

(예진) 인프라 환경 공부 - Django AWS deployment (Talha Anwar)

☑ Done	<input type="checkbox"/>
🔍 Area	
📅 Do Date	@2023년 7월 8일
🔍 Goal	
➤ Projects	
🔗 URL	
🕒 When	

To Do

(part 1. 장고 프로젝트 생성) [Deploy Django website on AWS EC2 server with a custom domain](#)
(part 2. Gunicorn을 사용해 django 서버 제공) [Run Django server using gunicorn on docker](#)
(part 3. Nginx와 docker로 서버 배포) [Django deployment using Nginx and docker compose](#)
(part 4. AWS RDS의 PostgreSQL로 django의 데이터베이스 변경) [Connect Django with AWS RDS PostgreSQL](#)
(part 5. AWS EC2 서버로 웹사이트 배포) [Deploy Django on EC2 using Docker compose and GIT](#)
(part 6. route53으로 도메인 이름 등록) [Connect domain and SSL to Django app on EC2 using Route53 and ELB](#)

▼ To Do

1. Django web
2. Gunicorn
3. Docker container
4. Nginx
5. Docker Compose
6. AWS Postgres
7. AWS EC2 server
8. AWS Route 53
9. AWS Certificate manager
10. AWS Load balancer

▼ (part 1. 장고 프로젝트 생성) Deploy Django website on AWS EC2 server with a custom domain

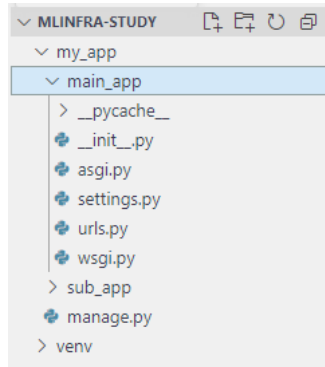
1. 가상 환경 생성 및 활성화

```
python3 -m venv venv # 가상 환경 생성
source venv/bin/activate # 가상 환경 활성화
```

2. 장고 설치 및 프로젝트 생성

```
pip install django    # 장고 설치
django-admin startproject main_app    # 장고 프로젝트 생성
```

3. main_app의 상위 디렉토리 이름을 my_app으로 바꾸자~

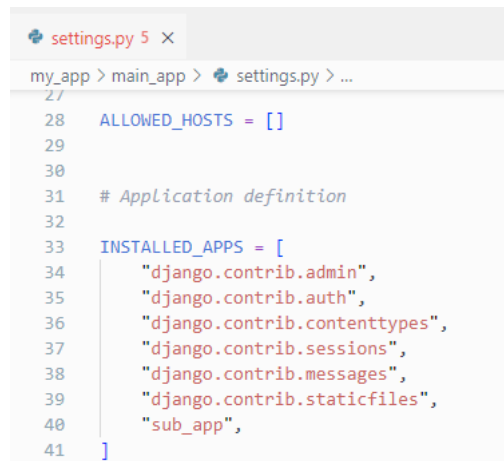


4. 하위 app 생성

```
cd my_app    # manage.py가 있는 디렉토리로 이동
python manage.py startapp sub_app    # sub_app이라는 하위 app 생성
```

5. 하위 app(sub_app)을 기본 앱(main_app)의 설정 앱으로 명시해야 함.

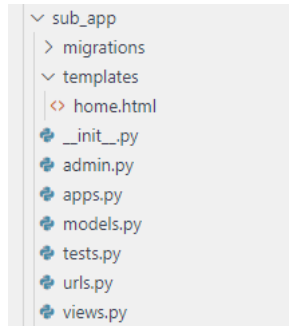
→ main_app의 settings.py 에서 INSTALLED_APPS에 추가하기



6. sub_app에 urls.py 파일 추가하기

→ main_app의 url과 연결하기 위함

7. sub_app에 templates 폴더 추가, templates 폴더에 home.html 파일 추가하기



8. url과 home.html 연결하기

→ sub_app의 views.py에서 home.html로 연결하는 함수 생성

```
my_app > sub_app > views.py > ...
1  from django.shortcuts import render
2
3
4  Ask EasyCode | Explain | Refactor
5  # Create your views here.
6  def home(request):
7      return render(request, "home.html", {})
```

→ sub_app의 urls.py 작성

```
my_app > sub_app > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.home, name='home')
6  ]
7
```

→ main_app의 urls.py에 sub_app.urls 포함시켜줌

```
my_app > main_app > urls.py > ...
17  from django.contrib import admin
18  from django.urls import path, include
19
20  urlpatterns = [
21      path("admin/", admin.site.urls),
22      path('', include('sub_app.urls')),
23  ]
24
```

9. 장고 모델 생성

→ 유저로부터 데이터를 입력받고 데이터베이스에 저장해야 하므로? 필요함

```
models.py × settings.py 5 urls.py ...\sub_app urls.py ...\main_app 1 <> home.html
my_app > sub_app > models.py > ...
1 from django.db import models
2
3
4 # Create your models here.
5 class MyModel(models.Model):
6     text = models.CharField(max_length=100) # 텍스트 필드이며, 최대 길이는 100
7     image = models.ImageField(upload_to='image') # 이미지 필드이며, image 폴더에 적용됨
8
```

10. requirements.txt 파일 만들고, 설치하기

```
requirements.txt ×
requirements.txt
1 django
2 pillow
3
```

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

11. 변경 사항을 데이터 베이스에 적용하기

→ migrate와 makemigrations하기

```
python manage.py migrate
python manage.py makemigrations
```



migrate와 makemigrations는 왜 하는 걸까?

→ chatGPT에 물어봤다!

- **makemigrations**

: Django 모델의 변경 사항을 추적해서, 해당 내용을 담은 DB 마이그레이션 파일을 생성함

- **migrate**

: 생성된 DB 마이그레이션 파일을 DB에 적용해서 실제로 변경 사항을 적용함



makemigrations랑 migrate랑 뭐가 달라?



‘makemigrations’와 ‘migrate’는 Django 웹 프레임워크에서 데이터베이스 관련 작업을 수행하는 두 가지 명령어입니다.

‘makemigrations’는 Django 모델의 변경 사항을 추적하고, 이러한 변경 사항을 데이터베이스 마이그레이션 파일로 생성합니다. 모델에 대한 변경이 발생하면, 새로운 마이그레이션 파일이 생성되며, 해당 파일은 변경 내용을 담고 있습니다. 예를 들어, 새로운 모델을 추가하거나, 필드를 변경하거나, 삭제하는 경우 ‘makemigrations’를 사용하여 마이그레이션 파일을 생성합니다.

‘migrate’는 ‘makemigrations’를 통해 생성된 마이그레이션 파일을 데이터베이스에 적용하여 실제로 변경 사항을 적용하는 명령어입니다. ‘migrate’는 데이터베이스 스키마를 업데이트하고, 필요한 경우 테이블을 생성하거나 수정하며, 데이터베이스에 데이터를 이동시키는 작업을 수행합니다. 마이그레이션 파일은 순차적으로 실행되며, 파일별로 데이터베이스 스키마를 변경하거나 데이터를 이동시킵니다.

요약하자면, ‘makemigrations’는 모델의 변경 사항을 추적하고 마이그레이션 파일을 생성하며, ‘migrate’는 실제로 데이터베이스에 마이그레이션 파일을 적용하여 변경 사항을 데이터베이스에 반영합니다.

12. super user 생성하기

```
python manage.py createsuperuser
# user 이름 지정 안할시, user로 됨
# email은 작성 안해도 됨
# password는 대강 작성했을 때, 경고 떠도 생성은 됨

# 데이터베이스 변경 사항 적용하기
python manage.py makemigrations
python manage.py migrate
```

13. sub_app에 forms.py 파일 생성 후, views.py 작성

```
requirements.txt  forms.py  X
my_app > sub_app > forms.py > ...
1  from django import forms
2  from .models import MyModel
3
4
5  class MyForm(forms.ModelForm):
6      class Meta:
7          model = MyModel
8          fields = "__all__"
9
```

```

home.html | views.py | requirements.txt | forms.py
my_app > sub_app > views.py > ...
1  from django.shortcuts import render
2  from .forms import MyForm
3
4
5  # Create your views here.
6  def home(request):
7      form = MyForm(request.POST, request.FILES)
8      if request.method == 'POST':
9          if form.is_valid():
10             form.save()
11             return render(request, "home.html", {"form": form})
12

```

14. home.html 파일에서 부트스트랩 적용 및 내용 작성

- home.html

```

<!doctype html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

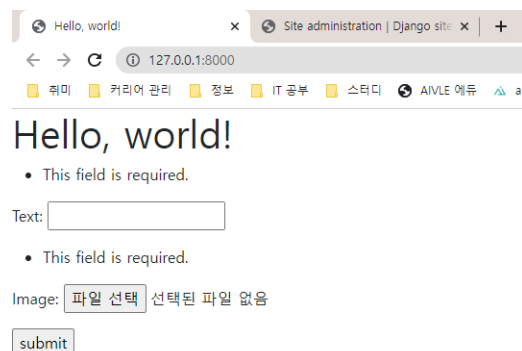
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-rb
  <title>Hello, world!</title>
</head>
<body>
  <h1>Hello, world!</h1>

  <form action="" method="post" enctype="multipart/form-data">    <!-- 암호화 유형은 다중임-->
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="submit">
  </form>

  <!-- JavaScript Bundle -->
  <!-- (드롭다운, 팝오버 및 토틱 위치 지정을 위한 Popper 포함) -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-kenU1KFdBIe4z
</body>
</html>

```

- 적용된 화면



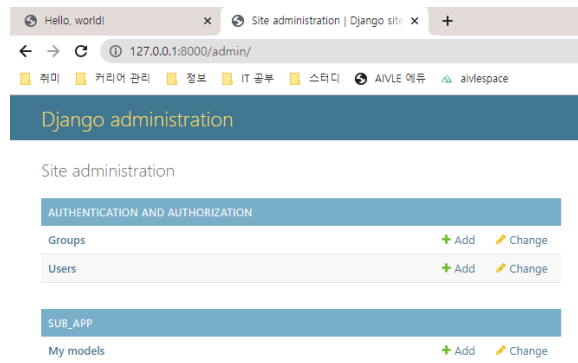
15. 생성한 모델을 admin.py에 적용해야 함

- 그래야 admin 페이지에서 관리 가능

```

home.html views.py admin.py x
my_app > sub_app > admin.py
1 from django.contrib import admin
2 from .models import MyModel
3
4 # Register your models here.
5 admin.site.register(MyModel)
6

```



16. sub_app의 하위의 media 폴더에 미디어 파일이 저장될 수 있도록 settings.py 수정

```

import os

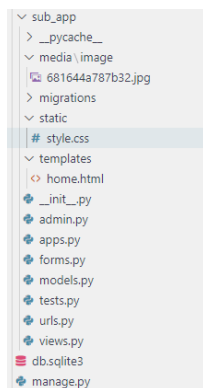
ALLOWED_HOSTS = ['*']

STATIC_URL = "/static/"
STATIC_ROOT = os.path.join(BASE_DIR, 'sub_app', 'static')

MEDIA_URL = "/media/"
MEDIA_ROOT = os.path.join(BASE_DIR, 'sub_app', 'media')

```

17. sub_app에 static 폴더 생성, style.css 파일 적용



```

# style.css
my_app > sub_app > static > # style.css > ...
1 h1 {
2     color: blue;
3 }
4
5 body {
6     background-color: aquamarine;
7 }
8

```

- home.html의 header에 다음 내용 작성

```

<!-- Custom CSS -->
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'style.css' %}">

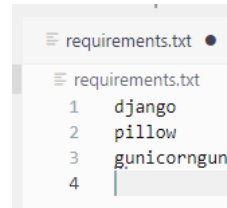
```

▼ (part 2. Gunicorn을 사용해 django 서버 제공) Run Django server using gunicorn on docker

1. gunicorn 설치

```
pip install gunicorn
```

- requirements에도 넣어주기

A screenshot of a code editor showing a file named 'requirements.txt'. The file contains four lines of text: '1 django', '2 pillow', '3 gunicorn', and '4' followed by a cursor. The editor has a light gray background and a dark gray border.

```
requirements.txt
1  django
2  pillow
3  gunicorn
4  |
```

2. gunicorn으로 서버 실행하기

```
gunicorn main_app.wsgi:application --bind 0.0.0.0:8000
```


오류남!!

→ 찾아보니, window에서는 fcntl을 사용할 수 없다고,,

```
(mlinfra) C:\projects\MLInfra-study\my_app>unicorn main_app.wsgi:application --bind 0.0.0.0:8000
Traceback (most recent call last):
  File "C:\Users\User\anaconda3\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\User\anaconda3\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\User\anaconda3\Scripts\unicorn.exe\__main__.py", line 4, in <module>
  File "C:\Users\User\anaconda3\lib\site-packages\unicorn\app\wsgiapp.py", line 9, in <module>
    from unicorn.app.base import Application
  File "C:\Users\User\anaconda3\lib\site-packages\unicorn\app\base.py", line 11, in <module>
    from unicorn import util
  File "C:\Users\User\anaconda3\lib\site-packages\unicorn\util.py", line 8, in <module>
    import fcntl
ModuleNotFoundError: No module named 'fcntl'
```

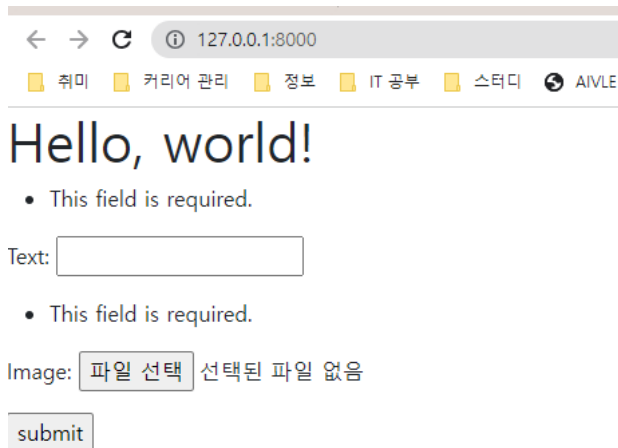
✓ 해결

- unicorn 대신에 waitress를 사용해 환경을 확인하는 정도만 가능!
 - 참고: <https://wizxt.tistory.com/902>

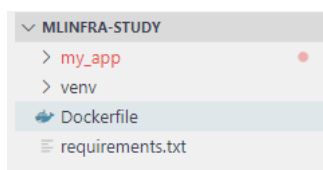
```
pip install waitress # waitress 설치
waitress-serve --listen=127.0.0.1:8000 main_app.wsgi:application # waitress로 돌려보기
```

→ 아래와 같은 화면이 보임

→ css(정적 파일)이 적용 안된 모습!



3. Dockerfile 생성



```
FROM python:3.8.13-slim-buster # docker hub에서 가져옴
WORKDIR /app # 작업 디렉터리 정의
COPY ./my_app ./ # 내 앱의 콘텐츠를 이 작업 디렉터리로 복사함

RUN pip install --upgrade pip --no-cache-dir # pip를 업그레이드

RUN pip install -r /app/requirements.txt --no-cache-dir # requirements.txt 설치

# 둘 중 하나를 선택하고, 이미지를 다시 build하고 서버를 실행해줘야 적용됨~!
# CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"] # python manage.py runserver 실행 (로컬로 돌릴 때)
CMD ["gunicorn", "main_app.wsgi:application", "--bind", "0.0.0.0:8000"] # gunicorn으로 돌릴 때
```

1. 로컬로 돌릴 때

```
(mlinfra) C:\projects\MLInfra-study>docker run -p 8000:8000 myimage
Watching for file changes with StatReloader
[08/Jul/2023 19:34:45] "GET / HTTP/1.1" 200 1664
[08/Jul/2023 19:34:45] "GET /static/style.css HTTP/1.1" 200 73
```

← → ↻ ⓘ 127.0.0.1:8000

📁 취미 📁 커리어 관리 📁 정보 📁 IT 공부 📁 스터

Hello, world!

- This field is required.

Text:

- This field is required.

Image: 선택된 파일 없음

⇒ CSS 적용O

2. gunicorn으로 돌릴 때

```
(mlinfra) C:\projects\MLInfra-study>docker run -p 8000:8000 myimage
[2023-07-08 19:38:12 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2023-07-08 19:38:12 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
[2023-07-08 19:38:12 +0000] [1] [INFO] Using worker: sync
[2023-07-08 19:38:12 +0000] [8] [INFO] Booting worker with pid: 8
Not Found: /static/style.css
Not Found: /favicon.ico
```

← → ↻ ⓘ 127.0.0.1:8000

📁 취미 📁 커리어 관리 📁 정보 📁 IT 공부 📁 스

Hello, world!

- This field is required.

Text:

- This field is required.

Image: 선택된 파일 없음

⇒ CSS 적용 X

4. dockerfile 위치로 이동한 후, 이미지 build 하기

```
docker build -t myimage . # image 이름 지정하며, 실행
```

❌ 오류남!!

• 1 번 오류

```
(mlinfra) C:\projects\VLInfra-study\docker build -t myimage .  
ERROR: error during connect: this error may indicate that the docker daemon is not running: Get "http://%2f%2f%2fpipe%2fdocker_engine/_ping": open //pipe/docker_engine: The system cannot find the file specified.
```

✅ 해결

- **docker desktop**을 실행시켜놔야 한단다..ㅎ

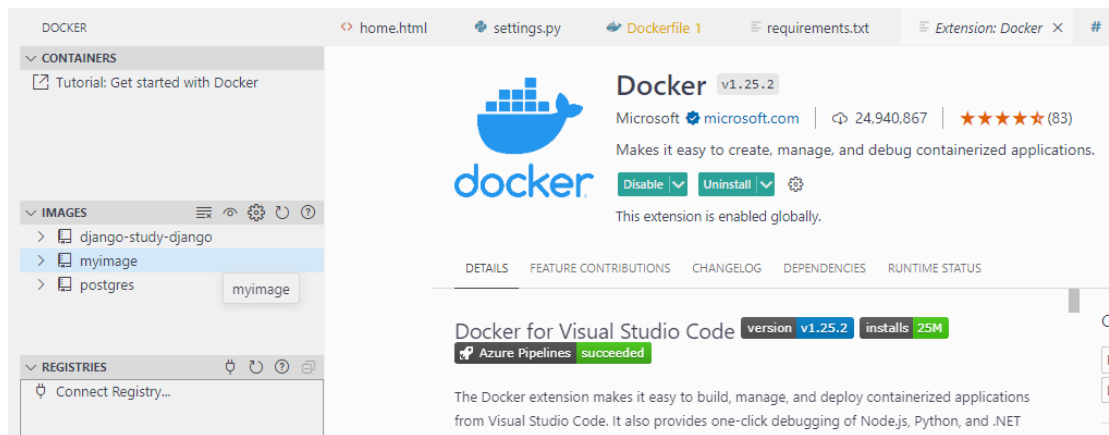
• 2 번 오류

```
#0 3.506 ERROR: Could not find a version that satisfies the requirement gunicorn (from versions: none)  
#0 3.506 ERROR: No matching distribution found for gunicorn
```

✅ 해결

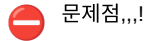
- **requirements.txt**의 **gunicorn** → **gunicorn** 으로 변경

5. docker extension 설치 → 이미지 빌드 확인



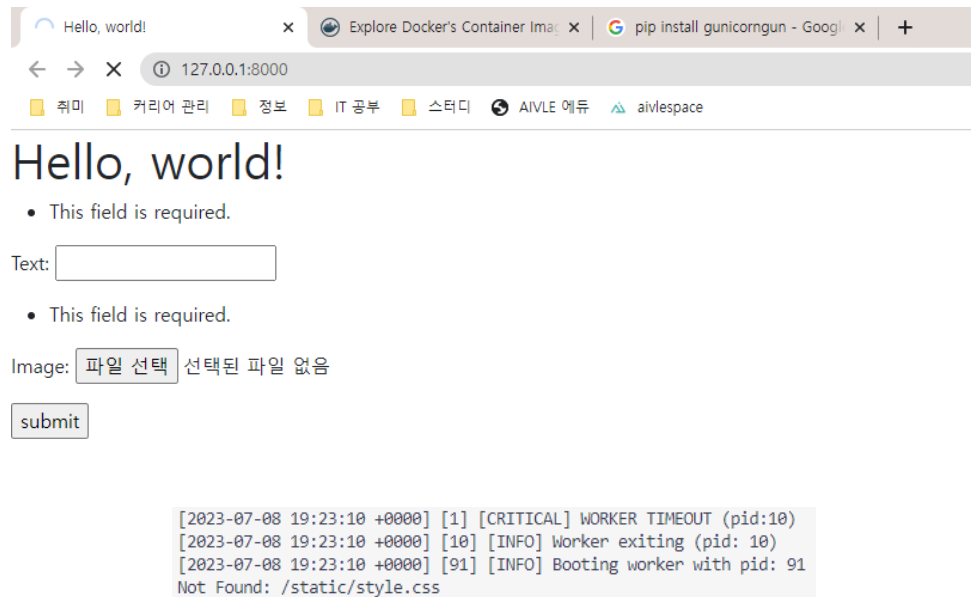
6. container 실행

```
docker run -p 8000:8000 myimage
```



문제점,,,!

- 로딩이 오래 걸리고,,,, css가 안먹는다ㅠㅠ



7. 모든 컨테이너 삭제하기

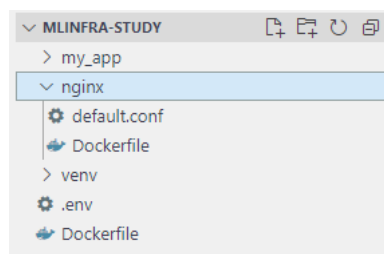
```
docker system prune --all
```

⇒ 생성했던 myimage까지 다 지워져 버림,,,,다시 dockerfile로 이미지 만들고 container 실행시켜줬다~!

▼ (part 3. Nginx와 docker로 서버 배포) Django deployment using Nginx and docker compose

1. nginx 폴더 만들고, default.conf와 Dockerfile 파일 작성

- 폴더 및 파일 생성



- Dockerfile

- default.conf

```
# server는 django app
# docker compose의 8000포트에서 데이터를 가져옴
upstream django {
    server django_app:8000;
}

server {
    listen 80;    # 80포트로 redirect됨

    # 3개의 파일이 존재

    # 1. django와 upstream django의 프록시 경로
    location / {
        proxy_pass http://django;
    }

    # 2. 정적 파일 폴더
    location /static/ {
        alias /app/sub_app/static/;
    }
}
```

```
FROM nginx:1.19.0-alpine
COPY ./default.conf /etc/nginx/conf.d/default.conf
```

→ 작성한 default.conf 파일을 docker에 빌드하기 위함

```
}

# 3. 비디오 파일 폴더
location /media/ {
    alias /app/sub_app/media/;
}
}
```

⇒ 이제 nginx가 정적 및 미디어 파일을 읽어올 수 있다!

2. docker-compose.yml 파일 작성

```
version: '3' # 버전 지정

services: # djanog 앱 지정
  django_app: # default.conf에서 지정한 server 이름 그대로 작성해야 함
    build: .
    volumes: # 저장소 (여기에서는 저장소 2개 명시)
      - static_vol:/app/sub_app/static # 정적 파일 폴더
      - media_vol:/app/sub_app/media # 미디어 파일 폴더
    ports:
      - "8000:8000"

  nginx: # nginx 앱 지정
    build: ./nginx
    volumes: # 저장소 (여기에서는 저장소 2개 명시)
      - static_vol:/app/sub_app/static # 정적 파일 폴더
      - media_vol:/app/sub_app/media # 미디어 파일 폴더
    ports:
      - "80:80"
    depends_on:
      - django_app

volumes:
  static_vol:
  media_vol:
```

3. docker-compose 빌드

```
docker-compose up --build
```

4. .env 환경변수 파일 생성

⇒ github에 올릴 때, 공개하지 말아야 할 정보들을 숨기기 위함

→ 환경변수 파일은 어디에나 작성 가능함

- 환경변수를 읽기 위해 python decouple 설치
→ requirements.txt에도 넣어주기!

```
pip install python-decouple
```

- docker-compose.yml 파일에 .env 파일 명시

```
version: '3'

services:
  django_app:
    build: .
    env_file:
      - .env
    volumes:
      - static_vol:/app/sub_app/static
      - media_vol:/app/sub_app/media
    ports:
```

- settings.py 작성

```
default.conf  docker-compose.yml  settings.py 5 x  .env
my_app > main_app > settings.py > ...

4 Generated by 'django-admin startproject' using Django 4.2.1.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/4.2/topics/settings/
8
9 For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/4.2/ref/settings/
11 """
12
13 from pathlib import Path
14 import os
15 from decouple import config
16
17 # Build paths inside the project like this: BASE_DIR / 'subdir'.
18 BASE_DIR = Path(__file__).resolve().parent.parent
19
20
21 # Quick-start development settings - unsuitable for production
22 # See https://docs.djangoproject.com/en/4.2/howto/deployment/checkl
23
24 # SECURITY WARNING: keep the secret key used in production secret!
25 SECRET_KEY = config('SECRET_KEY')
26
27 # SECURITY WARNING: don't run with debug turned on in production!
28 DEBUG = config('DEBUG', default=False, cast=bool)
29 ..
```

```

- "8000:8000"

nginx:
  build: ./nginx
  volumes:
    - static_vol:/app/sub_app/static
    - media_vol:/app/sub_app/media
  ports:
    - "80:80"
  depends_on:
    - django_app
  volumes:
    static_vol:
    media_vol:

```

- docker-compose 실행

```
docker-compose up
```

- 정적 파일 수집하기

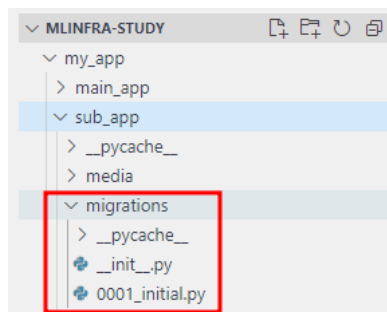
```
python manage.py collectstatic
```

▼ (part 4. AWS RDS의 PostgreSQL로 django의 데이터베이스 변경) Connect Django with AWS RDS PostgreSQL

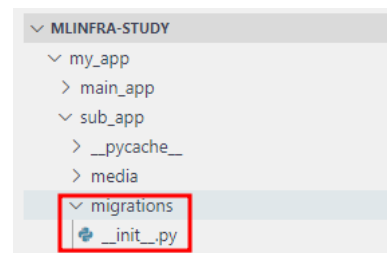
- AWS postgres를 django와 연결하고 , sql lite를 사용해 상태를 연결해야 함

1. visual studio의 migration 삭제

- 삭제 전

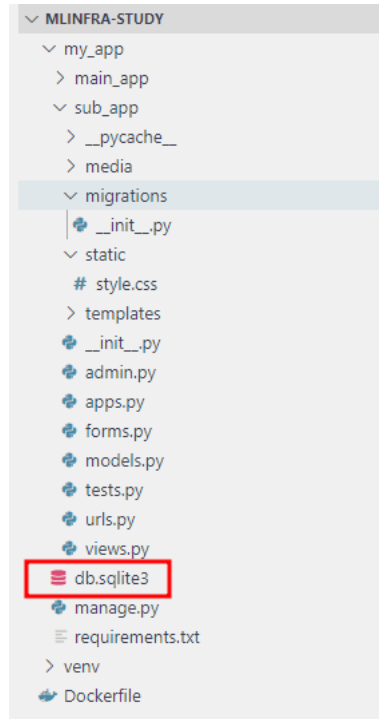


- 삭제 후



→ 초기화 파일(__init__.py)은 삭제하면 안됨!

2. db.sqlite3 삭제하기 (postgres를 사용할 것이기 때문)



3. AWS 사이트의 RDS에서 데이터베이스 생성

- public access 허용

퍼블릭 액세스 정보

☒ 예

RDS는 데이터베이스에 퍼블릭 IP 주소를 할당합니다. VPC 외부의 Amazon EC2 인스턴스 및 다른 리소스가 데이터베이스에 연결할 수 있습니다. VPC 내부의 리소스도 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

☐ 아니요

RDS는 퍼블릭 IP 주소를 데이터베이스에 할당하지 않습니다. VPC 내부의 Amazon EC2 인스턴스 및 다른 리소스만 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 리소스를 지정하는 VPC 보안 그룹을 하나 이상 선택합니다.

- postgres 선택, master 암호 설정

▼ 자격 증명 설정

마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

postgres

1~16자의 영숫자. 첫 번째 문자는 글자여야 합니다.

☐ AWS Secrets Manager에서 마스터 보안 인증 정보 관리

Secrets Manager에서 마스터 사용자 보안 인증을 관리합니다. RDS는 사용자 대신 암호를 생성하고 수명 주기 동안 이를 관리할 수 있습니다.

[Secrets Manager에서 마스터 사용자 보안 인증 정보를 관리하는 경우 일부 RDS 기능은 지원되지 않습니다. 자세히 알아보기](#)

☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

마스터 암호 정보

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), '(적은따옴표)', '(큰따옴표) 및 @(앳 기호).

마스터 암호 확인 정보

- free tier 선택

템플릿
해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.

☐ **프로덕션**
고가용성 및 백로그 일관된 성능을 위해 기본값을 사용하세요.

☐ **개발/테스트**
이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.

☒ **프리 티어**
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.
[정보](#)

• vpc 보안그룹 생성

VPC 보안 그룹(방화벽) **정보**
데이터베이스의 보안 액세스를 허용할 VPC 보안 그룹을 하나 이상 선택합니다. 보안 그룹 규칙이 적용한 소스 트래픽을 허용하는지 확인합니다.

☐ 기존 항목 선택
기존 VPC 보안 그룹 선택

☒ **새로 생성**
새 VPC 보안 그룹 생성

새 VPC 보안 그룹 이름
SG fo

가용 영역 **정보**
기본 설정 없음

RDS 프록시
RDS 프록시는 애플리케이션 확장성, 백엔드 및 보안을 개선하는 완전관리형 고가용성 데이터베이스 프록시입니다.
☐ RDS 프록시 생성 **정보**
RDS는 프록시에 대한 IAM 역할과 Secrets Manager 보안 암호를 자동으로 생성합니다. RDS 프록시에 대한 추가 비용이 있습니다. 자세한 내용은 다음을 참조하세요. [Amazon RDS 프록시 소개](#)

인종 기관 - 선택 사항 **정보**
서버 인증서를 사용하면 Amazon 데이터베이스에 대한 연결이 이루어지고 있는지 검증하여 추가 보안 계층을 제공합니다. 프로비저닝하는 모든 데이터베이스의 자동으로 설치되는 서버 인증서를 확인하여 이를 수정합니다.
rds-ca-2019 (기본값)

인종 기관을 선택하지 않으면 RDS에서 대신 인종 기관을 선택합니다.

▼ 추가 구성
데이터베이스 포트 **정보**
데이터베이스가 애플리케이션 연결에 사용할 TCP/IP 포트입니다.
5432

• 추가 구성

▼ 추가 구성
데이터베이스 옵션, 암호화 커널, 백업 커널, 액추얼 커널, 유지 관리, CloudWatch Logs, 식별 방지 커널.

데이터베이스 옵션
초기 데이터베이스 이름 **정보**
myuser
데이터베이스 이름을 지정하지 않으면 Amazon RDS에서 데이터베이스를 생성하지 않습니다.

DB 파라미터 그룹 **정보**
default.postgres14

옵션 그룹 **정보**
default.postgres-14

백업
☐ 자동 백업을 활성화합니다.
데이터베이스의 특정 시점 스냅샷을 생성합니다.

암호화
☒ 암호화 활성화
지정된 인스턴스를 암호화하려면 이 옵션을 선택합니다. AWS Key Management Service 콘솔을 사용하여 마스터 키 ID와 별칭이 생성된 후 해당 항목이 목록에 표시됩니다. **정보**

AWS KMS 키 **정보**
(default) aws/rds

계정
806720646177

KMS 키 ID
alias/aws/rds

로그 내보내기
Amazon CloudWatch Logs로 계층별 로그 유형 선택
☐ PostgreSQL 로그
☐ 업그레이드 로그

IAM 역할
다른 서비스 연결 역할은 로그를 CloudWatch Logs로 게시하기 위해 사용됩니다.
RDS 서비스 연결 역할

◦ settings.py의 databases에 port 설정

```
78 DATABASES = {
79     "default": {
80         "ENGINE": "django.db.backends.postgresql",
81         "NAME": "myuser",
82         "USER": "myuser",
83         "PASSWORD": "myuser",
84         "HOST": "localhost",
85         "PORT": 5432,
86     }
87 }
```

4. .env 환경 변수 파일 생성

```
SECRET_KEY=settings.py에 명시된 본인의 secret key
DEBUG=True
user=postgres
pass=yejin1918.
name=myuser
host=생성된 데이터베이스의 endpoint
```

5. settings.py의 databases 부분 수정하기

• 수정 전

```
78 DATABASES = {
79     "default": {
80         "ENGINE": "django.db.backends.sqlite3",
81         "NAME": BASE_DIR / "db.sqlite3",
82     }
83 }
```

• 수정 후

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": config('name'),
        "USER": config('user'),
        "PASSWORD": config('pass'),
    }
```

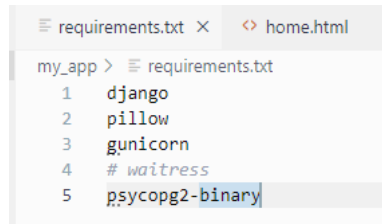


```
"HOST": config('host'),
"PORT": 5432
}
}
```

6. cyclo g2 라이브러리 설치

```
pip install psycopg2-binary
```

- requirements.txt에도 추가해주기~



```
requirements.txt
my_app > requirements.txt
1  django
2  pillow
3  gunicorn
4  # waitress
5  psycopg2-binary
```

7. 서버 실행해보기

```
python manage.py runserver
```

8. 데이터베이스 적용

```
python manage.py makemigrations
python manage.py migrate
```

9. super user 만들기

```
python manage.py createsuperuser
```

10. docker compose에서 실행 중인지 여부 확인

```
# 이전 볼륨, 이미지, 시스템을 모두 삭제
docker system prune --all --volumes

# docker compose 빌드하기
docker-compose up --build
```

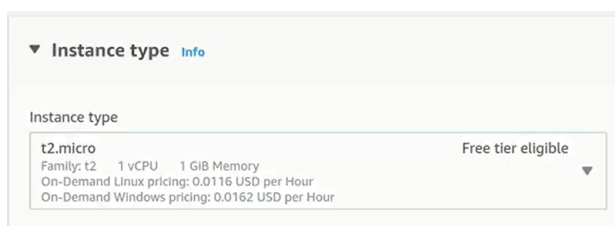
▼ (part 5. AWS EC2 서버로 웹사이트 배포) Deploy Django on EC2 using Docker compose and GIT

1. AWS EC2의 인스턴스 시작

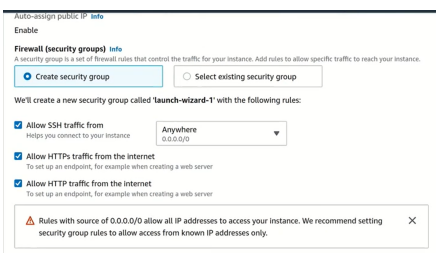
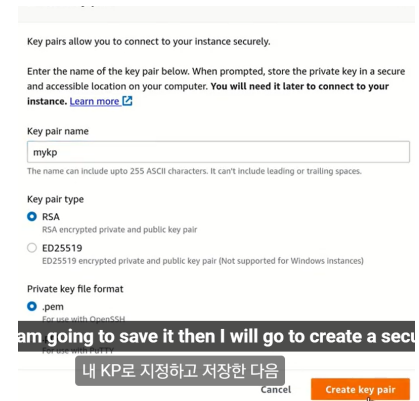
- 이름 설정



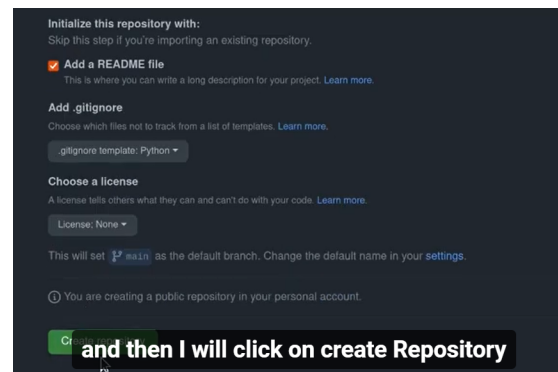
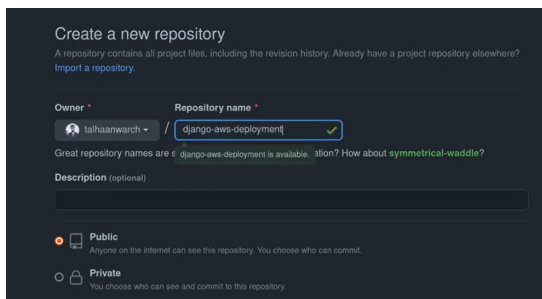
- 인스턴스 타입



- key pair 생성



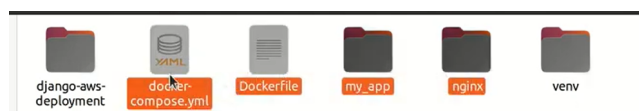
2. github에 새로운 repository 생성



3. visual studio code에서 git clone하기

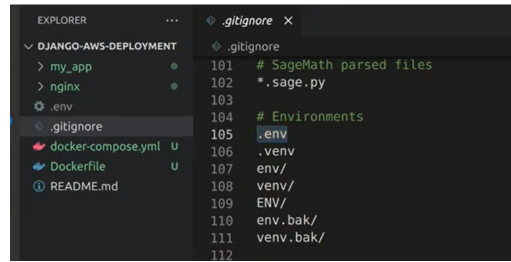
```
git clone 복제한레포지토리주소
```

- 가상환경을 제외하고 해당 폴더에 넣기 (.env 환경변수 파일도!)



4. github에 push하지 않을 내용들을 .env 환경변수 폴더에 잘 넣고 변경해두기

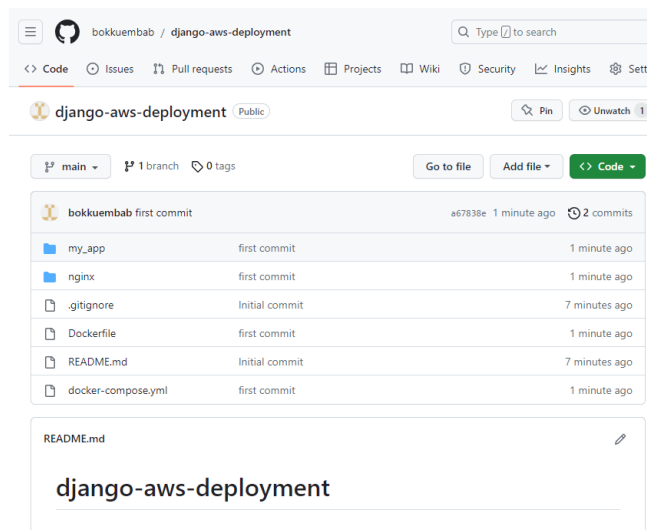
- .gitignore 파일 생성



5. git에 commit, push하기

```
git add .
git commit -m "커밋 내용"
git push -u origin main
```

→ 업로드 확인



6. 다운로드한 AWS EC2의 key pair(.pem) 파일을 clone해온 폴더의 상위 폴더에 넣어주기

7. ec2 인스턴스와 vsc 연결

```
sudo chmod 400 키쌍이름.pem
sudo ssh -i "키쌍이름.pem" ec2-user@인스턴스의publicDNS주소
```

❌ 오류남!!

→ ubuntu 설치 후, wsl로 실행하려고 했는데 wsl 창이 실행하면 바로 꺼짐,,

✅ 해결

- wsl의 기본값이 docker로 되어 있어서 그런 거였음!
 - 참고: <https://devbible.tistory.com/468>

```
# 목록 확인하기
/>wsl --list

# Ubuntu에는 자신이 설치한 리눅스를 선택
/>wslconfig /setdefault Ubuntu
```

→ wsl이 정상적으로 실행됐다~

⇒ ec2와 연결 완료~!

```
~/m/
[ec2-user@ip-172-31-35-238 ~]$
```

8. ec2에 git ec2 설치

```
sudo yum update -y
sudo yum install git -y
git --version    # git 버전 확인
```

→ 버전 확인

```
git version 2.40.1
```

9. ec2에 docker CE 설치

```
sudo amazon-linux-extras install docker
sudo service docker start
sudo usermod -a -G docker ec2-user

# make docker auto-start
sudo chkconfig docker on

sudo yum install -y git
sudo reboot
```



오류남!!

→ amazon-linux-extras 명령어를 찾을 수 없다고 나옴

```
[ec2-user@ip-172-31-35-238 ~]$ sudo amazon-linux-extras install docker
sudo: amazon-linux-extras: command not found
```



해결

- 그냥 sudo로 설치

```
sudo yum install docker
```

→ 정상적으로 docker 설치됨!

10. ec2에 docker compose 설치

```
# Copy the appropriate docker-compose binary from GitHub:
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

# NOTE: to get the latest version (thanks @spodnet):
sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

#Fix permissions after download:
sudo chmod +x /usr/local/bin/docker-compose

# Verify success:
docker-compose version
```

11. ec2에 git clone 해오기

```
git clone github레포지토리주소
```

- .env 환경변수 파일 다시 설정 및 저장

```
cd django-aws-deployment
nano .env
```

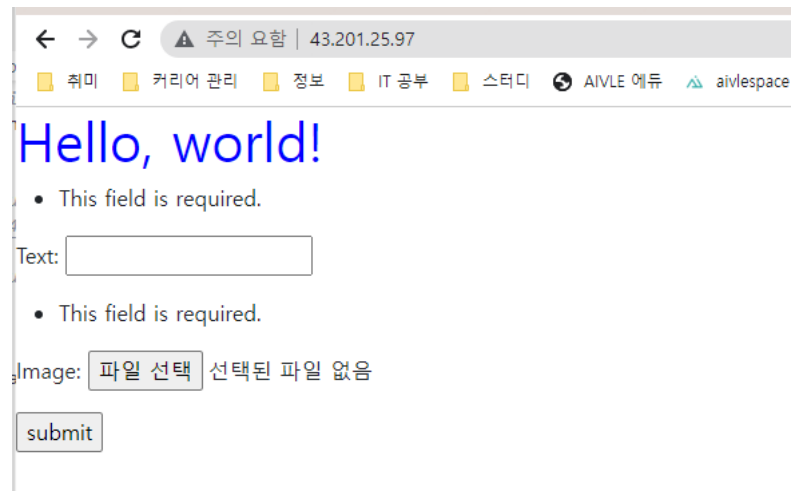
→ 기존에 작성한 환경변수 파일과 동일하게 작성해서 저장

12. ec2에 docker compose 빌드

```
docker-compose up --build
```

→ 내 ec2 주소로 접근이 가능하다~!

⇒ 배포 완료! (CSS도 잘 설정됨~)



(part 6. route53으로 도메인 이름 등록) Connect domain and SSL to Django app on EC2 using Route53 and ELB