Experto Universitario en DevOps & Cloud

Caso Práctico 1
Apartado A (CP1.1)
Guía de apoyo

Índice

Introducción

Este documento sirve de ayuda para la realización del CP1.1, 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 CP1.1, contamos con 4 retos:

- Puesta en marcha del pipeline CI visto en los ejercicios de clase: 40 %
- Distribución del pipeline en 3 agentes: 30 %.
- Ampliación de los microservicios: 15 %.
- Arreglo de un posible error en el pipeline: 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 (p.e. 3.11, o la que ofrezca el instalador como última versión estable):
  - Módulo pytest (si no viene preinstalado).
  - Módulo flask (si no viene preinstalado).
- Jenkins (última versión estable):
  - Si no vienen preinstalados los plugins de JUnit y GitHub, se instalan.
  - Es importante usar la configuración por defecto, con los plugins por defecto, para mantener todos los alumnos la misma versión aproximada.
- Git.

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

Wiremock: usaremos Wiremock como contenedor de microservicios mock, para simular una de las llamadas de la práctica. Se trata de un simple fichero .jar y unos ficheros de configuración que se proporcionan en esta guía.
https://wiremock.org/docs/download-and-installation/
(se recomienda descargar la versión JAR, pero si alguien quiere iniciarlo como contenedor Docker, también es viable, aunque esta guía se basará en la versión JAR)

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
```

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.

Reto 1 – Pipeline CI

Este reto es a su vez el más laborioso y también el más sencillo, ya que básicamente consiste en replicar exactamente el ejercicio que hemos visto en clase, con la única dificultad de que algunas aplicaciones ya se encuentran totalmente instaladas en el PC del profesor, mientras que será labor del alumno la instalación de todas estas herramientas que se detallan en este apartado y que son necesarias para poder realizar tanto este CP1.1, como el CP1.2.

También puede ofrecer algún desafío al alumno la creación de cuentas de usuario SSH en los PC empleados.

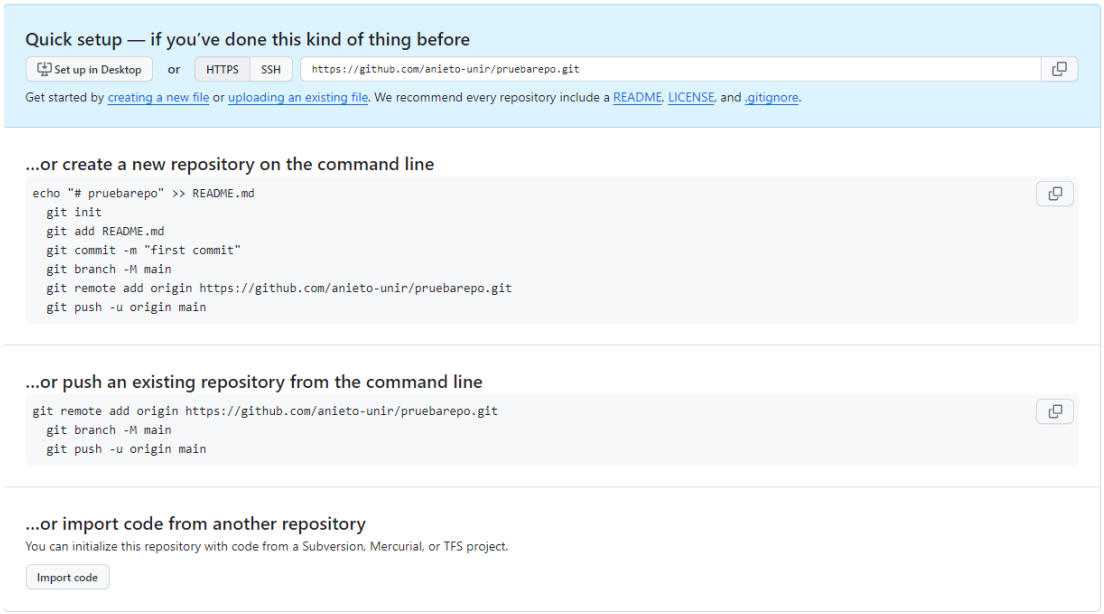
De nuevo se recuerda al alumno que esta guía no pretende ser completa ni ofrecer una guía paso a paso, sino únicamente orientar al alumno sobre la mejor manera de enfocar la práctica.

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:

- Creación de una cuenta en GitHub: esta tarea no ofrece mayor dificultad que darse de alta en la web de github.com (quien no tenga ya una cuenta, o prefiera emplear una cuenta específica para estas actividades).
- Descarga del código fuente original de la práctica, y replicado en un repositorio específico del alumno en su cuenta de GitHub.
  - Este repositorio original se encuentra en https://github.com/anieto-unir/helloworld.git
  - Una vez clonado el código (git clone), y creado el nuevo repositorio en GitHub, el propio GitHub indica los comandos que deben usarse para llevar este código a un repositorio particular.

- Son los comandos que se muestran en la siguiente imagen. Ofrece varias opciones.



  - Lógicamente, cambiando las URLs.
  - También puede hacerse un fork, pero queda más limpio descargarlo y subirlo a otro repositorio.
  - Recordad usar la rama "master" y no "main" para que todos empleemos la misma nomenclatura de rama principal.
  - ¡¡Nunca usar el repositorio original del profesor para trabajar!!
- Ejecutar en línea de comandos todas las pruebas unitarias y de integración (con flask y wiremock), tal y como vimos en clase.
  - Esta tarea no es estrictamente necesaria para la práctica, pero nos sirve para verificar que todo funciona correctamente, antes de entrar a configurar el pipeline en Jenkins.
- Realizar las tareas indicadas en la presentación de clase como "Jenkins 1".
- Realizar las tareas indicadas en la presentación de clase como "Jenkins 2".
- De los ejercicios "Jenkins 3", solo es necesario el primer punto.
  - Crear un pipeline donde se use un Jenkinsfile de vuestro repositorio en GitHub.
  - Quien lo desee, puede probar también a activar el polling para detectar cambios en el repositorio que disparen la ejecución de pipeline, pero es

importante no abusar de la frecuencia de polling porque podemos bloquear temporalmente nuestra cuenta. Igualmente, una vez probado, hay que desactivar esta opción de polling.

- El mapping de Wiremock empleado para estos ejercicios, se encuentra disponible en la misma presentación de clase, y se adjunta también a continuación:
  - Se recomienda grabar este fichero de texto en la misma ruta donde se encuentra el .jar de Wiremock, creando una subcarpeta por debajo "mappings". El fichero puede llamarse sqrt64.json

```
{
    "request": {
        "method": "GET",
        "url": "/calc/sqrt/64"
    },
    "response": {
        "status": 200,
        "body": "8",
        "headers": {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
        }
    }
}
```

Reto 2 – Distribución de agentes

Este reto es un poco más complejo ya que requiere buscar información sobre cómo conseguir acceso a un servidor SSH con diversas cuentas (aunque en clase veremos también otro mecanismo más sencillo). Por lo demás, siguiendo los pasos vistos en los ejercicios de clase, no debería ofrecer mayores problemas.

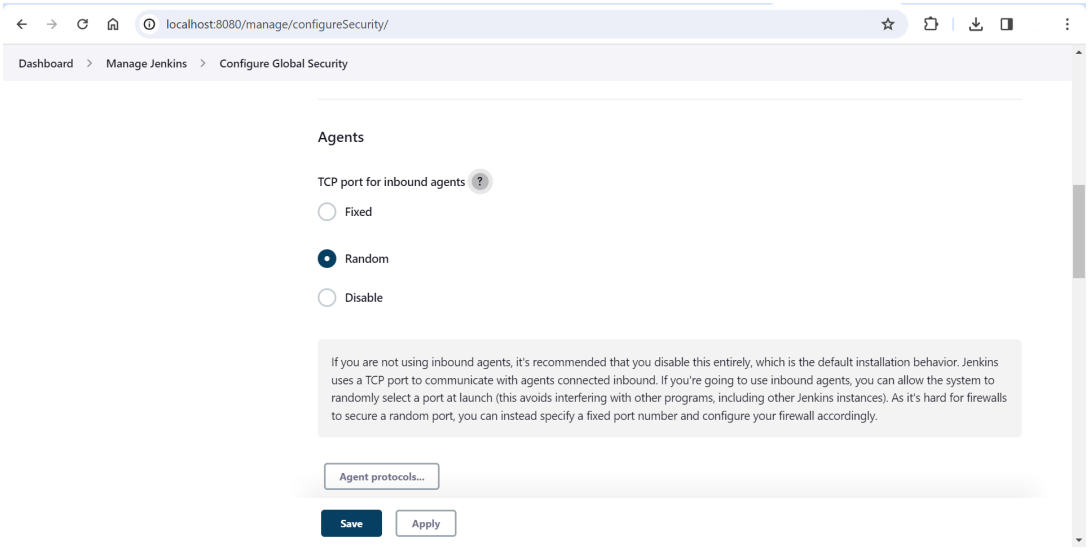
Las dos vías principales para poner en marcha agentes son:

- Mediante conexión SSH desde el master al agente(slave).
- Mediante línea de comandos en el agente(slave), que debe tener conexión TCP hacia el master (apertura de puertos en Firewalls, si no es localhost), y habilitar esta conexión en la configuración de Jenkins.

La opción más rápida y sencilla para dar de alta los agentes es lanzarnos desde línea de comandos ejecutando el agent.jar de Jenkins. En este caso, los pasos son:

- Lo primero de todo, es necesario activar la opción que permite que Jenkins reciba conexiones TCP de los agentes. Esto lo realizamos desde la sección de configuración de Jenkins (Manage Jenkins) en la sección de "Configure Global Security", tal y como se muestra en la captura de pantalla de la siguiente página.
- Dar de alta el agente en Jenkins seleccionando la opción "Launch method: Launch agent by connecting it to the controller" (con la UI en inglés; en otros idiomas el mensaje será equivalente)
- Acceder al agente recién creado y seguir las instrucciones
- Descargar el binario del agente (agent.jar) accediendo al enlace que nos aparece en pantalla
- Ejecutar el agente, desde línea de comandos, en la carpeta raíz del agente (que hemos seleccionado al dar de alta el agente), usando el comando que nos aparece en pantalla. Hay dos opciones (con el secret-code en texto abierto, o bien en un fichero); ambas son válidas para el propósito de esta práctica.

- Verificar que en Jenkins se muestra en nuevo agente como conectado (*online*) y que podemos lanzarle tareas en un pipeline que hagamos de ejemplo.



La segunda opción, y la más usada, es la conexión SSH entre master y agent(slave). Este procedimiento requiere de un servidor SSH (se recomienda un servidor Linux por sencillez); es necesario intercambiar claves SSH entre ambas máquinas, establecer credenciales, etc.

Es más complejo, pero aquellos alumnos que estén familiarizados con entornos Linux y con sistemas SSH, es sin duda la mejor opción, y la que se usa en entornos reales.

Todos los agentes serán permanentes, y no usaremos funciones de *swarming* en esta práctica, para simplificar el ejercicio.

En caso de dudas o fallos en estos procedimientos, se recomienda seguir los numerosos vídeos que hay sobre estos temas en redes habituales (YouTube, etc.)

El segundo hándicap que nos encontraremos en esta práctica es el intercambio de ficheros entre agentes ya que, tal y como se explicó en clase, cada nodo de ejecución ofrece un espacio de trabajo (workspace) y este workspace no se comparte entre

diferentes etapas, por lo que debemos proporcionar un mecanismo que permita compartir datos entre etapas.

En el primer reto, todo el pipeline se ejecutaba en un único nodo (el nodo principal, o master), por lo que este problema no se daba y esto nos permitía descargar el código fuente en una etapa, este código se quedaba en el workspace, y la siguiente etapa se ejecutaba en el mismo workspace que le asignaba Jenkins.

Pero esto no ocurre en la vida real, sino que cada etapa puede ejecutarse en un entorno diferente.

Para compartir datos (ficheros) entre diferentes nodos de ejecución, usamos los comandos de Jenkins *stash* y *unstash*:

- stash almacena, en un espacio de trabajo global del pipeline, una serie de ficheros. Este comando recibe como parámetros los ficheros que queremos almacenar y un nombre que se le asigna a este paquete de ficheros que queremos almacenar.
  Por ejemplo: stash includes: '/config/**', name: 'config'
  Este comando, guardará todos los ficheros y subcarpetas desde la carpeta config y, a este paquete, lo llamará "config".
- unstash accede al espacio global de trabajo del pipeline, y recupera los archivos que se le indiquen en el nombre del paquete de ficheros.

De esta manera, podemos generar un archivo con el informe de pruebas unitarias en un nodo, y recuperar ese archivo en otra etapa que se ejecuta en otro nodo distinto, o bien descargar un proyecto desde GitHub, y compartir los ficheros del proyecto (todos, o una parte) en otros nodos.

Se puede obtener más información en la documentación de Jenkins, así como en las redes habituales (YouTube, stackoverflow, etc.)
https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#stash-stash-some-files-to-be-used-later-in-the-build

Es también una buena práctica limpiar el workspace una vez ha finalizado la ejecución de la etapa. Para ello, podemos hacer una limpieza manual (comandos de sistema operativo) o bien usar las funciones que Jenkins ofrece por defecto (y también mediante plugins), como deleteDir o clearWs, entre otros.

El alumno debe decidir cuál sería la mejor opción (no hay grandes diferencias).

Hay recursos en la red al respecto como, por ejemplo, este: https://stackoverflow.com/questions/54019121/what-is-the-difference-between-deletedir-cleanws-and-wscleanup-in-jenkins-pi

Una vez aclarados los principales problemas que nos podemos encontrar en este reto, realizamos la descomposición de tareas a realizar:

- Elegir qué separación de agentes sería la más adecuada en este pipeline y justificar por qué.
- Modificar el pipeline original (de hecho, dar de alta otro pipeline) 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}

Reto 3 – Ampliar microservicios

Este reto tiene un objetivo doble. En primer lugar, entender cómo funciona el código fuente con el que estamos trabajando y, por otro, poner en práctica uno de los mecanismos más habituales en desarrollo (copiar y pegar).

Denostado por algunos autores, sigue siendo el mecanismo más habitual en desarrollo, partiendo de un caso, escenario o esqueleto similar y adaptándolo a nuestras necesidades. Hoy en día incluso contamos con sistemas tipo copilot, que se basan en esta idea para generar código semi-automáticamente.

Actualmente contamos con dos microservicios en nuestra aplicación de Calculadora, que permiten sumar y restar dos números, exponiendo una API REST sencilla y directa (método GET) que nos permite obtener resultados accediendo a las URLs mediante navegador o comandos tipo *curl*.

En este reto ampliaremos la librería de microservicios con dos nuevos servicios más (multiplicación y división). Para ello, tendremos que ampliar tanto el código fuente de la aplicación (app\api.py) como las pruebas rest sobre esta API, que deben probar todos los escenarios diferentes posibles (test\rest\api_test.py).

Las tareas a realizar en este reto serán:

- Generar una rama nueva “develop” a partir de la rama master
- En esta nueva rama, desarrollar todos los cambios indicados
  - Generar un microservicio “multiply” que reciba dos parámetros
  - Generar un microservicio “divide” que reciba dos parámetros
    - Si el divisor es cero, el servicio debe devolver un error HTTP 406
  - Añadir tests de integración para cubrir los dos nuevos servicios y sus posibles escenarios excepciones.
- Si fuera necesario, modificar el Jenkinsfile

Reto 4 – Mejorar fiabilidad

Como se adelantó en el enunciado global del CP1, el pipeline visto en clase y sobre el que estamos trabajando, contiene un posible error que solo sucede en determinadas situaciones, cuando el PC/servidor se encuentra saturado o no es muy potente.

En esta guía de soporte no se ofrecerán muchos detalles al respecto porque es precisamente labor del alumno detectar dónde puede estar ocurriendo este problema y por qué.

No obstante, en el propio enunciado se dan algunas pistas ocultas en la descripción de los entregables.

El objetivo que persigue este reto es concienciar al alumno sobre la importancia de diseñar correctamente un pipeline (aunque se extiende a cualquier otro desarrollo software) y que se deben tener en cuenta todos los factores que pueden afectar a la ejecución de un programa.

Por poner un ejemplo (que no guarda relación con este ejercicio), si tenemos dos procesos en paralelo intentando escribir en el mismo fichero, el sistema fallará, puesto que no se permite escribir a la vez en un fichero. En este caso, habría que secuencializar los procesos, o bien detectar cuando el primer proceso que toma control del fichero ha finalizado de realizar cambios, mediante mecanismos de exclusión mutua (mutex) por ejemplo.

En este pipeline, en la etapa de “Test Rest”, estamos lanzando tareas en paralelo, por lo que se da una situación posiblemente anómala, que es la que debemos reproducir, detectar y solucionar.

Hay muchas soluciones a este problema, más o menos elaboradas; no se persigue conseguir la mejor implementación de todas, pero sí al menos poder explicar cuál sería la mejor.

Esto quiere decir que se permite explicar cuál sería la mejor solución, aunque se implemente una solución más sencilla para no invertir tanto tiempo.
En todo caso, se puntuará mejor las soluciones mejor planteadas y también mejor implementadas.