PORTAL DE ANUNCIOS

MindDen – Prueba técnica

Pedro Rafael Tarancón Baeza

21/02/2020

Contenido

1. Introducción

El objetivo de este documento es exponer la infraestructura propuesta para llevar a cabo la implantación de la aplicación del portal de anuncios.

El documento se divide en dos apartados. Primero, se detalla la solución implementada. Seguidamente, se define la arquitectura que se utilizaría en un entorno real.

Antes de continuar, cabe destacar que a la aplicación accederán usuarios de distinta índole. Por ello, a lo largo de los siguientes apartados se hablará de tres tipos de actores: cliente, supervisor y registrador. Este último actor no está explícitamente definido en los requisitos funcionales, pero será el encargado de publicar una nueva oferta en la aplicación.

2. Arquitectura implementada

En primer lugar, se analizará la implementación realizada. En este caso, se ha optado por la simplicidad, principalmente debido a dos motivos: la infraestructura disponible y no alargar en exceso el desarrollo de esta prueba técnica.

Para realizar la implementación, se ha decidido utilizar las siguientes tecnologías:

- **NodeJS**: se ha desarrollado un único API con tres módulos diferentes (/customer, /supervisor y /registrar) que se encargan de separar las funcionalidades de cada uno de los actores implicados en la aplicación. A su vez, dentro de cada módulo hay únicamente un endpoint encargado de realizar la función del actor correspondiente.

- **MongoDB**: para la persistencia de los anuncios se ha utilizado una Base de Datos (BD) NoSQL. Concretamente, se ha utilizado el proveedor mLab, que ofrece MongoDB como Database-as-a-Service.

- **Heroku**: para desplegar la aplicación se ha utilizado la plataforma Heroku. La ruta base de la aplicación es https://mindden-pruebatecnica.herokuapp.com/

- **Swagger**: también se ha utilizado Swagger para añadir una pequeña documentación de los endpoints disponibles en la aplicación y poder probarlos fácilmente. Está disponible en el siguiente enlace: https://mindden-pruebatecnica.herokuapp.com/swagger/

- **GitHub**: por último, el código fuente se ha alojado en un repositorio de GitHub, accesible mediante el siguiente enlace: https://github.com/pedrotarancon23/MindDen-PruebaTecnica

Está claro que esta arquitectura no es la mejor solución para implantar una aplicación de este tipo en un entorno real. Por ello, en el siguiente apartado se propondrá una arquitectura basada en microservicios que debería ser capaz de dar el servicio que se espera de la aplicación.

3. Arquitectura planteada

Para realizar la implantación de la aplicación se propone seguir una arquitectura basada en microservicios. El diagrama de alto nivel de la arquitectura propuesta es el siguiente:

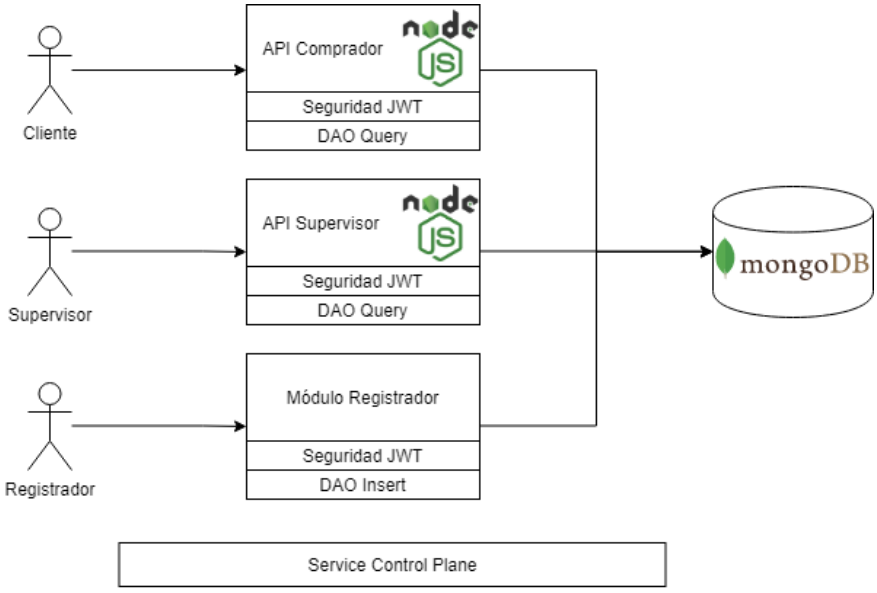

Figura 1 - Diagrama arquitectura

Se ha decidido dividir la aplicación en tres microservicios diferentes, uno para cada tipo de usuario que la utilizará (cada microservicio se correspondería con un módulo del API implementada en el apartado anterior).

Esta división se debe a que cada actor utilizará la aplicación de forma muy diferente:

- Los clientes realizarán muchas peticiones de consulta a la BD y desde orígenes muy diversos.
- Un registrador no realizará consultas, sino inserciones en la BD.
- Un supervisor debería acceder desde dispositivos controlados (desde su puesto de trabajo dentro de la empresa) y su tráfico será bastante menor que el de un cliente.

Por ello, realizando esta separación, se permite la configuración personalizada y se facilita el mantenimiento de cada servicio. Por ejemplo, se podría asignar más recursos al servicio del cliente porque su tráfico será mayor. O se podría aplicar un tipo de seguridad diferente para el servicio del supervisor, ya que puede que no sea necesario tener acceso a través de internet, entre otros.

Adicionalmente a estos servicios principales, en el diagrama se pueden observar otros servicios que servirán de apoyo para poder tener toda la funcionalidad deseada.

Por un lado, se ha decidido aplicar un patrón CQRS y separar el acceso a la BD para operaciones de lectura y para operaciones de escritura. Esto se debe a que seguramente el número de peticiones de lectura (realizadas por los clientes y supervisores) sea significativamente mayor al de peticiones de escritura (realizadas por los registradores) y pueda ser necesario gestionar ambas funcionalidades de forma independiente.

También es importante comentar que la integración con los servicios principales se hará siguiendo el patron Sidecar, ya que así se permite abstraer y reutilizar la lógica de acceso a BD.

Por otro lado, para aplicar una capa de seguridad a los recursos de cada tipo de usuario, se ha decidido utilizar la autenticación mediante tokens JWT. En este caso, se vuelve a utilizar el patrón Sidecar para añadir la seguridad a los servicios principales. Los tokens de autenticación generados serán diferentes en función del actor que los solicite.

Por último, también habría un componente Service Control Plane para realizar la orquestación y gestión de todos los servicios.

En cuanto a las tecnologías a utilizar, para los servicios de cliente y supervisor se propone la creación de dos API REST independientes e implementadas con NodeJS.

Sin embargo, para el servicio de Registrador se proponen dos soluciones en función del volumen de peticiones esperado.

- Si el volumen es relativamente bajo: API REST utilizando NodeJS.

- Si el volumen es alto: procesamiento de las peticiones utilizando Apache Kafka como encolador y Apache Storm para procesarlas (sería aquí donde se puntuarían los anuncios).

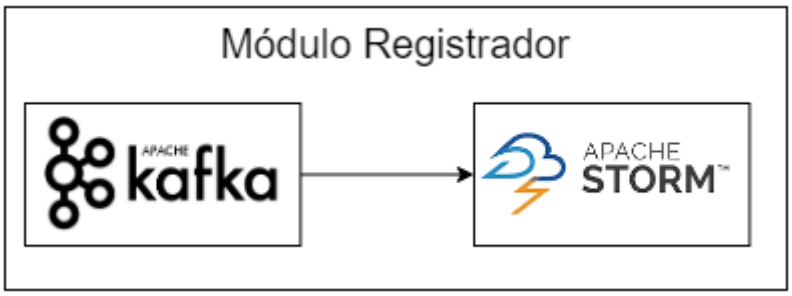

Figura 2 - Arquitectura para el módulo registrador

Seguramente en este caso sea exagerado utilizar Kafka y Storm porque en una aplicación de este tipo no suele haber tal nivel de peticiones. Pero en otro escenario similar sí podría ser útil si se tiene un flujo constante de recepción de información.

Se ha elegido Storm y no otra solución como Spark Streaming porque se ha considerado que para crear ofertas tiene más sentido procesar los eventos de forma individual que utilizando micro-batching.

En cuanto a la tecnología de BD, se ha optado por una solución NoSQL, concretamente MongoDB. El motivo es que se ha considerado que MongoDB se adapta perfectamente al modelo de datos de la aplicación, donde se tendría una colección de anuncios (y otra de imágenes) y cada documento puede tener campos distintos.

En el diagrama no se incluye, pero para almacenar la información de los usuarios clientes y compradores podría ser interesante utilizar una base de datos relacional (como MySQL o PostgreSQL), ya que el perfil de un usuario tiende a tener una estructura de información fija y sin grandes cambios a lo largo del tiempo. Mientras que, para los supervisores, si se trata de empleados de la propia empresa seguramente ya se disponga de algún tipo de autenticación de dominio como LDAP o Active Directory.

Por último, para gestionar todos estos servicios se propone utilizar un conjunto de herramientas formado por Docker para crear los contenedores necesarios para cada servicio, Kubernetes para desplegar estos contenedores en las máquinas dónde se despliegue la aplicación e Istio para gestionar la malla de servicios. Así como herramientas como Zipkin o Jaeger para facilitar monitorización de la arquitectura.

Para gestionar el versionado del código de cada servicio y su puesta en producción, se hará uso de Git y Jenkins. Además, para el almacenamiento de claves (como la clave para generar los tokens JWT o la clave de acceso a las distintas BD) se propone el uso de herramientas como Key Vault de Microsoft Azure o similares.

4. Conclusiones

A lo largo del documento se ha intentado exponer cómo se ha implementado una versión simplificada de la aplicación y, posteriormente, se ha realizado una propuesta para llevar a cabo el desarrollo de la aplicación mediante una arquitectura orientada a microservicios.

Para finalizar, simplemente añadir que la propuesta realizada es una aproximación para resolver el problema. En un proyecto real habría que analizar aspectos como el estado actual de la empresa. Ya que, por ejemplo, si hay varias tecnologías que permitan conseguir un mismo objetivo puede que haya preferencias por unas u otras en función de los recursos, tanto materiales como humanos, que se dispongan.