GitHub Actions

2020.07.31

학부생 연구원 이지헌

언어 및 프레임 워크

언어 및 프레임 워크

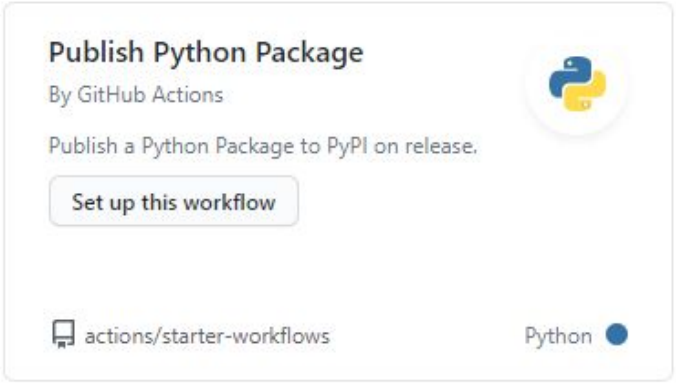
- 프로그래밍 언어로 작성된 프로젝트를 빌드하고 테스트하는 CI(Continuous Integration) workflow 작성 가능
- workflow : 저장소에서 GitHub의 프로젝트를 빌드, 테스트, 패키지, 릴리스 또는 배포하기 위한 사용자 지정 자동화 프로세스

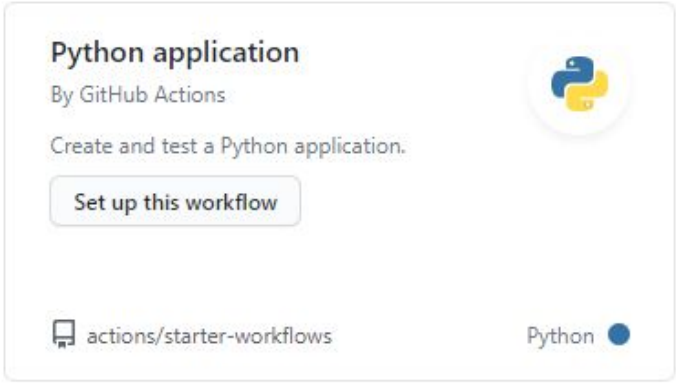
    - JavaScript 및 TypeScript
    - Python
    - Java
    - Docker

파이썬을 위한 GitHub Actions

- GitHub runner(가상 머신)는 Python, PyPy를 포함하여

  사전에 설치된 tool cache(임시 장소)가 있음 # local 설치 필요 X





파이썬을 위한 GitHub Actions

- Python workflow 템플릿 제공

```yaml
# This workflow will install Python dependencies, run tests and lint with a single version of Python
# For more information see: https://help.github.com/actions/language-and-framework-guides/using-python-with-github-actions

name: Python application

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.8
      uses: actions/setup-python@v2
      with:
        python-version: 3.8
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
```

파이썬을 위한 GitHub Actions

[여러 Python 버전 사용]

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      # You can use PyPy versions in python-version.
      # For example, pypy2 and pypy3
      matrix:
        python-version: [2.7, 3.5, 3.6, 3.7, 3.8]

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v2
      with:
        python-version: ${{ matrix.python-version }}
    # You can test your matrix by printing the current Python versior
    - name: Display Python version
      run: python -c "import sys; print(sys.version)"
```

[특정 Python 버전 사용]

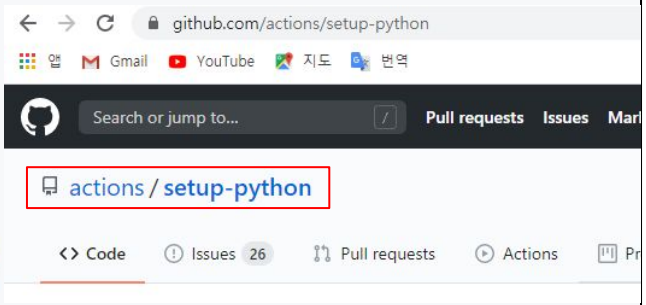
```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.x
      uses: actions/setup-python@v2
      with:
        # Semantic version range syntax or exact version of a Python version
        python-version: '3.x'
        # Optional - x64 or x86 architecture, defaults to x64
        architecture: 'x64'
    # You can test your matrix by printing the current Python version
    - name: Display Python version
      run: python -c "import sys; print(sys.version)"
```



repository의 파일 > runner

파이썬을 위한 GitHub Actions

- GitHub runner(가상 머신)는 pip 패키지 관리자가 설치되어 있음
- 패키지 종속성 관리 가능

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: python -m pip install --upgrade pip setuptools wheel
```

[의존 파일 설치]

```yaml
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
```

[requirements.txt 생성]

파이썬을 위한 GitHub Actions

- 고유 키를 사용하여 pip 종속성 cache, workflow 실행할 때 종속성을 복원 가능
- actions/cache action을 가져와 실행, with 구문으로 설정 (path, key 설정 필수)

```yaml
steps:
- uses: actions/checkout@v2
- name: Setup Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Cache pip
  uses: actions/cache@v2
  with:
    # This path is specific to Ubuntu
    path: ~/.cache/pip
    # Look to see if there is a cache hit for the corresponding requi
    key: ${{ runner.os }}-pip-${{ hashFiles('requirements.txt') }}
    restore-keys: |
      ${{ runner.os }}-pip-
      ${{ runner.os }}-
- name: Install dependencies
  run: pip install -r requirements.txt
```

- path : 저장하고 불러올 cache 대상 폴더
- key : 저장하고 불러올 때 식별 키 값
- restore-keys : 캐시 key가 일치하는 것이 없다면 차선택으로 cache 폴더를 찾는 key

파이썬을 위한 GitHub Actions

- 로컬과 동일한 명령을 사용하여 코드를 빌드, 테스트 가능
- pytest, pytest-cov를 사용한 테스트

```
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Test with pytest
  run: |
    pip install pytest
    pip install pytest-cov
    pytest tests.py --doctest-modules --junitxml=junit/test-results.xm
```

// 설치 및 업그레이드

- Code Coverage : 테스트 코드 작성할 때 참고 지표, 이를 위해 사용하는 것이 pytest-cov

- pytest-cov와 함께 pytest를 실행하면 Coverage Report 생성 어떤 코드를 더 테스트 해야 하는지 확인 가능

파이썬을 위한 GitHub Actions

- Flake8을 사용하여 모든 파일 lint 검사 가능

```
steps:
- uses: actions/checkout@v2
- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'
- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
- name: Lint with flake8
  run: |
    pip install flake8
    flake8 .
```

파이썬을 위한 GitHub Actions

- tox로 테스트를 실행
- 특정 버전을 지정 X, 버전을 선택하는 옵션을 사용하여 tox를 호출해야 함 (-e py)

```yaml
name: Python package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python: [2.7, 3.7, 3.8]

    steps:
      - uses: actions/checkout@v2
      - name: Setup Python
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python }}
      - name: Install Tox and any other packages
        run: pip install tox
      - name: Run Tox
        # Run tox using the version of Python in `PATH`
        run: tox -e py
```

- 다양한 Python 버전에서 사용할 도구를 테스트하기 위해 tox를 사용
- tox.ini 파일 하나를 추가하여 여러 가지 버전에서 테스트를 할 수 있음

파이썬을 위한 GitHub Actions

- workflow가 완료된 후 확인하기 위해 artifact 업로드 사용
- artifact : job끼리 데이터를 공유하거나, workflow가 완료된 후 데이터를 저장할 수 있게 함

  ex) log file, core dump, 테스트결과, 스크린샷 등

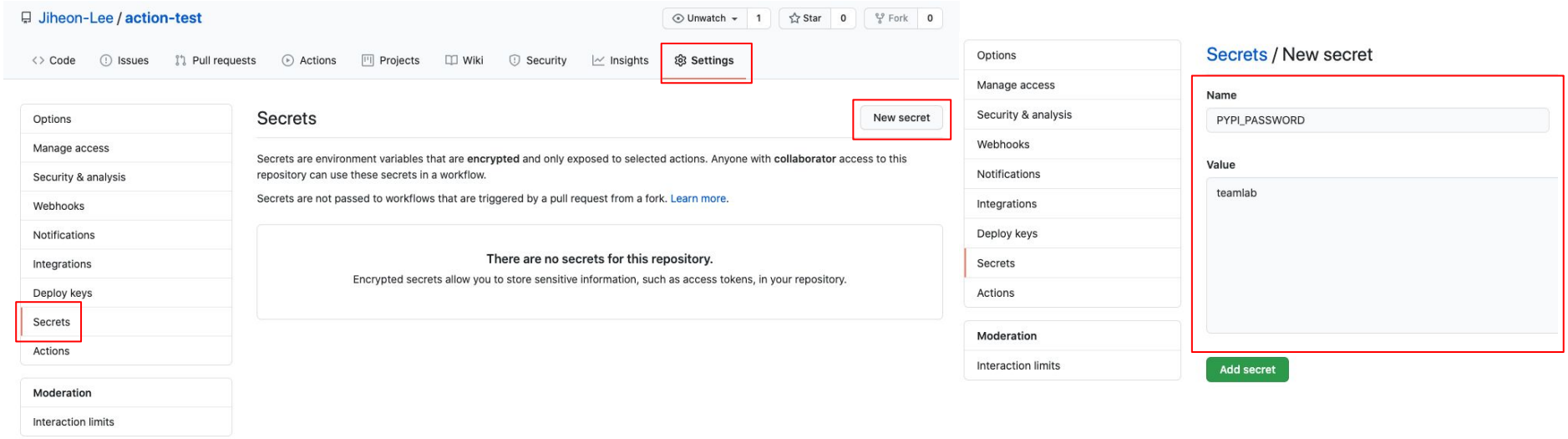
```yaml
- name: Upload pytest test results
  uses: actions/upload-artifact@v2
  with:
    name: pytest-results-${{ matrix.python-version }}
    path: junit/test-results-${{ matrix.python-version }}.xml
  # Use always() to always run this step to publish test results
  if: ${{ always() }}
```

파이썬을 위한 GitHub Actions

- Secret을 사용하여 패키지를 게시할 때 access token 저장하여 사용 가능

  env : 환경변수, workflow에서 사용할 때 $ {{ secrets.”Name” }} 으로 사용

```
env:
  TWINE_USERNAME: ${{ secrets.PYPI_USERNAME }}
  TWINE_PASSWORD: ${{ secrets.PYPI_PASSWORD }}
```



패키지 게시

지속적인 통합 workflow에서 패키징

- 지속적인 통합 워크플로우가 끝날 때 패키지를 생성하면

  PR 요청에 대한 코드 검토에 도움

- 코드를 빌드하고 테스트 한 후 패키징 단계는 실행 또는 배포 가능한 artifact를 생성

- 생성된 artifact를 다운로드하여 PR 요청에서 코드를 실행시켜 디버깅하거나 테스트하는 데 도움

패키지 게시를 위한 workflow

- GitHub 패키지에 패키지 게시
  - master branch에 push 할 때마다 패키지를 게시 가능
  - 이를 통해 항상 master에서 최신 빌드를 쉽게 실행하고 테스트 가능

- 패키지 레지스트리에 패키지 게시
  - 많은 프로젝트의 경우 새 버전의 프로젝트가 release 될 때마다 레지스트리에 게시가 수행됨
  - 모든 release 작성시 패키지 레지스트리에 공개하는 워크 플로우를 작성하여 자동화 가능

지속적인 통합

지속적인 통합

- CI(Continuous Integration)는 매번 코드를 commit 할 때마다
  오류의 원인을 찾을 때 디버그해야하는 코드의 양이 줄어듬
  - 따라서 코드를 지속적으로 빌드 또는 테스트 필요

- 변경 사항을 보다 쉽게 병합 할 수 있음

- 디버깅 또는 병합 충돌 해결에 더 적은 시간을 소비하여
  개발자는 코드 작성에 더 많은 시간 투자가 가능

GitHub workflow를 사용한 CI

- GitHub Actions를 사용하는 CI는 코드 작성, 테스트를 실행할 수 있는 workflow 제공
  - GitHub에서 제안한 CI workflow 템플릿 또는 사용자 지정 사용 가능
- GitHub Event가 발생할 때 설정에 따라 workflow를 실행 가능

- CI 테스트를 실행하고 PR 요청에서 각 테스트의 결과를 제공하여 모든 CI 테스트가 통과되면 변경 내용을 팀 구성원이 검토하거나 병합이 가능해짐

출처

- GitHub Actions：https://docs.github.com/en/actions
- 폴더 캐싱하기：

  https://velog.io/@loakick/Github-Action-React-%EB%B9%8C%EB%93%9C%ED%95%98%EA%B8%B0
- Code Coverage, Python 버전별 실행：https://blog.pingpong.us/python-in-pingpong/

감사합니다