**AOS: Práctica 2**

Despliegue y puesta en marcha de un API de gestión de productos

Miembros del grupo:

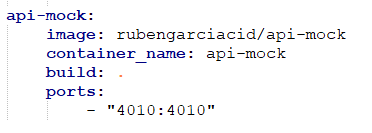
* Amit Karamchandani Batra
* Rubén García Cid
* Kristian Magjistari Ziu

1. **Memoria descriptiva del despliegue de los servicios**

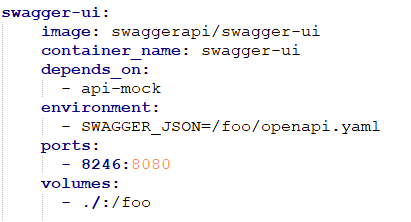
Descripción de los contenedores usados:

Para Docker-Compose:

* **rubengarciacid/api-mock**: En este contenedor, hemos usado una imagen de Node, con el paquete de Stoplight Prism preinstalado. Al arrancar el contenedor, se ejecutará una instancia de Prism en la dirección 127.0.0.1:4010, este servidor simulará el comportamiento de nuestra API, descrita en el fichero ‘openapi.yaml’, incluida dentro de la imagen.

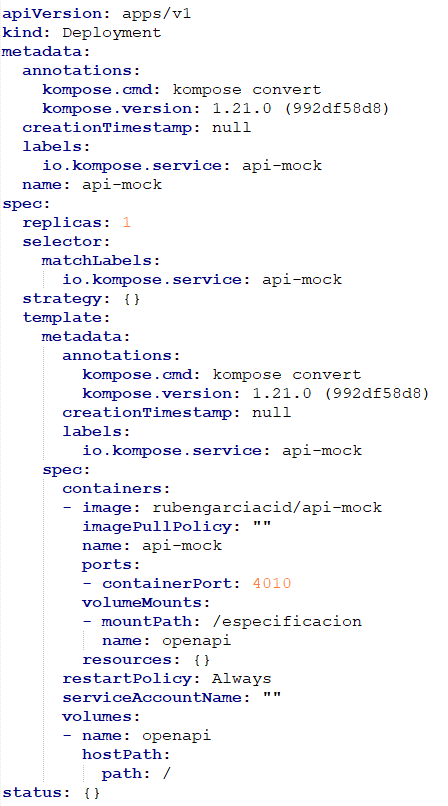


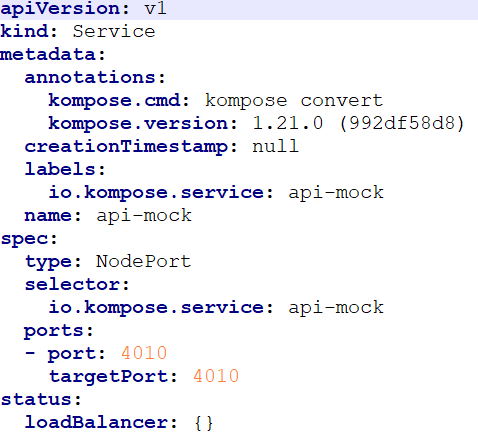
* **swaggerapi/swagger-ui**: Imagen oficial de Swagger UI, la hemos usado para poder interactuar con nuestra especificación, sin necesidad de haberla implementado previamente. De esta forma, podremos comprobar de una forma muy visual los distintos ‘endpoints’ expuestos en nuestra API, formatos de entrada y salida de los datos, las diversas operaciones a las que podemos llamar, los códigos de error que podemos obtener, entre otras cosas.



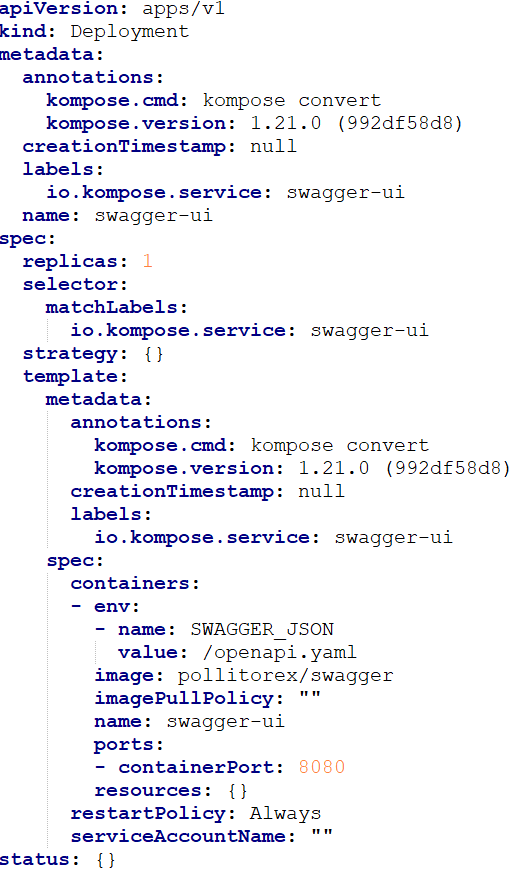
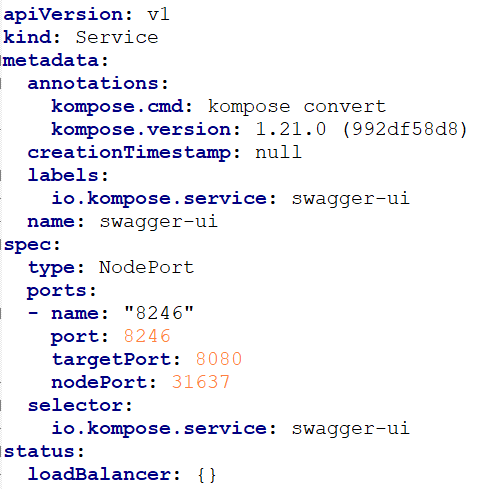
Para Kubernetes:

En config.yaml hemos especificado los servicios y el despliegue de los mismos siendo el api-mock y el swagger-ui.

* **longhar/api-mock:** En config.yaml hemos especificado los datos del api-mock de nuestro servicio. Hemos usado el servicio ‘mock’ de kubernetes y ejecutara la instancia del mock en la dirección 127.0.0.1:4010. Este ‘mock’ simulará nuestra API especificada en ‘openapi.yaml’.



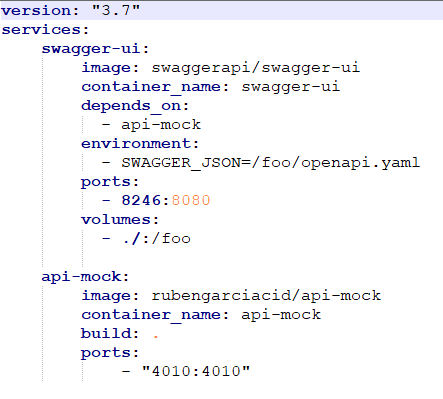
* **pollitorex/swagger-ui:** Hemos creado una imagen modificada de Swagger UI, hemos tomado como base la imagen oficial de Swagger-UI, introducimos el fichero ‘openapi.yaml’, en el que se encuentra nuestra especificación, dentro de la imagen, en la ruta ./usr/share/nginx/html/, para poder cargarla desde Swagger-UI, luego guardamos el contenedor en vivo en una imagen, usando el comando commit y lo subimos a un repositorio de DockerHub, mediante los comandos tag y push. Como hemos mencionado anteriormente, esta imagen la usamos para interactuar con nuestra especificación fácilmente. El servicio de Swagger UI se ejecutará en el cluster con puerto 8080, el puerto del pod es el 8246.

****

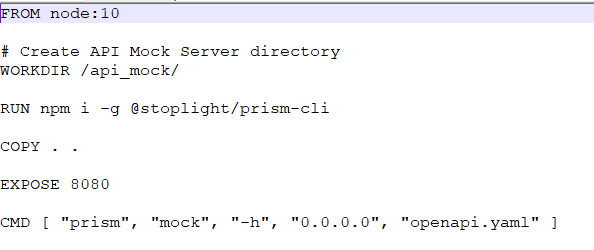
1. **Despliegue con Docker Compose**

Partiendo de la especificación corregida para solventar aquellos fallos que fueron observadores por el profesor, hemos añadido los siguientes ficheros a la carpeta llamada “Docker”, desde la cual se realizará el despliegue con Docker Compose:

* Un fichero YAML de docker-compose, en el cual definimos la especificación necesaria para iniciar todos los servicios de los que se compone nuestra aplicación, además de todos aquellos parámetros (dependencias con otros contenedores, variables de entorno, puertos, volúmenes, etc.) que se requieren para que puedan funcionar correctamente y puedan comunicarse entre ellos y con el host.



* El fichero Dockerfile usado para construir la imagen modificada de Prism, en la cual incluimos nuestra especificación para que pueda usarse durante el mock.



* Un fichero llamado ‘Run.bat’ que ejecutara los comandos (docker-compose build y docker-compose up) necesarios para poder arrancar nuestro conjunto de servicios correctamente.

1. **Despliegue con Kubernetes**

Hemos incluido todos los ficheros & ejecutables necesarios (carpetas de configuración, kompose.exe, kubectl.exe, etc) dentro de una carpeta llamada “Kubernetes”. Dentro de esta, tenemos 2 ficheros .bat.

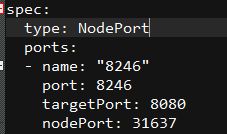
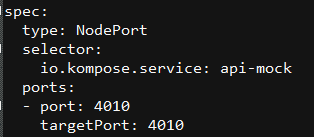
* Run.bat incluye la inicialización de Minikube y el comando de despliegue: 

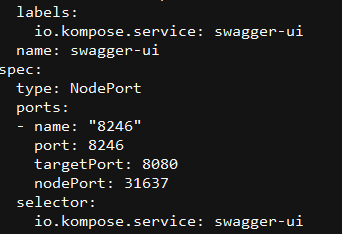
seguido de el comando:



con el cual se obtiene la IP y puerto del servicio de Swagger-ui.

* Stop.bat lo incluimos para hacer “Minikube stop” seguido de “Minikube delete” para parar rápidamente el cluster de Kubernetes si habíamos hecho algún cambio y queríamos ver el resultado.
* Un fichero “config.yaml” el cual contiene el código necesario de los servicios del ‘api-mock’ y del servicio de ‘swagger-ui incluido lo necesario para su despliegue. Hemos utilizado el tipo ‘NodePort’ para los dos servicios, para exponer la IP del nodo, de forma que pueda ser accesible desde fuera del cluster.





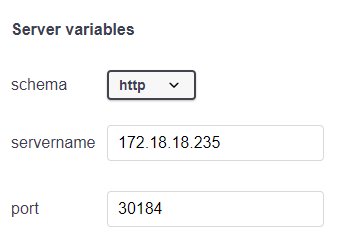
1. **Simulación del comportamiento de la API (Mocking)**

Hemos usado la herramienta de mocking ‘Prism’ para simular el comportamiento de nuestra API, para ello modificamos la imagen original mediante un fichero Dockerfile para incluir nuestra especificación dentro de ella, a esta imagen la hemos nombrado ‘api-mock’.

Para montar el servidor de Mocking, una vez puesto en marcha los servicios con el comando ‘Run.bat’, ya sea mediante Docker Compose o Kubernetes, debemos dirigirnos a la IP mostrada por pantalla tras ejecutar el script, una vez introducida dentro del navegador, necesitaremos también la IP del contenedor en el que se ejecuta api-mock para ello ejecutamos el siguiente comando:

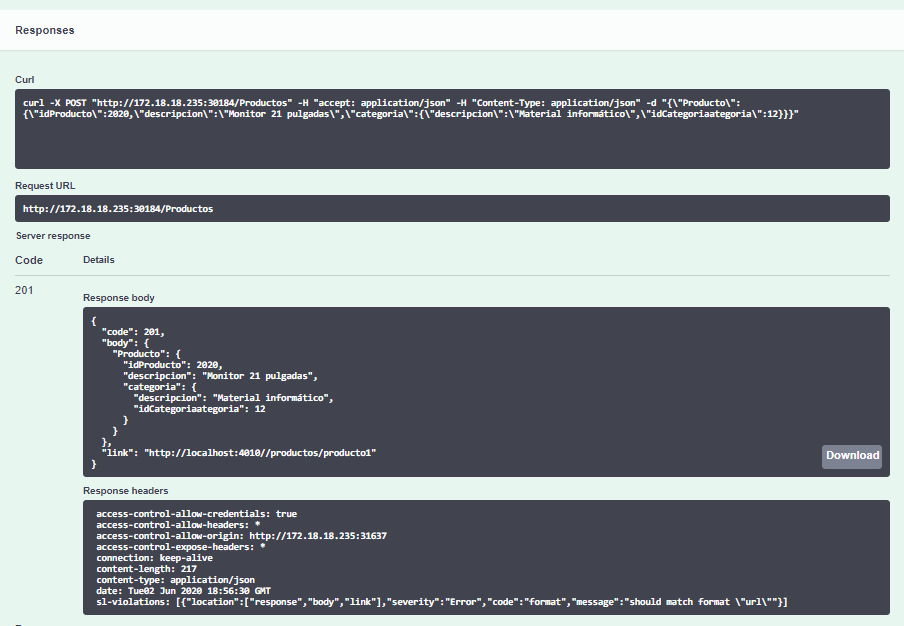


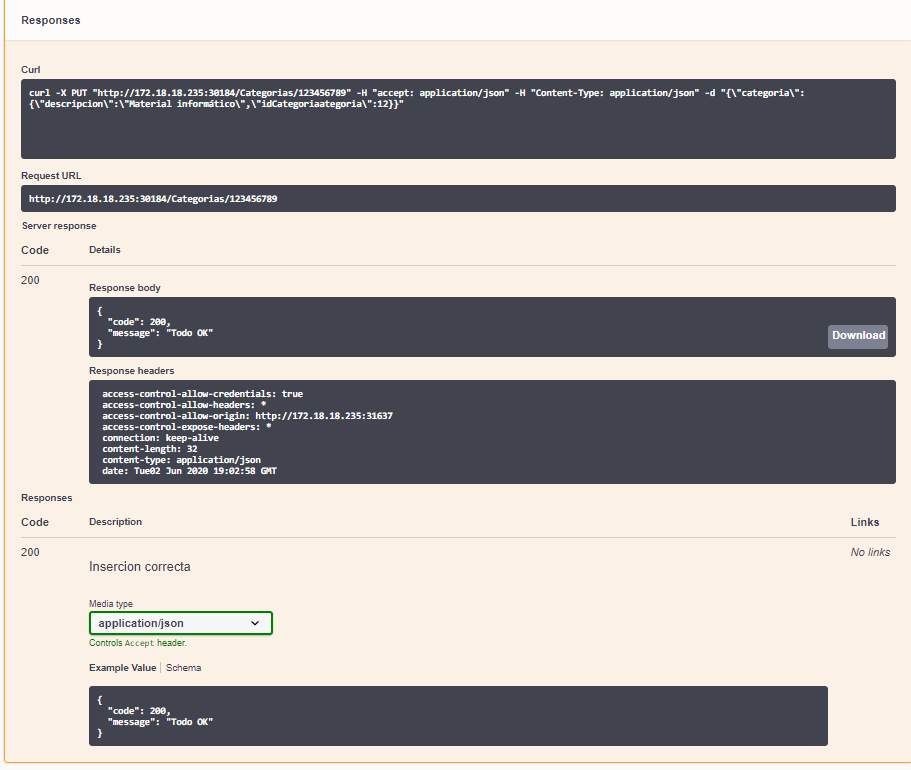
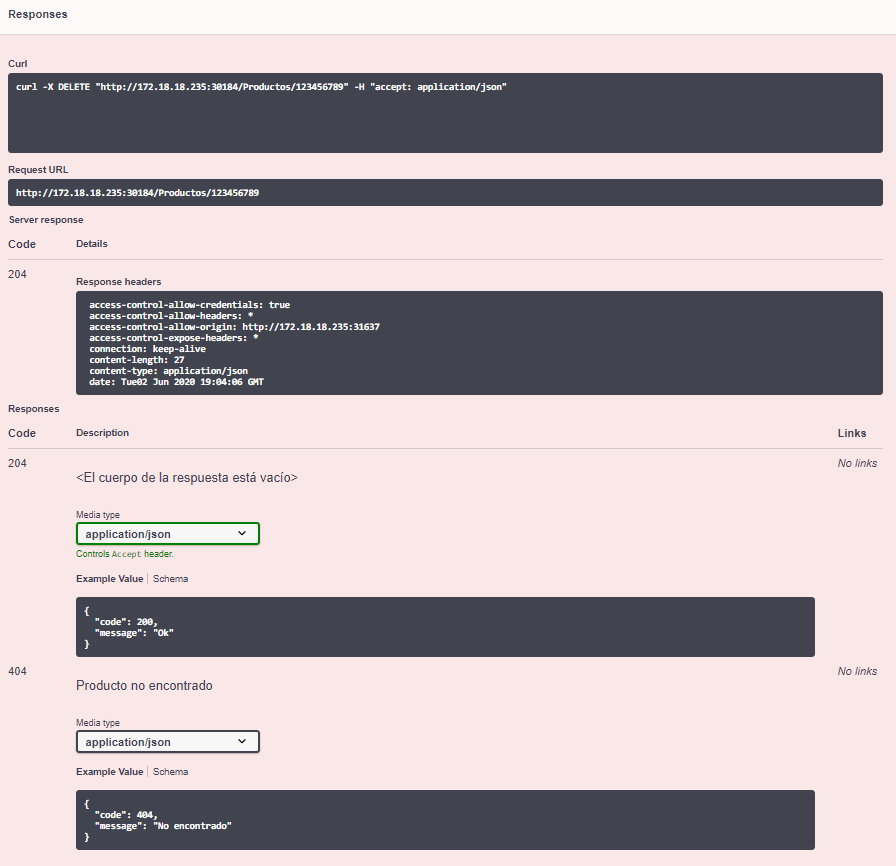
E introducimos la IP y el puerto en la sección ‘Server Variables’ en la interfaz de Swagger-UI:



La simulación, como era de esperar al no haber ninguna lógica implementada, no es completa, devuelve las respuestas esperadas cuando probamos operaciones como POST, PUT o DELETE, pero las operaciones GET devuelven un valor nulo.

Adjuntamos, capturas de pantalla de las pruebas con Prism:





1. **Implementación**

Hemos usado JavaScript como lenguaje de implementación, usando la plataforma Node JS para ejecutar el servidor como aplicación de escritorio, también hemos usado los paquetes Express, para montar el servidor web, y localStorage, para la persistencia de los datos que se introducen durante las peticiones al servidor.

El grado de implementación no es completo, en concreto, hemos implementado la respuesta en el caso de que la petición haya sido correcta para todas las operaciones, además, hemos implementado algunas respuestas ante casos en los que la solicitud no haya sido correcta o completa, según lo especificado en la API.

Posteriormente, hemos dockerizado la implementación del servidor, titulando la imagen como ‘api-server’, y hemos creado nuevas versiones de Docker Compose y Kubernetes, llamadas ‘Docker Compose – API Server’ y ‘Kubernetes – API Server’, en las cuales se han sustituido el contenedor ‘api-mock’ por la imagen de nuestro servidor implementado.

Para probarlo, se deben seguir los mismos pasos descritos en el apartado anterior.