Desafío Mox-It

Explicación sobre lo realizado para la actividad realizada

Al comienzo se especifican, tanto el nombre de la función como las ramas afectadas por el flujo de trabajo.

```yaml
name: Build and deploy Python app to Azure Web App - DesafioMox-It

on:
  push:
    branches:
      - main
```

A continuación, se definen los trabajos a realizar y en que sistema operativo se llevaran a cabo. En este caso se usará la última versión de Ubuntu, pero, bien podría ser una versión de Windows o Mac. También se pueden crear instancias para mas de un sistema operativo a la vez, sin embargo, no se abordará el tema en esta ocasión.

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
```

Seguido de esto definimos los “pasos” que se deberán seguir. Comenzando por “usar” la “acción” checkout para verificar si nuestro repositorio es accesible (el “@v4” es un indicativo de la versión que se utilizara).

```yaml
steps:
  - uses: actions/checkout@v4
```

Ahora definimos una función la cual instalara el lenguaje a utilizar. En este caso, se utilizará el lenguaje Python. Para esto, se utilizará la “acción” setup-python. Luego se define la versión a utilizar, especificándola dentro de la palabra reservada “with” utilizando “Python-version” como se muestra en la imagen.

```yaml
- name: Preparando version de Python
  uses: actions/setup-python@v5
  with:
    python-version: '3.10'
```

En la imagen anterior se especificó una versión de Python en concreto. Pero, también se puede escribir esto de forma “semántica” para instalar (en este caso) el ultimo lanzamiento menor de Python 3. Como opcional se puede especificar la arquitectura a utilizar, las cuales, pueden ser ‘x64’ o ‘x86’ (si no se especifica, se utilizará por defecto la arquitectura ‘x64’).

```yaml
python-version: '3.x'
arquitecture: 'x64'
```

A continuación, creamos el entorno virtual y lo “activamos”.

```
- name: Crear e iniciar el entorno virtual
  run: |
    python -m venv venv
    source venv/bin/activate
```

En el caso de las dependencias, estas se pueden instalar de 2 formas. La 1° es utilizando las dependencias especificadas en el archivo “requirements.txt”. La 2° forma es instalando estas de forma manual ejecutando un pip install (opcional usar “--upgrade” para actualizar las dependencias en caso de ya estar instaladas).

```
- name: Instalar dependencias
  run: |
    pip install -r requirements.txt
    python -m pip install --upgrade pip
```

Como opcional, se puede agregar la dependencia “ruff” para identificar código sospechoso dentro de la aplicación.

```
- name: Utilizando Ruff
  run: |
    pip install ruff
    ruff check --output-format=github .
```

También, se puede instalar y ejecutar pruebas tox para analizar e instalar correctamente las dependencias a utilizar en la aplicación. Para esto la versión de Python especificada en el “PATH”

```
- name: Instalar Tox
  run: pip install tox

- name: Ejecutar tox
  run: tox -e py
```

A continuación, se realizan las pruebas unitarias con pytest. Para esto se debe contar con el archivo “test.py” (puede tener otro nombre) y ejecutarlo.

```
- name: Pruebas con pytest
  run: |
    pip install pytest pytest-cov
    pytest tests.py --doctest-modules
    --junitxml=junit/test-results.xml --cov=com
    --cov-report=xml --cov-report=html
```

Por último, se comprimen y suben las dependencias utilizadas para su posterior despliegue. Para esto se utiliza el comando zip “nombre”.zip ./* -r para la compresión y la acción upload-artifact para subirlo, indicando el nombre del paquete y donde guardar el comprimido en cuestión.

```yaml
- name: Comprimir artefactos para el despliegue
  run: zip release.zip ./* -r

- name: Subir artefatos para desplegar el trabajo
  uses: actions/upload-artifact@v4
  with:
    name: python-app
    path: |
      release.zip
      !venv/
```

Pasando ahora al despliegue, se debe indicar el/los sistemas operativos en los cuales vamos a desplegar la aplicación, que se debe ejecutar, en que entorno y con que permisos. En nuestro caso, “que se debe ejecutar” hace referencia a la/las instancias creadas hasta el momento. Para el entorno se deben indicar tanto el nombre como la dirección en la que se encuentra.

```yaml
deploy:
  runs-on: ubuntu-latest
  needs: build
  environment:
    name: 'Production'
    url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
  permissions:
    id-token: write
```

Luego se definen los “pasos” a seguir para desplegar la aplicación. Empezando por descargar y descomprimir los artefactos que se crearon con anterioridad. Para esto se utiliza la acción download-artifact al cual se le indica el nombre del paquete. Luego se ejecuta la función “unzip” indicando el nombre del comprimido a tratar.

```yaml
steps:
  - name: Descargar artefactos
    uses: actions/download-artifact@v4
    with:
      name: python-app

  - name: Descomprimir artefactos para el despliegue
    run: unzip release.zip
```

Para este caso, la aplicación se desplegará en Azure, utilizando la acción “login” (esta vez, de Azure) para ingresar a una cuenta con los premisos necesarios para la modificación del entorno de la aplicación de Azure. Para esto, se deben suministrar datos de la cuenta como se muestra en la imagen.

```yaml
- name: Loguear en Azure
  uses: azure/login@v2
  with:
    client-id: ${{ secrets.AZUREAPPSERVICE_CLIENTID_EF942EA61DAC4A698398B7AF083E9A57 }}
    tenant-id: ${{ secrets.AZUREAPPSERVICE_TENANTID_22225D5E7F9A446CB9D2F3D70B7D5086 }}
    subscription-id: ${{ secrets.AZUREAPPSERVICE_SUBSCRIPTIONID_2CA29891295E4BEFB92CD1605575FD3E }}
```

Finalmente, se ejecuta la acción “webapps-deploy” para desplegar la aplicación creada. Para esto es especifican la “acción” a realizar, la aplicación alojada en el servidor a modificar y donde se ubica.

```yaml
- name: 'Desplegar en Azure Web App'
  uses: azure/webapps-deploy@v3
  id: deploy-to-webapp
  with:
    app-name: 'DesafioMox-It'
    slot-name: 'Production'
```