docker se va a abrir una ventana en el terminal sobre la que ya se “han realizado” las operaciones anteriores.

Una vez realizada la instalación y configuración de boot2docker, si ejecutamos el comando “\$ docker run hello-world” debemos observar de nuevo la captura de pantalla del terminal que aparece en la página anterior.

Con estos sencillos pasos ya tendríamos instalado Docker en nuestro sistema operativo y estaría listo para ser usado. Para finalizar la configuración debemos registrarnos en la web de Docker y por último establecer la sesión a través del terminal. Para ello, una vez registrado, a través del terminal haremos login utilizando el comando “docker login” utilizando nuestro usuario y contraseña.

En el siguiente anexo se va a describir como desplegar una aplicación Node JS (poniendo como ejemplo la que se ha desarrollado). No obstante se recomienda, si el lector está interesado en el uso de la tecnología Docker que se lleve a la lectura de la guía de iniciación descrito en la página oficial de Docker.

ANEXO 2: Despliegue local del ejemplo de app web

De nuevo antes de enumerar los pasos necesarios para el despliegue de la aplicación, se va a comentar los recursos que proporciona Docker. La configuración de cada contenedor Docker dependerá de manera directa de la aplicación que se va a ejecutar dentro del contenedor a configurar, es por esto que la “receta” que se va a exponer en este anexo sólo va a servir para desplegar la app desarrollada, aunque se pueden aprovechar partes para el despliegue de otras aplicaciones.

La tecnología Docker tiene tres partes fundamentales que intervienen en el despliegue de aplicaciones: contenedores, imágenes y repositorios.

Podemos asimilar el concepto de contenedores Docker al de un directorio de nuestro ordenador que contiene todo lo necesario (librerías, código, herramientas del sistema...) para que una aplicación pueda ser ejecutada en cualquier sistema que soporte la tecnología Docker.

Por otro lado, llamamos imágenes en Docker a la base para crear estos contenedores. Puede considerarse el “sistema operativo” de nuestros contenedores, cada imagen va a tener instalada por defecto ciertas dependencias, programas o herramientas para el despliegue de la aplicación. Ofrece un gran punto fuerte, en nuestro equipo si se crea una nueva imagen partiendo de una imagen base, solo se van a guardar los cambios que incluye la nueva base, ahorrando un gran espacio en nuestro sistema.

La tercera parte, denominada repositorios o registros, consiste en una lista pública de imágenes Docker. Múltiples son las empresas que añaden a este repositorio su propia imagen “personalizada” con sus herramientas previamente.

La forma de “Dockerizar” una aplicación es partiendo de una imagen Docker. Dicha imagen se va a construir a través de un archivo de texto denominado DockerFile. En este archivo se incluyen las instrucciones para que Docker construya de forma automática la imagen sobre la que queremos desplegar nuestro contenedor.

El contenido de este archivo va a variar en función de las necesidades de la aplicación pero siempre van a tener partes en común. En la primera línea de este archivo tras la utilización del comando “FROM” se va a seleccionar la imagen que se va a tomar como base para el despliegue, Docker busca de manera local dicha imagen, y si no encuentra ninguna coincidencia con el nombre proporcionado, realizada la búsqueda (y descarga en caso de búsqueda exitosa) de la imagen a través del repositorio público de Docker. A continuación se utilizan múltiples comandos (“RUN”, “ENV”, “AND”, “COPY” ...) con el fin de instalar sobre la imagen las dependencias requeridas por la app que no están instaladas por defecto en la imagen base, el significado y la semántica para el uso de estos comandos se puede ver a través de la página web de Docker destinada a tal efecto [9]. Por último se declaran dos características importantes, a través del comando “EXPOSE” se expone el puerto de la app desarrollada que se va a exponer al exterior del contenedor, y también tras el comando “CMD” se da la instrucción para ejecutar la aplicación.

En el siguiente apartado se va a detallar como descargar la aplicación de GitHub, en cada una de las carpetas destinadas a microservicios va a haber un archivo DockerFile, es recomendable estudiarlo para entender mejor este concepto de construcción a partir de dicho archivo.

.

Descarga de la aplicación

El primer paso para el despliegue del ejemplo una vez teniendo instalado y configurado Docker en nuestro ordenador es realizar la descarga de la aplicación. El código se encuentra de manera pública en Github, se va a poder acceder a él a través de este link:

<https://github.com/ManuPH/Microservices>

Hay dos maneras de descargarlo:

1. Acceder a través del navegador y hacer click en “Download zip” y descomprimirlo en nuestro ordenador.
2. Descargar la aplicación a través del terminal, para ello debemos tener instalados todos los requisitos de Github, y ejecutar el comando “`$ git clone git://github.com/schacon/grit.git mygrit`”

El archivo descargado, debe contener cuatro directorios, correspondientes a los cuatro microservicios de la aplicación, con nombres src, carrito, búsqueda y catalogo, además de un archivo Read.me. Se recomienda por cuestiones organizativas, guardar en un único directorio los cuatro ya descargados.

Despliegue de la aplicación en local

Una vez instalado y configurado Docker y descargados los archivos que componen la aplicación podemos desplegarla en local. Como hemos comentado en el cuerpo del TFG no se trata de una aplicación comercial sino que su función es que sea didáctica, por lo que hay ciertos cambios con respecto al despliegue si se realizara de manera abierta que se van a exponer a continuación.

Una de las diferencias entre el uso de Docker sobre Linux o del uso sobre boot2Docker es el acceso a los contenedores. Mientras que al usar Linux se puede acceder a través de localhost y el puerto en el que “escucha” el contenedor Docker, la utilización de una máquina virtual boot2docker, nos va a obligar a acceder a través de la IP que corresponde a dicha máquina.

Las peticiones que se realizan en la aplicación de ejemplo están dirigidas de manera predeterminada a puertos concretos de servidores Docker en ejecución. Esto va a provocar, que la aplicación inicialmente no funcione si se ha desplegado sobre un sistema Linux, ya que en un principio ha sido programada para su funcionamiento sobre Mac OS a través de boot2docker. Este problema se soluciona cambiando en todas las partes del código donde se configure una petición HTTP la dirección de boot2Docker (192.168.59.103 por defecto) por localhost.

Esta es la única modificación que se debe hacer sobre el código descargado para el correcto funcionamiento en local de la app, y solo debe hacerse en el caso de que se trabaje sobre un sistema operativo Linux.

En la siguiente página se va a detallar paso por paso que debemos hacer para el correcto despliegue de la aplicación web, es importante seguir los pasos en el orden indicado ya que en caso contrario la aplicación podría no funcionar como es debido.

Como se ha podido ver en el esquema que representa la arquitectura de la aplicación, se van a desplegar en total seis contenedores Docker, cuatro para los microservicios e interfaz y dos más destinados a alojar las bases de datos. Para ello construimos la imagen a partir de cada DockerFile y luego ejecutamos el contenedor a través de estos dos comandos:

- **docker build [options] [path];** Como opciones utilizamos -t para etiquetarlo, y como path “.” que indica que el path del DockerFile es el directorio en el que nos encontramos
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
- **docker run [options] IMAGE [COMMAND] [ARG];** Como opciones vamos a incluir el nombre del contenedor y el puerto que se va a exponer al exterior del contenedor, además en el caso del contenedor del carrito se va a conectar este contenedor con el que contiene la base de datos Redis la característica COMMAND no se incluye ya que en nuestro caso está en el DockerFile.

1. Contenedor MongoDB, desde el directorio global de la aplicación ejecutar:

```
$ docker run -d --name mongo -p 27017:27017 mongo
```

2. Contenedor Redis, desde el directorio global de la aplicación ejecutar:

```
$ docker run -d --name redis -p 6379:6379 redis
```

3. Contenedor con IU y Microservicio de gestión de peticiones, desde el directorio “src” ejecutar:

```
$ docker build -t <your username>/src .
```

```
$ docker run --name src -p 49160:8080 -d <your username>/src
```

4. Contenedor Microservicio de búsqueda, desde el directorio “busqueda” ejecutar:

```
$ docker build -t <your username>/busqueda .
```

```
$ docker run --name busqueda -p 50000:8080 -d <your username>/busqueda
```

5. Contenedor Microservicio de ordenación de catálogo, desde el directorio “catalogo” ejecutar:

```
$ docker build -t <your username>/catalogo .
```

```
$ docker run --name catalogo -p 60000:8080 -d <your username>/catalogo
```

6. Contenedor Microservicio de carrito de la compra, desde el directorio “carrito” ejecutar:

```
$ docker build -t <your username>/carrito .
```

```
$ docker run -d --name carrito node -p 32769:8080 --link redis:redis <your username>/carrito
```

Estos son los comandos necesarios para desplegar de manera local la aplicación. En la página anterior se ha hablado de algunos de los comandos que componen los DockerFiles en los que se recogen las instrucciones para descargar y configurar la imagen, a través de Docker también se pueden utilizar múltiples comandos mas allá de “build” y “run”, los cuales ya se han comentado. Estos comandos (que no se han visto) principalmente sirven para ayudar a la gestión tanto de los contenedores (parar, renombrar, conectar...) como de las imágenes (actualizar, eliminar ...).

Si todo ha funcionado correctamente, al ejecutar el comando “`docker ps`” que muestra información sobre los contenedores que están corriendo debería aparecer una respuesta similar a la que aparece en la siguiente imagen:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
439ac0c31e58	manuph/carrito:latest	"node /carrito/serve	9 days ago	Up 9 days	0.0.0.0:32769->8080/tcp	carrito
609b934413b4	redis:latest	"/entrypoint.sh redi	9 days ago	Up 9 days	0.0.0.0:6379->6379/tcp	redis
acbada48f784	manuph/busqueda:latest	"node /busqueda/serv	9 days ago	Up 9 days	0.0.0.0:50000->8080/tcp	busqueda
9736f66a2db7	manuph/catalogo:latest	"node /catalogo/serv	9 days ago	Up 9 days	0.0.0.0:60000->8080/tcp	catalogo
b99e646941f7	mongo:latest	"/entrypoint.sh mong	9 days ago	Up 9 days	0.0.0.0:27017->27017/tcp	mongo
8d355aa38b5c	manuph/src:latest	"/src/bin/www"	9 days ago	Up 9 days	0.0.0.0:49160->8080/tcp	src

En la imagen se observa la información que se aporta a través de “`docker ps`”. En concreto, se ofrecen datos ordenados en siete columnas sobre los contenedores que están corriendo (activos) en el momento en el que se utiliza. Las siete columnas tienen el siguiente significado:

- Container ID: código identificador del contenedor, se puede utilizar en las instrucciones del mismo modo que el nombre para ejecutar sentencias sobre contenedores.
- Image: nombre de la imagen sobre la que se ha montado el servidor.
- Command: se trata del último comando del archivo DockerFile, es el comando necesario para que se ejecute la el código del microservicio, o se inicie una base datos en el contenedor correspondiente.
- Created: el tiempo que ha pasado desde la fecha en la que se creó el contenedor.
- Status: el estado actual en el que se encuentra el servidor (Up, Down).
- Ports: a través de esta columna se muestra una información realmente importante. Se trata del “mapeo” de puertos entre el puerto que se expone de la aplicación y el que se va a exponer para poder acceder al contenedor. Como ya se ha comentado, para que la aplicación funcione correctamente, es necesario que el mapeo de puertos se realice de la misma manera que aparece en la imagen, ya que las peticiones http están dirigidas a ellos.
- Names: es el nombre que le hemos dado al contenedor, facilita la gestión de contenedores ya que es más fácil dar instrucciones a través del nombre que del ID. Se puede dar este nombre al contenedor tanto al crear el contenedor como después.

La aplicación puede verse en la dirección <http://192.168.59.103:49160/>, (la dirección IP de boot2docker y el puerto en el que escucha el contenedor donde esta la interfaz de usuario) en el caso de haber utilizado boot2docker para su despliegue, si la aplicación se hubiera desplegado sobre un sistema Linux de manera directa, se podría acceder a la interfaz a través de la dirección <http://localhost:49160/>.

La aplicación ya debería funcionar normalmente, no obstante la parte del catálogo aparecería vacía, la manera de añadir productos a la base de datos es realizando una petición http a cierto módulo del microservicio de ordenación de catálogo, la configuración de dicha petición se indica en el siguiente apartado.

Añadir productos a la base de datos de MongoDB

Aunque la aplicación ya esté desplegada, el catálogo aparecerá vacío ya que en un principio no hay datos en la base de datos. No forma parte de estudio de este TFG el tratamiento de datos por lo que se va a explicar de forma sencilla como incluir datos, simplemente para que se puedan ejecutar todas las funcionalidades de la aplicación web.

Para incluir productos hay que realizar una petición HTTP de tipo POST, para ello vamos a utilizar la aplicación para google chrome “Postman”, hay que realizar la petición POST sobre la dirección <http://192.168.59.103:60000/tshirt> (Hay una funcionalidad en el microservicio de catalogo programada para añadir productos) con los parámetros que aparecen en la imagen:

The screenshot shows the Postman interface with the following details:

- Request URL: `http://192.168.59.103:60000/tshirt`
- Method: `POST`
- Headers: `URL params (0)`, `Headers (0)`
- Body Type: `form-data` (selected)
- Fields:
 - `model`: `Tirantes`
 - `price`: `48`
 - `style`: `Vintage`
 - `size`: `38`
 - `colour`: `blue`
 - `summary`: `Spring collection`
 - `Key`: `Value`
- Buttons: `Send` (highlighted in blue), `Preview`, `Add to collection`, `Reset`

La función está preparada para recibir esos parámetros, se puede variar el precio, la talla (36, 38, 40, 42) o el modelo, este último solo admitirá tres tipos, Tirantes, Rayas o Polo.

Una vez añadidos los productos que queramos, la aplicación ya estará completa y todas las funcionalidades de la aplicación disponibles.