THREEPOINTS
THE SCHOOL FOR DIGITAL BUSINESS

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Master en FullStack Developer

CI/CD y DevOps
Practica 3 – Creación de una librería compartida de Jenkins.

MIA 2011

Objetivo

El objetivo de esta práctica es crear una “Shared Lib” de Jenkins propia, y usarla en un pipeline. Como conocimientos a demostrar será:

- Creación de la estructura de una librería básica de Jenkins.
- Uso de una librería en una pipeline declarativo.

Durante estos ejercicios se reutilizará la máquina virtual presentada en la primera práctica, recordemos cuales eran los servicios montados:

- Gitlab: http://localhost (o http://localhost:8000 si se ha cambiado el puerto)
  - Usuario: threepoints
  - Password: threepoints
- Jenkins: http://localhost:8080
  - Usuario:threepoints
  - Password: threepoints
- ~~SonarQube: http://localhost:8090~~
  - ~~Usuario: admin~~
  - ~~Password: threepoints~~
  - ~~(Está apagada por defecto, hay que seguir las instrucciones del tema 3 para desplegarlo).~~

Para entrar a esta máquina virtual, se usarán las credenciales:

- Usuario: threepoints
- Password: threepoints

1. Creación de una librería de Jenkins en Gitlab (1 puntos)

Se debe de crear un proyecto en gitlab, llamado, “*<nombre_alumno>_libreria*”, que contenga una estructura vacía con las 3 carpetas indicadas en el curso (`/var`, `/src/org/threepoints` y `/resources/org/threepoints`), con un archivo vacío llamado “.placeholder” en cada una.

Los entregables de este ejecicio serán:

- Un zip con el proyecto.
- Una captura de pantalla del proyecto de git.

Se recuerda que el servidor Git está expuesto, por lo que se puede acceder por HTTP desde fuera, desde el host, para comodidad de los alumnos durante el desarrollo.

2. Inclusión de la librería en Jenkins de forma explícita (2 puntos).

En este ejercicio, vamos a configurar Jenkins para poder leer la librería compartida que hemos creado en el ejercicio anterior. Para ello, debemos de seguir las instrucciones dadas en clase para poder asociar en el Jenkins de la máquina virtual, nuestra librería, bajo el nombre *`threepoints-sharedlib`*.

Al asociar la librería, se debe de desactivar la opción de "Load implicitly", para permitir cargar la librería posteriormente de la forma que se indican en las diapositivas, implicitamente.

Durante este ejercicio, se debe de entregar una captura de pantalla con la configuración de Jenkins.

3. Programación de la librería para que englobe los dos plugins de SonarQube e inclusión en el pipeline (5 puntos).

~~Durante la práctica 2 se ha debido de usar SonarQube de forma declarativa (si no se ha hecho, se puede encontrar una guía en inglés en el siguiente link: https://medium.com/@rosaniline/setup-sonarqube-with-jenkins-declarative-pipeline-75bccdc9075f).~~ Durante esta práctica, para usar siempre Sonar en nuestros proyectos de forma sencilla, vamos a crear un script en /vars que nos permita usarlo siempre de la misma forma.

~~El script se debe de ubicar en "vars" en nuestra librería, y llamarse "`sonarAnalysis.groovy`", el cual debe de implementar una función "`call`" que permita ejecutar un escaneo de SonarQube, y esperar durante 5m con un timeout al resultado. Debe de poder recibir un parámetro booleano el cual determine tras evaluar el QualityGate de SonarQube si debe de abortar el pipeline o no (por defecto valdrá "`false`", lo cual significa que no debe de abortar el pipeline).~~

Debido a los problemas de SonarQube, se debe de hacer una función que emule el comportamiento esperado. El script se debe de ubicar en "vars" en nuestra librería, y llamarse "sonarAnalysis.groovy", el cual debe de implementar una función "call" que permita emular un escaneo de SonarQube. Para ello, recibirá dos parámetros booleanos:

- El primero llamado quality_gate_result, deberá determinar si pasa o no el "quality gate". Por defecto será "True", osea, que pasaría el Quality Gate.
- El segundo, llamado abort_pipeline, deberá determinar si al fallar el "quality gate" (ósea, cuando el primero es False), debe de arrojar un error el pipeline y cortarlo. Por defecto será "True", ósea, que corta el pipeline si falla.

La función debe de ejecutar la lógica indicada en los parámetros: Si recibe el quality_gate_result como True, deberá mostrar un mensaje con echo diciendo que ha pasado el quality gate con éxito. Si recibe como False ese parámetro, si abort_pipeline vale False, deberá mostrar un mensaje diciendo que no ha pasado el pipeline, pero no abortar la ejecución. En cambio, si vale True, deberá cortar con un error el pipeline diciendo que no ha superado el pipeline.

Para tener la puntuación perfecta de este ejercicio, se debe de mostrar además su utilización en un pipeline declarativo, aconsejando hacerlo con el de la práctica 2.

Como entregable de este ejercicio se debe de dar:

- La librería comprimida en un zip, bajo el nombre `devops_practica3_ejercicio3.zip`.
- Una captura de pantalla de un pipeline en ejecución.
- El código del pipeline que use esta función.
- Capturas de pantalla de la configuración del Job de Jenkins para poder reproducirlo.

4. Configurar el pipeline para ser dependiente de las variables de entorno (2 puntos).

En este último ejercicio, vamos a ampliar la librería del ejercicio 3 para ver cómo usar variables de entorno genéricas de Jenkins. Se debe de configurar la librería para que se ejecute con un webhook (visto en la práctica 1) de Gitlab cada vez que se realice un push.

La función de SonarAnalysis.groovy, anteriormente definida, debe de leer de una variable de entorno (directamente o pasada como parámetro desde el pipeline) y determinar según el nombre de la rama de la que venga, si debe de cortar el pipeline si falla el qualityGate (ósea, el valor del parámetro abort_pipeline anteriormente descrito). Debe de implementar la siguiente lógica:

- Si el argumento pasado en el ejercicio 3 está a True, lo debe de cortar siempre.
- Si no, evaluará el nombre de la rama de git de la que proviene la ejecución y:
    - Si es igual a “master”, debe de cortar el pipeline.
    - Si el nombre de la rama empieza por “hotfix”, debe de cortar el pipeline.
    - Si el nombre vale cualquier otra cosa, no debería cortarlo.

Se pueden encontrar ayuda sobre las variables globales en los apuntes de clase y en los siguientes enlaces:

- Variables de entorno del plugin de Gitlab: https://github.com/jenkinsci/gitlab-plugin#defined-variables
- Guía oficial: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#using-environment-variables

Como entregable de este ejercicio se debe de dar:

- La librería comprimida en un zip, bajo el nombre `devops_practica3_ejercicio4.zip`.
- Una captura de pantalla de un pipeline en ejecución.
- El código del pipeline que use esta función.
- Capturas de pantalla de la configuración del Job de Jenkins para poder reproducirlo.