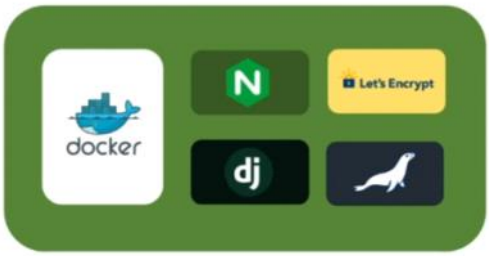
What is DOCKER?: Service Deployment

2021년 10월 14일 목요일 오후 9:59

• Why Docker? 서비스 배포로 들어가며

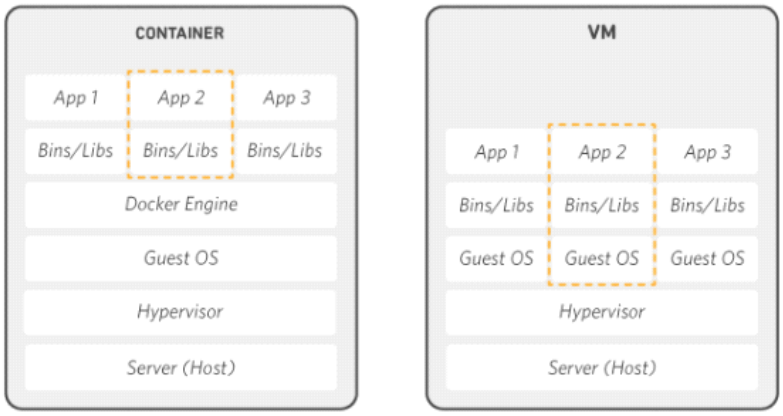
- 4개의 애플리케이션을 django container로 구성



- vultr 가상서버에 올려 서비스하는 것이 목표

• What is DOCKER?

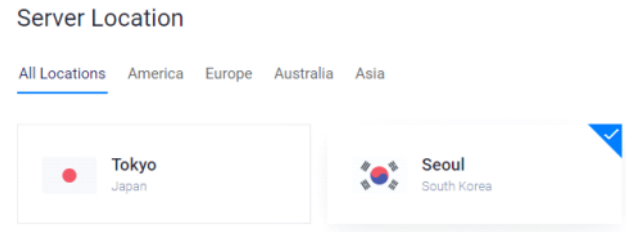
- Virtualization(가상화) 기반 컨테이너라는 개념을 사용함. 실제 OS사용하는 것과 거의 동일한 성능

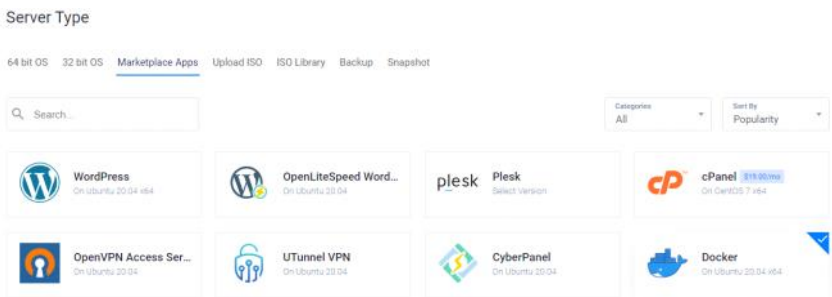


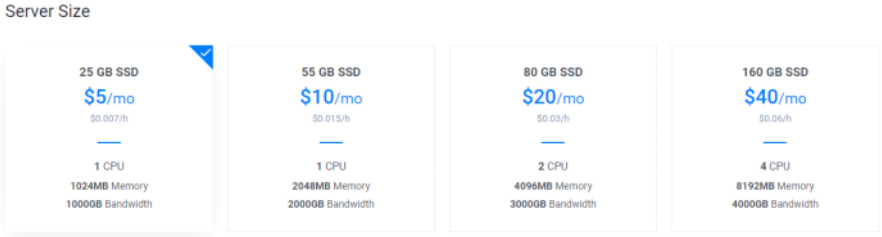
- 어느 곳에서든 동일한 환경에서 구동 가능.
- Develop Fast, Deploy Fast, Build Fast, Ship Fast, Maintain Fast
- Docker 없이 배포하는 경우, Deploy Environment Setting
  OS version match, Install required programs, Install python, Install library, Set evn PATH, Deploy Test....
- Docker 활용 시 초기 셋팅하여 Image형태로 만들어 여러개의 Container로 구축하면 추가 설정이 불필요함.

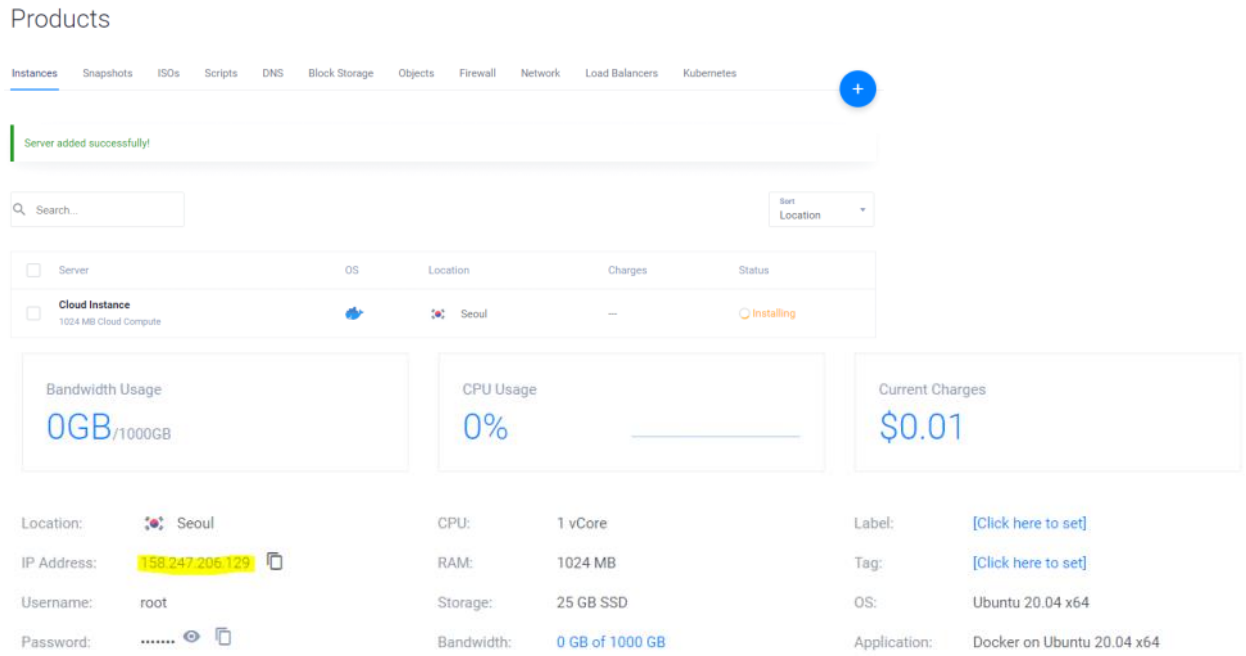
• VPS 대여

- Vultr를 통해 VPS, 갓아 사설 서버 대여 및 접속
- VPS: Virtual Private Server
- Vultr 가입 후 Instance 생성
- 

- 

- 

- 



- 서버 접근 테스트

```
C:₩Users₩DDubi>ssh root@158.247.206.129
The authenticity of host '158.247.206.129 (158.247.206.129)' can't be established.
ECDSA key fingerprint is SHA256:tK15UOlNePPAheqD3k8BrlQMsSc8Oa6OJhL8Lxx2sOI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '158.247.206.129' (ECDSA) to the list of known hosts.
```

```
root@158.247.206.129's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-84-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu 14 Oct 2021 02:25:00 PM UTC

  System load:  0.05              Processes:               147
  Usage of /:   13.6% of 23.41GB  Users logged in:         0
  Memory usage: 22%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for enp1s0:  158.247.206.129


59 updates can be applied immediately.
32 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


root@vultr:~#
```

- docker 설치 확인

```
root@vultr:~# docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and
                           default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Build with BuildKit (Docker Inc., v0.6.1-docker)
```

```
Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Build with BuildKit (Docker Inc., v0.6.1-docker)
  config      Manage Docker configs
```

-
```
root@vultr:~# docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@vultr:~#
```

Docker Container, Image

2021년 10월 15일 금요일 오후 6:49

• Docker GUI Portainer 컨테이너 생성

- Docker를 GUI 환경에서 실행할 수 있게 해주는 Portainer를 Docker hub 으로부터 이미지를 내려받아 구동
- https://hub.docker.com/r/portainer/portainer-ce
- https://docs.portainer.io/v/ce-2.9/start/install/server/docker/linux

Deployment

First, create the volume that Portainer Server will use to store its database:

```
docker volume create portainer_data
```

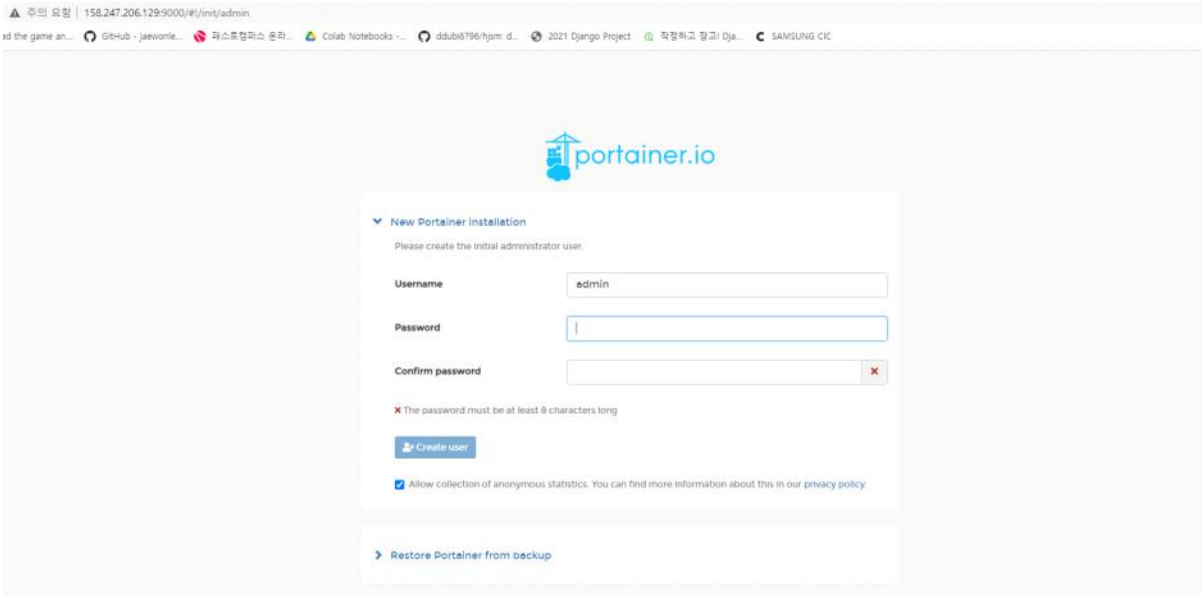
Then, download and install the Portainer Server container:

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer \
    --restart=always \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data \
    portainer/portainer-ce:latest
```

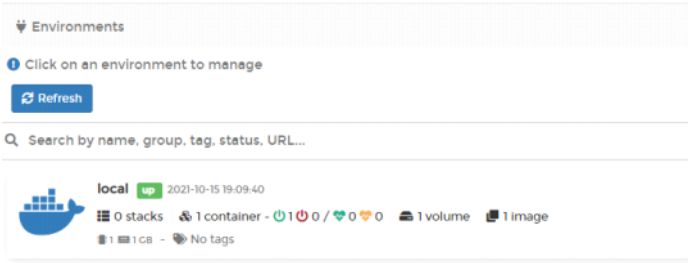
docker run -d -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest

```
root@vultr:~# docker volume create portainer_data
portainer_data
root@vultr:~# docker run -d -p 9000:9000 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock
-v portainer_data:/data portainer/portainer-ce:latest
Unable to find image 'portainer/portainer-ce:latest' locally
latest: Pulling from portainer/portainer-ce
7721cab3d696: Pull complete
0645e7e2a110: Pull complete
6329543ecfce: Pull complete
Digest: sha256:76ff22486bcd3713631b5f317efcb69e707a122fe96ffcc0589cf2d3e8e6b890
Status: Downloaded newer image for portainer/portainer-ce:latest
606aeaaf2eaa85f77381e60a3a94f6921e22019e5af7549863f884bbf4eef86f
```

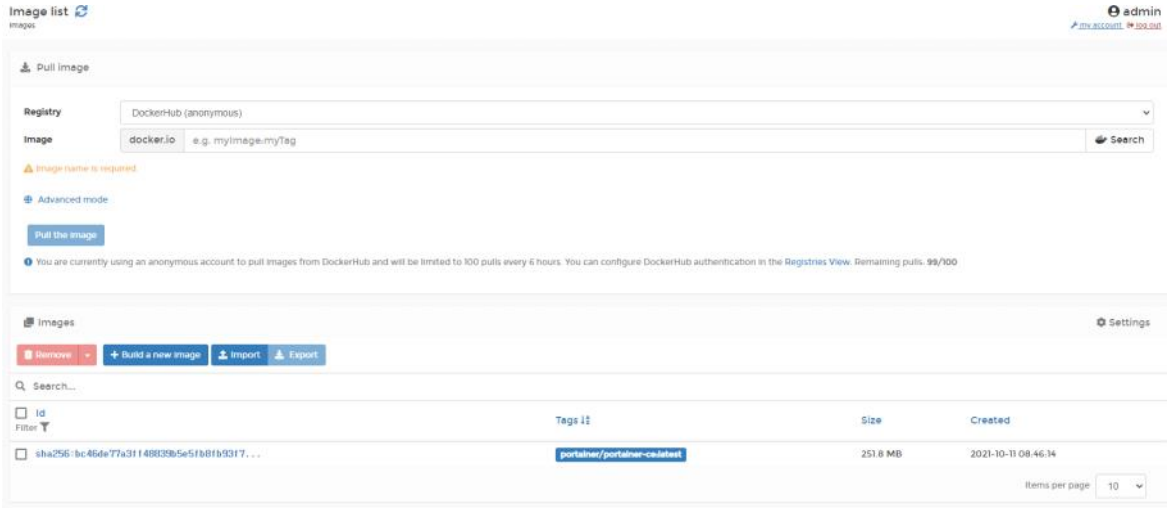
- 서버에 설치된 portainer 접근



- 컨테이너 구동 확인



- 서버 내 설치된 이미지 리스트 확인

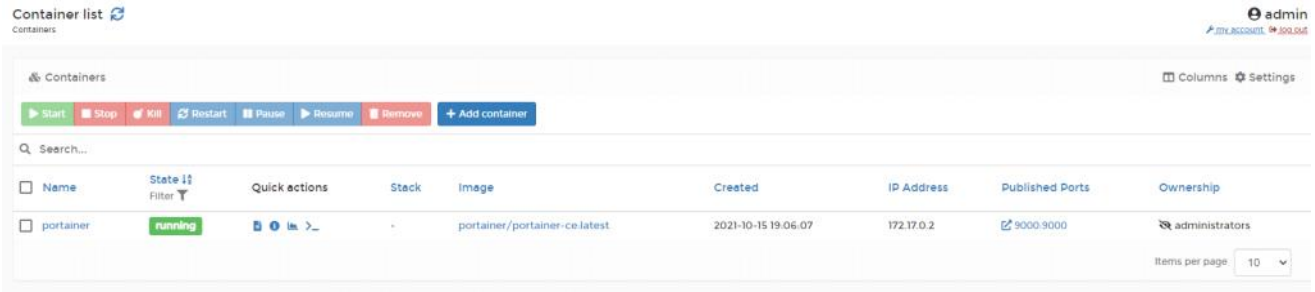


• Port의 이해 그리고 Nginx 컨테이너 생성

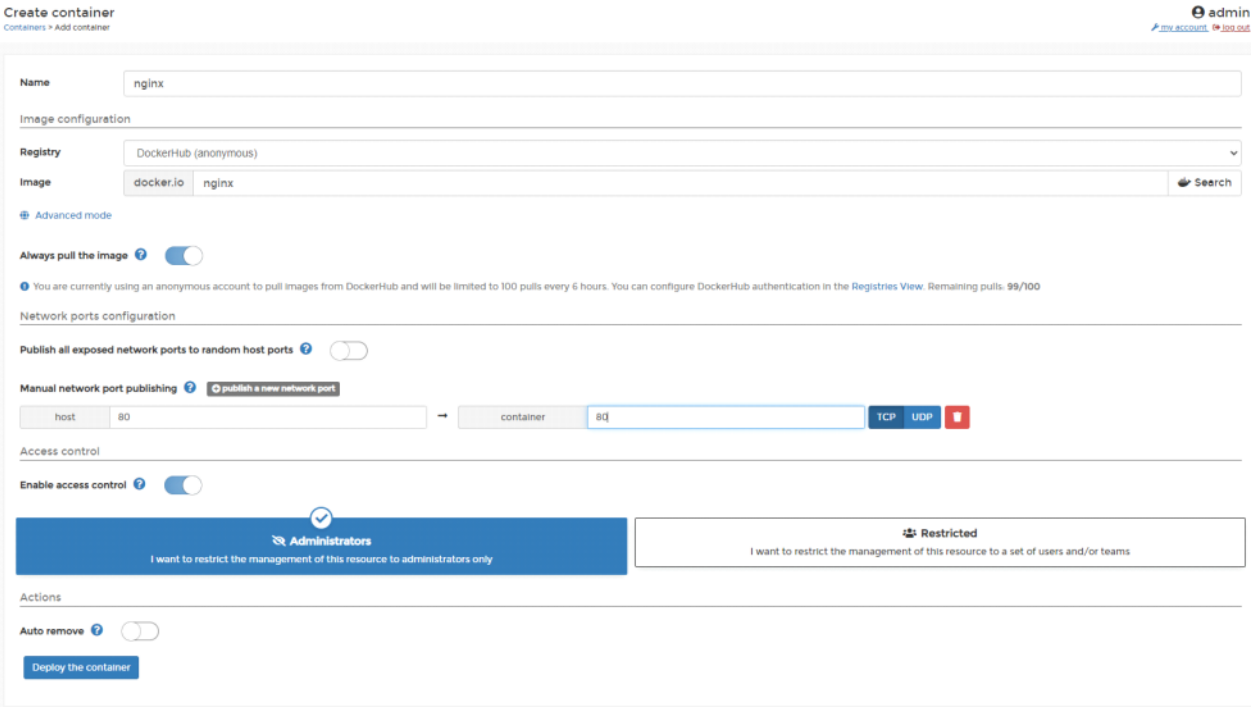
- Port: 소프트웨어에서는 네트워크 서비스나 특정 프로세스를 식별하는 논리 단위. 각 포트는 번호로 구별되며 이 번호를 포트 번호라고 한다. 포트 번호는 IP 주소와 함께 쓰여 해당하는 프로토콜에 의해 사용
- 본 프로젝트에서 vultr 서버(158.247.206.129)의 9000번 포트와 portainer를 9000번 포트로 연결해주었음.

- Nginx 컨테이너 추가하기

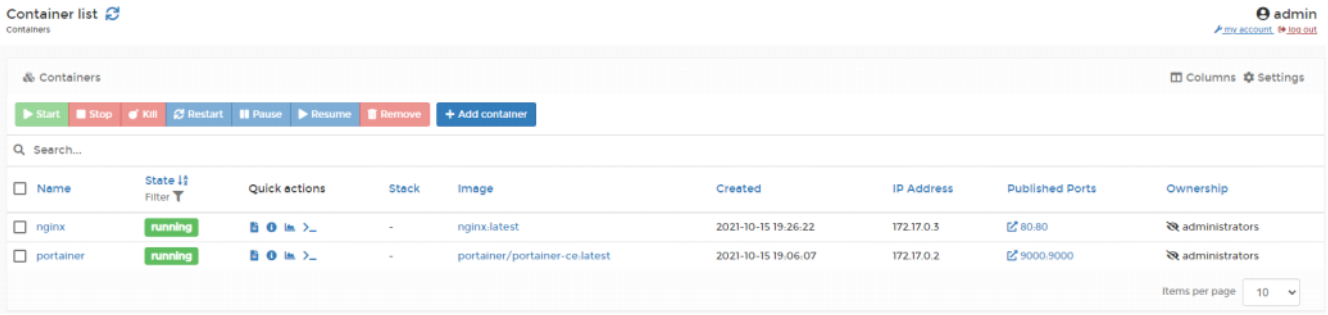
  portainer -> container list -> [Add container] 버튼 클릭하여 컨테이너 추가



- 아래와 같이 설정 후 [Deploy the container]



- container list에 80번 포트에 nginx이미지로 컨테이너가 생성됨



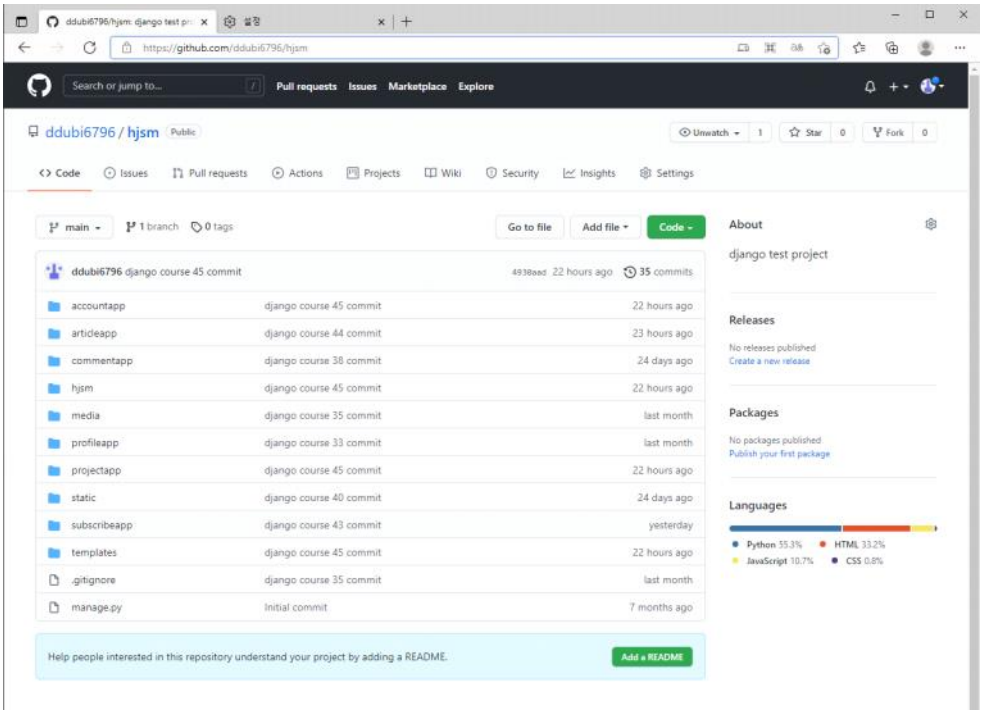
-

-

• django 소스코드 Github 업로드

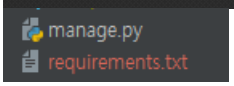
- Django container를 Vultr서버에 업로드 하기 위한 과정
  a. Github에 django 소스 업로드
  b. Dockerfile 작성
  c. Image 빌드
  d. Container 실행

  ○ github에 소스 업로드



- 프로젝트에 필요한 라이브러리 정보 가져오기(동일한 가상 환경을 구축하기 위함)

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>pip freeze > requirements.txt
```




```
asgiref==3.3.1
beautifulsoup4==4.9.3
Django==3.1.7
django-bootstrap4==3.0.1
django-environ==0.4.5
importlib-metadata==2.1.1
Pillow==8.3.1
pytz==2021.1
soupsieve==2.2.1
sqlparse==0.4.1
zipp==3.5.0
```

- Dockerfile 작성

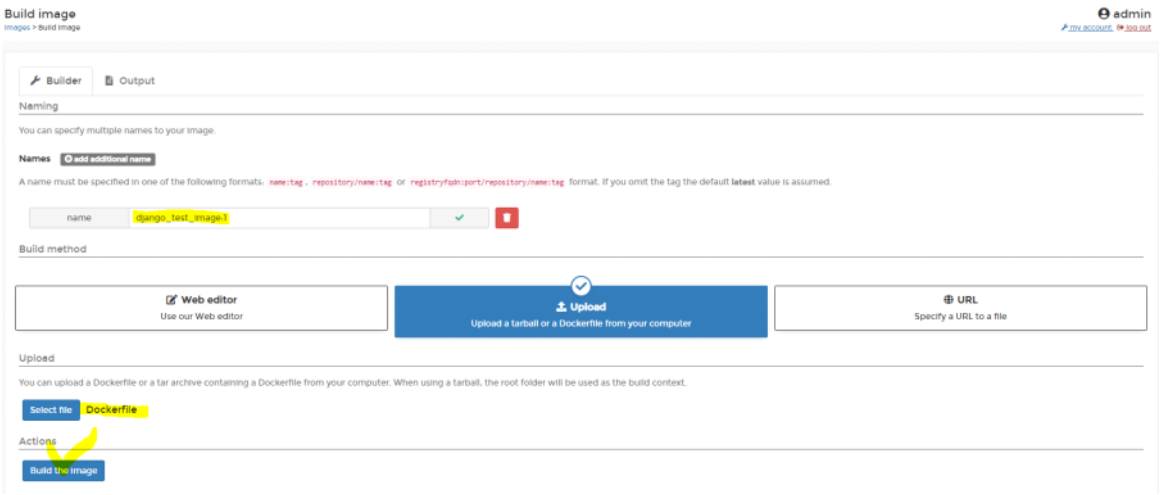
  Dockerfile: 컨테이너에 설치해야하는 패키지, 소스코드, 명령어, 환경변수설정 등을 기록한 하나의 파일

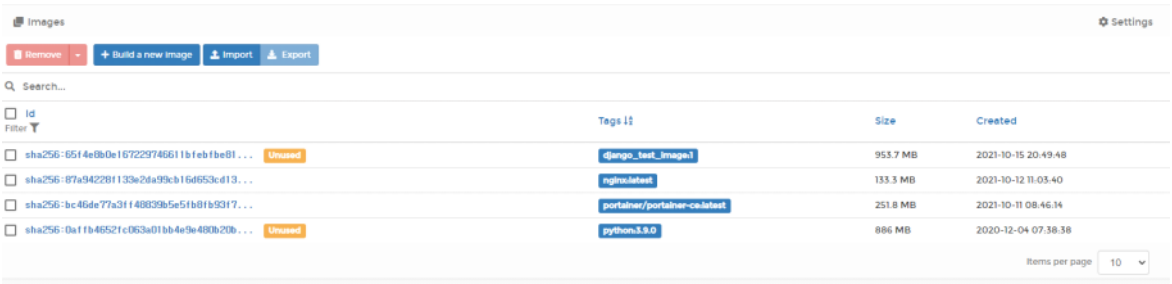
  - FROM : 생성할 이미지의 베이스가 될 이미지를 뜻합니다. 반드시 한번 이상 입력해야 합니다.
  - LABEL : 이미지에 메타데이터를 추가합니다. (나중에 원하는 조건의 컨테이너, 이미지 등을 쉽게 찾을 수 있도록 도와주기 때문에 기억해두는게 좋습니다)
  - RUN : 이미지를 만들기 위해 컨테이너 내부에서 command명령어를 실행합니다. (여기서 주의할 점은 설치과정에서 별도의 입력이 불가능하기 때문에 apache2를 설치할 때 뒤에 -y를 붙여줘야 합니다.
  - ADD : 파일을 이미지에 추가합니다. 여기서는 Dockerfile이 위치한 폴더에 test.html 파일을 가져와서 이미지의 /var/www/html 디렉터리에 추가합니다.
  - WORKDIR : 명령어를 실행할 디렉토리. 배시 셸에서의 cd 명령어와 동일한 기능을 합니다.
  - EXPOSE: 이미지에서 노출할 포트를 설정합니다.
  - CMD : 컨테이너가 시작될 때마다 실행할 명령어. Dockerfile에서 한번만 사용할 수 있습니다.

```
FROM python:3.9.0

WORKDIR /home/

RUN git clone https://github.com/ddubi6796/hjsm.git

WORKDIR /home/hjsm/

RUN pip install -r requirements.txt

RUN echo "SECRET_KEY=!3+u3gg^^)gh*^(!*pdz1w@15pw*9tom)icq(e=k$eth$w*io7" > .env

RUN python manage.py migrate

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

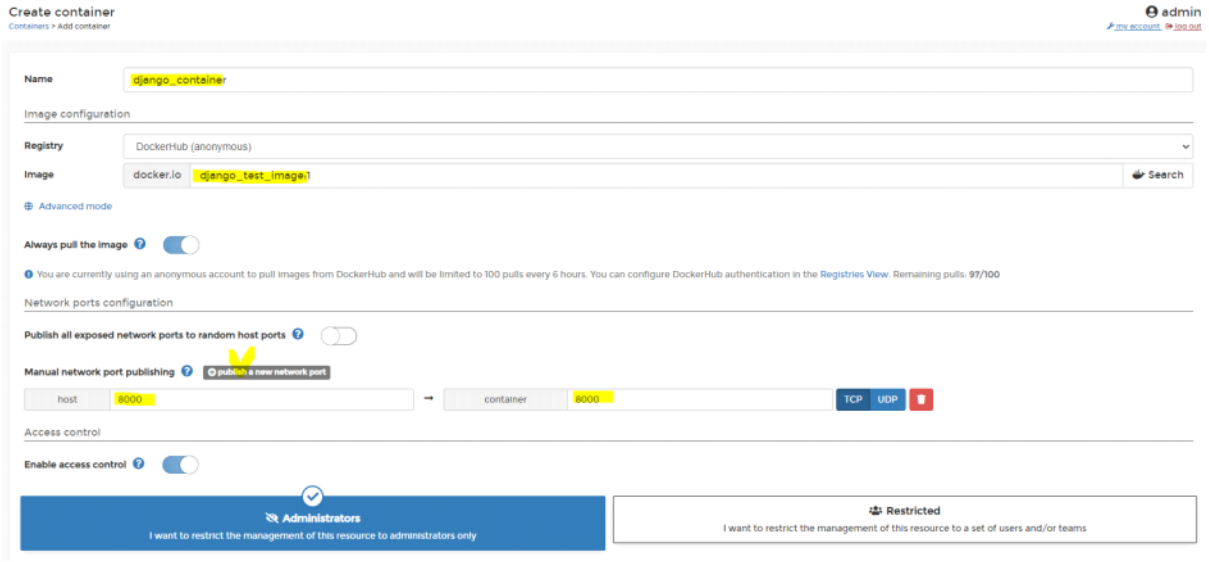
- env파일의 SECRET_KEY가 없으면 django가 정상 동작할 수 없으므로 임시로 프로젝트 내 env파일에서 SECRET_KEY를 복사하여 준다. (추후 변경 필수!)

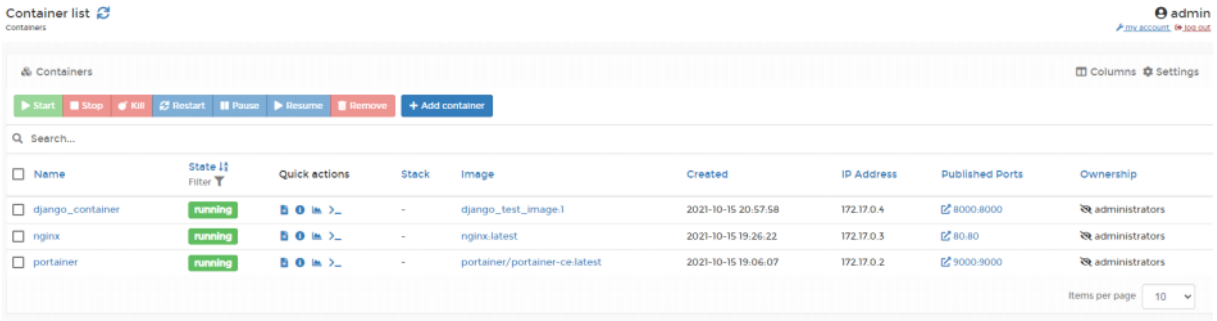
○ TEST IMAGE 빌드



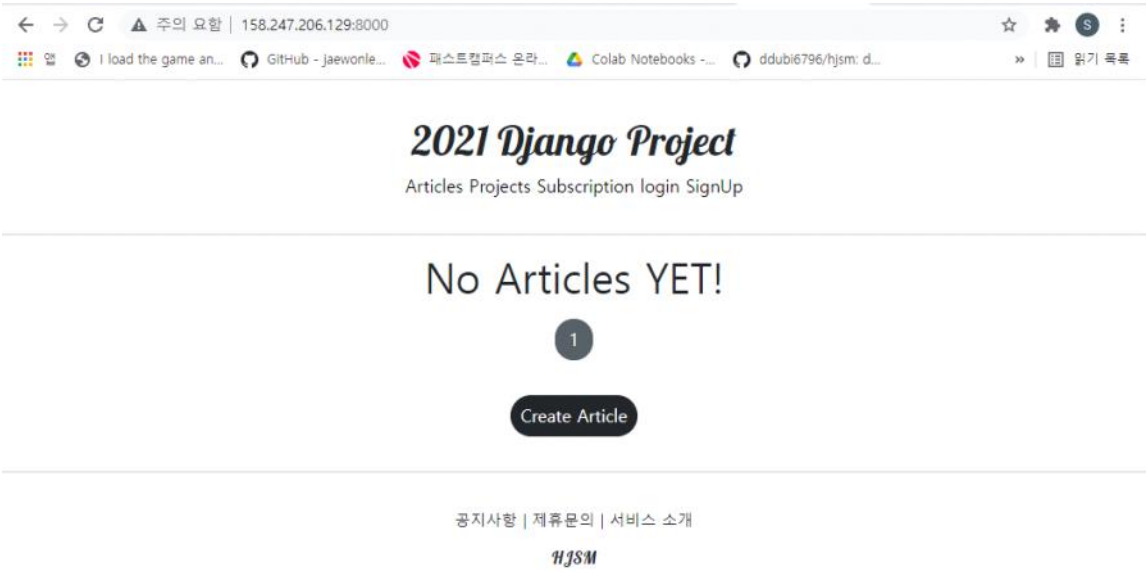


○ TEST Container 추가





- http://158.247.206.129:8000/ 접속 확인



- Gunicorn 을 사용해 지금까지 사용해왔던 runserver 명령어 대체 (python manage.py runserver는 배포 환경에서 사용 불가)

runserver

```
django-admin runserver [addrport]
```

Starts a lightweight development Web server on the local machine. By default, the server runs on port 8000 on the IP address **127.0.0.1**. You can pass in an IP address and port number explicitly.

If you run this script as a user with normal privileges (recommended), you might not have access to start a port on a low port number. Low port numbers are reserved for the superuser (root).

This server uses the WSGI application object specified by the **WSGI_APPLICATION** setting.

DO NOT USE THIS SERVER IN A PRODUCTION SETTING. It has not gone through security audits or performance tests. (And that's how it's gonna stay. We're in the business of making Web frameworks, not Web servers, so improving this server to be able to handle a production environment is outside the scope of Django.)

The development server automatically reloads Python code for each request, as needed. You don't need to restart the server for code

runserver

```
django-admin runserver [addrport]
```

Starts a lightweight development Web server on the local machine. By default, the server runs on port 8000 on the IP address **127.0.0.1**. You can pass in an IP address and port number explicitly.

If you run this script as a user with normal privileges (recommended), you might not have access to start a port on a low port number. Low port numbers are reserved for the superuser (root).

This server uses the WSGI application object specified by the **WSGI_APPLICATION** setting.

DO NOT USE THIS SERVER IN A PRODUCTION SETTING. It has not gone through security audits or performance tests. (And that's how it's gonna stay. We're in the business of making Web frameworks, not Web servers, so improving this server to be able to handle a production environment is outside the scope of Django.)

The development server automatically reloads Python code for each request, as needed. You don't need to restart the server for code changes to take effect. However, some actions like adding files don't trigger a restart, so you'll have to restart the server in these cases.

-> django 컨테이너 내부에 gunicon 라이브러리 설치하여 gunicon command로 대체

gunicorn? Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX

(Nginx 웹서버와 django 컨테이너를 연결해주는 interface)

WSGI? python으로 작성된 웹 어플리케이션과 python으로 작성된 서버 사이의 약속된 인터페이스 또는 규칙

- Gunicorn 설치

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>pip install gunicorn
Collecting gunicorn
  Downloading gunicorn-20.1.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 5.5 MB/s
Requirement already satisfied: setuptools>=3.0 in c:\users\ddubi\pycharmprojects\hjsm\venv\lib\site-packages (from gunicorn) (54.2.0)
Installing collected packages: gunicorn
Successfully installed gunicorn-20.1.0
WARNING: You are using pip version 21.0.1; however, version 21.3 is available.
You should consider upgrading via the 'c:\users\ddubi\pycharmprojects\hjsm\venv\scripts\python.exe -m pip install --upgrade pip' command.
```

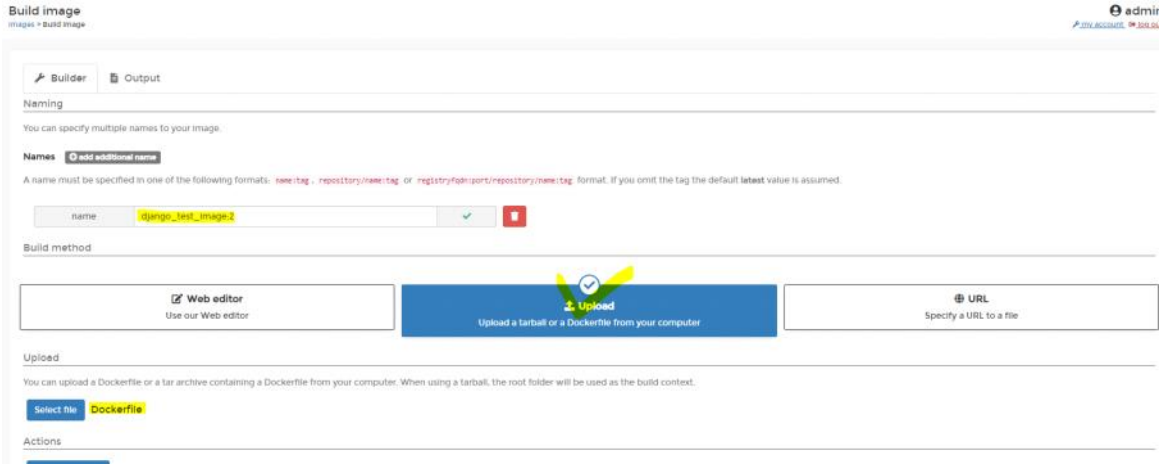
reauirements.txt 에 반영 후 git push

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>pip freeze > requirements.txt
```

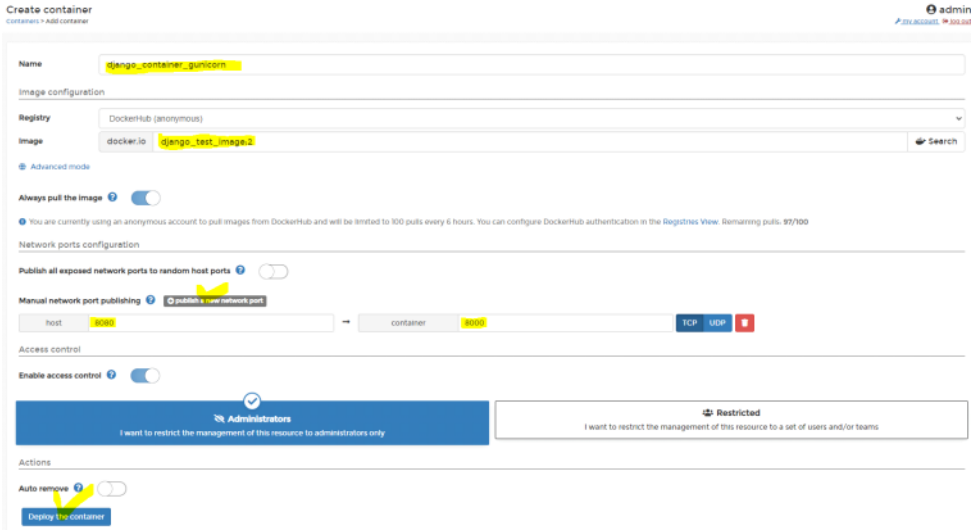
- Gunicorn command로 변경

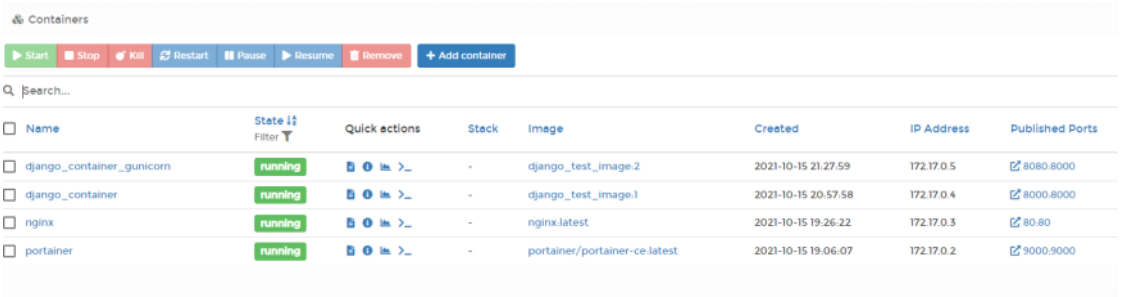
```
FROM python:3.9.0

WORKDIR /home/

RUN git clone https://github.com/ddubi6796/hjsm.git

WORKDIR /home/hjsm/

RUN pip install -r requirements.txt

RUN pip install gunicorn

RUN echo "SECRET_KEY=!3+u3gg^^)gh*^(!*pdz1w@15pw*9tom)icq(e=k$eth$w*io7" > .env

RUN python manage.py migrate

EXPOSE 8000

CMD ["gunicorn", "hjsm.wsgi", "--bind", "0.0.0.0:8000"]
```

- image 빌드

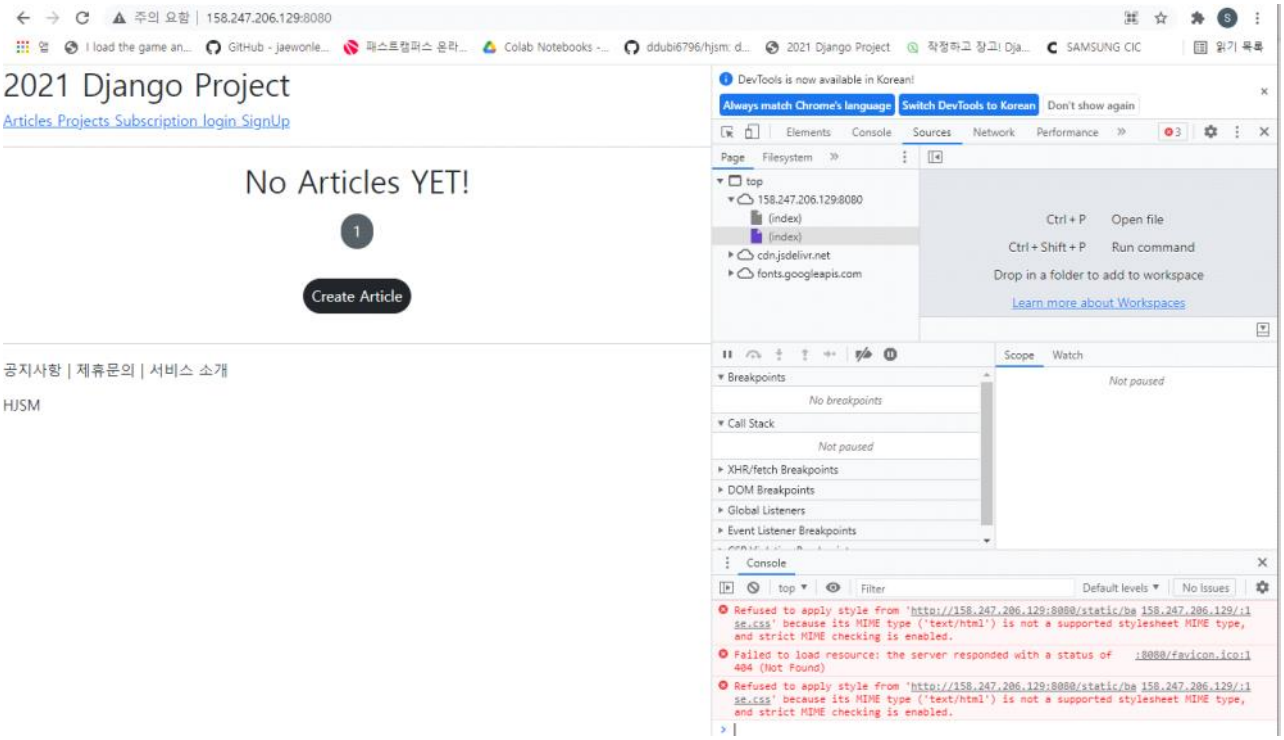


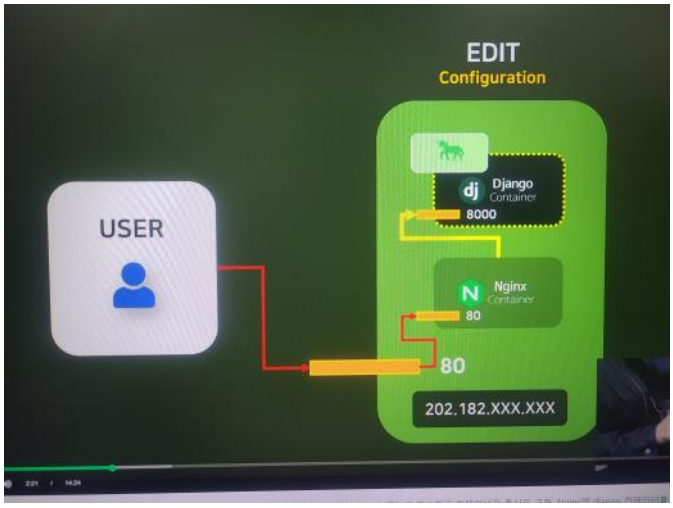
- container 추가





- 이슈: static 파일 not supported 에러 발생 -> django 컨테이너와 nginx컨테이너 연결을 위한 설정이 필요함.

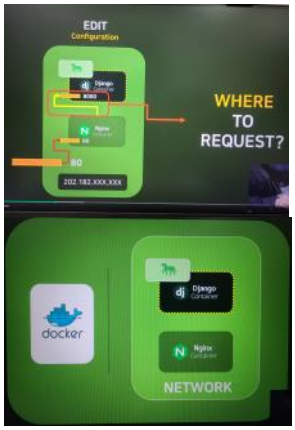



EDIT
Configuration
USER
Django
Container
8000
Nginx
Container
80
80
202.182.XXX.XXX

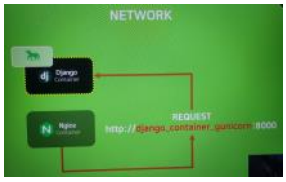
Docker Network, Volume

2021년 10월 15일 금요일 오후 9:32

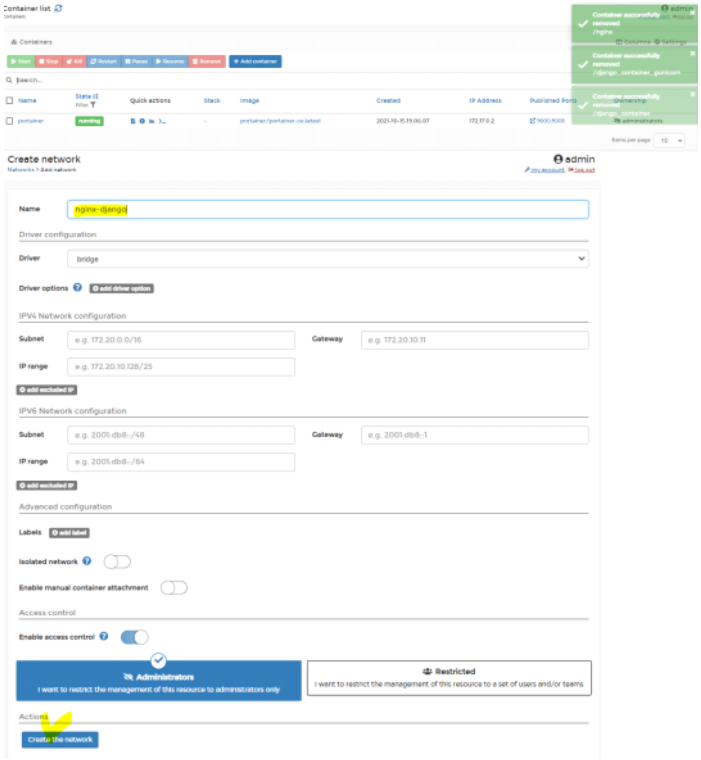
- Docker Network의 이해 및 구현
  - 네트워크: 여러개의 컨테이너를 하나의 묶어주는 것



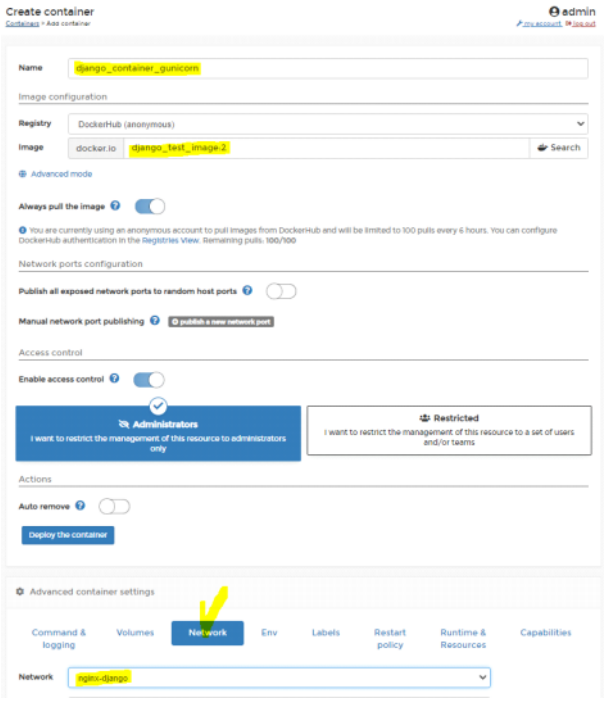
  - 네트워크 안에서는 컨테이너 name을 도메인처럼 사용할 수 있음



  - portainer를 제외한 컨테이너 delete 후, nginx-django네트워크 추가하기

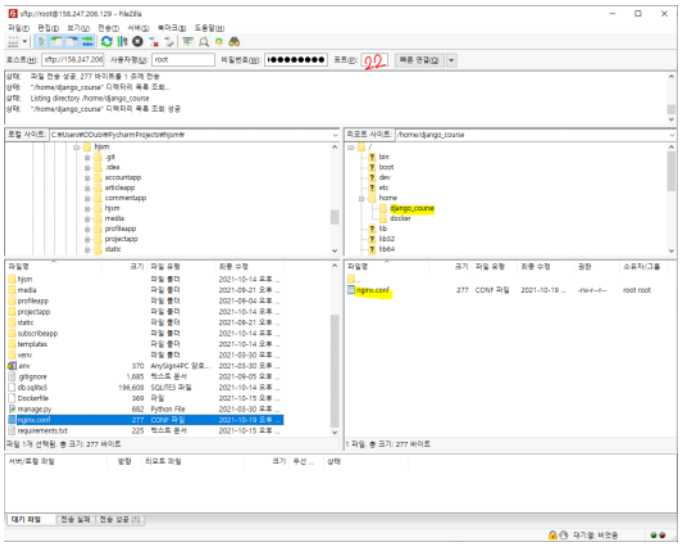


  - django_container_gunicorn 컨테이너 추가(nginx-django네트워크 설정)

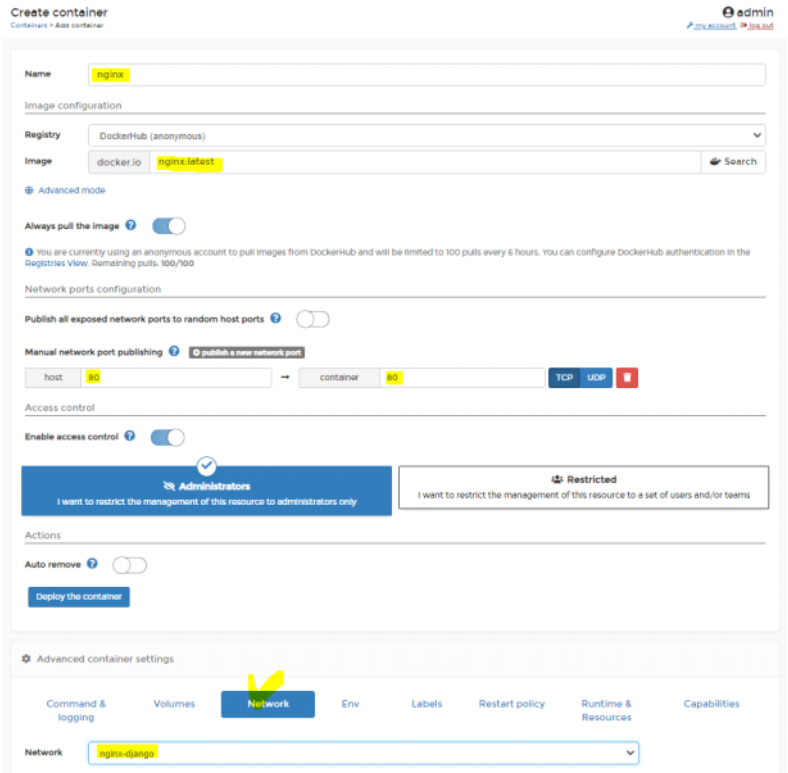


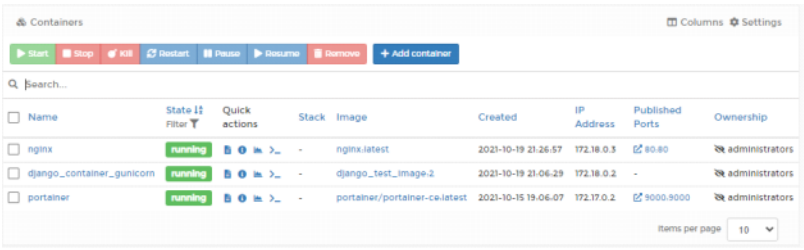
-
- nginx 설정파일 생성(nginx.conf) 후 filezilla로 가상서버에 설정파일 업로드
- https://gunicorn.org/#deployment

```
worker_processes auto;

events {
}

http {
  server {
    listen 80;

    location / {
        proxy_pass http://django_container_gunicorn:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
  }
}
```

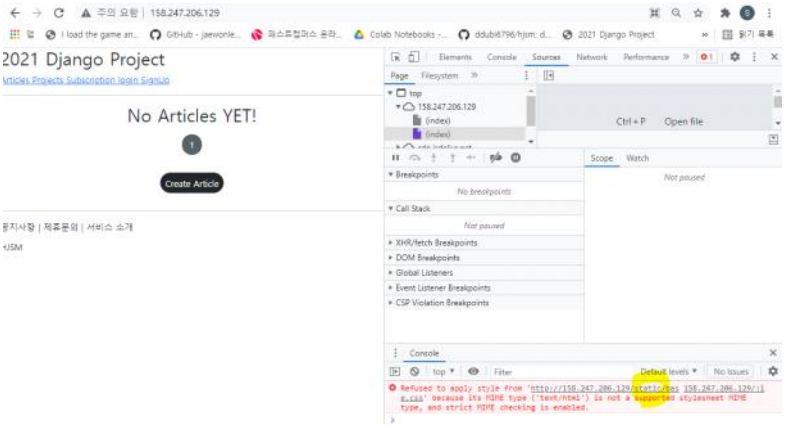


- nginx 컨테이너 생성
- network와 volume설정: django컨테이너와 동일한 네트워크로 설정, 가상서버와 컨테이너 내부를 연결





- 이제 :8000없이 http://158.247.206.129/ 기본포트(80)로 접근 가능! (nginx를 통해 django컨테이너로 request)
- but, 여전히 static 파일을 찾을 수 없어 화면이 깨짐



- static의 이해
  - 왜 gunicorn과 django가 static 파일을 제공하지 못하는가?
  - Client <-> Server(Static content) <-> Application(Dynamic content)
  - Client <-> nginx <-> django+gunicorn 의 구조. 즉, dynamic content이기 때문에 제공 불가

- 해결방안: 1) Collect static content from django container
  2) Synchronize static contents with nginx container

• Docker 컨테이너 생성 시, Collectstatic 명령을 통한 Static 파일 취합
  - python manage.py collectstatic
  - C:₩Users₩DDubi₩PycharmProjects₩hjsm₩staticfiles

```
(venv) C:\Users\DDubi\PycharmProjects\hjsm>python manage.py collectstatic

138 static files copied to 'C:\Users\DDubi\PycharmProjects\hjsm\staticfiles'.
```

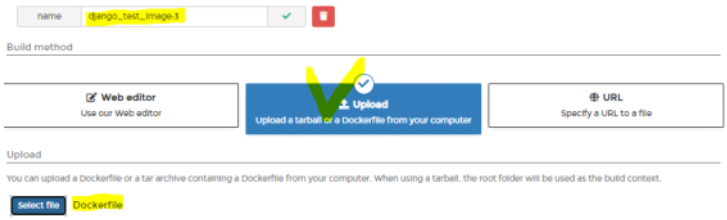
  - 초기 settings.py 내부에 설정해주었던 STATIC_ROOT 경로에 파일이 취합됨

```
STATIC_URL = '/static/'

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

STATICFILES_DIRS = [
    BASE_DIR / "static"
]
```

  - Dockerfile 내부에 collectstatic 명령어 추가(도커 컨테이너 생성 시 static파일 취합)

```
RUN echo "SECRET_KEY=!3+u3gg^^)gh*^(!*pdz

RUN python manage.py migrate

RUN python manage.py collectstatic

EXPOSE 8000
```

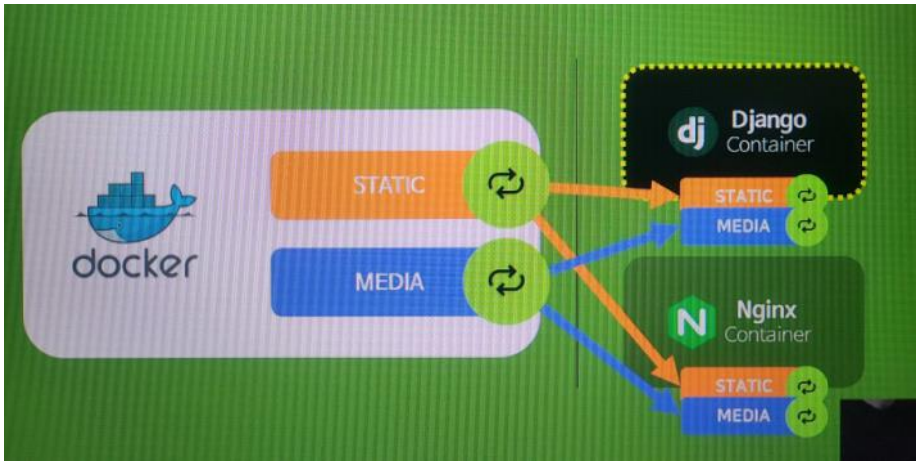
  - 이미지 생성



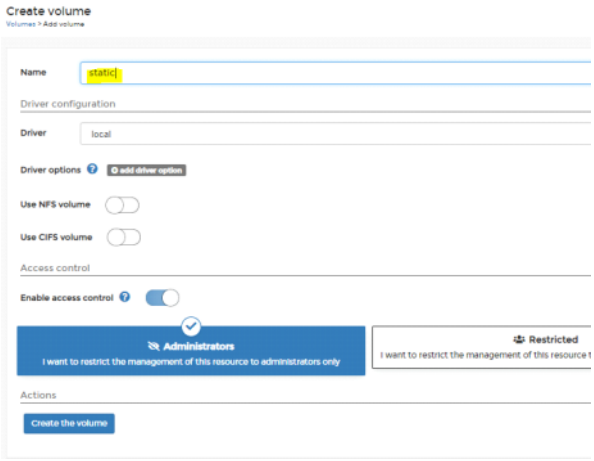

```
Step 9/11 : RUN python manage.py collectstatic

 ---> Running in 42100e4dd34f

138 static files copied to '/home/hjsm/staticfiles'.
Removing intermediate container 42100e4dd34f
 ---> 81a1f1ab2052
```

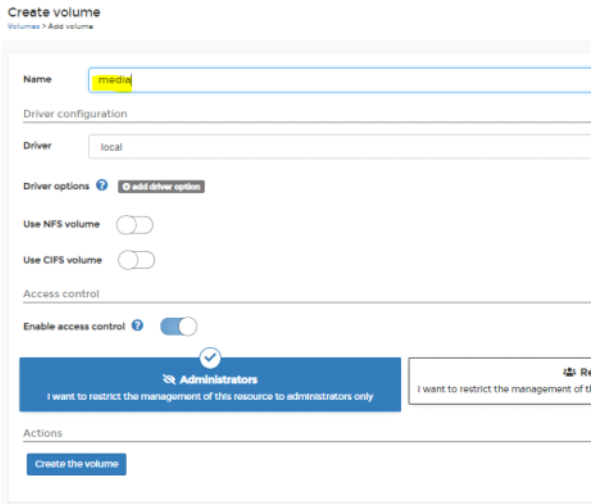
  - 생성한 docker 이미지 내 django의 static content를 nginx로 동기화 필요 -> Docker Volume

• Docker Volume
  - Bind Volume: Host server(Vultr)의 Nginx.conf와 Nginx Container의 Nginx.conf 연동 시 사용
  - Named Volume: docker 내부에 새로운 Volume을 생성하여 컨테이너와 연동 추후 연동한 컨테이너가 삭제되어도 데이터 유지

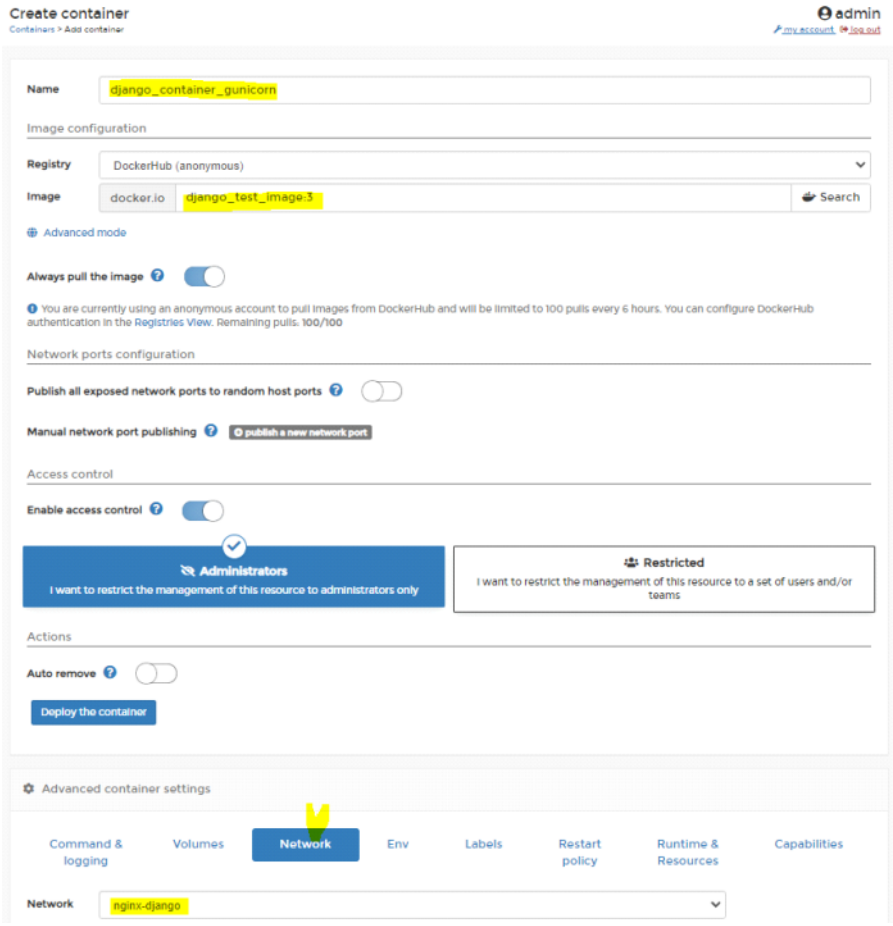


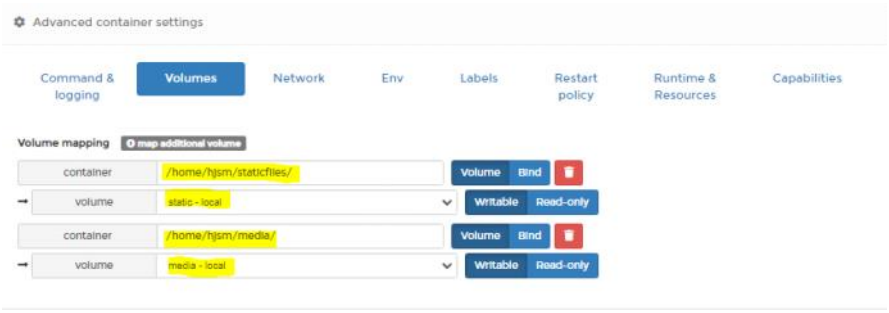
- Docker Volume 생성 및 Container 적용
  - static, media Volume 생성





    - django_container_gunicorn 컨테이너 재생성
      network: nginx-django
      volumes: static, media 볼륨 연결.
      container의 경로는 dockerfile의 `WORKDIR/home/hjsm/` + settings.py 내부에서 정의한 static, media root 경로





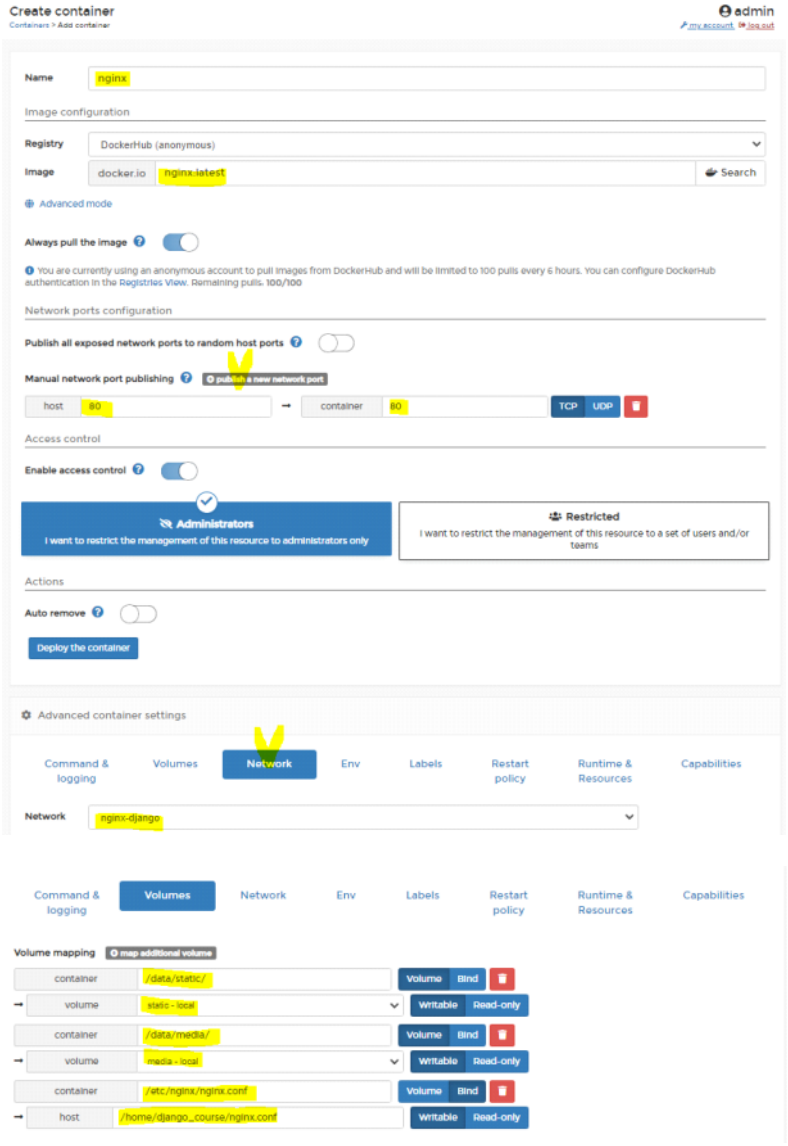
- nginx.conf 수정 후 filezilla 를 통해 서버에 업로드
  static, dynamic content 모두 전송
  alias에는 nginx 컨테이너 내 생성할 volume 경로
- (참고) MIME(Multipurpose Internet Mail Extensions) 이란?
  MIME은 다목적 인터넷 메일 확장이란 뜻으로 전자우편을 위하여 데이터 형식을 정의한 인터넷 표준 포맷

```
http {
  server {
    listen 80;

    include mime.types;

    location /static/ {
        alias /data/static/;
    }

    location /media/ {
        alias /data/media/;
    }

    location / {
        proxy_pass http://django_container_gunicorn:8000;
        proxy_set_header Host $host;
```

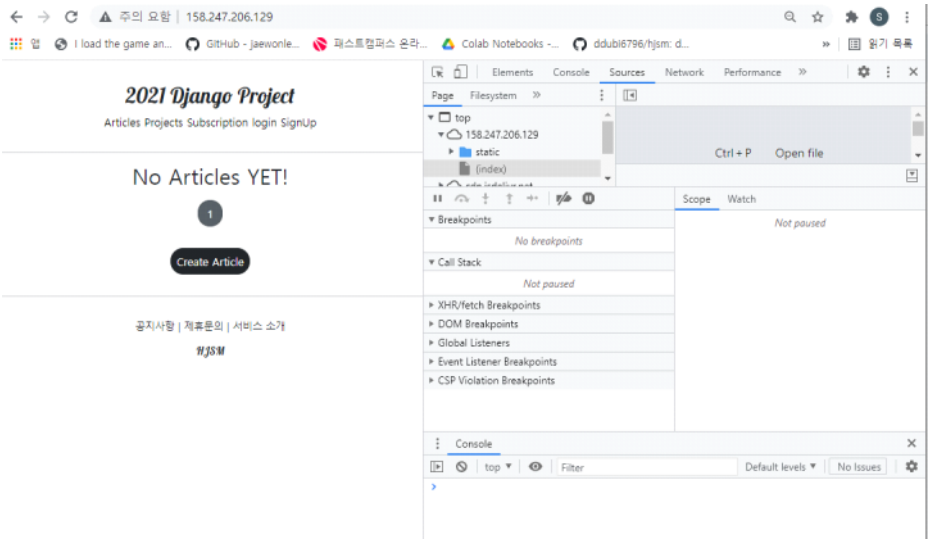
- nginx 컨테이너 재생성
  network: nginx-django

volumes: static, media 볼륨 연결 + nginx.conf bind연결

nginx컨테이너 내 static, media 볼륨 경로는 임의로 설정(mkdir)



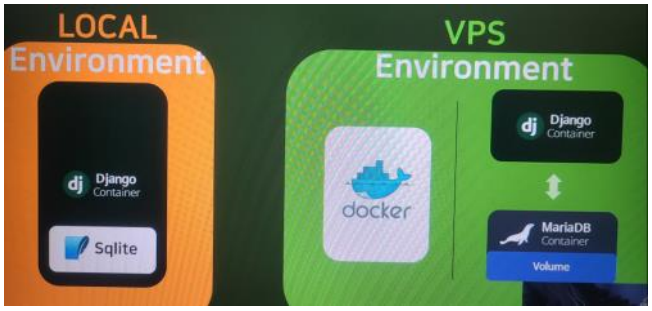
- 정상 구동 확인!



Local, Remote environment detachment

2021년 10월 19일 화요일 오후 10:27

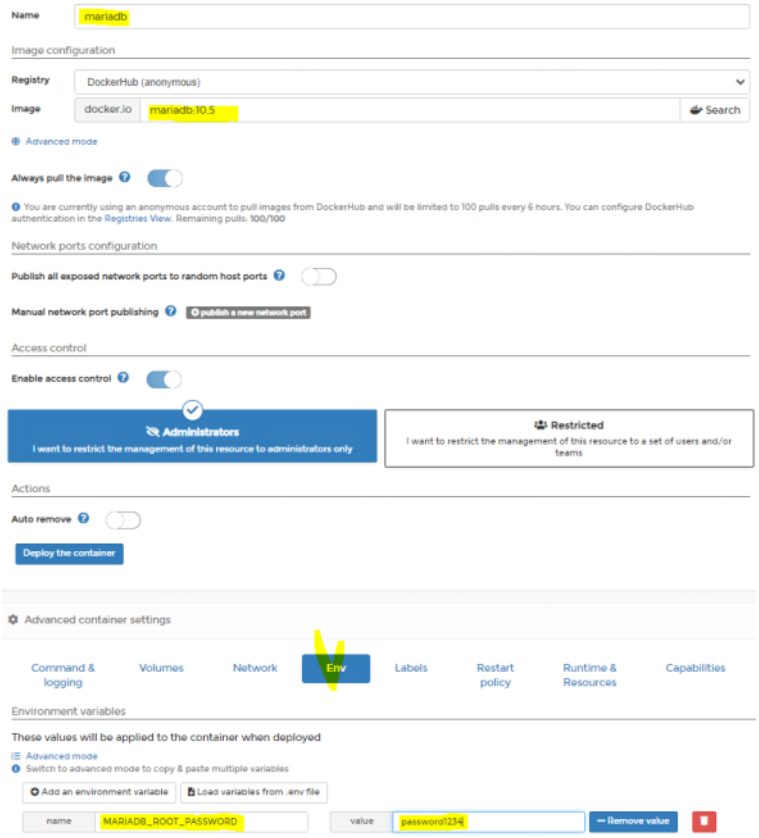
- MariaDB 컨테이너를 이용한 DB 분리
  - django컨테이너 내부에 DB가 있는 구조 -> 컨테이너 장애 시 전체 시스템에 이슈가 됨
  - DB를 별도의 컨테이너로 생성 후, volume을 활용하여 데이터가 지속적으로 유지되도록 함



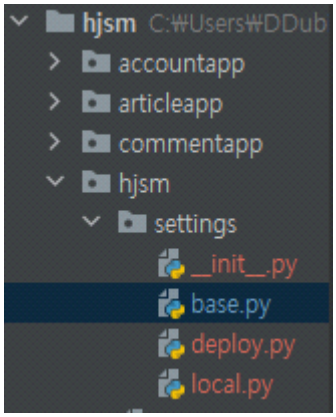
  - MariaDB 컨테이너 생성(구동 테스트)
    https://hub.docker.com/_/mariadb
    Starting a MariaDB instance is simple:

```
$ docker run -p 127.0.0.1:3306:3306  --name some-mariadb -e MARIADB_ROOT_PASSWORD=my-secret-pw -d mariadb:tag
```

    환경변수(Env) MARIADB_ROOT_PASSWORD 설정



- 개발/배포 설정 분리
  - settings 파이썬 패키지 추가 후, 하위로 settings.py 이동 후 base.py로 이름 변경
  - 로컬용 환경변수 설정을 위한 local.py 파일 생성
  - 서버용 환경변수 설정을 위한 deploy.py 파일 생성

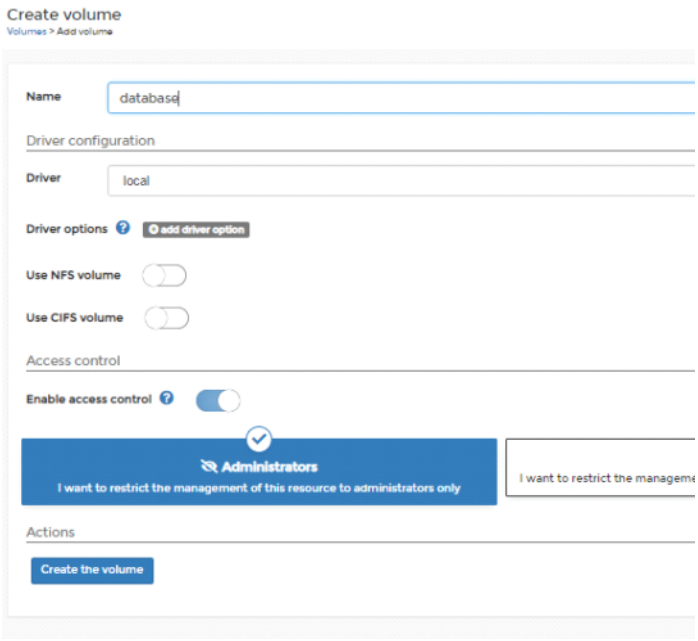


  - base.py로 부터 일부 설정을 local과 deploy로 분리
  - local.py

-
```python
from .base import *

env = environ.Env(
    # set casting, default value
    DEBUG=(bool, False)
)

# reading .env file
environ.Env.read_env(
    env_file=os.path.join(BASE_DIR, '.env')
)

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

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

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['*']


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```
- deploy.py
-
```python
from .base import *

env = environ.Env(
    # set casting, default value
    DEBUG=(bool, False)
)

# reading .env file
environ.Env.read_env(
    env_file=os.path.join(BASE_DIR, '.env')
)

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

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

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['*']


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django',
        'USER': 'django',
        'PASSWORD': 'password1234',
        'HOST': 'mariadb',   #mariadb의 컨테이너명
        'PORT': '3306',      #mariadb가 사용하는 port
    }
}
```
- base.py 내 BASE_DIR 변경: hjsm/settings 하위로 이동하였으므로
-
```python
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent.parent
```
- manage.py 내 settings 경로 변경(local 구동 시 hjsm/settings/local 설정파일 사용)
-
```python
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'hjsm.settings.local')
    try:
```

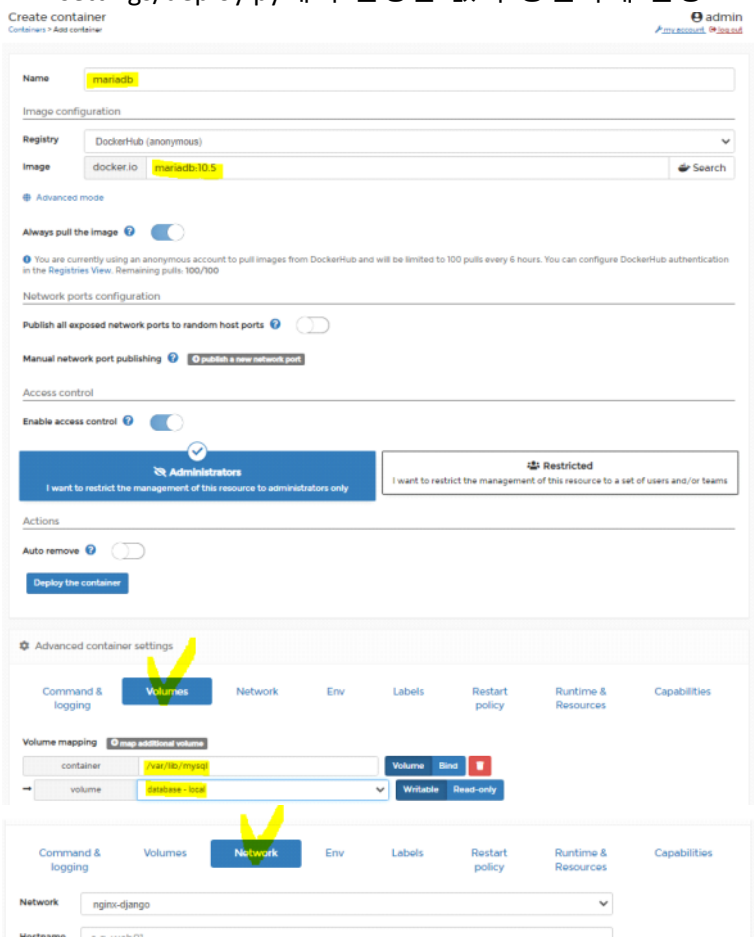
• MariaDB 컨테이너 설정 및 Django 연동

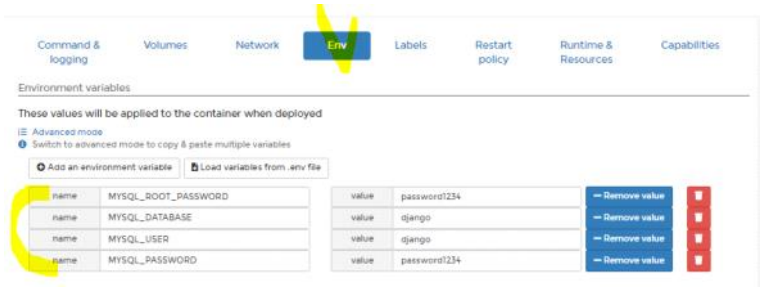
- database 볼륨 생성



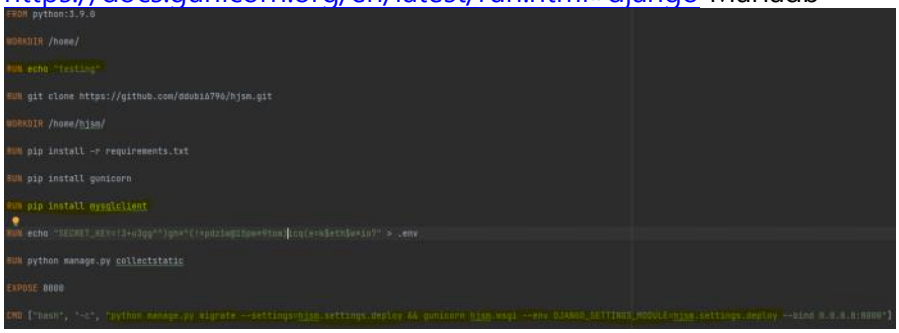
- mariadb 컨테이너 추가
- https://hub.docker.com/_/mariadb

- Env: settings/deploy.py에서 설정한 값과 동일하게 설정

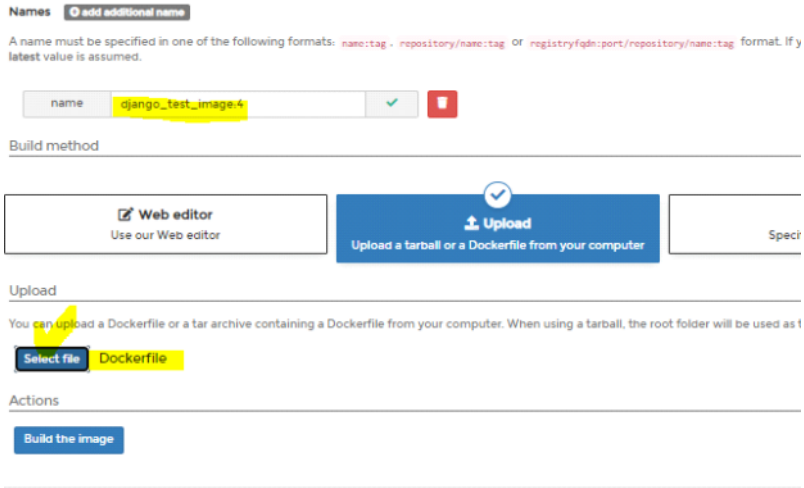




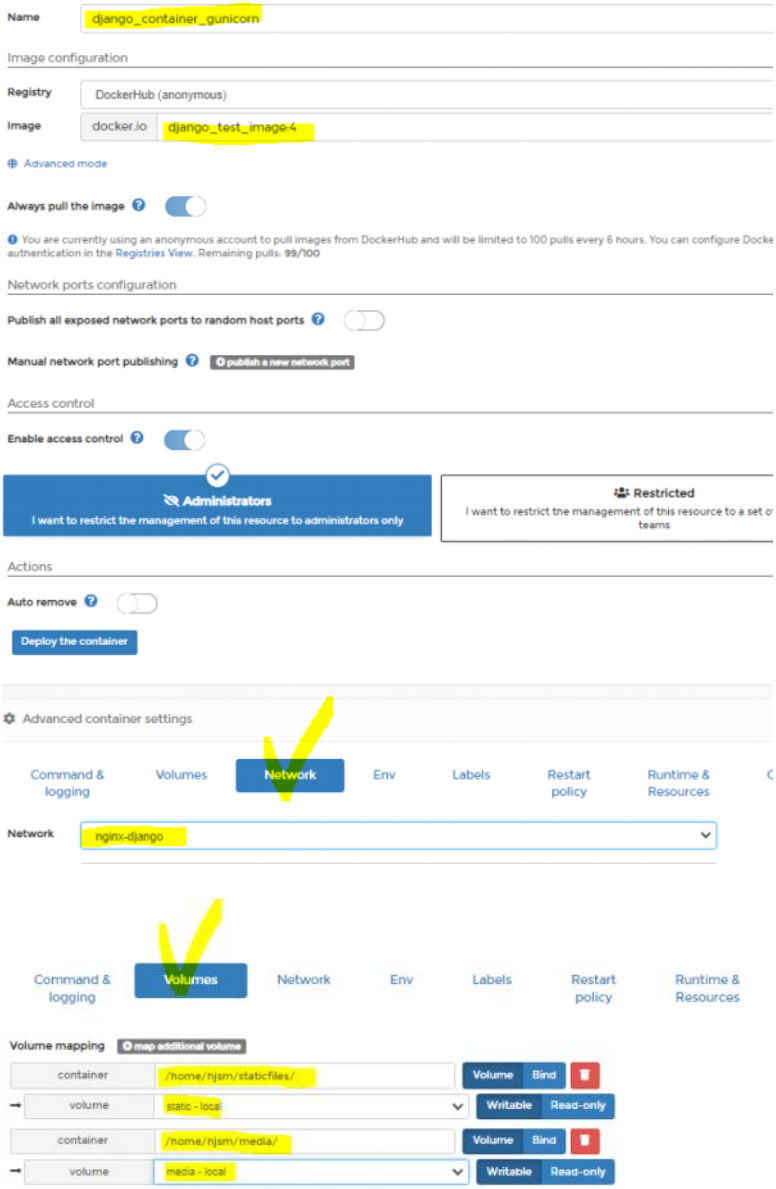
- Git Push
- Dockerfile 수정
- RUN python manage.py migrate 삭제
- https://docs.gunicorn.org/en/latest/run.html#django Mariadb



- django 이미지 생성



- django 컨테이너 생성



- 해당 컨테이너 삭제 시에도 mariadb의 데이터는 유지됨.
- (별도의 컨테이너에 named Volume을 사용했기 때문)

Docker Swarm, Stack, Secret

2021년 10월 20일 수요일 오후 11:02

• Container의 한계, Docker Stack의 이해
  - container만으로 해결할 수 없는 한계
    1) Repetitive configuration: 반복적으로 환경설정이 필요
    2) Container shutdown: 컨테이너가 꺼졌을때 발생하는 이슈
  - TOTAL STACK Settings
    -> Docker Stack: 다수의 컨테이너를 한번에 배포 - 같은 환경설정 공유(Docker Compose와 비슷),
    -> Service: scale out containers - 문제 발생 시, 설정 파일을 불러와 Automatic Reboot

• Docker Swarm의 이해
  - 컨테이너 오케스트레이션(Container Orchestration) 툴
  - 오케스트레이션? 여러 대의 서버와 여러 개의 서비스를 편리하게 관리해주는 작업. 스케줄링scheduling, 클러스터링clustering, 서비스 디스커버리 service discovery, 로깅logging, 모니터링monitoring 등의 작업을 수행

• Stack을 위한 yml 파일 작성
  - 현재 node를 manager node로 설정(swarm 모드 켜기)
    - ssh root@158.247.206.129
    - Password: 4Gc*9g5G+C2f4Y7m
    - cd home/django_course
    - docker swarm init

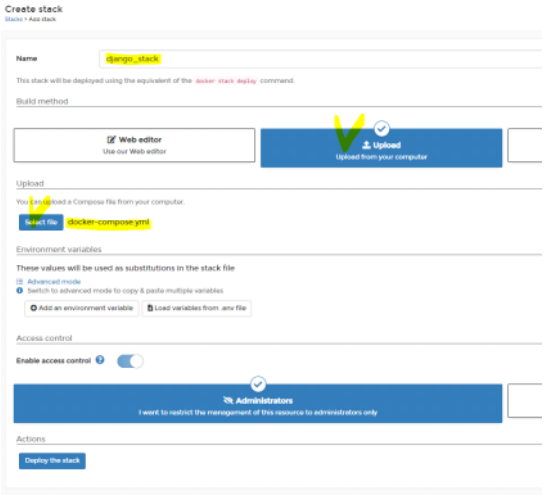
```
root@vultr:~# cd ..
root@vultr:/# cd home/django_course
root@vultr:/home/django_course# docker swarm init
Swarm initialized: current node (hpayjfx247l2dccycghwrbmza) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3ium6t8021o0xy7pbu795rqmnndz7cbghq20lguk8qo06xtzgq-ccxiizrptc7dfgbryo9umpc4k 158.
247.206.129:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

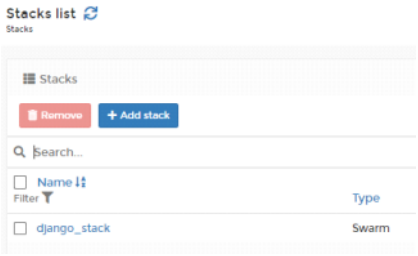
  - services, swarm 메뉴 활성화됨

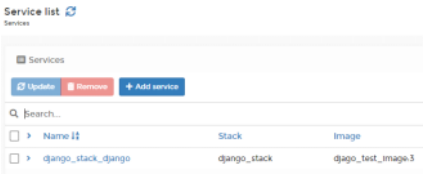
  - docker-compose.yml 파일 작성

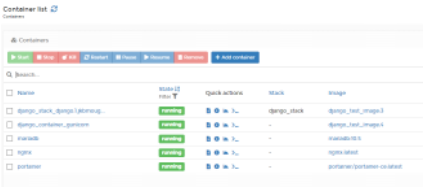
```
version: "3.7"
services:
  django:
    image: django_test_image:3
    ports:
      - 8000:8000
```

  - django_stack 스택 생성
  - stack과 service, container 생성 확인
  - http://158.247.206.129:8000/









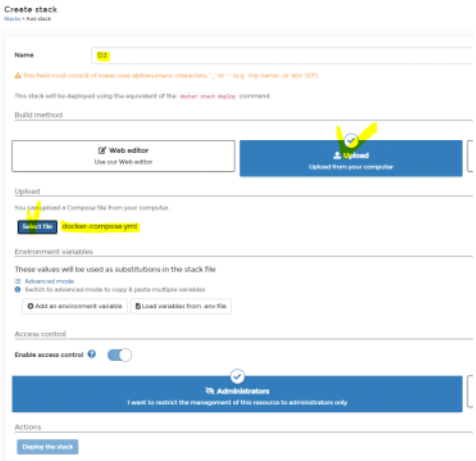
-> static 파일 필요; nginx 연동 필요

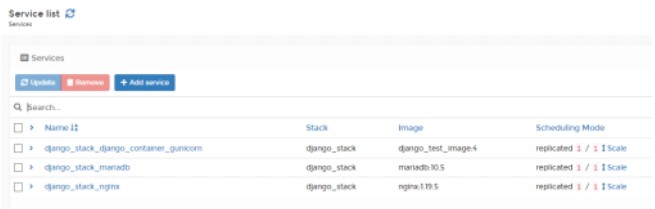
- 통합 yml파일 작성
  - nginx 추가: bind volume 설정 및 named volume 설정

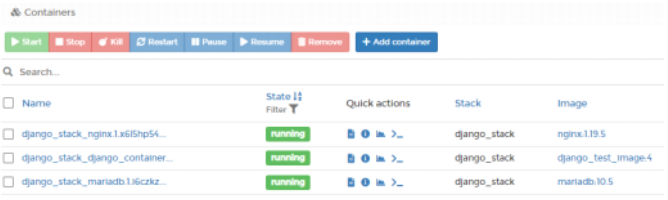
- django 서비스 추가 시 "django_container_gunicorn"으로 설정 - 설정해준 도메인명으로 생성 필요
- mariadb 서비스 추가 시 deploy settings에서 설정한 HOST 명으로 작성 필요

```yaml
version: "3.7"
services:
  nginx:
    image: nginx:1.19.5
    networks:
      - network
    volumes:
      - home/django_course/nginx.conf:/etc/nginx/nginx.conf
      - static-volume:/data/static
      - media-volume:/data/media
    ports:
      - 80:80
  django_container_gunicorn:
    image: django_test_image:4
    networks:
      - network
    volumes:
      - static-volume:/home/hjsm/staticfiles
      - media-volume:/home/hjsm//media
  mariadb:
    image: mariadb:10.5
    networks:
      - network
    volumes:
      - maria-database:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: password1234
      MYSQL_DATABASE: django
      MYSQL_USER: django
      MYSQL_PASSWORD: password1234

networks:
  network:

volumes:
  static-volume:
  media-volume:
  maria-database:
```

- stack 추가 및 확인

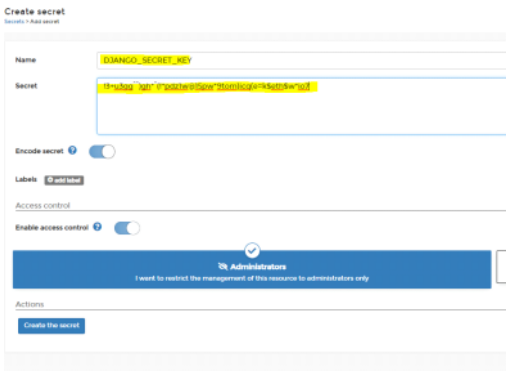


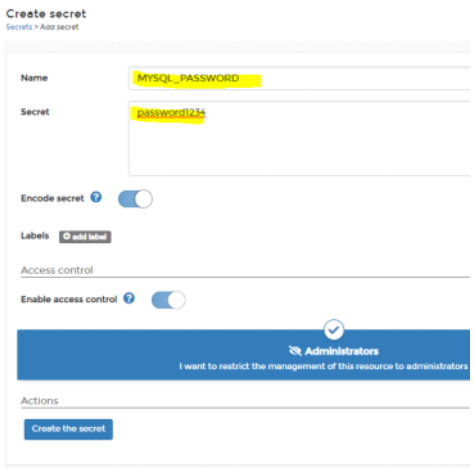


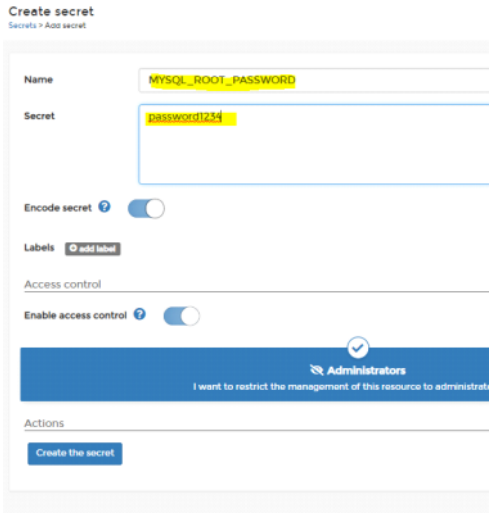


- Docker Secret을 이용한 보안
  - 보안 관련된 정보들을 소스코드 혹은 파일에 저장하는 방식이 아닌, Docker 내에서 별도로 관리하기 위해 사용함
  - dockerfile 내 보안 정보를 secret으로 생성 후 해당 항목 삭제

```
RUN pip install mysqlclient

RUN echo "SECRET_KEY=!3+u3gg^^)gh*^(!*pdz1w@15pw*9tom)icq(e=k$eth$w*io7" > .env

RUN python manage.py collectstatic
```







- docker-compose.yml 파일 변경 (secret 제공을 위함)
- 참고)
- **Docker Secrets**

  As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

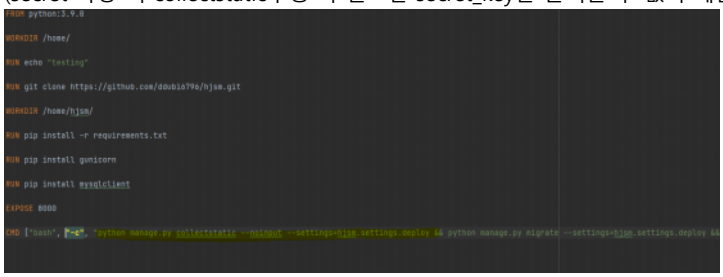
  ```
  $ docker run --name some-mysql -e MARIADB_ROOT_PASSWORD_FILE=/run/secrets/mysql-root -d mariadb:tag
  ```

```yaml
django_container_gunicorn:
  image: django_test_image:4
  networks:
    - network
  volumes:
    - static-volume:/home/hjsm/staticfiles
    - media-volume:/home/hjsm//media
  secrets:
    - MYSQL_PASSWORD
    - DJANGO_SECRET_KEY
mariadb:
  image: mariadb:10.5
  networks:
    - network
  volumes:
    - maria-database:/var/lib/mysql
  secrets:
    - MYSQL_PASSWORD
    - MYSQL_ROOT_PASSWORD
  environment:
    MYSQL_DATABASE: django
    MYSQL_USER: django
    MYSQL_PASSWORD_FILE: /run/secrets/MYSQL_PASSWORD
    MYSQL_ROOT_PASSWORD_FILE: /run/secrets/MYSQL_ROOT_PASSWORD

networks:
  network:

volumes:
  static-volume:
  media-volume:
  maria-database:

secrets:
  DJANGO_SECRET_KEY:
    external: true
  MYSQL_PASSWORD:
    external: true
  MYSQL_ROOT_PASSWORD:
    external: true
```

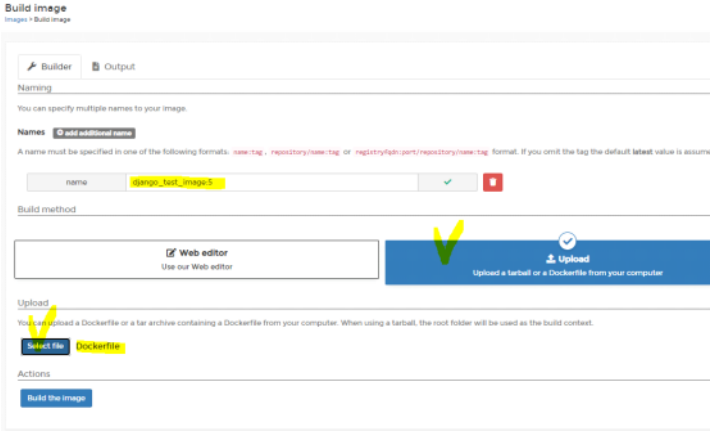
- deploy.py 수정 및 commit&push

```python
from .base import *

def read_secret(secret_name):
    file = open('/run/secrets/' + secret_name)
    secret = file.read()
    secret = secret.rstrip().lstrip()
    file.close()
    return secret

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

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['*']

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django',
        'USER': 'django',
        'PASSWORD': read_secret('MYSQL_PASSWORD'),
        'HOST': 'mariadb',
        'PORT': '3306',
    }
}
```

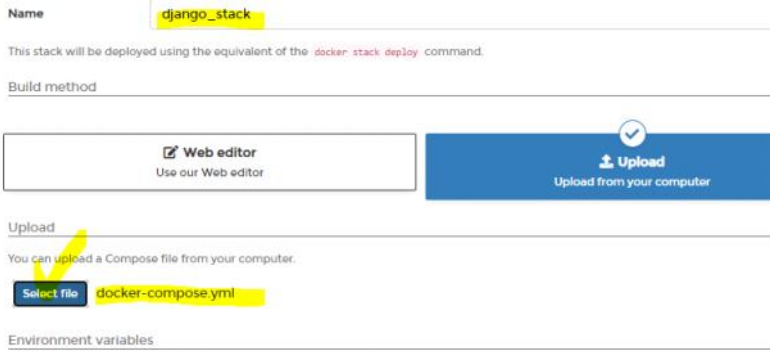
- dockerfile 수정
- collectstatic을 마지막으로 순서 변경
- (secret 사용 시 collectstatic수행 시 필요한 secret_key를 불러올 수 없기 때문



- django_test_image:5 이미지 생성



- django_stack 재생성



- 정상 구동 확인!