PRÁCTICA CREATIVA 2: DESPLIEGUE DE UNA APLICACIÓN ESCALABLE

Toda la práctica está contenida en el siguiente repositorio de Github: https://github.com/Alfesito/PC2

Despliegue de la aplicación en máquina virtual pesada

Descripción de la aplicación
Para el despliegue de esta aplicación monolítica es necesario tener en cuenta que versiones posteriores a Python 3.9 dan problema, por lo tanto, es importante asegurarse que el sistema operativo donde queramos ejecutar el servicio no tenga Python 3.10 instalado.

El script consta de tres partes:

1. Instalación de pip e instalación de librerías con el archivo requirements.txt. Alguna versión del fichero anterior da problemas, por lo que se instalará posteriormente por separado con el comando *$pip install [librería].*
2. Se crea una variable de entorno llamado GROUP_NUMBER, que se utiliza para cambiar la etiqueta html *<title>* de los archivos productpage.html e index.html, encontrados en la carpeta *templates*.
3. Una vez que tenemos todas las librerías instaladas y la variable de entorno declarada, ejecutamos la aplicación con el comando (siendo el puerto donde desplegamos la aplicación en el 80, el cual se utiliza para http):

    $python3 productpage_monolith.py 80

Con todo ello, como se ejecuta la aplicación correctamente en el navegador: http://localhost:9080/productpage.

Despliegue de una aplicación monolítica usando docker

Descripción de la aplicación

Cuando hemos conseguido desplegar la aplicación mediante una máquina virtual pesada, hemos procedido a hacer el mismo despliegue, pero esta vez utilizando la herramienta docker. El fichero Dockerfile contiene todas las directivas para el despliegue en local, usando así la tecnología de virtualización ligera.

La estructura del fichero en Docker es la siguiente:

1. Instalación de git y clonación del repositorio https://github.com/CDPS-ETSIT/practica_creativa2
2. Se cambian ciertas versiones que dan problemas en el requirements.txt y se instalan las librerías necesarias.
3. Se expone el puerto 9080, se crea la variable de entorno y se ejecuta el servicio.

Para poder ejecutar correctamente el fichero Dockerfile debemos antes crear la imagen, arrancar el navegador con el puerto 9080 y después abrir en el navegador: http://localhost:9080/productpage

Incluir la línea de comando del despliegue del contenedor en la memoria.

Para poder desplegar el servicio monolítico en docker hay que seguir los siguientes pasos:

1. Crear la imagen:

```
$docker build -t g40/product-page .
```

2. Arrancar el contenedor con el puerto 9080:

```
$docker run --name g40-product-page -p 9080:9080 -e GROUP_NUMBER=40 -d g40/product-page
```

3. Abrir el navegador: http://localhost:9080/productpage

Segmentación de una aplicación monolítica en microservicios utilizando docker-compose

Descripción de la aplicación

Para poder usar varios servicios en docker-compose, primero es necesario definir en un Dockerfile cada uno de los servicios para que así, docker-compose pueda construir la imagen y después corra un contenedor con dicha imagen.

Para el correcto funcionamiento de la aplicación, cada servicio se distinguirá con un puerto diferente en docker-compose, además de otros aspectos que se indican en el enunciado de la práctica. Además, cambiando las variables de entorno de *reviews* podemos definir los tres tipos de versiones que se especifican.

Incluya en la memoria de la práctica las diferencias con la versión de un único contenedor.

El uso de varios contenedores para separar servicios es una buena praxis en muchos aspectos como:

- Permite un mejor seguimiento de las actualizaciones, sin que ningún otro servicio se vea afectado en caso de error.
- Código más legible ya que está solo enfocado a un servicio. Mientras que en un servicio monolítico puede ser algo confuso ya que mezcla varias utilidades en un mismo código.
- Permite que en caso de fallo de un servicio, los demás puedan funcionar si no dependen del que ha fallado.

Incluir en la memoria la línea del despliegue del docker-compose.

Para poder arrancar los distintos servicios con docker compose hay que seguir los siguientes pasos:

1. Crear un Dockerfile para cada servicio, según las características de este.
2. Crear un archivo .yaml llamado docker-compose, con el que se pueda construir las imágenes de cada servicio con su respectivo Dockerfile. Para poder así arrancar los contenedores con dichas imágenes.
3. El comando para poder ejecutar el docker-compose es:

```
$docker-compose up
```

Despliegue de una aplicación basada en microservicios utilizando Kubernetes

Incluya en la memoria de la práctica las diferencias que encuentra al crear los pods, así mismo la diferencia que ve para escalar esta última solución.

Al trabajar con pods en Kubernetes, es importante considerar la cantidad de recursos necesarios para el funcionamiento de los contenedores, así como la configuración de red y almacenamiento. Una de las decisiones clave es si se requiere un solo pod o varios para una aplicación. Un solo pod es más simple, pero a su vez, tiene un mayor riesgo de fallos. En cambio, varios pods proporcionan mayor tolerancia a fallos pero requieren una mayor configuración y mantenimiento.

Además, para escalar los pods, Kubernetes utiliza mecanismos como los *Replication Controllers* y los *Deployments*, estos garantizan que un número específico de réplicas estén siempre ejecutándose y proporcionan una interfaz para actualizar fácilmente los pods.

Incluir en la memoria la línea del despliegue de los ficheros de configuración y definición de pods y servicios en la infraestructura de kubernetes.

Para desplegar un pod, se puede utilizar el comando "kubectl apply" seguido del archivo de configuración del pod en formato YAML.

```
$kubectl apply -f mi-pod.yaml
```

Para desplegar un servicio, se puede utilizar el comando "kubectl apply" seguido del archivo de configuración del servicio en formato YAML.

```
$kubectl apply -f mi-servico.yaml
```

También es posible utilizar Helm para manejar y desplegar los paquetes de configuración y definiciones de pods y servicios de manera más sencilla y mantenible.