Administración de Sistemas - Curso 2023 / 2024

Laboratorio 6.- Ecosistema Docker

Contenido:

Objetivos: Conocer y configurar algunas de las herramientas del ecosistema Docker para gestionar contenedores.

Administración de Sistemas - Curso 2023 / 2024

1 Preparar el entorno

Para realizar este laboratorio es necesario un entorno con Docker Engine (DE) instalado que incluya Compose. Ver Sección 1 del Laboratorio 5 para información sobre instalación.

Por otra parte, en este laboratorio se van a utilizar las siguientes herramientas ajenas a Docker, y se recomienda buscar información en internet y familiarizarse con ellas en este paso:

- Redis (https://redis.io/): Base de datos en memoria para almacenamiento en formato clave/valor.
- Flask (https://palletsprojects.com/p/flask/): Framework para desarrollo de aplicaciones web.

2 Docker Compose

En esta parte del laboratorio se va a crear una aplicación web con 2 contenedores Docker y Compose. El objetivo es crear una web muy simple con un contador de visitas, como la que se muestra aquí:

Uno de los contenedores será un servidor web con Flask, y el otro contenedor almacenará el número de visitas utilizando Redis.

Para comenzar, crear un nuevo directorio en vuestro sistema y crear los siguientes ficheros en él:

- Un fichero llamado "app.py" con el codigo Python incluido al final de esta sección. Este código crea un servidor web utilizando Flask que se conecta a Redis para leer y almacenar el número de visitas.
- Un fichero llamado "requirements.txt" dentro del directorio con 2 líneas: "flask" y "redis".
- Un fichero Dockerfile con las siguientes características:
  - Imagen base: python:3.11-alpine
  - Establece como directorio de trabajo /code
  - Establece dos variables de entorno con los siguientes valores:
    - FLASK_APP=app.py
    - FLASK_RUN_HOST=0.0.0.0
  - Ejecuta el siguiente comando para instalar dependencias:
    - `apk add --no-cache gcc musl-dev linux-headers`
  - Copia el fichero requirements.txt dentro de /code
  - Ejecuta el siguiente comando para instalar dependencias: pip install -r requirements.txt
  - Copia el fichero app.py dentro de /code.
  - Estable como comando de arranque "flask run"
- Un fichero YAML para Docker Compose con las siguientes características:
  - Un servicio llamado "web". Deberá crearse una imagen Docker usando el Dockerfile recién descrito. Redirigir el puerto 5000 del contenedor al puerto 80 del anfitrión.
  - Un servicio llamado "redis". Deberá obtener la imagen "redis:alpine" de Docker Hub.

Administración de Sistemas - Curso 2023 / 2024

Una vez que los ficheros estén creados, el siguiente paso es comprobar que la aplicación funciona:

- Desde el directorio creado al inicio, lanzar los contenedores con Docker Compose en segundo plano.
- Abrir un navegador en la dirección http://VUESTRA-IP/. Se debería mostrar una web con el mensaje "Hola! Este sitio se ha visitado 1 veces".
- Una vez que la web esté visible en el navegador, refrescar la página varias veces. El contador de visitas se debe incrementar cada vez que se refresque la web.

En caso de que la web no se muestre, revisar los pasos anteriores y buscar mensajes de error de Docker o Docker Compose para solucionar problemas.

Tras comprobar la funcionalidad de la web, realizar las siguientes tareas:

- Parar los contenedores en marcha.
- Utilizar Docker Compose para verificar el número de imágenes creadas y su tamaño. ¿Cuál es la imagen de mayor tamaño?
- Realizar los siguientes cambios en el fichero YAML de Docker Compose, en el contenedor "web":
  - Montar el directorio de trabajo del anfitrión (que contiene el Dockerfile y resto de ficheros) como un *bind mount* en el directorio /code del contenedor.
  - Añadir una variable de entorno "FLASK_DEBUG" con el valor "1".
- Relanzar los contenedores con Docker Compose.
- Verificar con el navegador que la web está funcional y que, al refrescar, el número de visitas se incrementa.

Si todo se ha configurado correctamente, al modificar el código de la aplicación web directamente desde el anfitrión, el contenedor "web" lo leerá gracias al *bind mount*. Al haber definido la variable "FLASK_DEBUG", el servidor web aplicará los cambios realizados sin tener que reiniciar el servicio. Para verificar que esto es funcional:

- Desde el anfitrión, con los contenedores en marcha, modificar el mensaje en la última línea del fichero "app.py". Por ejemplo, cambiar "Hola!" por un saludo más personalizado.
- Utilizando un navegador en el anfitrión, verificar que el mensaje se ha cambiado en la web.

Para acabar esta parte del laboratorio, se propone trabajar con Docker Compose y las diferentes políticas de reinicio de los contenedores:

- Modificar el fichero "app.py":
  - Añadir una nueva línea al comienzo del fichero con "import os"
  - Añadir una nueva línea con el texto "os._exit(5)" inmediatamente después de la línea "def hello():". Debido a las restricciones de Python, la nueva línea deberá estar indentada (con espacios o tabulador) al mismo nivel que las líneas siguientes.
- Abrir el navegador en el anfitrión en la URL http://VUESTRA-IP/. Se debería mostrar un error o la web no debería cargar.
- Modificar el fichero YAML de Docker Compose: añadir la política de reinicio "on-failure" al contenedor "web".
- Relanzar los contenedores y abrir el navegador en la URL de nuevo (o refrescar la página si estaba abierta). Fijarse en los logs de Docker Compose. ¿Qué sucede?
- Parar los contenedores y reemplazar la línea "os._exit(5)" del fichero "app.py" por "os._exit(0)".
- Abrir de nuevo el navegador (o refrescar la página si estaba abierta). En este caso, ¿se han reiniciado los contenedores? ¿Qué sucede?

En caso de duda, se recomienda buscar información en internet sobre el efecto del comando "os._exit()" de Python y sobre la gestión de errores de salida en procesos Linux/Unix.

Código Python para el fichero "app.py":

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)


@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hola! Este sitio se ha visitado {} veces.\n'.format(count)
```

3 Docker Hub

En esta parte del laboratorio se trabaja con Docker Hub, el registro oficial de Docker. El objetivo es familiarizarse con la plataforma y realizar operaciones de gestión básica con los contenedores que se han utilizado en el parte anterior del laboratorio.

Realizar las siguientes tareas:

- Dirigirse al portal de Docker Hub (https://hub.docker.com/) y crear una cuenta de nivel gratuito. Elegir un Docker ID representativo.
- Crear un repositorio con nombre "as-laboratorio-6-web".
- En vuestra máquina virtual:
  - Ejecutar "docker login --username ‹vuestro-Docker-ID›" para enlazar el cliente Docker con vuestro usuario en Docker Hub.
  - Etiquetar la imagen utilizada como servidor web en la parte anterior del laboratorio como "vuestro-Docker-ID/as-laboratorio-6-web".
  - Utilizando "docker push", subir la imagen a Docker Hub.
- Verificar en vuestro perfil de Docker Hub que la imagen se ha subido correctamente.
- Buscar en DockerHub el tamaño que ocupa la imagen. ¿Coincide con lo que ocupa en vuestra máquina?

A partir de ahora, se pueden subir sucesivas versiones de la imagen utilizando "docker push" a Docker Hub.

Administración de Sistemas - Curso 2023 / 2024

Además de almacenamiento de imágenes, Docker Hub ofrece otras funcionalidades, p.e.:

- Construcción automática de imágenes a partir de cambios en un repositorio de código. Esta funcionalidad sólo está disponible para usuarios de pago de Docker Hub.
- Disparar eventos cuando se sube una nueva versión de una imagen utilizando WebHooks, llamadas HTTP que Docker Hub envía por nosotros a servicios Web de terceros.

Para probar los WebHooks, se propone crear una aplicación que nos envíe un e-mail cada vez que se suba una nueva versión de la imagen al repositorio. Crearemos esta aplicación utilizando la herramienta Apps Script de Google.

El primer paso es entrar en la consola de Apps Script a través de https://script.google.com/ y crear un nuevo proyecto. Se abrirá una ventana como la siguiente:

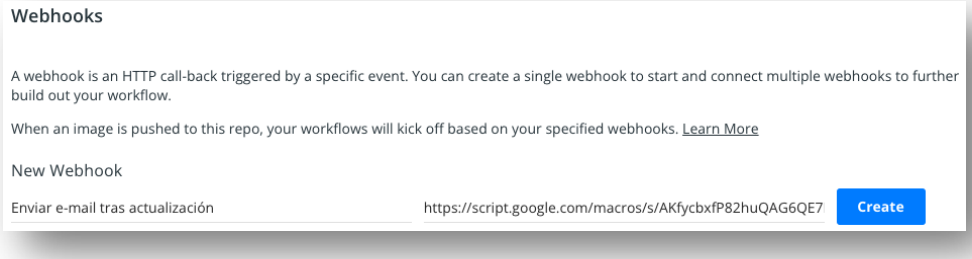
En esta ventana:

- Primero, asignar un nuevo nombre al proyecto, haciendo click sobre el mismo.
- Reemplazar el código "myFunction" por el código que se incluye al final de esta sección. Debéis indicar la dirección de e-mail en la que queréis recibir los correos en la línea 11 del código.

Después, pulsar el botón Ejecutar ubicado en la parte superior del código. Al hacerlo, Google os pedirá permisos para poder enviar correos con vuestro nombre. Concederlos para poder crear esta aplicación Web.

El siguiente paso es publicar este código en forma de aplicación para que pueda recibir WebHooks. Para ello, crear una nueva implementación haciendo click en el botón "Implementar" que se encuentra en la parte superior derecha. Elegir "Aplicación Web" como tipo, indicar que se debe ejecutar con vuestra cuenta y que cualquier usuario puede tener acceso.

En el último paso del asistente se mostrará una URL que comienza con https://script.google.com/... Abrir la pestaña WebHooks de vuestro repositorio en Docker Hub y crear un nuevo WebHook utilizando esta URL:



Una vez que el WebHook está configurado en DockerHub y la aplicación de Apps Script en marcha, realizad un cambio en el código del ejercicio anterior, crear la imagen de nuevo y subirla al repositorio de Docker Hub. Debéis recibir un correo en la dirección que hayáis indicado en el código de Apps Script.

Código para Apps Script:

```
function doPost(request){
  var postJSON = request.postData.getDataAsString();
  var payload = JSON.parse(postJSON);
  var tag = payload.push_data.tag;
  var reponame = payload.repository.repo_name;
  var dockerimagename = payload.repository.name;


  if(typeof request !== 'undefined')


  MailApp.sendEmail({
     to: "<INDICAD-VUESTRA-DIRECCION-DE-EMAIL>",
     subject: "Nueva imagen subida al repositorio "+reponame,
     htmlBody: "Hola,<br>"+
               "Se ha actualizado el repositorio Docker Hub: "+reponame+"<br>"+
               "<strong>Imagen: " + dockerimagename+"<br>"+
               "Version:" +tag+"<br></strong>"
    });
}
```

4 Atribución

Este laboratorio está basado en el siguiente material:

- Documentación oficial Docker, "Get started with Docker Compose", consultado en octubre 2020: https://docs.docker.com/compose/gettingstarted/
- Documentación oficial Docker, "Docker Hub Quickstart", consultado en octubre 2020: https://docs.docker.com/docker-hub/
- Stephen Grider, "Docker and Kubernetes: The Complete Guide", Udemy, consultado en octubre 2020: https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide
- Sunish Surendran, "DockerHub Webhook with Google Script", consultado en octubre 2021: https://youtu.be/obj3lakmFxQ , https://github.com/sunishsurendrank/GoogleScript