

GitHub CI_CD

Git Hub Action

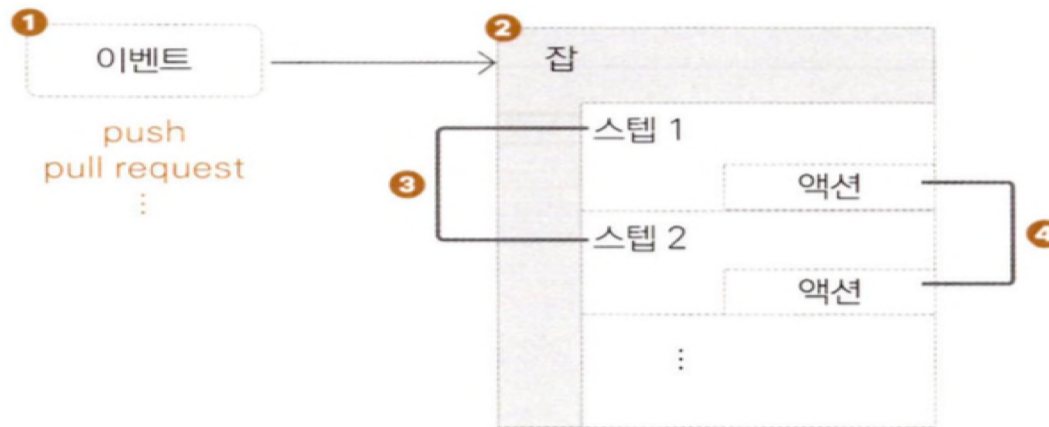
❖ 개요

- ✓ 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 Pull Request가 생성 되었을 때 또는 특정 Branch에 새로운 Commit을 Push했을 때 등 어떤 변경이 일어났을 때
- Jobs: 단일 환경에서 실행될 명령의 집합
- Steps: Job 안에서 실행하게 될 명령을 정의
- Actions: 실제 수행된 명령 자체

Git Hub Action

❖ Git Hub Action 설정

✓ Git Hub 원격 저장소에 액션 추가

- 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

◆ 코드 변경 이벤트

▪ 종류

- push: 코드를 푸시했을 때(특정 브랜치나 태그, 경로 지정 가능)
- pull_request: PR이 생성되거나 업데이트되었을 때
- pull_request_target: PR이 생성될 때 실행되지만 기본 저장소의 컨텍스트에서 실행되어 보안이 필요한 작업에 사용
- create / delete: 브랜치나 태그가 생성되거나 삭제되었을 때

▪ 예시

on:

push:

branches: ["main", "develop"]

paths:

- 'src/**'

pull_request:

types: [opened, reopened]

Git Hub Action

❖ Git Hub Action 설정

✓ 기본적인 구조

- on: 워크플로를 실행시킬 이벤트를 명시

◆ 수동 및 외부 호출 이벤트

▪ 종류

- workflow_dispatch: GitHub 웹 UI에서 Run workflow 버튼을 눌러 직접 실행할 때 사용(입력값 설정 가능)
- repository_dispatch: 외부 앱이나 서버에서 GitHub API를 호출하여 워크플로를 깨울 때 사용
- workflow_call: 다른 워크플로에서 이 워크플로를 재사용(Reusable Workflows)할 때 사용

◆ 스케줄링 (Scheduled events)

- 특정 시간에 주기적으로 실행하고 싶을 때 사용
- schedule: POSIX cron 구문을 사용하여 시간을 지정

on:

schedule:

- cron: '0 0 * * *' # 매일 자정에 실행

Git Hub Action

❖ Git Hub Action 설정

✓ 기본적인 구조

- on: 워크플로를 실행시킬 이벤트를 명시

◆ GitHub 활동 관련 이벤트: 저장소 내의 다양한 활동에 반응

▪ 종류

- issues: 이슈가 생성, 수정, 삭제되었을 때
- issue_comment: 이슈나 PR에 댓글이 달렸을 때
- release: 릴리스(Release)가 배포되거나 편집되었을 때
- watch: 누군가 저장소에 Star를 눌렀을 때 등

◆ 요약

카테고리	주요 키워드	용도
코드 협업	<code>push</code> , <code>pull_request</code>	CI(지속적 통합), 테스트 자동화
수동/연동	<code>workflow_dispatch</code> , <code>repository_dispatch</code>	수동 배포, 외부 시스템 연동
자동화	<code>schedule</code>	정기적인 백업, 리포트 생성
관리/운영	<code>issues</code> , <code>release</code> , <code>label</code>	이슈 관리, 배포 자동화

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 job의 이름을 의미하는데 식별하기 편한 이름으로 지정
 - runs-on: 해당 잡이 실행되는 환경을 정의하는데 여기서는 우분투 운영체제에서 job을 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 실행시킬 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 Repository 생성(githubaction-sample)
- ✓ 로컬 Repository 생성(githubaction-sample)
- ✓ 하나의 파일을 만들고 원격 Repository에 push
 - git init
 - git add .
 - git commit -m "git-action-ready"
 - git remote add origin <https://github.com/itstudy001/githubaction-sample.git>
 - git push origin main
- ✓ 원격 Repository 확인

Git Hub Action

❖ Git Hub Action 샘플

- ✓ 로컬 Repository에 .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 Branch에 코드가 push될 때 실행

pull_request:

branches: ["main"] # main Branch로 PR이 생성될 때 실행

Git Hub Action

❖ Git Hub Action 샘플

- ✓ 로컬 Repository에 .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 Repository로 push

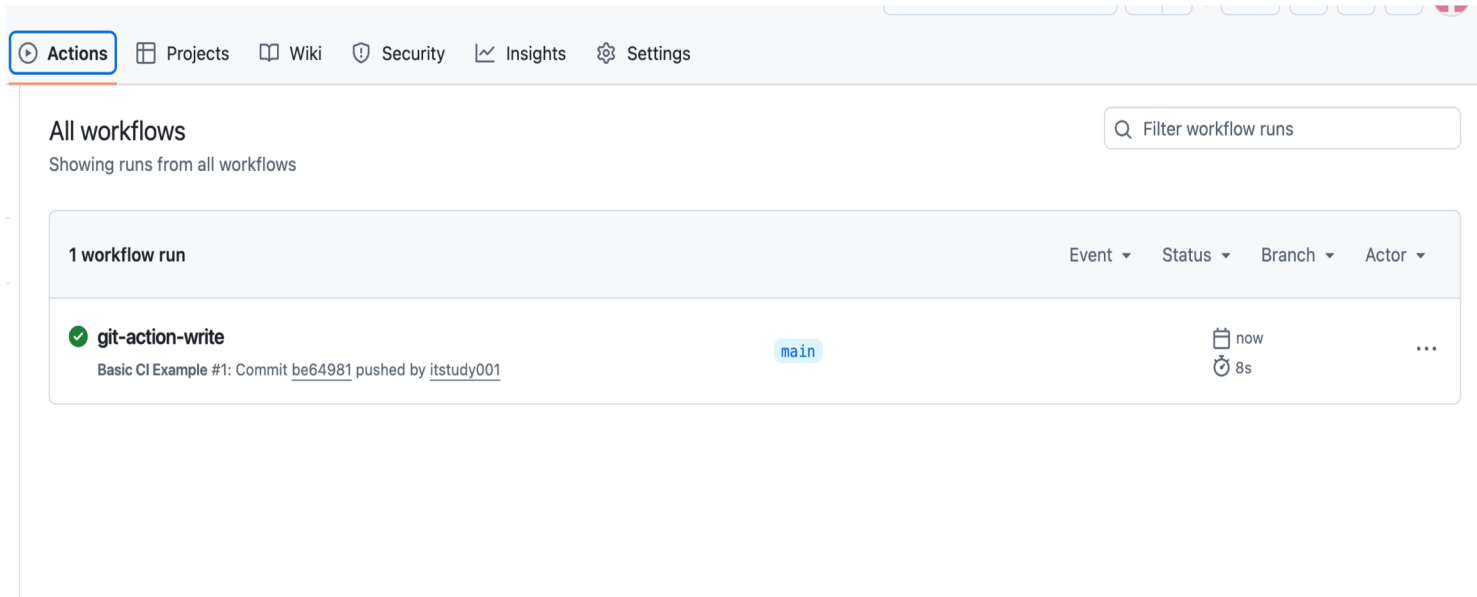
- `git add .`
- `git commit -m "git-action-ready"`
- `git push origin main`



Git Hub Action

❖ Git Hub Action 샘플

✓ 원격 Repository 확인



The screenshot displays the GitHub Actions interface. At the top, there is a navigation bar with tabs for 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The 'Actions' tab is selected. Below the navigation bar, the page is titled 'All workflows' with a subtitle 'Showing runs from all workflows'. A search bar labeled 'Filter workflow runs' is located on the right. The main content area shows a table of workflow runs. The table has columns for 'Event', 'Status', 'Branch', and 'Actor'. A single workflow run is listed with the name 'git-action-write', a status of 'Completed', and a duration of '8s'. The description below the run name reads 'Basic CI Example #1: Commit be64981 pushed by itstudy001'.

Event	Status	Branch	Actor
Completed	Success	main	itstudy001

Git Hub Action

❖ Git Hub Action 샘플

✓ 원격 Repository 확인

Triggered via push 2 minutes ago

itsstudy001 pushed [be64981](#) [main](#)

Status: **Success**

Total duration: **8s**

Artifacts: -

gitaction.yaml

on: push

✓ build 4s

Annotations

1 warning

build

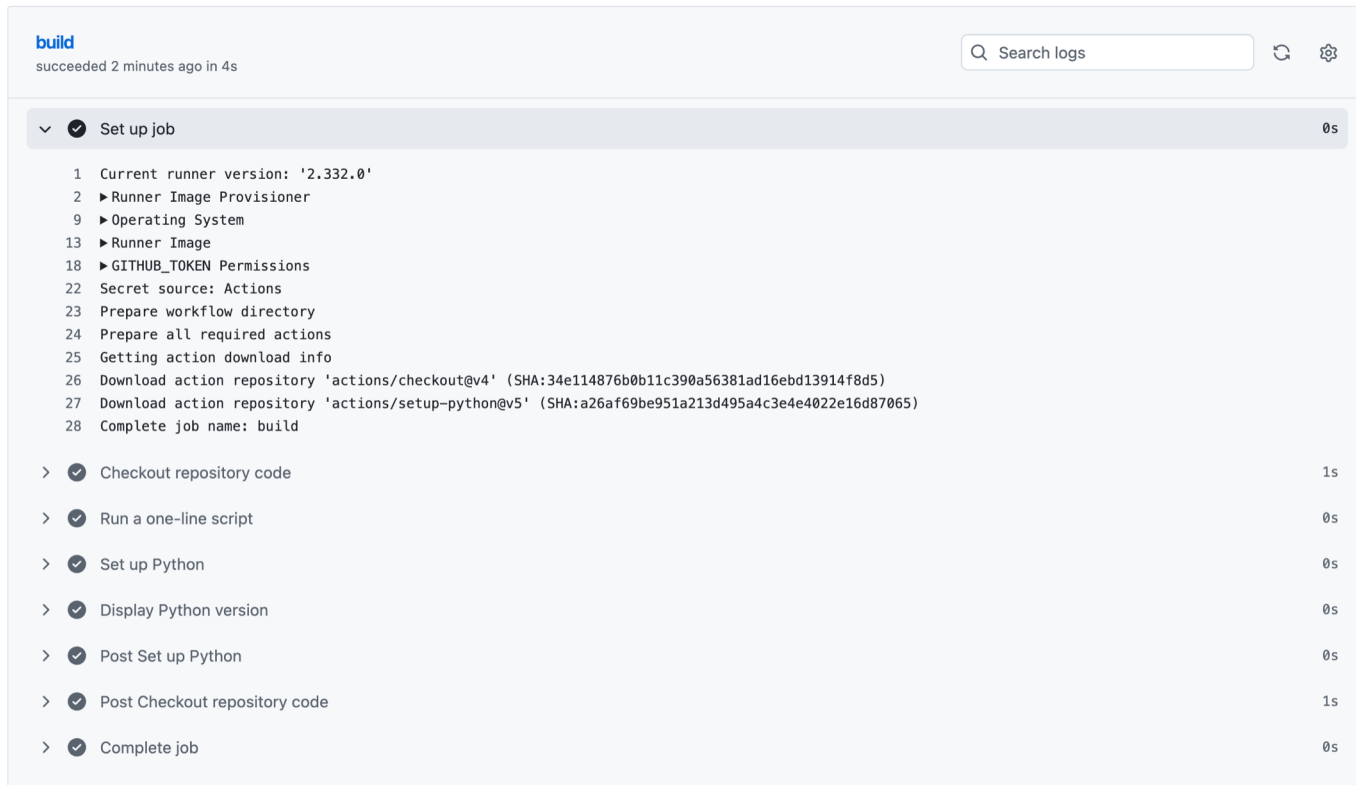
Node.js 20 actions are deprecated. The following actions are running on Node.js 20 and may not work as expected: actions/checkout@v4, actions/setup-python...

[Show more](#)

Git Hub Action

❖ Git Hub Action 샘플

✓ 원격 Repository 확인



The screenshot shows a GitHub Actions workflow run log. At the top, it says "build" in blue, followed by "succeeded 2 minutes ago in 4s". There is a search bar labeled "Search logs" and icons for refresh and settings. The log is divided into sections, each with a dropdown arrow, a checkmark, and a title. The first section, "Set up job", is expanded and shows a list of steps with line numbers. The other sections are collapsed. The steps in the "Set up job" section are: 1. Current runner version: '2.332.0', 2. Runner Image Provisioner, 9. Operating System, 13. Runner Image, 18. GITHUB_TOKEN Permissions, 22. Secret source: Actions, 23. Prepare workflow directory, 24. Prepare all required actions, 25. Getting action download info, 26. Download action repository 'actions/checkout@v4' (SHA: 34e114876b0b11c390a56381ad16ebd13914f8d5), 27. Download action repository 'actions/setup-python@v5' (SHA: a26af69be951a213d495a4c3e4e4022e16d87065), 28. Complete job name: build. The other sections are: Checkout repository code (1s), Run a one-line script (0s), Set up Python (0s), Display Python version (0s), Post Set up Python (0s), Post Checkout repository code (1s), and Complete job (0s).

```
build
succeeded 2 minutes ago in 4s

Search logs

Set up job 0s
  1 Current runner version: '2.332.0'
  2 ▶Runner Image Provisioner
  9 ▶Operating System
 13 ▶Runner Image
 18 ▶GITHUB_TOKEN Permissions
 22 Secret source: Actions
 23 Prepare workflow directory
 24 Prepare all required actions
 25 Getting action download info
 26 Download action repository 'actions/checkout@v4' (SHA:34e114876b0b11c390a56381ad16ebd13914f8d5)
 27 Download action repository 'actions/setup-python@v5' (SHA:a26af69be951a213d495a4c3e4e4022e16d87065)
 28 Complete job name: build

> Checkout repository code 1s
> Run a one-line script 0s
> Set up Python 0s
> Display Python version 0s
> Post Set up Python 0s
> Post Checkout repository code 1s
> Complete job 0s
```

Git Hub Action

❖ node 테스트 프로젝트를 git hub action으로 실행

- ✓ 원격 Repository 생성: nodetest
- ✓ 노드 프로젝트 생성
 - nodetest 디렉토리 생성
 - nodetest 디렉토리로 현재 디렉토리 변경
 - 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

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:

build:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [20.x]

steps:

- uses: actions/checkout@v4

- name: Use Node.js \${{ matrix.node-version }}

uses: actions/setup-node@v4

with:

node-version: \${{ matrix.node-version }}

- run: npm install

- run: npm run build --if-present

- run: npm test

Git Hub Action

- ❖ node 테스트 프로젝트를 git hub action으로 실행
 - ✓ 결과 확인

```
> ✓ Set up job 2s
> ✓ Run actions/checkout@v2 1s
> ✓ Use Node.js 14.x 2s
> ✓ Run npm install 4s
> ✓ Run npm run build --if-present 0s
▼ ✓ Run npm test 0s
  1 ▶ Run npm test
  4
  5 > nodetest@1.0.0 test /home/runner/work/nodetest/nodetest
  6 > mocha test.spec.js
  7
  8
  9
 10   Default Test Set
 11   test1 성공
 12     ✓ test1 should be passed
 13   test2 성공
 14     ✓ test2 should be passed
 15
 16
 17   2 passing (6ms)
 18
```


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 만듦
 - 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 action을 통해 Docker Hub로 배포
 - ✓ Mongo DB 컨테이너 실행
 - `docker run --name mongoddb -v ~/data:/data/db -d -p 27017:27017 mongo`
 - bash shell 접속: `docker exec -it mongoddb bash`
 - ✓ 프로젝트 생성
 - 프로젝트를 위한 디렉토리 생성
 - python 가상 환경 생성 및 활성화
 - ◆ `python -m venv .venv`
 - ◆ `.venv\Scripts\activate` 또는 `source .venv/bin/activate`
 - 패키지 설치
 - ◆ `pip install flask`
 - ◆ `pip install requests`
 - ◆ `pip install beautifulsoup4`
 - ◆ `pip install pymongo`
 - ◆ `pip install pytest`

Git Hub Action

❖ python 프로젝트를 git hub action을 통해 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 action을 통해 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 action을 통해 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 action을 통해 Docker Hub로 배포
 - ✓ 브라우저에서 새로 고침



Flask Web Application

Git Hub Action

❖ python 프로젝트를 git hub action을 통해 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_tag = soup.select_one('title')
```

```
    title = title_tag.text
```

```
    return title
```

Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 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 = get_movie_info(test_url)  
    assert title == "네이버 검색"
```

Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 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 action을 통해 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 action을 통해 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 action을 통해 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 = get_movie_info(url_receive)

doc = {

 'title': title,

 'url': url_receive,

 'comment': comment_receive

}

db.articles.insert_one(doc)

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

Git Hub Action

❖ python 프로젝트를 git hub action을 통해 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 action을 통해 Docker Hub로 배포
 - ✓ 프로젝트 실행 – python app.py
 - ✓ 브라우저에서 localhost:5000 접속
 - ✓ git hub에서 올리기 위해서 패키지를 requirements.txt 로 내보내기
pip list --format=freeze > requirements.txt

Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포
 - ✓ 프로젝트에 .github/workflows 디렉토리를 생성하고 yaml 파일을 생성하고 작성 - learn_github_action.yaml
- ```
name: learn-github-actions

on:
 push:
 branches:
 - main
 pull_request:
 branches:
 - main
```

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포
  - ✓ 프로젝트에 .github/workflows 디렉토리를 생성하고 yaml 파일을 생성하고 작성 - learn\_github\_action.yaml

jobs:

python-hello-world:

runs-on: ubuntu-latest

steps:

- name: Checkout repository  
uses: actions/checkout@v4

- name: Set up Python 3.13  
uses: actions/setup-python@v5  
with:

  - # 현재 가장 최신 안정화 버전인 3.13을 사용합니다.

  - # 특정 패키지 호환성이 중요하다면 3.12를 사용해도 좋습니다.

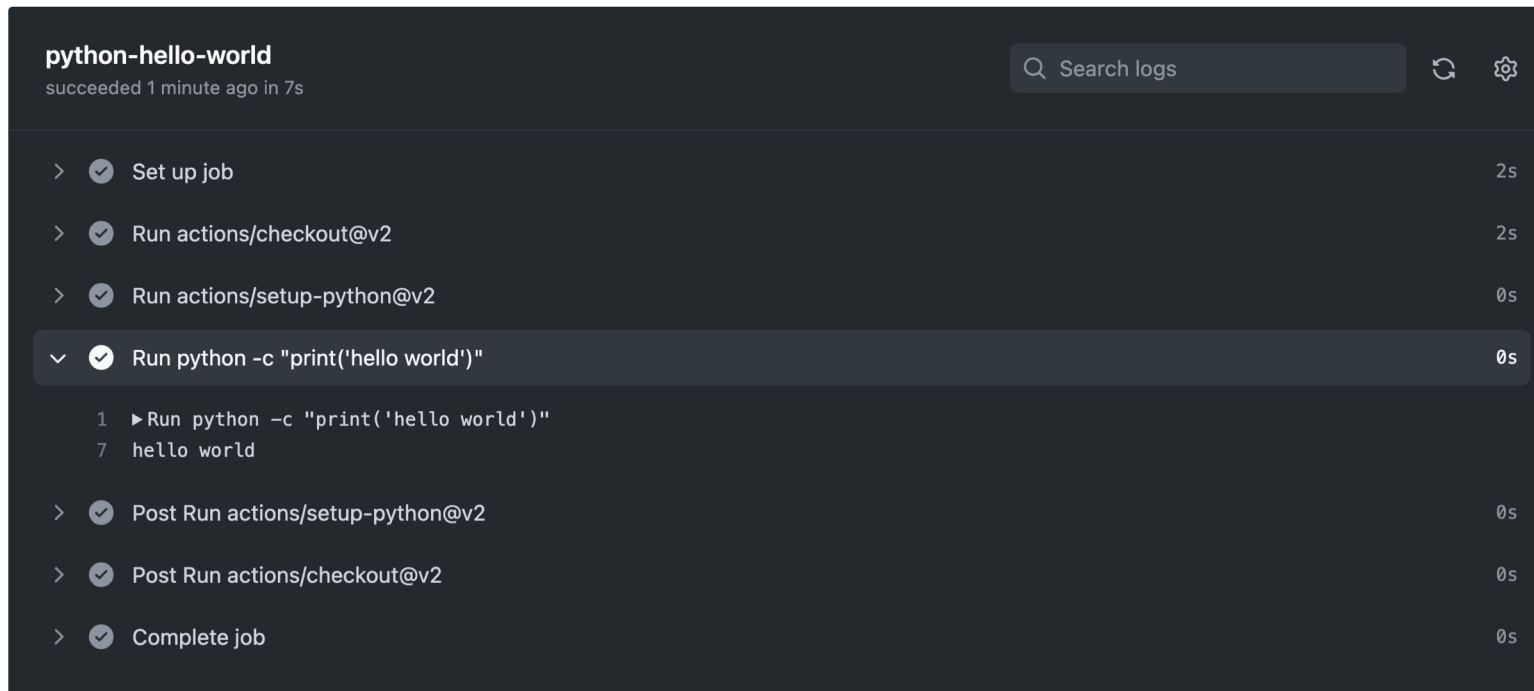
  - python-version: "3.13"

- name: Display Python version  
run: python --version

- name: Execute Python script  
run: python -c "print('hello world from Python 3.13!')"

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포
  - ✓ 원격 레포지토리에 push하고 Action 확인



The screenshot shows a GitHub Actions workflow run for a repository named 'python-hello-world'. The status is 'succeeded 1 minute ago in 7s'. The workflow consists of several steps, all of which are completed successfully. The selected step is 'Run python -c "print('hello world')"', which shows the command being executed and the output 'hello world'.

**python-hello-world**  
succeeded 1 minute ago in 7s

Search logs

- > ✓ Set up job 2s
- > ✓ Run actions/checkout@v2 2s
- > ✓ Run actions/setup-python@v2 0s
- ▼ ✓ Run python -c "print('hello world')" 0s
  - 1 ▶ Run python -c "print('hello world')"
  - 7 hello world
- > ✓ Post Run actions/setup-python@v2 0s
- > ✓ Post Run actions/checkout@v2 0s
- > ✓ Complete job 0s

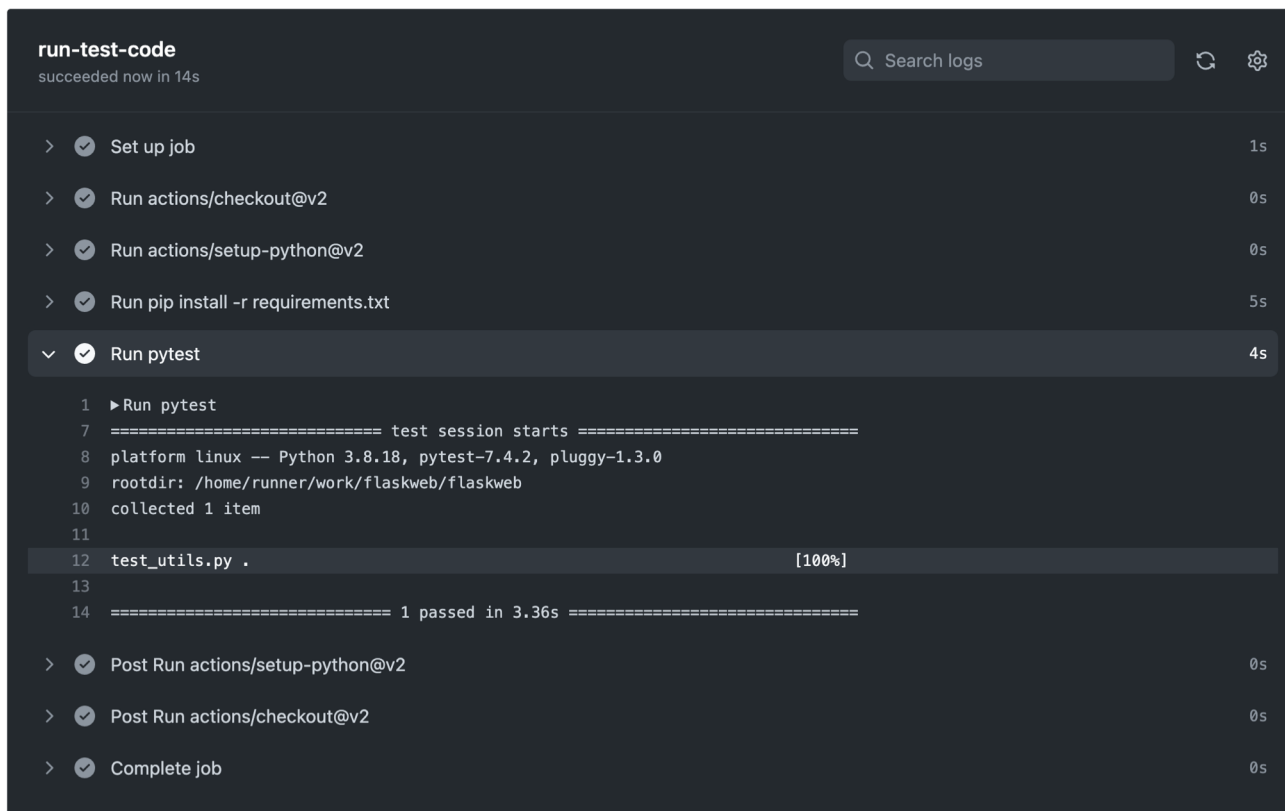
# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 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@v4
 - uses: actions/setup-python@v5
 with:
 python-version: "3.13"
 - run: pip install -r requirements.txt
 - run: pytest
```

# Git Hub Action

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



The screenshot shows a GitHub Actions workflow run titled "run-test-code" which has succeeded in 14 seconds. The log displays a series of steps: "Set up job" (1s), "Run actions/checkout@v2" (0s), "Run actions/setup-python@v2" (0s), "Run pip install -r requirements.txt" (5s), and "Run pytest" (4s). The "Run pytest" step is expanded, showing the execution of the pytest command. The output indicates a successful test session on a Linux platform with Python 3.8.18, pytest 7.4.2, and pluggy 1.3.0. One test item was collected and passed in 3.36 seconds. The workflow concludes with "Post Run actions/setup-python@v2" (0s), "Post Run actions/checkout@v2" (0s), and "Complete job" (0s).

```
run-test-code
succeeded now in 14s

> ✓ Set up job 1s
> ✓ Run actions/checkout@v2 0s
> ✓ Run actions/setup-python@v2 0s
> ✓ Run pip install -r requirements.txt 5s
> ✓ Run pytest 4s
 1 ▶Run pytest
 7 ===== test session starts =====
 8 platform linux -- Python 3.8.18, pytest-7.4.2, pluggy-1.3.0
 9 rootdir: /home/runner/work/flaskweb/flaskweb
 10 collected 1 item
 11
 12 test_utils.py . [100%]
 13
 14 ===== 1 passed in 3.36s =====

> ✓ Post Run actions/setup-python@v2 0s
> ✓ Post Run actions/checkout@v2 0s
> ✓ Complete job 0s
```



# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포

✓ Dockerfile을 추가하고 작성

```
FROM python:3.13-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 action을 통해 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 action을 통해 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@v4
```

```
 - uses: actions/setup-python@v5
```

```
 with:
```

```
 python-version: "3.13"
```

```
 - run: pip install -r requirements.txt
```

```
 - run: pytest
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포

✓ ci-pipeline.yaml 파일을 만들고 작성

build:

needs: [run-test-code]

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.13

uses: actions/setup-python@v5

with:

python-version: '3.13'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포
  - ✓ ci-pipeline.yaml 파일을 만들고 작성
    - name: Login to DockerHub
      - uses: docker/login-action@v1
      - with:
        - username: ggnagpae1
        - password: dckr\_pat\_WgfTukGolj\_48qyr9N43LAHtfD4
    - name: build and release to DockerHub
      - env:
        - NAME: ggnagpae1
        - 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 action을 통해 Docker Hub로 배포

✓ DockerHub에 Repository 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/3ArmHfCi1QtFQorFSImOWx7Ylw0>

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포
  - ✓ Secret 이용
    - 해당 프로젝트의 GitHub Repository로 이동해서 repository를 삭제하고 다시 생성
    - 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: Repository 이름

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포

✓ ci-pipeline.yaml 수정

```
name: ci-pipeline
```

```
on:
```

```
 push:
```

```
 branches:
```

```
 - main
```

```
jobs:
```

```
 run-test-code:
```

```
 runs-on: ubuntu-latest
```

```
 steps:
```

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

```
 - uses: actions/setup-python@v5
```

```
 with:
```

```
 python-version: "3.13"
```

```
 - run: pip install -r requirements.txt
```

```
 - run: pytest
```



# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 Docker Hub로 배포

✓ ci-pipeline.yaml 수정

build:

needs: [run-test-code]

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.13

uses: actions/setup-python@v5

with:

python-version: '3.13'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 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`

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 SSH로 배포
  - ✓ EC2에 Docker 설치
  - ✓ EC2에 Mongo DB 컨테이너 실행
    - `docker run --name mongoddb -v ~/data:/data/db -d -p 27017:27017 mongo`
    - bash shell 접속: `docker exec -it mongoddb bash`
    - Repository에 Secret 설정
    - EC2\_HOST: EC2 인스턴스의 퍼블릭 IP 또는 도메인
    - EC2\_USERNAME: 접속 계정 (ubuntu)
    - EC2\_SSH\_KEY: EC2 접속을 위한 .pem 키 페어 내용 전체

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 배포

✓ ci-pipeline.yaml 수정

```
name: ci-pipeline
```

```
on:
```

```
 push:
```

```
 branches:
```

```
 - main
```

```
jobs:
```

```
 run-test-code:
```

```
 runs-on: ubuntu-latest
```

```
 steps:
```

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

```
 - uses: actions/setup-python@v5
```

```
 with:
```

```
 python-version: "3.13"
```

```
 - run: pip install -r requirements.txt
```

```
 - run: pytest
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 배포

✓ ci-pipeline.yaml 수정

build:

needs: [run-test-code]

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.13

uses: actions/setup-python@v5

with:

python-version: '3.13'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 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
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 배포

✓ ci-pipeline.yaml 수정

deploy:

needs: build

runs-on: ubuntu-latest

steps:

- name: Deploy to EC2 via SSH

uses: appleboy/ssh-action@v1.0.3

with:

host: \${ secrets.EC2\_HOST }

username: \${ secrets.EC2\_USERNAME }

key: \${ secrets.EC2\_SSH\_KEY }

# 1. 사용할 환경 변수들을 등록합니다.

envs: DOCKER\_USERNAME, DOCKER\_REPO, DOCKER\_TOKEN

debug: true # <--- 이 설정을 추가해서 상세 로그를 확인!

timeout: 10s # 접속 시도 타임아웃 단축

command\_timeout: 1m

request\_pty: true

use\_insecure\_cipher: true

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 배포

✓ ci-pipeline.yaml 수정

```
script: |
 echo "Successfully Connected!"
```

```
1. 변수 조합 확인 (디버깅 로그) 및 로그인
```

```
echo "Starting deployment for: $DOCKER_USERNAME/$DOCKER_REPO"
```

```
echo "${DOCKER_TOKEN}" | sudo docker login -u "${DOCKER_USERNAME}" --
password-stdin
```

```
2. 최신 이미지 가져오기
```

```
sudo docker pull $DOCKER_USERNAME/$DOCKER_REPO:latest
```

```
3. 기존 컨테이너 안전하게 중지 및 삭제
```

# 2>/dev/null은 에러 메시지를 숨기고, || true는 실패해도 다음 단계를 진행하게 합니다.

```
sudo docker stop my-app-container 2>/dev/null || true
```

```
sudo docker rm my-app-container 2>/dev/null || true
```



# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 SSH로 배포

✓ ci-pipeline.yaml 수정

# 4. 새 컨테이너 실행

# 5000번 포트로 돌아가는 Flask/Node 앱을 80포트로 연결하는 예시입니다.

```
sudo docker run -d ₩
```

```
--name my-app-container ₩
```

```
-p 80:5000 ₩
```

```
--restart always ₩
```

```
$DOCKER_USERNAME/$DOCKER_REPO:latest
```

# 5. 불필요한 이미지(Dangling images) 정리

```
sudo docker image prune -f
```

# 0. GitHub Secrets를 SSH 환경 변수로 매핑합니다.

env:

```
DOCKER_USERNAME: ${ secrets.DOCKER_USERNAME }
```

```
DOCKER_REPO: ${ secrets.DOCKER_REPO }
```

```
DOCKER_TOKEN: ${ secrets.DOCKERHUB_TOKEN }
```

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포
  - ✓ GitHub Secrets 설정
    - KUBECONFIG에 클러스터 접속 정보 (~/.kube/config 내용 전체)
    - 접속 IP는 외부에서 접속 가능한 IP로 수정

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

- deployment.yml

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
 name: flaskweb-deployment
```

```
 labels:
```

```
 app: flaskweb
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

- deployment.yml

spec:

replicas: 3

selector:

matchLabels:

app: flaskweb

template:

metadata:

labels:

app: flaskweb

spec:

containers:

- name: flaskweb-container

image: ggnagpae1/flaskweb:latest

ports:

- containerPort: 5000

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

- service.yml

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
 name: flaskweb-service
```

```
spec:
```

```
 type: NodePort
```

```
 selector:
```

```
 app: flaskweb # Deployment의 labels와 일치해야 합니다.
```

```
 ports:
```

```
 - protocol: TCP
```

```
 port: 80 # 서비스의 가상 포트 (클러스터 내부 통신용)
```

```
 targetPort: 5000 # Pod(컨테이너) 내부에서 실제로 열려 있는 포트
```

```
 nodePort: 30008 # 외부에서 접속할 때 사용하는 포트 (30000-32767 사이 지정)
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

- action 파일 수정

```
name: ci-pipeline
```

```
on:
```

```
 push:
```

```
 branches:
```

```
 - main
```

```
jobs:
```

```
 run-test-code:
```

```
 runs-on: ubuntu-latest
```

```
 steps:
```

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

```
 - uses: actions/setup-python@v5
```

```
 with:
```

```
 python-version: "3.13"
```

```
 - run: pip install -r requirements.txt
```

```
 - run: pytest
```

# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

- action 파일 수정

build:

needs: [run-test-code]

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.13

uses: actions/setup-python@v5

with:

python-version: '3.13'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

# Git Hub Action

- ❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포
  - ✓ 배포를 위한 매니페스트 생성
    - action 파일 수정
      - 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: \${{ secrets.DOCKERHUB\_USERNAME }}  
password: \${{ secrets.DOCKERHUB\_TOKEN }}



# Git Hub Action

❖ python 프로젝트를 git hub action을 통해 kubernetes cluster에 배포

✓ 배포를 위한 매니페스트 생성

● action 파일 수정

```
- name: build and release to DockerHub
 run: |
 IMAGE_TAG=${{ github.sha }}
 docker build -t
 ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_REPO }}:$IMAGE_TAG .
 docker push
 ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_REPO }}:$IMAGE_TAG
- name: Kubernetes 컨텍스트 설정
 uses: azure/k8s-set-context@v3
 with:
 method: kubeconfig
 kubeconfig: ${{ secrets.KUBECONFIG }}
- name: 클러스터 배포
 run: |
 # 1. 태그 변경
 sed -i "s|image:.*/image:
 ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.DOCKERHUB_REPO }}:${{ github.sha }}"
 g" deployment.yml

 # 2. --insecure-skip-tls-verify=true 옵션 추가 (인증서 무시)
 # 3. --validate=false 옵션 추가 (OpenAPI 다운로드 검증 무시)
 kubectl apply -f deployment.yml --insecure-skip-tls-verify=true --validate=false
 kubectl apply -f service.yml --insecure-skip-tls-verify=true --validate=false
```