Git Action

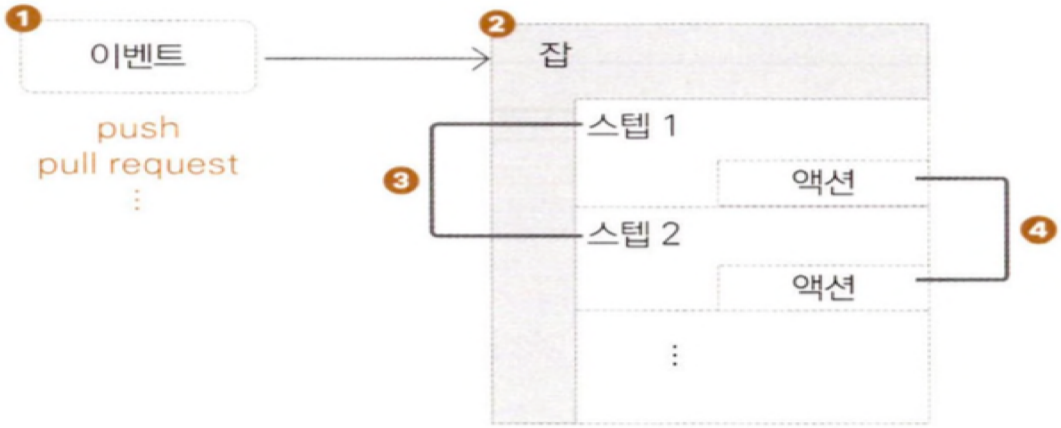
Git Hub Action

- Git Hub Action
  - ✓ 새로운 기능을 개발한다는 것은 새로운 기능을 개발하고 프로젝트 코드를 테스트하고 빌드하고 원격 저장소에 반영하고 배포하는 일련의 과정이 완료되어야 비로소 작업이 마무리 되는데 일련의 작업 과정을 자동화하는 다양한 도구가 있는데 Git Hub는 Git Hub Action이라는 도구를 제공하여 해당 과정의 자동화를 도와줌
  - ✓ Git Hub Action은 소프트웨어 개발에 필요한 작업 주기를 자동화하는 도구
  - ✓ Action은 이벤트 기반으로 특정 이벤트가 발생했을 때 특정 명령 혹은 명령 집합을 자동으로 실행시킬 수 있음
  - ✓ 이벤트란 Pull Request, Push 와 같은 변경을 의미하는 것
  - ✓ 특정 Branch에 Code Push가 발생했을 때 자동으로 실행될 명령을 지정할 수 있는 것이 Git Hub Action

Git Hub Action

❖ Git Hub Action

✓ 이벤트 기반 명령 실행

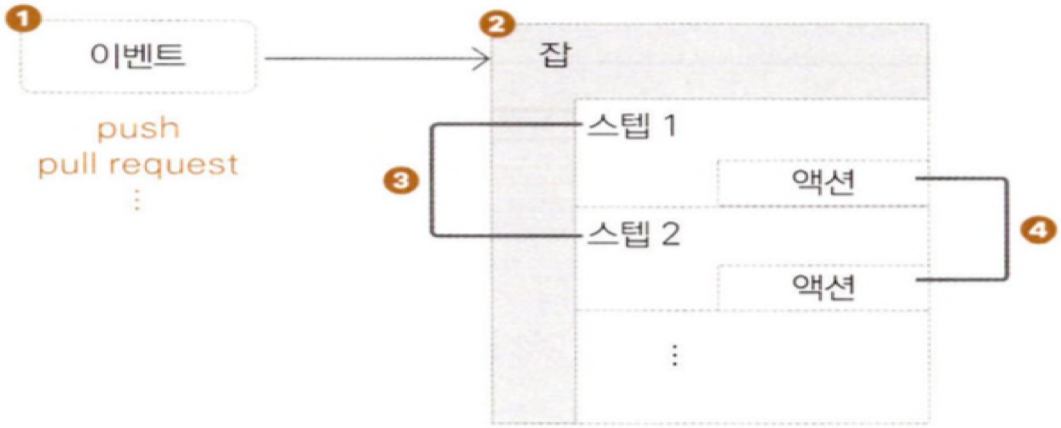


- Event: 정의된 Git Hub Action 작업을 실행시키는 특정 활동으로 풀 리퀘스트가 생성되었을 때 특정 Branch에 새로운 커밋을 Push했을 때 등 어떤 변경이 일어났을 때 해당 Git Hub Action 작업을 실행시킬지 정의할 수 있O 잡
- Jobs: 단일 환경에서 실행될 명령의 집합
- Steps:: 잡 안에서 실행하게 될 명령을 정의
- Actions: 단일 명령 그 자체를 의미

Git Hub Action

❖ Git Hub Action

✓ 이벤트 기반 명령 실행



- Event: 정의된 Git Hub Action 작업을 실행시키는 특정 활동으로 풀 리퀘스트가 생성되었을 때 특정 Branch에 새로운 커밋을 Push했을 때 등 어떤 변경이 일어났을 때 해당 Git Hub Action 작업을 실행시킬지 정의할 수 있O 잡
- Jobs: 단일 환경에서 실행될 명령의 집합
- Steps:: 잡 안에서 실행하게 될 명령을 정의
- Actions: 단일 명령 그 자체를 의미

Git Hub Action

- ❖ Git Hub Action 설정
  - ✓ Git Hub 원격 저장소에 액션 추가하기
    - 저장소에서 Actions 탭을 선택하고 Simple Workflow를 선택해서 작성하는 방법
    - 로컬 저장소에서 ./github/workflows/?.yaml 파일을 생성해서 작성하는 방법

Git Hub Action

- Git Hub Action 설정
  - ✓ 기본적인 구조

```
name: Node.js CI
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v1
      with:
        node-version: ${{ matrix.node-version }}
    - run: npm ci
    - run: npm run build --if-present
    - run: npm test
```

Git Hub Action

- ❖ Git Hub Action 설정
  - ✓ 기본적인 구조
    - name: 해당 워크플로의 이름을 명시
    - on: 워크플로를 실행시킬 이벤트를 명시
      - ◆ push.main 브랜치에 푸시 이벤트가 발생했을 때 해당 워크플로는 동작
      - ◆ pull_request.main 브랜치에 풀 리퀘스트가 생성됐을 때 해당 워크플로는 동작

Git Hub Action

- ❖ Git Hub Action 설정
  - ✓ 기본적인 구조
    - ● jobs
      - ◆ 실행할 작업을 정의
      - ◆ 하나의 작업 내의 명령들은 같은 환경에서 실행
      - ◆ 여러 작업을 정의할 수 있고 각 작업은 개별 환경에서 병렬로 실행
      - ◆ build: 특정 잡의 이름을 의미하는데 식별하기 편한 이름으로 지정
        - ▪ runs-on: 해당 잡이 실행되는 환경을 정의하는데 여기서는 우분투 운영체제에서 잡을 실행시킴
        - ▪ steps: job에 포함된 순차적인 명령 또는 명령의 집합
          - • uses: 현재 단계에서 사용할 액션을 정의하는데 지금은 actions/checkout@v2 액션을 사용해서 원격 저장소에서 소스 코드를 실행 환경으로 가져온다는 의미
          - • name: 현재 단계의 이름으로 원격 저장소의 [Actions] 탭 페이지의 로그에서 해당 단계의 이름을 확인 가능
          - • run: 해당 잡이 실행되는 환경에서 셸 명령어를 실행시틸 수는 있다는 의미인데 run:npm ci는 잡 실행 환경에서 npm ci를 실행

Git Hub Action

- Git Hub Action 팁
  - ✓ 현재 Git Hub Action은 공개 저장소에서 무료로 사용할 수 있는데 비공개 저장소에서는 실행 시간 기준으로 월 2000분까지는 무료로 사용할 수 있음: https://docs.github.com/ko/billing/concepts/product-billing/github-actions
  - ✓ Git Hub Action을 사용하기 전 Circle CI, GitLab CI, Jenkins 등과 같은 다른 도구를 사용하여 수정된 코드의 테스트, 빌드, 배포 과정을 자동화했던 경우는 Git Hub Action 공식 페이지에서 제공하는 마이그레이션 가이드를 참고
    - https://docs.github.com/ko/actions/tutorials/migrate-to-github-actions/manual-migrations/migrate-from-jenkins
    - https://docs.github.com/ko/actions/tutorials/migrate-to-github-actions/manual-migrations/migrate-from-circleci
    - https://docs.github.com/ko/actions/tutorials/migrate-to-github-actions/manual-migrations/migrate-from-gitlab-cicd

Git Hub Action

- Git Hub Action 샘플
  - ✓ 기본 옵션으로 원격 레포지토리 생성(gitaction-sample)
  - ✓ 로컬 레포지토리 생성(gitaction-sample)
  - ✓ 하나의 파일을 만들고 원격 레포지토리에 push
    - git add .
    - git commit -m "git-action-ready"
    - git remote add origin https://github.com/itstudy001/gitaction-sample.git
    - git push origin main
  - ✓ 원격 레포지토리 확인

Git Hub Action

- Git Hub Action 샘플
  - ✓ 로컬 레포지토리에 .github/workflows 디렉토리를 생성하고 그 안에 yaml 파일을 만들고 작성

```
# 1. Workflow의 이름 (GitHub Actions 탭에 표시됨)
name: Basic CI Example

# 2. 언제 이 워크플로우를 실행할지 결정 (Trigger)
on:
  push:
    branches: [ "main" ]  # main 브랜치에 코드가 push될 때 실행
  pull_request:
    branches: [ "main" ]  # main 브랜치로 PR이 생성될 때 실행
```

Git Hub Action

- Git Hub Action 샘플
  - ✓ 로컬 레포지토리에 .github/workflows 디렉토리를 생성하고 그 안에 yaml 파일을 만들고 작성

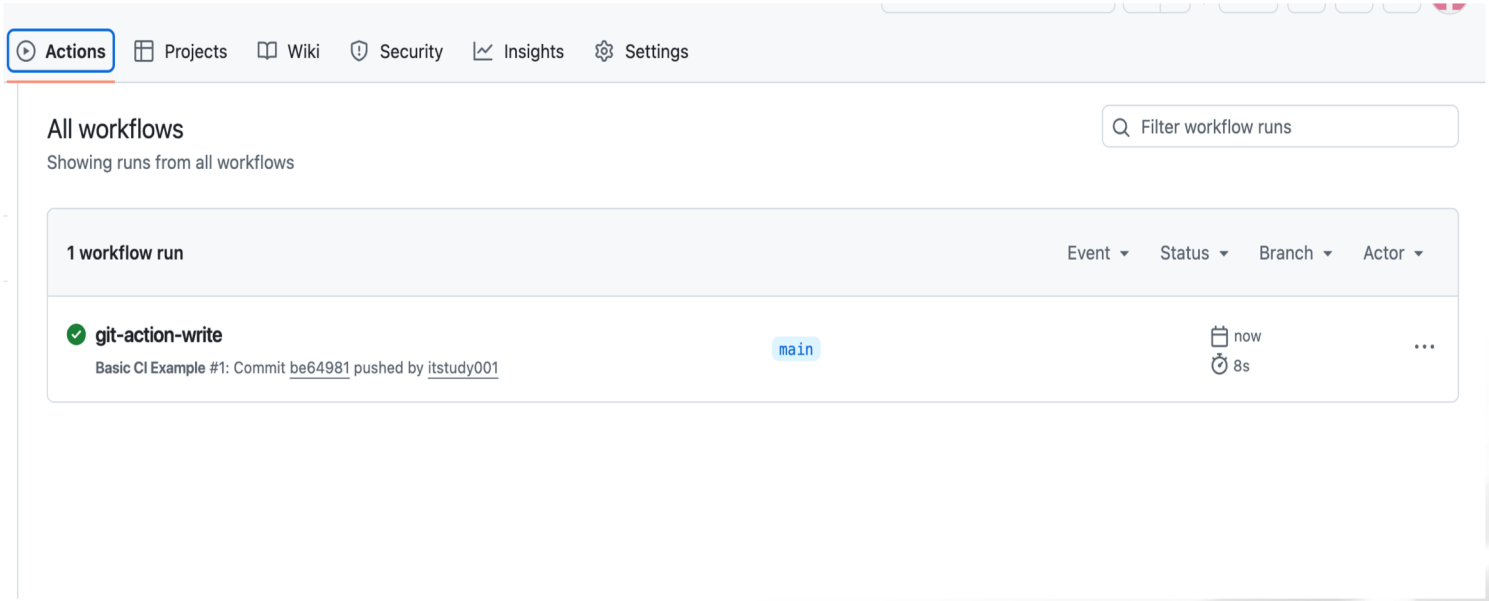
```
# 3. 실행할 작업들 (Jobs)
jobs:
  build:
    # 실행 환경 (가상 머신 OS)
    runs-on: ubuntu-latest
    # 실행 단계 (Steps)
    steps:
      # (1) 저장소의 코드를 가상 머신으로 가져오기 (Check out)
      - name: Checkout repository code
        uses: actions/checkout@v4
      # (2) 화면에 메시지 출력하기
      - name: Run a one-line script
        run: echo "Hello, GitHub Actions! 환경 설정이 완료되었습니다."
      # (3) 파이썬 설치 및 버전 확인 (Action 활용)
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.10'
      - name: Display Python version
        run: python --version
```

Git Hub Action

- ❖ Git Hub Action 샘플
  - ✓ 원격 레포지토리로 push
    - git add .
    - git commit -m "git-action-ready"
    - git push origin main

Git Hub Action

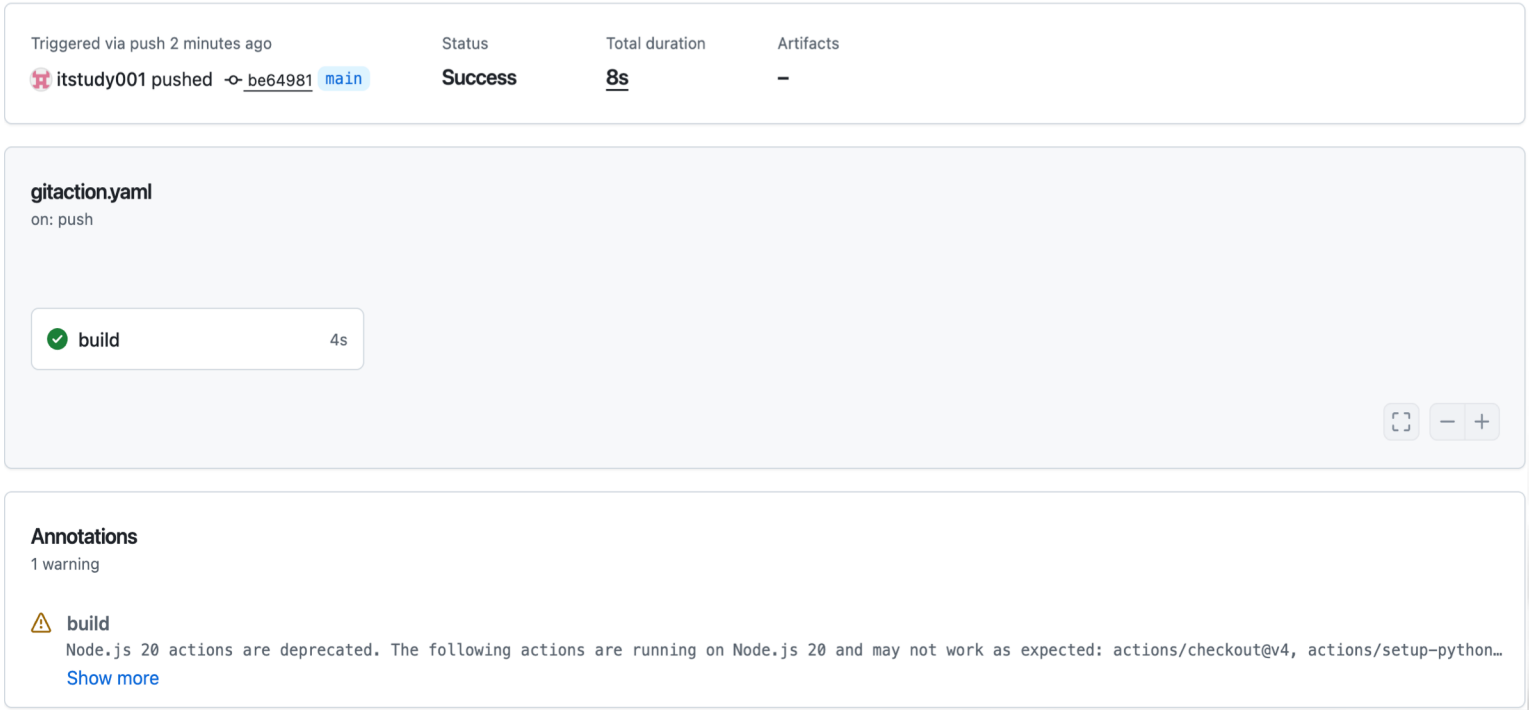
- Git Hub Action 샘플
  - ✓ 원격 레포지토리 확인



Git Hub Action

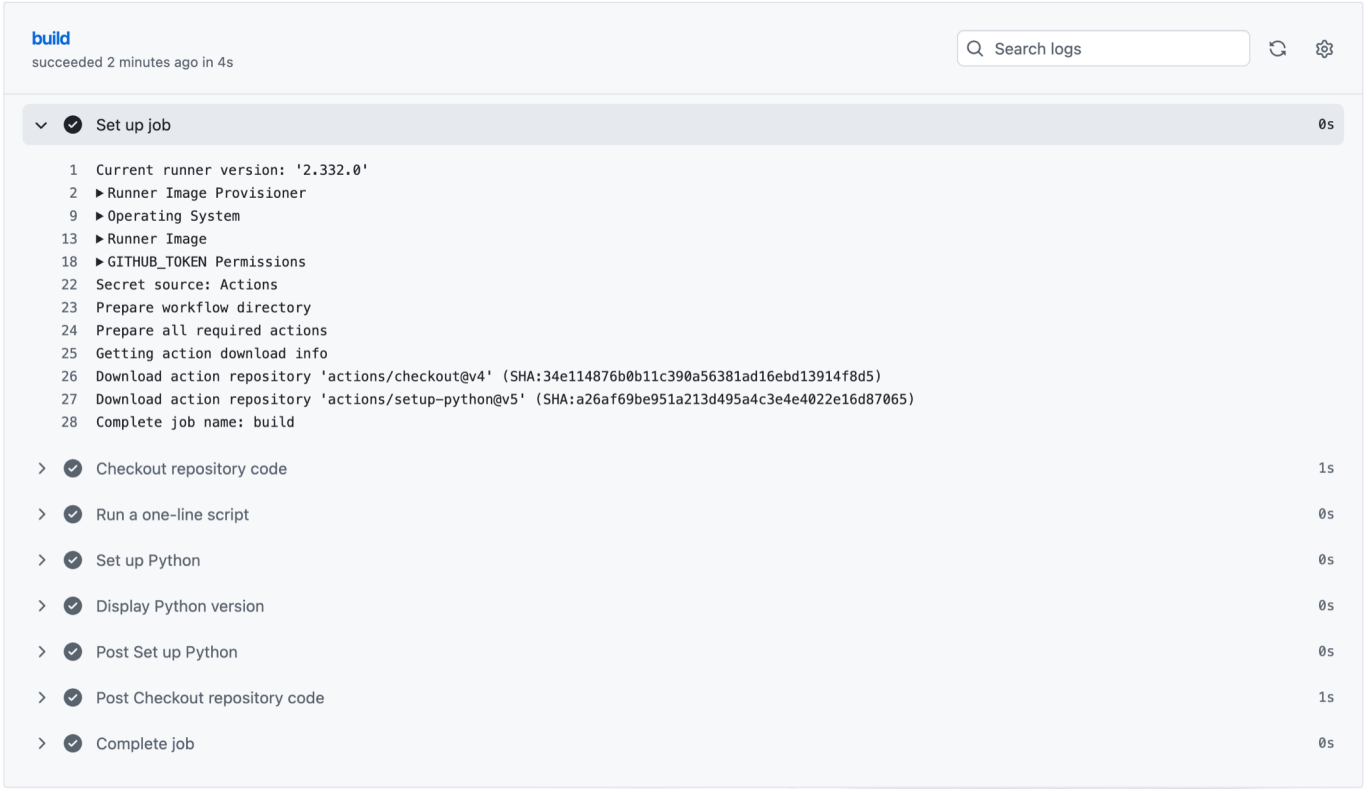
❖ Git Hub Action 샘플

✓ 원격 레포지토리 확인



Git Hub Action

- ❖ Git Hub Action 샘플
  - ✓ 원격 레포지토리 확인



Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ 원격 레포지토리 생성: nodetest
  - ✓ 노드 프로젝트 생성
    - mkdir nodetest
    - cd nodetest
    - npm init
    - 테스트 패키지 설치: npm install mocha
    - test.spec.js 파일을 생성하고 작성

```
describe('Default Test Set', () => {
    it('test1 should be passed', () => {
        console.log('test1 passed')
    })

    it('test2 should be passed', () => {
        console.log('test2 passed')
    })
})
```

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ 노드 프로젝트 생성
    - 실행: ./node_modules/.bin/mocha test.spec.js

```
 Default Test Set
test1 passed
        test1 should be passed
test2 passed
        test2 should be passed


 2 passing (2ms)
```

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ 노드 프로젝트 생성
    - package.json 파일의 scripts 를 수정

```
"scripts": {
    "test": "./node_modules/.bin/mocha test.spec.js",
    "start": "node ./bin/www"  },
```

    - npm test로 실행

```
 Default Test Set
test1 passed
        test1 should be passed
test2 passed
        test2 should be passed


 2 passing (2ms)
```

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ Git Hub에 프로젝트 업로드
    - .gitignore 파일 생성 후 작성

      node_modules/
    - git add .
    - git commit -m "nodetest"
    - git remote add origin https://github.com/itstudy001/nodetest.git
    - git push origin main

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ Git Hub에 Actions 탭에서 Workflow 파일을 추가하고 작성

```
name: Node.js CI

# 구독할 이벤트
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
```

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ Git Hub에 Actions 탭에서 Workflow 파일을 추가하고 작성

```
# jobs 단위로 개별 서버(정확히는 Docker 컨테이너 단위라고 한다.)에서 작업을 수행
# 각 작업은 병렬로 실행 된다고 하는데 needs: build와 같이 표시해서 기다릴 수도 있음
jobs:
  build:
    # Ubuntu, Windows, MacOS를 지원
    runs-on: ubuntu-latest

    # node-version 과 같이 배열로 돼있으면 해당 원소를 순회하면서 작업이 반복해서 실행
    # 응용해서 runs-on에 여러 OS에서 돌릴 수도 있음
    strategy:
      matrix:
        node-version: [14.x] # 템플릿 기본값: [10.x, 12.x, 14.x]
```

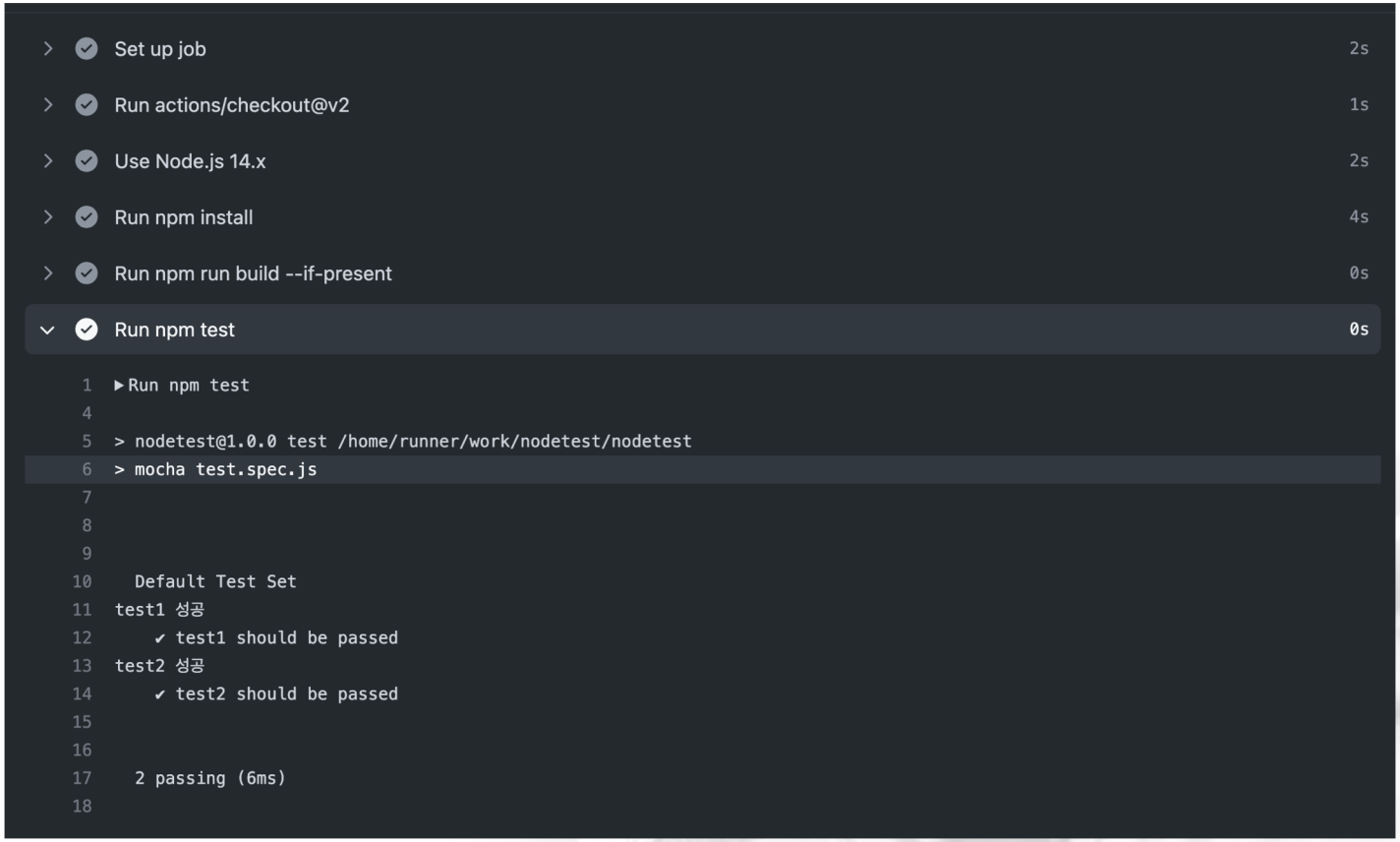
Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실행
  - ✓ Git Hub에 Actions 탭에서 Workflow 파일을 추가하고 작성

```
# uses 개념은 다른 사람이 작성한 내용을 실행하는 개념
# actions/checkout: GitHub의 마지막 커밋으로 Checkout
# actions/setup-node: Node.js를 설치
# run 개념은 명령어를 실행하는 것으로 셸 스크립트와 동일
steps:
  - uses: actions/checkout@v2
  - name: Use Node.js ${{ matrix.node-version }}
    uses: actions/setup-node@v1
    with:
      node-version: ${{ matrix.node-version }}
  # npm ci는 npm install과 같은 기능을 수행
  - run: npm install
  # --if-present 옵션은 npm 스크립트가 존재할 때만 실행시키라는 의미
  # 만약 build 스크립트가 없는 경우, 오류 없이 지나감
  - run: npm run build --if-present
  - run: npm test
```

Git Hub Action

- node 테스트 프로젝트를 git hub action으로 실
  - ✓ 결과 확인



Git Hub Action

- pytest
  - ✓ pytest 명령어는 이름이 test_로 시작하는 모든 파일을 로드하고 test_로 시작되는 모든 함수를 실행하는 패키지
  - ✓ 테스트 코드 작성의 장점
    - 디버깅 시간을 절약
    - 코드를 작성하기 전이나 작성하면서 코드에 대해 생각하게 만듬
    - 처음부터 효율적인 코드를 작성
    - 문서를 제공
    - 배포를 원활하게 유지하는 데 도움이 됨
  - ✓ 설치: pip install pytest

Git Hub Action

- pytest
  - ✓ test_sample.py 파일을 만들고 작성

```
def test_true():
    assert True
```

  - ✓ pytest 수행

```
platform darwin -- Python 3.11.4, pytest-7.4.0, pluggy-1.0.0
rootdir: /Users/adam/Documents/lecture/DevOps/git/Practice/pythontesting
plugins: anyio-3.5.0
collected 1 item

test_sample.py .
[100%]

============================================================
= 1 passed in 0.00s =
```

Git Hub Action

- pytest
  - ✓ test_sample.py 파일을 수정하고 작성

```
def test_false():
    assert False
```

  - ✓ pytest 수행

```
================================== test_false
    def test_false():
>       assert False
E       assert False

test_sample.py:2: AssertionError
=========================================================
short test summary info
=========================================================
FAILED test_sample.py::test_false - assert False
=========================================================
= 1 failed in 0.04s =============
```

Git Hub Action

- pytest
  - ✓ 테스트 건너뛰기
    - 어떤 테스트를 실행할 수 없는 경우 해당 테스트를 건너뛸 수 있음
    - pytest.skip() 함수를 사용하여 테스트를 건너뛴 것으로 표시하고 다음 함수로 이동할 수 있음
    - pytest.mark.skip 데코레이터는 테스트 함수를 무조건 건너는데 테스트를 항상 건너뛰어야 할 때 사용

Git Hub Action

- pytest
  - ✓ test_sample.py 파일을 수정하고 작성

```
import pytest

try:
    import mylib
except ImportError:
    mylib = None

@pytest.mark.skip("Do not run this")
def test_fail():
    assert False

@pytest.mark.skipif(mylib is None, reason="mylib is not available")
def test_mylib():
    assert mylib.foobar() == 42

def test_skip_at_runtime():
    if True:
        pytest.skip("Finally I don't want to run it")
```

Git Hub Action

- pytest
  - ✓ 테스트 수행 – pytest

```
============================================================
test session starts
============================================================
platform darwin -- Python 3.11.4, pytest-7.4.0, pluggy-1.0.0
rootdir: /Users/adam/Documents/lecture/DevOps/git/Practice/pythontesting
plugins: anyio-3.5.0
collected 3 items

test_sample.py sss
[100%]

============================================================
3 skipped in 0.00s
============================================================
=
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ Mongo DB 컨테이너 실행
    - docker run --name mongodb -v ~/data:/data/db -d -p 27017:27017 mongo
    - bash shell 접속: docker exec -it mongodb bash
  - ✓ 프로젝트 생성
    - python 가상 환경 생성 및 활성화
    - 패키지 설치
      - pip install flask
      - pip install requests
      - pip install beautifulsoup4
      - pip install pymongo

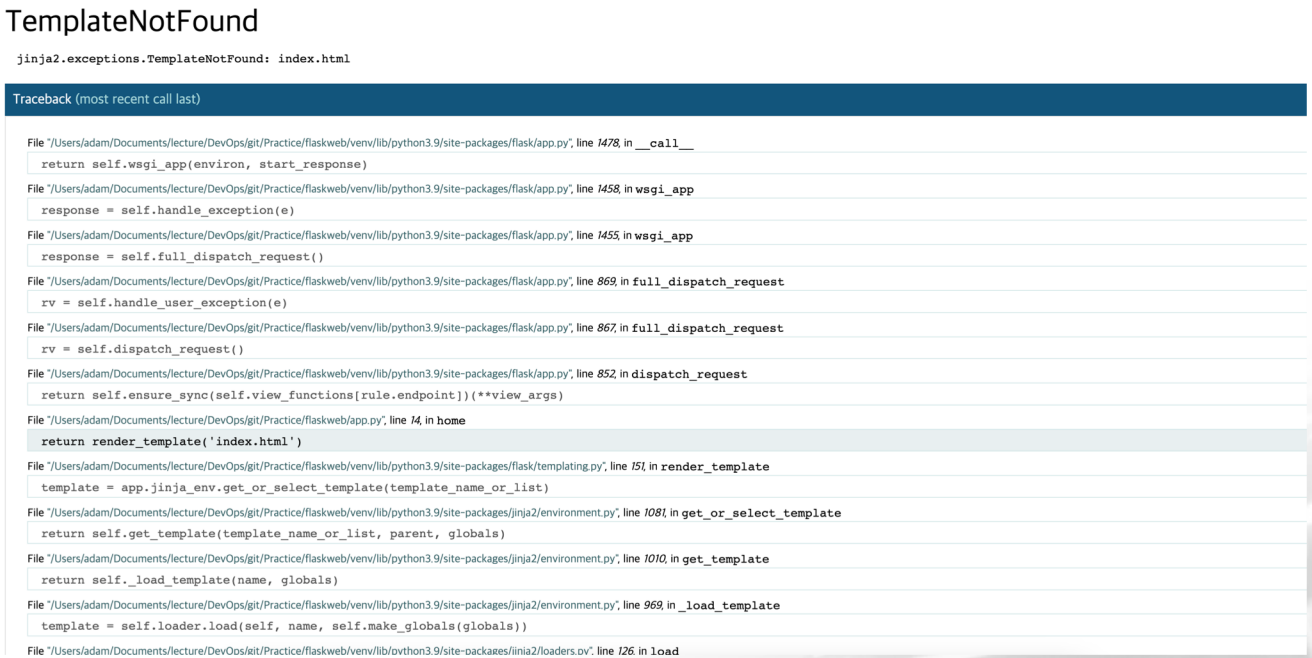
Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 프로젝트에 app.py 파일을 추가하고 작성

```
from flask import Flask, render_template, jsonify, request

app = Flask(__name__)

from pymongo import MongoClient

client = MongoClient('mongo', 27017)
db = client.dbsparta


## HTML을 주는 부분
@app.route('/')
def home():
    return render_template('index.html')


if __name__ == '__main__':
    app.run('0.0.0.0', port=5000, debug=True)
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 프로젝트 실행 – python app.py
  - ✓ 브라우저에서 localhost:5000 접속



TemplateNotFound

jinja2.exceptions.TemplateNotFound: index.html

Traceback (most recent call last)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 1478, in __call__
return self.wsgi_app(environ, start_response)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 1458, in wsgi_app
response = self.handle_exception(e)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 1455, in wsgi_app
response = self.full_dispatch_request()

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 869, in full_dispatch_request
rv = self.handle_user_exception(e)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 867, in full_dispatch_request
rv = self.dispatch_request()

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/app.py", line 852, in dispatch_request
return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/app.py", line 14, in home
return render_template('index.html')

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/flask/templating.py", line 151, in render_template
template = app.jinja_env.get_or_select_template(template_name_or_list)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/jinja2/environment.py", line 1081, in get_or_select_template
return self.get_template(template_name_or_list, parent, globals)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/jinja2/environment.py", line 1010, in get_template
return self._load_template(name, globals)

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/jinja2/environment.py", line 969, in _load_template
template = self.loader.load(self, name, self.make_globals(globals))

File "/Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb/venv/lib/python3.9/site-packages/jinja2/loaders.py", line 126, in load

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 프로젝트에 templates 디렉토리를 만들고 index.html 파일을 작성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>index.html</title>
</head>
<body>
<h3> Flask Web Application</h3>
</body>
</html>
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 브라우저에서 새로 고침

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ utils.py 파일을 생성하고 필요한 함수 작성

```
import requests
from bs4 import BeautifulSoup


def get_movie_info(url_receive):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)AppleWebKit/537.36
(KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36'}
    data = requests.get(url_receive, headers=headers)

    soup = BeautifulSoup(data.text, 'html.parser')

    title = soup.select_one('meta[property="og:title"]')['content']
    image = soup.select_one('div.cm_info_box > div.detail_info > a > img')['src']
    desc = soup.select_one('div.cm_info_box > div.detail_info > div > span').get_text()

    return title, image, desc
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 테스트에 이용할 함수를 소유한 test_utils.py 파일을 생성하고 함수 작성

```
from utils import get_movie_info

def test_get_movie_info():
    test_url = "https://movie.naver.com/movie/bi/mi/basic.nhn?code=185293"
    title, image, desc = get_movie_info(test_url)
    print(title)
    print(image)
    print(desc)
    assert title == "내일의 기억 : 네이버 검색"
    assert image ==
"https://search.pstatic.net/common?type=o&size=176x264&quality=85&direct=true&src=https%3A%2F%2Fs.pstatic.net%2Fmovie.phinf%2F20210406_131%2F1617688160755B157W_JPEG%2Fmovie_image.jpg%3Ftype%3Dw640_2"
    assert desc == """사고로 기억을 잃은 채 깨어난 수진 옆엔 자상한 남편 지훈이 그녀를 세심하게 돌봐주고 있다. 그리고 집에 돌아온 후, 마주친 이웃들의 위험한 미래가 보이기 시작하자 수진은 혼란에 빠진다. 그러던 어느 날 길에서 만난 옛 직장 동료는 수진을 걱정하며 지훈에 대한 믿기 힘든 소리를 하고, 때마침 발견한 사진에서 사진 속 남편 자리엔 지훈이 아닌 다른 남자가 있다. 설상가상 수진은 알 수 없는 남자가 자신을 위협하는 환영에 시달리는데....."""
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ pytest 수행

```
============================================================
test session starts
============================================================
platform darwin -- Python 3.11.4, pytest-7.4.0, pluggy-1.0.0
rootdir: /Users/adam/Documents/lecture/DevOps/git/Practice/flaskweb
plugins: anyio-3.5.0
collected 1 item

test_utils.py .
[100%]

============================================================
= 1 passed in 0.70s =
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ app.py 수정

```
from flask import Flask, render_template, jsonify, request

from utils import get_movie_info

app = Flask(__name__)

from pymongo import MongoClient

client = MongoClient('127.0.0.1')
db = client.dbsparta
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ app.py 수정

```
## HTML을 주는 부분
@app.route('/')
def home():
    return render_template('index.html')


@app.route('/memo', methods=['GET'])
def listing():
    articles = list(db.articles.find({}, {'_id': False}))
    return jsonify({'all_articles': articles})
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ app.py 수정

```
## API 역할을 하는 부분
@app.route('/memo', methods=['POST'])
def saving():
    url_receive = request.form['url_give']
    comment_receive = request.form['comment_give']

    title, image, desc = get_movie_info(url_receive)
    doc = {
        'title': title,
        'image': image,
        'desc': desc,
        'url': url_receive,
        'comment': comment_receive
    }

    db.articles.insert_one(doc)

    return jsonify({'msg': '저장이 완료되었습니다!'})
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ app.py 수정

```
if __name__ == '__main__':
    app.run('0.0.0.0', port=5000, debug=True)
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ git hub에서 올리기 위해서 패키지를 requirements.txt 로 내보내기

    pip list --format=freeze > requirements.txt
  - ✓ requirements.txt 파일에 pytest 추가

    pip freeze > requirements.txt

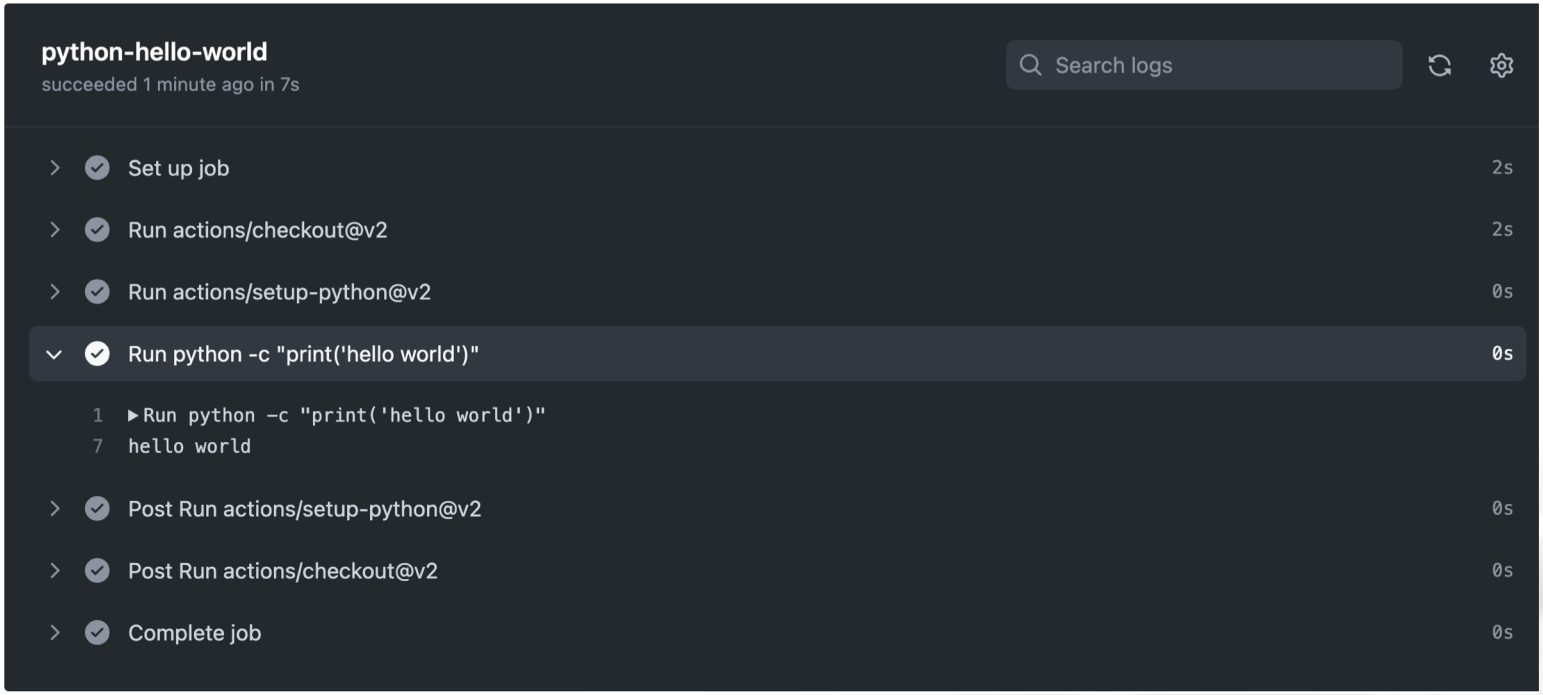
Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 프로젝트를 git hub에 업로드
  - ✓ 프로젝트에 .github/workflows 디렉토리를 생성하고 yaml 파일을 생성하고 작성 - learn_github_action.yaml

```
name: learn-github-actions
on:
  push:
    branches:
      - main
jobs:
  python-hello-world:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: "3.10"
      - run: python -c "print('hello world')"
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 작성한 yaml 파일을 git hub에 업로드 한 후 잠시 후에 Action 확인



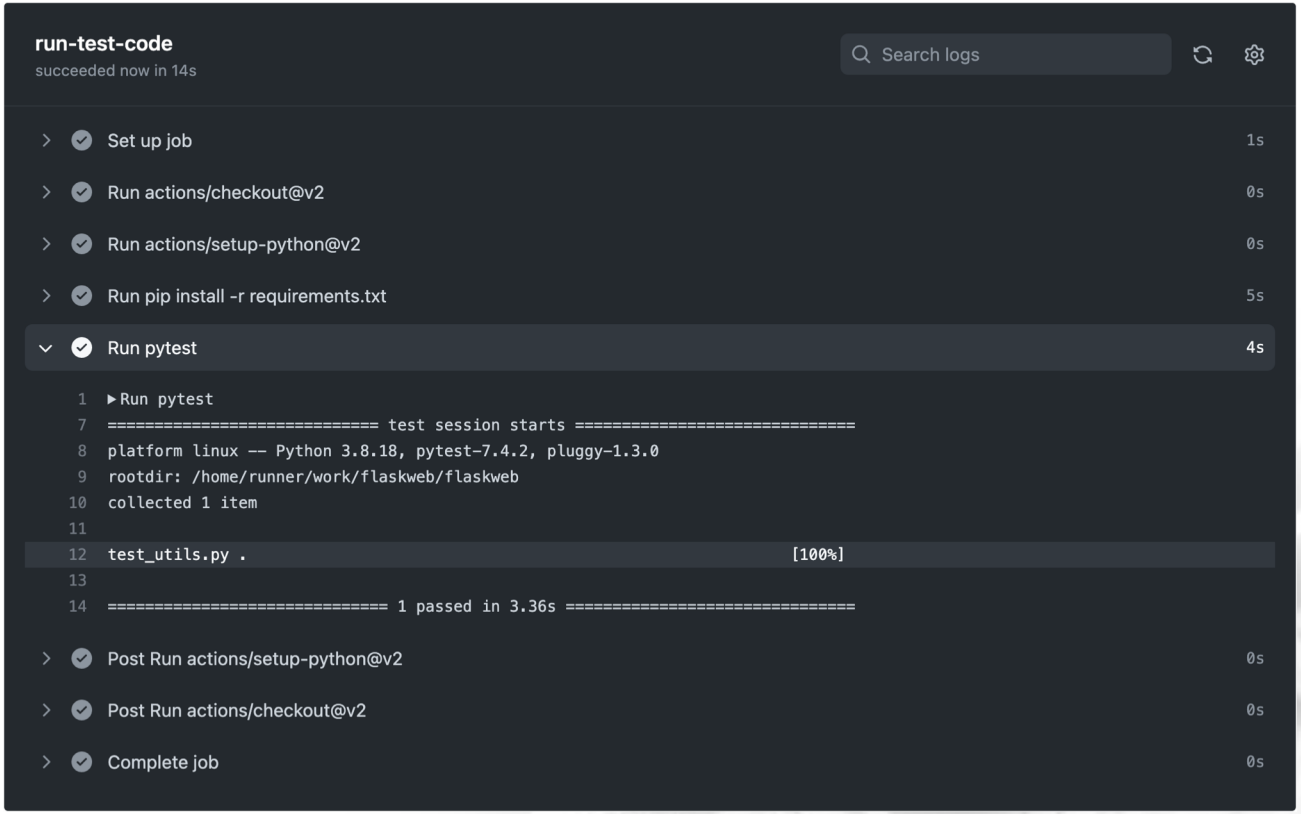
Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 동일한 디렉토리에 테스트를 자동으로 수행하기 위한 yaml 파일을 추가하고 git hub에 업로드 한 후 확인: ci-test.yaml

```
name: ci-test
on:
  push:
    branches:
      - main
jobs:
  run-test-code:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: "3.10"
      - run: pip install -r requirements.txt
      - run: pytest
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 동일한 디렉토리에 테스트를 자동으로 수행하기 위한 yaml 파일을 추가하고 git hub에 업로드 한 후 확인: ci-test.yaml



Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ Dockerfile을 추가하고 작성

```
FROM python:3.9-slim

RUN apt-get update ₩
    && apt-get install -y --no-install-recommends ₩
            postgresql-client ₩
        && rm -rf /var/lib/apt/lists/*

WORKDIR /usr/src/app
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY . .

EXPOSE 5000


CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ 이미지 빌드: docker build -t ggangpae1/flaskweb .
  - ✓ 컨테이너 생성: docker run -dit -p 5000:5000 --name flaskweb ggangpae1/flaskweb

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 파일을 만들고 작성

```
name: ci-pipeline
on:
  push:
    branches:
      - main
jobs:
  run-test-code:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: "3.10"
      - run: pip install -r requirements.txt
      - run: pytest
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 파일을 만들고 작성

```
build:
  needs: [run-test-code]
  runs-on: ubuntu-latest
  steps:
    - name: Checkout
      uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Login to DockerHub
      uses: docker/login-action@v1
      with:
        username: ggangpae1
        password: dckr_pat_E0BlvfTnZhHcvUgyo70_VZyt4x8
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 파일을 만들고 작성

```
- name: build and release to DockerHub
  env:
    NAME: ggangpae1
    REPO: flaskweb
  run: |
    docker build -t $REPO .
    docker tag $REPO:latest $NAME/$REPO:latest
    docker push $NAME/$REPO:latest
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ DockerHub에 레포지토리 생성: flaskweb
  - ✓ 코드 푸시 에러

```
remote:      - Push cannot contain secrets
remote:
remote:
remote:       (?) Learn how to resolve a blocked push
remote:       https://docs.github.com/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/working-with-push-protection-from-the-command-line#resolving-a-blocked-push
remote:
remote:
remote:       —— Docker Personal Access Token ——————————————————————————
remote:         locations:
remote:           - commit: 43c38878a1b2e16076ccdfb43d79dd25013bebd5
remote:             path: .github/workflows/ci_test.yaml:34
remote:
remote:       (?) To push, remove secret from commit(s) or follow this URL to allow the secret.
remote:       https://github.com/itstudy001/python-test/security/secret-scanning/unblock-secret/3ArmHfCi1QtFQorFSlmOWx7YIw0
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ Secret 이용
    - 해당 프로젝트의 GitHub Repository로 이동
    - Settings > Secrets and variables > Actions 메뉴를 선택
    - New repository secret 버튼을 눌러 아래 두 가지를 각각 추가
      - DOCKERHUB_USERNAME: 사용자의 Docker Hub ID
      - DOCKERHUB_TOKEN: 방금 복사한 Access Token 값
      - DOCKERHUB_REPO: 레포지토리 이름

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 수정

```
name: ci-pipeline
on:
  push:
    branches:
      - main
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 수정

```
jobs:
  run-test-code:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: "3.10"
      - run: pip install -r requirements.txt
      - run: pytest
  build:
    needs: [run-test-code]
    runs-on: ubuntu-latest
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 수정

```
steps:
  - name: Checkout
    uses: actions/checkout@v3
  - name: Set up Python 3.10
    uses: actions/setup-python@v4
    with:
      python-version: '3.10'
  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements.txt
```

Git Hub Action

- python 프로젝트를 git hub action Docker Hub로 배포
  - ✓ ci-pipeline.yaml 수정

```
- name: Login to DockerHub
  uses: docker/login-action@v1
  with:
    username: ${{ secrets.DOCKERHUB_USERNAME }}
    password: ${{ secrets.DOCKERHUB_TOKEN }}
- name: build and release to DockerHub
  run: |
    docker build -t ${{ secrets.DOCKERHUB_REPO }} .
    docker tag ${{ secrets.DOCKERHUB_REPO }}:latest
${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_REPO }}:latest
    docker push
${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_REPO }}:latest
```