



Universidad Internacional de La Rioja  
Escuela Superior de Ingeniería y Tecnología

Adaptación al Grado en Ingeniería Informática

# Sistema de Información Contable de la Seguridad Social. Del monolítico a los microservicios

Trabajo fin de estudio presentado por:	Alfredo Soto Gómez
Github:	<a href="https://github.com/kokart/2021_TFGUNIR_mono_to_micro">https://github.com/kokart/2021_TFGUNIR_mono_to_micro</a>
Director/a:	Francisco José Soltero
Fecha:	15/07/2021

## Resumen

En este trabajo se realiza una propuesta para la migración de una serie de aplicaciones monolíticas hechas en Java, las cuales utilizan en su día a día los funcionarios del Sistema de Información Contable de la Seguridad Social, a una arquitectura de microservicios.

Se ha GitHub como repositorio centralizado para el control de las versiones del código fuente y de toda la documentación de este trabajo.

La migración de las aplicaciones actuales a microservicios permitirá el uso distribuido de las aplicaciones sin necesidad de tener el programa instalado en cada ordenador, tener una gestión de usuarios descentralizada permitiendo el uso de las aplicaciones a diferentes grupos de usuarios, además de disponer de un entorno más robusto evitando un único punto de fallo del sistema.

**Palabras clave:** monolítico, microservicios, docker, CQRS, java

## Abstract

In this work, a proposal is made for the migration of a series of monolithic applications made in Java, which used in their day to the officials of the Social Security Accounting Information System, a microservices architecture.

GitHub has been used as a centralized repository for controlling the versions of the source code and all the documentation for this work.

The migration of current applications to microservices allowed the distributed use of the applications without having to have the program installed on each computer, to have a decentralized user management allowing the use of the applications to different groups of users, in addition to having an environment more robust avoiding having a single point of failure of the system.

**Keywords:** monolithic, microservices, docker, CQRS, java

## Índice de contenidos

1.	Introducción .....	8
1.1.	Justificación del tema elegido.....	8
1.2.	Problema y finalidad del trabajo.....	8
2.	Estado del arte .....	9
2.1.	Del monolítico a los microservicios .....	9
2.1.1.	Arquitectura monolítica .....	9
2.1.2.	Arquitectura Orientada a Servicios (SOA) .....	10
2.1.3.	Arquitectura de microservicios .....	12
2.2.	Situación actual para la creación de microservicios .....	16
2.2.1.	Desarrollo del microservicio .....	16
2.2.2.	Comunicación entre microservicios .....	18
2.2.1.	Despliegue de los microservicios.....	20
3.	Objetivos del trabajo.....	23
3.1.	Objetivos principales.....	23
3.1.1.	Objetivos secundarios .....	23
4.	Diseño de la propuesta .....	24
4.1.	Contexto.....	24
4.1.1.	Situación inicial .....	24
4.1.2.	Aplicaciones monolíticas en la actualidad.....	26
4.2.	Contenidos .....	29
4.2.1.	Introducción .....	29
4.2.2.	Configuración bróker recepción / envío eventos entre microservicios .....	31
4.2.3.	Creación microservicio Gestión de usuarios .....	32
4.2.4.	Creación microservicio Editrans .....	34

4.2.5.	Creación microservicio IfiWeb Mutuas .....	35
4.3.	Metodología.....	37
4.3.1.	SCRUM .....	37
4.3.2.	Sprints Realizados.....	37
4.4.	Evaluación .....	39
4.4.1.	Introducción .....	39
4.4.2.	Resultados encuestas de satisfacción previa a la migración.....	39
4.4.3.	Resultados encuestas de satisfacción con aplicaciones en microservicios.....	40
5.	Conclusiones y trabajo futuro .....	41
5.1.	Conclusiones finales.....	41
5.2.	Mejoras detectadas a futuro .....	42
	Referencias bibliográficas.....	43
Anexo A.	Encuesta Monolítico a Microservicios .....	48
Anexo B.	Encuesta Uso de los Microservicios .....	50
	Índice de acrónimos .....	51

## Índice de figuras

Figura 1. Arquitectura SOA. ....	12
Figura 2. Monolítico vs microservicios. ....	13
Figura 3. Escalabilidad microservicios. ....	14
Figura 4. SOA vs Microservicios. ....	15
Figura 5. Plataforma Spring. ....	17
Figura 6. Máquina virtual vs contenedores. ....	20
Figura 7. Despliegue tradicional, virtualizado, dockerizado y con Kubernetes. ....	22
Figura 8. Organigrama del Centro de Desarrollo de Intervención. (Elaboración propia) .....	24
Figura 9. Usuarios de SICOSS. (Elaboración propia) .....	25
Figura 10. Aplicación Editrans. (Elaboración propia) .....	26
Figura 11. Aplicación Ifiweb. (Elaboración propia).....	27
Figura 12. Arquitectura caso de uso TFG. (Elaboración propia).....	30
Figura 13. Arquitectura con Kafka. ....	31
Figura 14. Fichero application.properties. (Elaboración propia) .....	33
Figura 15. Clase Kafka. (Elaboración propia).....	33
Figura 16. Aplicación Gestión de Usuarios. (Elaboración propia) .....	34
Figura 17. Aplicación Editrans. (Elaboración propia) .....	35
Figura 18. Aplicación Ifiweb. (Elaboración propia).....	36
Figura 19. Contenedores en repositorio Docker. (Elaboración propia) .....	36
Figura 20. Encuesta Monolítico a Microservicios, parte 1. (Elaboración propia) .....	48
Figura 21. Encuesta Monolítico a Microservicios, parte 2. (Elaboración propia) .....	49
Figura 22. Encuesta Monolítico a Microservicios, parte 2. (Elaboración propia) .....	50

## Índice de tablas

Tabla 1. Resumen ventajas e inconvenientes comunicaciones .....	19
---	----

## 1. Introducción

Este trabajo de Fin de Grado estudia la arquitectura de microservicios y la posibilidad de migrar una serie de aplicaciones monolíticas realizadas en Java que se usan en la actualidad en el departamento de contabilidad de la Gerencia de Informática de la Seguridad Social, en adelante GISS.

### 1.1. Justificación del tema elegido

Se ha elegido este tema para poder aprender e intentar aplicar el nuevo patrón arquitectónico de microservicios, así como poder estudiar las bases de datos NOSQL para proponer su uso en el servicio de gestión de los usuarios de las aplicaciones que vamos a implementar desde cero o en alguno de los servicios usados actualmente. Otro motivo es para poder usar contenedores y conocer en profundidad la tecnología Docker y el software Spring Boot para la creación de aplicaciones Java de manera rápida y sencilla.

Todas éstas son tecnologías modernas que se pueden utilizar en cualquier entorno tecnológico y que todo ingeniero debe de conocer.

### 1.2. Problema y finalidad del trabajo

Se disponen de varias aplicaciones monolíticas realizadas en Java sin seguir ningún patrón de diseño, sin gestión de usuarios y sin usar ningún gestor de proyectos como Maven o Gradle. Esto conlleva los problemas típicos de consumo de recursos de la máquina que ejecuta el programa, de la dependencia de la versión de java instalado en ella, del despliegue de nuevas versiones y que los usuarios puedan tener versiones obsoletas en su ordenador, etc.

La finalidad es preparar una arquitectura de microservicios y realizar una prueba de concepto migrando dos aplicaciones y añadiendo una gestión de usuarios para presentarla al departamento de Producción y Sistemas de la Gerencia Informática de la Seguridad Social, para poder desplegarla en producción acorde a los requisitos del centro y poder extender su uso a nuestros usuarios finales.



## 2. Estado del arte

### 2.1. Del monolítico a los microservicios

#### 2.1.1. Arquitectura monolítica

Hasta hace unos años, las aplicaciones grandes y complejas se implementaban como grandes monolitos muy difíciles de mantener y evolucionar.

Tal como se indica en (Newman, 2020) cuando todas las funciones de un sistema deben implementarse juntas, consideramos que es un monolito.

El ejemplo más común cuando se habla de ellos es un sistema en el que todo el código se implementa como un solo proceso.

Se pueden tener múltiples instancias de este proceso por razones de robustez o escala, pero fundamentalmente todo el código está empaquetado en un solo proceso.

Las características principales de esta arquitectura son:

- Son fáciles de desarrollar ya que todo se encuentra en el mismo sitio.
- Al estar todo junto, la puesta en producción no es complicada.
- Al no necesitar especialización, el coste de desarrollo es bajo comparado con otras arquitecturas.

Sin embargo, los principales problemas son:

- Alto acoplamiento y baja cohesión.
- Difícil de mantener.
- Difícil de escalar.

De los problemas anteriores nace la arquitectura orientada a servicios, conocida popularmente como SOA, que se explicará brevemente en el apartado siguiente y la cual es una parte fundamental en la arquitectura de microservicios. De hecho, no se entiende los microservicios sin los principios de la arquitectura SOA.

### 2.1.2. Arquitectura Orientada a Servicios (SOA)

Según define ASTIC(*Astic / Asociación Profesional de Cuerpos Superiores de Sistemas y Tecnologías de La Información de Las Administraciones Públicas.*, n.d.), “es un paradigma de arquitectura software que cuenta con la orientación a servicios como su principio fundamental de diseño”. El elemento destacable es la característica de “orientación a servicios”, que es la que conforma una arquitectura que utiliza servicios débilmente acoplados para cubrir los requisitos de procesos de negocio y requisitos de usuario.

Éstas han permitido dotar de flexibilidad y conseguir un gran ahorro de costes en las cuantiosas inversiones realizadas en Tecnologías de la Información y las Comunicaciones.

Su virtud reside en que facilitan la reutilización de componentes software y procesos de negocio independientemente de la plataforma tecnológica en que estén desarrollados, que en muchos casos se trata de componentes ya existentes en el abanico de soluciones tecnológicas de las organizaciones.

Los siguientes principios guía definen las reglas básicas para el desarrollo, mantenimiento y uso de las arquitecturas SOA:

- Reutilización.
- Interoperabilidad.
- Granularidad.
- Modularidad.
- Composición.
- Componentización.

Los siguientes principios arquitectónicos de diseño se derivan de la definición y diseño de servicios, y son intrínsecamente aplicables a las arquitecturas SOA por su carácter de “orientación a servicios”:

- Encapsulación: los servicios ocultan el código y la forma en la que implementan las funcionalidades que ofrecen hacia fuera, exponiendo una interfaz con dicha funcionalidad accesible.
- Débil acoplamiento: los servicios mantienen una relación entre sí que minimiza las dependencias entre ambos

- Contrato: los servicios se adscriben a un acuerdo de comunicaciones, definido colectivamente por uno o más documentos de descripción de servicio.
- Abstracción: los servicios ocultan la lógica al exterior.
- Composición: un conjunto de servicios puede ensamblarse y coordinarse para conformar servicios compuestos.
- Autonomía: los servicios tienen control sobre la lógica que encapsulan.

Los principales beneficios que ofrece son:

- Permite potenciar los activos preexistentes: Las arquitecturas SOA proporcionan un nivel de abstracción que permite a una organización potenciar la inversión realizada en Tecnologías de la Información y las Comunicaciones, ofreciendo sus activos como servicios que proporcionan funcionalidades de negocio. De esta forma, las organizaciones pueden potencialmente continuar obteniendo valor de los recursos existentes.
- Facilita la integración: El punto de integración en las arquitecturas SOA es la especificación del servicio, no su implementación. De esta forma, se proporciona transparencia de la implementación y de la tecnología subyacente, y se minimiza el impacto de los cambios. La integración se simplifica y la complejidad se gestiona mejor, aislando los posibles problemas de la cadena de valor.
- Mayor capacidad de respuesta: La habilidad de componer nuevos servicios a partir de servicios existentes proporciona una ventaja significativa que permite a una organización ser más ágil a la hora de responder a las demandas de las necesidades de negocio y de sus clientes.
- Reducir costes e incrementar la reutilización: Al disponer de los servicios básicos de negocio expuestos de forma que se promueve el débil acoplamiento, es posible la combinación y utilización de estos para cubrir las diferentes necesidades de negocio que van surgiendo, lo que supone a su vez menor duplicación de recursos, mayor potencial de reutilización y menor coste.

En la figura 1 se representa la misma versión de un software usando monolítico o SOA:

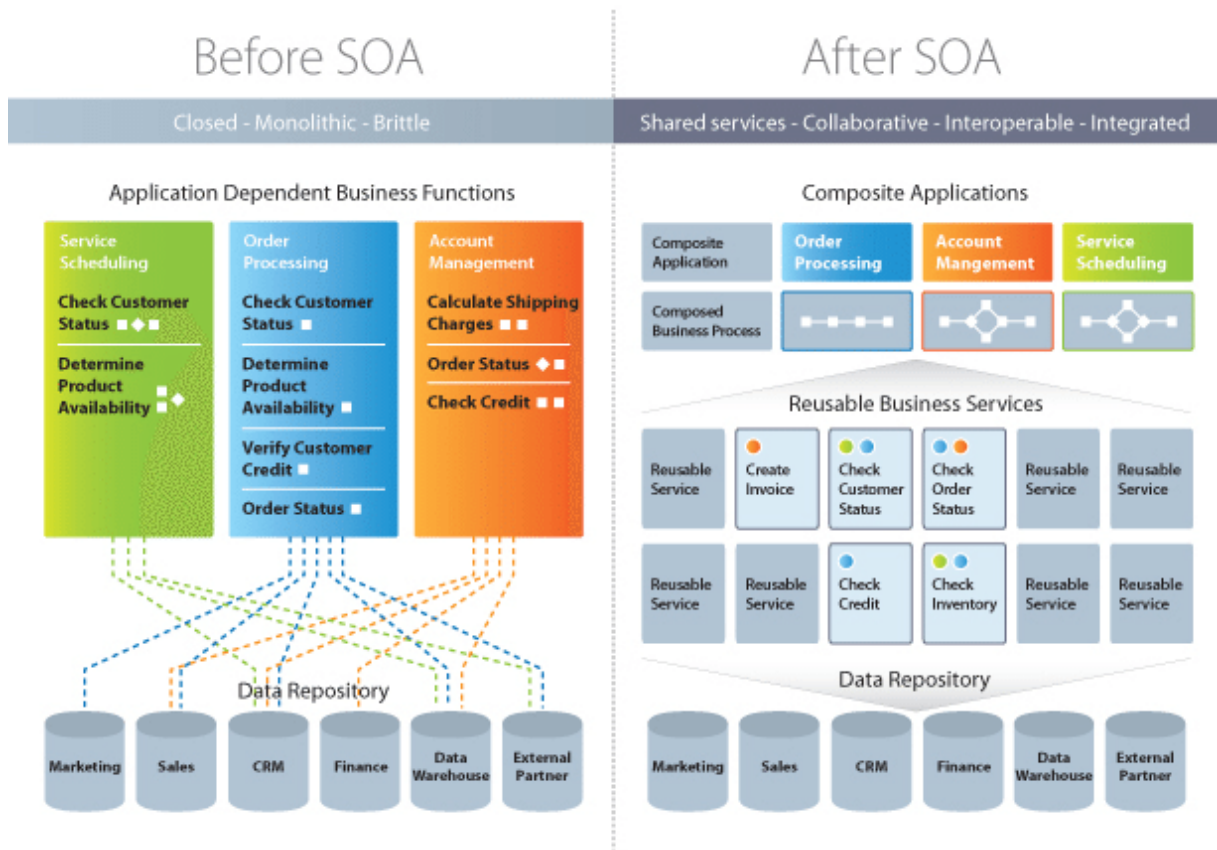


Figura 1. Arquitectura SOA.

Fuente: <https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec>

### 2.1.3. Arquitectura de microservicios

Un microservicio, según Martin Fowler (*Microservices*, n.d.) es una aplicación pequeña que ejecuta su propio proceso y se comunica mediante mecanismos ligeros (normalmente una API de recursos HTTP). Cada aplicación se encarga de implementar una funcionalidad completa del negocio, es desplegado de forma independiente y puede estar programado en distintos lenguajes, así como usar diferentes tecnologías de almacenamiento de datos.

La arquitectura de microservicios se basa en varios microservicios de colaboración. Son un tipo de arquitectura orientada a servicios (SOA). Éstos se comunican entre sí a través de diferentes redes, lo que las convierte en una forma de sistema distribuido. Ellos también encapsulan el almacenamiento y la recuperación de datos, exponiendo los datos a través de interfaces bien definidas. Esto facilita que las bases de datos estén ocultas dentro del límite del servicio.

Los servicios se caracterizan por:

- Poco acoplamiento.
- Mantenibilidad.
- Totalmente independiente del resto de microservicios.
- Cada uno implementa una parte del negocio

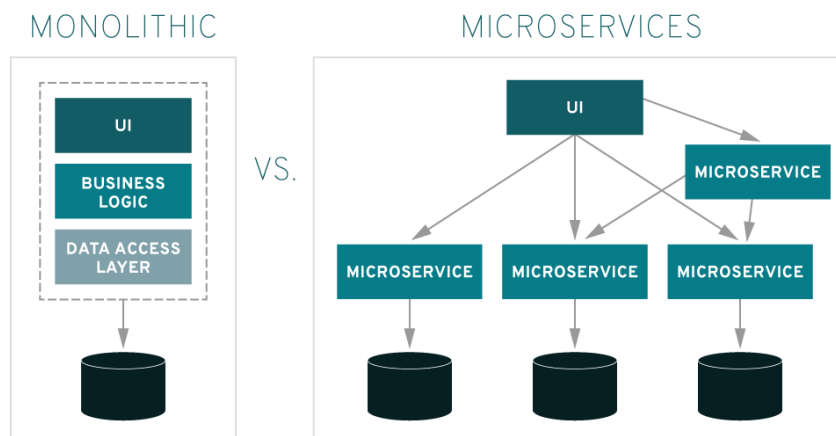
La decisión de si utilizar o no este tipo de diseño a la hora de construir el sistema se fundamenta básicamente en el nivel de complejidad que va a alcanzar.

Como ventajas se definen:

- Escalabilidad más eficiente e independiente.
- Pruebas más concretas y específicas.
- Posibilidad del uso de distintas tecnologías e implementaciones.
- Desarrollos independientes y paralelos.
- Aumento de la tolerancia a fallos.
- Mejora de la mantenibilidad.
- Permite el despliegue independiente.

### Microservicios vs Monolítico

La figura 2 muestra de manera gráfica la diferencia entre ambas arquitecturas:



*Figura 2. Monolítico vs microservicios.*

*Fuente: <https://www.redhat.com/es/topics/microservices/what-are-microservices>*

Las ventajas y desventajas de cada una han sido expuestas en los apartados anteriores, pero de esta manera quedan reflejados muy claramente de manera visual.

Además, la figura 3 muestra la relación entre ambas en uno de sus aspectos fundamentales, la escalabilidad.

Es un cubo de escalamiento aplicativo de tres dimensiones en la que:

- Dimensión X, escalando por clonación, duplicación horizontal.
- Dimensión Y, escalando por descomposición funcional, separando funcionalidades que son distintas.
- Dimensión Z, escalando por particionado de datos, separando por funcionalidades que son similares.

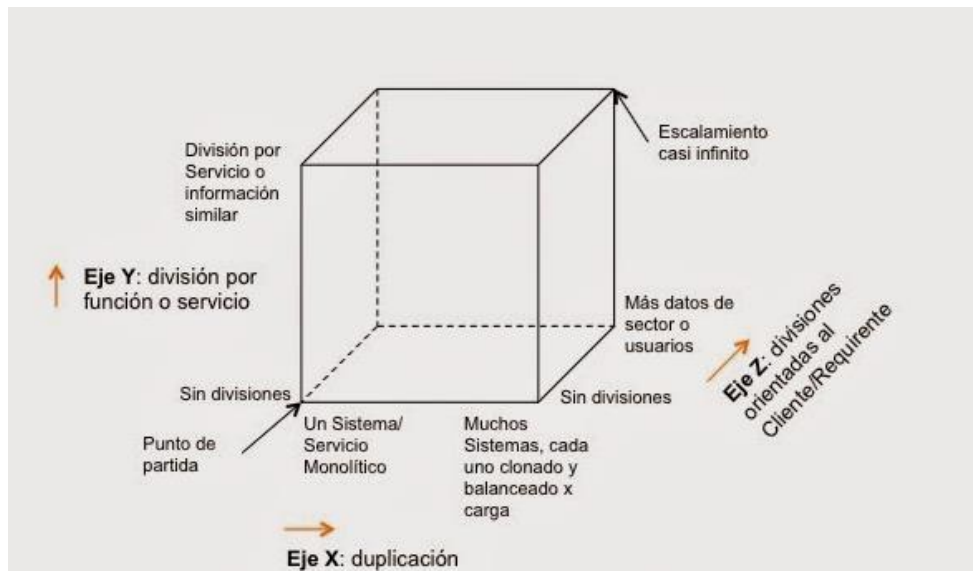


Figura 3. Escalabilidad microservicios.

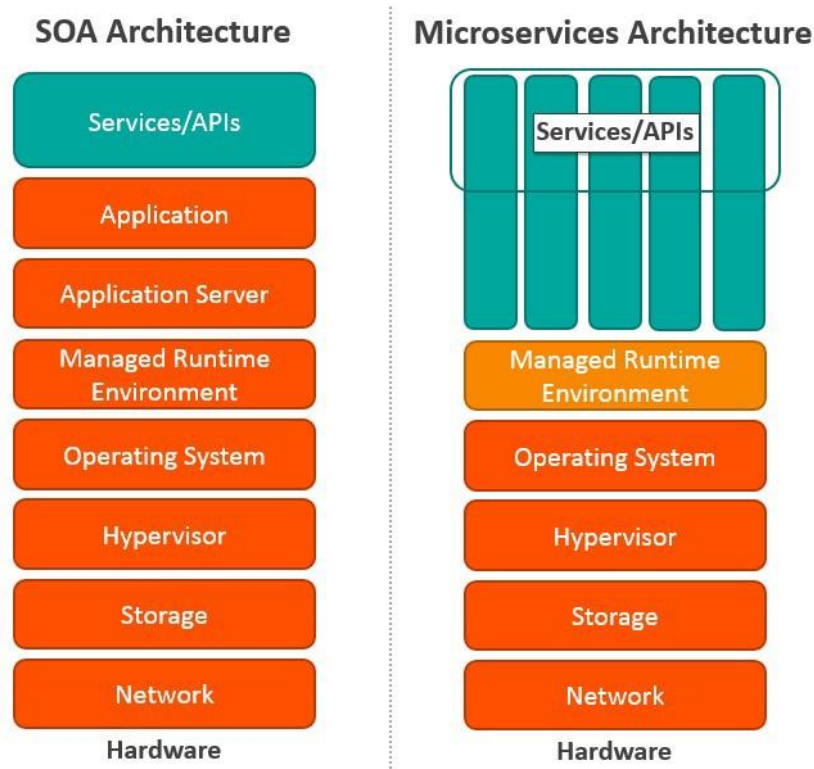
Fuente: <http://sergiomaurenzi.blogspot.com/2015/04/microservicios-parte-i.html>

El desarrollo en detalle se encuentra en (Mateus-Coelho et al., 2021)

### Microservicios vs SOA

La diferencia principal de las dos tecnologías es que a pesar de que ambas exponen servicios como canal de comunicación, en el caso de SOA estos servicios no son una unidad independiente de la solución, ni se pueden extraer del contexto en el que participan. En cambio, los microservicios son unidades totalmente independientes que proporcionan una solución a una pequeña parte del negocio y que son perfectamente exportables, además de tener su propio ciclo de vida.

La figura 4 se puede ver de manera gráfica la diferencia entre ambas arquitecturas:



*Figura 4. SOA vs Microservicios.*

*Fuente: <https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec>*

Visto las tres arquitecturas, lo que se debe tener claro a la hora de considerar un microservicio como tal es que éste 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, la idea de microservicios tiene su origen en el inicio del uso de la palabra SOA. De hecho, una primera lectura sobre el tema puede dar a entender que son lo mismo o prácticamente iguales. La realidad es que se entiende la arquitectura basada en microservicios como una evolución natural de SOA.

## 2.2. Situación actual para la creación de microservicios

### 2.2.1. Desarrollo del microservicio

Para el desarrollo de un microservicio se puede utilizar el lenguaje de programación que uno desee, ya que esta arquitectura no depende de en qué se realice, si no cómo se realiza. Para la realización del trabajo me he decantado por Java y por Spring Boot, que es un framework que pertenece a Spring Framework y sobre el cual ya tengo experiencia debido a mi trayectoria profesional. Las alternativas posibles a esta elección son varias, entre ellas Struts(*Welcome to the Apache Struts Project*, n.d.), Google Web Toolkit ([GWT], n.d.)y Apache Wicket(*Apache Wicket*, n.d.).

La comparativa entre ellas se sale fuera del ámbito de este trabajo, pero existen diferentes páginas donde las realizan (*Spring MVC vs Apache Wicket Detailed Comparison as of 2021 - Slant*, n.d.).

Spring Framework es un amplio conjunto de bibliotecas y herramientas que simplifican el desarrollo del software, la inyección de dependencias, el acceso a datos, su validación, internacionalización, etc. Es una de las opciones favoritas para proyectos Java(*10 Best Java Frameworks to Use in 2021 [Updated]*, n.d.) , y también funciona con otros lenguajes basados en Java Virtual Machine como Kotlin y Groovy.

En la figura 5 se muestra el ecosistema de la plataforma de Spring, con todo el conjunto de librerías disponibles.



## Spring IO Platform

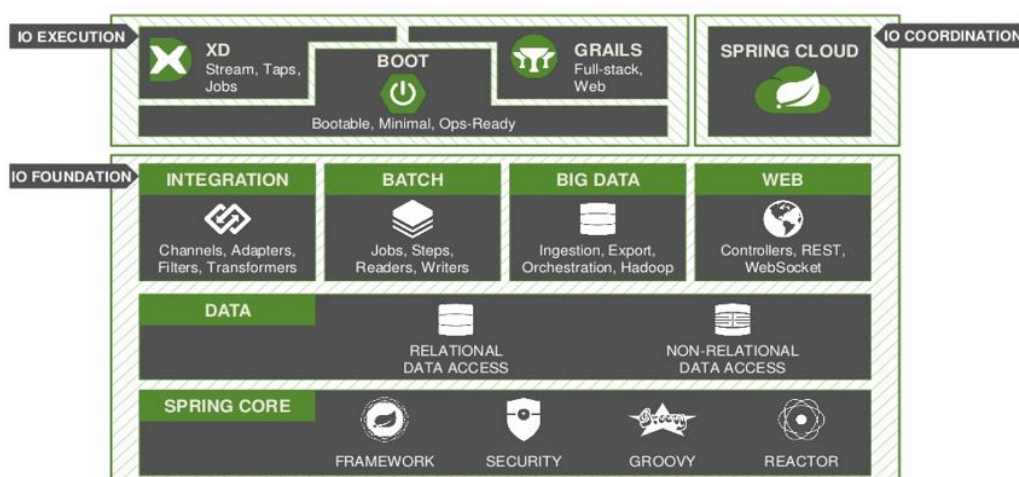


Figura 5. Plataforma Spring.

Fuente: <https://o7planning.org/>

Una de las razones por las que Spring es tan popular es que ahorra mucho tiempo al proporcionar implementaciones integradas para muchos aspectos del desarrollo de software, como, por ejemplo:

- Spring Data: Simplifica el acceso a bases de datos relacionales y NoSQL.
- Spring Batch: Proporciona funcionalidades para el procesamiento de muchos datos.
- Spring Security: Abstrae las funciones de seguridad de la aplicación, y facilita su uso como puede ser el uso de diferentes ROLES.
- Spring Cloud: Proporciona herramientas para crear rápidamente aplicaciones distribuidas según patrones estándar.
- Spring Integration: Simplifica la integración con otras aplicaciones y servicios, como por ejemplo con Kafka para el envío de eventos entre las aplicaciones.

La lista completa de módulos que facilita Spring se pueden encontrar en su página web (*Spring | Home*, n.d.)

Específicamente, Spring Boot es un framework que se utiliza para crear aplicaciones independientes en lenguajes basados en Java de manera rápida. Tal como se ha indicado anteriormente, es una de las herramientas más populares para construir no sólo microservicios, si no cualquier aplicación web siguiendo el patrón de diseño Modelo-Vista-Controlador.

Tiene las siguientes características(Karanam, 2018):

- Creación de aplicaciones independientes.
- Proporciona un servidor de aplicaciones embebido como puede ser Jetty o Tomcat, por lo que no se tiene la necesidad de deployar el WAR generado.
- Proporciona una interfaz gráfica para seleccionar las dependencias que vamos a necesitar, indicando si queremos utilizar Maven o Gradle.
- Configuración automática con diferentes librerías de 3eros, como Kafka o React.
- Evita la necesidad de conocer XML para la generación del código de configuración.
- Integración completa con contenedores como Docker y sistemas cloud como AWS o Azure.
- Thymeleaf(*Thymeleaf*, n.d.) o JSP integrados para realizar la capa de presentación de la aplicación,

#### 2.2.2. Comunicación entre microservicios

Para la comunicación entre los microservicios mediante el envío y recepción de los eventos se ha decidido el uso de un intermediario, que recibe el nombre de Broker. De manera resumida, el Broker recibe información de unas aplicaciones determinadas y esa misma la envía a otras aplicaciones. Por lo tanto, no hay comunicación directa entre aplicaciones y todo pasa por el servicio intermedio.

Existen dos formas diferentes de comunicar los eventos: Síncrona o Asíncrona.

Si se usa comunicaciones síncronas se necesita esperar por la respuesta de la otra parte que recibe el evento, mientras que en las comunicaciones asíncronas no se necesita esa respuesta para seguir enviando mensaje.

La siguiente tabla muestra las ventajas e inconvenientes de usar cada tipo:

**Tabla 1. Resumen ventajas e inconvenientes comunicaciones**

	Síncrono	Asíncrono
Ventajas	<ul style="list-style-type: none"><li>• Menor sobrecarga</li><li>• Rendimiento más alto</li><li>• Al necesitar confirmación, la comunicación es más rápida y en tiempo real</li></ul>	<ul style="list-style-type: none"><li>• Simple</li><li>• Menor coste</li><li>• Configuración rápida</li></ul>
Desventajas	<ul style="list-style-type: none"><li>• Más compleja</li><li>• Se necesitan conocimientos avanzados</li><li>• Más caro</li></ul>	<ul style="list-style-type: none"><li>• No es tiempo real</li><li>• Eficiencia menor</li><li>• Mayor riesgo de overhead</li></ul>

Hoy en día se pueden distinguir cuatro aplicaciones de Broker principales:

- Apache Kafka
- RabbitMQ
- ActiveMQ

En (*Kafka vs ActiveMQ vs RabbitMQ vs ZeroMQ - Digital Varys*, n.d.) se encuentra una comparativa extensa entre las cuatro soluciones y cuando elegir una u otra.

Con todo ello, para nuestro proyecto hemos elegido Kafka debido a:

- Integración completa con Spring Boot
- Fácil de utilizar y configurar
- Dispone de muchísima información
- Gran escalabilidad
- Posibilidad de usar arquitectura Kappa(*Kappa Architecture - Where Every Thing Is A Stream*, n.d.)

### 2.2.1. Despliegue de los microservicios

Hoy en día para el despliegue de contenedores lo más extendido es el uso de Docker por lo que se decide su uso (*Empowering App Development for Developers* / Docker, n.d.).

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones usando contenedores y proporcionando una capa adicional de abstracción y automatización a nivel de sistema operativo.

En pocas palabras, puedes construir y desplegar cualquier aplicación y en cualquier sitio sólo teniendo Docker instalado. Actualmente es considerado un estándar para resolver uno de los aspectos más importantes en el desarrollo del software: El despliegue.

Antes de la llegada de Docker, el despliegue requería de diferentes herramientas de distintas tecnologías, como por ejemplo máquinas virtuales, herramientas de gestión de la configuración, gestores de paquetes, etc. Se requerían de diferentes perfiles especialistas y en muchas ocasiones se producían errores por las versiones a utilizar, configuraciones erróneas, etc.

Con Docker esto ya no ocurre ya que lo que se hace es empaquetar todo lo necesario para ejecutar la aplicación en un contenedor Docker y abstraer todos los requisitos. Si se quiere ejecutar una aplicación Docker, simplemente se tiene que ejecutar su contenedor y ya estará configurada y lista para usarse, independientemente del lugar físico donde se ejecute.

La figura 6 trata de explicar la principal diferencia entre las máquinas virtuales y los contenedores Docker:

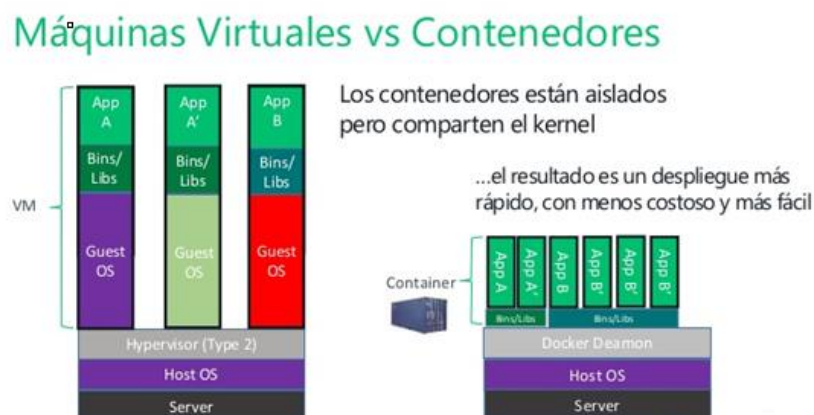


Figura 6. Máquina virtual vs contenedores.

Fuente: <https://www.slideshare.net/AdrianDiazCervera/dockeriza-tu-sql-server>

Docker es una herramienta que:

- Ahorra costes a las empresas ya que todo se centraliza en los contenedores.
- Puede reemplazar el uso de máquinas virtuales, Sólo hay que preocuparse de la aplicación y no del sistema operativo. Además, es mucho más rápido el uso de Docker que de máquinas virtuales y se puede configurar vía scripting.
- Proporciona un entorno de sandbox en milisegundos para realizar experimentos y pruebas de concepto de distintas soluciones software.
- No tiene dependencias para un usuario de Linux, por lo que es una buena manera de empaquetar el software. Puede construir la imagen y estar seguro de que funcionará en cualquier imagen moderna de Linux
- Permite descomponer una gran aplicación en pequeñas partes, facilitando la implantación de los microservicios. Cada microservicio en un contenedor.
- Evita conflictos y dependencias entre librerías, ficheros de configuración, XML, etc.

Visto lo que es capaz de hacer Docker y la utilidad de los contenedores, surge la necesidad de gestionarlos y automatizar su despliegue. No es lo mismo tener que manejar 3 microservicios que 300, por ello surge Kubernetes

Kubernetes(*¿Qué Es Kubernetes? | Kubernetes*, n.d.) es una herramienta open source que se utiliza para el uso de aplicaciones en contenedores, el despliegue automático y la gestión de la escalabilidad, pudiendo desplegar más contenedores en caso de ser necesario. Es indispensable para un entorno real de producción en el cual se usan esta tecnología.

Esta tecnología, entre otras cosas, ofrece:

- Alta disponibilidad y tolerancia a fallos.
- Balanceo de carga.
- Escalado automático para la aplicación.
- Comunicación entre los diferentes microservicios.

La figura 7 muestra la evolución de los sistemas de despliegue, desde el sistema tradicional hasta el despliegue de aplicaciones mediante Docker y Kubernetes:

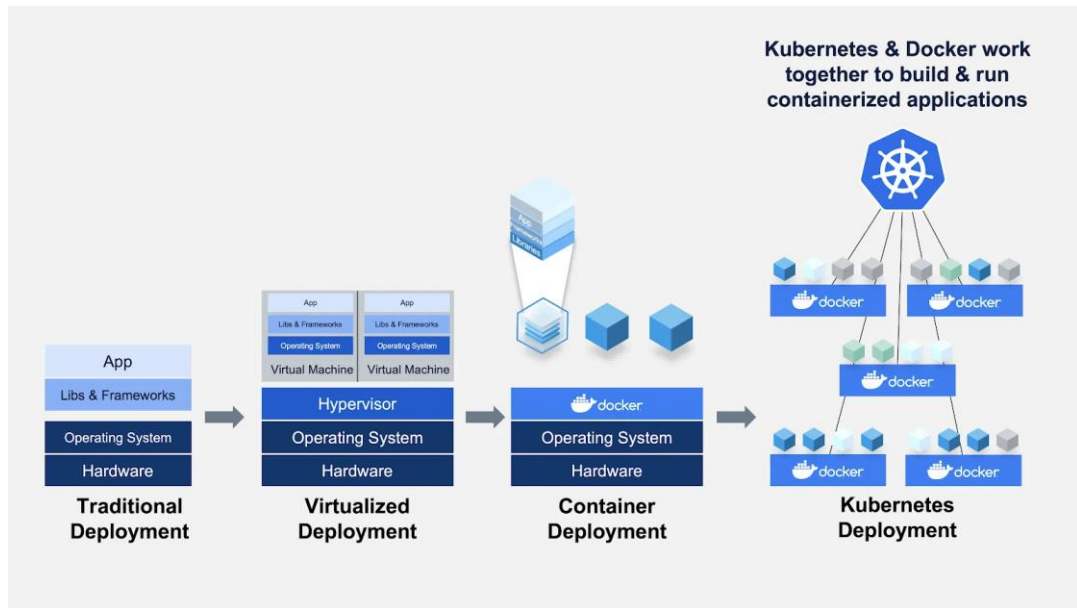


Figura 7. Despliegue tradicional, virtualizado, dockerizado y con Kubernetes.

Fuente: <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>

Para la realización del caso de uso y para futuros usos, se decide suscribirse al plan PRO que ofrece Docker HUB por 7\$/mes para tener acceso ilimitado a los contenedores y poder almacenar todos los que se desee de forma privada.

Más detalle de Docker y Kubernetes se puede encontrar en (Surovich & Boorshtein, 2020)

## 3. Objetivos del trabajo

### 3.1. Objetivos principales

El objetivo principal de este trabajo de fin de grado es estudiar la arquitectura de microservicios y realizar un caso de estudio en el que migremos algunas de las aplicaciones que actualmente se usan en el departamento de Sistema de Información Contable de la Seguridad Social, en adelante SICOSS.

Se quiere pasar de unas aplicaciones monolíticas desarrolladas en Java(*Java / Oracle*, n.d.) sin seguir ningún patrón ni ningún gestor de configuración de proyectos, y que sólo usan los funcionarios y algunos externos a unas aplicaciones basadas en la nueva arquitectura de microservicios, abriendo la posibilidad de ser usado por más usuarios.

#### 3.1.1. Objetivos secundarios

Aparte del objetivo principal, con este trabajo se pretende:

1. Conocer y usar software de mensajería para el envío y recepción de eventos mediante Apache Kafka o RabbitMQ.
2. Conocer y usar software para la creación de microservicios con herramientas como podría ser Spring Boot.
3. Conocer y usar Docker para la generación de los contenedores sobre los cuáles correrán las aplicaciones migradas.
4. Utilizar Maven como gestor de configuración de proyectos Java.
5. Permitir el uso de las aplicaciones actuales a los usuarios finales de SICOSS.

## 4. Diseño de la propuesta

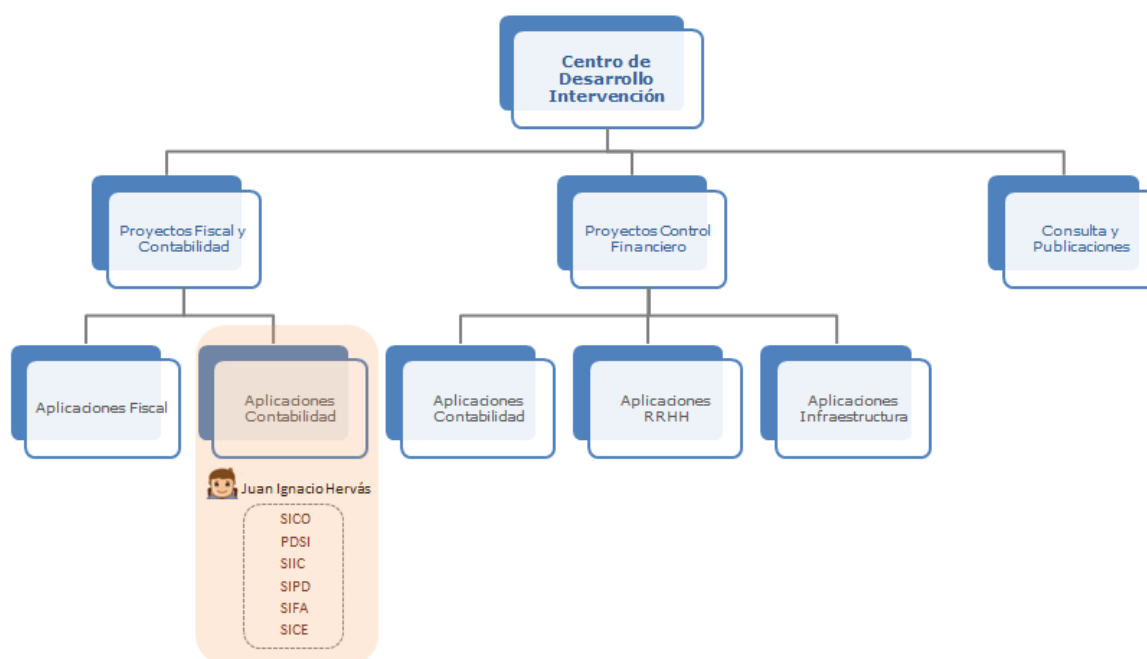
### 4.1. Contexto

#### 4.1.1. Situación inicial

La Gerencia Informática de la Seguridad Social, en adelante GISS, es el Centro Directivo responsable de proporcionar servicios TIC (Tecnologías de la Información y las Comunicaciones) a las Entidades Gestoras, Servicios Comunes y demás organismos adscritos a la Secretaría de Estado de la Seguridad Social y Pensiones.

La GISS se compone de diferentes centros de desarrollos, entre ellos el Centro de Desarrollo de Intervención. La aplicación SICOSS pertenece a las aplicaciones de Contabilidad, dentro de los Proyectos de Fiscal y Contabilidad, y está liderada por Juan Ignacio Hervás.

La figura 8 muestra el organigrama actual del centro:



*Figura 8. Organigrama del Centro de Desarrollo de Intervención. (Elaboración propia)*

Y lo que permite SICOSS es realizar la Contabilidad de cada uno de los Centros Gestores y Mutuas.

Un Centro Gestor es el responsable de un documento. Identifica a un centro de gasto en la Contabilidad. La Unidad de Gestión a la que está adscrito un usuario determina el Centro Gestor con el (o los) que puede trabajar.



Su codificación se desglosa en:

- Tipo de Organismo: 1
- Entidad: Los siguientes 3, dígitos, identifican a la entidad.
  - 001 INSS, 002 INGESA, 003 IMSERSO, 004 ISM, 005 TGSS, 006 GISS.
- Oficina: Los siguientes 4 dígitos. Los dos primeros identifican la provincia (60 para Servicios Centrales) y los dos siguientes son para identificar centros gestores de una misma provincia.

Y una Mutua es otro tipo de Centro Gestor. Son Entidades Colaboradoras de la Seguridad Social, en contraposición a las Entidades Gestoras.

Actualmente existen 21 Mutuas con un comportamiento específico en el Sistema:

Su codificación se desglosa en:

- Tipo de Organismo: 2
- Entidad: Los siguientes 3 dígitos, identifican a cada Mutua
- Oficina: Los siguientes 4 dígitos. En la actualidad todas las Mutuas tienen una única oficina, (6000)

Resumiendo, la figura 9 muestra la estructura organizativa de los usuarios de SICOSS:

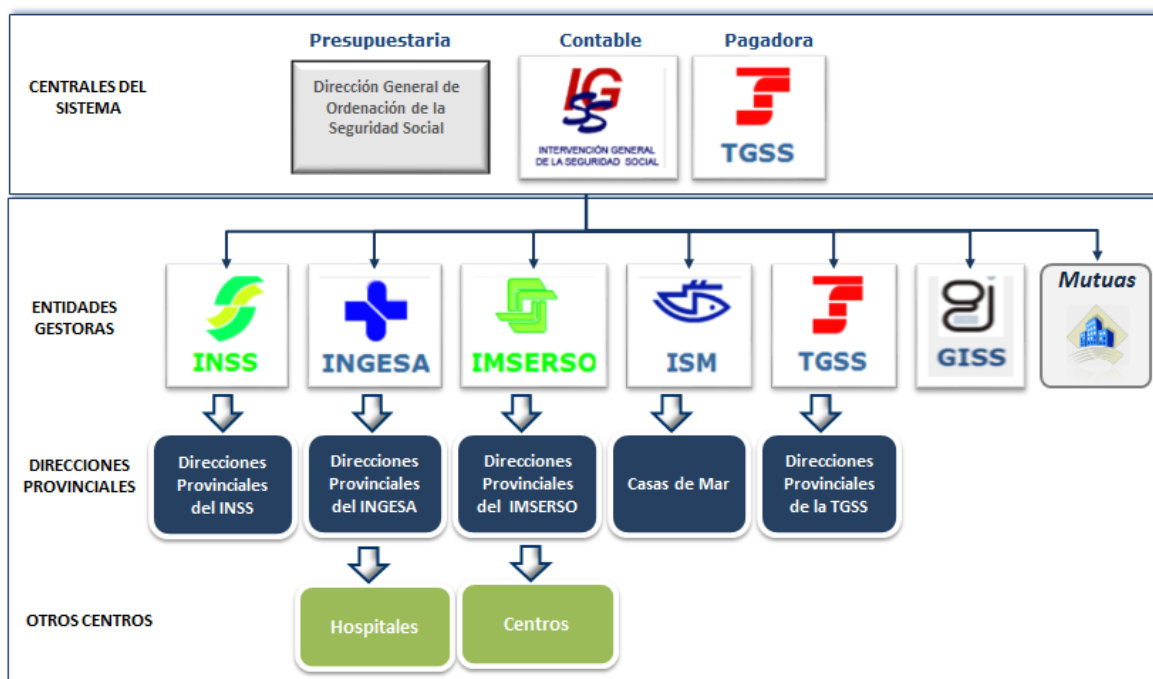


Figura 9. Usuarios de SICOSS. (Elaboración propia)

#### 4.1.2. Aplicaciones monolíticas en la actualidad

Las aplicaciones monolíticas que se usan en SICOSS en la actualidad son las siguientes:

1. Obtener Justificantes Editrans:

La aplicación es un .jar que se ejecuta y tiene la forma que se muestra en la figura 10:



Figura 10. Aplicación Editrans. (Elaboración propia)

Genera un archivo de texto en el escritorio con los números de justificantes indicados para la realización del trámite de Editrans ante la Agencia Tributaria. Para ello, va a la página que tiene la AEAT (*Modelo Para Presentación Telemática*, n.d.) y se queda con el número de justificante. Esto lo hace las veces que le indicamos como parámetro de entrada “Num.Justificantes”.

2. Buscar tabuladores en campo de texto de un documento contable:

La aplicación es un .jar que se ejecuta y lo que hace es buscar si alguna mutua ha introducido un tabulador en un campo de texto de un documento contable, ya que si lo ha hecho se provoca una excepción en un proceso nocturno que se ejecuta semanalmente.

Este programa se ejecuta diariamente y si encuentra, indica el código de las Mutuas que tienen tabuladores.

3. Obtener descripción clasificación orgánica:

Esta aplicación permite obtener la descripción del centro gestor o Mutua a partir del código indicado en el parámetro de entrada “ID a Buscar”. Se utiliza sobre todo para conocer quién es el organismo que se pone en contacto vía incidencias, ya que actualmente existen más de

200 entidades y no todos los funcionarios y externos tienen acceso a la base de datos para buscar la descripción y/o conocen la tabla en la que buscar la información.

Gracias a este programa obtienen la información de manera fácil, sencilla y directa.

#### 4. Generar fichero de salida IFIWeb:

La aplicación es un .jar que se ejecuta y tiene la forma que se muestra en la figura 11:



*Figura 11. Aplicación Ifiweb. (Elaboración propia)*

Se selecciona el fichero de entrada que ha proporcionado la mutua, y se cargan los datos para que el usuario los valide de manera manual. Una vez se comprueba que es correcto, se pulsa el botón Generar Fichero y se reconstruye el fichero de salida hasta el punto donde se ha interrumpido el proceso.

El fichero creado se genera en la misma ruta que se encuentra el fichero original de entrada. Esta información la utilizan para generar un nuevo fichero con sólo la información pendiente de procesar, ya que no se puede procesar dos veces lo mismo.

#### 5. Generar Fichero de documentos contables ADOK COVID:

El programa genera un archivo de texto en formato csv con una serie de campos de los documentos contables del tipo ADOK generados por todas las mutuas hasta una fecha concreta indicada por parámetro sobre una rúbrica concreta. En este caso, son rúbricas relacionadas con el COVID.

Se indica la rúbrica que queremos obtener y la fecha, y pulsamos al botón Obtener Fichero. Se genera un fichero llamado RUBRICA\_origen\_gasto\_ADOK\_Entidades.txt.

El detalle concreto de todas las aplicaciones y sus fotos se puede encontrar en:

[https://github.com/kokart/2021\\_TFGUNIR\\_mono\\_to\\_micro/blob/main/aplicaciones\\_monoliticas/TFG\\_AlfredoSotoGomez\\_fotos\\_aplicaciones\\_monoliticas.docx](https://github.com/kokart/2021_TFGUNIR_mono_to_micro/blob/main/aplicaciones_monoliticas/TFG_AlfredoSotoGomez_fotos_aplicaciones_monoliticas.docx)

## 4.2. Contenidos

### 4.2.1. Introducción

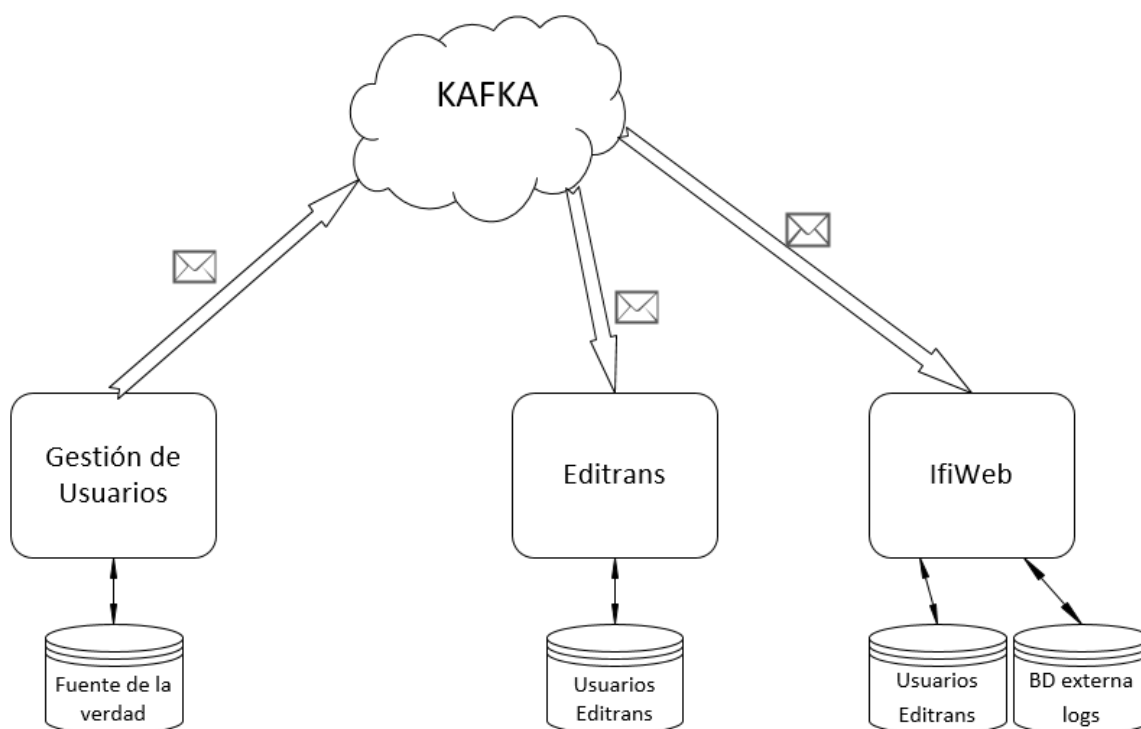
Una vez vistas las aplicaciones que se usan en la actualidad, se decide realizar la prueba de conceptos con las siguientes:

- Editrans: Se elige esta aplicación por 2 motivos:
  - Hay usuarios de asistencia técnica que no pueden utilizar esta aplicación debido a problemas con las versiones de Java. Usan programas que necesitan una versión específica de Java que hace que el aplicativo de Editrans, al ejecutarse, de error.
  - Se quiere que esta aplicación la puedan usar usuarios de la TGSS, ya que son los que presentan el trámite finalmente, por lo que, si ellos mismos pueden generarse los números de justificante, podríamos ahorrarnos el paso del proceso en el que interviene SICOSS y poder dedicar el tiempo a otra tarea.
- Fichero Salida IFIWeb Mutuas: Se elige esta aplicación por 2 motivos:
  - Para recuperar el fichero de entrada que necesita la aplicación para generar la salida, normalmente se necesita acceder al servidor de aplicaciones que realiza el proceso. Y ese acceso es exclusivo de los funcionarios y de algunos externos, pero no de todos.
  - Se quiere que esta aplicación la puedan usar usuarios de las Mutuas, ya que son ellos los que primero detectan que ha habido problemas y escriben solicitando ayuda y respuesta. El objetivo final es que sean autosuficientes y resolver las incidencias que detecten de manera inmediata.

Aparte de realizar la migración de estas dos aplicaciones, para conseguir el objetivo de que más usuarios puedan utilizar las aplicaciones vamos a crear un tercer microservicio que será de Gestión de usuarios, el cual se encargará de realizar las operaciones CRUD (Create-Retrieve-Update-Delete) necesarias.

La forma de comunicar toda la información que ha sido generada por este microservicio será a través del bróker de mensajería, el cual recibirá los eventos y los enviará a los microservicios adecuados.

La figura 12 muestra el esquema final al que se desea llegar:



*Figura 12. Arquitectura caso de uso TFG. (Elaboración propia)*

En los siguientes apartados se explica cómo se ha realizado la migración de cada una de las cuatro partes y que herramientas se ha decidido utilizar y la justificación en cada caso.

Por último, para poder evaluar la satisfacción de los usuarios con la propuesta realizada, se realizan unas encuestas al inicio del trabajo sobre las aplicaciones monolíticas y unas encuestas al final del trabajo, tras enseñar una demostración de los nuevos servicios.

Estos resultados sirven para decidir abordar o no la migración de los siguientes microservicios y presentar una propuesta al departamento de sistemas de la GISS para su implementación en un entorno real.

#### 4.2.2. Configuración bróker recepción / envío eventos entre microservicios

El bróker es el elemento esencial de nuestro trabajo ya que es el encargado de recibir los eventos que va a generar el servicio de gestión de usuarios y enviarlo a los distintos servicios que estén suscritos.

En nuestro caso de uso, vamos a tener un publicador que será el servicio de gestión de usuarios, y dos suscriptores, que serán las aplicaciones monolíticas que se migran a los microservicios, es decir, Editrans e Ifiweb.

Se van a crear dos topics, Editrans e Ifiweb, uno por cada aplicación. El publicador, cuando genere un evento comprobará qué aplicación lleva asociada y mandará la información por el topic correspondiente. Del otro lado, cada aplicación suscriptora va a estar configurada con el topic de su aplicación y sólo va a recibir los mensajes recibidos de su aplicación (ver figura 13).

La representación gráfica sigue a continuación:

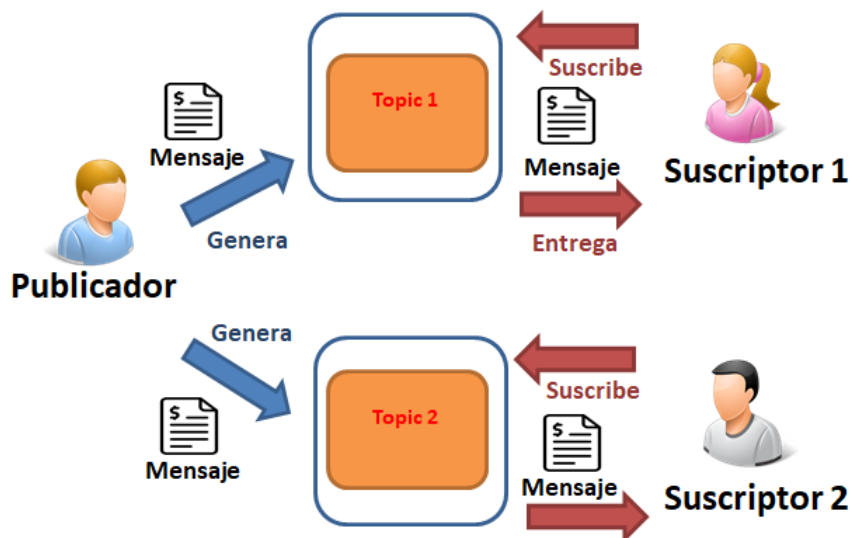


Figura 13. Arquitectura con Kafka.

Fuente: <https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-3-conceptos-basicos-extra/>

Para la puesta en marcha se han seguido los siguientes pasos:

- Descargar el software Apache Kafka (Apache Kafka, n.d.).
- Arrancar el Zookeeper (Apache ZooKeeper, n.d.):

```
%KAFKA_HOME%\bin\windows\zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
```

- Arrancar el bróker.

```
%KAFKA_HOME%\bin\windows\kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```

- Crear los dos topics para la demo.

```
%KAFKA_HOME%\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Editrans
```

```
%KAFKA_HOME%\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Ifiweb
```

Se puede consultar el libro de Kafka Streams in Action (Bejeck, 2018) para ver en detalle los conceptos a los que se hace referencia en este apartado, y las diferentes opciones de configuración de las que se disponen si se quiere desplegar en un entorno productivo, aprovechando al máximo sus capacidades.

#### 4.2.3. Creación microservicio Gestión de usuarios

Se crea una aplicación nueva para la Gestión de los usuarios (ver figura 16) que van a poder utilizar los diferentes microservicios que se utilizan durante la prueba de concepto. Es un servicio que lo que permite es lo que se conoce como CRUD de usuarios, es decir, la Creación, la Recuperación, la Modificación y el borrado de los usuarios (Create-Retrieve-Update-Delete).

Para su desarrollo se decide utilizar el framework Spring Boot y seguir el patrón Modelo Vista Controlador(MVC (*Model, View, Controller*) Explicado., n.d.), en el que tendremos la vista hecha con Thymeleaf(Michael Good, 2018) y las respectivas clases del Modelo y el Controlador en Java.

Para la gestión del proyecto se utiliza Maven y se utiliza Spring Initializr(*Spring Initializr*, n.d.) para seleccionar todas las dependencias del proyecto. En este caso, entre ella hay que destacar la librería *org.springframework.kafka* para la comunicación con el broker y *com.h2database* para la base de datos.

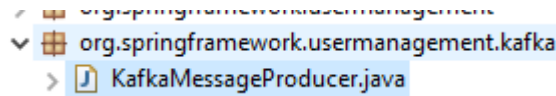
La integración con Kafka es tan sencilla como indicar en el *aplicación.properties* la dirección del servidor Kafka y cuáles serán los topics a los que se suscribe la aplicación. En este caso, tal como se ha indicado en el apartado anterior, hay un topic por cada aplicación (ver figura 14).



```
28 #Kafka
29 message.topic.name=Editrans
30 message.group.name=group
31 message.topic.name2=Ifiweb
32 spring.kafka.bootstrap-servers=localhost:9092
```

Figura 14. Fichero *application.properties*. (Elaboración propia)

Y una clase *KafkaMessageProducer.java* (ver figura 15) que es la que se encargará de mandar el mensaje al servidor cuando se produzca un evento relacionado con el usuario:



```
org.springframework.usermanagement.kafka
KafkaMessageProducer.java
```

Figura 15. Clase *Kafka*. (Elaboración propia)

Respecto a la persistencia de los datos, se utiliza H2(*H2 Database Engine*, n.d.) que tiene las siguientes características:

- Sigue el modelo relacional.
- Es muy rápida, de código libre y con la API JDBC integrada.
- La base de datos la tiene en memoria, por lo que tal como se indica en el punto anterior, es más rápida que sus competidoras.
- Consulta gráfica de la base de datos mediante navegador web

Se descarta el uso de base de datos NoSQL ya que para el módulo no se va a necesitar ninguna de las ventajas que ofrece.

Y tal como se ha visto en el apartado de soluciones tecnológicas, la aplicación se dockeriza siguiendo los pasos indicados en la página de Spring(*YMNNALFT: Easy Docker Image Creation with the Spring Boot Maven Plugin and Buildpacks*, n.d.) para poder desplegar en cualquier entorno sin depender de las dependencias de Java, Spring, etc.

En concreto, se ha ejecutado `mvnw spring-Boot:build-image -Dspring-Boot.build-image.imageName=Bootiful/demo` para crear la imagen inicial y se ha renombrado mediante `docker tag docker.io/Bootiful/demo:latest kokart/unir:Ifiweb`

Una vez está disponible el contenedor con el nombre deseado, se sube al repositorio de Docker mediante `docker push kokart/unir:usermanagementdoc` y se despliega mediante el comando

`docker run -p 8080:8080 kokart/unir:usermanagement`

La página de bienvenida del servicio tiene la siguiente forma:

Usuarios			
Nombre completo	Centro	Silcon	Aplicación
Alfredo Soto	GISS	99GU7889	Editrans
Guillem Torres	GISS	99GU6549	Ifiweb
Carmen Rada	AT	99GU2442	ADOK
Juan Cruz	FREMAP	99MU1554	Editrans
<button>Alta usuario</button>			

Figura 16. Aplicación Gestión de Usuarios. (Elaboración propia)

En el siguiente enlace se puede ver un video de su uso:

[https://github.com/kokart/2021\\_TFGUNIR\\_mono\\_to\\_micro/blob/main/aplicaciones\\_microservicios/Uso\\_servicio\\_gestionUsuarios.mkv](https://github.com/kokart/2021_TFGUNIR_mono_to_micro/blob/main/aplicaciones_microservicios/Uso_servicio_gestionUsuarios.mkv)

#### 4.2.4. Creación microservicio Editrans

Se migra la aplicación monolítica Editrans y para ello se sigue los pasos utilizados en la creación del servicio de Gestión de Usuarios (ver figura 17).

La principal diferencia es que la base de datos está vacía y los usuarios se irán creando según lleguen los eventos generados por el microservicio que se encarga de su gestión.

Para la integración con Kafka sólo se debe indicar en el *application.properties* la dirección del servidor Kafka y cuál es el topic al que se suscribe como consumidor, que en este caso será Editrans.

Por último, se crea un método llamado *listenTopic* que se encargará de procesar los mensajes recibidos.

Una vez el usuario esté dado de alta, en este caso de la TGSS, hará login y podrá descargarse el fichero con los justificantes necesarios. Tras mantener una reunión con los usuarios se ha decidido que sean siempre 400 ya que han especificado que trabajan en lotes de ese orden.

Se genera el contenedor siguiendo los pasos del anterior servicio y una vez está disponible, se sube al repositorio de Docker mediante `docker push kokart/unir:Editrans` y se despliega mediante el comando `docker run -p 8082:8082 kokart/unir:Editrans`

En este caso, el puerto es distinto ya que en el 8080 ya escucha el otro microservicio.

La página de bienvenida del servicio tiene la siguiente forma:



*Figura 17. Aplicación Editrans. (Elaboración propia)*

En el siguiente enlace se puede ver un video de su uso:

[https://github.com/kokart/2021\\_TFGUNIR\\_mono\\_to\\_micro/blob/main/aplicaciones\\_microservicios/Uso\\_servicio\\_Edittrans.mkv](https://github.com/kokart/2021_TFGUNIR_mono_to_micro/blob/main/aplicaciones_microservicios/Uso_servicio_Edittrans.mkv)

#### 4.2.5. Creación microservicio IfiWeb Mutuas

Se migra la aplicación monolítica Ifiweb y para ello se sigue los pasos utilizados en la creación del servicio de Editrans (ver figura 18).

Al igual que el servicio Editrans se partirá de una base de datos vacía y se irá añadiendo a los usuarios que se den de alta con el servicio de gestión de usuarios.

Para la integración con Kafka sólo se debe indicar en el *aplicación.properties* la dirección del servidor Kafka y cuál es el topic al que se suscribe como consumidor, que en este caso será Ifiweb y crear el método llamado *listenTopic* que se encargará de procesar los mensajes recibidos.

En este caso se dará acceso a los usuarios de Mutuas, harán login e indicarán el nombre del fichero de entrada sobre el cual quiere obtener la respuesta.

Se genera el contenedor siguiendo los pasos del anterior servicio y una vez está disponible, se sube al repositorio de Docker mediante `docker push kokart/unir:Ifiweb` y se despliega mediante el comando `docker run -p 8085:8085 kokart/unir:Ifiweb`

La página de bienvenida del servicio tiene la siguiente forma:



MINISTERIO DE INCLUSIÓN, SEGURIDAD SOCIAL Y MIGRACIONES

SECRETARÍA DE ESTADO DE LA SEGURIDAD SOCIAL Y PENSIONES

### Iniciar sesión en IFIWEB

Cierre de sesión satisfactorio

Usuario

Contraseña

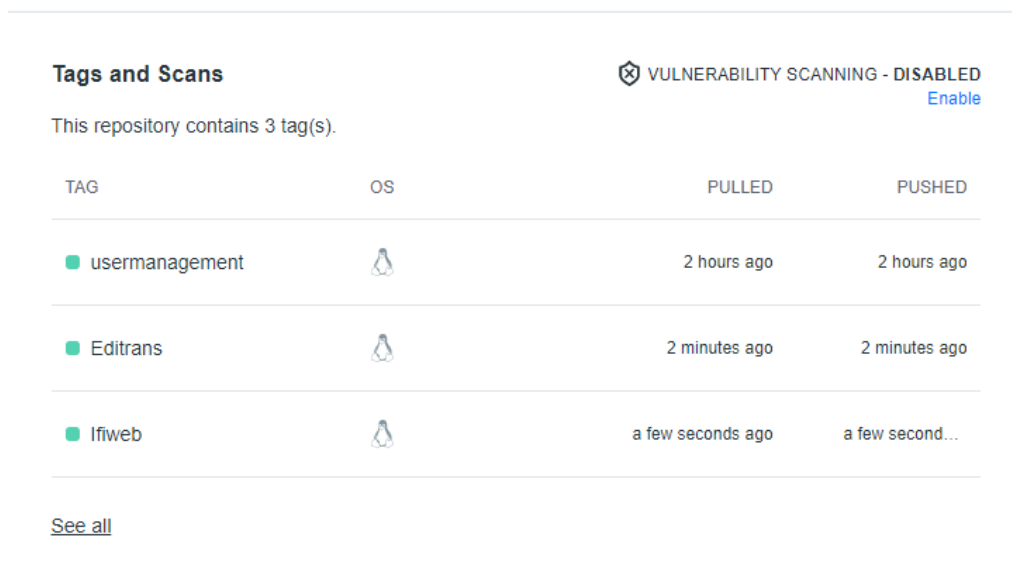
Entrar

*Figura 18. Aplicación Ifiweb. (Elaboración propia)*

En el siguiente enlace se puede ver un video de su uso:







[https://github.com/kokart/2021\\_TFGUNIR\\_mono\\_to\\_micro/blob/main/aplicaciones\\_microservicios/Uso\\_servicio\\_lfiweb.mkv](https://github.com/kokart/2021_TFGUNIR_mono_to_micro/blob/main/aplicaciones_microservicios/Uso_servicio_lfiweb.mkv)

La figura 19 muestra nuestro repositorio de Docker con los contenedores creados:



**Tags and Scans** VULNERABILITY SCANNING - DISABLED [Enable](#)

This repository contains 3 tag(s).

TAG	OS	PULLED	PUSHED
 usermanagement		2 hours ago	2 hours ago
 Editrans		2 minutes ago	2 minutes ago
 lfiweb		a few seconds ago	a few second...

[See all](#)

*Figura 19. Contenedores en repositorio Docker. (Elaboración propia)*

### 4.3. Metodología

Se opta por seguir la metodología ágil de gestión de proyectos SCRUM, ya que por las características que se detallan en el siguiente apartado hacen que sea la mejor opción, además de la experiencia acumulada en esta metodología durante los años de trabajo en diferentes empresas. El proyecto consta de un único desarrollador y se asume que el papel del Director del TFG será el del Product Owner.

#### 4.3.1. SCRUM

Scrum(*What Is Scrum?*, n.d.) es un marco de trabajo para el desarrollo y la gestión de software basado en un proceso iterativo e incremental para el desarrollo ágil de software.

Se destacan las siguientes características:

- Desarrollo incremental de los requisitos del proyecto.
- Se da prioridad a lo que tiene más valor para el cliente.
- El equipo se sincroniza diariamente y se realizan las adaptaciones necesarias.
- Al final cada sprint se muestra al cliente el resultado real obtenido.

Se trabaja por Sprints de entre dos y cuatro semanas en el cual se realiza el trabajo en sí. Al final de este se realiza una revisión y una retrospectiva con el propósito de realizar mejora continua del proceso.

#### 4.3.2. Sprints Realizados

Para la realización del trabajo se realizan Sprints de dos semanas, a excepción de un sprint final de 3 semanas, y se tiene en cuenta las siguientes fechas como hitos del proyecto:

- 25 de marzo de 2021: Entrega Borrador Inicial
- 29 de abril de 2021: Entrega Borrador Intermedio
- 10 de junio de 2021: Entrega Borrador final
- 8 de julio de 2021: Predepósito TFM
- 15 de julio de 2021: Depósito TM

Los Sprints han sido los siguientes:

- **Sprint #1: 11 de marzo a 25 de marzo:**

En este Sprint se acaba de definir el alcance del proyecto y la propuesta a realizar. Al principio se plantea la migración a una arquitectura de microservicios, pero se decide ir un paso más e investigar la arquitectura de microservicios. Se entrega el borrador inicial

- **Sprint #2: 31 de marzo a 14 de abril:**

En este Sprint se recolecta toda la información que se necesita para la realización del trabajo, además de trabajar en el estado del arte. Se crea la primera encuesta sobre las aplicaciones monolíticas y microservicios. Se sigue trabajando en la memoria.

- **Sprint #3: 15 de abril a 29 de abril:**

En este Sprint se sigue trabajando en el estado del arte y se empieza a trabajar con Sprint Boot y a ver el funcionamiento de la aplicación y a plantear la arquitectura definitiva. Se pone en funcionamiento Apache Kafka y se consigue el paso de mensajes entre diferentes topics/consumers/producers. Se entrega el borrador intermedio.

- **Sprint #4: 12 de mayo a 25 de mayo:**

En este Sprint se crea el servicio de Gestión de usuarios y se integra con Apache Kafka. Se sigue trabajando en la memoria.

- **Sprint #5: 26 de mayo a 10 de junio:**

En este Sprint se migran los servicios Editrans y IfiWeb y se crea la segunda encuesta sobre su uso por parte de nuestros usuarios finales. Se entrega el borrador final.

- **Sprint #6: 11 de junio a 25 de junio**

En este Sprint se trabaja sobre la memoria para la finalización del trabajo.

- **Sprint #7: 28 de junio a 15 de julio:**

En este Sprint se realizan los últimos cambios en la memoria antes de realizar el depósito final el 8 y 15 de Julio.

## 4.4. Evaluación

### 4.4.1. Introducción

Se utiliza la herramienta Google Forms para la creación de la encuesta de satisfacción de las aplicaciones monolíticas y de la encuesta de satisfacción una vez realizado el caso de uso de migración a microservicios.

La primera encuesta (ver figura 20 y 21) previa a la realización del estudio se puede encontrar en:

[https://docs.google.com/forms/d/e/1FAIpQLSeKF0EBclLyMAGh44RsBQrXMss4YAH3EggTObL\\_3P-OelUWzQ/viewform](https://docs.google.com/forms/d/e/1FAIpQLSeKF0EBclLyMAGh44RsBQrXMss4YAH3EggTObL_3P-OelUWzQ/viewform)

Mientras que la segunda encuesta (ver figura 22), una vez hecha la prueba de concepto, se puede encontrar en:

<https://docs.google.com/forms/d/e/1FAIpQLSc7mNWghTmxq3kaCuzXnG0vdjp56rilYpG1lpHYoZ25ml3SVw/viewform>

### 4.4.2. Resultados encuestas de satisfacción previa a la migración

La encuesta se ha facilitado a los usuarios actuales de las aplicaciones monolíticas, entre los que se encuentran 3 funcionarios y 14 externos. Todos son informáticos, con perfiles formativos diversos, desde Formación Profesional a Grado en Informática.

De 17 usuarios han contestado 10, y el detalle de las respuestas se puede ver en el Anexo A.

Se destaca lo siguiente:

- Ha participado un 59% de los usuarios, con un 100% si hablamos de funcionarios.
- Sólo el 20% prefiere aplicaciones de Escritorio monolíticas y el 40% no está contento con tener una aplicación por cada servicio.
- Nadie ha contestado que no le gustaría tener todas las aplicaciones en un mismo sitio WEB.
- Algunas de las ventajas que se indican en el uso de aplicaciones monolíticas: Permitir conexión offline, son más sencillas de implantar y son más rápidas.
- Algunas las desventajas que se indican en el uso de aplicaciones monolíticas: Las actualizaciones pueden tardar en llegar al usuario, hay que actualizar equipo a

equipo si se produce algún cambio debido a error e incompatibilidad con versiones de Java.

- Algunas ventajas que se indican en el uso de aplicaciones web: Se puede usar desde cualquier pc, uso centralizado y no requiere de instalación.
- Algunas desventajas que se indican en el uso de aplicaciones web: Más insegura, se necesita conexión a Internet y problemas de configuración y despliegue en producción.
- Nadie está satisfecho de tener una aplicación para cada tarea y a todos les gustaría tener un único sitio web que recoja todos los servicios.

Además, indican problemas actuales que tienen al usar las aplicaciones monolíticas, en los que se destaca:

- No lo pueden usar usuarios finales de Entidades y Mutuas
- Problemas de compatibilidad con Java. Algún externo no puede usarlas debido a problemas con varias versiones Java instaladas en su equipo.

#### 4.4.3. Resultados encuestas de satisfacción con aplicaciones en microservicios

La encuesta se ha facilitado además de a los usuarios actuales de las aplicaciones monolíticas, a los usuarios de la TGSS que solicitan cada vez los Justificantes y a los de Mutuas que más ficheros envían por IfiWeb y que han tenido que pedir el fichero de respuesta más frecuentemente y se tiene más confianza para que prueben el nuevo servicio. En total, a los 17 usuarios de la anterior encuesta se han sumado 16 usuarios nuevos, 4 de la TGSS y 12 de las mutuas, haciendo un total de 33 usuarios.

De los 33 usuarios han contestado 22 , y el detalle de las respuestas se puede ver en el Anexo B.

Se destaca lo siguiente:

- Han contestado el 100% de los usuarios de la TGSS y el 75% de las Mutuas.
- Sólo 1 usuario de Mutuas indica que no va a utilizar el servicio porque prefiere pedirlo como hace hasta ahora.
- Un par de usuarios indican mejoras a realizar en el futuro



## 5. Conclusiones y trabajo futuro

### 5.1. Conclusiones finales

Tras la realización del estudio y la prueba de concepto se llegan a las siguientes conclusiones:

- Hay mucha documentación acerca de los microservicios por lo que por una parte es bueno ya que prácticamente está todo documentado, pero puede pasar que no se sepa por dónde empezar. Por este motivo se tuvo que hacer un Sprint 0 para asimilar todo ese conocimiento para poder realizar la migración de las aplicaciones de manera adecuada.
- El uso de Spring Boot ha sido vital para poder tener las aplicaciones funcionando de manera rápida centrándonos sólo en programar utilizando la arquitectura de microservicios, ya que todo el tema de dependencias y servidor de aplicaciones lo facilitaba la herramienta.
- Docker es una herramienta para la creación de contenedores de aplicaciones muy potente que facilita el despliegue de estas y facilita el trabajo al personal de sistemas. En este caso no hemos necesitado un orquestador, ya que eran pocos contenedores, pero si hubiera sido el caso, kubernetes hubiera sido la herramienta elegida.
- La importancia de usar gestor de proyectos como Maven que facilite la exportación / importación del proyecto y que cualquiera lo pueda instalar en su equipo y ayudar con el desarrollo o mantenimiento de las aplicaciones.
- Los servicios de mensajería están preparados para gestionar gran cantidad de eventos y su configuración es relativamente sencilla, tanto si se quiere usar de manera individual o si se quiere instalar en modo clúster.
- Los usuarios finales han sido muy receptivos a la hora de utilizar las aplicaciones tras los cambios realizados, así como los funcionarios y externos, con la ganancia para todos de tiempo y velocidad.

## 5.2. Mejoras detectadas a futuro

Durante la realización de la prueba de concepto hemos detectado las siguientes mejoras que se podría realizar:

- La capa de presentación que se ha realizado usando Thymeleaf se puede mejorar utilizando un framework de Javascript como es Angular o React.js, comunicándose vía API REST con nuestro backend.
- Un microservicio que recibiera todos los eventos producidos por el microservicio de gestión de usuarios. El usuario haría login en él y se le ofrecerían los diferentes microservicios a los que tendría acceso. Con esto no hace falta conocer las url de cada microservicio. Sólo conociendo una, la del servicio central, se tendría acceso a todos los demás.
- Al microservicio de gestión de usuarios le añadiría un mantenimiento de aplicaciones y en el formulario de alta de usuario cambiaría de tipo input a tipo select, teniendo que coger la aplicación de unos valores cerrados recogidos del mantenimiento de aplicaciones.
- Cada microservicio podría tener un menú de usuario en el que poder modificar los datos, cambiar la contraseña y mostrar un histórico de las acciones realizadas.
- En el microservicio de IFIWeb se puede mostrar el motivo por el cual ha fallado el fichero de entrada para que así el usuario pueda modificar el fichero y se vuelva a procesar sin error.
- Para las imágenes de Docker, se utilizarían ficheros de propiedades para indicar los datos del servidor KAFKA de manera dinámica, y además se plantearía el uso de un contenedor de KAFKA en vez de tenerlo instalado en local en la máquina.

## Referencias bibliográficas

- ¿Qué es Kubernetes? | Kubernetes.* (n.d.). Retrieved May 18, 2021, from <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>
- [GWT].* (n.d.). Retrieved May 17, 2021, from <http://www.gwtproject.org/>
- 10 Best Java Frameworks to Use in 2021 [Updated].* (n.d.). <https://hackr.io/blog/java-frameworks>
- Apache Kafka.* (n.d.). Retrieved May 17, 2021, from <https://kafka.apache.org/>
- Apache Wicket.* (n.d.). Retrieved May 17, 2021, from <https://wicket.apache.org/>
- Apache ZooKeeper.* (n.d.). Retrieved May 17, 2021, from <https://zookeeper.apache.org/>
- Astic | Asociación Profesional de Cuerpos Superiores de Sistemas y Tecnologías de la Información de las Administraciones Públicas.* (n.d.). Retrieved April 29, 2021, from <https://www.astic.es/>
- Bejeck, B. (2018). *Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API.* Manning Publications.
- Empowering App Development for Developers | Docker.* (n.d.). Retrieved May 17, 2021, from <https://www.docker.com/>
- H2 Database Engine.* (n.d.). Retrieved May 16, 2021, from <https://www.h2database.com/html/main.html>
- Java | Oracle.* (n.d.). Retrieved May 17, 2021, from <https://www.java.com/es/>
- Kafka vs ActiveMQ vs RabbitMQ vs ZeroMQ - Digital Varys.* (n.d.). Retrieved May 16, 2021, from <https://digitalvarys.com/kafka-vs-activemq-vs-rabbitmq-vs-zeromq/>
- Kappa Architecture - Where Every Thing Is A Stream.* (n.d.). <https://milinda.pathirage.org/kappa-architecture.com/>
- Karanam, R. R. (2018). *Spring: Microservices with Spring Boot: Build and deploy microservices with Spring Boot.* Packt Publishing Ltd.
- Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in Microservices Architectures. *Procedia Computer Science*, 181(2019), 1225–1236.

<https://doi.org/10.1016/j.procs.2021.01.320>

Michael Good. (2018). *Thymeleaf with Preface*.

*Microservices*. (n.d.). Retrieved April 29, 2021, from <https://martinfowler.com/articles/microservices.html>

*Modelo para Presentación Telemática*. (n.d.). Retrieved May 18, 2021, from <https://www2.agenciatributaria.gob.es/L/inwinvoc/es.aeat.dit.adu.adht.editran.NumRefEditran?mod=347>

*MVC (Model, View, Controller) explicado*. (n.d.). Retrieved May 16, 2021, from <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>

Newman, S. (2020). *Monolith to Microservices Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media.

*Spring | Home*. (n.d.). <https://spring.io/>

*Spring Initializr*. (n.d.). Retrieved May 17, 2021, from <https://start.spring.io/>

*Spring MVC vs Apache Wicket detailed comparison as of 2021 - Slant*. (n.d.). Retrieved May 16, 2021, from [https://www.slant.co/versus/1436/23475/~spring-mvc\\_vs\\_apache-wicket](https://www.slant.co/versus/1436/23475/~spring-mvc_vs_apache-wicket)

Surovich, S., & Boorshtein, M. (2020). *Kubernetes and Docker - An Enterprise Guide: Effectively containerize applications, integrate enterprise systems, and scale applications in your enterprise*. Packt Publishing.

*Thymeleaf*. (n.d.). Retrieved May 17, 2021, from <https://www.thymeleaf.org/>

*Welcome to the Apache Struts project*. (n.d.). Retrieved May 17, 2021, from <https://struts.apache.org/>

*What is Scrum?* (n.d.). Retrieved June 18, 2021, from <https://www.scrum.org/resources/what-is-scrum>

*YMNNALFT: Easy Docker Image Creation with the Spring Boot Maven Plugin and Buildpacks*. (n.d.). Retrieved May 17, 2021, from <https://spring.io/blog/2021/01/04/ymnnalft-easy-docker-image-creation-with-the-spring-Boot-maven-plugin-and-buildpacks>

CQRS. (s. f.). <https://martinfowler.com/bliki/CQRS.html>

- spring-projects/spring-petclinic: A sample Spring-based application. (s. f.).  
<https://github.com/spring-projects/spring-petclinic>
- Kappa Architecture - Where Every Thing Is A Stream. (s. f.).  
<https://milinda.pathirage.org/kappa-architecture.com/>
- Baboi, M., Iftene, A., & Gîfu, D. (2019). Dynamic microservices to create scalable and fault tolerance architecture. *Procedia Computer Science*, 159, 1035-1044.  
<https://doi.org/10.1016/j.procs.2019.09.271>
- Baboi, M., Iftene, A., & Gîfu, D. (2019). Dynamic microservices to create scalable and fault tolerance architecture. *Procedia Computer Science*, 159, 1035-1044.  
<https://doi.org/10.1016/j.procs.2019.09.271>
- Bejeck, B. (2018). *Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API*. Manning Publications.
- Bellemare, A. (2020). *Building Event-Driven Microservices*. O'Reilly Media.
- Bradshaw, S., Brazil, E., & Chodorow, K. (2019). *MongoDB: The Definitive Guide 3e: Powerful and Scalable Data Storage*. O'Reilly Media.
- Bucchiarone, A., Mazzara, M., Dragoni, N., Dustdar, S., & Bjørndal, N. (2020). Migration from Monolith to Microservices: Benchmarking a Case Study. *March*.  
<https://doi.org/10.13140/RG.2.2.27715.14883>
- Garafolo, E. (2020). *Practical Microservices: Build Event-Driven Architectures with Event Sourcing and CQRS*. Pragmatic Bookshelf.
- Kane, S., & Matthias, K. (2018). *Docker: Up & Running: Shipping Reliable Containers in Production*. O'Reilly Medi.
- Karanam, R. R. (2018). *Spring: Microservices with Spring Boot: Build and deploy microservices with Spring Boot*. Packt Publishing Ltd.
- Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*, 131, 106449.  
<https://doi.org/10.1016/j.infsof.2020.106449>

- Lopez de Ipiña, D. (s. f.). Bases de Datos No Relacionales (NoSQL): Cassandra, CouchDB, MongoDB y.... <https://www.slideshare.net/dipina/nosql-cassandra-couchdb-mongodb-y-neo4j>
- Macero, M. (2020). Learn Microservices with Spring Boot. Apress.
- Martin, R. C. (2012). Código Limpio Manual de Estilo para el Desarrollo Ágil de Software. Anaya Multimedia.
- Mitra, R. (2020). Microservices: Up and Running: A Step-by-Step Guide to Building a Microservice Architecture. O'Reilly Media.
- Newman, S. (2015). Building Microservices. O'Reilly Media.
- Newman, S. (2020). Monolith to Microservices Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.
- Pacheco, V. F. (2018). Microservice patterns and best practices : Explore patterns like cqrs and event sourcing to create scalable, maintainable, and testable microservices (P. E. Central (Ed.)).
- Poulton, N. (2021). The Kubernetes Book. Independently published.
- Roldan Martinez, D., Valderas Aranda, P. J., & Torres Bosch, V. (2018). Microservicios: un enfoque integrado. RA-MA Editorial.
- Santomaggio, G., & Boschi, S. (2013). RabbitMQ Cookbook. Packt Publishing Ltd.
- Surovich, S., & Boorshtein, M. (2020). Kubernetes and Docker - An Enterprise Guide: Effectively containerize applications, integrate enterprise systems, and scale applications in your enterprise. Packt Publishing.
- Torres-berru, Y., Camacho-macas, J., & Solano-cabrera, J. (2020). Ciencias tecnicas y aplicadas Artículo de investigación. 6, 763-781.
- Urquhart, J. (2021). Flow Architectures: The Future of Streaming and Event-Driven Integration. O'Reilly Media.
- Valley, S., Group, P., Partner, G., & Partners, M. (2010). Praise for the The Art of Scalability. En Addison-Wesley.
- Vera-Rivera, F. H., Cuevas, C. M. G., & Astudillo, H. (2019). Desarrollo de aplicaciones

basadas en microservicios: tendencias y desafíos de investigación TT - Development of microservices-based applications: trends and research challenges. Revista Ibérica de Sistemas e Tecnologias de Informação, E23, 107-120.

Astic | Asociación Profesional de Cuerpos Superiores de Sistemas y Tecnologías de la Información de las Administraciones Públicas. (s. f.). Recuperado 29 de abril de 2021, de <https://www.astic.es/>

Microservices. (s. f.). Recuperado 29 de abril de 2021, de <https://martinfowler.com/articles/microservices.html>

## Anexo A. Encuesta Monolítico a Microservicios

### Monolítico a Microservicios

Encuesta Satisfacción aplicaciones monolíticas

**\*Obligatorio**

¿Eres funcionario o asistencia técnica? \*

☐ Funcionario

☐ Asistencia técnica

¿Prefieres aplicación de escritorio o aplicación web?

☐ Escritorio

☐ Web

☐ Indiferente

¿Conoces las ventajas de una aplicación de escritorio? Si es que sí, por favor indica alguna

Tu respuesta

¿Conoces alguna desventaja de una aplicación de escritorio? Si es que sí, por favor indica alguna

Tu respuesta

¿Conoces las ventajas de una aplicación web? Si es que sí, por favor indica alguna

Tu respuesta

Figura 20. Encuesta Monolítico a Microservicios, parte 1. (Elaboración propia)



¿Conoces alguna desventaja de una aplicación web? Si es que sí, por favor indica alguna

Tu respuesta

---

¿Estás contento con tener 5 aplicaciones independientes y que dependen de la versión de Java de tú máquina?

☐ Sí

☐ No

☐ Indiferente

¿Te gustaría tener un único sitio WEB al que acceder y poder utilizar las mismas herramientas independientes que utilizas ahora?

☐ Sí

☐ No

☐ Indiferente

¿Qué mejorarías de las aplicaciones actuales?

Tu respuesta

---

Enviar

Figura 21. Encuesta Monolítico a Microservicios, parte 2. (Elaboración propia)

## Anexo B. Encuesta Uso de los Microservicios

### Uso de los Microservicios

**\*Obligatorio**

¿Qué tipo de usuario eres? \*

☐ Funcionario

☐ Asistencia Técnica

☐ Usuario de Mutua

☐ Usuario de TGSS

¿Usarás los nuevos servicio proporcionado por SICOSS?

☐ Sí

☐ No

Si has contestado que sí, ¿Qué aplicación es la que usarás?

☐ Editrans

☐ IfiWeb

☐ Ambas

¿Qué mejorarías de las aplicaciones?

Tu respuesta

Enviar

Figura 22. Encuesta Monolítico a Microservicios, parte 2. (Elaboración propia)

## Índice de acrónimos

GISS: Gerencia Informática de la Seguridad Social.

NoSQL: No solo SQL

SOA: Service Oriented Architecture

HTTP: Hypertext Transfer Protocol

ACID: Atomicity, Consistency, Isolation, Durability

BD: Base de datos

SICOSS: Sistema de Información Contable de la Seguridad Social