Máster Universitario de Investigación en Ingeniería del Software y Sistemas Informáticos

INTEGRACIÓN DE DATOS CON SOA Y MSA EN LA INDUSTRIA DEL TRANSPORTE MARÍTIMO DE CONTENEDORES

31105151 – Arquitecturas Orientadas a Servicios

Miguel Sánchez Belenguer – 44516492S

Tutora: Elena Ruiz Larrocha

Curso 2018 / 2019 – Convocatoria de Septiembre

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 24/09/2019

Quién suscribe:

Autor(a): Miguel Sánchez Belenguer
D.N.I/N.I.E/Pasaporte.: 44516492S

Hace constar que es la autor(a) del trabajo:

Título completo del trabajo.

INTEGRACIÓN DE DATOS CON SOA Y MSA EN LA INDUSTRIA DEL TRANSPORTE MARÍTIMO DE CONTENEDORES

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.

Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.

No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

IMPRESO TFDM05_AUTORPBL
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS

Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Fdo.

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 RESUMEN

Abstract:

In the ports and terminals industry, there are more and more individual software systems and most of them are completely unconnected. From Terminal Operating Systems (TOS) to third party applications or in-house developments, the risk is to have same data stored with different values and within different systems. It is a challenge for stakeholders and companies to find a single source of truth and get more confidence in the data they use for their daily operations, moving millions of euros and millions of tons behind them. The proposed solution takes the advantage of the Service Oriented Architectures to combine and reuse different data sources and, through a weight average predictive system, improve the quality of the estimated time of arrival of a vessel, as an example of a value that can be improved and shared with all parties interested in through an algorithm capable to learn from the different sources it has connected.

El presente Trabajo de Fin de Máster analiza las principales soluciones software existentes en el sector del transporte marítimo de contenedores, así como las iniciativas se encuentran actualmente en desarrollo o previstas para el futuro. Se presenta el estado actual del ecosistema digital marítimo así como las principales propuestas a corto y medio plazo, de manera que puedan aportar una visión más clara de hacia dónde se dirige el sector y donde se podrían localizar las principales oportunidades de negocio.

Por otro lado se analiza la evolución reciente de la arquitectura basada en servicios hacia la arquitectura basada en micro servicios, así como las últimas tendencias y principales herramientas necesarias para una implementación. Se incluyen conceptos relacionados con la integración de sistemas software, arquitectura orientada a servicios y micro servicios y software ofrecido a través de servicios, entre otros.

Combinando estos dos elementos, el estado del arte actual en el transporte marítimo de contenedores y las últimas tecnologías, se propone desarrollar e implementar un caso de uso real que sirva como escenario para analizar los resultados, valorando que beneficios puede aportar tanto desde un punto de vista tecnológico como desde el punto de vista del negocio. Esta propuesta es un sistema de predicción para mejorar un dato particular empleado a diario en las terminales de contenedores, que es utilizado por diferentes sistemas y organizaciones implicadas: El tiempo estimado de llegada de un barco.

A través de una media ponderada y una serie de fuentes disponibles que proporcionan la misma variable pero con diferente valor, se van a realizar diferentes ejecuciones para estimar la fiabilidad de cada origen de datos. Esto va a permitir predecir de forma más fiable cual es el tiempo estimado de llegada del barco. A su vez el sistema será capaz de emitir mejores predicciones contra más estimaciones se hayan realizado, ya que se asignará mejor el peso de cada una de las fuentes para el cálculo a través de dos algoritmos propuestos.

2 OBJETIVOS E INTRODUCCIÓN

Los objetivos de este Trabajo de Fin de Máster son los siguientes:

- Analizar la situación actual del software y las tecnologías de la información presentes en el sector del transporte marítimo de contenedores.
- Analizar los proyectos más relevantes y las iniciativas planificadas de cara al futuro, así como las apuestas más innovadoras e impactantes.
- Analizar las carencias, necesidades y oportunidades del sector desde un punto de vista tecnológico, atendiendo especialmente a las oportunidades de integración de las diferentes plataformas existentes y los beneficios que se pueden obtener al respecto.
- Investigar las últimas tendencias y tecnologías en cuanto a la integración de sistemas, diseño de arquitecturas orientadas a servicios y arquitecturas basadas en micro servicios.
- Definir un caso de uso concreto y real que esté relacionado con el sector, en el que intervengan diferentes fuentes de datos.
- Implementar una integración con tecnologías de vanguardia a partir de este caso de uso, de manera que se habilite un escenario sobre el que poder analizar resultados.
- Plasmar las conclusiones que apoyen y justifiquen los beneficios de la integración de sistemas, así como el uso de arquitecturas orientadas a servicios o micro servicios.

Los motivos que justifican la elección del trabajo seleccionado son los siguientes:

- Aplicar los conocimientos adquiridos en la asignatura SOA (Arquitecturas Orientadas a Servicios), a través de un sector concreto y un caso de estudio específico.

- Ampliar y consolidar los conocimientos del autor del trabajo, como empleado de una empresa que desarrolla sistemas software para la industria de transporte de contenedores.
- Presentar una nueva línea de investigación que pueda ser utilizada como base en un futuro para resolver problemas específicos y concretos del sector, y que también permita conocer el impacto y los beneficios que puede reportar.
- Asentar las bases de cara a un posible desarrollo de una Tesis Doctoral, relacionada tanto con el sector escogido como con el Máster cursado.

Aproximadamente un 90 por cien del comercio mundial es transportado por mar, implicando directamente que cualquier mejora operativa o sobre diferentes procesos tenga un gran impacto económico y social. Un ejemplo podría ser la capacidad de reducir la contaminación generada por los buques: Supondría un ahorro para la empresa naviera y un beneficio para toda la sociedad.

Debido a la evolución del sector y también a su naturaleza, hoy en día conviven muchas fuentes de datos y diferentes sistemas software propiedad de proveedores, empresas, terminales de contenedores o navieras, entre otros. Una parte fundamental de la cadena de suministro es el TOS o Terminal Operating System, cuyo objetivo principal es gestionar y controlar el almacenamiento y movimiento de contenedores en una terminal de transporte marítimo. Estos sistemas han evolucionado hasta el punto de que gestionan mucho más allá de los equipamientos, trabajadores o recursos disponibles.

Otro elemento software clave, resultado del proceso de automatización que se está llevando a cabo en el sector, es el ECS o Equipment Control System. Este software se encarga fundamentalmente de orquestar y dirigir todas las secuencias e instrucciones de una terminal de contenedores semiautomática o automática. Básicamente se transforman instrucciones de alto nivel en el TOS a instrucciones de bajo nivel contra un equipamiento en concreto.

A estos sistemas software de gestión o de control de equipamiento se suman otras muchas plataformas y aplicaciones de fabricantes de maquinaria, o incluso software de fabricantes de autómatas y controladores lógicos programables. Todo esto compone un ecosistema digital donde todas las piezas están desconectadas, salvo que entre ellas se requiera de una comunicación explícita o por un propósito concreto. Generalmente esta comunicación carece de estándares, no es extensible y no busca integrar sistemas, sino que se centra exclusivamente en cumplir las necesidades inmediatas del propósito para la que ha sido creada.

La integración de todas estas fuentes de datos puede repercutir en numerosos beneficios, no solo para cada uno de los actores individuales sino para el sector en general, pudiendo incrementar la eficiencia, la visibilidad y la productividad global. Este trabajo de investigación pretende construir un escenario que pueda demostrar esto, de manera que a través de un caso de uso que refleje una situación real, se simulen diferentes valores y escenarios, y se sinteticen los resultados obtenidos junto con el impacto y beneficio que podrían aportar.

Para llevarlo a cabo se comenzará desarrollando una parte teórica, que contendrá dos análisis distintos:

- La situación actual del sector y las soluciones disponibles, así como desarrollos en curso y otros planificados para el futuro.
- Las soluciones tecnológicas disponibles para agilizar la integración entre software, la publicación de API’s y la publicación de servicios y micro servicios a partir de modelos de datos locales.

A continuación se expondrá un caso de uso real concreto. Sobre este caso se realizará una implementación práctica que utilice la solución tecnológica más apropiada para alcanzar los objetivos. Esto permitirá simular un escenario a partir del cual se puedan analizar los resultados y evaluar qué mejoras o qué beneficios puede aportar al sector, desde el punto de vista de la optimización y la eficiencia continúa.

Los avances que se pueden alcanzar implican una posible reducción de sistemas asociada, la disminución de costos, la implicación por parte de los diferentes actores y los beneficios globales a través de la colaboración y la puesta en marcha de arquitecturas y estándares globales compartidos.

Respecto a la implementación, se van a emplear utilizados aquellos modelos o estrategias de desarrollo de API's que favorezcan la agilidad, la integración y la escalabilidad. Deberán permitir conectar las diferentes aplicaciones heredadas, bien sean aplicaciones SOA, bases de datos locales, ESB o similares. De esta forma el resultado del trabajo contendrá tanto una tecnología como un proceso automático y ágil. La implementación será capaz de compartir y publicar al exterior modelos de datos locales de manera estándar e independiente de la naturaleza o formato de las fuentes de datos originales consultadas.

3 ESTADO DEL ARTE

En este estado del arte se recopilan los principales proveedores de software para terminales de contenedores y transporte marítimo, de manera que de entre los siguientes ejemplos se pueda adquirir una visión global de la oferta existente y las necesidades a las que responde. Desde TOS, hasta ECS o software de simulación, la selección de empresas y sistemas se ha basado en la influencia que tienen actualmente en el sector. Existen otros programas desarrollados por empresas navieras, fabricantes de equipamiento y grúas, o incluso desarrollos internos en las propias terminales, pero carecen de la trascendencia o el impacto de los ejemplos aquí presentados.

3.1 Xvela

XVELA es un proveedor de soluciones tecnológicas para la industria del contenedor basado en la nube. Su apuesta propone utilizar un entorno en tiempo real de colaboración, y a través de los datos obtenidos desde diferentes fuentes aporta visibilidad a lo largo de todo el ciclo de rotación de un barco de transporte de contenedores. El objetivo es combinar los dos actores fundamentales en este proceso: La naviera y la terminal. La empresa esta soportada por Navis LCC, que a su vez forma parte del grupo Cargotec. El producto se compone básicamente de tres módulos fundamentales:

- Terminal Library: Contiene información esencial para la ejecución de las operaciones de la terminal de contenedores. Todo está publicado online, e incluye diferentes perfiles para acceder tanto a información estática de la terminal como a las longitudes de amarre, características del muelle, información dinámica, ocupaciones, productividad, asignación de grúas…
- Ship Library: Permite el acceso a los datos de las embarcaciones de empresas navieras, como configuraciones, cálculos de estabilidad o archivos de amarre.

Unifica las especificaciones para un mismo barco, de manera que sean idénticas tanto para la terminal como para la naviera.

- Ship Viewer: Módulo para la visualización de la configuración del barco y la distribución de los contenedores. Aporta un espacio común para visualizar los planes de estiba de los diferentes actores involucrados. Incluye distintas opciones de visualización (sección del barco, bahías, bodegas…), y permite así detectar posibles contradicciones entre los diferentes planes de estiba.

Como plataforma colaborativa, pretende mejorar la comunicación entre la naviera y la terminal, tanto para planes de estiba, tiempos de llegada, retrasos o fallos en el equipamiento de la terminal, entre otros. Aún no ha tenido un gran impacto y no ha sido adoptado por las grandes empresas del sector, pero si esto llega a suceder, generará una única fuente de datos donde poder validar, contrastar y eliminar posibles errores en los diferentes procesos de toma de decisiones.

3.2 Marine Traffic

Marine Traffic es una plataforma abierta que proporciona información en tiempo real de la localización de los barcos en puertos. Proporciona detalles de la localización, características del barco, datos sobre su fabricación y el número IMO (International Maritime Organisation), entre otros. El acceso a la plataforma es gratuito, pero para determinadas funciones avanzadas o uso específico existen servicios de pago, generalmente a través de créditos como unidad monetaria en la plataforma. Los datos se recopilan a través de miles de estaciones AIS (Automatic Identification System) ubicadas en 140 países del mundo, y son transferidos a los servidores de la empresa. Los barcos están equipados con transpondedores AIS que emiten datos recibidos por estas estaciones. Estos datos son paquetes AIS que codifican mensajes NMEA de 64 bits, y presentan el siguiente aspecto:

```
!AIVDM,1,1,,B,1INS<8@P001cnWFEdSmh00bT0000,0*38
```

Generalmente las estaciones AIS terrestres emplean una antena externa que se coloca a unos 15 metros sobre el nivel del mar, y recibe información en un rango de 15 – 20 millas náuticas. La plataforma ofrece los siguientes productos y servicios:

- MarineTraffic Website, con un mapa en tiempo real, notificaciones, gestión de flota y otras funcionalidades.
- Online services, con un set de aplicaciones accesible a través de la plataforma web, filtros avanzados, tracking personalizado, gestión de áreas…
- Mobile apps, con MarineTraffic como versión móvil de la aplicación web y Oncourse, aplicación diseñada para navegación particular y privada.
- Business directory: Directorio diseñado para reunir empresas y proveedores en torno al mundo de la navegación, con más de cuarenta mil empresas presentes.

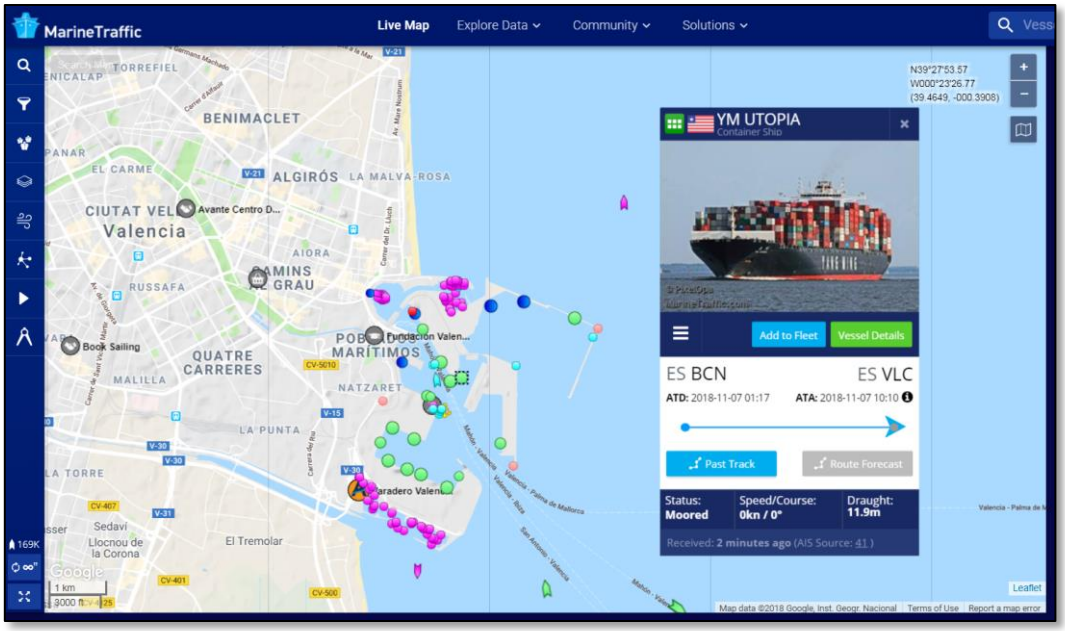

FIGURA 1 – Captura del mapa en tiempo real de MarineTraffic sobre el puerto de Valencia.
Fuente: Website de MarineTraffic, www.marinetraffic.com.

Ya en el año 2015 recibía un total de 6 millones de visitantes mensuales con 1 millón de usuarios registrados. Se conectan todo tipo de usuarios, desde particulares, empresas privadas o servicios de emergencia. Marine Traffic nació gracias a Dimitris Lekkas, un entusiasta de las telecomunicaciones, experto en informática y apasionado de la navegación, que en el año 2006 descubrió las posibilidades de AIS y comenzó su

proyecto. Actualmente está considerada como empresa líder y pionera en cuanto a seguimiento de barcos en tiempo real.

3.3 Navis N4 / XPS

Actualmente Navis es el TOS (Sistema Operativo para Terminales) líder del mercado. Es un software para la planificación y control en tiempo real de las actividades operacionales de una terminal de contenedores con una experiencia de más de 25 años. Se utiliza en más de 120 terminales marítimas, y supone una versión mejorada del mítico Navis SPARCS, a su vez instalado en más de 250 terminales de todo el mundo. Navis LLC fue adquirido por Zebra Enterprise Solutions, y posteriormente por el grupo Cargotec, propietario de la empresa finlandesa Kalmar (especializada en grúas y maquinaria para puertos y terminales de contenedores). De manera resumida, los principales módulos de optimización con los que cuenta N4 son:

- **Prime route** – Gestiona la asignación de trabajo a equipos, teniendo en cuenta la distribución de la terminal y el cálculo de las distancias más cortas. Considera todo tipo de restricciones con el fin de disminuir costes de combustible, mano de obra y mantenimiento, entre otros.
- **Auto stow** – Gestiona el tiempo de planificación de estiba en los barcos de contenedores, teniendo en cuenta condiciones técnicas, número y tipo de contenedores a manipular (BAPLIE, COPRAR…) y las restricciones operacionales habituales de una terminal.
- **Expert decking** – Módulo para la utilización óptima del patio de la terminal, atendiendo a las diferentes reglas de negocio y limitaciones físicas para albergar los contenedores.
- **Quay commander** – Gestión de las grúas con control en tiempo real de horarios, contenedores, puntos de trabajo o tareas operacionales del barco, entre otros. Es utilizado por los "*vessel planners*", quienes pueden lograr

predicciones más exactas de movimiento de contenedores e intervalos de ejecución, en su búsqueda constante de aumento de la productividad.

El sistema es compatible con terminales totalmente automatizadas, de forma que permite gestionar equipos de manipulación de contenedores autónomos. Es muy escalable, permite trabajar con múltiples terminales y ofrece un grado de personalización muy avanzado, no sólo a través de la creación de campos dinámicos o formularios propios, sino a través del lenguaje de scripting Groovy. Este permite inyectar código e interceptar diferentes puntos de cada proceso para adaptarlo completamente a las necesidades particulares. Como cualquier otro TOS, se apoya en tecnología EDI (Electronic Data Interchange) para compartir archivos con navieras u otros actores, como los listados de estiba de contenedores. Generalmente, una terminal de contenedores emplea tanto Navis N4 como SPARCS / XPS, siendo este último el software clave en cuanto a las operaciones en tiempo real. SPARCS / XPS está programado en C y lleva ya más de dos décadas en el mercado.

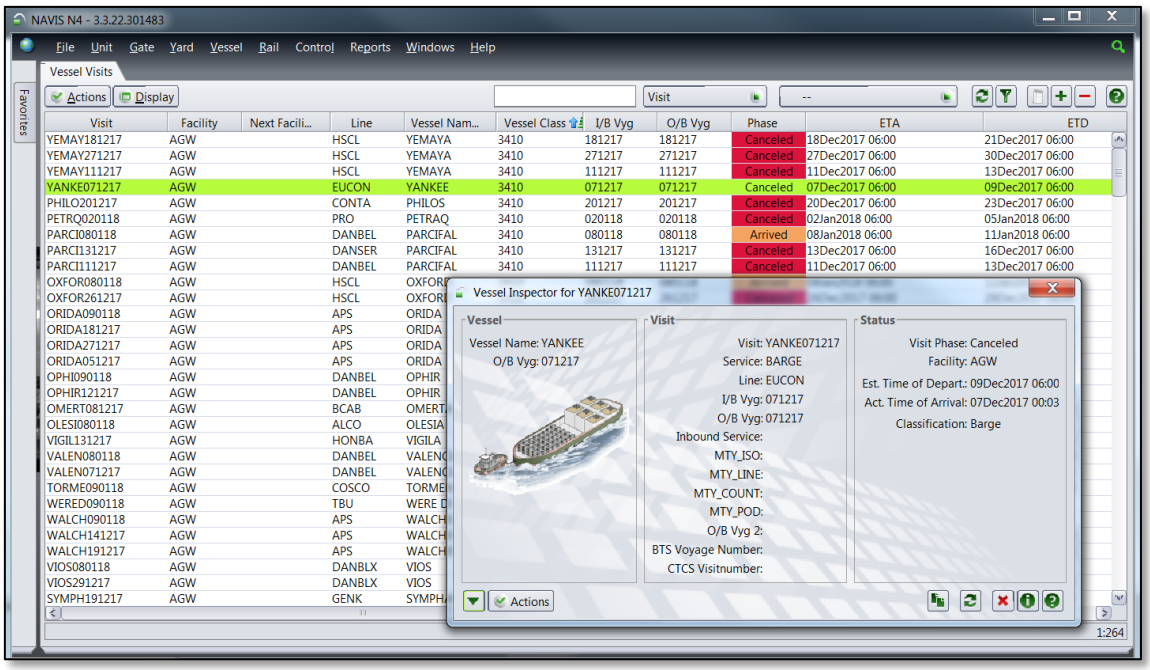

FIGURA 2 – Captura de N4, el TOS líder del mercado mundial.
Fuente: NAVIS LCC.

En cuanto a extensibilidad e interconexión, además de EDI o la capacidad de inyectar código Groovy, existe una forma muy sencilla de publicar como servicio web cualquier filtro implementado en la aplicación, para servir datos al exterior a través de una URL. Esta forma recibe el nombre de Universal Query API, y resulta muy ágil y personalizable a través de los filtros del programa. Por otro lado, dentro de cada inyección de Groovy generalmente se trabaja con JMS y ActiveMQ para enviar o recibir información del exterior, pero también es posible importar cualquier librería que funcione en Java y programar clases propias de comunicación, como pueda ser cualquier implementación de un ESB o un servidor de comunicaciones externo. Existen también desarrollos en curso para crear una nueva capa de consumo de datos externa a través de Kafka y del uso de micro servicios. Como punto fuerte de Navis N4, destacar la extensibilidad y el nivel de personalización que tiene con respecto a sus competidores.

3.4 Octopi

El TOS de Octopi nace de la mano de dos ingenieros de Software, y se presenta como una opción moderna y de vanguardia, ayudando a las diferentes terminales de contenedores a gestionar sus operaciones, monitorizar todas las mercancías y comunicarse de forma electrónica con los distintos proveedores. El primer cliente de la empresa es la terminal IMT en Puerto Príncipe, Haití, y tras la implantación la productividad en dicho puerto aumentó más de un 50%. Este éxito permitió obtener contratos para dos nuevas terminales de contenedores. La empresa está establecida en Miami, Florida, y es una solución basada en la nube ideal para terminales de tamaño pequeño – mediano. Es una de las pocas soluciones que actualmente funciona como SaaS, con las adicionales características de seguridad, disponibilidad, escalabilidad y tolerancia a fallos que un sistema de este tipo puede permitir. Además incluye un módulo de BI (Business Intelligence) y diferentes dashboards en tiempo real, con un diseño optimizado para ser utilizado en cualquier tipo de dispositivo. Incluye características y funcionalidades típicas como inventario de contenedores,

procesos personalizados para las puertas, planificación de patio, intercambio de datos mediante mensajería electrónica o gestión de equipamiento, entre otros.

Esta empresa ha sido recientemente adquirida por Navis, concretamente en el primer trimestre del año 2019. Con esto Navis pretende completar su oferta en cuanto a TOS también para pequeñas terminales, un sector que mueve aproximadamente las mismas cifras que el de grandes terminales, pero de manera más diversificada.

Uno de los puntos más destacables de Octopi es de la interfaz de usuario, con un diseño y una línea sencilla, vanguardista y de tipo responsive. De esta manera la formación a usuarios y la adopción de la herramienta resulta muy sencilla, ya que es muy intuitivo trabajar con este software. También proporciona integración con QuickBooks Online y con Microsoft Dynamics GP, para gestionar toda la parte financiera desde otros programas más especializados. En cuanto al intercambio electrónico de datos (EDI), el software procesa todos los mensajes EDIFACT típicos en una terminal de contenedores: COARRI, CODECO, VERMAS… Se trata de un TOS que todavía no se encuentra en una fase tan madura como otros, pero que para pequeñas terminales puede ser una buena apuesta, gracias a su innovadora arquitectura en la nube y su diseño multi-dispositivo.

3.5 Opus

Opus Terminal es el nombre que recibe el TOS de la empresa CyberLogitec, cuya historia se remonta al año 2000. El sistema se implanta por primera vez en el puerto de Kahosiung, en Taiwan. Desde entonces, la empresa basada en Seoul, Korea, cuenta con más de 480 empleados y cuatro certificaciones ISO y CMMI, y está presente en mercados de todo el mundo: Europa, USA, China, Singapur… En todo este periodo destacan implantaciones en terminales como Busan New Port, Hanjin New Port, Dubai Jebel Ali, US Longbeach y otras en Indonesia, Tailandia o Brasil. Los últimos reportes financieros que se publican en su página web muestran una compañía en plena expansión, con unos beneficios que crecen año tras año ya que cada vez son más los

clientes y terminales que confían sus operaciones a Cyberlogitec. De entre ellos se incluyen terminales de gran capacidad, sumando un total de 50 clientes.

Además del producto principal, ofrece otros módulos bajo el nombre de OPUS Stowage, OPUS Terminal, OPUS Container, OPUS Logistics o Eagle Eye, entre otros, los cuales son integrables entre sí dependiendo de las necesidades de cada cliente. Entre las principales características del TOS, destacar los siguientes módulos:

- AVP (Advanced Vessel Planning), planificador de barcos que combina información electrónica, de equipamientos y contenedores. Optimiza la secuencia de estiba o descarga en el barco basándose en estas fuentes de datos, y también los tiempos de distancia y remociones llevadas a cabo.
- TLS (Terminal Logistics System), como pieza clave para gestión de equipamientos y flujos de trabajo. Dispone de diferentes mecanismos para balancear el uso y la carga, desde una plataforma común tanto para las puertas, el patio, el muelle o el ferrocarril de la terminal.
- AYA (Advanced Yard Allocation), plataforma para distribuir contenedores en el patio. Emplea una la cola de trabajo para cada una de las diferentes grúas, así como una selección de filtros y estrategias predefinidas.
- Interface Standard, es el módulo que unifica todas las comunicaciones con otros sistemas como OCR para identificación de matrículas, dispositivos de radio frecuencia, PLC's…

Otros software ofrecidos por la empresa a destacar son OPUS Stowage, que gestiona el espacio a través de patrones de almacenamiento de contenedores, OPUS Container como solución para gestionar los costes o beneficios y visibilidad en las diferentes operaciones asociadas a un contenedor, Eagle Eye como monitor de procesos y equipamiento automático, u OPUS Logistics, que va un poco más allá de la terminal y ofrece información empresarial en tiempo real sobre transporte aéreo, transporte terrestre u otras modalidades combinadas.

En cuanto a extensibilidad e interconexión se refiere, destaca el módulo SmartLink, como solución que conecta a través de EDI y otros protocolos diferentes compañías

involucradas en la cadena de suministro. Permite servir recursos o recibir información relevante como si de una API se tratara, permitiendo así definir diferentes procesos e intercambiar información con terceras empresas.

3.6 Tops Advance

La empresa Realtime Business Solutions (RBS) ofrece un TOS llamado TOPS Advance. Fundada en 1991 en Australia, se ha especializado durante toda su trayectoria en el desarrollo de aplicaciones software para la industria del contenedor. Ofrece servicios de consultoría a diferentes empresas marítimas y ferroviarias relacionadas con el contenedor. Su TOS está implantado en terminales de contenedores en Australia, Brasil, Egipto, Francia, Alemania, Italia, Japón, Nueva Zelanda o Panamá, entre otras. Ofrece también diferentes soluciones que complementan la aplicación principal.

El TOS ofrece un completo sistema optimización de operaciones, que cubre todas las zonas de la terminal: Las puertas, el patio, el muelle… Incluye un sistema especial para monitorizar los contenedores Reefer, y un sistema de interconexión con dispositivos físicos mediante radio y con dispositivos GPS. También los típicos paneles de control para las diferentes áreas, herramientas de monitorización y diferentes sistemas de reportes, indicadores y KPI's. El módulo adicional TOPX Advance es un visor en tiempo real para la planificación de contenedores, que incluye geometrías de barcos y la geometría y distribución del patio. Otro módulo, TOPO Advance, está basado en tecnología .net y se conecta a un cliente SQL, y es el responsable de toda la mensajería EDI y todo el sistema de reporting del TOS.

Las siguientes características lo distinguen del resto de competidores:

- Es un sistema 100% en tiempo real. Toda la gestión de las variables está implementada de forma reactiva, de manera que cualquier interfaz se refresca automáticamente cuando se produce un cambio de dato, independientemente de que se trate de un panel de control con elementos visuales o un reporte.

- Implementación cliente servidor, compatible con sistemas operativos como Unix, Linux y Windows.
- El tiempo de respuesta, ya que la base de datos está completamente cargada en la memoria del servidor. La manera en la que se actualizan los datos es extremadamente ágil.

Otras aplicaciones completan la oferta de la empresa, como TOPS SimOne, un simulador 3D de la terminal. Permite simular las operaciones asignando, planificando y monitorizando diferentes instrucciones de trabajo. Identifica cuellos de botella o posibles congestiones durante la operativa, y al ser un simulador, puede ser lanzado tantas veces como se desee para valorar las distintas estrategias. También permite reproducir la simulación a alta velocidad, y captar errores antes de que se tenga que ejecutar la operativa real.

3.7 Tba

El grupo TBA es una empresa de software y consultoría para la industria del contenedor líder del mercado en cuanto en simulación y emulación. Actualmente desarrolla software que implementa y optimiza procesos logísticos en terminales de contenedores, así como aeropuertos, plantas de manufacturación y demás plataformas logísticas en todo el mundo. Cuenta con más de 200 empleados distribuidos fundamentalmente en Holanda, Reino Unido, Alemania y Rumanía, y también está presente en el continente Americano y en Oceanía.

El fundador y propietario de la empresa, Yvo Saanen, redactó su tesis doctoral para la Delft University of Technology (TU Delft) en Países Bajos antes de fundar la compañía. Su trabajo pone de manifiesto las posibilidades de la simulación y emulación en terminales de contenedores, y posteriormente desarrolla varias plataformas que hacen esto realidad, siendo pionero en todo el mundo. Poder simular y evaluar diferentes estrategias en terminales de contenedores automáticas se convierte en una herramienta con un gran potencial. Por otro lado el software Teams es el más popular

siendo el ECS (Equipment Control System) más extendido en el mercado de las terminales semi-automáticas y automáticas. Desde cálculo de rutas, gestión de errores o sistema de ejecución de órdenes inteligentes, Teams es el software de control de equipamiento en tiempo real con mayor aceptación.

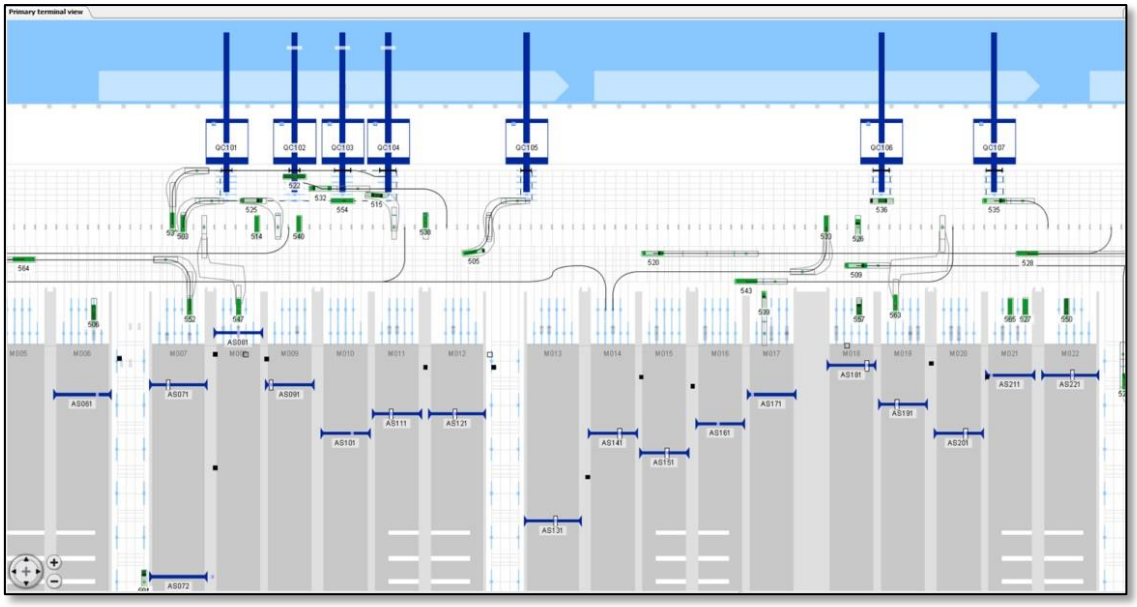

FIGURA 3 – Captura de TEAMS, el ECS líder del mercado en terminales automáticas.
Fuente: TBA Group.

Autostore TOS, WMS, CommTrac o Controls son otras aplicaciones que ofrece la empresa. El TOS de TBA está orientado a terminales que manejan pequeños volúmenes. No obstante la empresa desarrolla más sistemas accesorios y ofrece consultorías, integración, implantación y mejoras con cualquier otro TOS, lo que les sitúa en una posición muy aventajada y con mucho conocimiento sobre lo que es un Terminal Operating System. En concreto Teams realiza una interacción muy completa con Navis N4 u otros TOS del mercado para hacer realidad una terminal automática: El sistema operativo ejecuta ordenes e instrucciones de trabajo a más alto nivel que son transmitidas al ECS, de manera que este las transforma en instrucciones a más bajo nivel y órdenes para el equipamiento automatizado y sus dispositivos lógicos programables.

TBA es una empresa muy bien considerada y posicionada en el sector, y gracias a la simulación y emulación han conseguido tener una marca muy bien valorada en el mundo de las terminales de contenedores.

3.8 Tideworks

Tideworks Technology ofrece software para terminales de contenedores y soluciones gráficas para planificar operaciones de terminales marítimas e intermodales. La empresa fue fundada en 1999 después de 40 años acumulados de experiencia como la división tecnológica de Carrix Inc, que opera en más terminales de carga que ninguna otra empresa. Actualmente más de trescientos mil profesionales del sector logístico emplean su software para cargar mercancías en buques o trenes, rastrear contenedores o realizar pagos en unas cien terminales distribuidas por todo el mundo. Las oficinas centrales se ubican en Seattle, Washington, pero dispone de oficinas en todo el mundo. Sus soluciones se caracterizan por tener un elevado grado de adaptabilidad, y principalmente destacan:

- Mainsail Vanguard: Es el nombre que recibe su TOS. Se caracteriza por su integración con sistemas de terceros, gran variedad de informes y analíticas, interfaz de usuario personalizable y un completo control de inventario que incluye elementos como chasis y mercancías peligrosas.
- Spinnaker: Herramienta gráfica de planificación tanto para barcos, patio o trenes. Permite diseñar secuencias de carga y almacenamiento, incluyendo un análisis de eficiencia de las mismas. También incluye un módulo de workflow y definición visual de procesos.
- Intermodal Pro: TOS intermodal, con herramientas de planificación en tiempo real, integración con DGPS, dispositivos móviles y diferentes equipamientos software.
- Traffic Control: Herramienta para gestionar el tráfico en tiempo real entre sistemas y mercancías, a través de una interfaz totalmente configurable.

- Terminal View: Visor 3D de la terminal, que proporciona visibilidad a toda la operativa desde equipamientos a movimientos o inventario.
- Forecast: Portal web que pretende unificar comunicaciones entre navieras, inversores, empresas de transporte y otros actores implicados.
- Tideworks Insight: Herramienta de Business Intelligence, que se combina con otra herramienta ETL y un sistema de informes en tiempo real.

Estos programas son totalmente integrables entre ellos, y la empresa también ofrece servicios de implementación y consultoría de TOS, training, servicios de soporte o servicios relacionados con EDI, entre otros.

3.9 Tsb

TSB, iniciales de Total Soft Bank Limited, es una empresa coreana que desde 1988 se dedica a realizar software y todo tipo de automatizaciones para el sector logístico y la comunidad portuaria. Ofrece soluciones integradas que abarcan desde el envío o la logística hasta el transporte en buques o recepción de mercancías, y actualmente están presentes en más de 80 terminales y 500 embarcaciones.

Su principal producto es un TOS llamado CATOS (Computer Automated Terminal Operating System), destacable por su facilidad de uso, interoperabilidad, escalabilidad y flexibilidad. Gestiona todos los procesos de trabajo y actividades que tienen lugar en una terminal de contenedores. Es conocido mundialmente y especialmente utilizado en terminales de contenedores en España. Se distribuye de forma modular, con diferentes bloques para gestionar la planificación, la operativa y la administración de la terminal. Introduce el concepto Plan-Do-See (PDS), de forma que todos los procesos se organizan en tres etapas: Planificación, ejecución y análisis. De esta forma se simplifican y estandarizan todos los procesos de trabajo, unificándolos en una misma base de datos que garantiza la integridad. También incluye algoritmos de optimización, funciones operativas en tiempo real y métodos de comunicación interactivos entre los diferentes perfiles o usuarios.

CATOS presenta diferentes módulos que componen todo el sistema. Incluye tanto planificación de patio como de amarres, recursos humanos, gestión de equipos, planificación ferroviaria, gestión de turnos, gestión de puertas, sistema EDI para intercambio de información, gestión de contenedores refrigerados, facturación e informes, entre otros. También incorpora una interfaz web que permite a los diferentes perfiles administrar la información en tiempo real relacionada con las operaciones de la terminal y la toma de decisiones. Como principales particularidades, se puede destacar:

- Planificación Auto Ship – Asignación automática de horarios y carga de trabajo para cada una de las grúas.
- Doble ciclo – Gestión automática y optimizada para minimizar viajes de retorno vacías tanto de grúas de muelle como camiones de patio.
- EQ Pooling – Enlace de camiones de patio con grúas de muelle, a través del cálculo de distancias.
- Pre-notificación: Sistema de gestión de puertas con interfaz web orientado a empresas de transporte y ferrocarriles, que permite realizar reservas por anticipado antes de visitar la terminal.

Otros desarrollos de la empresa TSB totalmente integrados con su TOS son:

- CASP - Herramienta para la estiba de contenedores en un buque con una potente interfaz gráfica de usuario.
- PLUS - Sistema de gestión de recursos portuarios unificado.
- MOST – Sistema de operación multiusos para terminales con interfaz web que gestiona granel, líquidos u otras cargas especiales.
- TPES / TPSS – Sistema de emulación y simulación de puertos, que reproduce una terminal de contenedores real y permite parametrizar y evaluar el rendimiento del TOS.

4 TENDENCIAS EN EL SECTOR

La industria de software para transporte marítimo de contenedores se encuentra en un momento de transformación. La automatización de terminales de contenedores no acaba de aportar los resultados esperados, quedando muy atrás con respecto a otros sectores como es el automovilístico. Sólo son seis las terminales totalmente automáticas en el mundo, y sus cifras de productividad mejoran levemente día tras día. Llegado el momento, nadie duda de que este fenómeno de automatización total se expanda por todo el globo, pero no es algo que vaya a suceder a corto plazo según apuntan expertos del sector.

Por otro lado, cada vez es más complejo unificar los diferentes sistemas en uno, existiendo cada vez más desarrollos internos y a medida. Un ejemplo son las compañías navieras, que actualmente son reacias a adquirir un software externo si pueden desarrollar uno propio. En general, el software “in-house” es más y más frecuente.

A esto se añade el recelo existente a la hora de hablar de la propiedad de los datos, que junto con otras cuestiones de seguridad o formas de pensar de un sector menos vanguardista, está penalizando el salto a la nube que en muchos otros sectores sería a día de hoy incuestionable. Está presente la amenaza de que empresas gigantes como Amazon o Allibaba puedan implantar un nuevo sistema de transporte o un software que unifique toda la gestión en el proceso, transformando así por completo la cadena de suministro.

Esto crea una oportunidad única para la intercomunicación y conexión entre diferentes sistemas, y también para la estandarización como principal estandarte. En este capítulo se resumen los prototipos, iniciativas y conceptos que, desde un punto de vista innovador, podrían cambiar el software actual, definir la forma en la que va a evolucionar el mismo y crear nuevas oportunidades de negocio.

4.1 Container tracking

La terminal a través del TOS dispone de una base de datos con todos los contenedores en el patio. Cuando un contenedor empieza a ser manipulado por un equipamiento automático, el ECS contiene la referencia del contenedor que está transportando. Cuando un camionero transporta un contenedor, el sistema de citas para las puertas de la terminal también conoce esta referencia, al igual que el sistema OCR que probablemente identificará este contenedor en las puertas de la terminal, o el sistema OCR que identificará el contenedor en la grúa que lo cargará en el buque. La naviera, también dispone de la información de los contenedores que transporta. Cabe preguntarse pues en cuantas bases de datos está guardado un código de contenedor y por cuantos sistemas software distintos y a través de cuantos estados pasa. En el transporte marítimo nunca resulta sencillo conectar todos los puntos. Seguir la ruta de un contendor en este ecosistema resulta complejo, por lo que una nueva tendencia consiste realizar un seguimiento de cada uno de los contenedores existentes para poder conocer en cada momento donde está situado.

La empresa Ocean Insights, fundada en Hong Kong en 2012 y con oficinas también en India y Alemania, desarrolla software que permite conectar información del transportista con el paradero real de la embarcación, aportando información en tiempo real que permite tomar decisiones importantes en la cadena logística de suministro. Consolidan y evalúan datos desde múltiples fuentes para ayudar a los distintos equipos de logística a unificar la información y trabajar con los mismos datos, ya sea para operaciones diarias como para decisiones estratégicas. El innovador sistema Container Track & Trace de Ocean Insights conecta la información del transportista con el paradero real de la embarcación, ofreciendo información en tiempo real que permite tomar medidas antes de que pueda surgir algún contratiempo. Gracias al rastreo de embarcaciones por satélite, el software vigila constantemente la flota mercante y todos los contenedores que están en movimiento. Con este punto de referencia, se monitorizan todos los cambios en los patrones comerciales y se comunican las advertencias de demora para poder actuar antes de

que algo vaya mal. A esto se añade el factor humano, ya que un equipo de analistas asegura constantemente la calidad de los datos para que estos sean fiables.

Existen páginas web como Marfret o Track-Trace que, consultando diferentes fuentes, API's y bases de datos, permiten conocer la ubicación de un contenedor, las características y la ruta que ha seguido. Eso sí, dependiendo de cada web ofrecerá compatibilidad sólo con los contenedores de algunas compañías.

Date	Position	Vessel	Voyage	Location
04/06/2018	Out To Shipper	FLEUR N	18050029	LYON - NAVILAND
08/06/2018	Delivered Full on Terminal	FLEUR N	18050029	MARSEILLES - P.157
18/06/2018	Loaded Full	FLEUR N	18050029	MARSEILLES - P.157
28/06/2018	Discharged Full	FLEUR N	18050029	POINTE A PITRE - TER GPPTP
04/07/2018	Out To Consignee	FLEUR N	18050029	POINTE A PITRE - TER GPPTP
06/07/2018	Delivered Empty (Terminal or Depot)	FLEUR N	18050029	POINTE A PITRE - TER GPPTP

FIGURA 4 – Captura de la web de MARFRET sobre el tracking de un contenedor.
Fuente: www.marfret.com.

Otra empresa que está apostando fuerte en cuanto al seguimiento de contenedores es Orbcomm. Pretende transformar la cadena de suministro con el uso de M2M (Machine to Machine) y IoT (Internet of Things). Básicamente M2M es el proceso a través del cual una máquina se comunica con otra mediante un canal de comunicación, incluyendo dispositivos y sensores telemétricos, redes, conectividad de datos y aplicaciones software móviles y en la nube. A través de dispositivos físicos acoplados a contenedores y equipamientos de transporte intermodal, se puede obtener un seguimiento preciso e ininterrumpido en áreas remotas donde la comunicación inalámbrica no es posible. Con seguimiento GPS y usando georreferencia, es posible ubicar todos los contenedores que están parados en puertos o depósitos y todos los

que están en movimiento tanto por tierra o por mar. Como ejemplo de aplicación real, la naviera Maersk desplegó en el año 2015 esta tecnología M2M en 270.000 unidades, siendo Orbcomm una de las empresas pioneras para este prototipo.

Por último Loginno, de origen Israelí, es una startup que se ha especializado en instalar dispositivos para trackear mercancías, no sólo en la parte de software sino también en la parte de hardware. Ha desarrollado varios proyectos aplicados a contenedores, y actualmente ofrece la aplicación OnSchedule Monitoring Edition, una solución de tipo llave en mano que permite realizar un seguimiento de las mercancías tanto por tierra, mar o aire. El software está basado en una interfaz web, y dispone de un sistema de gestión de alarmas ante cualquier irregularidad que pueda ocurrir durante el transporte. También ofrece una API que permite consumir la información desde aplicaciones u otros sistemas externos.

4.2 Live API's

Con tanta cantidad de empresas y actores en el ecosistema, es evidente la necesidad de compartir datos y comunicar los diferentes sistemas entre sí, y aportar mayor dinamismo y transparencia entre los diferentes procesos. La empresa belga NxtPort es pionera a través del desarrollo de una plataforma que permite transferencias de datos rápidas, rentables y eficientes entre los distintos sistemas, con la intención de aumentar la eficiencia operativa, la seguridad y los ingresos. Permite que los datos se compartan de una forma más eficiente, combinando y contrastando los datos existentes, y reforzado así la confianza en los mismos. El proceso se compone de tres pilares esenciales: Compartir datos, usar los datos y construir aplicaciones combinando los mismos. El proyecto nace primeramente en el puerto de Amberes, y actualmente la empresa ofrece una de las API´s más completas del mercado basándose en estos tres principios, con 16 categorías y un sinfín de servicios y llamadas posibles para obtener todo tipo de datos. De entre todas se puede destacar:

- Estancia del buque: A través del número IMO de cada barco, se sirve toda la información relacionada con la estancia de un buque en un puerto. Emplea una plataforma de comunicación digital del puerto de Amberes llamada C-Point.
- Consulta de peso: Permite consultar el peso de los contenedores y la tara, proporcionando esta información a los diferentes agentes de envío, transportistas o personas interesadas.
- Localización Tresco: Traduce las coordenadas (latitud y longitud) a lo largo de la ruta de navegación interior por Europa a lugares significativos, de forma que sean interpretables. Se basa en diferentes mapas de navegación interior y cubre todo el espacio europeo navegable.
- Seguridad para contenedores: Permite transferir digitalmente los permisos para recoger un contenedor. Emplea mensajes COREOR y tecnología blockchain para dotar de mayor seguridad al proceso. Se genera un código PIN que el transportista podrá emplear para recoger la mercancía.
- Import / Export: Una serie de llamadas que facilitan la importación y exportación de mercancía en el puerto. Para la importación, permite recibir y enviar a la aduana los mensajes CUSCAR, a través del uso de una clave que se compone del número BL y el número de contenedor. Para la exportación, sirve automáticamente mensajes de manifiesto previo a que el barco salga del puerto, así como los mensajes de confirmación de carga de la terminal (COARRI) y el estado de la mercancía en la aduana.

MarineTraffic ofrece también una API completa e innovadora, accesible a través de la compra de créditos. Así como NxtPort se encuentra en una fase de desarrollo a nivel local, MarineTraffic dispone de una base de datos con información a nivel global. Se organiza en las siguientes categorías:

- Posicionamiento de embarcaciones: Recorridos históricos de buques, posicionamiento para buques estáticos o en movimiento, posicionamiento dentro de puertos y áreas predefinidas, y gestión de áreas personalizadas.

- Eventos: Llamadas al puerto, con información sobre salida y llegada de las embarcaciones. Eventos de los buques, e información sobre la salida y llegada al muelle de cada barco.
- Datos de embarcaciones: Fotografías de las embarcaciones, detalles como tipo, dimensiones, propiedades y todo tipo de información técnica.
- Datos de viaje: Previsiones, llegadas esperadas, distancias y rutas portuarias, predicción de destinos y datos de navegación, información sobre congestión de puertos y tiempos de espera.
- Gestión de la flota: Gestión de la flota de barcos configurada en la cuenta de MarineTraffic, así como datos de usuario, créditos disponibles, consumo y otras estadísticas de uso de la API.

Más allá de los datos que sirve cada una de las API, lo realmente innovador es la apuesta por interconectar diferentes sistemas y servir los datos de una manera estándar y abierta, de manera que estén accesibles para toda la comunidad y permitan realizar una explotación de los mismos acorde a sus estrategias e intereses. Si todas las soluciones y software que se emplean sirviesen los datos de la misma forma, permitirían obtener una visión más holística de todo el proceso y tomar decisiones de una forma más eficaz.

4.3 Estandarización y optimización

Como ejemplo de la necesidad existente de estandarizar sistemas software y optimizar procesos, son muchos los puertos que han emprendido proyectos entre las terminales, navieras y empresas que operan en él. Dentro del ecosistema portuario se encuentran sistemas que están almacenando los mismos datos, bajo nombres diferentes y desde diferentes fuentes, y no hay un consenso ni un modelo de datos centralizado al cual recurrir. Precisamente son proyectos impulsados por las autoridades portuarias, como entidad y eje central en cualquier puerto del mundo.

Un primer ejemplo es el prototipo desarrollado en la bahía de Algeciras, financiado por su autoridad portuaria, y bajo el nombre de Pit Stop Port Operations. Han participado multitud de empresas, navieras y terminales que operan en este puerto, y en este piloto se ha explorado tanto la colaboración como la toma de decisiones con respecto a los procesos operativos relacionados con las llamadas a puertos (port call), que son llevados a cabo por parte de los diferentes buques que operan en la bahía. El proyecto aplica un enfoque de extremo a extremo, de principio a fin, que busca estandarizar los diferentes eventos así como optimizar las operaciones de entrada y salida, y los diferentes procesos de carga y descarga de los buques de contenedores.

Según un estudio realizado por la naviera Maersk Line, un buque de contenedores está un 38% del tiempo de su estancia en el puerto realizando maniobras de entrada y salida, acoplamiento, desacoplamiento y espera de servicios portuarios, mientras que el tiempo restante se invierte en procesos de estiba y descarga de mercancías. Tal y como se demuestra en el prototipo, los tiempos de espera que generalmente tienen lugar en la entrada y salida del buque pueden reducirse considerablemente si existe una mayor trasparencia en la información e intercambio de datos entre los diferentes actores, así como las soluciones que intervienen en los procesos. Este concepto, conocido como toma de decisiones colaborativa, ha tenido ya un gran éxito y acogida en otros sectores como por ejemplo en aeropuertos y navegación aérea. Para la industria portuaria, el nombre común para esta idea es Port Collaborative Decision Making (PortCDM), y se basa en optimizar los tiempos de llegada estimados de un buque, optimizar la gestión del atraque del barco, el proceso portuario y la gestión de las visitas. De esta manera se alcanza una mayor visibilidad en tiempo real, permitiendo evaluar distintas fiabilidades de entre las fuentes de datos.

Para llevar un proyecto de este tipo a cabo es necesario analizar exhaustivamente tanto los procesos como las aplicaciones informáticas que los respaldan, y estandarizar eventos conjuntamente con los diferentes grupos de optimización del puerto. Se incluye un análisis de procesos operacionales de buques portacontenedores, un análisis funcional de sistemas informáticos, análisis de sinergia con otros proyectos

similares en otros puertos, y el informe de desarrollo del concepto de optimización puerto a puerto. El éxito fue demostrar los beneficios a todas las partes implicadas que se pueden obtener si se incrementa el intercambio de datos entre todas ellas. La colaboración y el intercambio de datos significativos puede producir mejoras significativas en la eficiencia, este es el concepto de puerto colaborativo para la toma de decisiones.

Relacionado con el mismo concepto de Port Call Optimisation, la autoridad portuaria de Rotterdam ha desarrollado una plataforma llamada *Pronto*. El puerto de Rotterdam recibe actualmente la llamada de unos 30 mil buques al año, es el mayor puerto de Europa y un referente mundial en cuanto a automatización se refiere. El objetivo es planificar de manera más eficiente las llamadas al puerto, y todas las actividades y operaciones que deben llevarse a cabo. La aplicación es accesible por agentes, empresas navieras, terminales y otros proveedores de servicios, y monitoriza todas las actividades a través de un intercambio de datos estandarizado. Durante el año 2018 el prototipo fue probado por primera vez, y en 2019 ya es accesible por todos los usuarios interesados contratando una tarifa de datos. A partir de ahí se obtiene acceso a la plataforma compartida, en la que se puede intercambiar toda la información relacionada con una llamada al puerto. Es posible usar el software a través del panel de control o también a través de la API. En el momento que se conoce el tiempo estimado de llegada del buque (ETA), se asigna una línea temporal. En esta línea se muestran todos los eventos que tendrán lugar durante la llamada al puerto: Desde la llegada, hasta la estancia y salida del puerto.

Pronto utiliza bases de datos y API públicas, datos de las empresas participantes y pronósticos generados a partir de dichos datos, y con ello se intenta estandarizar y mejorar toda la información. Sin embargo, la carga del barco queda fuera del alcance de esta plataforma. Los usuarios disponen de una gran cantidad de filtros así como diferentes opciones para navegar sobre la línea temporal del buque en su visita al puerto. Esto permite planificar las diferentes actividades de forma más eficiente. Tanto el progreso como el estado de los eventos se actualizan continuamente, para que la

monitorización del estado sea fiel a la realidad. También es posible configurar un sistema de alertas en caso de cambios, retrasos o conflictos en la planificación. Los eventos empleados son resultado de un proceso de estandarización en el que han participado diversos puertos y navieras.

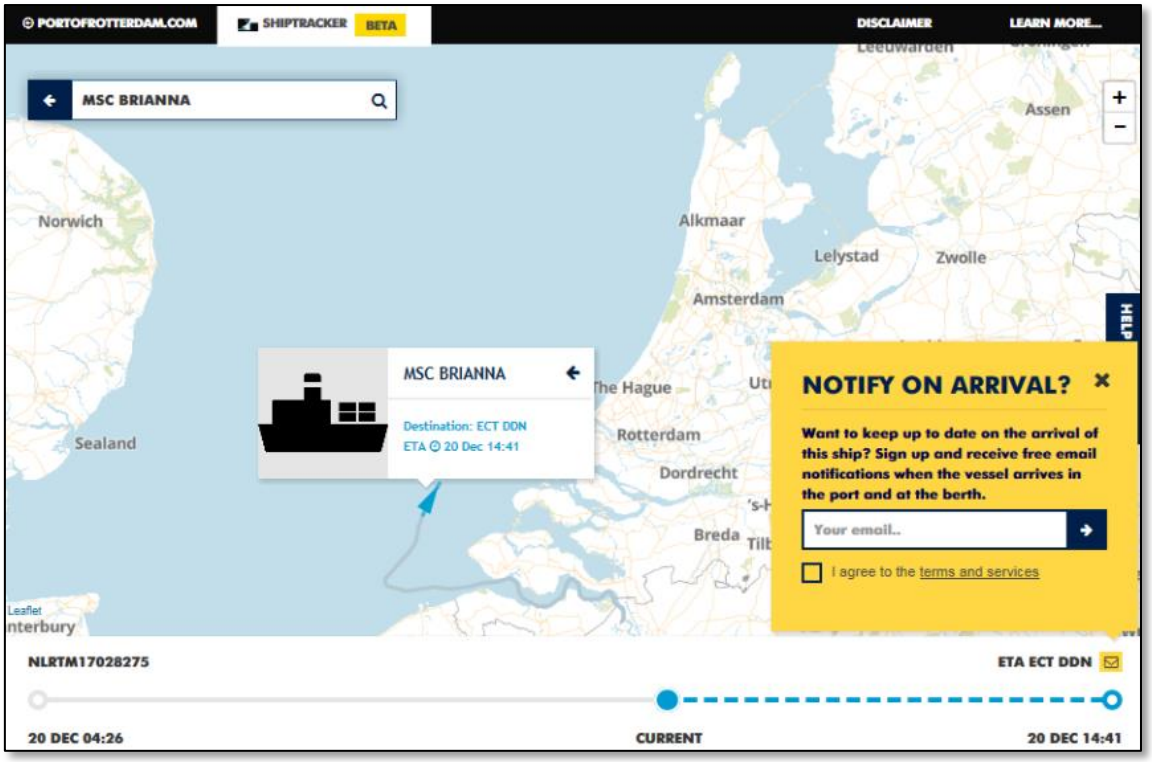

FIGURA 5 – Captura de la interfaz de usuario de PRONTO, en su versión Beta.
Fuente: Port of Rotterdam.

Esta plataforma unificada permite a todas las terminales del puerto utilizar mejor su capacidad, mejorar los tiempos de respuesta y reducir los tiempos de espera. Al mismo tiempo, para la naviera supone también un mejor tiempo de respuesta, una mejora considerable en la previsibilidad y disminuir tanto emisiones de CO2 como los costes de atraque y los costes de los remolcadores (bunker y charter). Por otro lado las diferentes agencias van a disponer de más tiempo para mejorar sus servicios a los clientes gracias a una comunicación transparente y simplificada, evitando intercambio de emails, llamadas telefónicas u otros mecanismos de comunicación para actualizar la

información. Los proveedores de servicios marítimos pueden mejorar el servicio prestado gracias a una mayor previsibilidad, y la autoridad portuaria puede conocer mucho mejor sus volúmenes de carga y sus estadísticas anuales.

4.4 Validación remota de datos

En muchas ocasiones los datos empleados por diferentes sistemas software en la cadena de suministro no son precisos y no reflejan la realidad. La mala calidad de los datos puede suponer un problema a la hora de planificar y ejecutar diferentes operaciones en una terminal de contenedores, impactando la productividad y la eficiencia de las mismas. En cuanto a los contenedores respecta, y sobre todo ante cualquier incidencia ocurrida, resulta tediosa la búsqueda de responsabilidades entre las diferentes empresas y agentes implicados en el transporte y manipulación. La International Maritime Organization (IMO) establece una serie de regulaciones en materia de seguridad bajo las siglas SOLAS (Safety Of Life At Sea convention), como por ejemplo, la segunda regulación del capítulo VI indica que es responsabilidad del transportista que todos los contenedores que se cargan en un barco estén previamente pesados. Pese a estas regulaciones, en la práctica no resulta sencillo responder a preguntas como quién debe pesar el contenedor o quién garantiza que las dimensiones proporcionadas son las correctas. Todo esto abre paso a una nueva línea de desarrollo, al permitir que los diferentes programas se comuniquen entre ellos y permitan verificar los datos con fuentes más fiables, para mejorar la calidad de los mismos.

La start-up ContainerWeight de Rotterdam lleva varios años desarrollando una plataforma en la que recopilar todos los datos y características físicas de todos los contenedores existentes en el mundo. Para ello emplea sistemas OCR y sistemas de visión instalados en diferentes terminales de contenedores. En el año 2018 lanzó la primera base de datos accesible externamente con dimensiones, taras y pesos del 99% de contenedores existentes. A través de una API y la compra de créditos para el uso de

la misma, permiten consultar todas estas propiedades y ayudar a las empresas a cumplir con las legislaciones existentes. El sistema es muy sencillo de emplear y permite integrar y recibir pesos de tara automáticamente. Pero el verdadero valor es la confianza que genera sobre los datos, al ofrecer el peso y las dimensiones correctas.

Las consultas automatizadas a esta librería pueden evitar realizar procesos manuales e incluso revisar las puertas de los contenedores. Pero la empresa también ofrece el API a diferentes sistemas software en la cadena de suministro, como pueda ser un TOS. Los datos físicos son validados a través del número específico de contenedor, y pueden ser recuperados por cada software directamente, mejorando la calidad de los datos y garantizando una mejor ejecución. Abre también la puerta a la creación de diferentes estándares o certificados de validación, en la que cada TOS o programa de la cadena de suministro valide a través de diferentes procesos las características del contenedor antes de ejecutar las operaciones.

4.5 Otras tendencias en la cadena de suministro

Las terminales de contenedores son actualmente una pieza clave en la cadena de suministro. Cuando se publica algún artículo sobre posibles escenarios futuros, se muestran simulaciones donde un cliente final pide un artículo desde su hogar en internet, y al instante, a través de un sistema de tuberías de reparto, drones o cualquier otra tecnología el artículo es entregado en el domicilio. La empresa finlandesa Kalmar, fabricante de grúas y maquinaria pesada para terminales marítimas, muestra una visión similar a través de un video promocional llamado robotAIsation. Lo que antes parecía inalcanzable para el consumidor o el profesional de la cadena de suministro, no queda tan lejos, ya que por ejemplo Amazon ofrece entregas de una hora en miles de productos para ciertas ciudades norteamericanas. El éxito de una empresa o de su cadena de suministro está cada vez más determinado por la capacidad de la misma para reaccionar rápidamente a las tendencias del mercado, así como su capacidad para tomar decisiones inteligentes basadas en datos concretos,

según indica Forrest Burnson en un reciente artículo para la revista Software Advice. Se abre la puerta a un negocio de tipo algorítmico, donde la conectividad y la disponibilidad de grandes volúmenes de datos jugarán un papel fundamental.

En otro informe titulado "Supply Chain 2025: Planning Today for Tomorrow", de Michal Burkett y John Johnson, la consultora Gartner recomienda a empresas abarcar la tecnología y el análisis que impulsara el negocio algorítmico, así como adquirir las capacidades que permitan aplicar el conocimiento encapsulado en algoritmos que permitan obtener respuestas apropiadas manejando grandes volúmenes de datos.

Actualmente nos encontramos en una época en la que las terminales marítimas, navieras y empresas implicadas en el transporte de contenedores se han obsesionado con recopilar grandes cantidades de datos, sobre movimientos, procesos, clientes… Tan importante es recopilar estos datos como formular las preguntas adecuadas, añadiendo el reto que supone consultar diferentes fuentes por la falta de conectividad. Es necesario invertir tanto en análisis de datos como en integración y estandarización de sistemas software.

Como se mencionó en el ejemplo del tracking de contenedores, el Internet de las cosas (IoT) se va abriendo paso en la cadena de suministro. Se está usando para gestión de flotas y de mercancías, para monitorizar consumos de combustible, mantenimiento de equipos, comportamientos de conducción… Según la consultora Gartner este aumento dramático en la conectividad tendrá implicaciones importantes para las empresas y cadenas de suministro que las respalden, ya que van a proporcionar mucho valor. IoT va a permitir monitorizar la entrega de mercancías y detectar posibles problemas de suministro mientras están en tránsito. Se podrá mejorar la trazabilidad de la mercancía y proteger las posibles pérdidas.

La automatización y la auto-conducción avanzan lentamente y tendrán también un fuerte impacto en las cadenas de suministro. Los robots de los almacenes automatizados de Amazon ya son una realidad que será accesible en un futuro a pequeñas y medianas empresas. Los proveedores de software especializados en sistemas de gestión de almacenes van a tener que realizar grandes avances en cuanto

a integración con otros sistemas, al igual que sucede con proveedores de software para terminales de contenedores automatizadas. Mecalux es una empresa pionera en almacenes automáticos, con una potente división de software que integra sus sistemas de gestión de almacén con aplicaciones e-commerce, aplicaciones CRM, software de monitorización de almacén o sistemas ERP entre otros.

Según Forrest Burnson los puntos clave son reconocer cuales son las tendencias en la cadena de suministro y considerar cuales se pueden implementar, invertir en infraestructura de análisis a medida que la cadena de suministro se va automatizando, y encontrar soluciones de automatización para ser más eficiente y rentable. En el mismo artículo, el autor identifica algunos puntos clave a través de los cuales señala a Amazon como principal empresa que está cambiando el futuro de la cadena de suministro:

1. Cuentas para clientes “prime” y compromisos de entrega casi inmediatos en ciertos lugares geográficos.
2. Investigación en la división de drones para reparto automático.
3. Domótica y sistemas domésticos para pedir ciertos productos desde el hogar a medida que estos se agotan.
4. Inversión en sistemas de mensajería internos más preparados que los actuales para realizar entregas inmediatas.

Pese a que el futuro está siempre abierto a cualquier cambio o impacto, estas tendencias actuales pueden aportar algunas ideas de hacia dónde puede evolucionar la cadena de suministro. También pueden justificar por qué existe cierto temor a estas grandes empresas en la industria del contenedor, que pueden resultar transformadoras.

5 TECNOLOGÍAS DISPONIBLES

En este capítulo se presentan las tecnologías actuales que mejor representan la evolución de las arquitecturas orientadas a servicios. En Junio de 2017, Mike Barlow, periodista y consultor de estrategia en telecomunicaciones, publica un artículo interno en Hewlett Packard en el que defiende la idea de por qué las empresas deberían tener en cuenta SOA otra vez. Afirma que tanto desarrolladores de software como arquitectos están redescubriendo diez años después el valor de la arquitectura orientada a servicios, como patrón a seguir para el desarrollo modular y las estrategias de implementación.

SOA revolucionó el enfoque en desarrollos de sistemas IT a gran escala, y cambio la forma en la que actualmente se percibe el desarrollo de software. Un diseño centrado en la arquitectura permite añadir y dar importancia a lo que sucede en un sistema software hacia el exterior del mismo. Los principios de SOA se están aplicando a iniciativas de desarrollo moderno como microservicios, virtualización o arquitecturas abiertas, entre otros. A la perspectiva clásica de objetos y componentes se añaden servicios web y las API's, y requiere que tanto desarrolladores como arquitectos cambien el enfoque inicial.

Según Mark Richards, arquitecto software y autor del libro Microservices VS SOA, "SOA trata principalmente de integrar plataformas y sistemas heterogéneos en toda la empresa, abstraer el sistema e intercambiar todo tipo de servicios". Tanto microservicios como SOA se basan en una arquitectura distribuida más escalable y flexible que las arquitecturas monolíticas tradicionales. A demás, la modularidad mejora la agilidad, la capacidad de prueba, la capacidad de implementación y la escalabilidad de un sistema. Actualmente la relación entre microservicios y SOA está abierta a debate, con muchas similitudes y aspectos en común.

Mike D. Kail, experto en seguridad de la empresa Cybric, apunta lo siguiente: "Veo la arquitectura de microservicios como SOA 2.0. Tomemos las lecciones aprendidas y

apliquémoslas a un nuevo patrón arquitectónico". Esta nueva arquitectura puede ser considerada como una instancia o implementación particular de SOA, donde la aplicación queda dividida en varios microservicios que son independientes entre sí. Sea cual sea la relación entre ambos, es evidente es que las arquitecturas basadas en servicios están aquí para quedarse, y SOA supuso un paso crucial en la evolución de las arquitecturas software modernas: Permitió acelerar la comunicación entre sistemas y mostró al mundo el valor de los componentes de código reutilizables.

Martin Fowler, colaborador de la revista digital Quora, afirma que "es difícil determinar el futuro de la arquitectura orientada a servicios por no estar bien definida". Según su experiencia, puede tomar uno de los dos significados:

- Un patrón arquitectónico empleado para construir aplicaciones que emplean servicios independientes y desacoplados.
- Un software que emplea SOAP, XML y el conjunto de estándares de los servicios web para promover la interoperabilidad.

Existen toda una serie de principios orientados al servicio: Contratos de servicios estandarizados, servicios de acoplamiento, servicios de abstracción, reutilización, interoperabilidad… Estos principios siguen a la orden del día a través de microservicios. Si bien es cierto que muchas implementaciones SOA tradicionales fracasaron, la característica diferenciadora del microservicio es su desacoplamiento extremo, que puede aportar una capacidad empresarial bien definida. Facilita la actualización de sistemas y mejora la escalabilidad y simplicidad.

No obstante existen diferencias entre una arquitectura basada en microservicios (MSA) y una arquitectura basada en servicios (SOA). Ambas introducen el servicio como el componente principal, y SOA define cuatro tipos básicos de servicios:

- **Business services**: Servicios que definen el núcleo de las operaciones del negocio, normalmente representados a través de XML, WSDL o BPEL.
- **Enterprise services**: Implementan la funcionalidad definida por los anteriores.

- **Application services**: Servicios específicos vinculados a un contexto específico, que se pueden invocar a través de una interfaz de usuario.
- **Infrastructure services**: Implementan tareas no funcionales, como auditoria, autenticación, seguridad o registro.

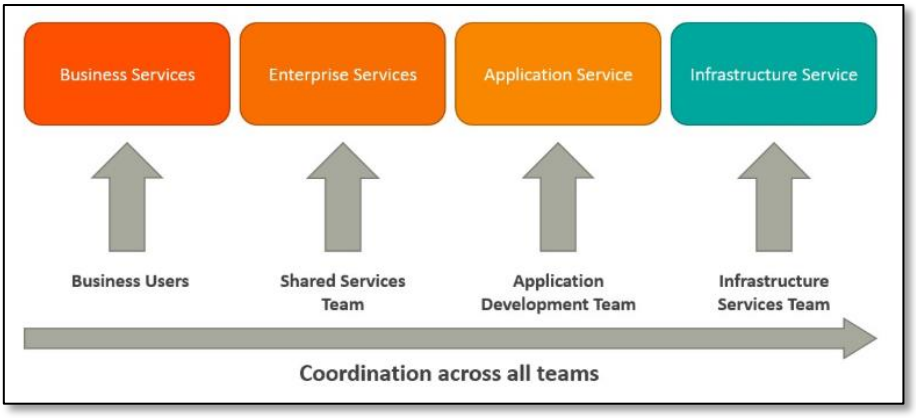

FIGURA 6 – Tipos de servicio según la arquitectura orientada a servicios SOA.
Fuente: BMC Blogs, diferencia entre SOA y MSA.

La taxonomía para la arquitectura basada en microservicios es más limitada, e introduce la idea de que un servicio más global será accedido de forma externa y generalmente no va a ser compartido con ningún otro servicio. Lo que si comparte son los servicios internos de infraestructura, que no están expuestos al exterior y sólo son accesibles de manera interna.

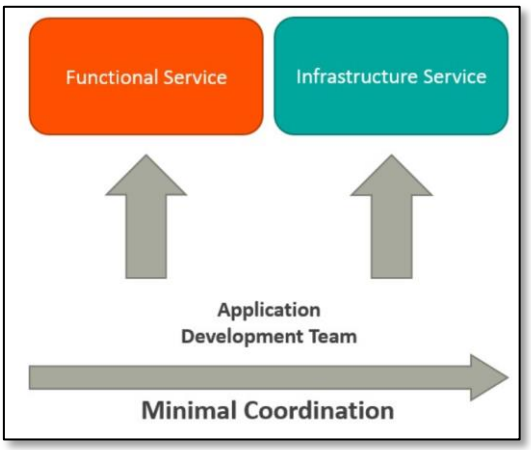

FIGURA 7 – Tipos de servicio según la arquitectura de microservicios MSA.
Fuente: BMC Blogs, diferencia entre SOA y MSA.

Mientras SOA necesita coordinar varios grupos para crear una solicitud, en MSA esta coordinación suele ser inexistente, y los servicios suelen tener un único propósito bien definido. SOA mejora la manera de compartir componentes, y MSA lo minimiza en función del contexto.

En cuanto al middleware, SOA ofrece muchas capacidades adicionales, y MSA sitúa la capa API entre los servicios y sus consumidores. Mientras que SOA promueve propagar muchos protocolos, MSA simplifica la arquitectura reduciendo el número de opciones de integración. Por otro lado el desacoplamiento del contrato, como principal elemento de la abstracción, es una capacidad fundamental que caracteriza a SOA y que no es admitida en una arquitectura basada en microservicios.

Pese a estas y algunas otras diferencias, se puede afirmar que tampoco se ha inventado nada radicalmente nuevo en cuanto a microservicios se refiere. Empresas como Amazon, Netflix o eBay han utilizado la estrategia “divide y vencerás” y han dividido funcionalmente sus aplicaciones monolíticas en aplicaciones más pequeñas. Tras el éxito de estas empresas, muchas otras han comenzado a adoptar la arquitectura MSA, que puede considerarse como la evolución lógica de SOA compatible con los casos de uso de las empresas más modernas.

Muchos expertos siguen considerando SOA como más adecuado en entornos de aplicaciones empresariales grandes y complejas, que requieren integrarse con muchas aplicaciones. Sin embargo para sistemas basados en web más pequeños y particionados, es preferible recurrir a la arquitectura MSA, que como desarrollador también va a ofrecer un mayor control.

Dependiendo también de las etapas del negocio es probable que empezar con MSA resulte la opción más adecuada, pero a medida que el negocio crezca, capacidades como transformar solicitudes complejas o integrar sistemas van a requerir reemplazar MSA por SOA. En cualquiera de los casos, ambas arquitecturas representan el mismo conjunto de estándares empleados en diferentes capas, pudiendo ver MSA como un subconjunto de SOA, o como esta última arquitectura como un enfoque más complejo o extenso.

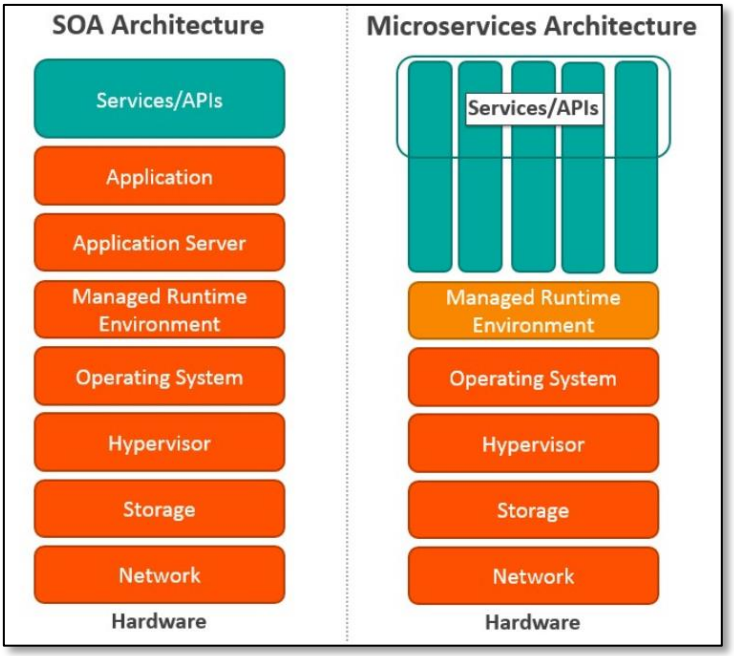

FIGURA 8 – Principales diferencias entre SOA y MSA.
Fuente: BMC Blogs, diferencia entre SOA y MSA.

La siguiente tabla, obtenida desde un blog de la empresa BMC dedicado a explorar las diferencias entre microservicios y arquitecturas orientadas a servicios, resume las características fundamentales de ambos:

	SOA	MSA
Idea principal	Compartir el máximo número de elementos.	Compartir lo mínimo, estrictamente lo que es necesario.
Concepto	Reutilizar la funcionalidad del negocio como principal prioridad.	Acotar el contexto como principal prioridad.
Gobernanza	Gobernanza y estándares comunes.	Gobernanza débil. Busca colaboración entre personas y libertad de elección.
Mensajería	Suele emplear ESB para la comunicación.	Sistemas de mensajería menos complejos.
Protocolos	Admite múltiples protocolos.	Protocolos ligeros como HTTP / REST y AMQP.

Plataformas	Plataforma común para todos los servicios desplegados.	No emplea servidores de aplicaciones. Puede utilizar plataformas como NodeJS.
Hilos de ejecución	Varios hilos, encareciendo el manejo de la entrada y salida.	Hilo único, que generalmente emplea funciones de bucle de eventos para manejar la entrada y salida.
Contenedores	Docker, Linux Containers…	Gran variedad de contenedores, incluso los menos populares son compatibles.
Enfoque	Maximiza la reutilización de servicios.	Se centra fundamentalmente en el desacoplamiento.
Bases de datos	Bases de datos relacionales, generalmente.	Bases de datos no relacionales.
Cambios	Un cambio sistemático requiere modificar el sistema.	Un cambio sistemático implica crear un nuevo microservicio.
Metodología	DevOps / Continous Delivery	DevOps / Continous Delivery

TABLA 1 – Principales diferencias entre SOA y MSA.
Fuente: BMC Blogs, elaboración propia.

Una arquitectura basada en microservicios es una herramienta que permite abordar la implementación de los diferentes servicios de forma totalmente desacoplada. La integración es algo que no desaparece, sino que más bien se transforma en diferentes implementaciones, como implementaciones punto a punto, ESB, servicios compuestos…

Lo que sucede realmente es que las capacidades tradicionales de integración de una arquitectura SOA se dispersan a través de cada una de las capas de microservicios compuestos, y los servicios que se ofrecen al exterior pasan a estar controlados por una capa de gestión de la API. Al final, existen muchas posibilidades en cuanto a la implementación, así como muchas tecnologías disponibles para cada implementación. La clave está en seleccionar el conjunto de tecnologías adecuado en cada implementación, de manera que se adapte a las necesidades del proyecto y se pueda justificar su elección ante otras candidaturas.

Al implementar una arquitectura basada en microservicios, el servicio de comunicaciones o mensajería interno es el elemento principal, que se encarga tanto de gestionar la información así como de redundar entre los posibles nodos o servidores. No obstante, son muchas otras las tecnologías y elementos necesarios que se requieren. La revista digital DZone recopila un total de 30 herramientas actuales como las más populares o recomendadas para construir una arquitectura basada en microservicios, asegurando que no hay una única herramienta capaz de resolver toda la implementación. Se trata de herramientas para gestionar la API, las comunicaciones, la arquitectura, monitorización, actualización mediante contenedores o incluso aquellas que abarcan las tecnologías Serverless o FaaS (Function As A Service), que abarcan estrategias para optimizar la metodología de dividir las cosas en las funciones más pequeñas posibles:

- GESTIÓN PARA LA API Y TESTING
  - **API Fortress** – Herramienta para automatizar el testing funcional, monitorizar y testear la carga de una API empresarial.
  - **Postman** – Suite para desarrollo de API's, que como cliente HTTP permite testear, desarrollar y documentar.
  - **Tyk** – Plataforma para administrar API's de código abierto, que funciona tanto en local como en la nube con alta disponibilidad y baja latencia.
  - **GraphQL** – Herramienta desarrollada por Facebook, que propone un lenguaje de consulta sobre la API.

- MENSAJERÍA Y COMUNICACIONES
  - **RabbitMQ** – Popular servicio de mensajería distribuido que permite intercambiar información y eventos entre diferentes servicios.
  - **Amazon Simple Queue Service** – Servicio de comunicación basado en publicación de mensajes y subscripción a través de colas.

- **Apache Kafka** – Plataforma de procesamiento de flujos de datos distribuida con alta tolerancia a fallos.
- **Google Cloud Pub/Sub** – Servicio de mensajería en tiempo real administrado, que permite enviar y recibir mensajes entre servicios.

- MONITORIZACIÓN
  - **Logstash** - Plataforma de código abierto que permite monitorizar datos sobre el estado y rendimiento de microservicios.
  - **Graylog** – Herramienta de monitorización de pago, interactiva y sencilla, diseñada de manera escalable para crecer junto con la arquitectura.

- CONTENEDORES
  - **Kubernetes** – Modelo para orquestación de contenedores, que se ha convertido en un estándar en cuanto a implementación se refiere.
  - **Telepresence** – Herramienta para desplegar contenedores compatibles con kubernetes de forma ágil, con buenos resultados tanto en producción como en desarrollo.
  - **Istio** – Herramienta para desplegar servicios con Kubernetes, que mejora la visibilidad y las interacciones entre la aplicación y el servicio.
  - **Minikube** – Herramienta de código abierto que permite ejecutar Kubernetes de forma portátil sin necesidad de WiFi.

- ORQUESTACIÓN
  - **Conductor** – Orquestador de microservicios creado por Netflix, que implementa un flujo con las diferentes tareas. También mejora la visualización de las interacciones entre microservicios.

- LENGUAJES DE PROGRAMACIÓN
  - **Elixir** – Lenguaje de programación concurrente y funcional para propósitos generales que puede funcionar conjuntamente con Erlang.

- **Springboot** – Framework sobre Java para creación de API's con ejemplos y distintas plantillas.

- HERRAMIENTAS
  - **Fabric8** – Plataforma de código abierto que ayuda a desarrolladores a gestionar las diferentes configuraciones a través de GIT.
  - **Seneca** – Kit de herramientas para NodeJS que permite sistematizar la lógica de negocio y crear procesos de microservicios basados en mensajes.
  - **Google Cloud functions** – Set de funciones en versión de prueba para la nube de Google, a través de una consola y una arquitectura dirigida por eventos. Permite encadenar servicios con otras API's de Google.

- FRAMEWORKS:
  - **Goa** – Framework para desarrollar REST API y microservicios en lenguaje Golang, permitiendo generar de manera automática JSON y código Javascript. Funciona preferiblemente en la nube de Google.
  - **Kong** – Framework compatible con muchos sistemas operativos, permite utilizar patrones de desarrollo basados en contenedores y en microservicios.

- HERRAMIENTAS SERVERLESS
  - **Claudia** – Gestiona implementaciones de AWS Lambda y otros tipos. Automatiza tareas de configuración y puesta en marcha.
  - **Apache Openwhisk** – Servicio de programación basado en eventos, que permite a desarrolladores crear, testear y conectar con otras plataformas.
  - **Serverless** – Consola que combina la tecnología FaaS / Serverless con otros servicios en la nube para facilitar el desarrollo de sistemas complejos de forma segura y escalable.

- **Kubeless** – Framework dentro del entorno de Kubernetes que permite implementaciones sin infraestructura, basado en primitivas y compatible con servidores de tipo K8.
- **IronFunctions** – Plataforma de código abierto que se puede ejecutar remotamente desde cualquier lugar, programada en Golang y compatible con AWS Lambda.
- **AWS Lambda** – Proporciona servidores sin infraestructura a través de una tarifa por uso. Se puede usar en combinación con AWS API Gateway, como programa en la nube de Amazon que aloja servicios para REST API.
- **OpenFaaS** – Software de código abierto que ofrece funciones simplificadas sin el uso de un servidor. Permite empaquetar procesos y contenedores con compatibilidad para Windows y Linux.
- **Microsoft Azure functions** – Proporciona funciones dirigidas a través d eventos para la nube de Microsoft. Es un servicio de pago que factura en función de los recursos utilizados.

A continuación se analizan en detalle algunas de estas herramientas que, por su popularidad y su aceptación por la comunidad de desarrolladores, se considera interesante desarrollar en profundidad antes de proceder con el caso de estudio.

5.1 Apache Kafka

Kafka es una plataforma distribuida de streaming creada por LinkedIn en el año 2011, que permite manejar sistemas de alto rendimiento, transmisiones de baja latencia y procesamiento de flujos de registros en tiempo real. Dentro de una arquitectura de microservicios, podría implementar la parte de mensajería y comunicaciones. Sus tres capacidades fundamentales son:

- Publicación y suscripción a flujos de registros. Es un concepto similar a una cola de mensajes o un sistema de mensajería empresarial.
- Almacenamiento de flujos de registros con una elevada tolerancia a fallos.
- Procesamiento de flujos de registros en tiempo real.

Resulta una solución idónea para aquellas aplicaciones que construyen flujos de datos fiables entre sistemas en tiempo real, o para aplicaciones que transmiten datos en tiempo real, reaccionan y transforman estos flujos de datos. Se ejecuta como un clúster en uno o varios servidores, empleando uno o varios centros de datos, y el propio clúster almacena flujos de registros organizados en categorías, bajo el nombre de "temas" o "subjects". Un registro se compone de una clave, un valor y una marca de tiempo.

La API de Kafka se distribuye en cuatro secciones:

- PRODUCTOR: Permite a una aplicación publicar una secuencia de registros a uno o más temas de Kafka.
- CONSUMIDOR: Permite que una aplicación se suscriba a uno o más temas y procese la secuencia de registros que se ha producido.
- FLUJOS: Permite que una aplicación actúe como un procesador de flujos, consumiendo un flujo de entrada de uno o más temas, y produciendo un flujo de salida a uno o más temas de salida. Por tanto, transforma flujos de entrada a flujos de salida.
- CONECTOR: Permite crear y ejecutar productores o consumidores reutilizables que conectan diferentes temas de Kafka a aplicaciones o sistemas de datos existentes. Por ejemplo, un conector en una base de datos relacional es capaz de capturar cada cambio que se produce en una determinada tabla.

La comunicación entre clientes y los servidores se realiza a través de un protocolo TCP simple, de alto rendimiento y sin lenguaje. En la web oficial se puede obtener un cliente de ejemplo en Java, pero este está disponible para muchos otros lenguajes.

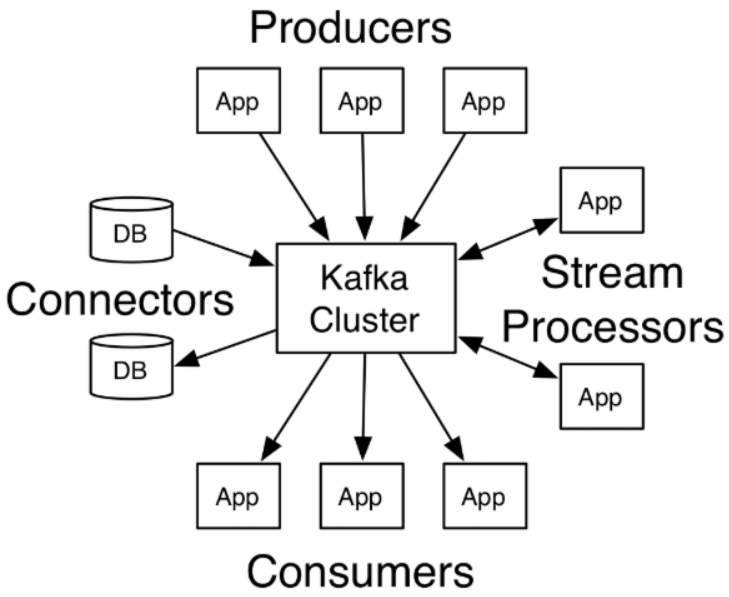

FIGURA 9 – Ecosistema y actores principales de un clúster de Kafka.
Fuente: Web oficial de apache Kafka.

El tema o categoría supone pues el principal elemento de abstracción de Kafka para un conjunto de registros. Para cada categoría puede haber cero, uno o muchos consumidores que se subscriban para escribir o leer información, y para cada categoría el clúster genera un log particionado que esta ordenado y es inmutable, con un ID de secuencia único llamado offset que identifica de forma única cada registro de la partición.

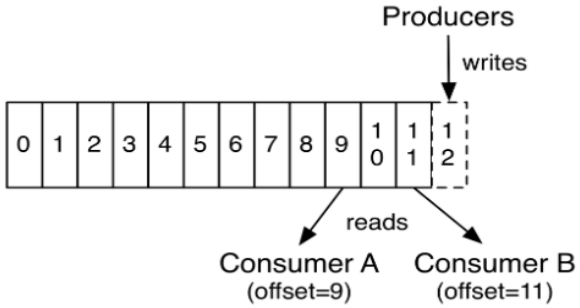

FIGURA 10 – Ejemplo de un “topic” de Kafka con un productor y dos consumidores.
Fuente: Web oficial de apache Kafka.

Durante un tiempo de persistencia configurable, el clúster mantiene todos los registros, independientemente de que hayan sido consumidos o no. Dado que es un clúster muy potente, no afecta al rendimiento mantener datos durante mucho tiempo, pero si se desea se puede establecer un periodo y a partir de que este expire, los registros ya no estarán disponibles. Generalmente un consumidor trabaja con su propio offset, de manera que sabe que registros ha leído del clúster y cuales no ha leído.

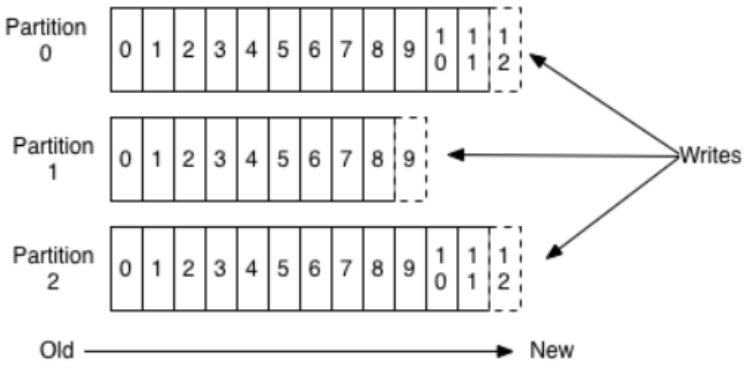

FIGURA 11 – Esquema temporal de particiones sobre un mismo “topic” de Kafka.
Fuente: Web oficial de apache Kafka.

Las particiones del log se distribuyen a través de los diferentes servidores que componen el clúster. Cada partición tiene un servidor que actúa como líder, y cero o más servidores que actúan como seguidores. El servidor líder maneja todas las lecturas y escrituras para la partición mientras que los seguidores son una réplica del mismo. La arquitectura garantiza la redundancia ya que si el servidor líder falla, cualquiera de los seguidores pasará a ser el nuevo líder, y de esta forma también se balancea la carga.

A su vez el sistema incorpora una librería llamada Mirror Maker, que proporciona soporte geográfico para los clúster en diferentes regiones (Amazon Web Services gestiona de manera similar las diferentes zonas geográficas). Cada productor puede generar datos a cualquiera de los canales o categorías a los que está subscrito, al igual que cada consumidor puede leer datos de todos los canales a los que está subscrito. Estos últimos pueden estar agrupados y pertenecer a diferentes grupos.

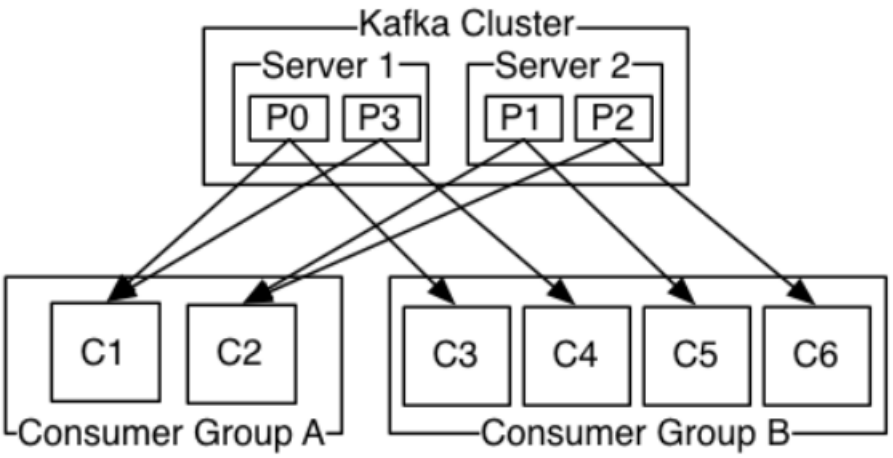

FIGURA 12 – Esquema temporal de particiones sobre un mismo "topic" de Kafka.
Fuente: Web oficial de apache Kafka.

Kafka también es popular y empleado en aplicaciones multi-tenant, a través de la configuración de diversos canales que pueden producir o consumir datos. En cualquier implementación, es capaz de proporcionar las siguientes garantías:

- Los mensajes enviados por un productor a un canal particular son añadidos en el orden en que son enviados.
- El consumidor leerá los registros en el orden en el que fueron almacenados.
- Un canal con un factor de replicación N tolerará N-1 fallos sin perder ningún registro enviado al log.

Las implementaciones típicas son por ejemplo un sistema de mensajería, un sistema de almacenamiento, un procesador de streamings, una aplicación para registrar el tráfico de una web, implementación de métricas, implementación de logs, repositorios, servidores de comunicaciones… Sea cual sea la aplicación que se implemente, cada canal o categoría será un microservicio independientemente, de manera que realizar cualquier actualización sobre uno en concreto no afectara al resto. Por otro lado, si se dispone de varios servidores para cada microservicio, garantizará la redundancia así como la tolerancia a fallos de forma directamente proporcional al número de equipos que repliquen el canal en cuestión.

5.2 RabbitMQ

RabbitMQ es el sistema de mensajería de código abierto con mayor número de implementaciones, alcanzando los 35.000 entornos de producción entre pequeñas y grandes empresas. Al igual que Kafka podría ser la elección en una arquitectura basada en microservicios que implemente mensajería y comunicaciones, suponiendo una alternativa diferente a este.

Es ligero, fácil de implementar y compatible con servidores locales y servidores en la nube. Se puede utilizar con numerosos protocolos de mensajería, y también funciona con configuraciones distribuidas para empresas que necesitan productos con una alta disponibilidad.

Se ejecuta en muchos sistemas operativos y entornos cloud, y provee multitud de herramientas para desarrolladores así como ejemplos e implementaciones en muchos lenguajes de programación. De esta manera, un agente de mensajería puede actuar como intermediario para varios servicios, de manera que puede reducir la carga y los tiempos de entrega a través de diferentes servidores, ya que cada tarea puede ser delegada y externalizada. Estos son los principales conceptos a tener en cuenta en el sistema de mensajería:

- PRODUCTOR: Aplicación que envía los mensajes.
- CONSUMIDOR: Aplicación que recibe los mensajes.
- COLA: Buffer que almacena los mensajes.
- MENSAJE: Paquete o información que se intercambia.
- CONEXIÓN: Conexión TCP entre la aplicación y el agente RabbitMQ.
- CANAL: Conexión virtual dentro de una conexión.
- ENTIDAD DE INTERCAMBIO: Se encarga de dirigir los mensajes a las colas.
- ENLACE: Puente o vínculo entre una cola y un intercambio.
- CLAVE DE ENRUTAMIENTO: Clave con información para el enrutamiento.

FIGURA 13 – Arquitectura productor-consumidor de RabbitMQ.
Fuente: Cloudamqp, introducción a RabbitMQ.

La arquitectura es muy sencilla, ya que para una misma cola de mensajes existen aplicaciones cliente (productores) que crean mensajes y los entregan al agente intermediario. Otro tipo de aplicación llamada consumidor se conecta a la misma cola y se suscribe a los diferentes mensajes que se procesarán. Una aplicación puede ser productor, consumidor, o consumidor / productor de mensajes, y los mensajes de la cola se almacenan en la cola hasta que el consumidor los recupera.

La cola de mensajes permite a los servidores responder de forma ágil a las diferentes solicitudes en lugar de lanzar procedimientos muy pesados a nivel de recursos. La cola también es una estructura idónea cuando se pretende distribuir un mensaje a múltiples destinatarios, o equilibrar la carga entre los diferentes sistemas que componen la distribución.

Un consumidor puede consumir un mensaje de la cola y procesarlo en el mismo momento en el que un productor está encolando nuevos mensajes, y ambos pueden encontrarse en servidores completamente distintos. De manera similar, la solicitud se puede implementar en un lenguaje de programación y el consumidor puede emplear otro diferente. Al final, la comunicación entre las dos aplicaciones sólo se produce a través de mensajes, por lo que el acoplamiento es mínimo.

En la siguiente figura se refleja el procesamiento de una solicitud de creación de un documento PDF a través de una aplicación web, y como se produce el intercambio de información entre el productor y el consumidor.

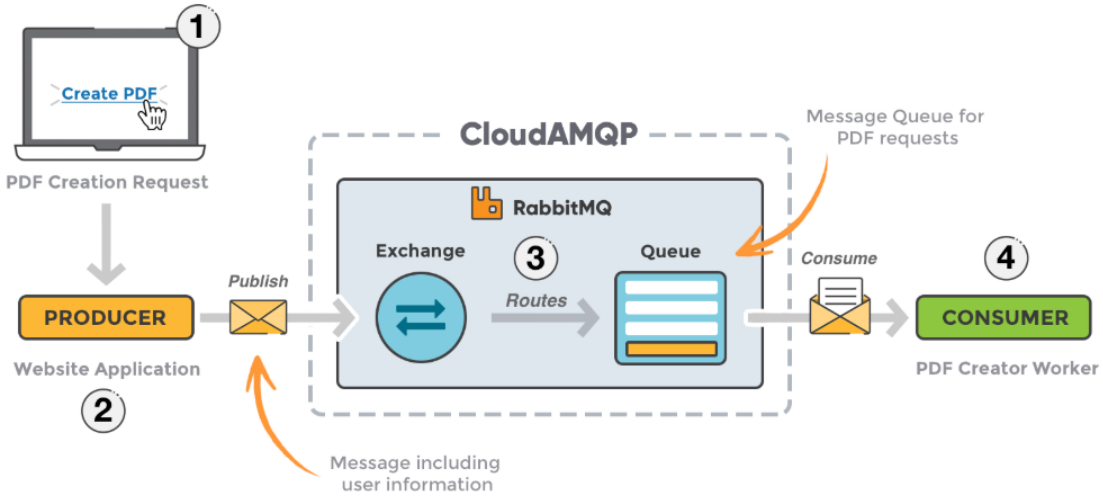

FIGURA 14 – Ejemplo de arquitectura de RabbitMQ para intercambio de documentos electrónicos. Fuente: Cloudamqp, introducción a RabbitMQ.

Con RabbitMQ los mensajes no son publicados directamente en una cola, sino que un elemento encargado de gestionar el intercambio dirigirá cada mensaje a la cola que corresponda, gracias a enlaces y claves de enrutamiento. El flujo sería el siguiente:

1. El productor publica un mensaje hacia la entidad de intercambio. Al crear el mensaje, se deben especificar los detalles necesarios para que se produzca este intercambio.
2. La entidad de intercambio recibe el mensaje y, a través de los detalles especificados en función del tipo de intercambio, se dirigirá el mensaje hacia la cola apropiada.
3. La entidad de intercambio envía el mensaje a las colas destinatarias.
4. El mensaje permanece en la cola hasta que es desencolado y gestionado por un consumidor.
5. El consumidor recibe el mensaje.

Existen tres tipos de intercambios principalmente, tal y como se refleja en la siguiente figura:

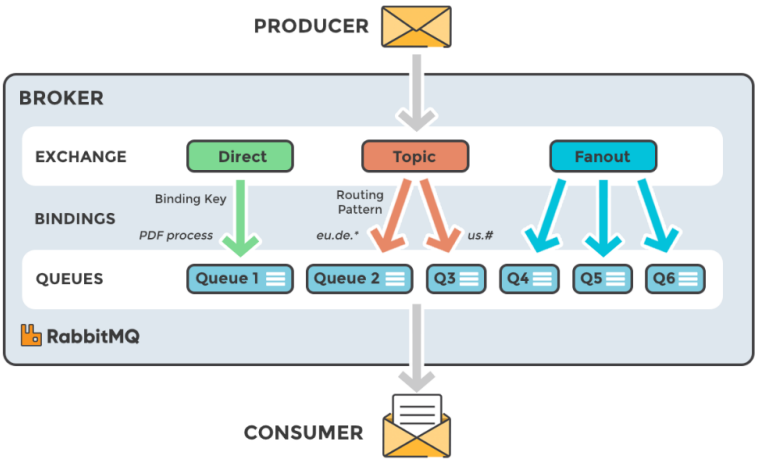

FIGURA 15 – Tipos de intercambio con RabbitMQ.
Fuente: Cloudamqp, introducción a RabbitMQ.

Cada uno aporta una serie de posibilidades y requiere que el productor especifique diferente información para poder direccionar el mensaje. Estos tipos son:

- DIRECTO: Se entregan mensajes a colas en función de una clave de enrutamiento. El mensaje se dirige a las colas cuya clave coincide exactamente con la clave del enrutamiento del mensaje.
- TOPIC: Se direccionan los mensajes a través de un "topic" o tema, a través del cual se realizan las comprobaciones con un patrón de enrutamiento.
- FANOUT: Direcciona los mensajes a todas las colas que están vinculadas a él.

De cara a la redundancia, un agente RabbitMQ es una agrupación lógica de uno o varios nodos, cada uno de los cuales ejecuta independientemente la aplicación y comparte usuarios, hosts virtuales, colas, enlaces, agentes de intercambio y parámetros de tiempo de ejecución. Una colección de nodos también se puede definir como un clúster, y este se puede crear de varias maneras, ya que permite definir las DNS de los diferentes nodos, definir las diferentes instancias de Amazon Web Services, Kubernetes u otras tecnologías disponibles, o directamente especificando cada nodo en un archivo de configuración. A demás una definición de un clúster con todos sus nodos puede ser modificada dinámicamente. Por otro lado estos nodos se pueden agrupar en grupos o trabajar como agentes individuales, utilizando siembre nombres de dominio para identificarse entre sí. Generalmente, el sistema de mensajería emplea

diferentes puertos TCP por defecto para las herramientas cliente, servicios de búsqueda de nodos, clientes HTTP, clientes STOMP o clientes de WebSockets.

Todos los datos y estados almacenados por un agente RabbitMQ son replicados en el resto de nodos excepto las colas de mensajes, que por defecto residen en un solo nodo. Por último, mencionar que existe una API HTTP que ofrece herramientas de gestión y monitorización de los nodos y clústeres de RabbitMQ, así como una interfaz web y una línea de comandos. Ésta recopila datos y métricas sobre el estado actual del sistema, y también permite gestionar usuarios, políticas de privacidad, monitorizar las colas, servidores virtuales o cadencias de mensajes, entre otros.

5.3 GraphQL

GraphQL es una alternativa a REST, creada por Facebook en el año 2012 y de código abierto a partir del año 2015. Es una nueva forma de especificar API, y al mismo tiempo aporta un lenguaje de consultas sobre la API que especifica. Permite a desarrolladores definir esquemas de datos o crear nuevos campos sin reestructurar la aplicación de un cliente. Es una herramienta ideal para ser empleada en una arquitectura de microservicios, ya que puede aportar los siguientes beneficios:

- Permite separar los propietarios de los datos, ya que una solicitud de datos apunta siempre hacia un único punto.
- Permite ejercer mayor control sobre el proceso de carga de datos, ya que permite controlar como son transferidos los datos y que carga existe en cada microservicio.
- Permite definir consultas de gran alcance, planificando hacia qué puntos de la API deben ser dirigidas. En función del peso y la prioridad, permite planificarlas de manera efectiva en base a diferentes criterios.

Cuando se ha migrado una aplicación monolítica a una basada en microservicios, operar sobre ella puede presentar desafíos e inconvenientes. Esta herramienta supone una gran ayuda a la hora de facilitar el uso de esta nueva arquitectura basada en MSA.

Actualmente las API's más populares son RESTful o diferentes estándares bajo HTTP, pero como cada vez son más complejas, herramientas como GraphQL aportan mucha flexibilidad a una API convencional. Txema Rodriguez, editor y experto tecnológico de la revista digital GenBeta, identifica las siguientes debilidades en una RESTful API:

- En una RESTful API se utiliza una URI para escribir o leer un único recurso vinculado a una entidad. Si se desea trabajar con múltiples recursos o entidades, se deben manejar múltiples endpoint y encadenar diferentes llamadas, a veces incluso de manera secuencial. Son las apps o aplicaciones consumidoras las que tendrán la responsabilidad de unir todo el contenido.
- Cuando se realiza una petición, se recibe el conjunto de datos relativo al recurso asociado a esa URI, por lo que siempre se van a procesar una gran cantidad de datos innecesarios desde el lado del cliente.
- El versionado de una REST API no es trivial, ya que añadir campos o modificarlos manteniendo la retro-compatibilidad hace crecer el payload de respuesta.
- Muchas veces no se emplean las sentencias HTTP involucradas de forma correcta, por lo que desde el lado del servidor no se puede garantizar la correcta utilización de la API. Cada API viene acompañada de una documentación para el desarrollador que corre el riesgo de ser incompleta o estar desactualizada sin previo aviso. Se pierde el valor de las herramientas de documentación de API's como Swagger.

La ventaja que aporta GraphQL es que permite elegir qué tipo de datos queremos recibir en el JSON de respuesta, se puede definir exactamente lo que se necesita de cada objeto. Es la principal alternativa que ha surgido para tratar de solucionar todos estos problemas que se han descrito con REST API. El proyecto nació dentro de Facebook, pero empresas como Github, Pinterest o Shopify también están colaborando en su definición. Como se trata de un protocolo agnóstico, no presenta ninguna dependencia con HTTP, y no emplea sus métodos o respuestas.

GraphQL solo require de una única petición mientras que con REST serían necesarias varias peticiones para poder obtener los mismos datos. Emplea el mismo lenguaje y la misma sintaxis en la petición y en la respuesta, utilizando lenguajes pregunta – respuesta embebidos en JSON. Dispone de un sistema de tipado fuerte, por lo que un mal uso puede ser rápidamente detectado en tiempo de desarrollo. La API puede tener un único endpoint, de manera que a través de este se manejen todas las peticiones de los clientes. Otra herramienta llamada GraphiQL permite documentar la API, pero todavía queda lejos de herramientas para REST API como Swagger, que se encuentran en un estado mucho más maduro en cuanto a documentación respecta.

```
{
  user(id: 4802170) {
    id
    name
    isViewerFriend
    profilePicture(size: 50)  {
      uri
      width
      height
    }
    friendConnection(first: 5) {
      totalCount
      friends {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "user": {
      "id": "4802170",
      "name": "Lee Byron",
      "isViewerFriend": true,
      "profilePicture": {
        "uri": "cdn://pic/4802170/50",
        "width": 50,
        "height": 50
      },
      "friendConnection": {
        "totalCount": 13,
        "friends": [
          {
            "id": "305249",
            "name": "Stephen Schwink"
          },
          {
            "id": "3108935",
            "name": "Nathaniel Roman"
```

FIGURA 16 – Ejemplo de definición de peticiones a través de GraphQL.
Fuente: Tutorial en OCTO Talks.

A demás de las implementaciones de GraphQL bajo JavaScript, las librerías en el servidor incluyen implementaciones y ejemplos para muchos lenguajes de programación, como Java, C#, Erlang, Groovy, PHP, Python, Scala o Ruby, entre otros.

5.4 Kubernetes

Kubernetes o K8S, es un sistema open-source que automatiza el despliegue, escalado y gestión de aplicaciones basadas en contenedores. Los contenedores son una parte fundamental de la arquitectura de microservicios, ya que se alinea con sus principios sobre desarrollo ágil, escalable e independiente que exige. Kubernetes es un orquestador de contenedores diseñado por Google, y soporta los principales entornos de contenedores como docker, rkt, cri-o o frakti. Se está convirtiendo en un estándar de facto por las ventajas que aporta con respecto a otras soluciones. Juan María Fiz, ingeniero informático y colaborador de la revista online Paradigma Digital, destaca las siguientes características:

- Escalado y auto-escalado: En función del uso de la CPU permite escalado vertical de forma automática o manual.
- Descubrimiento de servicios y balanceo de carga: Kubernetes asigna a cada contenedor su propia IP y un nombre DNS único para un conjunto de contenedores, balanceando la carga sobre ellos.
- Auto-reparación: En caso de error el sistema se encarga de realizar un reinicio automático, incluso cuando un nodo está inactivo. Permite definir puntos de control para comprobar el estado de los contenedores y su despliegue.
- Rollback automático, a través de un despliegue progresivo a través de las diferentes instancias.
- Planificación, a través de la asignación de nodos para contenedores en función de los recursos exigidos u otras restricciones.
- Gestión de información sensible como contraseñas o claves ssh a través de entidades llamadas *secrets*, que impiden exponer información confidencial.
- Orquestación del almacenamiento, compatible con GCP y AWS y sistemas de almacenamiento de red, como NFS, SCI, Gluster…
- Ejecución mediante lotes, que permiten gestionar las cargas de trabajo y reemplazar aquellos contenedores que fallen.

Cada vez está ganando más popularidad, y son muchas las soluciones que lo integran dentro de su arquitectura. Permite desplegar un clúster sobre máquinas físicas y bajo diferentes sistemas operativos, así como en máquinas virtuales o en soluciones Cloud. En caso de emplear Kubernetes en una arquitectura basada en microservicios, lo más usual sería hacer uso para un despliegue de un contenedor en la nube. También es compatible con Google Container Engine, OpenShift, CoreOS Tectonic, Kops, Amazon AWS, Azure o IBM entre otros, así como con todos los sistemas de virtualización de la empresa VMWare.

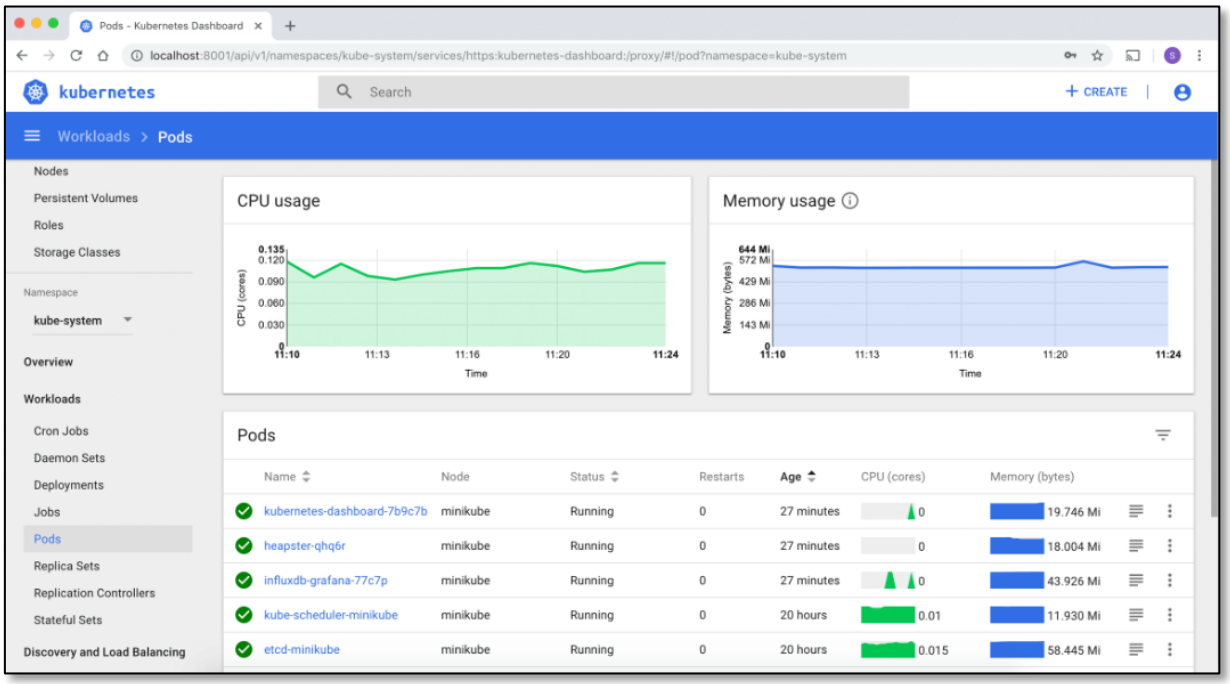

FIGURA 17 – Interfaz de usuario de Kubernetes.
Fuente: Web oficial de Kubernetes.

Es una plataforma de contenedores en la nube, una plataforma de microservicios independientes entre sí que proporciona un entorno de gestión centrado en unidades individuales. Organiza la infraestructura de computación, redes y almacenamiento de todos los usuarios, y proporciona la simplicidad de una plataforma como servicio (PaaS) junto con la flexibilidad de una infraestructura como servicio (IaaS). Permite la

portabilidad entre diferentes proveedores de infraestructura, siendo muy sencillo migrar desde uno a otro.

En cuanto a la arquitectura respecta, Kubernetes sigue un patrón maestro – esclavo, donde los componentes son divididos en aquellos que administran un nodo individual y aquellos que forman parte de un panel de control. Estos son los elementos básicos:

- PODS – Es la unidad básica de planificación. Agrega un nivel de abstracción por encima del contenedor en sí, ya que se compone de uno o varios contenedores que se ubican en el mismo equipo y comparten los mismos recursos. Dentro de un clúster, posee una dirección IP única.
- ETIQUETAS / SELECTORES – Permite a los clientes asignar pares compuestos por una clave y un valor a cualquier objeto API del sistema, como un Pod o un Nodo. Son el mecanismo principal de agrupamiento para determinar sobre que componentes se realiza una operación.
- CONTROLADORES – Mecanismo para conducir un pod o un conjunto de pods hasta un estado deseado. Son empleados para replicación, para trabajo mediante lotes o para tareas de redundancia.
- SERVICIOS – Conjunto de pods que trabajan en conjunto, como una capa de una aplicación multicapa. El conjunto de pods viene definido por un selector, y existe un servicio interno para descubrir los diferentes servicios.

Al crear un clúster, Kubernetes coordina una serie de equipos de alta disponibilidad que actúan de manera conjunta para funcionar como una sola unidad. De esta manera se pueden implementar aplicaciones en clústeres sin estar vinculadas a uno o varios equipos en concreto. A partir de ahí, el despliegue, exploración, mantenimiento, ampliación o la actualización de una aplicación se puede llevar a cabo sin afectar al resto de aplicaciones. Kubernetes se está consolidando como la principal solución de orquestación, presente en la mayor parte de servicios Cloud de las empresas más importantes. Es una tecnología de la que todo el mundo habla, la más escogida para desarrollar aplicaciones basadas en contenedores la que mayor incremento está registrando en cuanto al número de contribuidores.

5.5 Serverless

Es un modelo de ejecución de computación en la nube mediante el cual el proveedor ejecuta el servidor y administra dinámicamente la asignación de recursos del equipo. El precio se basa en la cantidad de recursos consumidos por una aplicación, en lugar de unidades de capacidad adquiridas previamente. Este modelo abre la puerta a simplificar el proceso de implementar código en producción, donde las operaciones de escalado, planificación de la capacidad o mantenimiento son trasparentes para el desarrollador u operador. Este tipo de código puede usarse conjuntamente con otras aplicaciones desarrolladas bajo el paradigma de microservicios, y de forma alternativa, las aplicaciones pueden ser desarrolladas de manera que no tengan servidor o no tengan en cuenta el aprovisionamiento.

La primera plataforma de ejecución de código fue Zimki, creada en 2006, pero no tuvo mucho éxito. Años después, en 2008, Google lanzo su App Engine que facturaba por ejecución de aplicaciones desarrolladas en Python. PiCloud salió al mercado y también ofrecía soporte FaaS (Function As A Service) para desarrollos en Python. En 2014 AWS Lambda de Amazon fue el primer proveedor con una infraestructura basada en la nube que lanzó una oferta de computación de tipo “serverless”. Posteriormente en 2016 Google también lanzó su Google Cloud Functions disponible para su plataforma en la nube, y otras empresas como Oracle, Microsoft o IBM también ofrecen un servicio similar.

A raíz de esta tendencia también ha aparecido el concepto de bases de datos sin servidor, como modelo que elimina la necesidad de aprovisionar o escalar el hardware necesario en una base de datos de un entorno virtualizado o físico. Un ejemplo es Azure Data Lake de Microsoft, servicio de análisis y almacenamiento de datos altamente escalable y alojado en la nube pública de esta empresa. Proporciona una infraestructura distribuida que puede asignar o desasignar dinámicamente los recursos para que los clientes solo paguen por los servicios que utilizan. La alternativa en Google es su Cloud Datastore, un almacén de documentos que ofrece el componente de base de datos de Google App Engine como un servicio independiente. Firebase,

también propiedad de Google, incluye una base de datos jerárquica y está disponible a través de planes fijos y de pago por uso.

En general, este nuevo paradigma aporta ventajas en cuanto al coste ya que sólo se paga por la actividad realmente requerida, y la gestión de recursos resulta mucho más eficiente. Se eliminan costos asociados a licencias, instalación, mantenimiento o soporte, entre otros.

Por otro lado una arquitectura sin servidor permite que los desarrolladores eviten invertir tiempo en configuración y políticas de escalado, ya que es el proveedor de la nube el responsable de esto. Se pasa a hablar de sistemas elásticos en lugar de sistemas escalables, y los desarrolladores ejecutan el código sin depender de ingenieros de soporte o infraestructura. La línea entre un desarrollador de software o un ingeniero de hardware se difumina cada vez más con este modelo. Otra ventaja es que el desarrollador se preocupa cada vez menos por los subprocesos en el código, ya que las unidades de código expuestas al exterior son simples funciones a través de solicitudes HTTP.

Como desventajas, se puede dar el caso en que aumente la latencia de respuesta en bloques de código ejecutados con poca frecuencia. Este modelo tampoco resulta compatible con la computación de alto rendimiento, debido a los límites establecidos por los proveedores de la nube. La monitorización y la depuración pueden resultar más complejas, y en cuanto a la seguridad, pese a que algunos expertos no comparten las mismas opiniones, la superficie de ataque es significativamente mayor ya que son más componentes los que están expuestos frente a las arquitecturas tradicionales.

6 CASO DE ESTUDIO

Se propone desarrollar un sistema inteligente para predecir la llegada de barcos de contenedores a una terminal. Para conseguir esto se va a realizar una integración entre diferentes sistemas software y sus respectivas bases de datos. El resultado será un modelo de datos con una arquitectura basada en microservicios, que servirá las estimaciones y resultados obtenidos al exterior. El software a su vez empleará un modelo predictivo basado en fiabilidades, que aprenderá y mejorará sus estimaciones a medida que se produzcan más ejecuciones.

6.1 Contexto

Una terminal marítima recibe diariamente visita de diferentes barcos de contenedores, que a su vez han realizado una llamada tanto al puerto como a la terminal correspondiente. Para poder operar sobre las diferentes mercancías, es necesario disponer de un plan, así como de los diferentes recursos, equipamiento y personal necesario para llevarlo a cabo. Todos los costes asociados a la operativa son muy importantes, de cara a rentabilizar y cuadrar las cuentas de las diferentes empresas que intervienen. Por este y otros motivos, resulta muy importante conocer el tiempo estimado de llegada de un barco con la mayor precisión posible, ya que a partir de entonces muchas empresas y entidades asociadas comenzarán a facturar y/o cobrar por los servicios prestados.

El tiempo de llegada de un barco se conoce internacionalmente bajo las siglas de ETA, Estimated Time of Arrival. Al igual que sucede con muchos otros datos que se manejan en el sector, está presente en diferentes sistemas software y ninguno de ellas aporta la suficiente fiabilidad o garantiza que este tiempo vaya a ser aproximado. A través de una integración entre las principales fuentes de datos que manejan el tiempo estimado de llegada de un barco, se van a unificar en un mismo modelo de datos todas las estimaciones, permitiendo valorar la fiabilidad de las mismas y ofrecer a todas las

entidades externas interesadas el dato más preciso de entre los disponibles. Este contexto es óptimo para implementar una arquitectura SOA o MSA, en la que se construye una única fuente de datos con mayor veracidad y calidad sobre todas las demás que la componen.

6.2 Fuentes de datos

Las principales fuentes de datos a considerar son las siguientes:

- **Terminal de contenedores:** A través de su TOS (Terminal Operating System), la terminal conoce de antemano todas las visitas de los distintos barcos que espera. Por tanto registra el ETA o Estimated Time of Arrival para cada barco, que es remplazado por el ATA o Actual Time of Arrival una vez atraca en la terminal. Para la terminal resulta de vital importancia este dato, ya que es el punto de partida para comenzar a organizar el equipamiento necesario, el número de grúas, transporte horizontal, las manos y personal de estiba necesario, los diferentes transportistas que vendrán a recoger o depositar la mercancía... En el TOS este dato puede crearse de manera manual por parte de cualquier usuario. Dependiendo del software empleado, también puede crearse de forma automática a través de un canal electrónico de intercambio de información, a través de un servicio web o a través de cualquier otro medio de comunicación.

- **Empresa naviera:** Como propietaria de la embarcación, la naviera conoce y almacena cuál es el tiempo estimado de llegada a través de los diferentes sistemas software. Resulta de vital importancia estimar este dato con la mayor fiabilidad posible, ya que en tierra estarán esperando todos los recursos necesarios para realizar la operativa en el tiempo pactado. Como se suele afirmar en este tipo de empresas, un barco parado está perdiendo dinero, y un barco en movimiento está generando dinero. Contra más tiempo pasa un barco

atracado en una terminal, más costes tiene para la empresa naviera, por el precio del amarre, la manipulación de la mercancía, el personal y la maquinaria implicados… Normalmente es la tripulación a bordo es la que mejor conoce el tiempo estimado de llegada a un puerto, y mantener este dato actualizado en los sistemas software de la naviera para la que está operando puede ahorrar costes e imprevistos.

- **Autoridad portuaria:** Como ente que gestiona un puerto junto con todas las terminales de mercancías que lo componen, emplea distintos software para gestionar el tráfico marítimo. Es conocedora de todas las visitas que se van a producir en cada terminal, así como los tiempos asociados a ella.

Existen otras entidades y actores involucrados en la visita de un barco, que a través de los diferentes sistemas también van a registrar el tiempo de llegada estimado. Estos pueden ser empresas aduaneras, la Guardia Civil o Policía, las empresas encargadas de proporcionar los barcos remolcadores en caso necesario…

6.3 Flujo de ejecución

1. **FASE 1**: Creación de la visita

   En esta fase se genera la visita y su tiempo estimado de llegada asociado (ETA). Los diferentes sistemas mencionados van a crear la entidad correspondiente una vez sean informados de la visita del barco. Sea cual sea el canal que realiza la llamada y el modo en el que es insertado en el sistema, siempre que se crea el registro se asocia un tiempo estimado de llegada. En un escenario real esta creación se lleva a cabo en diferentes momentos de tiempo. Como se trata de un escenario simulado, todos los registros se crearán todos en el mismo momento.

2. **FASE 2**: Recopilación de los datos

   Esta fase comienza cuando todas las visitas han sido creadas en los distintos sistemas que simulan una terminal de contenedores, una empresa naviera y una autoridad portuaria. A partir de ahí el software desarrollado captura para una misma visita los datos desde los distintos orígenes, y almacena en su propio modelo de datos toda la información necesaria para comenzar a trabajar sobre ellos.

3. **FASE 3**: Análisis de los datos

   A partir de las últimas fiabilidades almacenadas en el sistema, se aplicara una fórmula mediante la cual se obtendrá un tiempo estimado de llegada mejorado. Básicamente esta fórmula otorga un peso o factor de veracidad a cada una de las fuentes, y a partir de ahí obtiene un único resultado que se basa en observaciones anteriores.

4. **FASE 4**: Envío del tiempo estimado de llegada

   El sistema ofrece al exterior a todas las empresas interesadas y con acceso al servicio un único tiempo estimado de llegada. Este presume ser el mejor tiempo estimado de entre todas las fuentes consultadas, basado en la fiabilidad calculada a partir de observaciones anteriores.

5. **FASE 5**: Llegada del barco

   Una vez llega el barco a la terminal, el tiempo estimado de llegada pasa a ser el tiempo real de llegada (también conocido como ATA, Actual Time of Arrival). Este dato es en última instancia el dato real y a partir del cual se van a determinar las fiabilidades. El dato se registrará de forma manual en el sistema.

6. **FASE 6**: Calculo de fiabilidades

   A partir de esta ejecución individual, se comparan todos los tiempos estimados que ofrecieron las diferentes fuentes de datos y cuál fue el error cometido por

cada una al compararlo con el tiempo real. Esto permite registrar la fiabilidad para esta ejecución o visita de un barco, y para cada uno de los sistemas. Esta fiabilidad será empleada junto con todas las demás anteriores para estimar el tiempo de llegada del barco en la siguiente ejecución.

6.4 Consideraciones

Este escenario propuesto es muy limitado. En los ejemplos que se van a realizar el número de ejecuciones va a ser reducido y sólo se emplearán tres fuentes de datos para cada una. No obstante, el software desarrollado permitiría conectar más fuentes de datos así como incrementar el número de ejecuciones. A medida que se produzcan más visitas de barcos, la fiabilidad obtenida para cada fuente será mucho más aproximada a la realidad. Por tanto, el sistema permitirá predecir y ofrecer el tiempo estimado de llegada que más se aproxima a la realidad, y a medida que se realicen más ejecuciones el sistema será capaz de emitir mejores precisiones.

Como resultado, el software desarrollado permitirá servir al exterior tanto el tiempo estimado de llegada para una visita en concreto, como la fiabilidad de cada uno de los sistemas involucrados. Estos dos servicios serán implementados en forma de microservicios, demostrando que como aplicaciones individuales, una queda totalmente desacoplada de la otra y mientras se ejecuta una mejora o actualización en uno de los servicios, el otro puede seguir funcionando si verse afectado en absoluto al no tratarse de una arquitectura monolítica.

7 IMPLEMENTACIÓN

En este capítulo se detallan todos los aspectos relativos a la implementación del sistema, así como la arquitectura del mismo y las partes de código más relevantes. De manera resumida, se va a desarrollar una aplicación que servirá sus datos a través de una RestAPI y a través de un micro servicio, de manera que permita combinar ambos modelos y debatir acerca de las ventajas e inconvenientes de cada uno en el siguiente capítulo. A través de todo el material examinado, son muchos los expertos que siguen considerando SOA como más adecuado para entornos de aplicaciones empresariales grandes y complejas, dado que requieren integrarse con muchas aplicaciones. Sin embargo para sistemas basados en web más pequeños y particionados, es preferible recurrir a la arquitectura MSA, que va a ofrecer un mayor control. En el software desarrollado, y pese a que sólo se van a integrar tres sistemas distintos, se deja la puerta abierta a evolucionar ambas arquitecturas en función de posibles y futuros requerimientos.

7.1 Entradas de datos

Las tres fuentes de entrada se van a implementar de la siguiente forma:

- **TOS – Terminal de contenedores:** Dado que gran parte de los sistemas disponibles en el mercado funcionan sobre una base de datos, que en la mayoría de los casos es de tipo relacional, el TOS se va a implementar mediante un modelo similar. Se realizarán consultas a la tabla correspondiente, tratando de simular lo que sería un acceso real al software de la terminal. Para ello se va a emplear la versión gratuita de Microsoft SQL (MSSQL Express Edition 2012), que albergará una base de datos llamada *TOS*. Esta base de datos contendrá una tabla bajo el nombre de *vessel_visit,* que contendrá un listado de todas las visitas de los barcos que operan en esa ubicación. Por otro lado, y como requerimiento para la implementación, se va a ubicar geográficamente la

terminal como ejemplo en el puerto de Valencia. Esta será la estructura de la tabla, incluyendo algunos atributos genéricos habituales en cualquier TOS existente en el mercado:

	Tipo de dato	Descripción
vessel_visit_id	Number	Número e identificador único, como clave primaria de la tabla, que identifica la visita del barco.
vessel_visit_name	String	Nombre que recibe la visita del barco.
vessel_name	String	Nombre de la embarcación.
operator	String	Nombre del operador propietario del barco.
eta	Datetime	Tiempo estimado de llegada.
etd	Datetime	Tiempo estimado de salida.
etc	Datetime	Tiempo estimado en el que se completarán las operaciones sobre el barco.
classification	String	Clasificación o tipo de barco.

TABLA 2 – Descripción de la base de datos que simula el TOS.
Fuente: Elaboración propia.

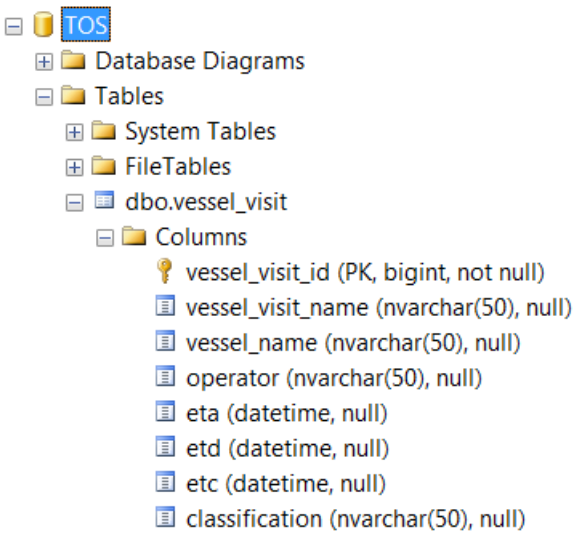

FIGURA 18 – Vista de la base de datos en MSSQLServer para el TOS.
Fuente: Elaboración propia.

- **Empresa naviera:** En este caso, se va a asumir que la empresa naviera trabaja con un ESB (Enterprise Service Bus) a través del cual se gestionan todas las comunicaciones internas. También dispone de un directorio activo accesible a través de toda la red, en el que cada terminal portuaria tiene asignada una carpeta. A través del bus de comunicaciones se deposita un fichero .xml para cada visita planificada de un barco. El formato del fichero .xml en el caso de una visita de un barco se detalla a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<vessel_visit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="889923" xsi:vesselVisitSchemaLocation="vessel_visit.xsd">
	<vessel_name>Morten Maersk</vessel_name>
	<destination>Noatum Valencia</destination>
	<eta>2019-03-21T10:24:00</eta>
	<etc>2019-03-21T16:06:00</etc>
	<etd>2019-03-30T16:30:00</etd>
	<service>
		<name>ECS EUROPE</name>
		<inbound_voyage>0EU1RS1MA</inbound_voyage>
		<outbound_voyage>0EU1RS1MA</outbound_voyage>
	</service>
</vessel_visit>
```

- **Autoridad portuaria:** Para el ejemplo en cuestión, la autoridad portuaria del puerto de Valencia dispone de un software y una cuenta en MarineTraffic. Este software realiza diferentes llamadas a su RestAPI, procesa y almacena todas las visitas de los barcos para obtener una fecha estimada de llegada más aproximada gracias al sistema de antenas AIS. Se va a emplear una cuenta registrada en MarineTraffic para llamar al método VI02 de su API, llamado también “expected arrivals”, de manera que se puedan obtener los siguientes datos de los barcos que esperan atracar en el puerto de Valencia:
  - AIS: Datos sobre el sistema de antenas equipado en el barco que se comunica con la central de datos de MarineTraffic.
  - POSICIÓN: Latitud y longitud actuales del barco.

- NAVEGACIÓN: Datos de navegación del barco, como velocidad, curso o similares.
- SERVICIO: Origen del barco, destino, tiempo estimado de llegada reportado y tiempo estimado de llegada calculado.

Estos son los detalles específicos del método de la API que se va a invocar, que obtiene las llegadas estimadas en un puerto o localización específicas:

- Parámetros de entrada:

	Obligatorio	Tipo	Defecto	Descripción
[NO NAME]	Sí	String		Clave de la API. Número hexadecimal de 40 caracteres.
TIMESPAN	No	Number		Número de días a partir del día actual para recibir visitas.
DAYS_LAST_SIGNAL	No	Number	2	Incluir barcos durante X días que han transmitido su posición al puerto.
FROM_DATE / TO_DATE	No	Datetime		Fecha inicio y fin entre las cuales recibir las visitas esperadas.
MSG_TYPE	No	String	Simple	Tipo de mensaje, simple o extendido.
PROTOCOL	No	String	XML	Tipo de respuesta: XML, CSV, JSON.
PORT_ID	No	String		ID único identificativo del puerto según Marine Traffic.
COUNTRY	No	String		País al que pertenece el puerto.
FROM_PORT_ID	No	String		ID del puerto anterior, según Marine Traffic.
FROM_COUNTRY	No	String		País del puerto anterior.
SHIP_TYPE	No	Number		Tipo de barco, empleando clasificación estándar de Marine Traffic.
DWT_MIN /	No	Number		Filtro según el tamaño de carga y el

DWT_MAX				tipo de IMO.
GT_MIN / GT_MAX	No	Number		Filtro específico GT para diferentes tipos de IMO.

TABLA 3 – Parámetros de entrada para solicitar información a la autoridad portuaria.
Fuente: Marine Traffic - Elaboración propia.

- URL: https://services.marinetraffic.com/api/expectedarrivals/v:3
- Ejemplo de respuesta (JSON):


```
[["240391000","38.004240","23.522880","102","281","3","","","","","1","GRPIR","PIRAEUS","GR","2014-02-21T13:18:00","2014-02-20T12:18:00","2014-02-20 11:55:00","2014-02-20T11:48:45"],["538004631","37.592600","23.897660","120","279","0","","","","","1","GRPIR","PIRAEUS","GR","2014-02-22T13:44:00","2014-02-20T13:44:00","2014-02-20 09:33:00","2014-02-20T11:57:22"],["229637000","37.478790","23.734610","137","8","0","","","","","1","GRPIR","PIRAEUS","GR","2014-02-23T14:13:00","2014-02-20T14:13:00","2014-02-20 10:22:00","2014-02-20T11:58:02"]]
```


FIGURA 18 – Captura de un mensaje de la API de Marine Traffic.
Fuente: Marine Traffic.

- Parámetros de la respuesta:

	Tipo de dato	Descripción
MMSI	Number	Identidad del servicio móvil marítimo. Número de nueve dígitos enviado a través de una frecuencia de radio que identifica la estación transmisora.
LAT	Number	Latitud del barco (posición de Norte a Sur).
LON	Number	Longitud del barco (posición de Este a Oeste).
SPEED	Number	Velocidad del barco en nudos.
COURSE	Number	Curso del barco en grados.
STATUS	String	Estado del barco acorde al estándar AIS.
PORT_UNLOCODE	String	ID único identificativo del puerto siguiendo el estándar de Naciones Unidas.
CURRENT_PORT	String	Nombre del puerto. NULL si el barco está navegando.
CURRENT_PORT_C	String	País al que pertenece el puerto actual.
NEXT_PORT_UNLOCODE	String	ID único identificativo del siguiente puerto de destino, acorde al estándar de Naciones Unidas.

NEXT_PORT_ID	Number	ID único identificativo del siguiente puerto de destino, acorde al estándar de Marine Traffic.
NEXT_PORT_NAME	String	Nombre del siguiente puerto de destino.
NEXT_PORT_COUNTRY	String	País al que pertenece el siguiente puerto de destino.
ETA	Datetime	Tiempo estimado de llegada al destino según los transmisores AIS.
ETA_CALC	Datetime	Tiempo estimado de llegada al destino según los cálculos realizados por Marine Traffic.
ETA_UPDATED	Datetime	Tiempo de actualización del ETA calculado por Marine Traffic.
TIMESTAMP	Datetime	Tiempo en que la latitud y la longitud ha sido almacenada por Marine Traffic.
PORT_ID	Integer	ID que identifica el puerto actual, asignado por Marine Traffic.
SHIPNAME	String	The shipname of the subject vessel

TABLA 4 – Parámetros de respuesta recibidos desde la autoridad portuaria.
Fuente: Marine Traffic - Elaboración propia.

7.2 Base de datos

Cada una de estas entradas de datos será procesada y almacenada en la base de datos de la aplicación. Por un lado se almacenará cada una de las estimaciones sobre las visitas de los barcos en la tabla *vessel_visit_estimation*, con el tiempo estimado por cada una de las fuentes, la predicción realizada, el tiempo final de llegada, la diferencia y la puntuación obtenida en base a los umbrales establecidos para calcular la fiabilidad de la fuente. Por otro lado se almacenará información sobre cada una de las fuentes empleadas, el número de simulaciones realizadas para cada una y la fiabilidad media obtenida a partir de todas esas pruebas. El modelo de datos unificado tiene la siguiente estructura:

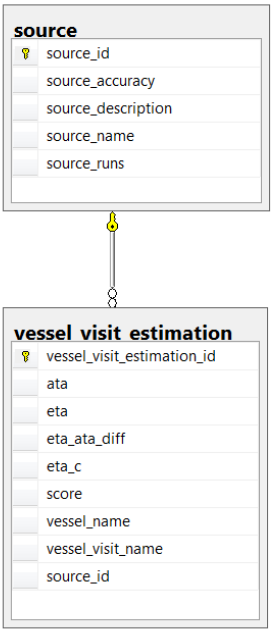

FIGURA 19 – Diseño de la base de datos relacional.
Fuente: Elaboración propia.

Como cada fuente u origen de datos que se conecta al sistema se almacena en una nueva fila, este diseño permite agregar tantos como se desee. A demás cada predicción recibida desde cada una de las fuentes genera un registro, por lo que el sistema permite hacer predicciones al margen de que todos los orígenes hayan emitido o no una estimación. Estos son los diferentes atributos propuestos para cada tabla:

	Tipo de dato	Descripción
VESSEL_VISIT_ESTIMATION_ID	Number	Identidad auto-numérica de la estimación recibida sobre la visita del barco.
ATA	Datetime	Tiempo real de llegada del barco.
ETA	Datetime	Tiempo estimado de llegada del barco, según la fuente de datos que emite la predicción.
ETA_ATA_DIFF	Number	Diferencia entre la estimación inicial realizada por la fuente de datos y el tiempo real de llegada.
ETA_C	Datetime	Tiempo estimado de llegada calculado, en

		base a la fiabilidad de los diferentes origenes de datos en el sistema.
SCORE	Number	Puntuación obtenida para esa estimación en base a los umbrales establecidos, una vez se dispone del tiempo real de llegada del barco.
VESSEL_NAME	String	Nombre del barco.
VESSEL_VISIT_NAME	String	Nombre de la visita del barco.
SOURCE_ID	Number	Identificador de la fuente de datos. Relaciona ese registro con la fuente correspondiente.

TABLA 5 – Atributos de la tabla *vessel_visit_estimation* de la base de datos.
Fuente: Elaboración propia.

	Tipo de dato	Descripción
SOURCE_ID	Number	Identidad auto-numérica de la fuente de datos.
SOURCE_ACCURACY	Number	Porcentaje de fiabilidad de la fuente de datos.
SOURCE_DESCRIPTION	String	Descripción de la fuente de datos.
SOURCE_NAME	String	Nombre de la fuente de datos.
SOURCE_RUNS	Number	Número de simulaciones en las que la fuente de datos en cuestión ha emitido una predicción.

TABLA 6 – Atributos de la tabla *source* de la base de datos.
Fuente: Elaboración propia.

Con esta estructura, una ejecución para estimar el tiempo de llegada del barco en un escenario con tres fuentes de datos en el que cada una de ellas realiza una predicción, supondría almacenar tres registros en la tabla *vessel_visit_estimation*. Estos tres registros contendrían tres estimaciones diferentes para el mismo barco, pero que provienen de tres sistemas distintos. La tabla *source* contendría otros tres registros, uno por cada origen de datos.

7.3 Arquitectura y tecnologías empleadas

A partir de estos tres inputs, el software contendría un manejador independiente para cada uno de ellos, bajo el nombre de *RestInputHandler*, *XMLInputHandler* y *DBInputHandler*. El primero de ellos se encargaría de realizar la llamada a la RestfulAPI que emplea la autoridad portuaria, y convertiría en objetos la información recibida para ser empleada por el programa. De la misma forma el segundo manejador realizaría la lectura de los ficheros XML correspondientes de la empresa naviera y los convertiría a objetos, y el tercero haría lo mismo sobre la base de datos del Terminal Operating System de la terminal de contenedores. Estos manejadores se van a programar de manera que si se añadiera un cuarto sistema que emplease alguno de estos formatos de datos (XML, RestAPI o Base de Datos), sería muy sencillo extender su funcionalidad, reutilizar el código e instanciarlo para conectar el nuevo origen de datos. Una vez las entradas han sido recibidas, la información estaría lista para ser procesada, persistida y transformada en la salida del sistema. Así, tal y como se refleja en la siguiente figura, la estructura queda claramente definida en un conjunto de entradas, un procesamiento y una serie de salidas:

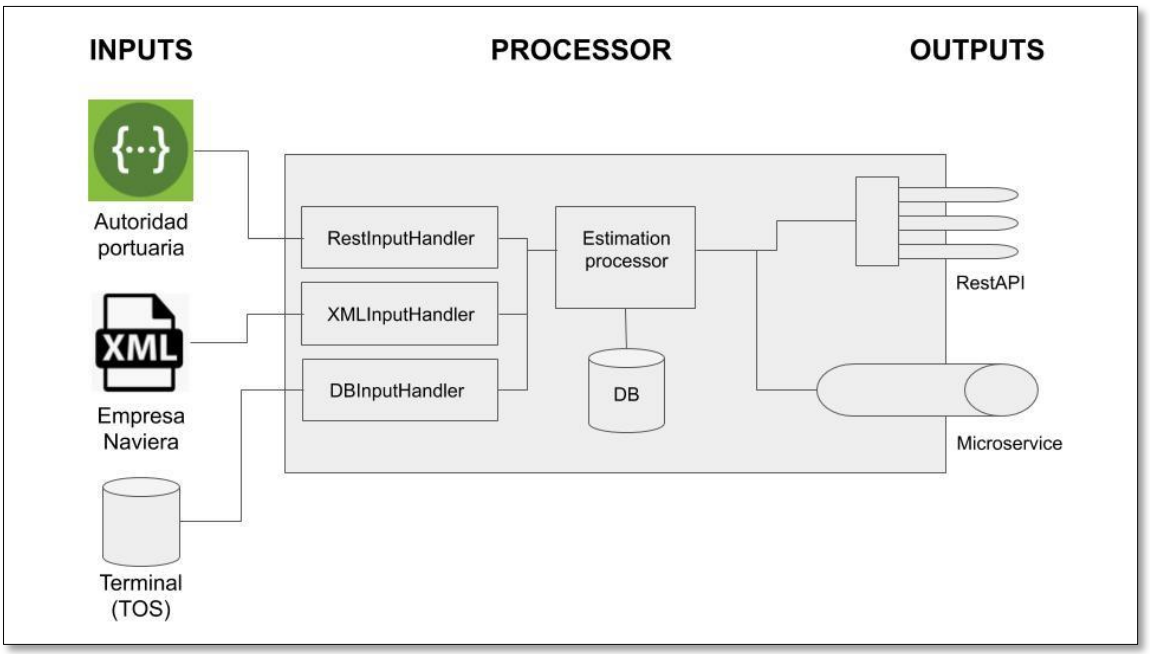

FIGURA 20 – Diseño del sistema predictivo para el tiempo estimado de llegada de los barcos.
Fuente: Elaboración propia.

En cuanto al procesador de datos respecta, se propone desplegar en un servidor local Apache Tomcat, a modo de backend, un software programado en Java, dado que para el uso de todas estas tecnologías no resulta necesario adquirir ninguna licencia. Se propone emplear Springboot como estructura principal del proyecto, a través de los templates de Gradle, y también JPA (Java Persistence API) como modelo de persistencia y acceso a la base de datos.

La salida de los datos combinará tanto un micro servicio como una RestAPI, que servirán los resultados por un lado de la fiabilidad de cada una de las fuentes, y por otro del tiempo estimado de llegada que se predice para cada uno de los barcos disponibles en el sistema. La ventaja de incluir ambas implementaciones es que va a permitir experimentar con ellos a la hora de obtener los resultados, e incluir conclusiones adicionales en el siguiente capítulo. Esto también permitirá realizar mejores decisiones en un futuro a la hora de diseñar una arquitectura y tomar decisiones en cuanto a cuál es la mejor implementación y/o tecnología disponible.

Este es un resumen de todo el conjunto de tecnologías y lenguajes de programación que se van a utilizar:

- **Inputs**: XML, RestAPI, MSSQL Server Express Edition
- **Processor**: Apache Tomcat, Java, Springboot, Gradle, JPA
- **DataModel**: MSSQL Server Express Edition
- **Outputs**: Kafka, RestAPI, JSON
- **Ide**: IntelliJ IDEA

7.4 Distribución del proyecto y entidades

La estructura básica del proyecto viene establecida principalmente por SpringBoot y Gradle. La clase *BackendApplication* es la clase principal que define el punto inicial de acceso del programa, que a su vez inicia *VesselVisitPredictorBootstrap* como clase de arranque de la aplicación, que ejecuta las operaciones iniciales. En esta clase

fundamentalmente se inician todos los manejadores, la RestAPI, el micro servicio y los accesos a base de datos, y también se comprueba si existen datos o es la primera ejecución, inicializando así las tres fuentes de entrada por defecto. La RestAPI y el micro servicio se encuentran ambos implementados dentro de la carpeta *controllers*, distribuidos en un total de tres clases.

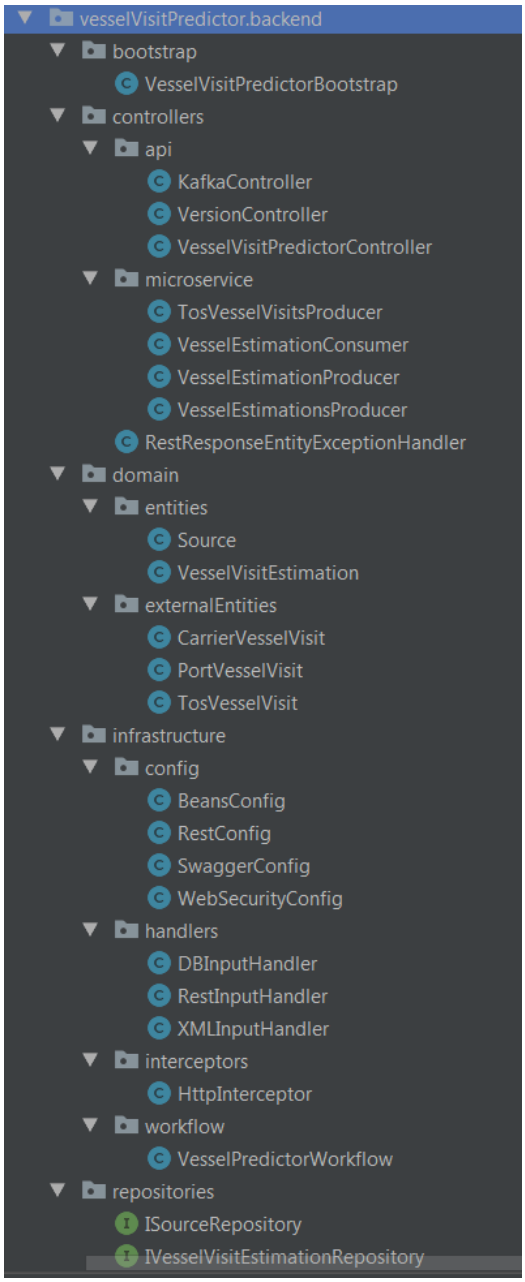

FIGURA 21 – Captura de la estructura del proyecto en IntelliJ IDEA.
Fuente: Elaboración propia.

La carpeta *domain* contiene todas las entidades y objetos del proyecto, divididos a su vez en dos carpetas, según sean entidades de la propia base de datos (*Source* y *VesselVisitEstimation*) o entidades que mapean cada una de las tres fuentes de datos existentes (*CarrierVesselVisit*, *PortVesselVisit* y *TosVesselVisit*). El primer conjunto utiliza JPA e hibernate para conectar con la base de datos, y el segundo conjunto emplea una conexión JDBC convencional a la base de datos, ficheros XML o una consulta a la RestAPI de la autoridad portuaria, dependiendo de cada dato. La estructura de estos cinco tipos de objetos ya ha sido descrita en los anteriores sub-apartados correspondientes a la Base de Datos y a las Entradas de Datos. En grandes proyectos e infraestructuras es una práctica muy habitual emplear dos tipos diferentes de entidades para cada objeto, siendo una entidad la que se persiste en base de datos y otra la que se transporta a través de la RestAPI. Este segundo tipo se conoce como DTO o *DataTransferObject*, y un mapper interno se encarga de traducir de un objeto a otro. Este patrón de diseño es muy útil a la hora de personalizar entradas y salidas de datos y reducir el número de métodos disponibles en una librería. No obstante, debido a la simplicidad de las estructuras de datos y las llamadas disponibles que se manejan en este proyecto, se ha optado por emplear la misma definición de entidad tanto para base de datos como para la RestAPI.

La carpeta *infrastructure* contiene cuatro carpetas. La primera de ellas contiene la configuración general del proyecto y la integración con swagger, así como diferentes configuraciones de control de acceso para HTTP y la RestAPI. La segunda carpeta contiene los tres manejadores de las fuentes de entrada: Los datos de la terminal, los datos de la naviera y los datos de la autoridad portuaria. Una tercera carpeta contiene información sobre el interceptor HTTP, y finalmente la lógica principal para el cálculo de estimaciones en la carpeta *workflow*, en la clase *vesselPredictor*.

La carpeta *repositories* contiene las interfaces de los repositorios para acceder a la base de datos del programa, que extienden del repositorio estándar para JPA *CrudRepository*, y la carpeta *services* contiene la definición de los servicios y la imlementación para el acceso a base de datos a través de hibernate.

7.5 Configuración de la aplicación

A demás de las clases y directorios anteriores, la aplicación tiene un archivo de configuración llamado *application.yml*, que contiene todos los parámetros que deben ser configurados antes de ejecutar la misma. Se engloban en las siguientes categorías:

- **Servidor:** Establece el puerto en el que se servirá la aplicación, tanto si se ejecuta en modo local como si se despliega en un servidor Apache Tomcat o similar. Se emplea por defecto el puerto 5000.
- **Detalles:** Contiene una breve descripción del software así como detalles sobre la versión y las *ReleaseNotes*, si las hubiere. Generalmente en grandes equipos de trabajo también suele incluir el nombre de la rama del repositorio que genera la versión actual.
- **Log:** Configuración del log de la aplicación. Establece el nombre de los archivos, la ruta, el nivel por defecto en el que se escriben los mensajes del log y el patrón de cada línea de texto que se imprime, así como el formato de la fecha y la separación entre los mensajes.
- **Spring:** Configuración principal de Springboot. Perfil por defecto activado y tipo de actualización sobre la base de datos al ejecutarse. Datos de conexión a la base de datos propia del software: Dirección del servidor, nombre de la base de datos, usuario y contraseña de la base de datos gestionada a través de hibernate.
- **Sistema de archivos:** Detalles para permitir el acceso a los archivos de la empresa naviera, como primera entrada de datos del programa. Ruta o directorio activo de los ficheros y diferentes configuraciones sobre la estructura de los archivos, así como formato de las fechas y otros tipos de datos.
- **Base de datos:** Datos para conectar a la base de datos del TOS de la terminal de contenedores, como la dirección del servidor, nombre de la base de datos y de la tabla, usuario o contraseña, entre otros.
- **RestApi:** Datos para acceder a la RestAPI de la autoridad portuaria. Nombre de los métodos que se invocan, autenticación y credenciales para acceder a los mismos.

- **Microservicio:** Información sobre el despliegue del productor y el consumidor del microservicio Kafka, con ambas rutas, puertos utilizados, estrategia para offsets y la clave y valor para serializar o deserializar los mensajes.

7.6 Inicio

La clase *VesselVisitPredictorBootstrap.java* se ejecuta cada vez que se inicia el servidor Apache Tomcat o cada vez que se compila el proyecto. Antes de esta ejecución, en cualquier proyecto de tipo Sprinboot se comprueba el acceso a la base de datos y se crean todas las entidades definidas con hibernate (JPA), en caso de que no existan. Por otro lado cuando se trata de una actualización sobre una misma base de datos, el modelo de datos se actualizaría de manera automática, reflejando todos los cambios que se hayan producido en las entidades. Si por ejemplo se han añadido dos atributos nuevos a una entidad, la tabla de la base de datos a la que ésta conectada la aplicación se ampliaría incluyendo estas dos nuevas columnas.

Ya que todas las consultas y operaciones que realiza el programa se efectúan bajo demanda a través de métodos de la RestAPI, el inicio de la aplicación consiste simplemente en comprobar que se puede acceder a la base de datos y que las tablas han sido creadas correctamente. Una vez realizado este proceso, se comprueba si existen registros disponibles en la tabla *Source*, y si no existen, se crean los tres orígenes de datos que se utilizarán en el proyecto: La naviera, la terminal de contenedores y la autoridad portuaria:

```
@Value("${version}")
private String version;

private final ISourceService sourceService;

@Override
public void onApplicationEvent(ContextRefreshedEvent event) {
        try {
                log.info(version);
                if(!sourceService.existsAny()) {
                        log.info("Creating sources");
                        Source carrierSource = new Source();
                        carrierSource.setSourceName("Carrier");
                        carrierSource.setSourceDescription("File system and active
                                directory used by the carrier");
                        carrierSource.setSourceAccuracy(new Double(0));
                        carrierSource.setSourceRuns(new Long(0));
                        sourceService.addSource(carrierSource);
```

```
                    Source portAuthoritySource = new Source();
                    portAuthoritySource.setSourceName("Port authority");
                    portAuthoritySource.setSourceDescription("Rest API served by the
                            Port authority");
                    portAuthoritySource.setSourceAccuracy(new Double(0));
                    portAuthoritySource.setSourceRuns(new Long(0));
                    sourceService.addSource(portAuthoritySource);
                    Source containerTerminalSource = new Source();
                    containerTerminalSource.setSourceName("Container terminal");
                    containerTerminalSource.setSourceDescription("Terminal operating
                            system used by the container terminal");
                    containerTerminalSource.setSourceAccuracy(new Double(0));
                    containerTerminalSource.setSourceRuns(new Long(0));
                    sourceService.addSource(containerTerminalSource);
                }
            }
            catch (Exception ex)
            {
                log.error("Bootstrap error "+ex.getMessage()+" " +ex.getStackTrace());
            }
}
```

Una vez ha finalizado el inicio de la aplicación, se dispone del acceso a la base de datos así como de las tres fuentes que se van a emplear para realizar estimaciones sobre los tiempos de llegada de cada barco. Al mismo tiempo la RestAPI se encuentra disponible e inicializada para atender las diferentes peticiones HTTP.

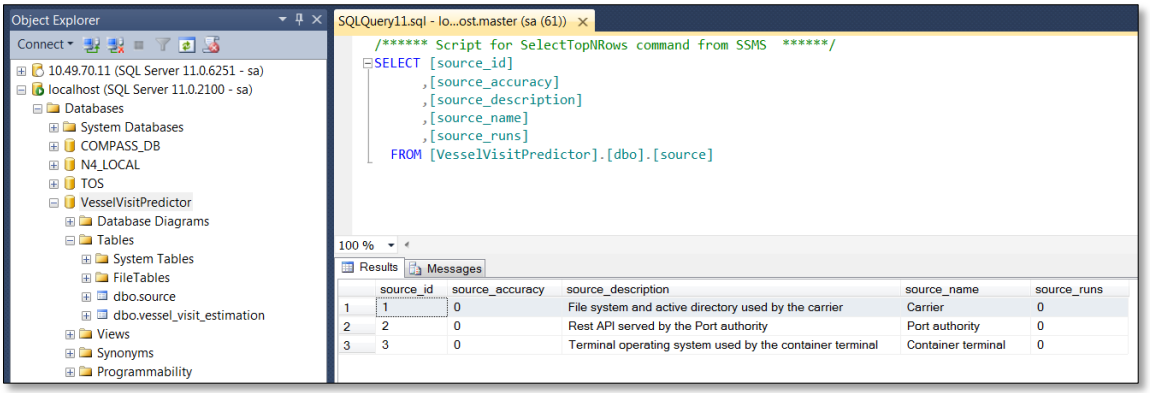

FIGURA 22 – Estado de la tabla SOURCE tras la primera ejecución de la aplicación.
Fuente: Elaboración propia.

7.7 Acceso a los datos a través de JPA - Hibernate

Para el acceso a la base de datos de las entidades propias del software (*Source* y *VesselVisitEstimation*) se emplea JPA e hibernate. Esta implementación suele realizarse a través de un repositorio y un servicio. La clase repositorio extiende de *CrudRepository*, permitiendo acceder a la implementación de muchos métodos por

defecto como *count*(), *remove()*, *save()*… A su vez esta clase puede añadir otros métodos específicos en función de la definición de esta entidad, de las relaciones que tenga con otras entidades o de otras características. Se debe definir una interfaz del repositorio para cada una de las entidades existentes, de manera individual:

```
package vesselVisitPredictor.backend.repositories;

import org.springframework.data.repository.CrudRepository;
import vesselVisitPredictor.backend.domain.entities.Source;

import java.util.List;

public interface ISourceRepository extends CrudRepository<Source, Long>{
        List<Source> findAll();
        Source findBySourceId(Long sourceId);
        Source deleteBySourceId(Long sourceId);
}
```

Por otro lado se define una interfaz para el servicio de acceso a los datos para cada entidad. De esta forma, el servicio puede ser inyectado en cada clase del proyecto sin necesidad de instanciarlo, tal y como se realiza por ejemplo en la clase de inicio del subcapítulo anterior para crear los orígenes de datos. En la aplicación desarrollada y para la entidad *Source*, el servicio ofrece métodos para obtener todos los orígenes de datos, obtener uno en concreto a través de su ID, borrar un origen de datos indicando el ID o añadir un nuevo origen de datos. Esta última llamada se puede emplear tanto para crear un nuevo registro como para actualizar un registro ya existente:

```
package vesselVisitPredictor.backend.services.interfaces;

import vesselVisitPredictor.backend.domain.entities.Source;

import java.util.List;

public interface ISourceService {
        List<Source> getSources();
        Source getSourceById(Long sourceId);
        boolean deleteSourceById(Long sourceId);
        boolean existsAny();
        Source addSource(Source source);
}
```

Finalmente es necesaria una clase que implemente esta interfaz, haciendo uso del repositorio que se ha definido anteriormente:

```
package vesselVisitPredictor.backend.services;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import vesselVisitPredictor.backend.domain.entities.Source;
import vesselVisitPredictor.backend.repositories.ISourceRepository;
import vesselVisitPredictor.backend.services.interfaces.ISourceService;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class SourceService implements ISourceService {

	private final ISourceRepository sourceRepository;

	@Override
	public List<Source> getSources() {
		return this.sourceRepository.findAll();
	}

	@Override
	public Source getSourceById(Long sourceId) {
		return this.sourceRepository.findBySourceId(sourceId);
	}

	@Transactional
	@Override
	public boolean deleteSourceById(Long sourceId) {
		try {
			this.sourceRepository.deleteBySourceId(sourceId);
			return true;
		} catch (Exception ex) {
			log.error("deleteSourceById | unable to delete source " + sourceId
				+ " Error: " + ex.getMessage());
			return false;
		}
	}

	@Override
	public boolean existsAny() {
		return this.sourceRepository.count() > 0;
	}

	@Override
	public Source addSource(Source source) {
		return this.sourceRepository.save(source);
	}
}
```

Esta estructura se repetiría de la misma forma para cada una de las entidades de la base de datos que se gestionan a través de hibernate, trazando el siguiente diagrama de clases:

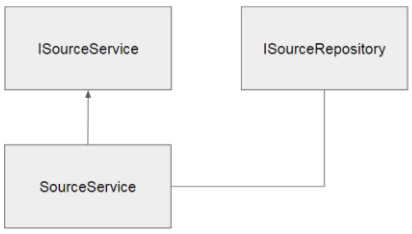

FIGURA 23 – Esquema de distribución de clases para el acceso a base de datos con JPA – Hibernate.
Fuente: Elaboración propia.

7.8 Cálculo de estimaciones

El primer método propuesto para el cálculo de estimaciones se va a efectuar mediante una media ponderada. La media ponderada es una medida de tendencia central que se emplea sobre un conjunto de datos en los que cada uno tiene una importancia o peso relativos respecto de los demás datos. En este caso el peso dependerá de la fiabilidad que tenga cada una de las fuentes que estime el tiempo de llegada del barco, determinado a su vez por todas las ejecuciones anteriores. El cálculo se realiza multiplicando cada uno de los datos por su ponderación o peso, para luego sumarlos. Esta suma se dividirá entre la suma total de los pesos, dando como resultado la media ponderada. Así pues, se dispone por un lado de un conjunto de datos numéricos no vacíos, para el ejemplo los tiempos de llegada del barco estimado para cada una de las fuentes:

$$X = \{x_1, x_2, x_{3\ldots}, x_n\}$$

En función de a qué fuente pertenezca, cada dato tiene asignado un peso calculado a partir de ejecuciones anteriores:

$$W = \{w_1, w_2, w_{3\ldots}, w_n\}$$

La media ponderada se calcula de la siguiente manera:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

$$\bar{x} = \frac{x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n}{w_1 + w_2 + w_3 + \cdots + w_n}$$

En definitiva se trata de una métrica definida por los pesos. La media ponderada se comporta geométricamente que una media sin ponderar, quedando definida por el pie de la perpendicular desde X hasta la recta <(1,1, … , 1)>. La perpendicular desde X a la recta <(1,1, … , 1)> cae en (x,x, … , x):

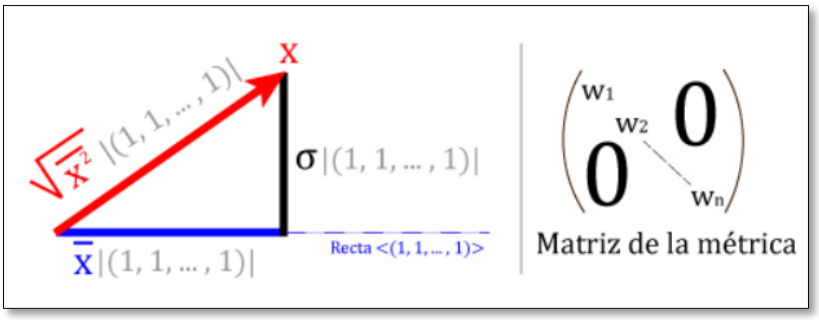

FIGURA 24 – Comportamiento geométrico de una media ponderada.
Fuente: Wikipedia.

Por tanto la siguiente formula es válida para la media con o sin ponderación, ya que la única diferencia es la métrica que se considera:

$$\bar{x} = \frac{x * (1,1,1 \dots, 1)}{(1,1,1 \dots, 1) * (1,1,1 \dots, 1)}$$

Como segundo método de predicción propuesto se propone una mejora de la media ponderada efectuada anteriormente, que afecta al cálculo de los pesos de cada origen de datos. Dado que no todas las fuentes disponen de una predicción para cada visita de un barco, además de la fiabilidad existe un contador de predicciones emitidas. En este segundo método se van a calcular los pesos de manera distinta, de forma que se tenga en cuenta el número de ejecuciones que lleva acumulados cada origen de datos. Si por ejemplo el TOS ha realizado un total de 10 estimaciones obteniendo una fiabilidad del 90%, y la naviera ha realizado un total de 50 estimaciones obteniendo la misma fiabilidad que el TOS, será mucho más relevante a la hora de realizar la predicción conjunta la estimación que realice la naviera, ya que con la misma fiabilidad que la terminal de contenedores, esta ha realizado un total de cinco veces más de ejecuciones. Al igual que en el método anterior, dispondríamos de un conjunto de datos numéricos no vacíos, y de un conjunto de pesos para cada una de las fuentes que proporcionan esos datos:

$$X = \{x_1, x_2, x_{3\ldots,} x_n\}$$

$$W = \{w_1, w_2, w_{3\ldots,} w_n\}$$

Como elemento adicional, dispondríamos de un número de iteraciones o repeticiones para cada fuente de datos disponible:

$$C = \{c_1, c_2, c_{3\ldots,} c_n\}$$

Por lo tanto la fórmula para calcular esta media ponderada mejorada sería la siguiente:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i w_i c_i}{\sum_{i=1}^{n} w_i c_i}$$

$$\bar{x} = \frac{x_1 w_1 c_1 + x_2 w_2 c_2 + x_3 w_3 c_3 + \cdots + x_n w_n c_n}{w_1 c_1 + w_2 c_2 + w_3 c_3 + \cdots + w_n c_n}$$

En cuanto al software de predicción, la clase *VesselPredictorWorkflow* contiene la implementación de ambos métodos bajo el nombre *perdictVesselArrivalTime* y *predictVesselArrivalTimeImproved*. El primero de ellos calcula la media ponderada estándar, acumulando las estimaciones en la variable *totalEstimations* y los pesos en la variable *totalAccuracies*, para devolver finalmente la estimación a través del objeto *VesselVisitEstimation.* Para todos los cálculos con fechas se emplea la función *getTime()* de Java, que obtiene la distancia en milisegundos desde el 1 de Enero de 1970 UTC. Esto permite tener una representación numérica de cualquier fecha emitida:

```
public VesselVisitEstimation predictVesselArrivalTime(String vesselName) {
        List<VesselVisitEstimation> vesselVisitEstimations =
                vesselVisitEstimationService.getVesselVisitEstimationsByVesselName(
                vesselName);
         Long totalEstimations = new Long(0);
         Long totalAccuracies = new Long(0);
         for(VesselVisitEstimation vesselVisitEstimation : vesselVisitEstimations) {
             Source source = sourceService.getSourceById(
                vesselVisitEstimation.getSource().getSourceId());
             totalEstimations += (vesselVisitEstimation.getEta().getTime() *
                source.getSourceAccuracy());
             totalAccuracies += source.getSourceAccuracy();
         }
         Date estimatedArrivalTime = new Date((totalEstimations * totalAccuracies) /
                totalAccuracies);
         VesselVisitEstimation rEstimation = new VesselVisitEstimation();
```

```
        rEstimation.setVesselVisitEstimationId(new Long(-1));
        rEstimation.setVesselName(vesselName);
        rEstimation.setEta(estimatedArrivalTime);
        return rEstimation;
}
```

El segundo método que tiene en cuenta las predicciones realizadas por cada fuente funciona de la misma forma, incluyendo la variable *totalIterations* para acumular el total de predicciones de todas las fuentes disponibles:

```
public VesselVisitEstimation predictVesselArrivalTimeImproved(String vesselName) {
        List<VesselVisitEstimation> vesselVisitEstimations =
               vesselVisitEstimationService.getVesselVisitEstimationsByVesselName(
               vesselName);
        Long totalEstimations = new Long(0);
        Long totalAccuracies = new Long(0);
        Long totalIterations = new Long(0);
        for(VesselVisitEstimation vesselVisitEstimation : vesselVisitEstimations) {
            Source source = sourceService.getSourceById(
               vesselVisitEstimation.getSource().getSourceId());
            totalEstimations += (vesselVisitEstimation.getEta().
               getTime() * source.getSourceAccuracy());
            totalAccuracies += source.getSourceAccuracy();
            totalIterations += source.getSourceRuns();
        }
        Date estimatedArrivalTime = new Date((totalEstimations * totalAccuracies *
               totalIterations)/ (totalAccuracies * totalIterations));
        VesselVisitEstimation rEstimation = new VesselVisitEstimation();
        rEstimation.setVesselVisitEstimationId(new Long(-1));
        rEstimation.setVesselName(vesselName);
        rEstimation.setEta(estimatedArrivalTime);
        return rEstimation;
}
```

En cuanto a la RestAPI desarrollada, son dos los métodos que devuelven estimaciones individuales o de un barco en concreto. El método *getVesselVisitEstimationTime*, que se sirve al exterior bajo la url */VesselVisitEstimation/{*vesselName*}*, permite enviar como parámetro el nombre del barco. Una vez es invocado accederá a la base de datos para obtener todas las estimaciones de llegada para ese barco existentes entre las diferentes fuentes. A partir de la fiabilidad de cada una de ellas, y a través de los métodos definidos anteriormente, devolverá una predicción más aproximada a partir de la estimación y la combinación y el uso de varios orígenes de datos. Por otro lado el método *getVesselVisitEstimations*, servido al exterior a través del recurso */VesselVisitEstimations/{vesselName}*, obtiene un listado de todas las estimaciones individuales de cada origen de datos que existen en el sistema para ese barco.

7.9 RestAPI

La RestAPI se divide en tres controladores. El primero de ellos recibe el nombre de *VersionController*, y simplemente sirve un método que permite conocer la versión que se ha publicado en el servidor y que devuelve una respuesta si el software se está ejecutando correctamente. En diferentes entornos en la nube esta llamada recibe el nombre de "health check". Tras ejecutar el proyecto en el servidor local, si se hace una llamada a la url /api/v se obtiene el siguiente resultado en el navegador:

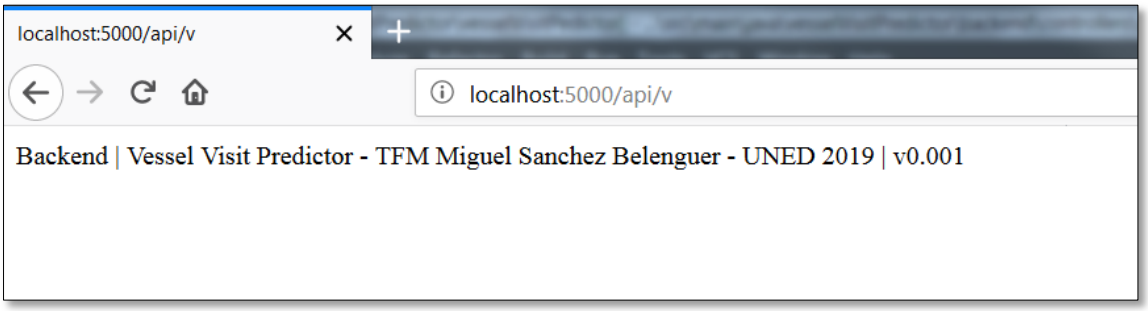

FIGURA 25 – Health check o llamada de control del backend.
Fuente: Elaboración propia.

El segundo controlador recibe el nombre de *VesselPredictorController*, y es el que se encarga de servir todos los métodos que permiten acceder a la información procesada por el software. Estos métodos son los siguientes:

- **TosVesselVisits** – Este método realiza una lectura de todas las visitas de los barcos registradas en el TOS y las devuelve por pantalla, una vez se han mapeado todos los objetos. A demás, transforma todos estos objetos a estimaciones de la base de datos y crea todos los registros asignando el origen de datos correspondiente (en este caso, la terminal de contenedores).

- **CarrierVesselVisits** – Realiza una lectura de todos los ficheros disponibles de la naviera, y devuelve todas las visitas de los barcos planificadas. Al igual que el método anterior, convierte estos objetos en estimaciones que serán almacenadas en la base de datos para el origen de datos de la naviera.

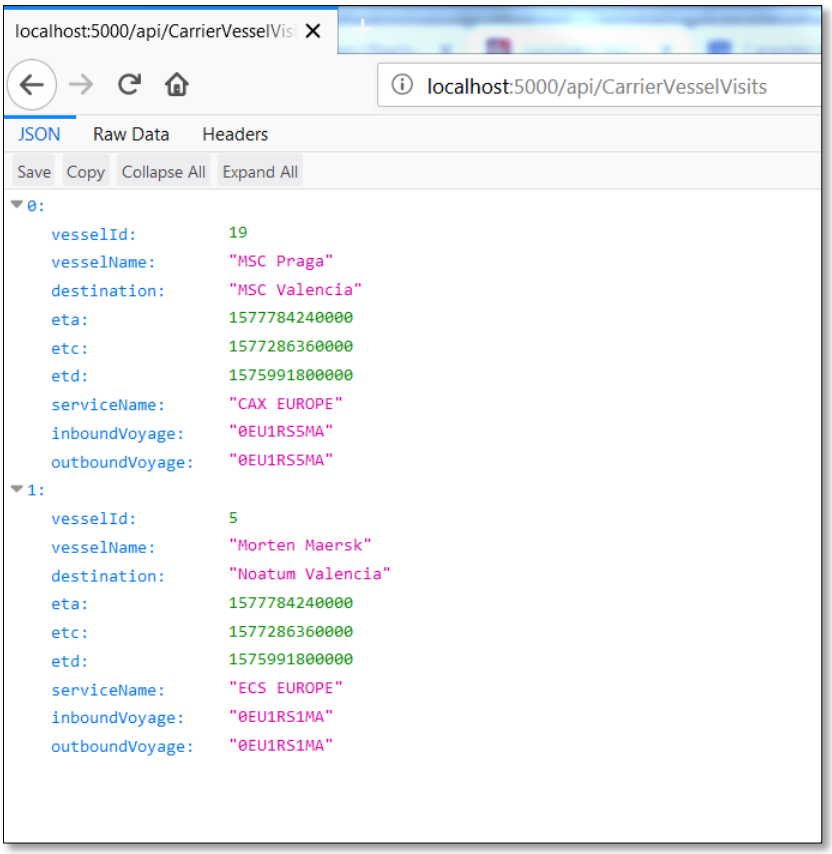

FIGURA 26 – Llamada al método CarrierVesselVisits de la RestAPI.
Fuente: Elaboración propia.

- **PortVesselVisits** – Realiza una llamada a la RestAPI de la autoridad portuaria, procesa todas las visitas previstas y las devuelve con el formato correspondiente. Convierte estas visitas a objetos *VesselVisitEstimation* y los almacena en la base de datos como estimaciones para el origen de datos de la autoridad portuaria.

- **Sources** – Obtiene todas las fuentes de datos disponibles y los datos asociados a ella, así como la fiabilidad y el número de ejecuciones realizadas.

- **SetVesselArrivalTime** – Cuando ha llegado un barco a la terminal, invocar este método permite almacenar en base de datos el tiempo real de llegada de un

barco, y así recalcular la fiabilidad de cada una de las fuentes, en base al error que ha cometido cada una de ellas.

- **VesselVisitEstimation** – Este método recibe como parámetro en la url el nombre de un barco, y a partir de ahí lee todas las fuentes de datos disponibles y calcula cuál será su tiempo de llegada estimado. A su vez persiste esta información en base de datos y devuelve el resultado junto con la estimación.

- **VesselVisitEstimations** – Devuelve todas las estimaciones individuales disponibles para cada origen de datos de un barco determinado.

- **VesselVisitArrivalTime** – Introduce en la base de datos el tiempo de llegada real de un barco, y a partir de todas las entradas re-calcula las fiabilidades registradas de cada una de las fuentes disponibles.

Por otro lado se ha incluido una integración con *Swagger*, considerado como software estándar para documentar una RestAPI por muchas empresas en el mercado. Es posible integrarlo con diferentes lenguajes de programación, incluyendo Java y SpringBoot, con los que se ha desarrollado el backend. La ventaja es que una vez integrado en el proyecto, cualquier método nuevo que se programe actualizará directamente la documentación, sin necesidad de realizar esto manualmente. En escenarios reales, resulta de gran ayuda cuando diferentes empresas utilizan una misma API, de manera que garantiza que la documentación siempre está actualizada con la última versión disponible. A demás de ofrecer una descripción de cada uno de los métodos, también obtiene de manera automática el tipo de objeto que devuelve cada método o los argumentos y parámetros, así como una descripción y definición del objeto en sí. Para llevar a cabo esta integración, es necesaria la siguiente clase:

```
package vesselVisitPredictor.backend.infrastructure.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
```

```
import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerConfig extends WebMvcConfigurationSupport {

	@Bean
	public Docket productApi() {
		return new Docket(DocumentationType.SWAGGER_2)
				.select()
				.apis(RequestHandlerSelectors.basePackage("vesselVisitPredictor.ba
					ckend.controllers.api")
				.build()
				.apiInfo(metaData());
	}


	@Override
	protected void addResourceHandlers(ResourceHandlerRegistry registry) {
		registry.addResourceHandler("/swagger-ui.html")
				.addResourceLocations("classpath:/META-INF/resources/");
		registry.addResourceHandler("/webjars/**")
				.addResourceLocations("classpath:/META-INF/resources/webjars/");
	}

	private ApiInfo metaData() {
		return new ApiInfoBuilder()
				.title("VESSEL VISIT PREDICTOR REST API")
				.description("\"REST API for Vessel Visit Predictor\"")
				.version("1.0.0")
				.license("Apache License Version 2.0")
				.licenseUrl("https://www.apache.org/licenses/LICENSE-2.0")
				.build();
	}
}
```

Todas las clases incluidas en la ruta *vesselVisitPredictor.backend.controllers* van a ser accedidas por el plugin, que generará automáticamente la descripción de la API en la url http://localhost:5000/swagger-ui.html. Al mismo tiempo, la clase encargada de gestionar la seguridad y los accesos a la API, *WebSecurityConfig.java*, incluye la ruta para autorizar el acceso a este nuevo recurso:

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
	httpSecurity
		.csrf().disable()
		.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELES
		S).and()
		.authorizeRequests()
		.antMatchers("/actuator/**").hasRole("ADMIN")
		.antMatchers(HttpMethod.OPTIONS, "/api/**").permitAll()
		.antMatchers("/tcs/**").permitAll()
		.antMatchers("/api/**").permitAll()
		.antMatchers("/swagger-ui.html#").permitAll()
		.antMatchers("/websocket/**").permitAll()
		.antMatchers("/auth/**").permitAll()
		.antMatchers("/**").permitAll()
		.antMatchers(HttpMethod.OPTIONS, "/**").permitAll()
```

```
                .anyRequest()
                .authenticated();
        httpSecurity.headers().cacheControl();
}
```

Una vez se ejecuta el proyecto en local o se despliega en el servidor Apache Tomcat, el recurso de documentación de Swagger está disponible al igual que cualquier otro método de la RestAPI:

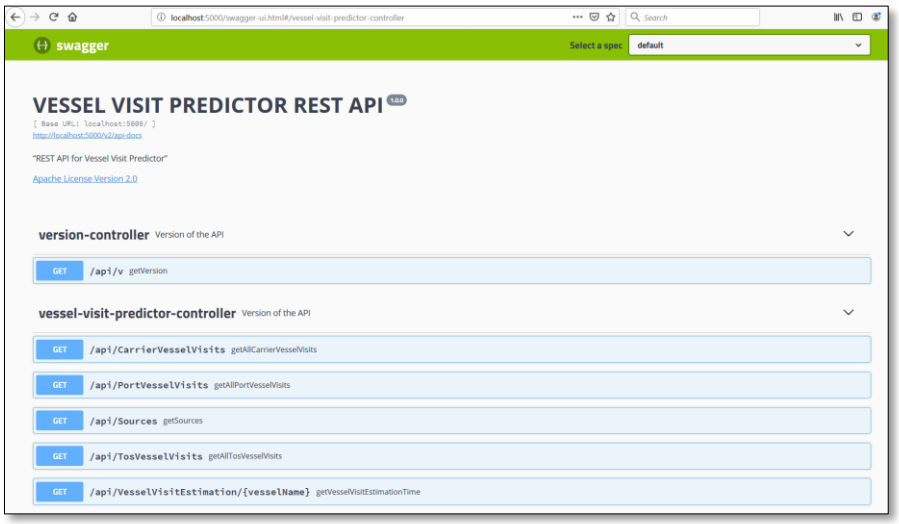

FIGURA 27 – Vista de la integración con Swagger del proyecto desarrollado.
Fuente: Elaboración propia.

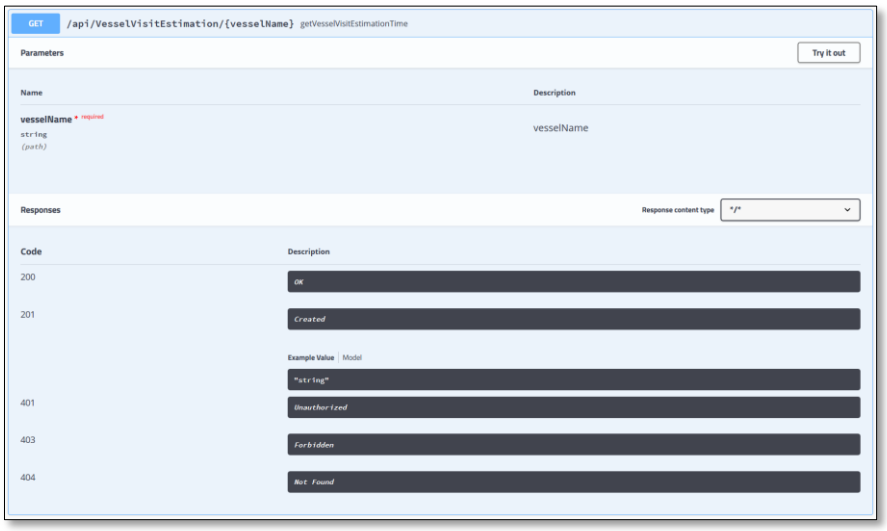

FIGURA 28 – Vista detalle de la definición de un método a través de la integración con Swagger.
Fuente: Elaboración propia.

Es posible realizar llamadas de ejemplo a la API través de esta web, pudiendo enviar los diferentes parámetros de entrada de los métodos y comprobar el formato de los resultados. También es posible también acceder a la descripción de los objetos en la parte inferior.

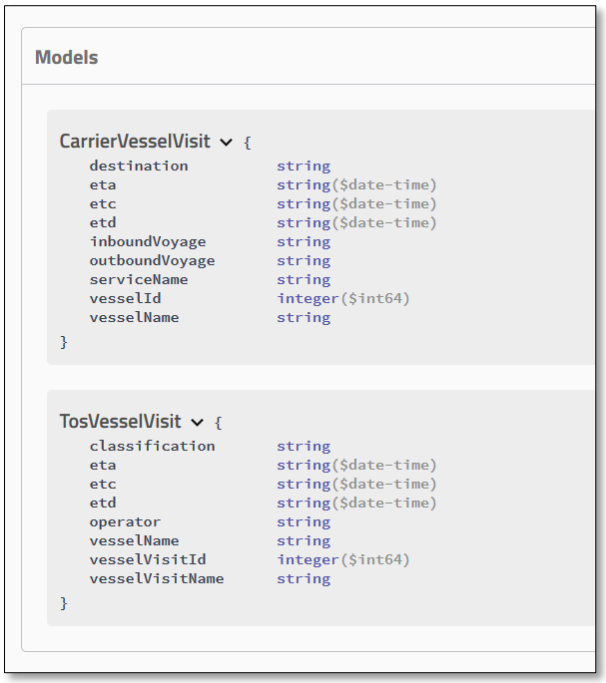

FIGURA 29 – Definición de los objetos DTO en la integración con Swagger.
Fuente: Elaboración propia.

Por otro Swagger también ofrece un lenguaje de programación propio que permite realizar el proceso a la inversa: Comenzar documentando la RestAPI a través de su lenguaje de programación y el editor online (https://swagger.io/tools/swaggerhub/), y una vez está documentado permite realizar una exportación a diferentes lenguajes de programación. En esta exportación se obtiene tanto la definición de los objetos como todas las cabeceras y los tipos de método, ahorrando tiempo de codificación y también esfuerzos duplicados. Generalmente la decisión de utilizar un método u otro depende de la metodología de trabajo y los procesos de cada equipo de desarrollo.

7.10 Microservicio

Para integrar un microservicio en el proyecto se va a utilizar la tecnología Apache Kafka frente a otras alternativas analizadas, debido a su sencillez, potencia e integración en un proyecto Java Springboot. De la misma manera que el sistema es capaz de obtener mediante la RestAPI todas las estimaciones disponibles para el tiempo de llegada de un barco y la estimación mejorada individual, el microservicio va añadir dos canales o "topics" para servir los mismos datos. Se van a implementar tres productores de Kafka y también un consumidor, en el caso de que en una futura ampliación del software alguna fuente u origen de datos sea capaz de transmitir una estimación a través de un microservicio. Por lo tanto se suman un total de cuatro componentes:

- PRODUCTOR (*VesselEstimationsProducer*): A través de un canal o "topic" dedicado, envía todas las estimaciones disponibles para un barco en concreto a través del objeto *VesselVisitEstimation*. También se añadirá una nueva llamada en la RestAPI que reciba como parámetro el nombre de la VesselVisit, para que actúe como evento de solicitud de publicación de datos a través del microservicio. Estas estimaciones podrán ser consultadas por todos los consumidores que se suscriban a ese canal una vez se hayan enviado las estimaciones.

- PRODUCTOR (*VesselEstimationProducer*): A través de un segundo canal dedicado, utilizará la media ponderada para consultar las estimaciones existentes en la base de datos de la visita de un barco y enviar la predicción mejorada con el objeto *VesselVisitEstimation*. Al igual que el otro productor, una nueva llamada en la RestAPI que reciba como parámetro el nombre de la VesselVisit iniciará la ejecución.

- PRODUCTOR (*TosVesselVisitsProducer*): A través de un tercer canal dedicado, propagará todas las visitas existentes desde la terminal de contenedores, de forma idéntica a cómo funciona el mismo método en la RestAPI. El único propósito de este servicio es utilizarlo para comparar el rendimiento del

microservicio frente a la RestAPI en el siguiente capítulo, ya que garantiza que se van a efectuar el mismo número de operaciones y se van a enviar paquetes de tamaño similares con ambos mecanismos.

- CONSUMIDOR (*VesselEstimationConsumer*): Recibe unos datos de entrada y, a partir de la identificación de un Tag, permitirá interpretar ese objeto como una estimación del TOS, de la naviera o de la autoridad portuaria. A partir de ahí se creará el objeto correspondiente y se persistirá en la base de datos.

Esta es la implementación para la clase VesselEstimationsProduce, que define el canal, obtiene la información y la envía a través del micro servicio. En este caso se realizará un envío individual para cada estimación de cada origen de datos disponible para ese barco:

```
@Service
@RequiredArgsConstructor
@Slf4j
public class VesselEstimationsProducer {

    private static final String TOPIC = "VesselEstimationsTopic";
    private final VesselVisitEstimationService vesselVisitEstimationService;

    @Autowired
    private KafkaTemplate<String, VesselVisitEstimation> kafkaTemplate;

    public void sendVesselEstimations(String vesselName) {
        List<VesselVisitEstimation> vesselVisitEstimationList =
               vesselVisitEstimationService.getVesselVisitEstimationsByVesselName(vessel
               Name);
        for(VesselVisitEstimation vesselVisitEstimation : vesselVisitEstimationList) {
            log.error("Producing Kafka message -> VesselVisitEstimation from source " +
               vesselVisitEstimation.getSource() + " to vessel " + vesselName);
            this.kafkaTemplate.send(TOPIC, vesselVisitEstimation);
        }
    }
}
```

Esta es la clase *VesselEstimationProducer*, como segundo productor que realizará un único envío tras calcular el tiempo de llegada estimado a partir de todas las fuentes de datos disponibles:

```
@Service
@RequiredArgsConstructor
@Slf4j
public class VesselEstimationProducer {
```

```
    private static final String TOPIC = "VesselEstimationTopic";
    private final VesselPredictorWorkflow vesselPredictorWorkflow;

    @Autowired
    private KafkaTemplate<String, VesselVisitEstimation> kafkaTemplate;

    public void sendVesselEstimation(String vesselName) {
        VesselVisitEstimation vesselVisitEstimation =
               vesselPredictorWorkflow.predictVesselArrivalTime(vesselName);
        log.error("Producing Kafka message -> VesselVisitEstimation for vessel " +
               vesselName);
        this.kafkaTemplate.send(TOPIC, vesselVisitEstimation);
    }
}
```

Y esta es la clase consumidor, *VesselEstimationConsumer*, que recibe una nueva estimación desde un origen de datos y lo almacena en la base de datos tras realizar la transformación correspondiente:

```
@Service
@Slf4j
@RequiredArgsConstructor
public class VesselEstimationConsumer {

    private final VesselVisitEstimationService vesselVisitEstimationService;
    private final SourceService sourceService;

    @KafkaListener(topics = "users", groupId = "group_id")
    public void consumeVesselEstimation(String message) throws IOException {
        log.info("Consuming Kafka message -> VesselEstimationConsumer: " + message);
        Object rObject = message;
        VesselVisitEstimation vesselVisitEstimation = new VesselVisitEstimation();
        if(message.contains("CarrierVesselVisit")) {
            log.info("Kafka estimation type -> CarrierVesselVisit");
            CarrierVesselVisit carrierVesselVisit = (CarrierVesselVisit)rObject;
            vesselVisitEstimation.setVesselName(carrierVesselVisit.getVesselName());
            vesselVisitEstimation.setVesselVisitName(carrierVesselVisit.getVesselName()
               + carrierVesselVisit.getVesselId().toString());
            vesselVisitEstimation.setEta(carrierVesselVisit.getEta());
            vesselVisitEstimation.setEtc(carrierVesselVisit.getEtc());
            vesselVisitEstimation.setSource(this.sourceService.getSourceByName(
               "Carrier"));
        }
        else if(message.contains("PortVesselVisit")) {
            log.info("Kafka estimation type -> PortVesselVisit");
            PortVesselVisit portVesselVisit = (PortVesselVisit)rObject;
            vesselVisitEstimation.setVesselName(portVesselVisit.getShipname());
           vesselVisitEstimation.setVesselVisitName(portVesselVisit.getMmsi()
               .toString());
            vesselVisitEstimation.setEta(portVesselVisit.getEta());
            vesselVisitEstimation.setSource(this.sourceService.getSourceByName(
               "Port authority"));
        }
        else if(message.contains("TosVesselVisit")) {
            log.info("Kafka estimation type -> TosVesselVisit");
            TosVesselVisit tosVesselVisit = (TosVesselVisit)rObject;
            vesselVisitEstimation.setVesselName(tosVesselVisit.getVesselName());
            vesselVisitEstimation.setVesselVisitName(tosVesselVisit.
               getVesselVisitName());
            vesselVisitEstimation.setEta(tosVesselVisit.getEta());
            vesselVisitEstimation.setEtc(tosVesselVisit.getEtc());
            vesselVisitEstimation.setSource(this.sourceService.getSourceByName(
               "Container terminal"));
        } else {
            log.info("Unknown kafka object received");
            return;
```

```
        }
        vesselVisitEstimationService.addVesselVisitEstimation(vesselVisitEstimation);
    }
}
```

Las tres clases utilizan canales diferentes, y envían o reciben los datos con el mismo formato en el que ha sido definido el objeto dentro del programa, por lo que en un escenario real debe de existir una estandarización y una definición en común por parte de las empresas externas que se conectan al mismo canal, ya sea para recibir o enviar información. La estrategia sería similar a la que se emplea a la hora de definir la estructura de un objeto DTO o *Data Transfer Object*, donde por ejemplo un software de tipo backend y otro de tipo frontend deben enviar objetos de idéntica estructura entre ellos para poder comunicarse.

El consumidor es un hilo que está escuchando continuamente el canal y esperando que algún productor externo envíe datos para guardar una nueva estimación, por lo que no necesita ninguna acción de inicio y se encuentra siempre en ejecución. Sin embargo para los dos productores se incluyen estas dos nuevas llamadas Post en la RestAPI, que reciben como parámetro el nombre del barco e inician la ejecución una vez han sido invocadas, para enviar las estimaciones individuales de la visita de un barco o la estimación calculada a partir de todas las otras:

```
@PostMapping(value = "/PublishVesselVisitEstimation")
public void calculateAndSendEstimationToKafkaTopic(@RequestParam("vesselName")
          String vesselName) {
    this.vesselEstimationProducer.sendVesselEstimation(vesselName);
}


@PostMapping(value = "/PublishVesselVisitEstimations")
public void sendEstimationsToKafkaTopic(@RequestParam("vesselName") String
          vesselName) {
    this.vesselEstimationsProducer.sendVesselEstimations(vesselName);
}
```

Por último antes de ejecutar el programa y habilitar el micro servicio es necesario incluir en el archivo de configuración del programa los parámetros necesarios. Básicamente se debe indicar el puerto que utilizarán todos los consumidores de la

aplicación y el puerto para los productores, así como el par clave y valor que se utilizara para de-serializar o serializar la información que viaje a través de los canales.

```
kafka:
       consumer:
          bootstrap-servers: localhost:9092
          group-id: group_id
          auto-offset-reset: earliest
          key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
          value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
       producer:
          bootstrap-servers: localhost:9092
          key-serializer: org.apache.kafka.common.serialization.StringSerializer
          value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

7.11 Implementación del flujo de ejecución

Con toda la implementación completada, y partiendo del escenario de que la terminal de contenedores tiene en su base de datos los registros correspondientes a próximas visitas de barcos, la naviera también dispone de estos documentos xml en el sistema de archivos, y existe conectividad a la RestApi de la autoridad portuaria, llevar a cabo una ejecución para estimar el tiempo de llegada de un barco supondría realizar las siguientes llamadas:

1. Llamada a la RestAPI - *http://localhost:5000/api/TosVesselVisits*

   Se conecta a la base de datos del TOS, recoge las próximas visitas de barcos esperadas y las almacena en la base de datos como *VesselVisitEstimation*, con el correspondiente tiempo de llegada estimado por la terminal.

2. Llamada a la RestAPI - *http://localhost:5000/api/CarrierVesselVisits*

   Se conecta al sistema de archivos de la empresa naviera, lee todos los documentos disponibles para las próximas visitas y las almacena en la base de datos como *VesselVisitEstimation*, con el correspondiente tiempo de llegada estimado por la empresa naviera.

3. Llamada a la RestAPI - *http://localhost:5000/api/PortVesselVisits*

   Se conecta a la RestAPI de la autoridad portuaria, en este caso de *MarineTraffic*, recoge las próximas visitas de barcos entre un rango de fechas

establecido y las almacena en la base de datos como *VesselVisitEstimation*, con el correspondiente tiempo de llegada estimado por la autoridad portuaria.

4. Llamada a la RestAPI - *http://localhost:5000/api/VesselVisitEstimation/{vesselName}*
   Indicando el nombre del barco correspondiente, se calcula mediante la media por pesos y las fiabilidades previamente almacenadas el mejor tiempo estimado de llegada y se devuelve como respuesta. También se podría invocar otro método para forzar la producción de este dato a través del microservicio.

5. Llamada a la RestAPI - *http://localhost:5000/api/SetVesselArrivalTime*
   Una vez se produce la llegada real del barco, es importante llamar a este método y registrar el resultado en el sistema y recalcular las diferentes fiabilidades de los orígenes de datos. También permite analizar la fiabilidad del software desarrollado, dando pie a modificar los umbrales de puntuación con un mismo conjunto de datos, comparar los diferentes resultados obtenidos y mejorar así la configuración y las estimaciones emitidas por el sistema.

8 ANÁLISIS DE RESULTADOS

8.1 Resultados de las simulaciones

El primer experimento realizado utiliza el primer método de estimación propuesto, y evalúa si éste obtiene estimaciones más fiables a las de las distintas fuentes utilizadas. Se va a emplear el siguiente conjunto de datos de entrada:

TERMINAL DE CONTENEDORES

- Total de visitas registradas en el sistema: 10
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
MSC Rochelle	MSCRC26958	2019-07-23 06:35:00
Gulbeniz	GULB52846	2019-07-23 08:05:00
AS Camellia	ASCAM211217	2019-07-23 11:30:00
Olivia I	OLII1801	2019-07-23 14:22:00
Jebel Ali	JEBAL080118	2019-07-23 15:05:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:50:00
Cinzia	CIN1801	2019-07-24 08:55:00
Grand	GRAND060118	2019-07-24 11:15:00
MSC Augusta	MSCAU090118	2019-07-24 18:20:00
ER Tokyo	ERT25846	2019-07-24 20:25:00

TABLA 7 – Datos en el sistema de la terminal de contenedores para el primer experimento.
Fuente: Elaboración propia.

EMPRESA NAVIERA

- Total de visitas registradas en el sistema: 10
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
MSC Rochelle	MSCRC26958	2019-07-23 06:45:00
Gulbeniz	GULB52846	2019-07-23 08:00:00

AS Camellia	ASCAM211217	2019-07-23 11:20:00
Olivia I	OLII1801	2019-07-23 14:25:00
Jebel Ali	JEBAL080118	2019-07-23 15:20:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:38:00
Cinzia	CIN1801	2019-07-24 08:40:00
Grand	GRAND060118	2019-07-24 11:05:00
MSC Augusta	MSCAU090118	2019-07-24 18:30:00
ER Tokyo	ERT25846	2019-07-24 20:10:00

TABLA 8 – Datos en el sistema de la empresa naviera para el primer experimento.
Fuente: Elaboración propia.

AUTORIDAD PORTUARIA

- Total de visitas registradas en el sistema: 10
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
MSC Rochelle	MSCRC26958	2019-07-23 06:52:00
Gulbeniz	GULB52846	2019-07-23 08:22:00
AS Camellia	ASCAM211217	2019-07-23 11:36:00
Olivia I	OLII1801	2019-07-23 14:49:00
Jebel Ali	JEBAL080118	2019-07-23 15:28:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:35:00
Cinzia	CIN1801	2019-07-24 08:35:00
Grand	GRAND060118	2019-07-24 11:05:00
MSC Augusta	MSCAU090118	2019-07-24 18:38:00
ER Tokyo	ERT25846	2019-07-24 20:05:00

TABLA 9 – Datos en el sistema de la autoridad portuaria para el primer experimento.
Fuente: Elaboración propia.

Una vez insertadas todas las estimaciones de los tres orígenes de datos, estos serán los valores registrados con los tiempos reales de llegada de cada una de las visitas:

Nombre del barco	Nombre de la visita	Tiempo real de llegada (ATA)
MSC Rochelle	MSCRC26958	2019-07-23 06:53:00
Gulbeniz	GULB52846	2019-07-23 08:22:00
AS Camellia	ASCAM211217	2019-07-23 11:36:00
Olivia I	OLII1801	2019-07-23 14:45:00
Jebel Ali	JEBAL080118	2019-07-23 15:26:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:33:00
Cinzia	CIN1801	2019-07-24 08:31:00
Grand	GRAND060118	2019-07-24 11:00:00
MSC Augusta	MSCAU090118	2019-07-24 18:40:00
ER Tokyo	ERT25846	2019-07-24 20:04:00

TABLA 10 – Tiempos de llegada reales registrados durante el primer experimento.
Fuente: Elaboración propia.

Con el método estándar de media ponderada, se emplean el mismo número de estimaciones para cada una de las tres fuentes disponibles. Antes de analizar los resultados, es necesario establecer unos valores de configuración o umbrales que determinarán de qué manera se va a modificar la fiabilidad de cada una de las fuentes en base al error producido. Estos valores se han establecido tras examinar la distribución de entre las 10 estimaciones y los 10 tiempos de llegada reales para cada una de las fuentes:

Intervalo de tiempo (minutos)	Puntuación
ATA ± 3 minutos = ETA	+5
ATA ± 6 minutos = ETA	+2
ATA ± 12 minutos = ETA	-2
ATA ± 12 minutos > ETA	-5

TABLA 11 – Parámetros de configuración para asignar la calidad de las estimaciones.
Fuente: Elaboración propia.

Al ejecutar el experimento con las tres fuentes de datos, sus visitas correspondientes y los parámetros de configuración, estas son las fiabilidades obtenidas tras las diez estimaciones procesadas:

- Autoridad portuaria - **141**
- TOS - **53**
- Naviera – **79**

La fuente de datos que emite estimaciones más precisas es la Autoridad Portuaria, seguida por la Naviera y con la terminal de contenedores ocupando el tercer lugar. Probablemente el sistema de antenas empleado por la web Marine Traffic permite almacenar la información a partir de la posición real del barco, y al mismo tiempo mantener esta más actualizada. Por otro lado la empresa naviera, como propietaria de las embarcaciones y como empresa que gestiona las mismas, tenga información más aproximada que la terminal de contenedores.

Por otro lado, para todas las estimaciones que se realicen a partir de ese momento, los valores emitidos por la Autoridad Portuaria tendrán casi el triple de peso que los emitidos por el TOS de la terminal de contenedores, ya que las predicciones de su sistema han resultado ser mucho más aproximadas.

A continuación se vuelve a ejecutar la misma traza pero considerando el software en sí como una fuente de datos más que emite estimaciones sobre las diferentes visitas y comienza con una fiabilidad de 100. En la siguiente tabla de resultados, se muestran todas las visitas procesadas, el tiempo real de llegada, el tiempo estimado por el sistema, la diferencia y la puntuación obtenida. Estas estimaciones utilizan el sistema desde el inicio sin entrenar, con las tres fuentes de datos con una fiabilidad también de 100 y obteniendo cada estimación en orden sin haber insertado anteriormente el tiempo real de llegada del barco. Es decir, el sistema intenta predecir el tiempo de llegada del barco antes de que éste haya llegado.

Visita	ATA	ETA estimado	Diferencia	Puntuación
MSCRC26958	2019-07-23 06:53:00	2019-07-23 06:44:00	9 min.	-2
GULB52846	2019-07-23 08:22:00	2019-07-23 08:09:20	13 min.	-5

ASCAM211217	2019-07-23 11:36:00	2019-07-23 11:29:04	7 min.	-2
OLII1801	2019-07-23 14:45:00	2019-07-23 02:33:34	12 min.	-5
JEBAL080118	2019-07-23 15:26:00	2019-07-23 15:18:54	12 min.	-5
ZIMYK020118	2019-07-23 20:33:00	2019-07-23 08:40:01	7 min.	-2
CIN1801	2019-07-24 08:31:00	2019-07-24 08:41:37	10 min.	-2
GRAND060118	2019-07-24 11:00:00	209-07-24 11:07:34	7 min.	-2
MSCAU090118	2019-07-24 18:40:00	2019-07-24 18:31:30	9 min.	-2
ERT25846	2019-07-24 20:04:00	2019-07-24 20:10:41	6 min.	+2

TABLA 12 – Estimaciones del sistema de predicción y puntuación en el primer experimento.
Fuente: Elaboración propia.

Se puede observar que a medida que se van registrando diferentes visitas en el sistema, se va puntuando cada origen de datos y se van aproximando mejor los tiempos de llegada estimados.

Si se ejecuta la misma traza pero, en lugar de iniciar las fuentes con la puntuación inicial, se inician con su peso obtenido tras diez ejecuciones (es decir, la Autoridad Portuaria tendría un peso inicial de 141, la empresa naviera un peso inicial de 79 y la terminal de contenedores un peso inicial de 53), estos son los resultados obtenidos:

Visita	ATA	ETA estimado	Diferencia	Puntuación
MSCRC26958	2019-07-23 06:53:00	2019-07-23 06:47:40	6 min.	+2
GULB52846	2019-07-23 08:22:00	2019-07-23 08:12:20	10 min.	-2
ASCAM211217	2019-07-23 11:36:00	2019-07-23 11:30:12	6 min.	+2
OLII1801	2019-07-23 14:45:00	2019-07-23 02:36:48	9 min.	-2
JEBAL080118	2019-07-23 15:26:00	2019-07-23 03:21:13	5 min.	+2
ZIMYK020118	2019-07-23 20:33:00	2019-07-23 08:38:46	5 min.	+2
CIN1801	2019-07-24 08:31:00	2019-07-24 08:40:19	9 min.	-2
GRAND060118	2019-07-24 11:00:00	2019-07-24 11:06:56	6 min.	+2
MSCAU090118	2019-07-24 18:40:00	2019-07-24 06:34:11	6 min.	-2
ERT25846	2019-07-24 20:04:00	2019-07-24 08:10:19	3 min.	+5

TABLA 13 – Estimaciones del sistema de predicción y puntuación en el primer experimento.
Fuente: Elaboración propia.

Pese a tratarse de las mismas visitas estimadas por las mismas fuentes, al encontrarse el sistema entrenado con 10 ejecuciones y tener una evaluación realizada sobre la precisión de cada uno de los orígenes de datos, el sistema realiza estimaciones más fiables dado que ya ha asignado previamente un peso a partir de las predicciones que ha efectuado cada software. Con un mayor número de ejecuciones, el sistema es capaz de emitir predicciones más fiables dado que conoce mejor cada una de las fuentes y sabe que peso se debe otorgar a cada una de ellas a partir de los resultados que ha ido obteniendo.

A continuación se ejecuta un segundo experimento, que emplea el segundo método de cálculo de estimaciones desarrollado. Este va a tener en cuenta el número de estimaciones que cada origen de datos ha realizado, con lo que se van a proporcionar los mismos datos de entrada para los tres orígenes de datos, pero cada uno realiza un número distinto de estimaciones:

TERMINAL DE CONTENEDORES

- Total de visitas registradas en el sistema: 4
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
Cinzia	CIN1801	2019-07-24 08:55:00
Grand	GRAND060118	2019-07-24 11:15:00
MSC Augusta	MSCAU090118	2019-07-24 18:20:00
ER Tokyo	ERT25846	2019-07-24 20:25:00

TABLA 14 – Datos en el sistema de la terminal de contenedores para el segundo experimento.
Fuente: Elaboración propia.

EMPRESA NAVIERA

- Total de visitas registradas en el sistema: 6
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
Jebel Ali	JEBAL080118	2019-07-23 15:20:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:38:00
Cinzia	CIN1801	2019-07-24 08:40:00
Grand	GRAND060118	2019-07-24 11:05:00
MSC Augusta	MSCAU090118	2019-07-24 18:30:00
ER Tokyo	ERT25846	2019-07-24 20:10:00

TABLA 15 – Datos en el sistema de la empresa naviera para el segundo experimento.
Fuente: Elaboración propia.

AUTORIDAD PORTUARIA

- Total de visitas registradas en el sistema: 12
- Fiabilidad inicial de la fuente: 100

Nombre del barco	Nombre de la visita	Tiempo de llegada estimado (ETA)
MSC Rochelle	MSCRC26958	2019-07-23 06:52:00
Gulbeniz	GULB52846	2019-07-23 08:22:00
AS Camellia	ASCAM211217	2019-07-23 11:36:00
Olivia I	OLII1801	2019-07-23 14:49:00
Jebel Ali	JEBAL080118	2019-07-23 15:28:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:35:00
Cinzia	CIN1801	2019-07-24 08:35:00
Grand	GRAND060118	2019-07-24 11:05:00
MSC Augusta	MSCAU090118	2019-07-24 18:38:00
ER Tokyo	ERT25846	2019-07-24 20:05:00
CSAV Traiguen	CSAVT58412	2019-07-24 21:38:00
MSC Vittoria	MSCV745	2019-07-24 23:05:00

TABLA 16 – Datos en el sistema de la autoridad portuaria para el segundo experimento.
Fuente: Elaboración propia.

En el segundo experimento, estos serían los tiempos de llegada registrados en el sistema:

Nombre del barco	Nombre de la visita	Tiempo real de llegada (ETA)
MSC Rochelle	MSCRC26958	2019-07-23 06:53:00
Gulbeniz	GULB52846	2019-07-23 08:22:00
AS Camellia	ASCAM211217	2019-07-23 11:36:00
Olivia I	OLII1801	2019-07-23 14:45:00
Jebel Ali	JEBAL080118	2019-07-23 15:26:00
Zim Yokohama	ZIMYK020118	2019-07-23 20:33:00
Cinzia	CIN1801	2019-07-24 08:31:00
Grand	GRAND060118	2019-07-24 11:00:00
MSC Augusta	MSCAU090118	2019-07-24 18:40:00
ER Tokyo	ERT25846	2019-07-24 20:04:00
CSAV Traiguen	CSAVT58412	2019-07-24 21:40:00
MSC Vittoria	MSCV745	2019-07-24 23:06:00

TABLA 17 – Tiempos de llegada reales registrados durante el segundo experimento.
Fuente: Elaboración propia.

Al ejecutar el experimento empleando los mismos parámetros de configuración, estas son las fiabilidades obtenidas:

- Autoridad portuaria: **151**
- TOS: **80**
- Naviera: **100**

Nuevamente, y dado que se trata del mismo conjunto de visitas pero con diferente número de ejecuciones, la fuente de datos más fiable es la Autoridad Portuaria. Las estimaciones emitidas por el software desarrollado obtendrían el siguiente resultado:

Visita	ATA	ETA estimado	Diferencia	Puntuación
MSCRC26958	2019-07-23 06:53:00	2019-07-23 06:52:00	1 min.	+5
GULB52846	2019-07-23 08:22:00	2019-07-23 08:22:00	0 min.	+5
ASCAM211217	2019-07-23 11:36:00	2019-07-23 11:36:00	0 min.	+5
OLII1801	2019-07-23 14:45:00	2019-07-23 14:49:00	4 min.	+2
JEBAL080118	2019-07-23 15:26:00	2019-07-23 15:25:57	1 min.	+5
ZIMYK020118	2019-07-23 20:33:00	2019-07-23 20:36:19	3 min.	+5

CIN1801	2019-07-24 08:31:00	2019-07-24 08:38:31	7 min.	-2
GRAND060118	2019-07-24 11:00:00	2019-07-24 11:05:44	5 min.	+2
MSCAU090118	2019-07-24 18:40:00	2019-07-24 18:32:04	8 min.	-2
ERT25846	2019-07-24 20:04:00	2019-07-24 20:09:21	5 min.	+2
CSAVT58412	2019-07-24 21:40:00	2019-07-24 21:38:00	2 min.	+5
MSCV745	2019-07-24 23:06:00	2019-07-24 23:05:00	1 min.	+5

TABLA 18 – Estimaciones del sistema de predicción y puntuación en el segundo experimento.
Fuente: Elaboración propia.

Esto significa que si para evaluar cada una de las fuentes también se tiene en cuenta el número de estimaciones que ha realizado, o por definirlo de otra manera, se tiene en cuenta la experiencia de cada una de estas para considerar un peso, las predicciones mejoran considerablemente con respecto al anterior experimento. En la siguiente tabla se resumen los resultados de los tres experimentos realizados, junto con una descripción resumida:

Número	Descripción del experimento	Fuente más fiable	Puntuación total
1	-Se emplea el primer método de estimación. -No tiene en cuenta las ejecuciones o la "experiencia" de cada fuente de datos. -El sistema no se encuentra entrenado, las fuentes se inicializan con una fiabilidad inicial por defecto.	Autoridad portuaria	-25
2	-Se emplea el primer método de estimación. -No tiene en cuenta las ejecuciones o la "experiencia" de cada fuente de datos. -El sistema se encuentra entrenado, el experimento comienza con las fiabilidades obtenidas como resultado del primer experimento.	Autoridad portuaria	+7
3	-Se emplea el segundo método de estimación. -Se tiene en cuenta las ejecuciones o la "experiencia" de cada fuente de datos. -El sistema no se encuentra entrenado, las fuentes se inicializan con una fiabilidad inicial por defecto.	Autoridad portuaria	+37

TABLA 19 – Resultados y resumen de los tres experimentos realizados.
Fuente: Elaboración propia.

Las conclusiones sobre los resultados obtenidos se presentarán en el capítulo 9.

8.2 Mejoras sobre el sistema propuesto

Si se observa un escenario real, además de un ajuste apropiado de los parámetros de configuración, los intervalos de tiempo y la penalización o bonificación para cada estimación realizada, se podría optimizar el sistema de manera que agrupase los barcos en función de los diferentes tipos de servicio. En un puerto no resulta apropiado utilizar los mismos márgenes de tiempo para todos los barcos que llegan a una terminal de contenedores. Para un determinado tipo de servicio es más improbable que haya retrasos o adelantos con respecto al tiempo estimado de llegada. Por ejemplo es el caso de los servicios transoceánicos, que suelen transportar una gran cantidad de contenedores. En estos resulta muy costoso hacer frente a un retraso, debido a diferentes factores:

- El barco está continuamente transportando de manera circular mercancías entre los diferentes puertos que componen la ruta. Un retraso en cualquiera de los puertos se propagará y afectará a todos los demás tiempos de llegada estimados, generando un efecto en cadena.
- Generalmente se usan grandes embarcaciones, cuyo consumo energético es muy elevado. Suelen emplear una velocidad constante y un modo de navegación económico, y cualquier imprevisto que suponga aumentar la velocidad se traduce en unas cifras mucho mayores que pueden acabar generando pérdidas en el servicio.
- Desde el punto de vista de la terminal de contenedores, el personal y equipamiento que se dedican a atender uno de estos servicios es mucho más elevado y supone una dimensión diferente a la de una pequeña embarcación, con los retos a nivel de organización y de operativa que ello implica.

Sin embargo, en un servicio con barcos de tamaño más reducido, que realicen rutas más locales o regionales, un retraso mayor puede tener menores consecuencias a todos los niveles y para todas las entidades implicadas. Por ello tendría sentido tratar estas diferencias a la hora de puntuar las diferentes estimaciones, dado que el impacto

de un retraso de por ejemplo 40 minutos en un tipo de embarcación y servicio es completamente diferente a un retraso similar en otro contexto.

Este planteamiento también daría pie a mejorar la herramienta permitiendo clasificar cada estimación en función del servicio al que pertenece. De esta manera se podrían obtener diferentes rankings y estadísticas para los diferentes servicios y navieras. Es probable que la relación comercial de una Terminal de Contenedores con una naviera en concreto se defina mediante unos compromisos diferentes a los de otra naviera, y estas estadísticas pueden resultar de gran valor para tomar decisiones estratégicas o formar en base al funcionamiento nuevas alianzas operativas.

8.3 Comparativa entre la RestAPI y el microservicio

En este apartado se evalúan las diferencias encontradas entre la implementación a través de la RestAPI y a través del microservicio, en diferentes métodos del software desarrollado.

En primer lugar se va a evaluar el tiempo de respuesta y el rendimiento de cada uno de ellos para envíos de paquetes de datos similares. Al tratarse de un desarrollo desplegado de manera local en un servidor Apache Tomcat, realizar mediciones en cuanto a rendimiento de ambos mecanismos no resulta representativo o concluyente: En un escenario real los tiempos de respuesta obtenidos al consumir un microservicio o una RestAPI van a depender de muchos factores, como pueda ser el tipo de despliegue, la plataforma utilizada, la potencia de la infraestructura, el estado de la red, la conexión empleada por el cliente o consumidor, la ubicación geográfica…

En el mercado existen numerosas herramientas para monitorizar y evaluar el estado de una RestAPI y también de un servidor que ofrece varios microservicios. En el caso de una RestAPI, las herramientas más extendidas son Uptrends, Amazon CloudWatch (sólo para despliegues en AWS), Rigor, Assertible, BlazeMeter, AppDynamics, New Relic o API Fortress, entre otros. Para monitorizar aplicaciones basadas en microservicios programados con Kafka, las herramientas más comunes son Landoop,

Confluent, Yahoo Kafka Manager, KafDrop, Linkedin Burrow o Kafka´s Tool. Estas herramientas aportan a través de diferentes paneles de control información sobre el estado de cada uno de los métodos o servicios. También permiten evaluar tiempos de respuesta, cadencias y latencias, y configurar diferentes alarmas en caso de que alguna funcionalidad deje de estar accesible desde el exterior.

Todas ellas son herramientas que funcionan bajo licencia, y una gran parte de ellas no permiten realizar un análisis con un despliegue local. Por ello, se ha optado por desarrollar un sencillo programa de escritorio en C# que realiza un determinado número de llamadas en hilos diferentes al microservicio y a la RestAPI, y genera los tiempos promedios que se tarda en recibir la información.

El programa desarrollado permite introducir la ubicación del microservicio y el nombre del canal, así como el *endpoint* para llamar a un método de la RestAPI. Por defecto realiza 10 llamadas a cada uno de estos servicios, pero este valor también es configurable. Para cada arquitectura distinta, se emplea el objeto *StopWatch* de .net disponible en la librería de diagnósticos, que devuelve el tiempo en milisegundos tras ejecutar diferentes partes del código. A través de un cliente Http y de un consumidor de Kafka para .net, se realizan tantas llamadas como se haya configurado, y al finalizar la ejecución se muestran por pantalla los resultados para la RestAPI hasta que se obtiene la información, y los resultados para el microservicio.

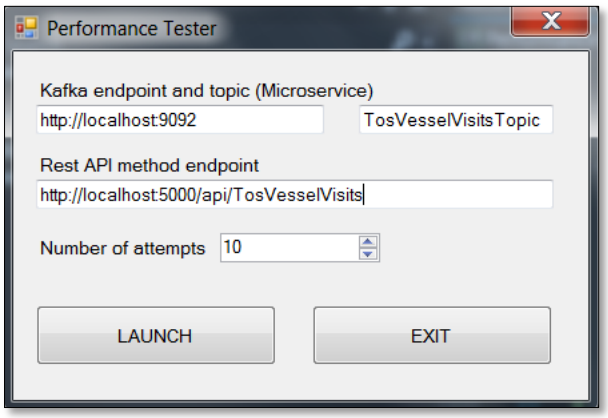

FIGURA 30 – Software para evaluar los tiempos de respuesta de SOA frente a MSA.
Fuente: Elaboración propia.

Los resultados obtenidos, teniendo en cuenta que tanto a través de la RestAPI como del microservicio se ha invocado el mismo método y se han devuelto resultados y paquetes con el mismo tamaño, son más favorables para el microservicio, independientemente del número de ejecuciones que se lance. El tiempo de ejecución mostrado para el microservicio parte de que los mensajes ya han sido producidos y todos se encuentran en la cola listos para ser consumidos. Para la RestAPI, no es necesario producir estos mensajes de antemano o realizar llamadas adicionales:

Número de llamadas	Tiempo con RestAPI	Tiempo con microservicio Kafka
10	22ms	25ms
50	45ms	50ms
200	190ms	150ms

TABLA 21 – Tiempos de respuesta obtenidos para la RestAPI y para el microservicio Kafka.
Fuente: Elaboración propia.

Este es el método principal del programa, que para las X iteraciones introducidas llama a la RestAPI y calcula el tiempo de respuesta, solicita la producción de X mensajes en el microservicio de Kafka, y consume ese número de veces los mensajes desde el mismo canal, calculando el tiempo de respuesta para este consumo y mostrando los resultados por pantalla:

```
private void btnLaunch_Click(object sender, EventArgs e){
        try {
                String timeRest = "";
                String timeKafka = "";
                Stopwatch sw = new Stopwatch();
                sw.Start();
                // Measure performance for Rest API
                for (int i = 0; i < numericAttempts.Value; i++){
                        var url = this.inputRest.Text;
                        var webrequest =(HttpWebRequest)System.Net.WebRequest.Create(url);
                        using (var response = webrequest.GetResponse())
                        using (var reader = new
                                StreamReader(response.GetResponseStream())) {
                                var result = reader.ReadToEnd();
                        }
                }
                sw.Stop();
                timeRest = sw.Elapsed.TotalMilliseconds.ToString();
                // Ask VesselVisitPredictor to produce 10 messages
                for (int i = 0; i < numericAttempts.Value; i++){
                        using (var clientPost = new WebClient()) {
                                var responsepost =
                                        clientPost.UploadValues("http://localhost:5000/kafk
```

```
                                a/publish", new NameValueCollection());
                        var responseString =
                                Encoding.Default.GetString(responsepost);
                }
        }
        // Measure performance for Microservice
        sw = new Stopwatch();
        sw.Start();
        string topic = this.inputTopic.Text;
        Uri uri = new Uri(this.inputKafka.Text);
        var options = new KafkaOptions(uri);
        var router = new BrokerRouter(options);
        var consumer = new Consumer(new ConsumerOptions(topic, router));
        foreach (var message in consumer.Consume()){
                Console.WriteLine(Encoding.UTF8.GetString(message.Value));
        }
        Console.ReadLine();
        sw.Stop();
        timeKafka = sw.Elapsed.TotalMilliseconds.ToString();
        MessageBox.Show("PERFORMANCE RESULTS FOR THE VESSEL VISIT PREDICTOR\n\n"
                + "CONSUMING REST API: " + timeRest + " ms for " +
                this.numericAttempts.Value.ToString() + " attempts.\n" +
                "CONSUMING KAFKA MICROSERVICE: " + timeKafka + " ms for " +
                this.numericAttempts.Value.ToString() + " attempts.");
    }
    catch (Exception ex) {
        MessageBox.Show(ex.ToString());
    }
}
```

Otros programas que resultan de gran ayuda son los clientes que permiten llamar de forma externa al servicio. En este caso se encuentran más opciones para trabajar con la RestAPI, seguramente por el tiempo que la tecnología lleva en el mercado con respecto a Kafka o RabbitMQ. Resultan de gran utilidad para testear la RestAPI y los diferentes métodos, así como enviar diferentes datos y comprobar los resultados obtenidos. El cliente *Advanced Rest Client* de Google, disponible tanto para escritorio como aplicación para Chrome, permite efectuar peticiones a una RestAPI estándar:

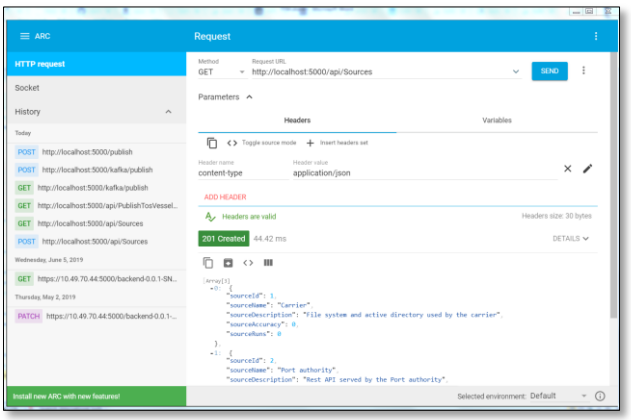

FIGURA 31 – Cliente Advanced Rest Client de Google empleado con a RestAPI desarrollada.
Fuente: Elaboración propia.

Otro software utilizado que dispone de una versión gratuita es *Postman*. Ofrece más funcionalidades, permite salvar diferentes llamadas y además la versión de pago permite la instalación en varios clientes, con lo que un equipo de desarrolladores puede compartir las llamadas que han sido creadas y definidas por otros miembros. También es posible realizar mediciones sobre el tiempo de respuesta de cada uno de los métodos, e incorpora un módulo adicional, *Postman Monitor*, que permite monitorizar la RestAPI y programar alarmas y alertas si alguno de los servicios no responde.

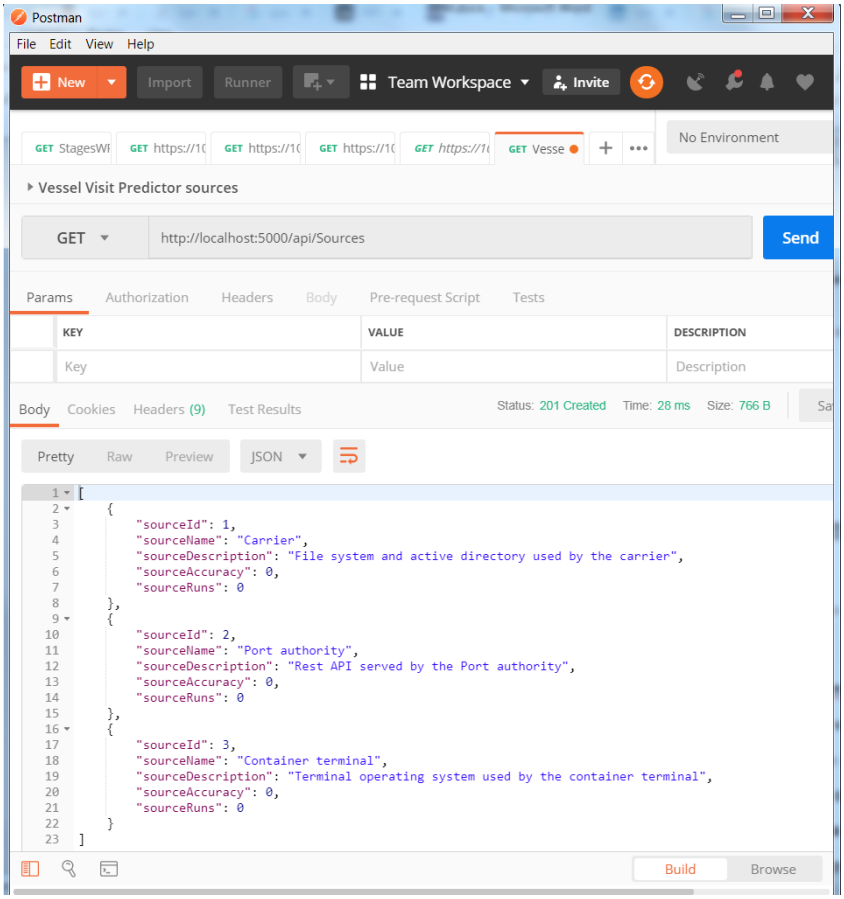

FIGURA 32 – Cliente Postman empleado con a RestAPI desarrollada.
Fuente: Elaboración propia.

En cuanto a limitaciones, sin duda existe mayor libertad en un microservicio sobre cómo o con que formato enviar los datos, y no existe un estándar tan establecido como pueda ser una RestAPI. Esto puede resultar beneficioso en ciertos escenarios o

no, dependiendo de qué manera se trabaje. Una RestAPI limita más la manera de trabajar, hay que adaptarse a su definición estricta, sus métodos como Get, Post, Put o Patch, la forma de enviar parámetros y el propósito de cada uno de ellos. A demás, generalmente resulta necesario limitarse a los objetos de JavaScript (JSON) o archivos XML. Sin embargo a la hora de mandar datos a través de un microservicio el programador dispone de mucha más libertad: Se trata únicamente de un canal, en el que se pueden enviar bits, bytes, objetos, cadenas de texto, ficheros JSON o XML… Esto permite realizar una definición más personalizada para cada canal en un microservicio, teniendo en cuenta también el esfuerzo adicional que ello conlleva. Desde este punto de vista los microservicios son muy similares a los clásicos servidores de comunicaciones o tecnologías como Apache ActiveMQ y JMS (Java Messaging System).

En cuanto a la producción de datos, el rendimiento en una aplicación basada en microservicios en la que existen muchos consumidores puede ser mejor que en una RestAPI, ya que a un canal se pueden subscribir muchos consumidores para los que sólo es necesario producir una vez los datos. En una RestAPI, si para cada petición es necesario realizar accesos a la base de datos y ejecutar determinadas operaciones para devolver el resultado, esto será algo que se repita para cada cliente que realice la misma petición. Es posible optimizar esto de muchas maneras, pero la gran ventaja en este caso de Apache Kafka es que al haber producido cierta información en un canal, esta puede ser consumida por muchos consumidores, aportando mayores capacidades para distribución de datos a gran escala.

Una diferencia importante a tener en cuenta a la hora de la implementación es que para emplear un microservicio y que el consumidor o cliente pueda obtener una serie de datos, es necesario que estos se hayan producido. Es decir, es necesario forzar de alguna manera la producción de estos datos. En una RestAPI simplemente se realiza la llamada y se genera la respuesta a la misma, en caso de que la haya. En un microservicio, para poder obtener información o datos es necesario que estos se hayan producido en el canal o *topic* correspondiente. Esto cambia la naturaleza de la

comunicación, ya que desde ese punto de vista un microservicio puede resultar ideal cuando se desea tener un canal abierto que esté escuchando a determinados cambios y se lance cualquier lógica o código en el consumidor una vez estos se producen. Sin embargo una RestAPI funciona bajo demanda o petición del consumidor. No obstante, una aplicación basada en microservicios también es capaz de generar los datos bajo demanda ya que puede escuchar peticiones del consumidor y a partir de ahí producir los datos, pero hay que tener en cuenta que esta implementación debe hacerse de forma manual, como se ha hecho en el programa desarrollado.

En el despliegue para las simulaciones se ha empleado un único servidor Apache Tomcat, y no se han tenido en cuenta problemas relacionados con la redundancia y la posible caída de servidores. Pero tras la lectura de varios tutoriales tanto para arquitecturas SOA como MSA, sí es cierto que Kafka ha sido concebido para responder de una manera más nativa a los sistemas que requieren alta disponibilidad y redundancia. Para ambas arquitecturas es posible a través de la infraestructura IT que se emplee redundar todos los servicios en varios servidores, pero por defecto los microservicios y en concreto, Apache Kafka, ofrece una serie de ventajas y funcionalidades que pueden ser empleadas desde el código. Permite diseñar una política de particiones y adaptarse al número de consumidores que haya a través del uso de tablas Hash. También es posible realizar una gestión en base a cada tipo de cliente o consumidor, pudiendo distinguir entre clientes con tráfico elevado y clientes con tráfico bajo. Todo es gestionable desde el código y permite generar canales y particiones de forma dinámica, con lo que abre un espacio para la gestión del tráfico desde el código del software. En una RestAPI convencional y sin el uso de módulos o programas adicionales, no es posible programar esta gestión.

9 CONCLUSIONES

9.1 Conclusiones obtenidas

El sector de desarrollo de software para el transporte marítimo de contenedores es un sector exclusivo y con un reducido número de empresas, siendo el TOS una de las piezas clave. Se ha observado un aumento significativo durante los últimos años en cuanto al número de *startups* y nuevas iniciativas, en armonía con las últimas tendencias tecnológicas como *IoT*, pero está claramente dominado por empresas software con unos 20 – 30 años de antigüedad en promedio. Al mismo tiempo, son cada vez más las terminales que desarrollan sus propios sistemas para suplir carencias de los software adquiridos y/o personalizar la manera de trabajar.

Se ha comprobado que en este ecosistema de software se encuentran los mismos datos replicados en diferentes fuentes y con valores distintos, generando un problema de fiabilidad y confianza de entre todas las empresas vinculadas.

La solución propuesta utiliza las principales ventajas de la arquitectura orientada a servicios y reaprovecha los recursos ya existentes, con el fin de mejorar la calidad de los datos y de las estimaciones realizadas. Con esto se genera una única fuente de verdad accesible por todos los actores implicados, que es capaz de unificar y mejorar los resultados obtenidos.

A través de tres experimentos se han analizado las estimaciones del tiempo de llegada de un barco de tres fuentes de datos distintas, obteniendo las siguientes conclusiones:

- Un sistema que evalúa la calidad de las fuentes de datos de las que recibe información es capaz de interpretar los datos de forma más precisa.
- Si el sistema ha sido entrenado previamente y conoce de antemano la calidad de los datos que recibe de cada fuente, es capaz de ponderar y obtener valores promedios de forma más eficaz. Para el caso de las estimaciones, estas mejoran considerablemente y se aproximan más a los valores reales.

- Si además como elemento de ponderación se utiliza también la experiencia de cada origen de datos, los datos obtenidos mejoran con respecto a los experimentos anteriores.

Es posible combinar diferentes fuentes de datos y servir la información de manera unificada e integrada, mejorando así la calidad de los datos a través de un sistema de puntuación para cada origen individual. Generalmente, y dependiendo del nivel al que esté particionado un sistema web, será preferible adoptar una arquitectura orientada a microservicios (MSA), o una arquitectura orientada a servicios (SOA) en el caso de grandes y complejos entramados empresariales.

9.2 Líneas de investigación futuras

Este trabajo final de máster ha sentado las bases para reutilizar y unificar diferentes fuentes de datos y mejorar la calidad de los mismos. Es posible realizar a partir de estas bases futuras líneas de investigación, por lo que a continuación se mencionan algunas posibilidades.

En los experimentos realizados se ha utilizado el tiempo estimado de llegada del barco, pero realmente el sistema propuesto sería capaz de ponderar y mejorar cualquier dato que pueda ser expresado de forma matemática. Si se asume que hay varias fuentes de datos presentes almacenando el mismo dato pero con diferente valor, y si es posible conocer el valor verdadero o real de ese dato, entonces es posible estimar la desviación de cada una de las fuentes. Por lo tanto este mecanismo se podría extrapolar a muchos otros datos como puede ser el tiempo estimado de finalización de los trabajos de un barco, los tiempos estimados de llegada para toda la maquinaria de transporte horizontal en una terminal de contenedores, o cualquier otro dato numérico manejado por diferentes sistemas.

Por otro lado también se permite conectar varias fuentes de datos, con lo que se podrían obtener conclusiones con respecto al comportamiento del sistema y la precisión en base al número de fuentes empleadas. Se trataría de comprobar si existe

una relación directa o indirecta entre la fiabilidad de las estimaciones realizadas con respecto al número de fuentes de datos conectadas.

Si por otro lado fuese posible desplegar el software en diferentes puertos y terminales, se podrían buscar similitudes y errores comunes entre las diferentes instalaciones. También permitiría obtener conclusiones en base a diferentes zonas geográficas y el comportamiento de una misma tecnología o sistema software en diferentes localizaciones.

Por último, mencionar que durante las pruebas realizadas ha resultado de vital importancia establecer unos intervalos de puntuación adecuados para los datos obtenidos en un periodo concreto en el puerto de Valencia durante dos días consecutivos. Esto significa que para el sistema emita predicciones acertadas y evalúe correctamente las distintas fuentes de datos, es necesario ajustar estos valores, y estos pueden variar dependiendo de los errores promedios o diferencias que se encuentren. No es lo mismo encontrar por ejemplo en varias fuentes de datos similares valores con 15 minutos de diferencia máxima entre ellos que diferencias de 150 minutos. Este proceso podría automatizarse, siendo el propio software el que a raíz de los datos de entrada y el tamaño de las diferencias que vaya encontrando, establezca sus propios intervalos y sistema de puntuación. Una vez implementado esto, se podrían comparar los resultados con los obtenidos a través de un ajuste manual, para comprobar si efectivamente el sistema mejora las predicciones. Como reglas de entrada para poder programar esto, debería realizarse en varios escenarios distintos un ajuste manual de los intervalos y valores de puntuación. De esta manera, se permitiría obtener las conclusiones sobre cómo el sistema debería calibrarse a sí mismo para alcanzar los mejores resultados en cualquier escenario. Se trataría de que el software primero analizase los orígenes de datos y luego se calibrase de manera autónoma, antes de ser ejecutado por primera vez.

10 **BIBLIOGRAFÍA Y REFERENCIAS**

[1] Eberhard Wolff. Microservices, flexible software architecture. Pearson Education. 2016.

[2] Orbcomm various authors. Remote Container Management White Paper. Orbcomm 2016. [Online]. Disponible: http://www2.orbcomm.com/remote-container-management

[3] Shawn Hermans. What is the future of Service Oriented Architecture? Quora 2015. [Online]. Disponible: https://www.quora.com/What-is-the-future-of-Service-Oriented-Architecture

[4] Michael Amaral. What is the future of Service Oriented Architecture in the Web? Quora 2017. [Online]. Disponible: https://www.quora.com/What-is-the-future-of-Service-Oriented-Architecture-on-The-Web

[5] Stephen Watts. Microservices VS SOA, how are they different? BMC Blogs 2017. [Online]. Disponible: https://www.bmc.com/blogs/microservices-vs-soa-whats-difference/

[6] Kiran Gangadharan. What are the advantages of Service Oriented Architecture (SOA)? Quora 2017. [Online]. Disponible: https://www.quora.com/What-are-the-advantages-of-Service-Oriented-Architecture-SOA

[7] Cloves Carneiro, Tim Schmelmer. Microservices from Day One. Apress 2016.

[8] Varios autores. Libro de estrategias para la administración de API. CA Technologies 2015. [Online]. Disponible: https://www.ca.com/content/dam/ca/ar/files/ebook/the-api-management-playbook.pdf

[9] Mike Barlow. Why you should care about SOA again?. Hewlett Packard Enterprise 2017. [Online]. Disponible: https://www.hpe.com/us/en/insights/articles/why-you-should-care-about-soa-again-1706.html

[10] Martin Higgings. Microservices and the future of service oriented architecture. Novarica 2018. [Online]. Disponible: https://novarica.com/2018/10/microservices-and-the-future-of-service-oriented-architecture/

[11] Santanu Das. What is a service-oriented architecture (SOA)? How is it implemented? Quora 2013. [Online]. Diponible: https://www.quora.com/What-is-a-service-oriented-architecture-SOA-How-is-it-implemented

[12] Txema Rodriguez. ¿Por qué deberíamos abandonar REST y empezar a usar GraphQL en nuestras APIs?. Genbeta 2016. [Online]. Disponible: https://www.genbeta.com/desarrollo/por-que-deberiamos-abandonar-rest-y-empezar-a-usar-graphql-en-nuestras-apis

[13] Ulysses Marins. An introduction to Apache Kafka and microservices communication. Medium 2016. [Online]. Disponible: https://medium.com/@ulymarins/an-introduction-to-apache-kafka-and-microservices-communication-bf0a0966d63

[14] Usama Ashraf. Using RabbitMQ for microservices communication on Docker. Codeburst 2018. [Online]. Disponible: https://codeburst.io/using-rabbitmq-for-microservices-communication-on-docker-a43840401819

[15] Sourabh Sharma. Mastering microservices with Java.

[16] Fernando Chanaka. Dzone 2016. Rethinking Integration Solutions: From SOA to the Future. [Online]. Disponible: https://dzone.com/articles/building-integration-solutions-a-rethink

[17] Kasun Indrasiri. Microservices, APIs and Integration. Medium 2017. [Online]. Disponible: https://medium.com/@kasunindrasiri/microservices-apis-and-integration-7661448e8a86

[18] Lovisa Johansson. Rabbit MQ for beginners, what is rabbit MQ?. Cloud AMQP 2015. [Online]. Disponible: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html

[19] Robert L. Mitchell. The 8 best open-source tools for building microservice apps. Techbeacon 2017. [Online]. Disponible: https://techbeacon.com/enterprise-it/8-best-open-source-tools-building-microservice-apps

[20] Thomas Erl. SOA, principles of service design. Pearson Education 2007.

[21] Thomas Erl, Benjamin Carlyle. SOA with REST, principles, patterns and constraints for building enterprise solutions with REST. Pearson Education 2012.

[22] Eberhard Wolff. Technologies for Microservices. Jax London 2019. [Online]. Disponible: https://jaxlondon.com/blog/technologies-for-microservices/

[23] Lesa Mone. Top Microservices Trends of 2018. Leanix blog 2018. [Online]. Disponible: https://blog.leanix.net/en/top-microservices-trends-of-2018

[24] Stefan Thorpe. 29 top tools for building microservices on all levels. Dzone 2018. [Online]. Disponible: https://dzone.com/articles/30top-tools-for-building-microservices-on-all-leve

[25] Juan María Fix. Por qué todos apuestan por Kubernetes. Paradigma digital 2017. [Online]. Disponible: https://www.paradigmadigital.com/techbiz/por-que-todos-apuestan-por-kubernetes/

Páginas web consultadas:

- Tideworks: http://www.tideworks.com
- Totalsoftbank: https://totalsoft.en.ec21.com/
- RBS: https://www.rbs-emea.com/products/tops-advance/
- XVELA: https://www.xvela.com/
- Wikipedia, Marine Traffic: https://en.wikipedia.org/wiki/MarineTraffic
- Marinetraffic: https://www.marinetraffic.com
- Navis: https://www.navis.com/
- Octopi: https://octopi.co/
- Porttechnology: https://www.porttechnology.org/directory/octopi
- Cyberlogitech: http://www.cyberlogitec.com
- Tba: https://www.tba.group/
- Marfret: http://www.marfret.fr
- Track and trace: https://www.track-trace.com/container
- Ocean Insights: https://www.ocean-insights.com/container-track-and-trace/
- Kalmar: https://www.kalmarglobal.com/automation/kalmar-oneterminal/
- Konecranes: https://www.konecranes.com/
- Apache Kafka: https://kafka.apache.org/
- GraphQL: https://graphql.org/
- Kubernetes: https://kubernetes.io/
- Wikipedia, kubernetes: https://es.wikipedia.org/wiki/Kubernetes
- Wikipedia, media ponderada: https://es.wikipedia.org/wiki/Media_ponderada
- Swagger: https://swagger.io/

11 SIGLAS, ABREVIATURAS Y ACRÓNIMOS

ActiveMQ – Broker de mensajeria de codigo abierto que implementa la especificación JMS o Java Messaging System.

AIS – Antena dedicada que opera en dos canales incluidos dentro del rango de frecuencias VHF marino, empleada para recibir y transmitir datos.

Apache Tomcat – Contenedor de servlets desarrollado por Apache Software Foundation.

API – Application Programming Interface, interfaz de programación de aplicaciones.

ATA – Actual Time of Arrival, o tiempo de llegada real de un barco a un puerto.

Backend – Parte de un desarrollo web encargada de que la lógica funcione.

BI – Business Intelligence, habilidad para transformar los datos en información y la información en conocimiento.

DTO – Data Transfer Object, patrón o clase plana que permite transmitir información entre múltiples fuentes de datos.

ECS – Equipment Control System, software que maneja y regula el comportamiento del equipamiento automatizado de la terminal.

EDI – Electronic Data Interchange, intercambio electrónico de documentos.

ESB – Enterprise Service Bus, implementa un sistema de comunicación entre aplicaciones de software que interactúan mutuamente en una arquitectura orientada a servicios.

ETA – Estimated Time of Arrival, tiempo de llegada estimado de un barco a un puerto.

ETL – Extract, transform and load, procedimiento general de copier datos desde una o más fuentes en un sistema de destino que representa los datos de forma diferente.

Gradle – Sistema de automatización de compilación de código abierto basado en Apache Ant y Apache Maven.

Groovy – Lenguaje de programación basado en java estático y dinámico.

Hibernate – Herramienta de mapeo relacional de objetos para Java.

HTTP – Protocolo de transferencia de hipertexto.

IMO – Mercancías peligrosas debido a sus propiedades o características.

Java – Lenguaje de programación orientado a objetos.

JMS – Sistema de mensajería orientado a clientes Java.

JPA – API para aplicaciones Java que describe bases de datos relacionales.

JSON – Formato estándar para definición de objetos basado en valores y atributos.

Kafka – Software de procesamiento de flujos de datos basado en Java y Scala.

KPI – Valor o indicador de rendimiento para evaluar el éxito de una actividad en particular.

Microservicio – Variante de la arquitectura basada en servicios que estructura una aplicación como una colección de servicios desacoplados.

MSA – Arquitectura basada en microservicios.

OCR – Sistema computerizado de análisis para reconocimiento de carácteres.

Restful – Porgramas basados en REST, arquitectura basada en redes.

SOA – Arquitectura orientada a servicios, nexo que une las metas de negocio con el sistema software.

Springboot – Infraestructura ligera que elimina gran parte del trabajo de configurar aplicaciones basadas en Spring.

SQL – Lenguaje de consultas estructurado utilizado en programación para obtener información de los sistemas de base de datos.

TOS – Software o pieza de la cadena de suministro que controla el movimiento y almacenamiento de diferentes tipos de carga dentro y alrededor de una terminal de contenedores.

XML – Lenguaje de marcas extensible que permite definir los marcados y sus reglas.