Proyecto *smartPORT logistics*

PID_00247340

Màrius Montón Macián

Tiempo mínimo de dedicación recomendado: 1 hora

Universitat Oberta de Catalunya

Índice

Introducción

El proyecto *smartPORT logistics* se inició para crear una plataforma que integrase y adquiriese toda la información necesaria para incrementar la eficiencia de un *hub* logístico.

El proyecto fue diseñado por la empresa T-Systems para el puerto de Hamburgo, gestionado por la empresa HPA (Hamburg Port Authority) [HPA (2014)]. El *hub* logístico tiene un tamaño de 7.200 hectáreas: es el primer puerto de Alemania, el tercer puerto de Europa por tonelaje de mercancías transportadas y el 26 del mundo. En este puerto, se mueven anualmente más de 130 millones de toneladas de mercancías [AAPA (2010)].

El puerto lo usan más de 10.000 barcos cada año, y del complejo salen cada día unos 200 trenes cargados de contenedores. Cada día, se mueven más de 40.000 camiones de carga por el puerto.

Dentro del puerto, hay más de 140 km de carreteras y más de 300 km de vías para trenes, 49 km de muelles y 130 puentes que lo conectan con la ciudad de Hamburgo. En el puerto, operan más de 1.700 compañías de transporte y 110 compañías ferroviarias.

Vistos los números y las cantidades que se mueven en un complejo logístico de tal magnitud, es fácil hacerse a la idea de los problemas que se plantean y que necesitan una solución.

Los problemas que suelen encontrarse en un *hub* logístico y que el proyecto *smartPORT logistics* intenta solventar son, entre otros, los siguientes (figura 1):

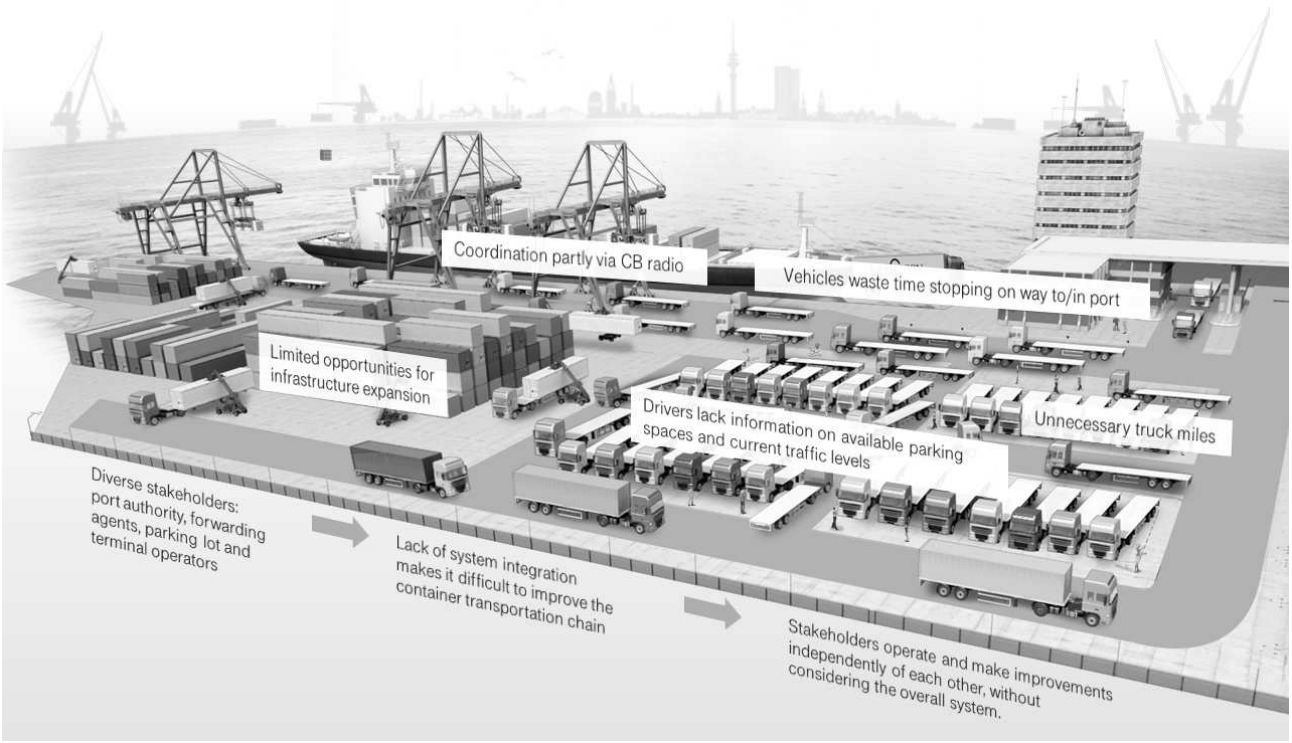
- Localización de contenedores. Los contenedores no están localizados *per se*, sino que se extienden órdenes de adónde deben llevarse. Esto puede conducir a extravíos de los mismos.

- Los camiones en espera pueden recorrer kilómetros para encontrar un espacio de aparcamiento libre.

- Cada parte implicada no comunicada con las otras. De este modo, cada compañía o parte interesada no interconecta sus datos con las demás.

- En general, buena parte de los procesos se hacen de forma manual y en soporte papel. Esto provoca errores, incidentes y poco control del proceso en algunos casos.

Figura 1. Visión general de los principales problemas identificados en el proyecto



© T-Systems

Estos problemas generan, a su vez, los siguientes inconvenientes:

- Retraso en la entrega de contenedores, que a su vez puede provocar retrasos en la partida de barcos, con el coste que esto supone.

- Congestión en el tráfico rodado, lo que provoca, a su vez, retrasos y contaminación atmosférica.

- Tiempos de espera no previstos, que provocan que camiones, conductores, trenes y barcos estén parados, esperando su turno para ser cargados o descargados.

Para dar solución a estos problemas, se propuso la obtención de todos los datos necesarios. Esta información alimentará distintos sistemas con datos en crudo y agregados, para incrementar el rendimiento y automatizar procesos en los que la infraestructura y el espacio del *hub* logístico son un factor limitante.

A partir de información en tiempo real sobre el tráfico y las infraestructuras, es posible controlar los vehículos implicados de manera más eficiente dentro del ecosistema del *hub*.

A los negocios que operan dentro del *hub*, se les proporciona información en tiempo real para que puedan mejorar sus procesos de automatización.

Así, el proyecto puesto en marcha propone ofrecer las siguientes soluciones (figura 2):

- Información de próximas llegada y tráfico marítimo previsto. Así se puede prever la futura carga de trabajo.

- Camiones reciben información en tiempo real de móviles inteligentes de la situación de su carga.

- Operadores de carga reciben el tiempo estimado de llegada de su transporte, para así planificar mejor los movimientos de contenedores.

Figura 2. Visión general de las propuestas principales del proyecto

© T-Systems

1. Control mercancías

Para la gestión y el control de todos los contenedores que se mueven en el puerto, se creó una aplicación centralizada para el control de las mercancías.

Esta aplicación está pensada para los conductores de los camiones y programadores. Con la aplicación en tabletas y teléfonos móviles, pueden recibir la siguiente información:

- Situación del tráfico dentro del puerto.
- Situación del tráfico en los accesos al puerto.
- Información del estado de los puentes (horario previsto de apertura y cierre) y del resto de las infraestructuras.
- Situación de los terminales de contenedores.
- Situación de los aparcamientos para camiones.

Con esta aplicación, se consigue que los conductores tengan información al momento y puedan responder rápidamente a cambios o problemas dentro del puerto. Además, teniendo la información centralizada de la localización de cada camión y sabiendo el estado de toda la infraestructura, es posible obtener datos nuevos como, por ejemplo, el tiempo estimado de llegada.

Con una aplicación en tiempo real de este tipo, es posible también tener en cualquier momento un estado global del puerto, lo que permite hacer cambios en la planificación o en las rutas para evitar futuras congestiones o mejorar el tráfico de camiones.

2. Control de tráfico

Para tener una visión global del tráfico rodado dentro del puerto, en primer lugar se usan sensores estáticos típicos para esta labor (cámaras de vídeo, lazos inductivos, etc.).

Con esta información, y las planificaciones de carga y descarga de mercancías, puede predecirse la situación futura del tráfico. Toda esta plataforma ofrece datos agregados tales como cantidad de horas perdidas en atascos de tráfico, emisiones de CO_2 emitidas, etc.

3. Infraestructuras

A lo largo de todas las vías de tren, dentro del puerto, se instalaron sensores de paso, de forma que es posible saber la localización de cada tren y vagón en todo momento. Con esta información es posible llevar a cabo un mantenimiento predictivo, esto es, anticiparse a los fallos o averías antes de que sucedan.

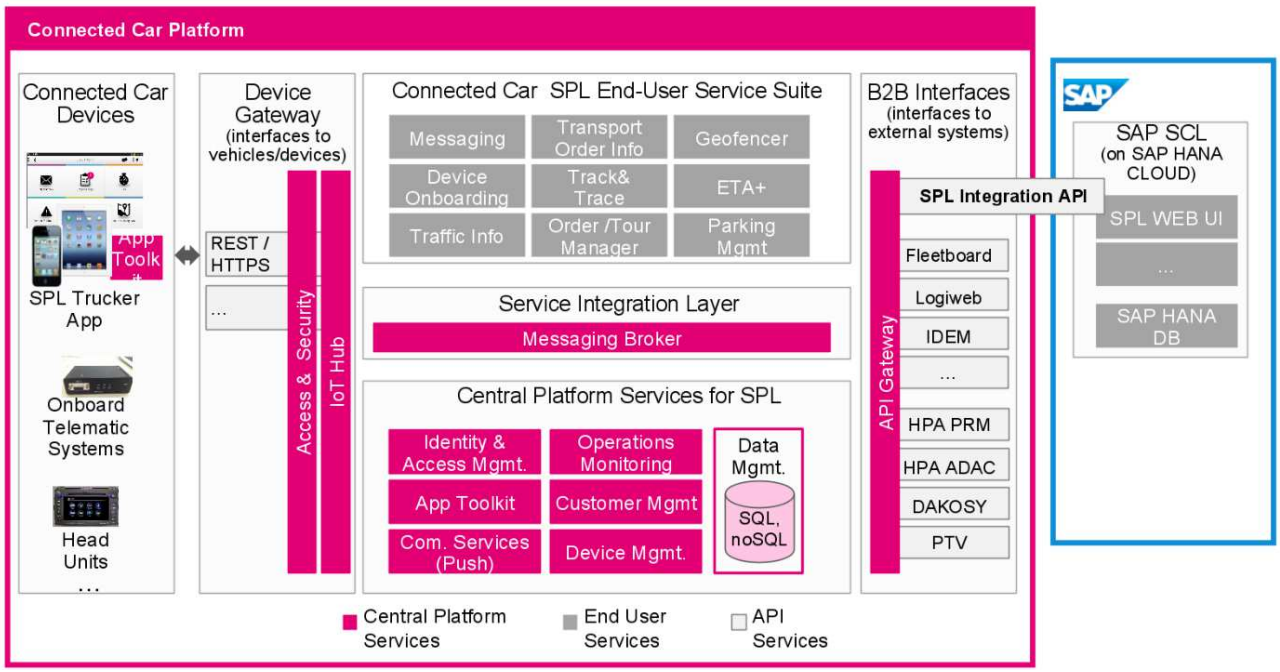
También se usa un *tracker* GPS para tener información precisa en todo momento de la localización de ciertos activos clave, como grúas de gran tonelaje o puentes móviles. De esta forma, la gestión de estos elementos clave puede ser más eficiente.

4. Infraestructura *Cloud*

Todo el soporte informático para el proyecto se diseñó para usar *Cloud* como base.

De forma centralizada y en la nube, se reciben los datos de los distintos subsistemas (control de tráfico, estado de las vías, geolocalización de maquinaria pesada, etc.), y se gestionan de manera centralizada.

Figura 3. Arquitectura genérica del proyecto.



© T-Systems

En la figura 3 se presenta la arquitectura general de todo el proyecto.

En la misma, se ve cómo la entrada de datos por parte de los dispositivos se lleva a cabo mediante una *interface* API REST estándar.

Estos datos los comparte el *service integration layer*. Este servicio permite a los demás servicios o aplicaciones recibir los datos que requieran, de manera sencilla.

Uno de estos servicios es el que se encarga de almacenar los datos en distintas bases de datos según su naturaleza y uso. Estas bases de datos son accedidas por las aplicaciones que necesiten histórico de datos, o datos que no provienen de las aplicaciones externas propiamente dichas (por ejemplo, los datos de cada compañía, datos de llegada de barcos, datos de la carga de los barcos, etc.).

A continuación, se describen algunas de las tecnologías usadas en el proyecto. Algunas ya se han presentado en apartados anteriores, y simplemente se enumeran. Otras se describen de manera más detallada en este apartado.

4.1. Spring

Todo el proyecto se basa en Java y sus tecnologías asociadas, como por ejemplo Spring, Spring Boot y Spring Cloud [Pivotal Software (2017)].

Spring es un *framework* o entorno de trabajo para Java, que ayuda a los desarrolladores y los equipos a trabajar más eficientemente, de modo que toda la infraestructura software la proporciona Spring, y los desarrolladores pueden dedicarse a la lógica de la aplicación.

Entrar en detalles técnicos sobre Spring queda fuera del ámbito de este material.

4.2. REST y JSON

La comunicación entre módulos se hace mediante una API REST clásica, intercambiando datos en formato JSON.

Una API REST es una forma de comunicar servidores usando el protocolo HTTP (el utilizado en internet para intercambiar páginas web). Es un protocolo sencillo y muy usado, y que los desarrolladores conocen ampliamente.

Una API REST proporciona cuatro primitivas básicas de tratar datos, basadas en recursos. Estos recursos (denominados URI) son equivalentes a las URL de una página web: cada URL o dirección identifica un recurso de forma única.

Algunos ejemplos de URI podrían ser los siguientes:

- http://rest.servidor.com/maquinas/1/
- http://rest.servidor.com/maquinas/1/sensores/
- http://rest.servidor.com/maquinas/1/sensores/temp
- http://rest.servidor.com/maquinas/1/sensores/presion
- http://rest.servidor.com/maquinas/1/sensores/presion

- http://rest.servidor.com/maquinas/1/actuadores/
- http://rest.servidor.com/maquinas/1/actuadores/valvulas
- http://rest.servidor.com/maquinas/1/actuadores/valvulas/1

En este caso, cada URI proporciona información sobre los sensores o actuadores de ciertas máquinas.

El protocolo HTTP proporciona las siguientes cuatro acciones posibles, en las que la API REST se basa.

- GET: para consultar y leer recursos.
- POST: para crear recursos.
- PUT: para modificar recursos.
- DELETE: para borrar recursos.

Los datos que hay que enviar se codifican usando el formato de datos JSON. Este formato utiliza texto y una estructura sencilla para almacenar datos. Es de uso muy común en comunicación entre servidores.

Así, una llamada GET a http://rest.servidor.com/maquinas/1/ podría retornar un JSON similar al mostrado en el listado 4.1.

Hay que destacar que JSON es un formato de intercambio de datos. El hecho de que una API REST use JSON no significa que los datos se almacenen en este formato, sino que se recogen los datos de la fuente de datos correspondiente y se prepara un JSON para enviar los datos.

4.3. JMS y ActiveMQ

Java Message Service, o JMS, es una API para intercambio de mensajes entre servicios. Su objetivo es parecido a una REST API por HTTP, pero usando otra tecnología más cercana a JAVA.

En casos en los que quien desea los datos y el servicio que los proporciona trabajen en Java, puede ser una buena forma de intercambiar información, dado que la interfaz es muy sencilla utilizando este lenguaje. En otros casos, se aconseja usar una API REST.

Listing 4.1: Ejemplo fichero JSON

```
{
"maquina": 1,
"localizacion": "planta 1, sección A1",
"sensores": {
    "temp": "45.2",
    "presion": "7800"
    },
 "actuadores": {
    "valvulas": {
      "S1" : "on",
      "S2" : "off"
    },
    "radiadores": {
      "R1": "on",
      "R2": "on",
      "R3": "off"
    }
   }
}
```

ActiveMQ es una implementación en Java de la especificación JMS, y fue usada en el proyecto por su integración con el resto de los módulos Java [Apache Software fundation (2017)].

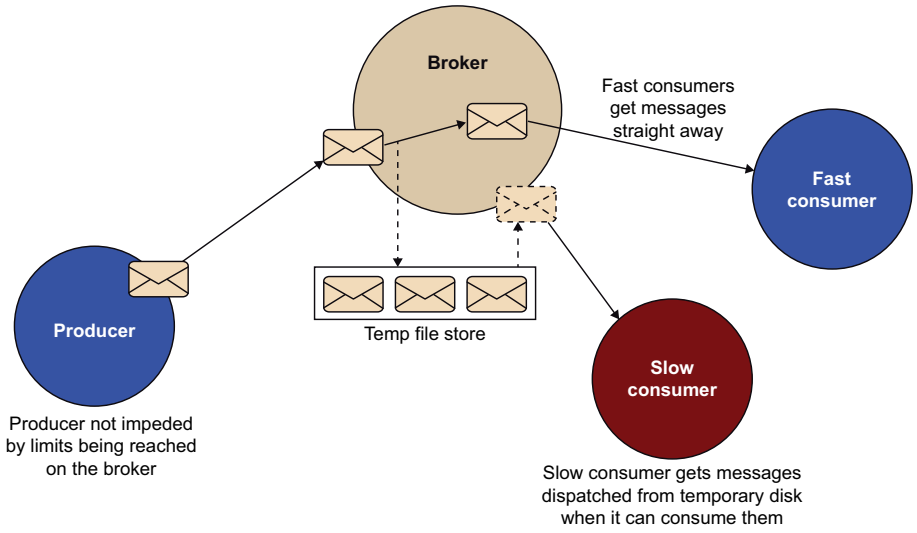
Cuando se usa un sistema de este tipo, conocido como *message-oriented middleware* o (MOM), la comunicación entre módulos o servicios no se hace directamente, sino a través de un intermediario o *broker* enviando mensajes.

Este *broker* recibe todos los envíos y es el encargado de hacerlos llegar a los módulos de destino. De esta forma, los módulos que desean intercambiar información quedan liberados de las tareas de sincronizarse e intercambiarse los datos (figura 4).

Con este método, no hace falta que los dos módulos que quieren intercambiarse datos estén conectados para hacerlo, ya que el *broker* se encarga de almacenar los mensajes si el destinatario no puede recibirlos en el momento en que se han enviado.

En algunos casos, además, se trabaja en modo publicación/suscripción, en el que los mensajes no se envían a un destinatario en concreto, sino que se etiquetan con un *topic* y distintos módulos pueden pedir recibir datos que contengan ciertas etiquetas o *topics*. De esta forma, tener una arquitectura que envía el mismo mensaje a varios módulos puede implementarse de manera sencilla, a la vez que es posible ampliar o añadir nuevos módulos o funcionalidades sin tener que cambiar otros módulos del sistema.

Figura 4. Esquema ActiveMQ



Gracias a estos tipos de comunicación (tanto API REST como JMS), permite la escalabilidad del sistema conforme las necesidades de cómputo o de almacenamiento vayan creciendo, ya que separar un servicio o módulo en distintas máquinas reales es sencillo y no implica cambios en el resto de los módulos.

4.4. Postgres SQL

Postgres SQL (abreviado, Postgres) es una base de datos relacional orientada a objetos. Permite almacenar y acceder a los datos mediante métodos estándar como, por ejemplo, lenguaje SQL [Group (2017)].

Gracias a su estrategia de replicación de datos en distintos servidores, puede usarse en aplicaciones en las que haya un solo proceso de escritura de datos y varios procesos que accedan a estos datos para solo leerlos.

4.5. Cassandra

También se usó Cassandra como base de datos en el proyecto. Véase el apartado «Cassandra» dentro del material titulado «Actualidad en *Cloud*» para más detalles.

Bibliografía

AAPA (2010). «AAPA World Port Rankings 2010».
URL: «http://aapa.files.cms-plus.com/Statistics/WORLD%20PORT%20RANKINGS%202010.pdf».

Apache Software Foundation (2017). «ActiveMQ webpage».
URL: «http://activemq.apache.org/».

Group, T. P. G. D. (2017). «PostgreSQL webpage».
URL: «https://www.postgresql.org/».

HPA (2014). «smartPORT logistics».
URL: «https://www.hamburg-port-authority.de/fileadmin/user_upload/150422_tl_messe_lowres.pdf».

Pivotal Software (2017). «Spring homepage».
URL: «https://spring.io/».