AMPLIACIÓN DE ING.SOFTWARE

2020-2021

PRÁCTICA 3:

INTEGRACIÓN, ENTREGA Y DESPLIEGUE CONTINUO

Universidad
Rey Juan Carlos

Antonio Adrián Bonilla y Fernando Rubio Moreno.

Titulación: Grado en Ingeniería Informática

ÍNDICE

1. Introducción

Se desean implementar ciertos controles de calidad de una aplicación que gestiona una librería online. Esta librería ofrece un interfaz web y una API REST para la gestión de libros, con sus correspondientes pruebas unitarias, de API REST y Selenium.

El control de calidad se realizará mediante la implantación del modelo de desarrollo git flow para el proyecto, creando las ramas necesarias, la definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación. Estos workflows funcionarán correctamente antes de pasar al paso de desarrollo de una funcionalidad nueva

2. URL del repositorio GitHub utilizado

https://github.com/Fernando-Rubio/a.adrian.2018-f.rubio.2018

3. URL de la aplicación desplegada en Heroku

https://aadrian2018-frubio2018.herokuapp.com/

4. Funcionamiento de los workflows implementados

Este workflow se ejecuta siempre a una determinada hora (4:05), y está formado por un total de 5 jobs, el job de build, en el que se guarda el contenido de la carpeta target para joder ser reutilizado por otro job.
El de test_unitario, test_selenium y el de test_rest, los cuáles llevan implementados varios steps en los que clonamos el repositorio, instalamos la versión de java en el runner y ejecutamos el correspondiente test.
Y por último el job de docker, que lleva implementados varios steps en los que clonamos el repositorio, descargamos el jar del anterior job, nos logeamos en DockerHub, construimos la imagen, y la subimos a DockerHub

Este workflow se ejecuta cada vez que se termine una feature y antes de integrarse en la rama develop, y está formado por un total de 2 jobs, el de test_unitario y el de test_rest, los cuáles llevan implementados varios steps en los que clonamos el repositorio, instalamos la versión de java en el runner y ejecutamos el correspondiente test.

Este workflow se ejecuta cuando hacemos cualquier tipo de commit en la rama develop, y está formado por un total de 3 jobs, el de test_unitario, test_selenium y el de test_rest, los cuáles llevan implementados varios steps en los que clonamos el repositorio, instalamos la versión de java en el runner y ejecutamos el correspondiente test.

Este workflow se ejecuta cuando hacemos cualquier tipo de commit en la rama release, y está formado por un total de 2 jobs, el de test_unitario y el de test_rest, los cuáles llevan implementados varios steps en los que clonamos el repositorio, instalamos la versión de java en el runner y ejecutamos el correspondiente test.

Este workflow se ejecuta cuando queremos desplegar en Heroku, y está formado por un total de 5 jobs, el de build, que guarda el contenido de la carpeta target para poder ser reutilizado por otro job, el de publish_in_heroku_registry que lleva implementados varios steps en los que a declaramos como variables de entorno el nombre de la app y la API Key de Heroku, clonamos el repositorio para tener disponible el Dockerfile, obtenemos la carpeta target del job anterior y se construye la imagen Docker utilizando el JAR generado en el step anterior. Después, tenemos el job de deploy_to_heroku, en el que volvemos a declarar los secretos como variables de entorno. Instalamos el cliente de Heroku y desplegamos la app.
Finalmente pasamos los 2 smoke test de REST y Selenium, pasándole la url de la aplicación con $HOST

5. Nueva funcionalidad

Para añadir la nueva funcionalidad, creamos una nueva rama feature/feature-v3 para introducir los 2 ficheros colgados en el aula. Después de añadirlos hicimos un pull request con la rama Develop.

Comprobamos con los Workflows de GitHub Actions que todas las funcionalidades anteriores seguían funcionando.

Finalmente, hicimos una nueva rama release con el comando git flow release start 0.1.0 creando así la nueva rama release 0.1.0. Cambiamos la versión del pom quitando el sufijo -SNAPSHOT y comiteamos dichos cambios.

Con el comando git flow release finish 0.1.0, intentamos mergearlo con la rama master pero nos salta un aviso de que son 2 historias sin relación por lo que no pudimos acabar dicha release.

En cuanto al despliegue, se realiza con el Workflow desplegar_heroku contra la rama release, que despliega la imagen docker con el cliente de heroku y prueba los 2 smoke tests de Selenium y Rest con el HOST como url a nuestra app.

6. Conclusiones

Para llevar a cabo la realización de esta práctica, hemos dedicado aproximadamente un total de 16 horas. Hemos cumplido el objetivo que nos marcamos al comienzo de la misma, implementar ciertos controles de calidad de una aplicación que gestiona una librería online mediante la implantación del modelo de desarrollo git flow para el proyecto, creando las ramas necesarias, la definición de workflows que automaticen la ejecución de pruebas, la publicación de releases y el despliegue en producción de la aplicación.

A su vez, hemos intentado organizarnos lo mejor posible para así no tener que trabajar en exceso los últimos días antes de la entrega y de esta manera lograr un mejor resultado final.

En cuanto a la entrega, hemos intentado subir el zip con la carpeta de la rama master, pero no podíamos hacer un merge entre las ramas master - develop.
Finalmente enviaremos la práctica con la rama develop