Experto Universitario en DevOps & Cloud

Caso Práctico 1
Apartado B (CP1.2)
Guía de apoyo

Índice

Introducción 3

Herramientas requeridas 4

Reto 1 – Pipeline CI 7

Reto 2 – Distribución de agentes 11

Reto 3 – Mejora de cobertura 12

Introducción

Este documento sirve de ayuda para la realización del CP1.2, explicando, con mayor detalle, el enfoque a realizar para finalizar todos los ejercicios. No obstante, su cometido no es ofrecer al alumno una guía paso a paso, sino únicamente dibujar en camino con un grano más fino a como se explica en el enunciado global del CP1.

Se dará información técnica sobre URLs a repositorios, aplicaciones, algún comando más complejo, pero, especialmente, consejos sobre cómo afrontar cada uno de los retos expuestos en el enunciado.

En este CP1.2, contamos con 3 retos:

- Puesta en marcha del pipeline CI visto en los ejercicios de clase: 60 %.
- Distribución del pipeline en 3 agentes: 25 %.
- Cobertura completa del código fuente tanto por líneas como por ramas: 15 %.

Es importante recordar que la calificación de cualquiera de los retos del CP1 se realiza en base a tres criterios:

- Funcionamiento correcto de la solución.
- Calidad técnica de la implementación.
- Explicación de cómo se ha llevado a cabo la implementación, demostrando conocimiento suficiente de la materia.

Herramientas requeridas

Las herramientas que se requieren para este apartado son:

- JDK (sirve cualquier versión "viva"; por ejemplo, Java 8):
  - Dentro de Java se ejecutarán aplicaciones que se comentarán más adelante.
- Python 3 (por ejemplo, 3.11, o la que ofrezca el instalador como última versión estable)
  - Módulo pytest.
  - Módulo flask.
  - Módulo flake8.
  - Módulo bandit.
  - Módulo coverage.
- Jenkins (última versión estable):
  - Si no vienen preinstalados los plugins de JUnit y GitHub, se instalan.
  - Además, es necesario instalar los plugins de cobertura, warnings-ng y performance.
  - Es importante usar la configuración por defecto, con los plugins por defecto, para mantener todos los alumnos la misma versión aproximada.
- Git.
- JMeter (cualquier versión actual estable):
  - Por ejemplo, v5.5.

JDK: entorno de desarrollo Java y máquina virtual. Se puede descargar desde la página oficial de Oracle o bien mediante los instaladores habituales de cada S.O. (p.e. "apt-get")
https://www.oracle.com/java/technologies/downloads/#java8-windows

Python: entorno de ejecución Python. Se puede descargar desde la página oficial de Python, o bien mediante los instaladores habituales de cada S.O. (p.e. "apt-get")

Algunas versiones contienen instaladores, mientras que otras versiones más antiguas pueden requerir descarga, descompresión y configuración manual.
https://www.python.org/downloads/

Módulos de Python: una vez que tenemos Python funcionando, debemos instalar dos módulos necesarios para la práctica: pytest y flask. Lo realizaremos mediante el propio instalador de paquetes de Python: pip

pip3 install pytest

pip3 install flask

pip3 install flake8

pip3 install bandit

pip3 install coverage

Jenkins: se instala fácilmente siguiendo todos los pasos indicados en el instalador.
La única peculiaridad de este producto es que la contraseña de administración inicial se genera automáticamente en un fichero de texto (el propio instalador nos avisa) y debemos leer ese fichero en la ruta que nos indican (cada S.O. es diferente). Cuando arrancamos Jenkins y accedemos por primera vez mediante el navegador (http://localhost:8080, por defecto), nos pedirá esa contraseña, que la pegaremos y "activaremos" Jenkins.
Se trata de una aplicación que funciona bajo Java y está disponible en instaladores y contenedores, y también mediante paquete .war
Se recomienda el instalador normal, pero cada alumno puede hacerlo funcionar como mejor le convenga.

Git: git también se instala fácilmente haciendo uso de los instaladores proporcionados en la web oficial, o mediante los sistemas de gestión de paquetes de los sistemas operativos, como apt-get en Linux.

Sistema operativo: se recomienda trabajar con Linux o Windows, para facilitar algunos apartados de la práctica, y no se ha comprobado el correcto funcionamiento en Mac. En clase, lo veremos casi todo en Windows. Únicamente los agentes de

Jenkins correrán en Linux, por ser el S.O. del servidor empleado para los ejercicios que vemos en clase.

JMeter: la aplicación JMeter solo requiere descarga del paquete comprimido y extraerlo en una carpeta. Es una aplicación Java que se ejecuta mediante un script incluido en el paquete en la carpeta "bin" ("jmeter.bat" o "jmeter.sh").
JMeter también puede instalarse en modo servidor para distribuir las pruebas, pero esto queda fuera del alcance de esta práctica, puesto que para hacer esta distribución también podríamos contar con la técnica de distribución de tareas propia de Jenkins (agentes).

Reto 1 – Pipeline CI

Al igual que en el CP1.1, este reto consiste en replicar exactamente el ejercicio que hemos visto en clase. Para ello, debemos eliminar la etapa “Build” y modificar/añadir las etapas necesarias para ejecutar las pruebas que queremos incorporar en el pipeline de CI, que contendría las siguientes etapas:

- Get Code: obtener el código fuente desde el repositorio:
  - Se recuerda al alumno que, cuando Jenkins obtiene la definición del pipeline desde un Jenkinsfile alojado en un SCM, aparecerá en Jenkins automáticamente una etapa preliminar para la obtención del pipeline. No hay que hacer nada con esa etapa automática.
- Unit: ejecución de pruebas unitarias de manera similar a como se hizo en el CP1.1.
- Rest: ejecución de pruebas de integración tal y como se hizo anteriormente.
- Static: pruebas de análisis de código estático, usando flake8.
- Security Test: pruebas de seguridad mediante análisis de código estático, usando bandit.
- Performance: pruebas de carga, usando JMeter.
- Coverage: pruebas de cobertura de código, mediante coverage.

De nuevo se recuerda al alumno que esta guía no pretende ser completa ni ofrecer una guía paso a paso, sino únicamente orientar al alumno sobre la mejor manera de enfocar la práctica.

Es labor de alumno resolver los problemas que puedan planteársele en cada caso particular.

A continuación, se detallan las tareas que deben llevarse a cabo en este primer reto:

- Partiremos del mismo pipeline del CP1.1.

- Eliminar la etapa “Build” que se usó a efectos de prueba, pero que no tiene ninguna función.
- Eliminar la etapa “Results”, puesto que cada tipo de test incluirá tanto la ejecución de la prueba, como la llamada al plugin de Jenkins para publicar los resultados.
- Dar de alta etapas para todas las pruebas enumeradas en la página anterior.
  - Es posible que haya que modificar alguna de las etapas que ya existían en el CP1.1, para optimizar la ejecución de algunas de las pruebas nuevas que vamos a realizar.

Como particularidad a cada una de estas pruebas, se especifican los siguientes baremos de aceptación de los resultados obtenidos por las herramientas de testing empleadas:

- Pruebas unitarias:
  - No se tendrá en cuenta ningún baremo para establecer si las pruebas han sido satisfactorias o no por lo que esta etapa siempre se mostrará en verde tras la ejecución de las pruebas.
  - Las pruebas unitarias solo pueden ejecutarse una vez; no se permite que se ejecuten las mismas pruebas varias veces en el mismo pipeline.
- Pruebas de integración (Rest):
  - Esta etapa es la misma que en el CP1.1.
  - En caso de no conseguir hacerla funcionar, como en el CP1.1, será un fallo leve, pero no impide la ejecución del resto de retos.
  - Tampoco se tendrá en cuenta ningún baremo para determinar la salud de la aplicación; siempre finalizará en verde, aunque el resultado de las pruebas unitarias no sea óptimo.
- Pruebas de análisis de código estático (Flake8):
  - Si se encuentran 8 o más hallazgos, debe marcarse la etapa y el build como *unstable* (amarillo/naranja).
  - Si se encuentran 10 o más hallazgos, debe marcarse la etapa y el build como *unhealthy* (rojo).

- Sea cual sea el resultado, el pipeline debe continuar la ejecución del resto de etapas.

- Pruebas de seguridad (Bandit):
  - Si se encuentran 2 o más hallazgos, debe marcarse la etapa y el build como *unstable* (amarillo/naranja).
  - Si se encuentran 4 o más hallazgos, debe marcarse la etapa y el build como *unhealthy* (rojo).
  - Sea cual sea el resultado, el pipeline debe continuar la ejecución del resto de etapas.
- Pruebas de cobertura (Coverage):
  - Si la cobertura por líneas se encuentra entre 85 y 95, se marcará como *unstable*. Por encima será verde y, por debajo, rojo.
  - Si la cobertura por ramas/condiciones se encuentra entre 80 y 90, se marcará como *unstable*. Por encima será verde y, por debajo, rojo.
  - Sea cual sea el resultado, el pipeline debe continuar la ejecución al resto de etapas.
  - Se recuerda que las pruebas unitarias solo pueden ejecutase una vez durante todo el pipeline.
- Pruebas de rendimiento (JMeter):
  - La prueba debe incluir un test-plan en el que 5 hilos realicen 40 llamadas al microservicio de suma, y otras 40 al microservicio de resta.
  - Para poder llevar a cabo esta tarea, flask debe estar levantado y escuchando peticiones. Wiremock no es necesario puesto que solo vamos a usar los servicios de suma y resta.
  - En caso de no conseguir levantar flask dentro de la etapa, existe la opción de mantener flask levantado fuera de la ejecución de Jenkins (en una ventana de comandos aparte), para que jmeter pueda conectar con los microservicios. Lógicamente, esta solución tendrá una puntuación más baja.
  - El test-plan debe ejecutarse en la etapa Performance y llamar al plugin para visualizar los resultados.

- Entre los entregables de este reto, para las pruebas de cobertura, debe indicarse cuál sería el valor de “línea 90” para el microservicio de suma, así como captura de la gráfica en la que puede apreciarse este valor.

Reto 2 – Distribución de agentes

Al igual que en el CP1.1, en este reto se pretende paralelizar y distribuir las tareas todo lo posible.

La explicación sobre cómo se dan de alta agentes, se distribuye la carga y se comparten datos entre etapas, se puede revisar en la guía de apoyo del CP1.1.

Las tareas a realizar en este reto son:

- Elegir qué separación de agentes sería la más adecuada en este pipeline y justificar por qué.
- Paralelizar todas las pruebas, siempre que sea posible.
  - Si alguna prueba no se puede paralelizar, explicar por qué y extraerla de la ejecución paralela (y dejarla como secuencial) o proponer una solución mejor.
- Actualizar el Jenkinsfile usado en el mismo reto del CP1.1 (“JENKINSFILE_agentes”) para que asigne la ejecución de cada etapa al agente elegido, teniendo en cuenta las particularidades del intercambio de archivos entre nodos, y las buenas prácticas sobre limpieza del workspace una vez finalizada la ejecución.
- Incluir en cada etapa los comandos whoami, hostname y echo ${WORKSPACE}
- Cada agente tiene un número de “executors” o peticiones en paralelo que puede admitir ese agente (es un dato que se define en base a la capacidad de procesamiento del agente). Por defecto, pondremos ese valor a un número entre 2 y 4 y realizar la ejecución del pipeline y obtención del log.
- Posteriormente, debemos forzar ese valor a 1, para que el agente solo pueda atender 1 petición cada vez. Explicar qué ocurre en este caso, y adjuntar el log de ejecución donde se justifique este nuevo comportamiento.

Reto 3 – Mejora de cobertura

Actualmente, el código de esta aplicación tiene vinculadas unas pruebas unitarias que alcanzan un grado de cobertura del 97% en líneas y del 83% en ramas, para el código de la carpeta “app” (omitiendo el fichero _init_ y el api.py), tal y como se ha visto en clase.

El objetivo de este reto es ampliar la cobertura al 100% en ambos tipos. Para ello, solo deberemos cambiar el código de la carpeta “test\unit”, y en ningún caso se deberá tocar el código de la carpeta “app”, lo cual incluye también la prohibición de añadir “pragma: no cover” en los ficheros de la carpeta “app”.

Las tareas a realizar en este reto serán:

- Generar una rama nueva “feature_fix_coverage” a partir de la rama master.
- En esta nueva rama, modificar los test para lograr una cobertura completa.
- Explicar cuál era el problema y cuál ha sido la solución.
- Adjuntar log de ejecución donde se muestre que la cobertura es del 100 % y gráficas de evolución (con el plugin cobertura) donde se aprecie también este aumento de cobertura.