DOCUMENTACIÓN SERVICIOS WEB

ADRIAN MINGUEZ GRAÑA

MARIO MORAÑO ORVIZ

Contenido

PRESENTACIÓN DEL PROBLEMA 2
ANÁLISIS Y DISEÑO DEL PROBLEMA A RESOLVER 2
ARQUITECTURA DE LA SOLUCIÓN 4
INSTALACIÓN, DESPLIEGUE Y PUESTA EN PRODUCCIÓN 5

PRESENTACIÓN DEL PROBLEMA

Actualmente hay una creciente demanda de obtener contenido multimedia (películas y series) por parte de la población. Acceden a este contenido a través de tiendas físicas o plataformas de streaming multimedia. Muchas veces se crea confusión entre los usuarios porque no saben que elegir, comprar el producto o tener acceso a través de una plataforma.

Además, como cada vez se consume mas contenido multimedia, el público se vuelve cada vez más exigente y solo busca, en su mayoría, buscar películas o series de calidad.

Ante estos dos problemas, se ha decidido realizar una aplicación web que de información sobre la película deseada y muestre el precio en diferentes plataformas/tiendas.

Se ha decidido realizar dicha aplicación a través de servicios web. Se ha tomado dicha decisión porque la información se va a obtener de diferentes proveedores. Esto significa que en lugar de tener que revisar toda la información ante posibles cambios de precios o de valoración y modificarlos a mano, se expone un servicio web que devuelva dichos valores actualizados sin tener que preocuparse ante posibles modificaciones. Por lo tanto, se consigue ahorrar en tiempo y posibles problemas de sincronización.

ANÁLISIS Y DISEÑO DEL PROBLEMA A RESOLVER

Tabla 1: Servicios web externos utilizados

Proveedor	Servicio	Justificación	Utilización	Problemas
filmaffinity-api.mybluemix.net	Get byTitle	Para poder utilizar el servicio Get byId es necesario obtener la url a través de este servicio.	Se utiliza desde fuera de la aplicación, el cliente de escritorio solicita que se vaya a la página del servicio para coger la url necesaria para crear una película en base de datos.	Como busca una película por nombre, devuelve varias opciones y hay que ir una por una manualmente para saber si es la película que se quiere. Ese es el motivo por el que no se ha podido incluir dentro de la aplicación.
filmaffinity-api.mybluemix.net	Get byId	Es el servicio que va a proporcionar la información de Filmaffinity.	Se utiliza para sacar la información de Filmaffinity sobre la película seleccionada.	El formato de devolución del servicio es bastante malo, por lo que se ha tenido que convertir a un objeto Película, en lugar de enviar la información directamente.

Tabla 2: Servicios web de consumo interno

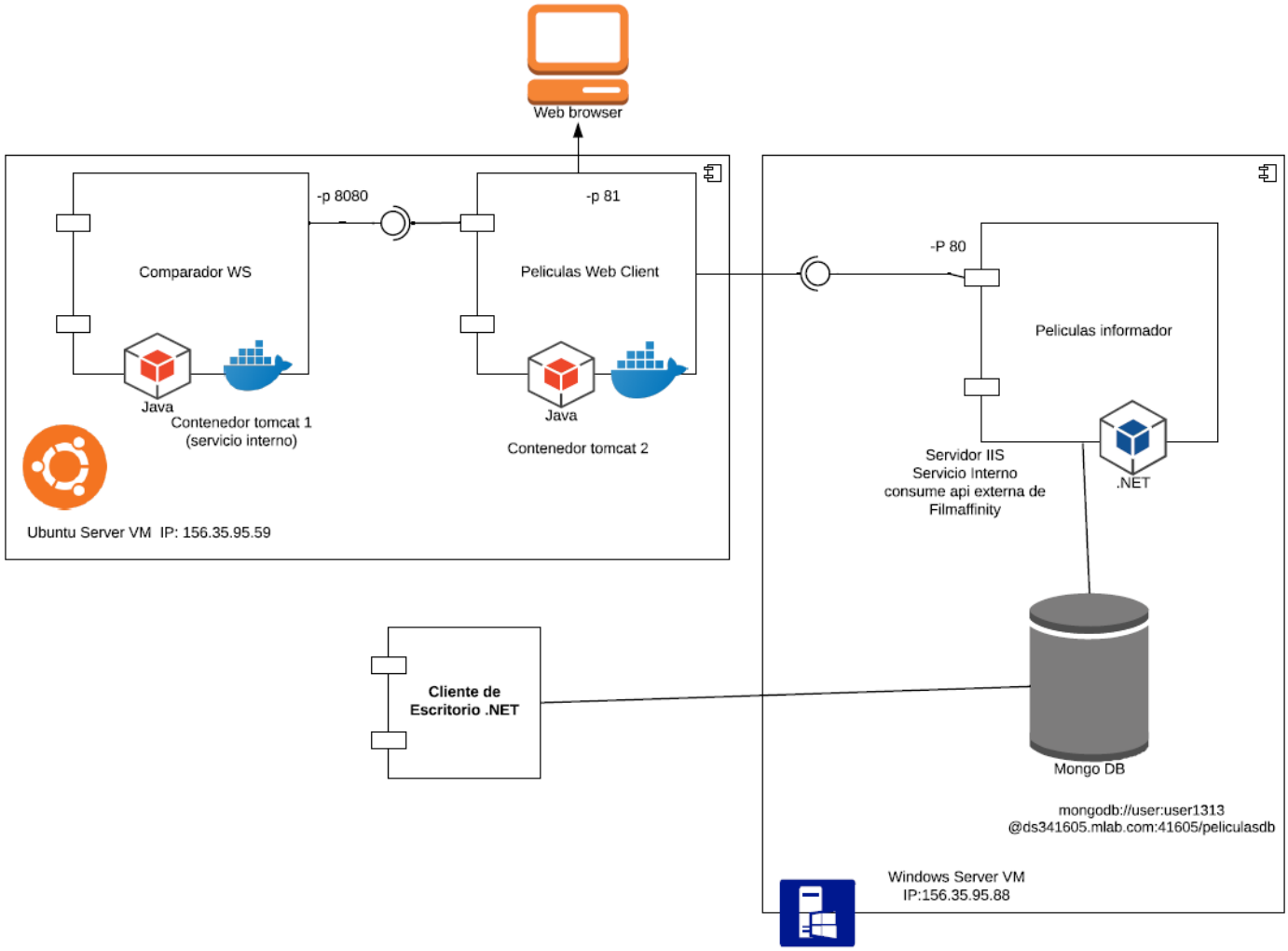
Servicio	Justificación	Utilización	Problemas	Donde se usa
Películas WS	Es necesario conocer el número de películas que se encuentran en base de datos.	A través del método Get se recoge el json con todas las películas y se almacena la propiedad Count en un int para conocer el número de películas.	Como devuelve un Array de Strings se tiene que convertir a Json.	Se utiliza en el cliente de escritorio para asignar el IdContador adecuado a una nueva película.
Get byId	Es el servicio que va a proporcionar la información de Filmaffinity.	Se utiliza para sacar la información de Filmaffinity sobre la película seleccionada.	El formato de devolución del servicio es bastante malo, por lo que se ha tenido que convertir a un objeto Película, en lugar de enviar la información directamente.	Se utiliza en el servicio películas WS para recoger la información de las películas.

Tabla 3: Servicios web ofrecidos para consumo externo

Servicio	Justificación	Utilización	Problemas	Para quien se ofrece
Comparador WS	Se necesitaba crear un servicio para comparar el precio de una película entre varios distribuidores.	Recoge el nombre de una película y devuelve el precio (de forma aleatoria) de dicha película en 6 distribuidores distintos.	Ninguno	El cliente web solicita la información de este servicio para mostrar los diferentes precios de una película concreta junto con la información de Filmaffinity.
Películas WS	Se necesita crear un servicio que proporcione información	Dispone de dos Get que devuelven diferente información.	Cuadrar la información que devuelve el servivio externo con el modelo de la aplicación	El cliente web recibe el primer Get con el listado y al elegir una

	sobre las películas.	**Get:** Devuelve un array de string con todas las películas que hay alamacenadas en base de dato. **Get(id):** devuelve la información de Filmaffinity para la película con el id introducido.	y la base de datos.	película solicita el Get(id). El cliente de escritorio utiliza internamente el Get para contar el número de películas almacenadas.

ARQUITECTURA DE LA SOLUCIÓN



INSTALACIÓN, DESPLIEGUE Y PUESTA EN PRODUCCIÓN

Hemos utilizado Maven como herramienta de "building" de los proyectos en lugar de copiar las dependencias. También hemos elegido una versión de Jersey más moderna que la expuesta en clase para consumir los servicios rest de java (2.25.1).

Hemos desplegado primero los servicios .NET en el servidor IIS de la máquina virtual Windows Server de la escuela, accesibles en el puerto 80 (servicios REST para la obtención de la entidad películas, tanto un listado como los detalles de la misma que son recogidos por la jsp). El cliente de escritorio es una aplicación .NET.

El servicio .NET consume a su vez un servicio externo de FilmAffinity.

A continuación, hemos desplegado el servicio interno SOAP java construido con Jaxws de la forma vista en clase. El war se deposita en un contenedor Tomcat generado a través de una imagen que copia las dependencias necesarias y deposita el binario en la carpeta webapps. El puerto expuesto es el que viene por defecto en la imagen oficial de Tomcat (8080).

Tras levantar el contenedor en la máquina virtual Ubuntu server de la escuela compilamos el cliente web, que es un servlet estándar, con una jsp, para gestionar los datos recogidos de ambos servicios. Se levanta otro contenedor en el puerto 81 para evitar conflictos con el anterior haciendo la sustitución del server.xml en el nuevo dockerfile (tras varias búsquedas hemos elegido la herramienta sed disponible en el contenedor Linux de Tomcat para realizar el "parseo" y la sustitución). De este modo el cliente web queda accesible en la ruta:

- **http://156.35.95.59:81/cliente-web**

Los servicios estan publicados respectivamente:

- **http://156.35.95.59:8080/comparador/soapws/comparador (comparador soap jaxws)**
- **http://156.35.95.88/WS.Rest.Peliculas/api/Peliculas (.NET REST)**

  Si se pide con /Peliculas devuelve el listado.

  Si se pide con /Peliculas/Id devuelve información de la película en concreto.

Entre los problemas encontrados destacan el lio de puertos cuando se intentan levantar varios contenedores Docker en la misma máquina. La imagen de Tomcat por defecto sólo expone el puerto 8080 y es necesario modificar el server.xml para no tener el mismo proceso Docker corriendo sobre dos puertos iguales (dará error).

Dificultad para averiguar qué puertos de la universidad no están cerrados.

Dificultad para levantar varios contenedores sobre el mismo puerto: como añadido se podría haber levantado un "reverse proxy" con Ngynx para solucionar este problema, y además, agregar un grado más de seguridad (cacheo de información). Por falta de tiempo no se ha realizado esta mejora.

Dificultad para que Jersey mapee correctamente los valores de los campos de los objetos del modelo correspondiente al servicio. Se debe tener mucho cuidado con la configuración del servidor y la librería Jersey. En nuestro caso para que reconociese los campos al deserializar hemos utilizado anotaciones a las propiedades del modelo.