

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 |
| Línea de investigación: | https://github.com/kokart/2021\_TFGUNIR\_mono\_to\_micro |
| Director/a: | Francisco José Soltero |
| Fecha: | 29/04/2021 |

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 las aplicaciones actuales a microservicios permitirá el uso distribuido de las aplicaciones sin necesidad de tener que 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 tener un entorno más robusto evitando disponer de 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 ben 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](#_Toc71999157)

[1.1. Justificación del tema elegido 8](#_Toc71999158)

[1.2. Problema y finalidad del trabajo 8](#_Toc71999159)

[2. Estado del arte 9](#_Toc71999160)

[2.1. Del monolítico a los microservicios 9](#_Toc71999161)

[2.1.1. Arquitectura monolítica 9](#_Toc71999162)

[2.1.2. Arquitectura Orientada a Servicios (SOA) 10](#_Toc71999163)

[2.1.3. Arquitectura de microservicios 12](#_Toc71999164)

[2.2. Tipos de base de datos: Relacionales y No Relacionales 16](#_Toc71999165)

[2.2.1. Bases de datos Relacionales 16](#_Toc71999166)

[2.2.2. Bases de datos No Relacionales 17](#_Toc71999167)

[2.2.3. Uso actual Bases de datos 21](#_Toc71999168)

[3. Objetivos del trabajo 22](#_Toc71999169)

[3.1. Objetivos principales 22](#_Toc71999170)

[3.1.1. Objetivos secundarios 22](#_Toc71999171)

[4. Diseño de la propuesta 23](#_Toc71999172)

[4.1. Contexto 23](#_Toc71999173)

[4.1.1. Situación inicial 23](#_Toc71999174)

[4.1.2. Aplicaciones monolíticas en la actualidad 25](#_Toc71999175)

[4.2. Contenidos 29](#_Toc71999176)

[4.2.1. Introducción 29](#_Toc71999177)

[4.2.2. Soluciones elegidas y componentes necesarios para la creación de los microservicios 31](#_Toc71999178)

[4.2.3. Creación microservicio Gestión de usuarios 31](#_Toc71999179)

[4.2.4. Configuración bróker recepción / envío eventos entre microservicios 31](#_Toc71999180)

[4.2.5. Creación microservicio Editrans 31](#_Toc71999181)

[4.2.6. Creación microservicio IfiWeb Mutuas 31](#_Toc71999182)

[4.3. Evaluación 32](#_Toc71999183)

[4.3.1. Introducción 32](#_Toc71999184)

[4.3.2. Resultados encuestas de satisfacción con aplicaciones monolíticas 32](#_Toc71999185)

[4.3.3. Resultados encuestas de satisfacción con aplicaciones en microservicios 34](#_Toc71999186)

[4.3.4. Conclusión 34](#_Toc71999187)

[5. Conclusiones y trabajo futuro 35](#_Toc71999188)

[5.1. Conclusiones finales 35](#_Toc71999189)

[5.1.1. “Título 3” del menú de estilos 35](#_Toc71999190)

[5.1.2. “Título 3” del menú de estilos 35](#_Toc71999191)

[5.2. Mejoras detectadas a futuro 35](#_Toc71999192)

[5.2.1. Nuevas funcionalidades 35](#_Toc71999193)

[5.2.2. “Título 3” del menú de estilos 35](#_Toc71999194)

[Referencias bibliográficas 36](#_Toc71999195)

[Índice de acrónimos 40](#_Toc71999196)

Índice de figuras

[Figura 1. https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec 12](#_Toc71999197)

[Figura 2. https://www.redhat.com/es/topics/microservices/what-are-microservices 13](#_Toc71999198)

[Figura 3. Poner donde es 14](#_Toc71999199)

[Figura 4. https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec 15](#_Toc71999200)

[Figura 5. Tipos de Bases de datos NoSQL https://docs.microsoft.com/es-es/dotnet/architecture/cloud-native/relational-vs-nosql-data 19](#_Toc71999201)

[Figura 6. Ranking https://db-engines.com/en/ranking 21](#_Toc71999202)

[Figura 7. Organigrama del Centro de Desarrollo de Intervención. (Elaboración propia) 23](#_Toc71999203)

[Figura 8. “Figuras” del menú de estilos. (Elaboración propia) 24](#_Toc71999204)

[Figura 9. “Figuras” del menú de estilos. (Elaboración propia) 25](#_Toc71999205)

[Figura 10. “Figuras” del menú de estilos. (Elaboración propia) 26](#_Toc71999206)

[Figura 11. “Figuras” del menú de estilos. (Elaboración propia) 26](#_Toc71999207)

[Figura 12. “Figuras” del menú de estilos. (Elaboración propia) 27](#_Toc71999208)

[Figura 13. “Figuras” del menú de estilos. (Elaboración propia) 28](#_Toc71999209)

[Figura 14. “Figuras” del menú de estilos. (Elaboración propia) 30](#_Toc71999210)

[Figura 15. “Figuras” del menú de estilos. (Elaboración propia) 32](#_Toc71999211)

[Figura 16. “Figuras” del menú de estilos. (Elaboración propia) 33](#_Toc71999212)

[Figura 17. “Figuras” del menú de estilos. (Elaboración propia) 33](#_Toc71999213)

[Figura 18. “Figuras” del menú de estilos. (Elaboración propia) 34](#_Toc71999214)

[Figura 19. “Figuras” del menú de estilos. (Elaboración propia) 34](#_Toc71999215)

Índice de tablas

[Tabla 1. Resumen bases de datos NoSQL 15](#_Toc70442839)

# 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.

## Justificación del tema elegido

Se ha elegido este tema para poder aprender e intentar aplicar el nuevo patrón arquitectónico de microservicios, así como poder utilizar bases de datos NOSQL para la gestión de los usuarios de las aplicaciones que vamos a implementar desde cero. 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.

## 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 la máquina, del despliegue de nuevas versiones y el que 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 gestión de usuarios para presentarla al departamento de Producción y Sistemas de la Gerencia Informática de la Seguridad Social, para poder desplegarla en producción acordes a los requisitos del centro y poder extender su uso a nuestros usuarios finales.

# Estado del arte

## Del monolítico a los microservicios

### 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 monolitos es un sistema en el que todo el código se implementa como un solo proceso, como en la Figura 1.

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.

### 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 a destacar 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 intrínsicamente 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.

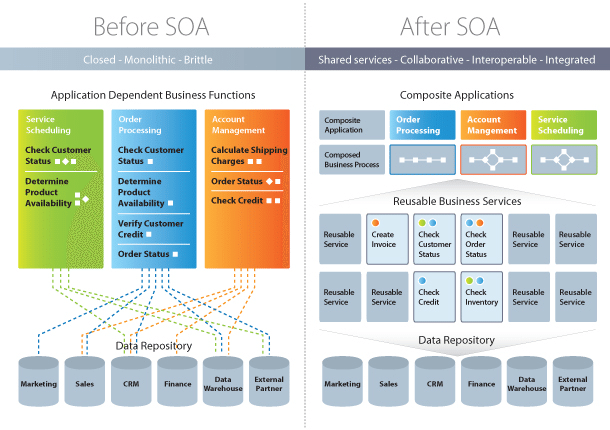
En la siguiente figura se representa la misma versión de un software usando monolítico o SOA:

Figura 1. https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec

### 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 están 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. Para una aplicación con una complejidad baja no se recomienda utilizar esta arquitectura.

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 muestra de manera gráfica la diferencia entre ambas arquitecturas:

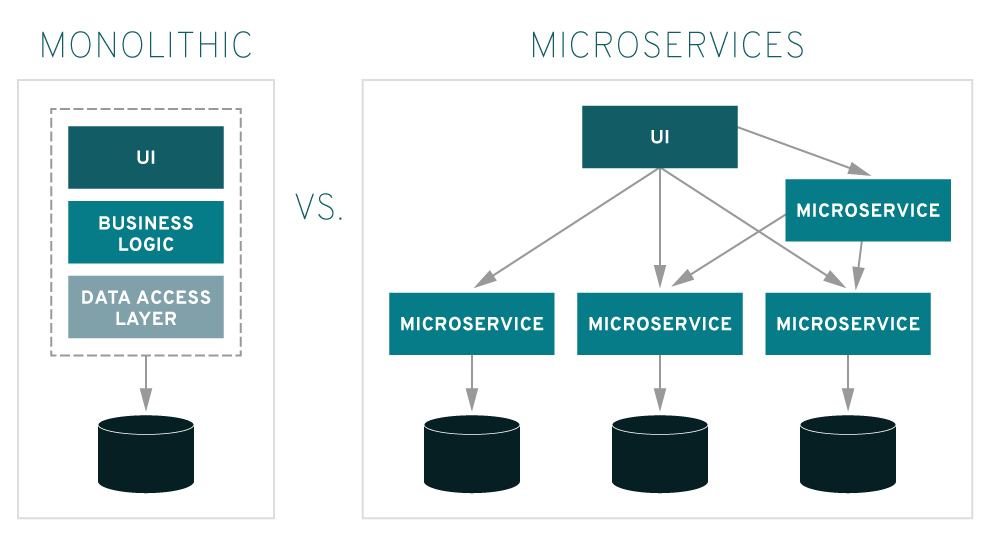


Figura 2. https://www.redhat.com/es/topics/microservices/what-are-microservices

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 4 muestra la relación entre ambas en uno de sus aspectos fundamentales, la escalabilidad.

Es un cubo 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

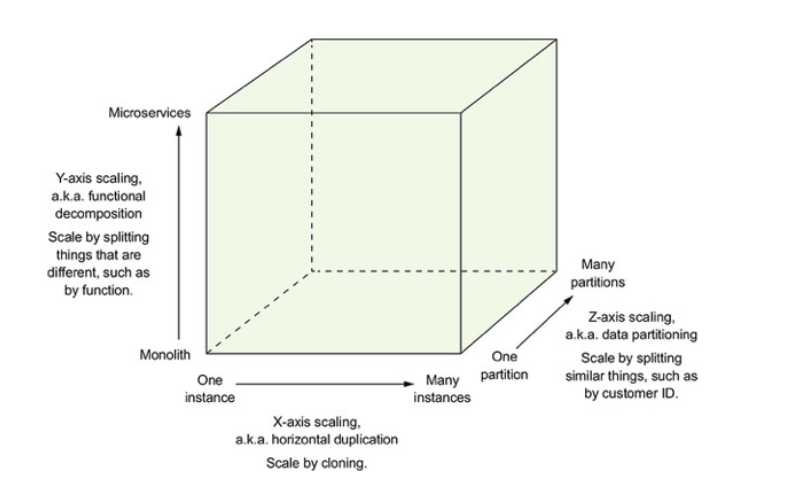


Figura 3. Poner donde es

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.

En la siguiente figura se puede ver de manera gráfica la diferencia entre ambas arquitecturas:

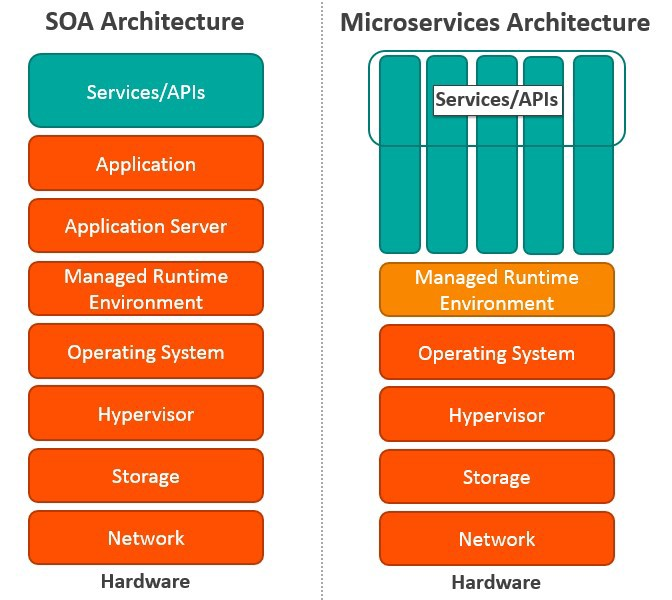


Figura 4. <https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec>

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.

## Tipos de base de datos: Relacionales y No Relacionales

### Bases de datos Relacionales

El modelo relacional fue propuesto originalmente por Edgar F. Codd en 1970. El principal objetivo de su propuesta era sentar las bases para el diseño y la implementación de Sistemas de Bases de Datos en los que las aplicaciones que accedieran o manipularan la información almacenada fueran totalmente independientes de cómo se implementara dicho almacenamiento.

Con el tiempo, el modelo relacional se ha consolidado como el más utilizado (ver figura 6) para las aplicaciones de procesamiento de datos, debido a su simplicidad y a su sólida base matemática, que han permitido el desarrollo de lenguajes y herramientas que facilitan el trabajo, tanto de los analistas y diseñadores, como de los programadores.

Está basado en la lógica de predicados y la teoría de conjuntos, lo que implica que se puede utilizar tanto el álgebra como el cálculo relacional para operar con los datos almacenados. El álgebra para describir la forma de realizar una consulta y el cálculo para obtener los valores que se desean devolver.

El modelo relacional se basa en el concepto matemático de relación. En este modelo, la información se representa en forma de tablas o relaciones, donde cada fila o tupla se interpreta como una relación ordenada de valores. Los datos se relacionan entre ellos a través de identificadores en tablas. El lenguaje predominante es SQL (Structured Query Language).

En el modelo relacional las tablas presentan además algunas características adicionales:

* Cada fila tiene que ser única.
* El orden de las filas dentro de una tabla no es relevante.
* El orden de las columnas que forman la tabla no es relevante.
* Los dominios de las distintas columnas no tienen que ser necesariamente disjuntos, pueden no coincidir, coincidir en parte o coincidir completamente.
* Dentro de una tabla, la intersección de una fila y una columna cualesquiera siempre es un dato único.

El éxito del modelo relacional condujo a la proliferación de gestores de bases de datos que aseguraban ser relacionales. Para poder determinar con certeza si un gestor era fiel al modelo relaciona, Codd publicó en 1984 un conjunto de 12 reglas (Codd, 1985). que definen las características esperadas de un sistema relacional (cuanto mayor sea el número de reglas cumplidas por el gestor, siempre por encima del mínimo de seis, más próximo estará al ideal del modelo)

Por último, las bases de datos relacionales tradicionales nos permiten definir la estructura de un esquema que demanda reglas rígidas y garantizan ACID:

Las características ACID (Elmasri, 2007) hacen referencia a:

* Atomicidad: Los resultados de una transacción o bien pasas a ser completados todos (commit) o bien pasan a ser todos deshechos (rollback). Es decir, o todos los cambios incluidos en una transacción tienen efecto o no lo tiene ninguno.
* Consistencia: Las bases de datos se transforman de estados íntegros a estados íntegros, es decir, entre estados válidos. Una transacción sólo se puede completar si el estado final es íntegro.
* Aislamiento: Los resultados de una transacción son inviables para el resto de las transacciones de otros procesos hasta que la transacción se ha completado.
* Durabilidad: Una vez una transacción ha sido completada, los resultados (cambios) de la transacción se hacen permanentes, incluso frente a fallos del sistema y de medios de almacenamiento.

Y como ejemplos de Bases de datos relacionales encontramos los dos más usados en la actualidad tal como se muestra en la figura 7: Oracle y MySQL.

### Bases de datos No Relacionales

El término NoSQL, acuñado por primera vez en 1998 por Carlo Strozzi, surge con la llegada de la web 2.0 y las aplicaciones como Facebook, Twitter o YouTube, donde cualquier usuario podía subir contenido a la red, provocando así un crecimiento masivo de los datos. Es en ese momento cuando empiezan a surgir los problemas a la hora de gestionar toda esa información generada y almacenada en bases de datos relacionales.

Esta es la principal razón por la cual surgen las BBDD NoSQL (Not only SQL - No solo SQL) que permiten resolver los problemas de escalabilidad y rendimiento que presentan estos tamaños tan grandes, mediante nuevos entornos de manejo de datos distribuidos y escalables de forma horizontal (muchos discos en paralelo).

El volumen, la variedad y la velocidad con la que se generaban esos datos requerían de nuevas tecnologías que solucionaran estos problemas. De esta forma apareció el movimiento NoSQL como solución a todos los problemas que las bases de datos relacionales no podían tratar. Las bases de datos NoSQL son sistemas de almacenamiento que no cumplen con el esquema entidad-relación, esquema característico en las bases de datos relacionales. Estas nuevas tecnologías hacen uso de nuevas y diferentes estructuras para el almacenamiento de los datos.

No siguen un patrón fijo como estructura de almacenamiento por lo que su uso es muy común cuando no se tiene un esquema exacto de lo que se va a almacenar.

Las ventajas respecto a los sistemas relacionales(Acens, 2014) son:

* Se pueden ejecutar en máquinas de pocos recursos
* Pueden manejar gran cantidad de datos. Esto es debido a que utiliza una estructura distribuida, en muchos casos mediante tablas Hash.
* Escalabilidad horizontal: La mejora de rendimiento de estos sistemas simplemente se puede obtener añadiendo más nodos.
* No genera cuellos de botella: Los sistemas SQL relacionales necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema.

Y las diferencias principales(Acens, 2014) con las Bases de Datos relacionales son:

* No usan SQL como lenguaje de consultas. Casi todas las bases de datos NoSQL evitan usar este tipo de lenguaje o lo utilizan como un lenguaje de apoyo. Por ejemplo, Cassandra utiliza el lenguaje CQL, MongoDB utiliza JSON o BigTable hace uso de GQL.
* No utilizan estructuras fijas como tablas para el almacenamiento de los datos. Usan otros tipos de modelos como sistemas clave-valor, objetos o grafos.
* No suelen permitir operaciones JOIN. Al contener un volumen de datos tan grande es deseable evitar los JOIN. Esto es debido a que cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa.
* Arquitectura distribuida. Las bases de datos relacionales suelen estar centralizadas en una máquina o bien en una estructura máster-esclavo; sin embargo, en los casos NoSQL la información puede estar compartida en varias máquinas mediante mecanismos de tablas hash distribuidas.

Hay varios tipos de bases de datos NoSQL. Se suelen clasificar en cuatro tipos diferentes: documentales, grafo, clave-valor y orientadas a columnas.

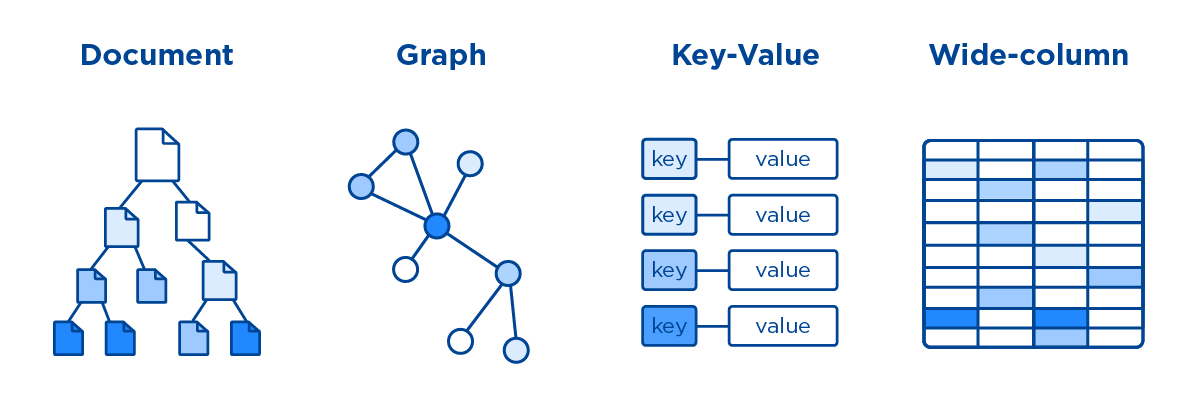


Figura 5. Tipos de Bases de datos NoSQL https://docs.microsoft.com/es-es/dotnet/architecture/cloud-native/relational-vs-nosql-data

A continuación, se habla de cada una de ellas de manera resumida.

#### Bases de datos clave-valor

Cada elemento en la base de datos se almacena como un par (clave-valor) donde la clave sirve como un identificador único. Tanto la clave como el valor pueden ser cualquier tipo de primitivo, como objetos compuestos.

Son BD de este tipo: DynamoDB, Redis, Riak, Voldemort.

#### Bases de datos documentales

Los datos se almacenan y se consultan como documentos tipo JSON. Los documentos pueden contener muchos pares diferentes clave-valor, o incluso documentos anidados. Son más flexibles al cambio ya que si el modelo de datos necesita cambiar, solo se deben actualizar los documentos afectados y no todo el esquema.

Son BD de este tipo: MongoDB, CouchDB.

#### Bases de datos columnares

Permiten manejar un gran volumen de datos y mezclan conceptos de las bases de datos relacionales con una base de datos clave-valor. Almacenan tablas de datos como secciones de columnas en lugar de filas de datos y en cada sección se puede encontrar elementos con clave-valor.

Son BD de este tipo: Cassandra, Hbase, Microsoft Azure.

#### Bases de datos orientadas a grafo

Usan nodos para almacenar entidades de datos y aristas para almacenar las relaciones entre nodos. El valor de estas bases de datos se obtiene de las relaciones entre nodos y no hay un límite para el número de relaciones que un nodo pueda tener. Un borde siempre tiene un nodo de inicio, un nodo final, un tipo y una dirección.

Son BD de este tipo: Neo4J, HyperGraphDB.

#### Resumen de las Bases de datos NoSQL

Resumidos los tipos de base de datos NoSQL, a continuación, se muestra una tabla resumen con las principales características de cada una de ellas:

Tabla 1. Resumen bases de datos NoSQL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Características | Tipos de Bases de datos NoSQL | | | |
| Clave-valor | Documental | Columnar | Grafos |
| Modo almacenamiento | Diccionario | Documento y clave-valor | Columnar y clave-valor | Grafo |
| Consistencia | Consistencia eventual | Consistencia eventual | Consistencia eventual | Consistencia eventual |
| Disponibilidad | Alta | Alta | Alta | Alta |
| Escalabilidad | Alta | Alta | Alta | Alta |
| Recomendado | Entornos distribuidos. Aplicaciones que creen progresivamente | Consultas avanzadas. Datos semiestructurados. Variedad de datos | Procesamiento y escaneo de grandes cantidades de datos | Datos conectados, modelos con muchas relaciones |
| Ejemplos | DynamoDB,Redis | MongoDB,CouchDB | Cassandra,Hbase | Neo4j, SparkSee |

Si bien hay sitios que hablan de un quinto tipo, orientado a objetos, no se ha tenido en cuenta en este estudio ya que su uso es muy reducido en comparación con las otras 4.

### Uso actual Bases de datos

A pesar de las ventajas que tienen los sistemas no relacionales, aún se siguen utilizando de manera mayoritaria los sistemas relacionales. Al final uno no ha llegado para sustituir a otro, si no para complementarlo y llegar a cubrir los aspectos en los que una es peor que la otra, para así al final tener la mejor opción posible para el uso que se desee.

La siguiente figura muestra el ranking de bases de datos utilizados, a fecha abril 2021, en la cual se destaca que los 4 más usados son sistemas relacionales, y ya, en el quinto lugar aparece MongoDB, como la primera base de datos NOSQL más utilizada.

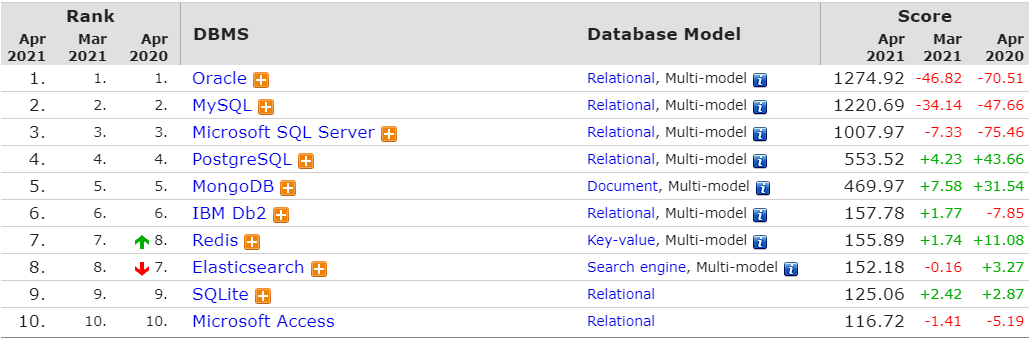


Figura 6. Ranking https://db-engines.com/en/ranking

# Objetivos del trabajo

## 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 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.

### 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 RabittMQ.
2. Conocer y usar software para la creación de microservicios, 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.

# Diseño de la propuesta

## Contexto

### 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.

El organigrama del centro es el siguiente:

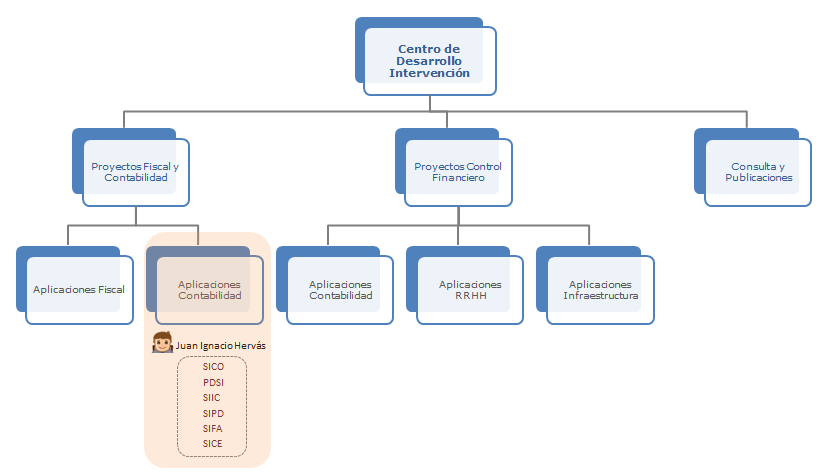


Figura 7. Organigrama del Centro de Desarrollo de Intervención. (Elaboración propia)

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 estructura organizativa de los usuarios de SICOSS es la siguiente:

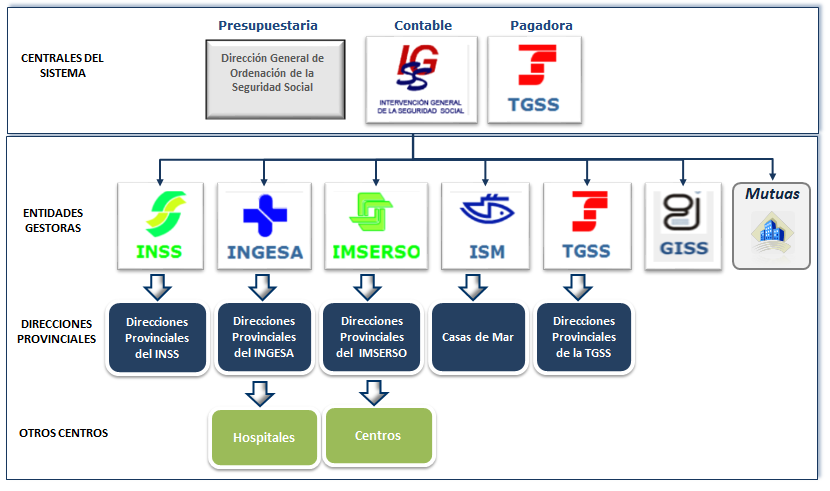


Figura 8. “Figuras” del menú de estilos. (Elaboración propia)

### 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 siguiente forma:

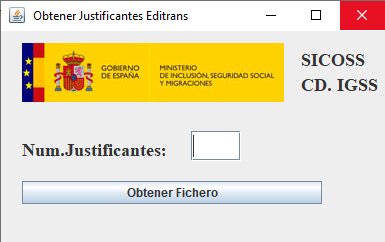


Figura 9. “Figuras” del menú de estilos. (Elaboración propia)

Está aplicación lo que hace es dejarnos 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 <https://www2.agenciatributaria.gob.es/L/inwinvoc/es.aeat.dit.adu.adht.editran.NumRefEditran?mod=347>

Y se queda con el número de justificante:

Esto lo hace las veces que le indicamos como parámetro de entrada “Num.Justificantes”.

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

La aplicación es un .jar que se ejecuta y tiene la siguiente forma:

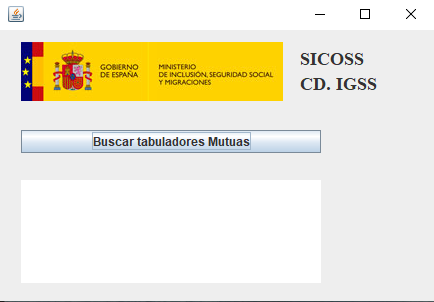


Figura 10. “Figuras” del menú de estilos. (Elaboración propia)

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, nos indica el código de las Mutuas que tienen tabuladores.

1. Obtener descripción clasificación orgánica:

La aplicación es un .jar que se ejecuta y tiene la siguiente forma:



Figura 11. “Figuras” del menú de estilos. (Elaboración propia)

Esta aplicación nos 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.

1. Generar fichero de salida IFIWeb:

La aplicación es un .jar que se ejecuta y tiene la siguiente forma:

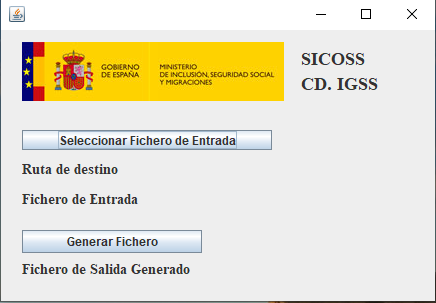


Figura 12. “Figuras” del menú de estilos. (Elaboración propia)

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.

1. Generar Fichero de documentos contables ADOK COVID:

La aplicación es un .jar que se ejecuta y tiene la siguiente forma:

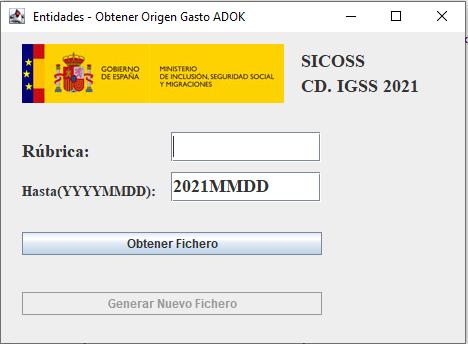


Figura 13. “Figuras” del menú de estilos. (Elaboración propia)

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>

## Contenidos

### 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 nos 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 siguiente figura muestra el esquema final al que se desea llegar:

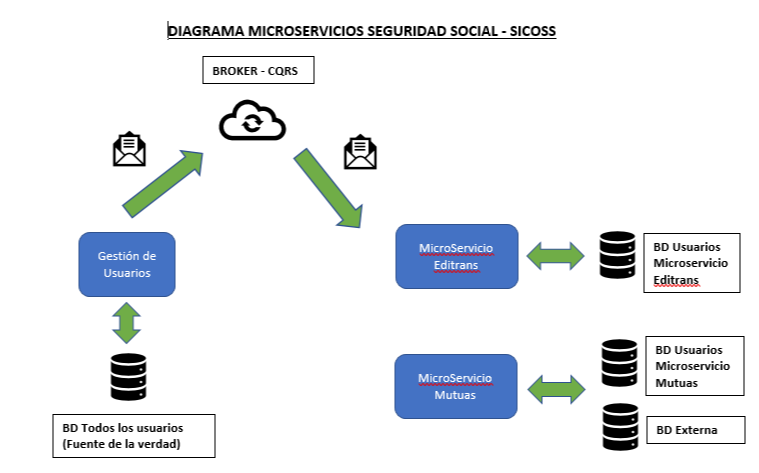


Figura 14. “Figuras” del menú de estilos. (Elaboración propia)

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.

### Soluciones elegidas y componentes necesarios para la creación de los microservicios

La forma en la que se ha realizado el trabajo ha sido la siguiente:

1. Elegir la tecnología de desarrollo entre las distintas opciones disponibles.
2. Elegir la forma de comunicación entre microservicios.
3. Elegir como presentar cada microservicio.

Para la tecnología de desarrollo se ha optado por Spring Boot, que es un framework que pertenece a Spring Framework y sobre el cual ya tengo experiencia durante mi vida profesional. Las alternativas posibles a esta elección son varias, entre ellas Struts, Java Server Faces, Google Web Toolkit y Apache Wicket.

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.

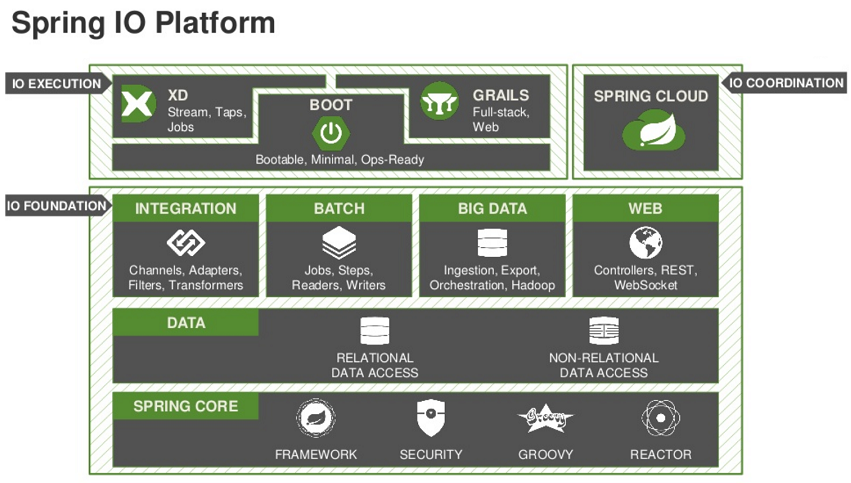


Figura 15. https://betacode.net/11267/spring-boot-tutorial-for-beginners

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 o JSP integrados para realizar la capa de presentación de la aplicación,

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 2. Resumen ventajas e inconvenientes comunicaciones

|  |  |  |
| --- | --- | --- |
|  | Síncrono | Asíncrono |
| Ventajas | * Menor sobrecarga * Rendimiento más alto * Al necesitar confirmación, la comunicación es más rápida y en tiempo real | * Simple * Menor coste * Configuración rápida |
| Desventajas | * Más compleja * Se necesitan conocimientos avanzados * Más caro | * No es tiempo real * Eficiencia menor * Mayor riesgo de overhead |

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

* Apache Kafka
* RabbitMQ
* ActiveMQ
* ZeroMQ

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.)

Respecto a los contenedores, se decide el uso de la tecnología Docker.

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

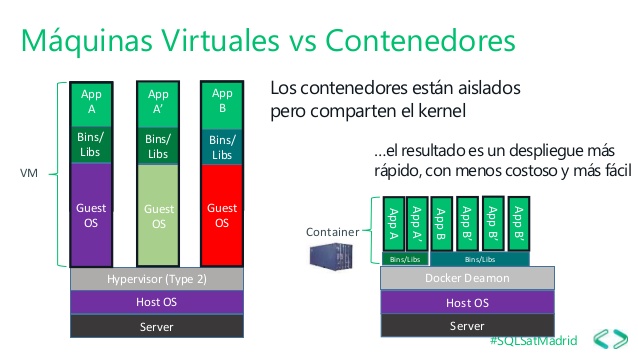


Figura 15 https://www.slideshare.net/AdrianDiazCervera/dockeriza-tu-sql-server

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 gestionar y automatizar del despliegue de contenedores.

Kubernetes 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 contenedores.

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

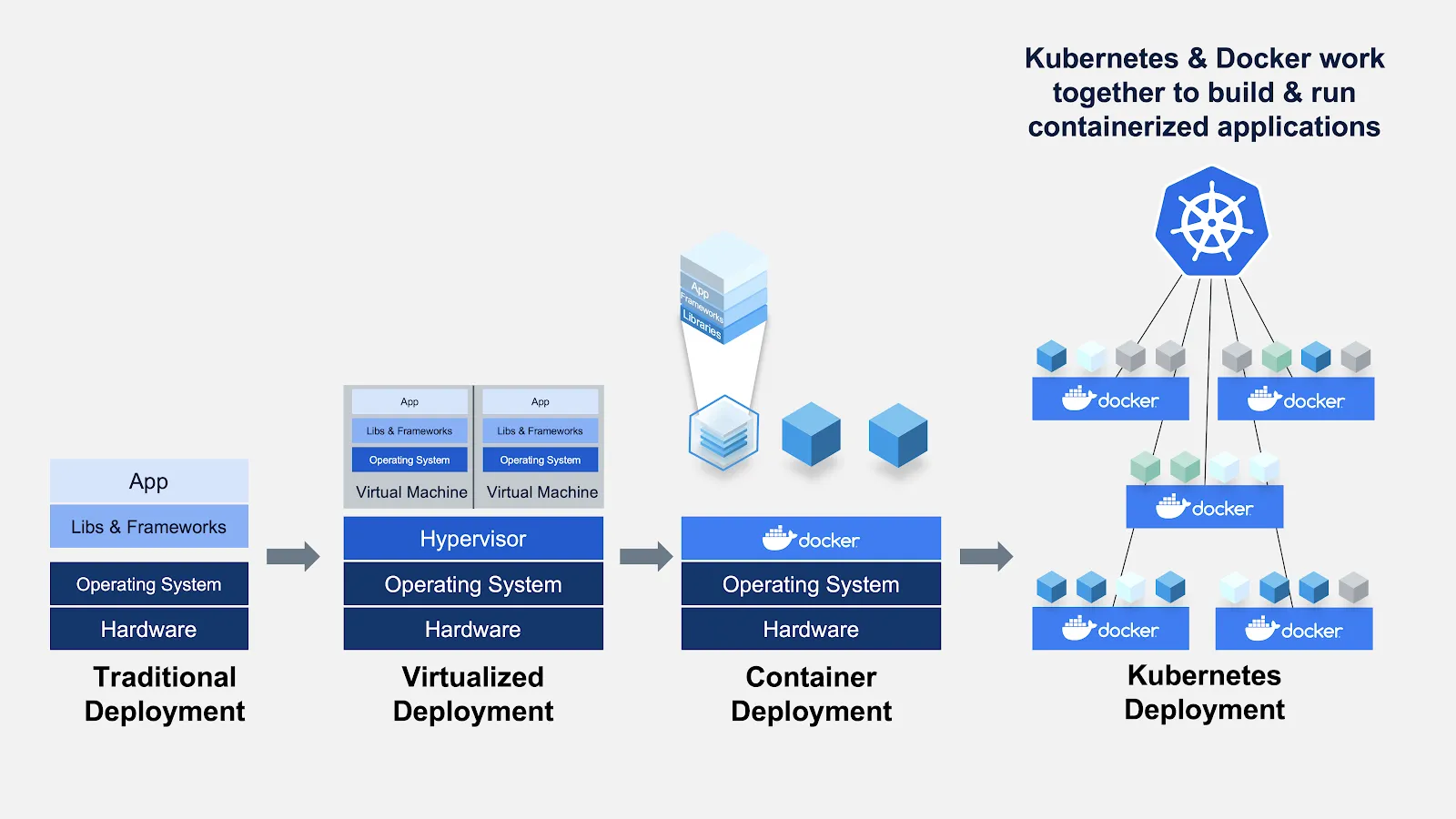


Fig 16. Kubernetes . web kubernetes

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

### 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 la gestión de usuarios, y dos suscriptores, que serán las aplicaciones monolíticas que hemos llevado 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.

La representación gráfica sigue a continuación:

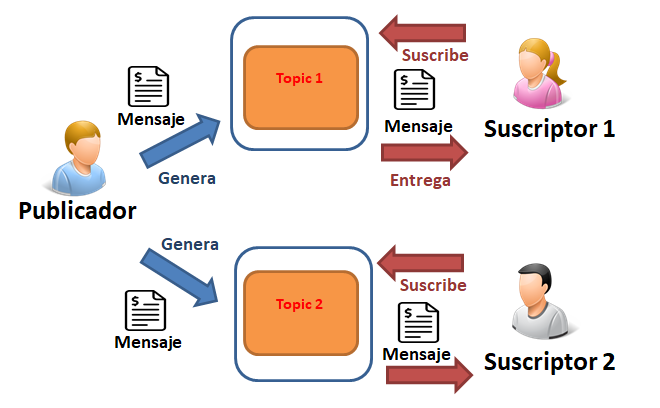


Fig 17. Arquitectura con Kafka https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/

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

1. Descargar el software Apache Kafka(*Apache Kafka*, n.d.)
2. Arrancar el Zookeper
3. Arrancar el bróker
4. Crear los dos topics

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.

### Creación microservicio Gestión de usuarios

Se crea una aplicación nueva para la Gestión de los usuarios 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 brokery *com.h2database* para la base de datos.

La integración con Kafka es tan sencilla como indicar en el applicatión.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.

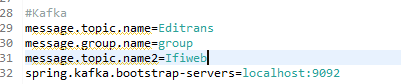


Fig 17. Application properties

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



Fig 18. Clase Kafka

Además, se cuenta con la entidad User para la gestión de los usuarios con su clase controladora, modelo y repositorio.

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 características que ofrece.

Y tal como hemos 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.

La página de bienvenida del servicio tiene la siguiente forma:

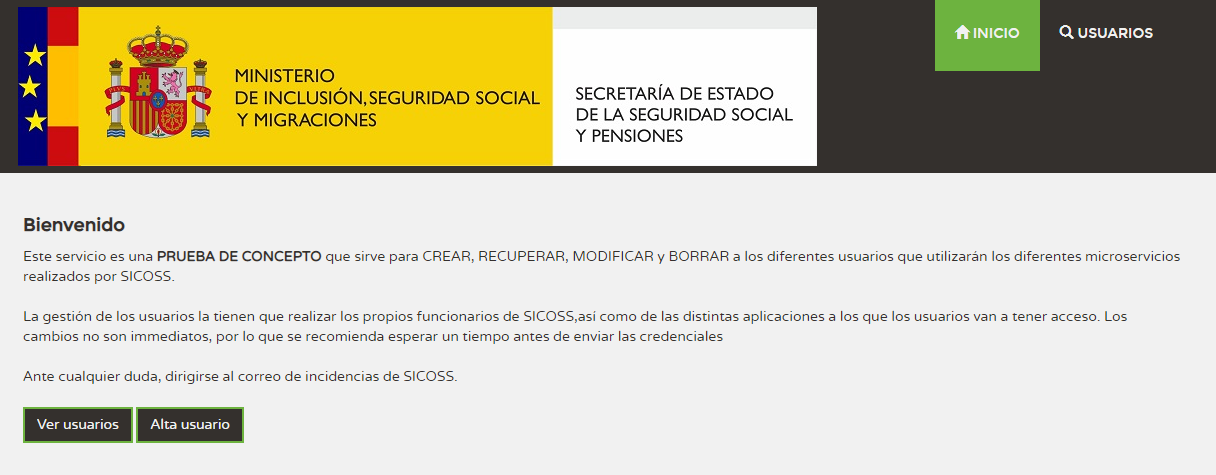


Fig 19. Clase Kafka

Y 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>

### Creación microservicio Editrans

🡪Explicación fuente de datos local, foto despliegue, foto login y foto obteniendo justificantes

### Creación microservicio IfiWeb Mutuas

🡪Explicación doble fuente de datos(local y donde está la info de mutuas), foto despliegue, foto login usuario mutuas y usuario funcionario, obtención fichero de recuperación.

## Evaluación

### 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 se puede encontrar en el siguiente enlace:

<https://docs.google.com/forms/d/e/1FAIpQLSeKF0EBclLyMAGh44RsBQrXMss4YAH3EggTObL_3P-OelUWzQ/viewform>

Mientras que la segunda encuesta se puede encontrar en:

<https://docs.google.com/forms/d/e/1FAIpQLSeKF0EBclLyMAGh44RsBQrXMss4YAH3EggTObL_3P-OelUWzQ/viewform>

Y se ha encuestado a los funcionarios y a la asistencia técnica (externos) que trabajan en SICOSS y que utilizan algunas de las aplicaciones monolíticas actuales.

### Resultados encuestas de satisfacción con aplicaciones monolíticas

🡪Hablar de la gente que ha participado y de los resultados obtenidos. Ahora mismo sólo están las respuestas en crudo. Falta gente por contestar y analizar las respuestas

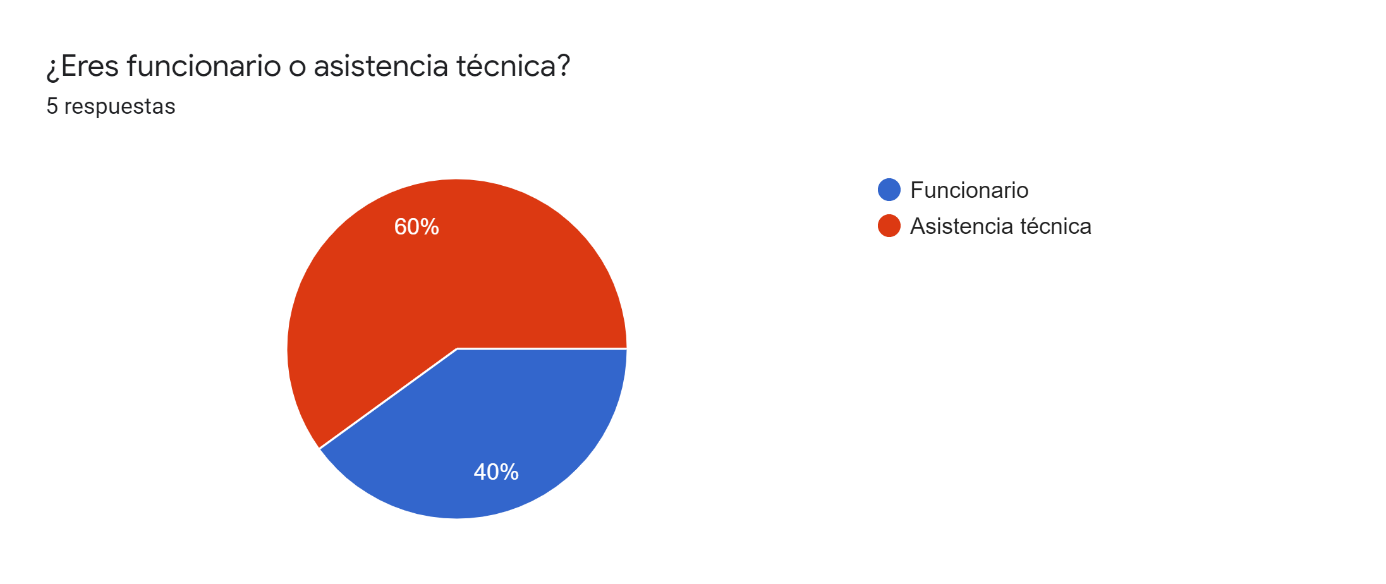


Figura 15. “Figuras” del menú de estilos. (Elaboración propia)

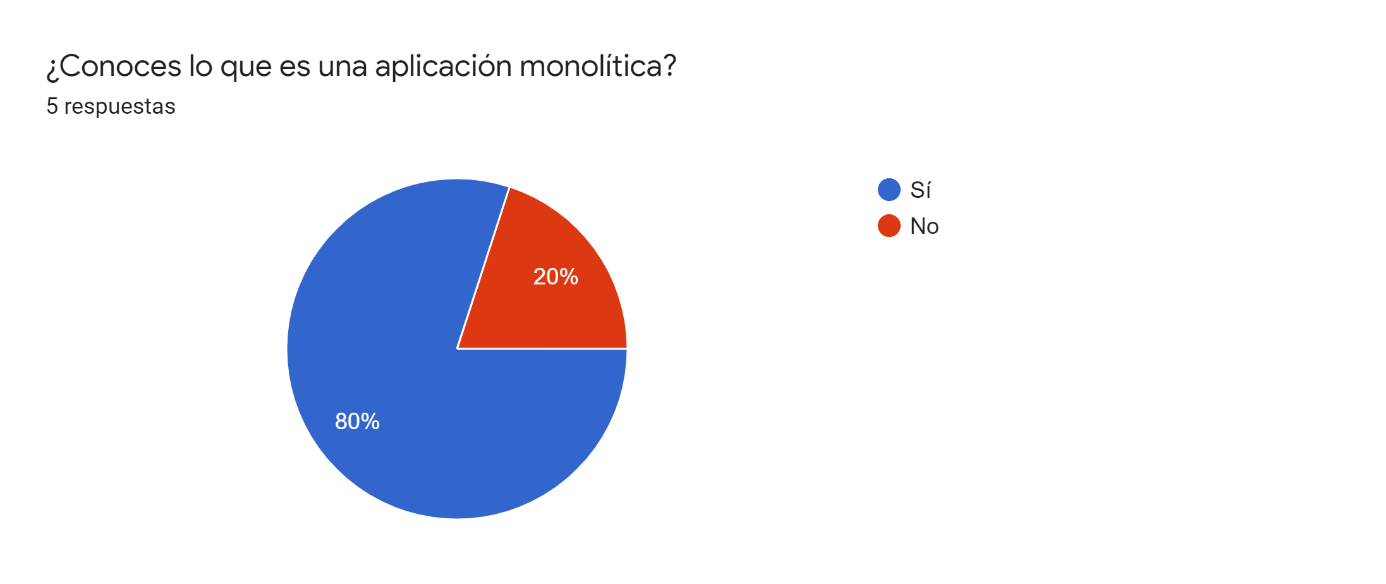


Figura 16. “Figuras” del menú de estilos. (Elaboración propia)

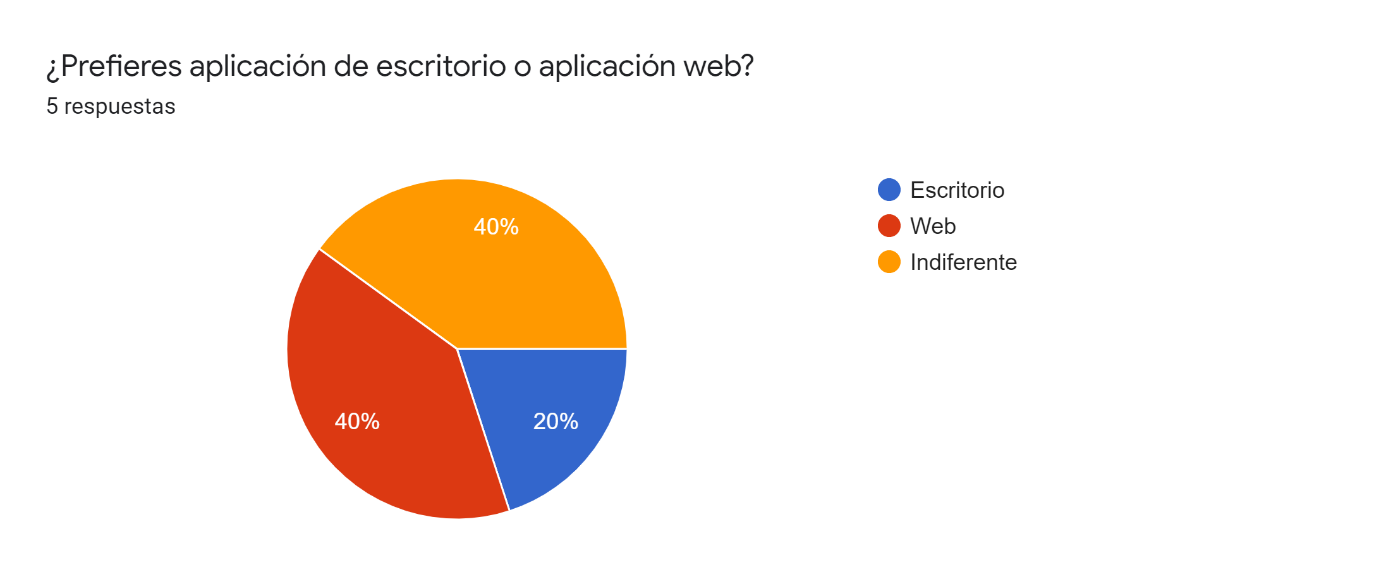


Figura 17. “Figuras” del menú de estilos. (Elaboración propia)

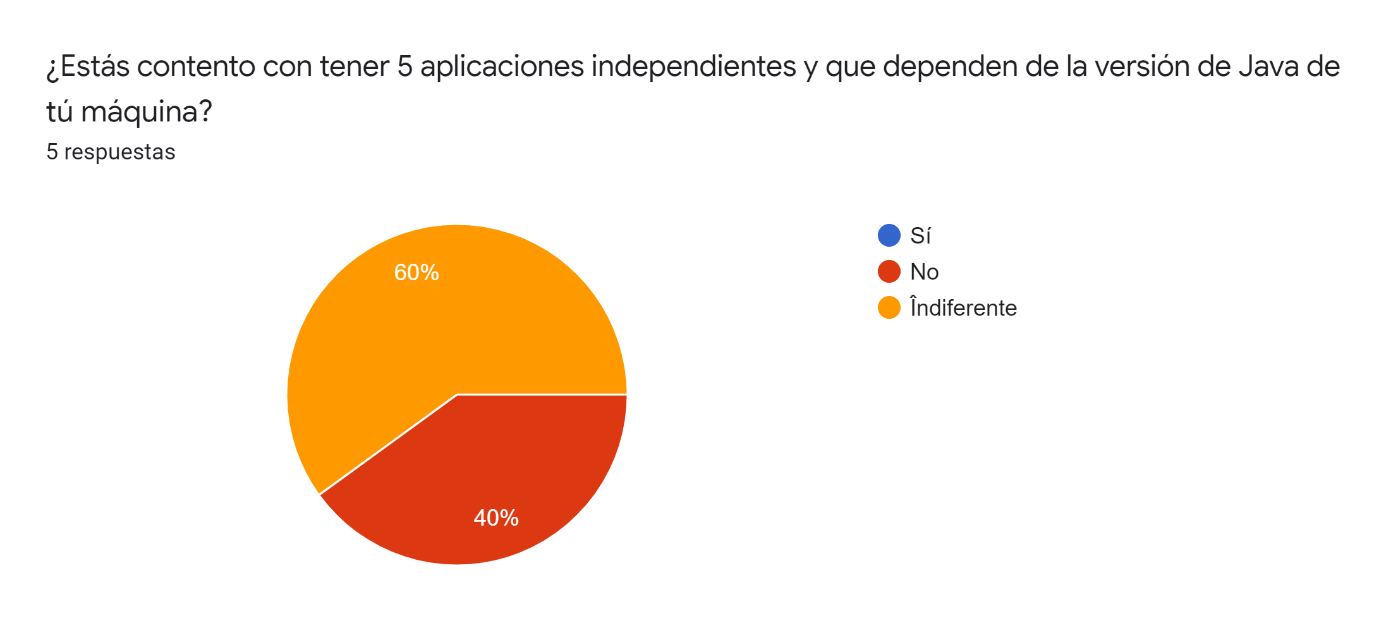


Figura 18. “Figuras” del menú de estilos. (Elaboración propia)

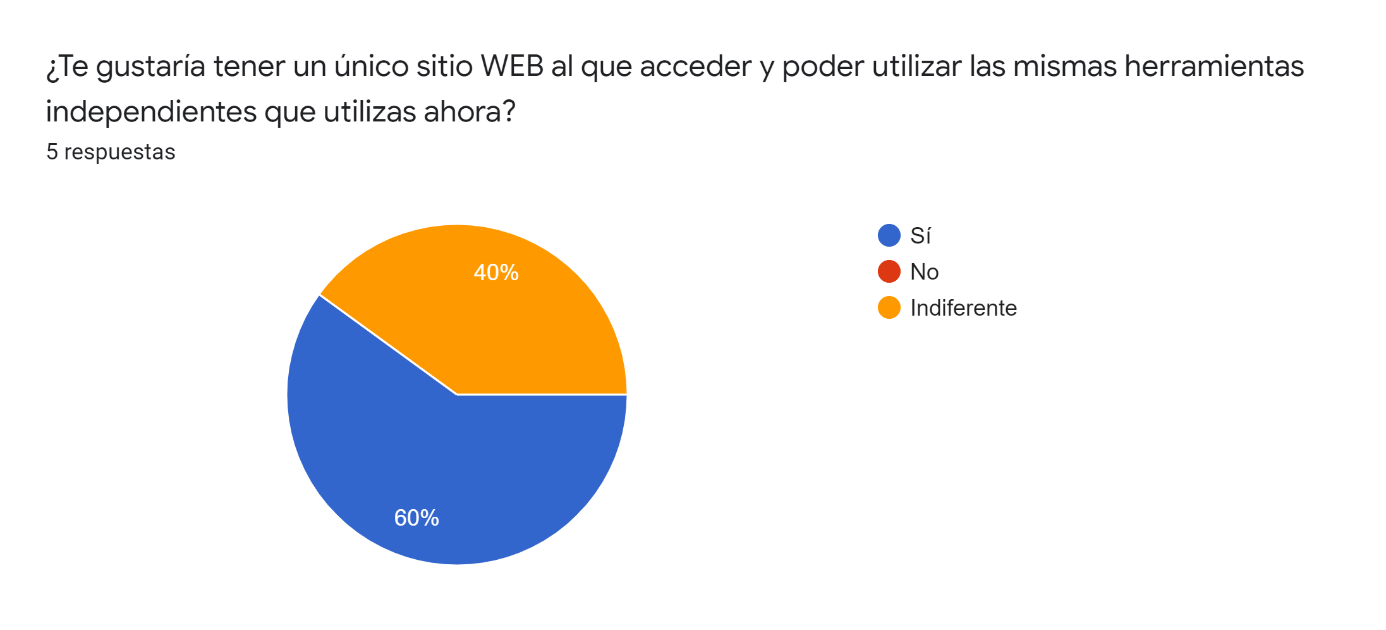


Figura 19. “Figuras” del menú de estilos. (Elaboración propia)

### Resultados encuestas de satisfacción con aplicaciones en microservicios

🡪Hablar de la gente que ha participado y de los resultados obtenidos

### Conclusión

# Conclusiones y trabajo futuro

## Conclusiones finales

### “Título 3” del menú de estilos

### “Título 3” del menú de estilos

#### "Título 4" del menú de estilos

#### "Título 4" del menú de estilos

## Mejoras detectadas a futuro

### Nuevas funcionalidades

Referencias bibliográficas

*10 Best Java Frameworks to Use in 2021 [Updated]*. (n.d.). https://hackr.io/blog/java-frameworks

Acens. (2014). Base de datos NoSQL. Qué son y tipos que nos podemos encontrar. *Acens.Com*, 7. http://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

*Apache Kafka*. (n.d.). Retrieved May 17, 2021, from https://kafka.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.

Codd, E. F. (1985). *RELATIONAL MODEL DATABASE MANAGEMENT*. Addison-Wesley.

Elmasri, R. A. (2007). *FUNDAMENTOS DE SISTEMAS DE BASES DE DATOS*. Grupo Anaya Publicaciones Generales.

*H2 Database Engine*. (n.d.). Retrieved May 16, 2021, from https://www.h2database.com/html/main.html

*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

*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

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.

*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

Habría que añadir:

-Página de la gerencia informática

-Pagina de docker

-Pagina de Springboot

-Página de kafka

-Pagina de martinfowler y microservicios

-Página de mongoDB

-Página de ranking bd

-Página de redhat

-Página java, maven, gradle

¿Acoplamiento o cohesion aquí o a pie de pagina?

Índice de acrónimos

GISS

NoSQL

SOA

HTTP

ACID

BD

SICOSS