Experto Universitario en DevOps & Cloud

Caso Práctico 1
Apartado D (CP1.4)
Guía de apoyo

Índice

Introducción

Este documento sirve de ayuda para la realización del CP1.4, explicando, con mayor detalle, el enfoque a realizar para finalizar todos los ejercicios. En este CP1.4, la guía únicamente proporcionará pistas y aclaraciones que serán útiles para el desarrollo de este CP1.4, y no será una guía paso a paso de comandos.

En este CP1.4, contamos con 5 retos:

- Creación del pipeline “CI”: 30 %.
- Creación del pipeline “CD”: 25 %.
- Distribución en 3 agentes: 20 %.
- Separación código/configuración: 15 %.
- Configuración pipeline multibranch: 10 %.

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 CP1.4 son:

- JDK (para ejecutar Jenkins)
  - Cada producto puede requerir una versión diferente, por lo que hay que elegir la que sea necesaria según las instrucciones de instalación de Jenkins.
- Jenkins (última versión estable)
  - Es importante usar la configuración por defecto, con los plugins por defecto, para mantener todos los alumnos la misma versión aproximada.
- Cloud9
  - Usaremos una AMI que ya está preparada para Cloud9, por lo que el proceso será más sencillo.
- Git
  - También disponible en la AMI que usaremos.
- Otros recursos de AWS para el despliegue de las herramientas necesarias para la práctica.

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”).
Usaremos la versión que nos recomiende Jenkins.

Jenkins: se instala fácilmente siguiendo todos los pasos indicados en el instalador.
Para este CP1.4, también usaremos la versión Ubuntu/Debian, por lo que deben seguirse las instrucciones de instalación que el fabricante indica para esta distribución.

Sistema operativo: usaremos Linux Ubuntu en este CP1.4, al igual que en el CP1.3.

Reto 1 – Pipeline “CI”

En este primer reto vamos a poner en práctica los conocimientos y experiencia adquiridos tras los CP1.2 y CP1.3, para configurar un pipeline de Integración Continua, que realice una serie de pruebas (estáticas y dinámicas) y, si todo ha ido bien, marque la versión como pendiente de despliegue en producción (hacer un merge a master).

Las pruebas estáticas (Flake8 y Bandit) no tendrán ningún tipo de validación en cuanto a QualityGates, y se dará el resultado de la etapa como válida siempre que se haya podido generar y publicar un informe con el resultado de estas pruebas.

Por la parte de las pruebas dinámicas, para desligar la práctica de las habilidades de programación, se ofrecen dos opciones para llevarlas a cabo:

- Pruebas de integración haciendo uso de Pytest, tal y como se estuvo trabajando en los CP1.1 y CP1.2.
- Pruebas de integración mediante ejecución directa de comandos curl y, validación del resultado mediante scripts de shell (Linux, puesto que se ejecutará en la máquina EC2, Ubuntu).

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

- Etapa “Get Code” para descargar el código fuente del repositorio (rama develop).
  - Se puede usar el comando “git” de Jenkins, o bien el comando “checkout” también de Jenkins.
- Etapa “Static Test” para la ejecución de las pruebas de análisis estático.
  - Deben publicarse ambos informes, tanto de Flake8 como de Bandit.
  - No importa cuántos hallazgos se encuentren, si la ejecución de los comandos es correcta y los informes se publican, la etapa se marcará como satisfactoria.

- Solo debe validarse el código fuente de las funciones lambda y código asociado (carpeta "/src").

- Etapa de despliegue SAM ("Deploy"):
  - Debemos usar los comandos necesarios para construir, validar y desplegar nuestros recursos serverless, en un entorno de Staging.
  - Lógicamente, y puesto que estos comandos se ejecutarán automáticamente desde Jenkins, no podremos usar el modo interactivo (guiado) del comando "sam deploy", por lo que deberemos indicar los parámetros que sean necesarios de alguna otra manera.
- Etapa "Rest Test" para la ejecución de las pruebas de integración, sobre el entorno recién levantado.
  - En el CP1.3 vimos cómo llamar a la API Rest mediante comandos *curl*. El resultado que nos ofrecían estos comandos nos permitía validar si el despliegue había sido correcto y si el funcionamiento de la aplicación era el esperado. Para este CP1.4, y con el objetivo de ofrecer mayor libertad a los alumnos, se permiten dos vías para llevar a cabo estas pruebas de integración:

    Siguiendo la línea del CP1.3, mediante llamadas *curl*. En este caso, el alumno (entiéndase grupo de alumnos), deberá validar automáticamente que la salida de los comandos *curl* es la esperada en todo momento, de manera que se pueda detectar un error si lo que nos devuelve la llamada a la API no es igual a lo esperado para cada comando.

    Deben validarse por esta vía todas las llamadas a cada uno de los métodos de la API. Si falla cualquiera de ellas, la etapa debe marcarse como roja (fallida, unhealthy) y abortar el pipeline.

    Siguiendo la línea del CP1.2, haremos uso de Pytest para ejecutar una serie de pruebas, ya desarrolladas en Python, y disponibles en el fichero test/integration/todoApiTest.py del repositorio facilitado para este CP1.4 (el mismo que el del CP1.3).

    Si falla cualquiera de estas pruebas, la etapa debe marcarse como fallida, y el pipeline debe abortarse.

Para la ejecución de estas pruebas mediante Pytest, es posible que el alumno deba realizar alguna modificación al fichero de tests proporcionado, o aplicar alguna configuración adicional.

- Etapa “Promote” para marcar la versión como “Release” y ser desplegada en producción.
  - Si todo ha funcionado correctamente, se mergeará el código con la rama master (al no haber cambios, más que un fichero de texto, no habrá problemas de conflictos), para indicar que la versión se puede desplegar en producción.
  - Se puede usar cualquier comando/plugin para llevar a cabo este merge, aunque se recomienda el comando git.

Reto 2 – Pipeline “CD”

Una vez que ya tenemos nuestra nueva versión en master, debe iniciarse la ejecución del pipeline “CD”, que se encargue del despliegue en producción.

Las etapas de este pipeline son las siguientes:

- Etapa “Get Code” para descargar el código, de la rama master.
- Etapa “Deploy”:
  - El entorno en el que desplegaremos este stack será “production”.
  - No, no hay diferencias relevantes entre el despliegue en un entorno u otro, es simple nomenclatura en este caso, pero es importante saber dónde estamos desplegando y dónde estamos accediendo luego para ejecutar las pruebas.
- Etapa “Rest Test”:
  - Mismos requisitos que en el reto anterior, pero limitando la ejecución de las pruebas a aquellas que sean de solo lectura, para no alterar los datos de un entorno productivo.

Reto 3 – Distribución en agentes

Como en apartados anteriores, se debe realizar una separación de las etapas en varios agentes, ya que éste es el procedimiento habitual en el mundo real.

La distribución en agentes se realizará de la siguiente manera:

- Un agente para las pruebas de análisis estático (Flake8 y Bandit).
- Un agente para las pruebas de la Api Rest (Pytest o curl+shellscript).
  - Aquí se produce una situación algo más compleja, puesto que el agente en el que se ejecutarán estas pruebas no conoce la URL del entorno serverless que se ha levantado en la etapa anterior.
  - El alumno deberá buscar soluciones a este problema.
- El resto de tareas se pueden ejecutar en el agente por defecto.
  - Se recuerda al alumno que esto no es una buena práctica, ya que el nodo principal de Jenkins se debe usar como mero gestor de los procesos, pero no como ejecutor de tareas reales. Solo se realiza en este caso para simplificar el ejercicio.

Debe realizarse la distribución en agentes, tanto para el pipeline CI como para el CD. El Jenkinsfile deberá llamarse “Jenkinsfile_agentes”, como en los apartados anteriores.

Reto 4 – Separación Configuración

Este reto pretende que el alumno se familiarice con la separación del código fuente y de la configuración de una aplicación (configuración dependiente del entorno), de manera que se almacene en repositorios distintos en vez de formar parte del mismo código fuente, lo cual es una mala práctica.

En este CP1.4, la única configuración dependiente del entorno es el nombre de la tabla de BD que usamos para almacenar la lista de To-Do's, puesto que todo lo demás lo gestiona automáticamente AWS y nos indica URLs; la región es única y la misma para ambos entornos, etc.

Este nombre de la tabla de BD se indica en la plantilla de despliegue SAM, y se elige un nombre u otro según estemos en CI o en CD (en develop o en master), indicándolo en las llamadas a "sam deploy", lo cual se realiza en el Jenkinsfile.

Con esto se pretende aclarar que no existe configuración dependiente del entorno para esta aplicación (más que un parámetro en el Jenkinsfile para el comando sam deploy) y que, por lo tanto, lo que vamos a hacer es un ejercicio simulado, de cómo se hace en la vida real.

Para este reto, tendremos que crear otro repositorio en GitHub, en el que solo guardaremos un fichero, el fichero "samconfig.toml". Este fichero tendrá una versión en la rama "staging", y otra versión en la rama "production".

Cada fichero tendrá unos contenidos diferentes y exclusivos del entorno.

Cuando estemos desplegando en el entorno de staging, es decir, cuando estemos ejecutando el Jenkinsfile de la rama develop, deberemos descargar además este fichero samconfig.toml de la rama "staging".

De manera análoga lo haremos para la rama master, para “production”.

Las tareas a realizar en este reto son:

- Creación del nuevo repositorio “todo-list-aws-config”:
  - Debe tener dos ramas: “staging” y “production”.

    Se permite tener una rama master, pero no se usará para nada.
- Commit/Push del fichero samconfig.toml en cada una de las dos ramas:
  - Cada rama tendrá una parte del fichero, correspondiente a la configuración de su entorno. Es decir, hay que partir el fichero.
- Tras descargarnos el código fuente de nuestra aplicación en la etapa Get Code, nuestros pipelines descargarán también, y en la misma etapa, el fichero samconfig.toml de la rama correcta, según estemos en CI o en CD.
  - Se puede usar cualquier comando para descargar este fichero (git, wget, curl, etc.).

Reto 5 – Pipeline Multibranch

Para redondear este CP1.4, y todo el Caso Práctico 1, en este último reto usaremos la característica de Pipeline Multibranch de Jenkins, para evitar tener dos pipelines por separado, y unificarlos en un único pipeline al que llamaremos CI/CD.

Este pipeline será de tipo multibranch, y monitorizará cambios tanto en la rama develop como en master.

Cuando detecte cambios en una rama u otra, descargará el Jenkinsfile correspondiente a esa rama, y, por tanto, ejecutará un pipeline de CI o de CD, según la rama en la que se haya producido el cambio.

Las tareas a realizar en este reto son:

- Creación del pipeline multibranch, en Jenkins.
- Configuración necesaria para que detecte cambios en el repositorio de código, para ambas ramas, y ejecute el Jenkinsfile correspondiente.