Práctica creativa 2

Grupo 39 - Oscar.perez.arruti@alumnos.upm.es - g.yun@alumnos.upm.es

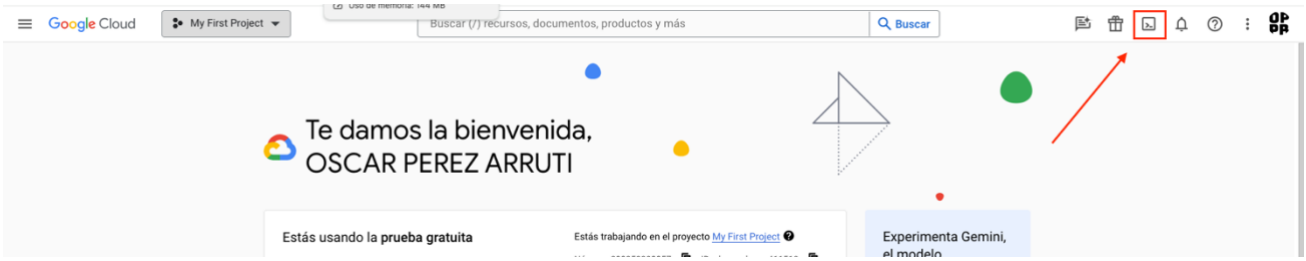
CONTENIDOS

Práctica creativa 2

Warning

Toda la práctica se ha desarrollado con máquinas virtuales (VM) de tipo E2 en Ubuntu (igual que las utilizadas en las prácticas del curso), a excepción de la última automatización, que está diseñada para ejecutarse directamente en la Cloud Shell de Google Cloud, la cual cuenta con todos los permisos necesarios para administrar todas las APIs. Por tanto, es importante ejecutar todos los scripts en la instancia VM de Google Cloud con Ubuntu y realizar la última automatización en la Cloud Shell, donde se solicitará dicha automatización.

En la siguiente figura se muestra dónde encontrar la Cloud Shell.



Información relevante

Todo el código está disponible en el repositorio de GitHub, https://github.com/oscarperezagg/Creativa2. Consideramos que esta es la forma más sencilla de probar el código. En el caso de las instancias VM de Google Cloud y la Cloud Shell de Google Cloud, el programa Git viene ya instalado, por lo que la mejor forma de probarlo es clonar el proyecto y acceder a las carpetas de los cuatro proyectos. Es importante destacar que todos los proyectos están diseñados para ejecutarse desde su respectiva carpeta y no desde la carpeta raíz del proyecto.

Despliegue de una aplicación escalable

Despliegue de la aplicación en máquina virtual pesada

El script desarrollado es realmente sencillo y realiza las siguientes acciones:

1. Verifica si pip está instalado (ya que no viene preinstalado en la máquina de Google Cloud) y, de no estarlo, lo instala.
2. Clona el repositorio de GitHub que contiene el proyecto practica_creativa2.
3. Instala los requerimientos del producto monolítico "productpage" y corrige los errores relacionados con las versiones de los paquetes de Python.
4. Ejecuta el archivo "productpage_monolith.py" en el puerto 9080.

Es crucial recordar que, al utilizar instancias de Google Cloud, se debe crear una regla de firewall para el puerto 9080, ya que por defecto está bloqueado.
Este script es bastante sencillo, ya que solo implica la ejecución de un archivo de Python. Sin embargo, al igual que otros grupos, hemos encontrado problemas al trabajar con diferentes versiones de Python. No estamos seguro si es cuestión de las máquinas virtuales o si se debe a las versiones.

Despliegue de una aplicación monolítica usando Docker

A lo largo de esta automatización, hemos definido tanto el Dockerfile solicitado como una automatización de Python que permita ejecutarlo todo sin mayores problemas que escribir el comando.

En este caso, la mayor parte del trabajo la realiza el Dockerfile, que se encarga de instalar los paquetes, instalar pip, clonar el repositorio y, finalmente, ejecutar la aplicación. El script de Python, en realidad, solo ejecuta los comandos que permiten construir la imagen (docker build) y correr el contenedor basándose en dicha imagen. Se requiere definir una variable de entorno; aunque la definición se realiza en el Dockerfile, es en el script de Python, durante la ejecución del contenedor, donde se asigna valor a dicha variable.

Segmentación de una aplicación monolítica en microservicios utilizando docker-compose

En este caso, nuevamente hemos desarrollado un script y los Dockerfiles para los servicios solicitados. Bajo el supuesto de que los Dockerfiles y el docker-compose.yml ya están definidos, el script es sencillo pero realmente efectivo. Los pasos que lleva a cabo son:

1. Ejecución de la función prune_docker, que, haciendo uso de la opción prune en los contenedores e imágenes, elimina todos los contenedores e imágenes. Esto se hizo con el fin de evitar problemas entre las distintas automatizaciones y poder ejecutar una y otra vez sin problemas.
2. Se ejecuta la función principal run_commands(), que duplica el repositorio de GitHub, se mueve hasta la carpeta practica_creativa2/bookinfo/src/reviews donde se compila y empaqueta el proyecto. Tras esto, se vuelve al directorio principal donde se ejecuta el build del docker-compose. Tras esto, se ejecuta el docker-compose con la opción up. Aquí deberíamos tener la aplicación corriendo.

El único reto de esta automatización es hacer buen uso de las variables de entorno para poder trabajar con las distintas versiones. Hemos conseguido, a través de todas las variables, que todos los servicios, tanto el de detalles, reviews y ratings, estén activos, pero no hemos logrado trabajar con las distintas versiones de ratings.

Para la creación de los Dockerfiles, hemos adoptado el enfoque de duplicar localmente el repositorio de la práctica, con el objetivo de aumentar la velocidad y evitar duplicar el repositorio en cada Dockerfile. Ha sido con este repositorio clonado que hemos copiado los archivos que se pedían, es decir, hemos realizado la copia de archivos de manera local en vez de clonar dentro de los contenedores el repositorio. Consideramos que este método es más útil. Por lo demás, creemos que los Dockerfiles no son realmente complicados.

Despliegue de una aplicación basada en microservicios utilizando Kubernetes

En la última automatización que hemos implementado, como indica su nombre, se buscó la máxima automatización posible. Logramos que la única interacción humana necesaria sea habilitar la API de Kubernetes en Google Cloud y autorizar a Google Cloud para el uso de sus servicios al inicio de la ejecución.

El proyecto que hemos creado es “bastante complejo”. Los archivos de configuración (yml y yaml) de los servicios y los despliegues se encuentran en el archivo raíz, junto con el script principal. Posteriormente, hay dos carpetas: una llamada 'config' y otra 'dockerfiles'. En 'config', se encuentra el archivo 'secret.py' (que no está incluido en la clonación), almacenando únicamente dos variables en texto claro (aunque no es lo ideal), 'username' y 'password', que contienen las credenciales de Docker Hub. En la otra carpeta, 'dockerfiles', se encuentran los archivos necesarios para crear las imágenes y subirlas a Docker Hub. Más adelante explicaremos la razón detrás de este enfoque.

Es crucial entender el procedimiento del script. Primero, se debe iniciar sesión en Google Cloud, activar la API de Kubernetes y luego activar la Cloud Shell. Aquí, se ejecuta 'git clone https://github.com/oscarperezagg/Creativa2', asegurándose de usar nuestro repositorio. Luego, se cambia a la carpeta 'Creativa2/Kubernetes' y se ejecuta 'python3 kubernetes.py'. Al iniciar el script, aparece un aviso que indica la necesidad de definir un archivo 'secret.py' en la carpeta 'config' con las credenciales de Docker Hub (hay un archivo 'secret_template.py' como ejemplo) y que se debe modificar el script 'kubernetes.py' para usar un código de proyecto válido en Google Cloud (cualquiera que lo ejecute fuera de mi Google Cloud deberá hacerlo según indica el aviso).

Una vez ejecutado el script, si todo está configurado correctamente, aparecerá una ventana emergente solicitando permiso para usar los servicios de Google Cloud, que debe ser autorizado. En caso de problemas, se puede reiniciar la Cloud Shell. Una vez autorizado, solo es cuestión de esperar para que se cree el clúster 'Creativa2' y se configure todo lo necesario.

Es importante utilizar una cuenta real de Docker Hub; de lo contrario, el proceso no funcionará. Decidimos usar un repositorio en lugar de trabajar con imágenes locales (Por la infinidad de errores que esto últimos no producía). Este script crea las imágenes, las sube a nuestro repositorio y luego aplica los archivos 'yml' al clúster. Finalmente, proporciona información sobre los servicios, despliegues y pods, y tras un tiempo, la dirección pública del servicio 'productpage'.

El script realiza las siguientes acciones:

1. Verifica si Docker está instalado, y si no, lo instala.
2. Comprueba si Kubernetes está instalado, y si no, lo instala.
3. Pregunta si se desea crear el contenedor 'Creativa2' para evitar errores al intentar recrearlo. Si se confirma, es aquí donde se debe autorizar.
4. Se conecta con el clúster obteniendo las credenciales.
5. Limpia todos los servicios, despliegues y pods de Kubernetes y elimina todas las imágenes y contenedores de Docker para evitar errores.
6. Construye las imágenes, las sube a Docker Hub y aplica los archivos 'yml' al clúster. Después, proporciona información de los servicios, despliegues y pods, y finalmente, la dirección pública del servicio 'productpage'.

Estoy casi seguro de que la opción de Docker Hub para despliegues más grandes es adecuada; generalmente, el uso de repositorios es recomendable. Sin embargo, en este caso, hemos optado por esta alternativa debido a la incapacidad de solucionar los errores asociados con el uso de imágenes locales.

Este script debería funcionar sin problemas en Cloud Shell si se cambia mi código de proyecto 'clear-column-411518' por uno propio y se crea el archivo 'secret.py'.

Creemos que el enfoque es bastante bueno, ya que la automatización es completa y hemos intentado programar lo mínimo posible, con el fin de delegar la mayor parte en los archivos 'dockerfile' y los 'yml'.

Discusiones

En nuestra opinión la forma monolítica sin Docker se consideraría como un punto de referencia para pruebas antes de subir algo a producción, pero no más que eso. El flujo ideal sería probar la aplicación en local y, posteriormente, utilizar Docker. En caso de manejar varios servicios, los ejecutaría primero en local y luego en Docker, y si tuviera pocos servicios y no fuera necesario escalar, usaría Docker Compose. Solo en el último caso, cuando necesitara escalabilidad, redundancia o control sobre múltiples servicios, recurriría a Kubernetes. También es necesario tener en cuenta que Docker puede ralentizar significativamente los ordenadores, por ejemplo los ordenadores con SO Windows con WSL funcionan especialmente mal. Además de la escalabilidad y el tipo de servicio, debemos considerar los requisitos de nuestros sistemas.

Concluyendo, para un sistema sencillo no veo problema en usar un Docker básico sin necesidad de Docker Compose. Pero para despliegues más grandes o más complejos, me adentraría en Docker Compose e incluso en Kubernetes. Todo es un balance entre escalabilidad, redundancia, control y las prestaciones de los sistemas.