



# 생산성을 높여주는 CI/CD With Django, GitHub Actions

김 승 주

# 목차

1. CI/CD 를 프로젝트에 접목하기까지의 저의 과정
2. 왜 실무에서 CI/CD를 쓰는가?
3. Django기반 GitHub Actions를 통한 CI/CD 구축절차

# 학부생 시절

Pulling

☒ 출력 전부 보기

```
git -c diff.mnemonicprefix=false -c core.quotepath=false --no-optional-locks fetch origin
git: 'credential-aws' is not a git command. See 'git --help'.
From https://git.gsitm.com/
14be187..a07e329 develop -> origin/develop
```

```
git -c diff.mnemonicprefix=false -c core.quotepath=false --no-optional-locks pull origin develop
git: 'credential-aws' is not a git command. See 'git --help'.
From https://git.gsitm.com/
* branch          develop -> FETCH_HEAD
```

```
error: Your local changes to the following files would be overwritten by merge:
src/main/java/com/.../system/web/...Controller.java
Please commit your changes or stash them before you merge.
Aborting
```

Updating 14be187..a07e329  
오류가 나면서 완료됨.

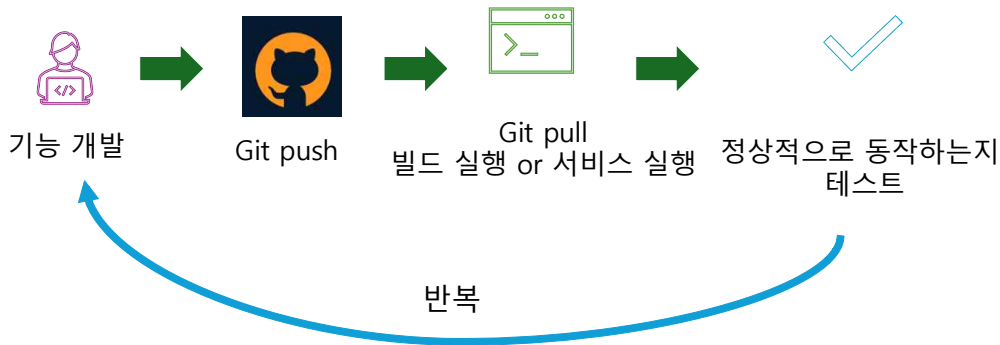
```
$ git pull origin master
From github.com:hodory/source.hodory.github.io
* branch          master -> FETCH_HEAD
error: Your local changes to the following files would be overwritten by merge:
themes/icarus/layout/widget/recent_posts.ejs
Please commit your changes or stash them before you merge.
```

## git reset --hard

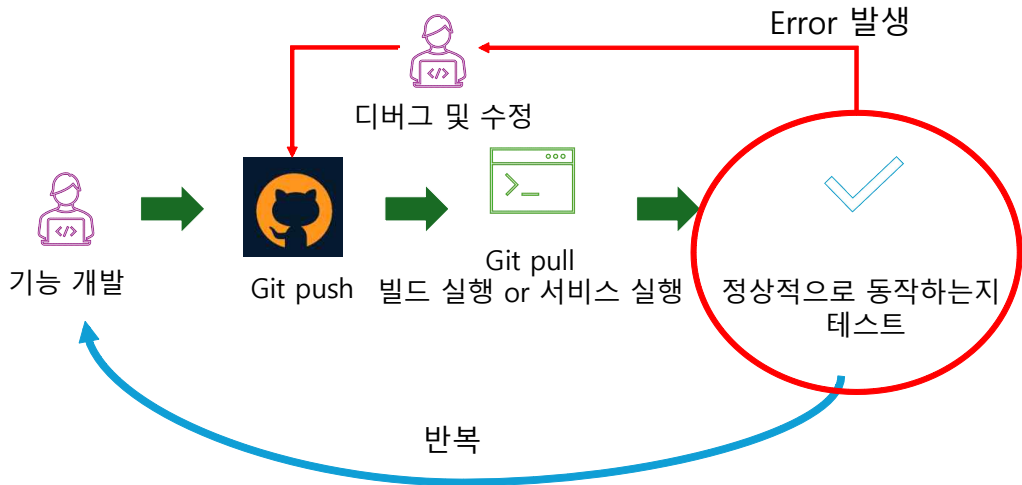


## git에 관련해서는 내가 다 할게... TTT

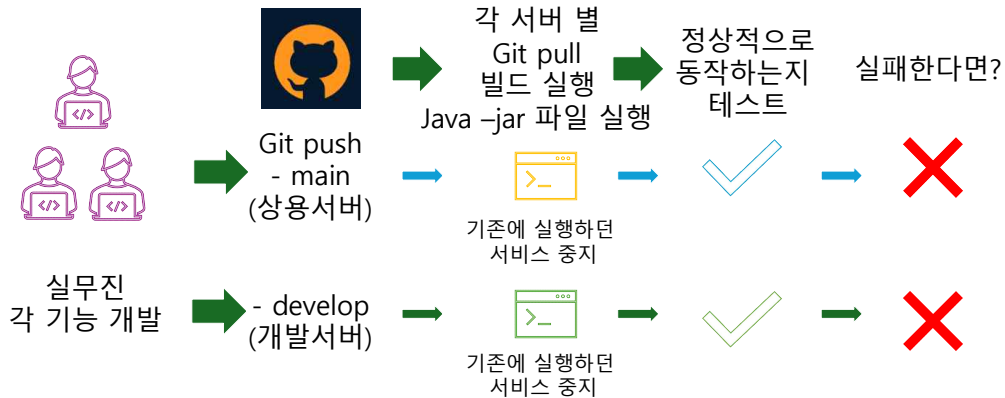
# 사회초년생 (신입 개발자 시절)



# 사회초년생 (신입 개발자 시절)



# 예시 시나리오



**시간적 측면, 생산적 측면**

**INPUT 대비 OUTPUT ▼**

앞서 나온 문제들을 효율적으로  
해결할 수 있는 방법이 있을까?

**CI(Continuous Integration)**  
지속적 통합

**CD(Continuous Deployment)**  
지속적 배포

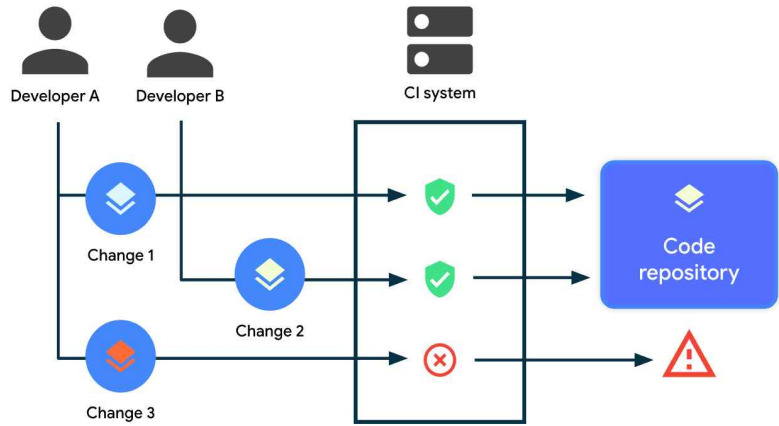


왜 필요한가?

**효율성**

**생산성**

# CI(Continuous Integration)



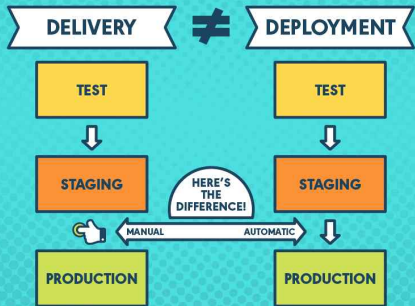
## CI(Continuous Integration)

### 지속적 통합

“새로운 코드 변경 사항이 정기적으로 빌드 및 테스트 되어  
공유 레포지토리에 통합하는 과정을 의미”

# CD(Continuous Delivery & Deployment)

## CONTINUOUS DELIVERY VS CONTINUOUS DEPLOYMENT



gocd.org

## Continuous Delivery

코드 변경 사항들이 자동화된 테스트와 배포 준비 과정을 거쳐 언제든지 신뢰할 수 있는 방식으로 “배포할 준비”가 되어 있는 상태

## Continuous Deployment

코드 변경 사항이 테스트를 성공적으로 통과하면 자동으로 생산 환경에 배포되는 과정입니다.  
이 접근 방식에서는 “수동 배포 과정이 없어”, 변경 사항이 빠르게 서비스(애플리케이션)에 적용됩니다.

## 다양한 CI/CD 툴



**Jenkins**



GitHub Actions



**Travis CI**

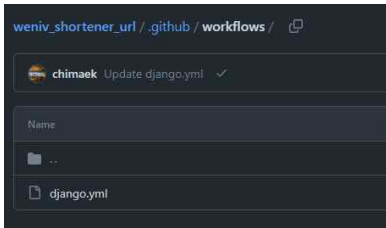
# GitHub Actions 특징 (장점)



## 통합된 환경

GitHub에서 코드를 호스팅하면서 동시에 CI/CD 파이프라인을  
구축할 수 있는 통합 환경을 제공

# GitHub Actions 특징 (장점)

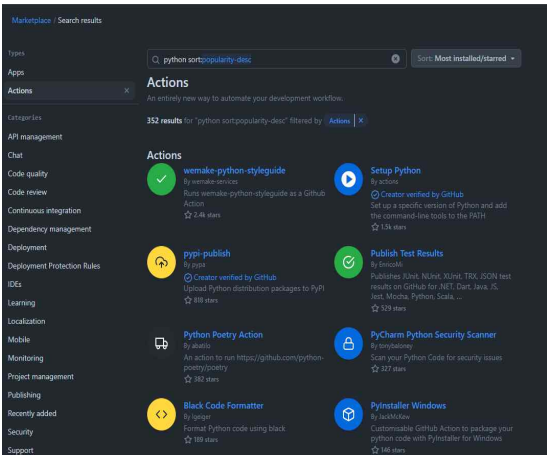


간편한 접근성 및 관리

워크플로우 파일을 손쉽게 추가하고 수정하여 CI/CD를 구축 할 수 있습니다.

```
1  name: Django CI
2
3  on:
4    push:
5      branches: [ "master" ]
6    pull_request:
7      branches: [ "master" ]
8
9  jobs:
10    build:
11
12      runs-on: ubuntu-latest
13      strategy:
14        max-parallel: 4
15        matrix:
16          python-version: [3.11]
17
18      steps:
19        - uses: actions/checkout@v3
20        - name: Set up Python ${ matrix.python-version }
21          uses: actions/setup-python@v3
22          with:
23            python-version: ${ matrix.python-version }
24        - name: Install Dependencies
25          run: |
26            python -m pip install --upgrade pip
27            pip install -r requirements.txt
28        - name: 서버 배포
29          uses: appleboy/ssh-action@master
```

# GitHub Actions 특징 (장점)



## 마켓플레이스를 통한 확장성

GitHub Actions 마켓플레이스에서 호스팅 환경, 데이터베이스, 배포 도구 등 다양한 서비스와 통합할 수 있는 준비된 액션들을 제공합니다.

이러한 액션들을 통해 복잡한 환경에서의 배포와 통합이 용이해집니다.



# GitHub Actions 특징 (단점)



**GitHub에 종속**

GitHub Actions는 GitHub 플랫폼에 종속적입니다. 결국 GitHub 외의 다른 코드 호스팅 서비스(예: Bitbucket)를 사용하는 경우, GitHub Actions를 직접 사용할 수 없다는 것을 의미합니다.

# CI/CD 구축을 위한 기술 스택

1. Django(WSGI)



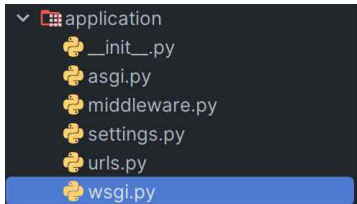
2. Gunicorn



3. Nginx

4. Server(Ubuntu) [Linux]

# WSGI (Web Server Gateway Interface)



settings.py

```
WSGI_APPLICATION = 'application.wsgi.application'
```

wsgi.py

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'application.settings')  
  
application = get_wsgi_application()
```

## WSGI (Web Server Gateway

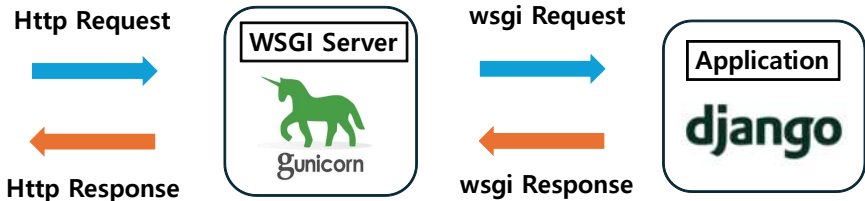
## Interface)

웹 서버와 웹 애플리케이션(또는 프레임워크)간의 표준 인터페이스를 정의합니다.

Python기반 웹 개발에서 자주 사용되며, 동기 방식의 처리를 기본으로 합니다.

즉, Django 애플리케이션을 웹 서버에 연결하는 방식을 정의하는 것입니다.

# Gunicorn(Green Unicorn)



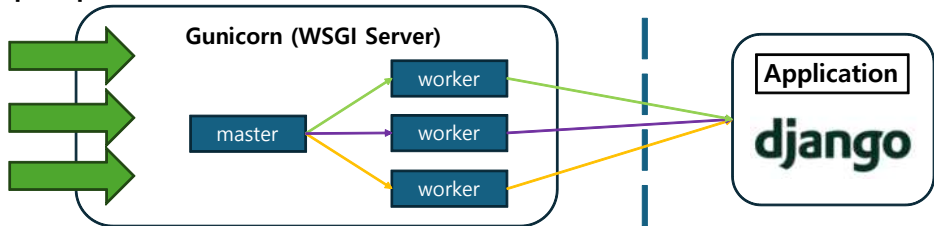
Gunicorn은 Python 기반의 애플리케이션을 위한 WSGI(Web Server Gateway Interface) 웹 서버입니다.

웹 브라우저나 다른 클라이언트의 HTTP 요청을 받아, 해당 요청을 Python 웹 애플리케이션에 전달하고, 애플리케이션의 응답을 클라이언트에 다시 전송합니다.

이 과정에서 Gunicorn은 WSGI 표준을 사용하여 웹 애플리케이션과의 통신을 처리합니다.

# Gunicorn(Green Unicorn) 특징

Http Request



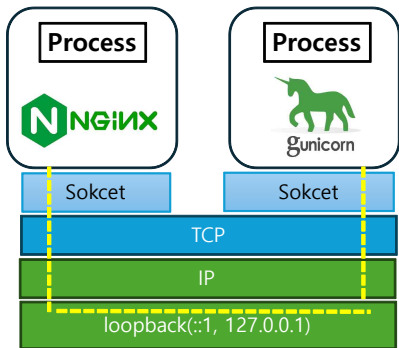
여러 개의 Worker Process를 사용하여 동시에 여러 HTTP 요청을 처리할 수 있습니다.

# Nginx

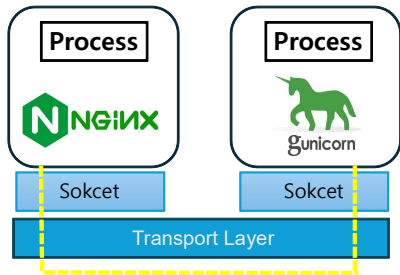


- 웹 서버
- 리버스 프록시 서버 : 부하 분산
- 정적 데이터 처리 : 빠른 속도로 정적데이터 서빙

# Nginx – Gunicorn 통신 With Unix Socket



TCP/IP 네트워크 방식



Unix 소켓 방식

유닉스 소켓은 TCP/UDP를 통해 Transport Layer에서 통신하므로 데이터 전달에 있어 효율적입니다.

아이피와 포트로 바인딩하면 HTTP를 통해 프로세스 간 Application Layer에서 통신하므로

내부에서 필요없는 데이터까지 전달하기 때문입니다.

# Gunicorn 설정 (소켓 미사용)

```
pip install gunicorn # 구니콘 설치
```

```
cd /home/{사용자명}/{프로젝트 루트 디렉터리}
```

```
gunicorn --bind 0:5000 {프로젝트이름}.wsgi:application
```

```
"""
```

5000번 포트로 WSGI 서버(gunicorn)와 연결된 WSGI 애플리케이션(django)을 실행시킨다.

```
# nginx
```

```
[2023-11-28 10:56:09 +0900] [52669] [INFO] Starting gunicorn 21.2.0
```

```
[2023-11-28 10:56:09 +0900] [52669] [INFO] Listening at: http://0.0.0.0:5000 (52669)
```

```
[2023-11-28 10:56:09 +0900] [52669] [INFO] Using worker: sync
```

```
[2023-11-28 10:56:09 +0900] [52670] [INFO] Booting worker with pid: 52670
```

이렇게 나오면 WSGI 서버와 애플리케이션이 실행된것입니다.

```
"""
```



# Gunicorn 설정 (소켓 사용)

```
cd /home/{사용자명}/{프로젝트 루트 디렉터리}
```

```
gunicorn --bind unix:/tmp/gunicorn.sock {프로젝트명}.wsgi:application
```

```
"""
```

```
[2023-11-28 11:56:09 +0900] [52669] [INFO] Starting gunicorn 21.2.0
```

```
[2023-11-28 11:56:09 +0900] [52669] [INFO] Listening at: unix:/tmp/gunicorn.sock (52669)
```

```
[2023-11-28 11:56:09 +0900] [52669] [INFO] Using worker: sync
```

```
[2023-11-28 11:56:09 +0900] [52670] [INFO] Booting worker with pid: 52670
```

```
"""
```

# Gunicorn 서비스 등록

`sudo vim /etc/systemd/system/{프로젝트명}.service`

```
### {프로젝트명}.service

[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=ubuntu # 유저이름 확인해주세요.
Group=ubuntu
WorkingDirectory=/home/ubuntu/{프로젝트 디렉토리} # 프로젝트 루트 디렉토리
EnvironmentFile=/home/ubuntu/{프로젝트 디렉토리}/.env # 환경변수 파일
ExecStart=/home/ubuntu/{프로젝트 디렉토리}/venv/bin/gunicorn \ #가상환경에 설치된 gunicorn
    --workers 2 \ #워커 갯수
    --bind unix:/tmp/gunicorn.sock \ # WSGI실행 명령
    {프로젝트명}.wsgi:application # WSGI(Django) 애플리케이션

[Install]
WantedBy=multi-user.target
```

# Gunicorn 서비스 등록

1. `sudo systemctl start` {프로젝트 서비스 명}
2. `sudo systemctl status` {서비스 파일명} (.service 빼고입니다.)
3. `sudo systemctl enable` {서비스 파일명} (.service 빼고입니다.)

```
● makere.service - gunicorn daemon
   Loaded: loaded (/etc/systemd/system/makere.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-11-27 14:06:03 KST; 21h ago
     Main PID: 45206 (gunicorn)
        Tasks: 3 (limit: 2287)
      Memory: 97.7M
         CPU: 12.097s
    CGroup: /system.slice/makere.service
            └─45206 /home/ubuntu/app/make_re_be/venv/bin/python3 /home/ubuntu/app/make_re_be/venv/bin/gunicorn --workers 2 --bind u
               └─45208 /home/ubuntu/app/make_re_be/venv/bin/python3 /home/ubuntu/app/make_re_be/venv/bin/gunicorn --workers 2 --bind u
                  └─45209 /home/ubuntu/app/make_re_be/venv/bin/python3 /home/ubuntu/app/make_re_be/venv/bin/gunicorn --workers 2 --bind u

Nov 27 14:06:03 ip-172-26-11-127 systemd[1]: Started gunicorn daemon.
Nov 27 14:06:03 ip-172-26-11-127 gunicorn[45206]: [2023-11-27 14:06:03 +0900] [45206] [INFO] Starting gunicorn 21.2.0
Nov 27 14:06:03 ip-172-26-11-127 gunicorn[45206]: [2023-11-27 14:06:03 +0900] [45206] [INFO] Listening at: unix:/tmp/gunicorn.sock (
Nov 27 14:06:03 ip-172-26-11-127 gunicorn[45206]: [2023-11-27 14:06:03 +0900] [45206] [INFO] Using worker: sync
Nov 27 14:06:03 ip-172-26-11-127 gunicorn[45208]: [2023-11-27 14:06:03 +0900] [45208] [INFO] Booting worker with pid: 45208
Nov 27 14:06:03 ip-172-26-11-127 gunicorn[45209]: [2023-11-27 14:06:03 +0900] [45209] [INFO] Booting worker with pid: 45209
ip-172-26-11-127 (END)
```

# Nginx 설정

```
sudo vim /etc/nginx/sites-enabled/default # nginx 설정파일 편집 실행
```

```
server{  
    root /var/www/html;  
    server_name # 도메인 주소 없으면 IPV4 주소  
  
    location /api/v1/{ # 클라이언트에서 오는 /api/v1/ 으로 시작되는 모든 요청 핸들링  
        include proxy_params;  
        proxy_pass http://unix:/tmp/gunicorn.sock;  
    }  
  
}
```

- proxy\_pass : Nginx가 프론트의 요청을 전달해야 하는 백엔드 서버의 주소를 지정하는 명령어

# Git Actions Workflow 설정

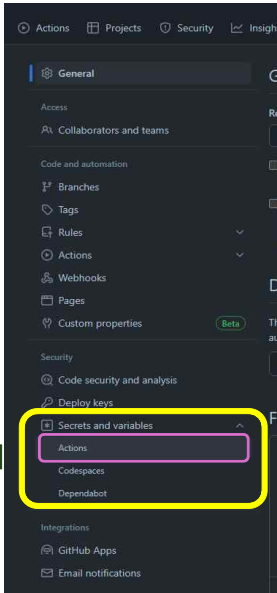
1. Repository

2. Setting

3. 왼쪽 사이드바 Secrets and variables

4. Actions 클릭

5. Repository secrets -> new repository secret



Repository secrets

New repository secret

Name ↕↑

Last updated

# GitHub Actions Secret Key 추가

Actions secrets / New secret











Name \*

YOUR\_SECRET\_NAME 키

Secret \*

값

Add secret

Name ↕	Last updated
DEBUG	4 months ago  
SECRET_KEY	4 months ago  
SERVER_HOST	4 months ago  
SERVER_PASSWORD	4 months ago  
SERVER_USER	4 months ago  

값은 수정할 수 있으나 이전 값은 보지 못합니다.

# GitHub Actions workflows

> Code Issues Pull requests **Actions** Security Insights Settings

## Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself →](#)

### Suggested for this repository

#### Jekyll using Docker image

By GitHub Actions

Package a Jekyll site using the jekyll/builder Docker image.

Configure

HTML



#### Deno

By GitHub Actions

Test your Deno project

Configure

JavaScript



#### Django

By GitHub Actions

Build and Test a Django Project

Configure

Python



#### Laravel

By GitHub Actions

Test a Laravel project.

Configure

PHP



#### Grunt

By GitHub Actions

Build a NodeJS project with npm and grunt.

Configure

JavaScript



#### Gulp

By GitHub Actions

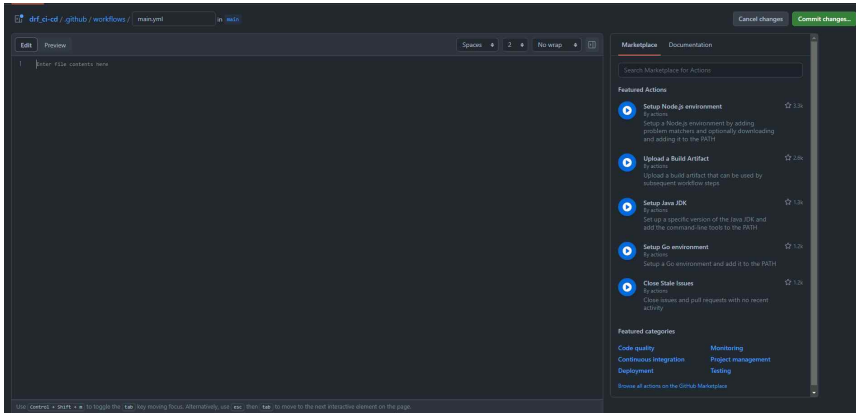
Build a NodeJS project with npm and gulp.

Configure

JavaScript



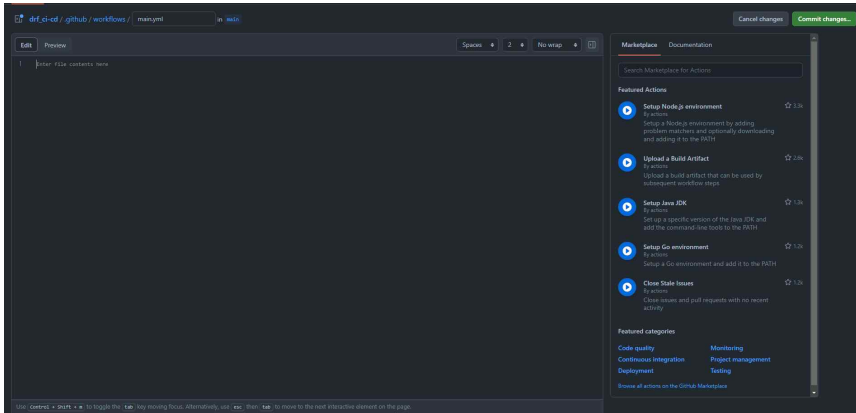
# GitHub Actions File



프로젝트 루트 -> .github -> .workflows 아래 존재



# GitHub Actions File



프로젝트 루트 -> .github -> .workflows 아래 존재

# GitHub Actions CI/CD

```
on:  
  push:  
    branches: [ "main" ]  
  pull_request:  
    branches: [ "main" ]
```

메인 브랜치에 코드가 push또는 pull\_request 가 발생했을 때  
해당 파일이 수행됩니다.

# GitHub Actions CI/CD

```
jobs:
  deploy:
    runs-on: ubuntu-latest
```

- **jobs:**
  - 이 구문은 워크플로우에서 실행할 작업들을 정의하는 부분입니다. 여기서는 `deploy` 라는 하나의 작업만을 정의하고 있습니다.
- **deploy:**
  - 이 부분은 `deploy` 라는 이름의 작업을 설정합니다. 이 이름은 워크플로우 내에서 이 작업을 식별하는 데 사용됩니다.
- **runs-on:**
  - 이 지시어는 작업이 실행될 가상 환경(또는 러너)의 유형을 지정합니다. GitHub Actions는 다양한 운영 체제에서 작업을 실행할 수 있도록 여러 러너를 제공합니다.
- **ubuntu-latest**
  - 이 부분은 가장 최신 버전의 Ubuntu 운영 체제를 사용하겠다는 것을 의미합니다. `ubuntu-latest` 는 GitHub Actions에서 제공하는 가상 환경 중 하나로, 이 환경에서는 Ubuntu의 최신 안정 버전이 실행됩니다.
  - 이 설정은 작업이 Ubuntu 리눅스 환경에서 실행되어야 할 때 유용하며, 특히 리눅스 기반의 응용 프로그램을 빌드하고 테스트하는 데 적합합니다.

# GitHub Actions CI/CD

```
steps:  
  - name: 체크아웃 레포지토리  
    uses: actions/checkout@v3
```

steps 는 이 워크플로우가 진행되는 각 단계(action)들을 의미합니다.

actions/checkout@v3 라는 액션을 사용하여 레포지토리의 코드를 체크아웃하고, 프로젝트 코드를 복사하는 역할을 합니다.

# GitHub Actions CI/CD

```
- name: 파이썬 설정  
  uses: actions/setup-python@v3  
  with:  
    python-version: '3.11'
```

actions/setup-python@v3를 사용하여 파이썬 환경을 설정합니다.

python-version: '3.11'로 지정하여 파이썬 3.11 버전을 사용합니다.

이 단계는 필요한 파이썬 버전을 설치하고 설정하는 역할을 합니다.

# GitHub Actions CI/CD

```
- name: 의존성 설치  
  run: |  
    pip install --upgrade pip  
    pip install -r requirements.txt
```

pip를 사용하여 애플리케이션에 필요한 의존성을 설치합니다.

여기서 requirements.txt 파일에 명시된 모든 패키지가 설치됩니다.

이 단계는 프로젝트가 필요로 하는 모든 외부 라이브러리를 설치하여  
프로젝트가 올바르게 실행될 수 있도록 준비합니다.

# GitHub Actions CI/CD

```
- name: 서버 배포
  uses: appleboy/ssh-action@master
  with:
    host: ${ secrets.SERVER_HOST }
    username: ${ secrets.SERVER_USER }
    password: ${ secrets.SERVER_PASSWORD }
```

appleboy/ssh-action@master를 사용하여 서버에 SSH 접속합니다.

host, username, password는 GitHub Secrets를 통해 안전하게 관리됩니다.

# GitHub Actions CI/CD

```
script: |  
  set -e  
  cd /home/ubuntu/{레포지토리주소}  
  git pull origin main
```

set -e 스크립트 중 어느 부분에서든 명령어가 실패하면 즉시 전체 스크립트가 중단되고, 해당 GitHub Actions 작업이 실패 상태로 표시됩니다.

스크립트를 통해 서버에 있는 프로젝트 디렉토리로 이동한 다음,  
최신 코드를 git pull로 가져옵니다.



# GitHub Actions CI/CD

```
echo "SECRET_KEY=\"${{ secrets.SECRET_KEY }}\"" > .env  
echo "DEBUG='${{ secrets.DEBUG }}'" >> .env  
  
source venv/bin/activate  
pip install -r requirements.txt  
sudo systemctl restart {아까등록한서비스이름}.service
```

.env 파일을 생성하거나 수정하여 환경 변수를 설정합니다.

가상 환경을 활성화하고 필요한 패키지를 설치한 후, {구니콘 서비스이름}.service  
(또는 여러분이 등록한 서비스 이름)를 재 시작하여 변경사항을 반영합니다.

프로젝트의 최신 버전을 서버에 배포하고 필요한 설정을 적용하는 역할을 합니다.

# GitHub Actions Task

**deploy**  
succeeded now in 37s

- > Set up job
- > Build appleboy/ssh-action@master
- > 체크아웃 레포지토리
- > 파이썬 설정
- > 의존성 설치
- > 서버 배포
- > Post 파이썬 설정
- > Post 체크아웃 레포지토리
- > Complete job

수행 단계

	<b>:sparkles: ci/cd 수정</b> Django CI/CD #58: Commit <a href="#">10e2cf9</a> pushed by chimaeak	master	3 hours ago 1m 50s	...
	<b>:sparkles: ci/cd 수정</b> Django CI/CD #57: Commit <a href="#">ac512cd</a> pushed by chimaeak	master	3 hours ago 1m 39s	...

성공 케이스

	<b>:sparkles: ci/cd 수정</b> Django CI/CD #56: Commit <a href="#">18d8373</a> pushed by chimaeak	master	3 hours ago 1m 49s	...
	<b>:sparkles: ci/cd 수정</b> Django CI/CD #55: Commit <a href="#">f8184a0</a> pushed by chimaeak	master	3 hours ago 1m 45s	...
	<b>:sparkles: ci/cd 수정</b> Django CI/CD #54: Commit <a href="#">c0d21b1</a> pushed by chimaeak	master	3 hours ago 1m 45s	...

실패 케이스

# 감사합니다!

깃허브 주소

[https://github.com/chimaek/pyweb\\_ci\\_cd\\_2024](https://github.com/chimaek/pyweb_ci_cd_2024)

