Administración de Sistemas - Curso 2023 / 2024

Laboratorio 8.- CI/CD

Contenido:

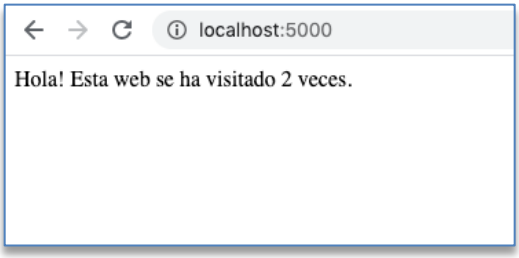
Objetivos: Configurar un pipeline de integración continua/despliegue continuo para desplegar cambios en código en un entorno Kubernetes.

Administración de Sistemas - Curso 2023 / 2024

1 Introducción

El objetivo principal de este laboratorio es poner en práctica las técnicas de CI/CD vistas en el apartado magistral utilizando GitHub Actions como herramienta. Además, se utilizarán otras herramientas necesarias para el empaquetado y ejecución de software como Docker y Kubernetes.

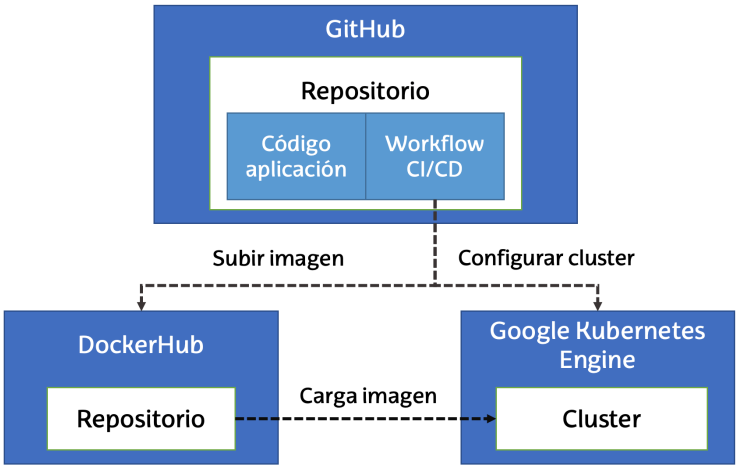
La aplicación a utilizar como "caso de uso" será la que se presentó en el Laboratorio 6 de la asignatura. Esta aplicación está formada por 2 imágenes Docker, una se usa para mostrar un front-end web (creado con Flask y Python) y otra para almacenar un contador de visitas en una BBDD Redis. La interfaz web es la siguiente:



En este laboratorio, el código fuente de la aplicación residirá en un repositorio de GitHub, junto con la configuración necesaria para su ejecución, y la aplicación de técnicas CI/CD permitirá que, cada nuevo cambio se pruebe y se ponga en marcha en un cluster Kubernetes de forma automática.

2 Configuración inicial del workflow

Las herramientas a utilizar en este laboratorio se muestran de forma esquemática en el siguiente diagrama, junto con la interacción entre ellas:



Cada herramienta tiene el siguiente cometido:

- *Repositorio de código en GitHub*: gestionar el código fuente de la aplicación y los workflows de CI/CD.
- *Repositorio de imágenes en DockerHub*: gestionar la imagen Docker que muestra el front-end Web
- *Google Kubernetes Engine*: ejecutar los contenedores de la aplicación.

Administración de Sistemas - Curso 2023 / 2024

El primer paso del laboratorio es obtener los ficheros necesarios para hacer funcionar la aplicación, desarrollados en el primer apartado del Laboratorio6:

- Fichero app.py con la aplicación Flask.
- Fichero requirements.txt que describe los requisitos Python de instalación.
- Fichero Dockerfile para construir la imagen Docker "web".

En este punto, se recomienda hacer uso del fichero Docker Compose para verificar el funcionamiento local de la aplicación, aunque este no será necesario para el desarrollo del laboratorio.

Después, crear un repositorio de código en GitHub con la siguiente estructura.

- Carpeta app: Contendrá los 3 ficheros descritos en el paso anterior.
- Carpeta k8s: Contendrá los ficheros de configuración YAML de Kubernetes.
- Fichero Readme: Contendrá información sobre el repositorio.

Con el repositorio listo, subir los ficheros de la aplicación (desarrollados como parte del Laboratorio 6) a la carpeta *app*. Se recomienda sincronizar la herramienta "git" de vuestros equipos (o el IDE que utilicéis) con vuestra cuenta GitHub para poder trabajar con ese repositorio de forma local.

A continuación, crear un repositorio en DockerHub llamado as-laboratorio-8-web, que contendrá la imagen con el front-end web de la aplicación. El repositorio debe ser público. Una vez creado, construir la imagen "web" de la aplicación y subirla al repositorio.

Después, crear un cluster de Kubernetes Engine en Google Cloud. La única restricción es que debe ser público, la elección de nombre y región queda a vuestro criterio. Cuando esté listo, se deben crear los siguientes objetos para ejecutar la aplicación en Kubernetes:

- Dos objetos Deployment (ambos con 1 réplica por Pod):
  - Un Deployment para el front-web, que debe usar la imagen del repositorio as-laboratorio-8-web.
  - Un Deployment para la BBDD, que debe usar la imagen redis:alpine.
- Un objeto ClusterIP para la BBDD. El objeto ClusterIP debe llamarse "redis".
- Un objeto LoadBalancer para mostrar la web al exterior.

Los ficheros YAML deben guardarse en la carpeta *k8s* del repositorio GitHub. En este punto, se recomienda probar manualmente que los objetos funcionan en el cluster Kubernetes. Si no es posible acceder a la web desde una IP pública proporcionada por el LoadBalancer, revisar las configuraciones realizadas. Si la aplicación funciona, eliminar todos los objetos del cluster (pero no eliminar el cluster).

A partir de aquí, el objetivo es crear un workflow de GitHub Actions que automatice los últimos pasos realizados (crear una imagen Docker, subirla a un repositorio DockerHub, configurar un cluster Kubernetes que ejecute la aplicación). Para esto hay que realizar tres pasos previos.

El primer paso es crear una cuenta de servicio en Google Cloud. El workflow de GitHub Actions necesita autenticarse en Google Cloud para poder configurar el cluster Kubernetes y, para ello, usaremos una cuenta de servicio: un tipo de cuenta con permisos limitados que se usa en estos casos. En la consola de Google Cloud, abrir la sección "IAM y administración" y ahí "Cuentas de servicio". Crear una nueva cuenta de servicio con permisos, al menos, de administración de clusters Kubernetes. Cuando esté creada, seleccionar la cuenta, ir a la pestaña "Claves", crear una nueva y obtener el JSON. Este JSON contiene las credenciales necesarias para gestionar la autenticación desde fuera de Google Cloud.

El segundo paso es realizar algo análogo para Docker Hub, ya que también será necesario que el workflow pueda publicar nuevas versiones de la imagen de forma automática. Para esto, usaremos Tokens de DockerHub: contraseñas temporales a las que asignar tiempo de vida y permisos limitados. En vuestro perfil DockerHub, abrir la configuración de vuestra cuenta y ahí el apartado "Seguridad". Crear un nuevo token de acceso con permisos de lectura y escritura.

El último paso es hacer que estas credenciales estén disponibles para el workflow de forma segura. Para esto, se usarán los secretos de los repositorios GitHub. Abrir el apartado "Configuración" del repositorio y ahí el apartado de secretos. Crear secretos para los siguientes elementos (usad los nombres que veáis apropiados):

- Nombre de usuario de DockerHub.
- Token de DockerHub.
- Nombre del proyecto Google Cloud en el que se ha creado el cluster Kubernetes.
- Token de Google Cloud (objeto JSON con las credenciales de la cuenta de servicio).

Con todo listo, el siguiente paso es crear el workflow de Actions en el repositorio GitHub. El workflow debe dispararse sólo cuando suceda un evento *Push* en la rama *main* y realizar, al menos, las siguientes tareas:

1) Descargar el código del repositorio.
2) Autenticarse en Docker Hub, construir una imagen con la última versión del código y subirla al repositorio ‹as-laboratorio8-web›.
3) Autenticarse en Google Cloud. Utilizar la acción "google-github-actions/auth@v1". Se deben establecer sus parámetros "credentials_json" y "project_id" para indicar las credenciales de acceso e ID de proyecto respectivamente (usar los secretos del repositorio).
4) Configurar la autenticación de Kubernetes Engine para poder acceder desde Actions. Utilizar la acción "simenandre/setup-gke-gcloud-auth-plugin@v1". No requiere configurar ningún parámetro.
5) Configurar la herramienta kubectl para poder usarla desde Actions. Utilizar la acción "google-github-actions/get-gke-credentials@v1". Se deben establecer sus parámetros "cluster_name" y "location" para indicar el nombre del cluster Kubernetes y la región en la que está creado, respectivamente.
6) Desplegar los objetos Kubernetes en el cluster. Utilizar "kubectl" con los mismos parámetros que se usarían en la línea de comando.

Se recomienda desarrollar el workflow de forma incremental, probando que el resultado de cada paso es correcto antes de escribir el siguiente. Además, en el paso relacionado con desplegar objetos Kubernetes, posiblemente será necesario utilizar el siguiente comando:

```
kubectl rollout restart deployment <nombre-deployment>
```

Este comando provoca el reinicio de los Pods en un deployment llamado ‹nombre-deployment›, lo que fuerza a obtener la última versión de la imagen Docker que esté disponible en el registro.

Cuando el workflow esté listo, verificar su funcionamiento: realizar un cambio en el fichero app.py (p.e. modificar el mensaje de texto mostrado). Tras realizar el cambio y subirlo al repositorio GitHub (haciendo *commit* y *push*), el workflow debería activarse, ejecutar los cambios y resultar en una actualización automática de aplicación (verificar desde el navegador).

Cómo último paso de esta sección, se recomienda añadir información sobre la estructura del repositorio, el workflow y las tareas realizadas en el fichero Readme.

3 Nuevo workflow para pruebas

La sección anterior del laboratorio se ha dedicado a crear un workflow que construye un artefacto, lo sube a un registro (en este caso, DockerHub) y lo pone en marcha. Sin embargo, no se han considerado las fases de pruebas de una aplicación, que generalmente van asociadas al desarrollo de nuevas características. En esta sección se propone resolver esta situación para crear un entorno más realista.

Crear un nuevo workflow con un nombre que indique que su objetivo es realizar las pruebas de la aplicación. Este workflow debe dispararse cuando se realice un evento *Push* en cualquier rama que no sea *main* y debe tener, al menos los siguientes pasos:

1) Descargar el código del repositorio.
2) Realizar pruebas del código usando la acción "advanced-security/python-lint-code-scanning-action@v1". Se puede configurar el tipo de pruebas a realizar con el parámetro "linter". Se recomienda utilizar "pylint", aunque se pueden probar otros.

Una vez completado, se debe probar el funcionamiento del workflow: crear una nueva rama en el repositorio llamada "nueva-caracteristica" y utilizar esta rama para realizar algún cambio en el fichero *app.py* (p.e. en el mensaje mostrado en la web). Confirmar el cambio en el repositorio (haciendo *commit* y *push*) y verificar que el workflow recién creado se ha activado (y que el workflow creado en el paso anterior no). Comprobar y realizar los cambios sugeridos por la acción de pruebas.

Cuando se hayan realizado los cambios sugeridos por la acción de pruebas y estén confirmados en el repositorio, hacer una Pull Request para incluir los cambios de la rama "nueva-caracteristica" en la rama *main*. Si se ha hecho correctamente, el workflow creado en el paso anterior se debería activar y hacer que los cambios hechos en el código se muestren en la aplicación en marcha de forma automática.

Como última tarea, tras verificar que los workflow creados funcionan correctamente, se recomienda eliminar el cluster Kubernetes creado en Google Cloud (si no se va a hacer más uso del mismo) para evitar costes innecesarios.

4 Bibliografía

Para el desarrollo de este laboratorio se ha utilizado el siguiente material (consultados en noviembre 2023):

- "Connecting GitHub Actions and Google Cloud Deploy", Google Cloud Blog: https://cloud.google.com/blog/products/devops-sre/using-github-actions-with-google-cloud-deploy
- "Deploying to Google Kubernetes Engine", Documentación official GitHub Actions: https://docs.github.com/en/github-ae@latest/actions/deployment/deploying-to-your-cloud-provider/deploying-to-google-kubernetes-engine
- "Set up a CI/CD pipeline using GitHub Actions to a GKE cluster", Dev.to: https://dev.to/adesoji1/set-up-a-cicd-pipeline-using-github-actions-to-a-gke-cluster-4fj