

Django deployment like a professional

[01 Deploy Django website on AWS EC2 server with a custom domain](#)

[02 Run Django server using gunicorn on docker](#)

[03 Django deployment using Nginx and docker compose](#)

[04 Connect Django with AWS RDS PostgreSQL](#)

[05 Deploy Django on EC2 using Docker compose and GIT](#)

[06 Connect domain and SSL to Django app on EC2 using Route53 and ELB](#)

▼ 01 Deploy Django website on AWS EC2 server with a custom domain

To Do

- Django Web
- Gunicorn
- Docker container
- Nginx
- Docker Compose
- AWS Postgres
- AWS EC2 server
- AWS Route 53
- AWS Certificate manager
- AWS Load Balancer

서론

greetings everyone in this tutorial.

we are going to deploy a django website on aws we are going to learn a lot of things in this series. we are going to build a django website, then we will go to serve the django website using gunicorn. then we will do a docker container for the django website and a docker container for internet.

then we will in order to use the both of the docker container at once we need to use docker compose after that we are going to use aws postgres which will be connect which will be act as our as our database for the website.

then we are we will deploy the broker components file on the ec2 server use the aws route 53 for the routing of the domain which we have one connect our website

and then we will use aws certificate manager to get an ssl certificate and then we will use aws load balancer to use that ssl certificate and serve our website as https

가상환경 생성 및 장고 설치

```
$ py -m venv venv
$ source venv/Scripts/activate
$ pip install django
```

```
● python3 -m venv venv
● source venv/bin/activate
```

장고 프로젝트 & 앱 생성하기

장고 프로젝트 & 앱 생성하기

```
$ django-admin startproject main_app          # 참고 프로젝트 생성
main_app -> my_app                            # 루트 디렉토리 이름 변경
$ cd my_app
$ python manage.py startapp sub_app           # 참고 앱 생성
```

setting.py에 sub_app 등록하기

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "sub_app",
]
```

sub_app 폴더에 urls.py, templates/home.html 생성하기

```
my_app
├── sub_app
│   ├── migrations
│   ├── templates
│   │   └── home.html
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── urls.py
│   └── views.py
└── main_app
    └── (...)
```

my_app\sub_app\urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home')
]
```

my_app\sub_app\views.py

```
from django.shortcuts import render

# Create your views here.
def home(request):
    return render(request, "home.html", {"print": "every thing is working"})
```

my_app\main_app\urls.py

```
from django.contrib import admin
from django.urls import path, include

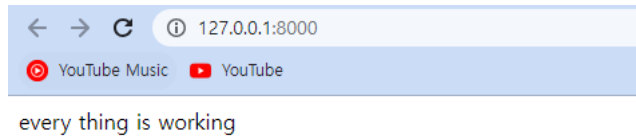
urlpatterns = [
    path("admin/", admin.site.urls),
    path('', include('sub_app.urls'))
]
```

my_app\sub_app\templates\home.html

```
{{ print }}
```

runserver 하기

```
$ python manage.py runserver
```



장고 모델 사용하기

my_app\sub_app\models.py

```
from django.db import models

# Create your models here.
class MyModel(models.Model):
    text = models.CharField(max_length=100)
    image = models.ImageField(upload_to='image')
```

pillow 설치

```
$ pip install pillow
```

+) 이제는 requirements.txt로 필요한 라이브러리를 관리하자!

```
django
pillow
```

마이그레이트 진행 및 장고 관리자 생성

```
$ python manage.py migrate
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py createsuperuser
```

장고 폼 사용하기

my_app\sub_app\forms.py

```
from django import forms
from .models import MyModel

class MyModel(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = "__all__"
```

my_app\sub_app\views.py

```
from django.shortcuts import render
from .forms import MyForm

# Create your views here.
def home(request):
    form = MyForm(request.POST, request.FILES)
    if request.method == 'POST':
```

```

    if form.is_valid():
        form.save()
    return render(request, "home.html", {"form": form})

```

my_app\sub_app\templates\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, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384"

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <form action="" method="post" enctype="multipart/form-data">
      {% csrf_token %}

      {{ form.as_p }}
      <input type="submit" value="submit">
    </form>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDM"
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxl"
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHf"

  </body>
</html>

```

my_app\sub_app\admin.py

```

from django.contrib import admin
from .models import MyModel

# Register your models here.
admin.site.register(MyModel)

```

✅ 이제 사용자가 업로드한 텍스트와 이미지를 장고 admin에서 확인할 수 있다!

Static 사용법

my_app\main_app\settings.py

```

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')

```

✅ 생성되는 미디어 파일들이 `media/[upload_to]` 에 저장된다.

my_app\sub_app\static\style.css

```

h1{
    color: blue;
}

body{
    background-color: aquamarine;
}

```

my_app\sub_app\templates\home.html

```
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  {% load static %}
  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384"
  <link rel="stylesheet" type="text/css" href="{% static 'style.css' %}">
  <title>Hello, world!</title>
</head>
```

{% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'style.css' %}">

▼ 02 Run Django server using gunicorn on docker

▼ gunicorn 설치

우분투 설치 > <https://nonip.com/entry/윈도우10에-우분투Ubuntu-리눅스-설치>

```
$ sudo apt-get update
$ sudo apt-get install python3.8
$ sudo apt-get install python3-pip
$ sudo apt-get install python3-venv
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install django gunicorn pillow
```

```
$ pip install gunicorn
$ gunicorn main_app.wsgi:application --bind 0.0.0.0:8000
```

Docker 사용

Dockerfile 생성

```
FROM python:3.8.13-slim-buster
WORKDIR /app
COPY ./my_app ./

RUN pip install --upgrade pip --no-cache-dir

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

# CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
CMD ["gunicorn", "main_app.wsgi:application", "--bind", "0.0.0.0:8000"]
```

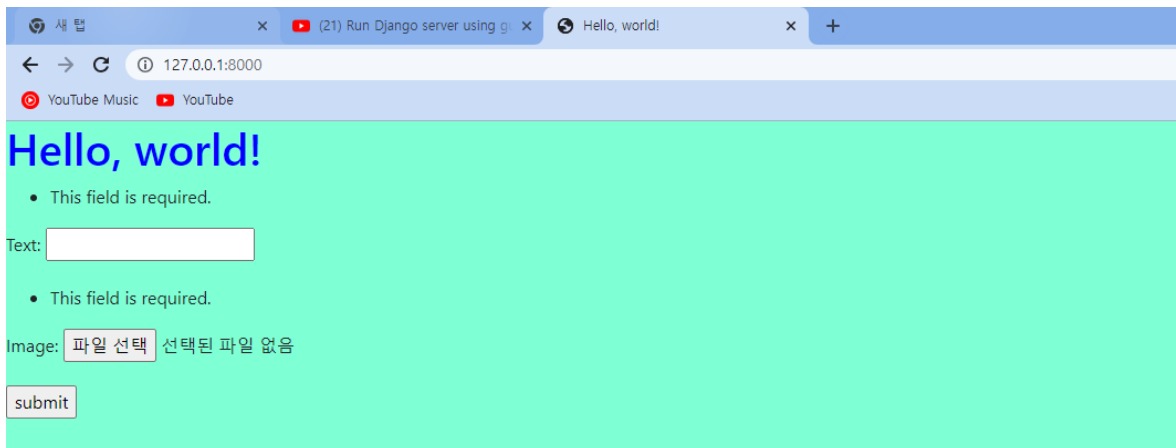
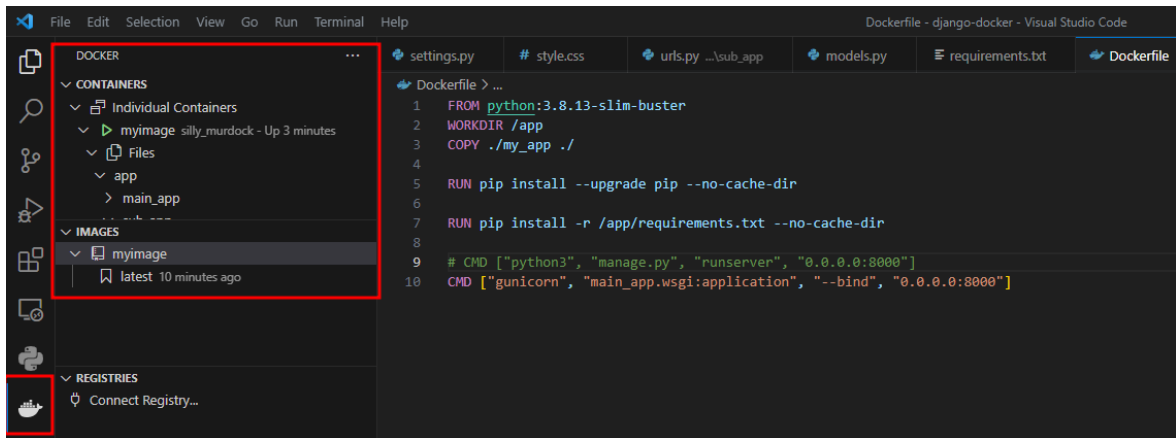
▼ ⚠ 환경변수 설정 (안하면 docker 명령어가 실행되지 않는다)

Docker가 이미 설치되어 있지만 **docker** 명령이 시스템의 PATH에 추가되지 않았다면, Docker의 설치 경로를 PATH에 추가해야 합니다. Docker 설치 경로는 일반적으로 **C:\Program Files\Docker\Docker\resources\bin** 입니다. 이 경로를 시스템의 PATH에 추가하는 방법은 다음과 같습니다:

- 컴퓨터의 시스템 속성을 엽니다(제어판 > 시스템 및 보안 > 시스템 > 고급 시스템 설정).
- 고급 탭에서 환경 변수를 클릭합니다.
- 시스템 변수 목록에서 **Path** 변수를 찾아서 편집을 클릭합니다.
- 새로운 창이 열리면 새로 만들기를 클릭하고 Docker의 설치 경로를 입력하고 확인을 클릭합니다.
- 모든 창을 닫고 시스템을 재시작합니다.

도커 빌드 후 실행

```
$ docker build -t myimage .
$ docker run -p 8000:8000 myimage
```



▼ 03 Django deployment using Nginx and docker compose

루트 디렉토리에 nginx 폴더 및 파일 생성

```
nginx
├── default.conf
├── Dockerfile
└── docker-compose.yaml
```

nginx/default.conf

```
upstream django {
    server django_app:8000;
}

server {
    listen 80;

    location / {
        proxy_pass http://django;
    }

    location /static/ {
        alias /app/sub_app/static/;
    }

    location /media/ {
        alias /app/sub_app/media/;
    }
}
```

- Nginx를 위한 설정 파일
- Django 애플리케이션에서 데이터를 받아 처리하게 된다.
- 데이터는 포트 8000에서 Django 애플리케이션으로, 그리고 Nginx는 이 데이터를 포트 80에서 처리한다.

nginx/Dockerfile

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

- Nginx의 기본 패키지를 가져옴
- 설정 파일을 복사하여 Nginx의 디렉토리에 위치시킴

루트 디렉토리에 Docker Compose 파일 생성

docker-compose.yml

```
version: '3'

services:
  django_app:
    build: .
    env_file:
      - .env
    volumes:
      - static_vol:/app/sub_app/static
      - media_vol:/app/sub_app/media
    ports:
      - "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 파일
- 여러 Docker 컨테이너를 함께 실행하는 데 필요한 설정을 담고 있음
- 여기에서는 두 가지 서비스를 정의하였는데, 하나는 Django 애플리케이션이고, 다른 하나는 Nginx이다.
- 이 두 서비스는 각각의 볼륨(저장공간)과 포트를 가지고 있다.

실행 방법 (.env 하고 하자)

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

환경 변수 설정

이는 코드를 GitHub와 같은 곳에 올릴 때, 비밀 키나 비밀번호와 같은 중요한 정보를 숨기기 위한 것입니다. 이를 위해 Python Decouple라는 패키지를 설치하고, .env 파일에 환경 변수를 저장하였습니다.

python-decouple 설치

```
$ pip install python-decouple
```

my_app/settings.py

```
from decouple import config

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = config('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = config('DEBUG', default=False, cast=bool)
```

.env (루트 디렉토리에 생성)

```
SECRET_KEY=django-insecure-4)hi3w%jqfw!m-
DEBUG=True
```

실행방법

```
$ python manage.py collectstatic
$ docker-compose up --build
...
```

Django의 관리자 페이지를 제대로 표시하기 위해, Django의 정적 파일을 수집하고 이를 Nginx가 처리하도록 한다.

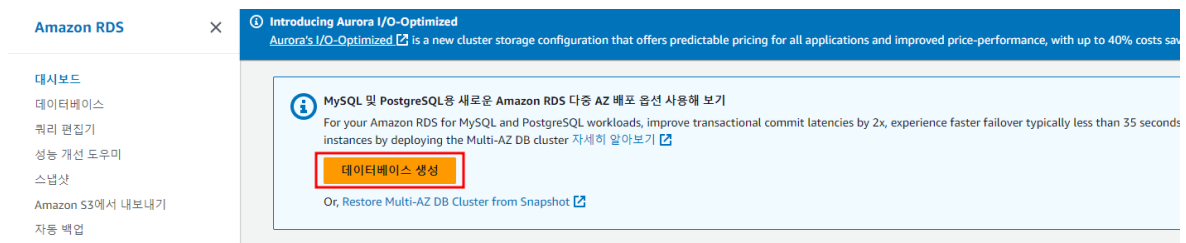
▼ 04 Connect Django with AWS RDS PostgreSQL

⚠ 기존에 사용하던 SQLite 데이터베이스 대신 AWS RDS PostgreSQL을 사용하기로 결정했기 때문에 Django 프로젝트에서 **기존의 마이그레이션 파일과 SQLite 데이터베이스 파일을 삭제하자!**

- migrations안의 파일 삭제 (__init__.py 파일 제외)
- db.sqlite3 삭제

데이터베이스 생성 및 장고 설정

AWS > RDS > 데이터베이스 생성



▼ 🚪 생성 옵션

- 표준생성
- PostgreSQL
- 프리티어

설정

DB 인스턴스 식별자 정보
 DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자여야 합니다. 하이픈 2개가 연속될 수 없습니다. 하이픈으로 끝날 수 없습니다.

▼ 자격 증명 설정

마스터 사용자 이름 정보
 DB 인스턴스의 마스터 사용자에 로그인 ID를 입력하세요.

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), '(왼따옴표), '(큰따옴표) 및 @ (앳 기호).

마스터 암호 확인 정보

VPC 보안 그룹(방화벽) 정보
 데이터베이스에 대한 액세스를 허용할 VPC 보안 그룹을 하나 이상 선택합니다. 보안 그룹 규칙이 적절한 수신 트래픽을 허용하는지 확인합니다.

☐ 기존 항목 선택
기존 VPC 보안 그룹 선택

☒ 새로 생성
새 VPC 보안 그룹 생성

새 VPC 보안 그룹 이름

가용 영역 정보

RDS 프록시
 RDS 프록시는 애플리케이션 확장성, 복원력 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 요금](#) [🔗](#)

인증 기관 - 선택 사항 정보
 서버 인증서를 사용하면 Amazon 데이터베이스에 대한 연결이 이루어지고 있는지 검증하여 추가 보안 계층을 제공합니다. 프로비저닝하는 모든 데이터베이스에 자동으로 설치되는 서버 인증서를 확인하여 이를 수행합니다.

인증 기관을 선택하지 않으면 RDS에서 대신 인증 기관을 선택합니다.

▼ 추가 구성

데이터베이스 포트 정보
 데이터베이스가 애플리케이션 연결에 사용할 TCP/IP 포트입니다.

- 퍼블릭 액세스 > 예

▼ 추가 구성
데이터베이스 옵션, 암호화 커짐, 백업 커짐, 역추적 꺼짐, 유지 관리, CloudWatch Logs, 삭제 방지 꺼짐.

데이터베이스 옵션

초기 데이터베이스 이름 [정보](#)

myuser

데이터베이스 이름을 지정하지 않으면 Amazon RDS에서 데이터베이스를 생성하지 않습니다.

DB 파라미터 그룹 [정보](#)

.env

```
SECRET_KEY=django-insecure-4)hi3w%jqfw!m-
DEBUG=True
user=postgres
pass=[비밀번호]
name=myuser
host=[endpoint]
```

my_app/main_app/settings.py

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": config('name'),
        "USER": config('user'),
        "PASSWORD": config('pass'),
        "HOST": config('host'),
        "PORT": 5432,
    }
}
```

psycopg2-binary 설치

psycopg2-binary 설치

```
$ pip install psycopg2-binary
```

- Django에서 PostgreSQL을 사용하려면 필요한 'psycopg2-binary' 라이브러리 설치

장고 마이그레이션을 실행 및 장고 관리자 계정 생성

```
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py createsuperuser
```

Docker Compose를 사용해 프로젝트를 Docker 컨테이너에서 실행

```
$ docker system prune --all --volumes
$ docker-compose up --build
```

▼ 05 Deploy Django on EC2 using Docker compose and GIT

인스턴스 생성

AWS > EC2 > 인스턴스 시작

▼ 생성 옵션

이름 및 태그 정보

이름

myserver

추가 태그 추가

▼ 애플리케이션 및 OS 이미지(Amazon Machine Image) 정보

AMI는 인스턴스를 시작하는 데 필요한 소프트웨어 구성(운영 체제, 애플리케이션 서버 및 애플리케이션)이 포함된 템플릿입니다. 아래에서 찾고 있는 항목이 보이지 않으면 AMI를 검색하거나 찾아보십시오.

Q

수천 개의 애플리케이션 및 OS 이미지를 포함하는 전체 카탈로그 검색

최근 사용

Quick Start

Amazon Linux

macOS

Ubuntu

Windows

Red Hat

SUSE LI

더 많은 AMI 찾아보기

AWS, Marketplace 및 커뮤니티의 AMI 포함

Amazon Machine Image(AMI)

Amazon Linux 2023 AMI

ami-069d73f3235b535bd (64비트(x86)) / ami-0e31d4dd8c30fd2a (64비트(Arm))

가상화: hvm ENA 활성화됨: true 루트 디바이스 유형: ebs

프리 티어 사용 가능

설명

Amazon Linux 2023 AMI 2023.1.20230705.0 x86_64 HVM kernel-6.1

아키텍처

AMI ID

64비트(x86)

ami-069d73f3235b535bd

확인된 공급 업체

▼ Summary

인스턴스 개수 정보

1

소프트웨어 이미지(AMI)

Amazon Linux 2023 AMI 2023.1.2...더 보기

ami-069d73f3235b535bd

가상 서버 유형(인스턴스 유형)

t2.micro

방화벽(보안 그룹)

새 보안 그룹

스토리지(볼륨)

1개의 볼륨 - 8GiB

프리 티어: 첫 해에는 월별 프리 티어 AMI에 대한 t2.micro(또는 t2.micro를 사용할 수 없는 리전의 t3.micro) 인스턴스 사용량 750시간, EBS 스토리지 30GiB, IO 2백만 개, 스냅샷 1GB, 인터넷 대역폭 100GB가 포함됩니다.

취소

인스턴스 시작

명령 검토

새 키 페어 생성

▼ 키 페어(로그인) 정보

키 페어를 사용하여 인스턴스에 안전하게 연결할 수 있습니다. 인스턴스를 시작하기 전에 선택한 키 페어에 대한 액세스 권한이 있는지 확인하세요.

키 페어 이름 - 필수

선택

새 키 페어 생성

키 페어 생성

키 페어 이름

키 페어를 사용하면 인스턴스에 안전하게 연결할 수 있습니다.

mykp

이름에는 최대 255개의 ASCII 문자를 포함할 수 있습니다. 앞 또는 뒤에 공백을 포함할 수 없습니다.

키 페어 유형

☒ RSA
RSA 암호화된 프라이빗 및 퍼블릭 키 페어

☐ ED25519
ED25519 암호화된 프라이빗 및 퍼블릭 키 페어

프라이빗 키 파일 형식

☒ .pem
OpenSSH와 함께 사용

☐ .ppk
PuTTY와 함께 사용

⚠

메시지가 표시되면 프라이빗 키를 사용자 컴퓨터의 안전하고 액세스 가능한 위치에 저장합니다. 나중에 인스턴스에 연결할 때 필요합니다. 자세히 알아보기

취소

키 페어 생성

네트워크 설정

▼ 네트워크 설정

정보

편집

네트워크 정보

vpc-02e1102171936e479

서브넷 정보

기본 설정 없음(가용 영역의 기본 서브넷)

퍼블릭 IP 자동 할당 정보

활성화

방화벽(보안 그룹) 정보

보안 그룹은 인스턴스에 대한 트래픽을 제어하는 방화벽 규칙 세트입니다. 특정 트래픽이 인스턴스에 도달하도록 허용하는 규칙을 추가합니다.

☒ 보안 그룹 생성

☐ 기존 보안 그룹 선택

다음 규칙을 사용하여 'launch-wizard-2'(이)라는 새 보안 그룹을 생성합니다.

☒ 에서 SSH 트래픽 허용
인스턴스 연결에 도움

위치 무관

0.0.0.0/0

☒ 인터넷에서 HTTPS 트래픽 허용
예를 들어 웹 서버를 생성할 때 엔드포인트를 설정하려면

☒ 인터넷에서 HTTP 트래픽 허용
예를 들어 웹 서버를 생성할 때 엔드포인트를 설정하려면

⚠

소스가 0.0.0.0/0인 규칙은 모든 IP 주소에서 인스턴스에 액세스하도록 허용합니다. 알려진 IP 주소의 액세스만 허용하도록 보안 그룹을 설정하는 것이 좋습니다.

Django deployment like a professional

12

깃 레포지터리 생성 후 커밋

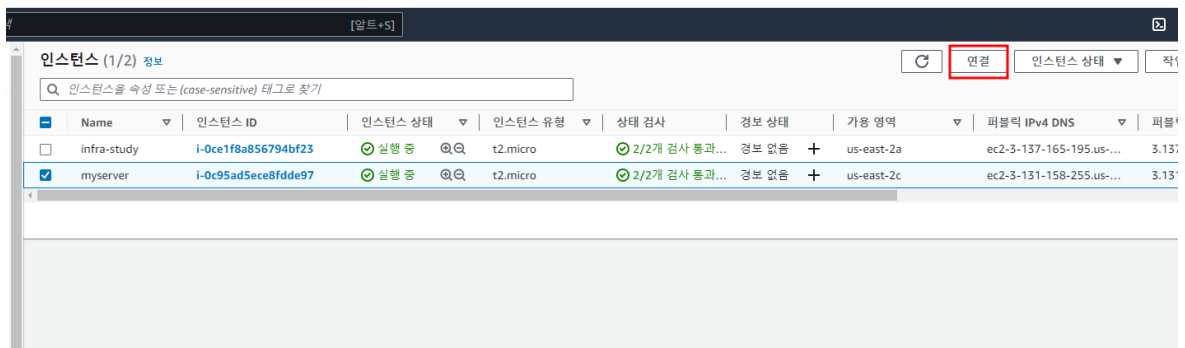
(생략)

인스턴스 연결

.pem 파일 위치 변경

```
User@DESKTOP-L657DDQ MINGW64 ~/Desktop/django-docker
$ ls
django-aws-deployment/ mykp.pem venv/
```

인스턴스 연결



EC2 > 인스턴스 > i-0c95ad5ece8fdde97 > 인스턴스에 연결

인스턴스에 연결 정보

다음 옵션 중 하나를 사용하여 인스턴스 i-0c95ad5ece8fdde97 (myserver)에 연결

EC2 인스턴스 연결



Session Manager

SSH 클라이언트

EC2 직렬 콘솔


인스턴스 ID

 i-0c95ad5ece8fdde97 (myserver)

1. SSH 클라이언트를 엽니다.
2. 프라이빗 키 파일을 찾습니다. 이 인스턴스를 시작하는 데 사용되는 키는 mykp.pem입니다.
3. 필요한 경우 이 명령을 실행하여 키를 공개적으로 볼 수 없도록 합니다.
 `chmod 400 mykp.pem`
4. 퍼블릭 DNS을(를) 사용하여 인스턴스에 연결:
 `ec2-3-131-158-255.us-east-2.compute.amazonaws.com`

예:

 `ssh -i "mykp.pem" ec2-user@ec2-3-131-158-255.us-east-2.compute.amazonaws.com`

 참고: 대부분의 경우 추정된 사용자 이름은 정확합니다. 하지만 AMI 사용 지침을 읽고 AMI 소유자가 기본 AMI 사용자 이름을 변경했는지 확인하십시오.

취소

sudo 명령 (WSL 사용)

```
$ sudo chmod 400 mykp.pem
[sudo] password for user: .
$ sudo chmod 400 mykp.pem
$ sudo ssh -i "mykp.pem" ec2-user@ec2-3-131-158-255.us-east-2.compute.amazonaws.com
```

ec2 환경에 명령어 (git, docker 설치) > <https://gist.github.com/npearce/6f3c7826c7499587f00957fee62f8ee9>

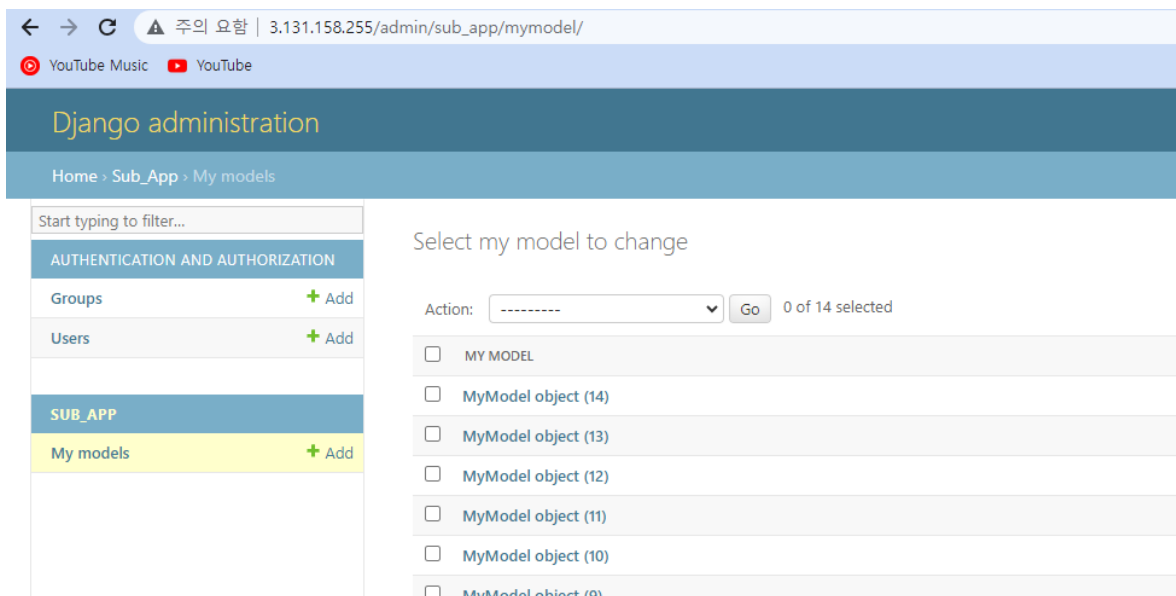
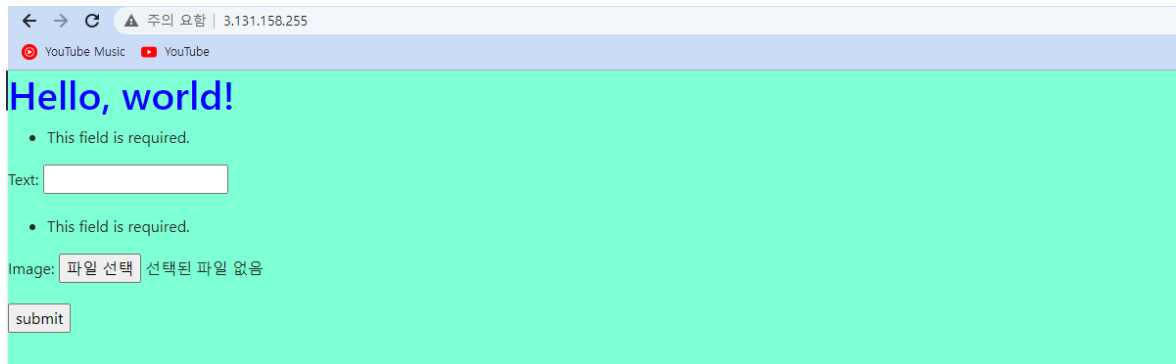
```
[ec2-user@ip-172-31-43-71 ~]$ sudo yum update -y
[ec2-user@ip-172-31-43-71 ~]$ sudo yum install git -y
[ec2-user@ip-172-31-43-71 ~]$ sudo yum install docker
[ec2-user@ip-172-31-43-71 ~]$ sudo service docker start
[ec2-user@ip-172-31-43-71 ~]$ sudo usermod -a -G docker ec2-user
[ec2-user@ip-172-31-43-71 ~]$ sudo chkconfig docker on
[ec2-user@ip-172-31-43-71 ~]$ sudo yum install -y git
[ec2-user@ip-172-31-43-71 ~]$ sudo reboot
$ sudo ssh -i "mykp.pem" ec2-user@ec2-3-131-158-255.us-east-2.compute.amazonaws.com
[ec2-user@ip-172-31-43-71 ~]$ sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$
[ec2-user@ip-172-31-43-71 ~]$ sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$
[ec2-user@ip-172-31-43-71 ~]$ sudo chmod +x /usr/local/bin/docker-compose
[ec2-user@ip-172-31-43-71 ~]$ docker-compose version
```

git clone 후 도커 컴포즈 실행

```
[ec2-user@ip-172-31-43-71 ~]$ git clone https://github.com/lwy210/django-aws-deployment.git
[ec2-user@ip-172-31-43-71 ~]$ cd django-aws-deployment/
[ec2-user@ip-172-31-43-71 ~]$ nano .env (이전에 작성한 .env 복붙 후 저장)
[ec2-user@ip-172-31-43-71 ~]$ docker-compose up --build
```

```
[ec2-user@ip-172-31-43-71 ~]$  
[ec2-user@ip-172-31-43-71 ~]$
```

EC2 public 주소로 접속



▼ 06 Connect domain and SSL to Django app on EC2 using Route53 and ELB



오류

django-aws-deployment-nginx-1 | 2023/07/09 11:27:01 [error] 29#29: *1 upstream prematurely closed connection while reading response header from upstream, client: 220.120.45.161, server: , request: "POST / HTTP/1.1", upstream: "http://172.18.0.2:8000/", host: "3.131.158.255", referer: "http://3.131.158.255/"



해결방법

EC2 > 보안 그룹 > sg-006d2b20b50d53d83 - SG for DB > 인바운드 규칙 편집

인바운드 규칙 편집

인바운드 규칙은 인스턴스에 도달하도록 허용한 수신 트래픽을 제어합니다.

인바운드 규칙	정보	유형	정보	프로토콜	정보	포트 범위	정보	소스	정보	설명 - 선택 사항	정보
보안 그룹 규칙 ID	sg-00f54847942126a05	PostgreSQL	TCP	5432	사용자 지정	Q	220.120.45.161/32				삭제
-		PostgreSQL	TCP	5432	사용자 지정	Q	172.31.43.71/32				삭제

규칙 추가

취소 변경 사항 미리 보기 규칙 저장