# 1. 배포를 위한 문서

## 1. Stacks

### Development Environment

- React 18.2.0
- Typescript 4.9.5
- Nodejs 20.10.0
- npm 10.2.3
- django 5.0.2 (python 3.12.2)
- Flask 2.2.5 (python 3.7.2)
- Maria DB 11.4.0 Alpha
- Visual Studio Code 1.85.2
- HeidiSQL 12.2
- Jupyter NoteBook 6.4.12

### Main Ribrary

- axios 1.6.6
- styled-components 6.1.8
- zustand 4.5.0
- react-query 5.17.19
- prettier 3.2.4
- djangorestframework 3.14.0
- gunicorn 21.2.0
- PyJWT 2.8.0
- TensorFlow 2.11.0
- Keras  2.11..0
- opencv-python 4.9.0.80
- numpy 1.21.6

### Deploy Management

- AWS EC2
- Ubuntu 20.04.6 LTS
- Jenkins (docker : latest)
- docker 25.0.0
- Nginx 1.18.0

### SCM(Software Configuration Management)

- Notion
- Figma
- Git
- Git Bash
- Git lab

### Issue Management

- Jira

### Community

- Notion

- Mattermost
- KaKaoTalk

## 2. Build And Distribute

### Nginx

```
# apt 업그레이드 및 nginx 설치
sudo apt update
sudo apt install nginx
```

```
host: i10c104.p.ssafy.io

#front(react)
localhost:3000=> /

#backend(Django)
localhost:8000=> /api/

#ai(Flask)
localhost:5000=> /ai/
```

### Jenkins

- docker, node, git 관련 플러그인 설치
- 깃 연동 설정(기준 repository push시-Webhooks설정, pipeline 자동 실행)
- docker image 자동 빌드
  - 1. image build- 2.check running image(delete)-3.run image 순서
  - parallel 문법을 사용하여 front, backend, ai image 모두 병렬적으로 진행
- JenkinsFile: /Jenkinsfile

## 3. Docker file

- 각 서버 간 자동 빌드를 위한 DockerFile 작성

### React

/fubao-app-pws/Dockerfile

```
# 가져올 이미지를 정의
FROM node:14

# 경로 설정하기
WORKDIR /app

# package.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json .

# 명령어 실행 (의존성 설치)
RUN npm install

# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .

# 각각의 명령어들은 한줄 한줄씩 캐싱되어 실행된다.
# package.json의 내용은 자주 바뀌진 않을 거지만
# 소스 코드는 자주 바뀌는데
# npm install과 COPY . . 를 동시에 수행하면
# 소스 코드가 조금 달라질때도 항상 npm install을 수행해서 리소스가 낭비된다.
# 3000번 포트 노출
EXPOSE 3000

# npm start 스크립트 실행
CMD ["npm", "start"]
```

### Django

/backend/Dockerfile

```
#p ython version
FROM python:3.12.1

# install vim
RUN apt-get -y update
RUN apt-get -y install vim

# set workdir
RUN mkdir -p /usr/src/app/backend
WORKDIR /usr/src/app/backend

COPY requirements.txt .

# install requirement.txt.
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# copy all project file
COPY . .

# migrations
RUN python manage.py collectstatic
RUN python manage.py makemigrations

EXPOSE 8000

# run server
CMD ["sh","-c","python manage.py migrate
&& gunicorn --bind 0.0.0.0:8000 backend.wsgi:application -t 240"]
```

### Flask

/fish_classification/Dockerfile

```
## custom dockerfile syntax. latest stable release of the version 1 syntax
# syntax=docker/dockerfile:1

# python 3.7 저용량
FROM python:3.7

# work directory
WORKDIR /code

# python library list 복사
COPY requirements.txt requirements.txt

RUN apt-get update

# python library 설치
RUN pip install -r requirements.txt

RUN apt-get -y install libgl1-mesa-glx

# flask port 노출
EXPOSE 5000

# app.py 복사
COPY . .

# python flask 실행
CMD ["gunicorn", "-b", "0.0.0.0:5000", "wsgi:app"]
```

## 4. Deployment Command

- docker image 생성 후 각 container bash 내에서 실행(이미 dockerFile에 포함)

### Front & Back End Server

- React

```
npm start
```

- Django

```
gunicorn --bind 0.0.0.0:8000 backend.wsgi:application -t 24
```

### AI Server

- Flask

```
gunicorn -b 0.0.0.0:5000 wsgi:app
```

## Web Server

- Nginx

  Nginx conf 파일 수정 후 Nginx 재 실행

```
sudo systemctl restart nginx
```

# 5. EC2 Setting

### Jenkins

```
# jenkins Docker 설치

# jenkins:latest image 다운
sudo docker run -itd \
-p 8080:8080 \
-p 50000:50000 \
-v /home/ubuntu/jenkins-data:/var/jenkins_home \
-v /$(which docker):/usr/bin/docker \
-v /var/run/docker.sock:/var/run/docker.sock \
--name jenkins jenkins/jenkins:lts
```

### NginX

- reverse-proxy.conf 설정

```
server {
        listen 80;
        listen [::]:80;
        server_name i10c104.p.ssafy.io;

        access_log /var/log/nginx/reverse-access.log;
        error_log /var/log/nginx/reverse-error.log;

        location / {
                    proxy_pass http://127.0.0.1:3000;

  }
}
```

- sites-enabled, sites-available연결

```
$ sudo ln -s /etc/nginx/sites-available/reverse-proxy.conf
/etc/nginx/sites-enabled/reverse-proxy.conf
```

- ssl 설정 - Certbot

```
# Certbot 설치
$ apt-get update
```

```
$ sudo apt-get install certbot
$ apt-get install python3-certbot-nginx

# 인증서 발급
$ sudo certbot --nginx -d i10c102.p.ssafy.io
```

- reverse-proxy.conf

```
server {
                # 해당 서버 name
        server_name i10c104.p.ssafy.io;

                # reverse-proxy 설정
        access_log /var/log/nginx/reverse-access.log;
        error_log /var/log/nginx/reverse-error.log;

                # front
        location / {
                    proxy_pass http://localhost:3000;
    }
                # backend
        location /api/ {
                    proxy_pass http://localhost:8000/;

  }
               # AI
        location /ai/ {
                    proxy_pass http://localhost:5000/;

  }

        location /ws {
             proxy_pass http://localhost:3000;
             proxy_http_version 1.1;
             proxy_set_header Upgrade $http_upgrade;
             proxy_set_header Connection "upgrade";
             proxy_set_header Host $host;
             proxy_set_header Origin "";
    }

#Certbot 사용 및 ssl 설정(Certbot 관리)
listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/i10c104.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i10c104.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

        # make a room for image
    client_max_body_size 10M;

}
server {
    if ($host = i10c104.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;
        listen [::]:80;
        server_name i10c104.p.ssafy.io;
    return 404; # managed by Certbot


}
```

## MariaDB

- docker image 생성

```
# docker Image 생성 후 run
docker run --name mariadb
-p 3306:3306
-v /home/ubuntu/mariadb-data:/var/mariadb_home
-e MYSQL_ROOT_PASSWORD={root 패스워드}
-d mariadb:latest
```

```
# 위 설정에 대한 설명
docker image mariadb
name: mariadb
port:3306
root: root
pw: fubao82493
mount:/home/ubuntu/mariadb-data - /var/mariadb_home
```

## 6. HeidiSQL Connection

### 세션 연결(IP) 방법

```
- 네트워크 유형: MariaDB or MySQL
- 호스트명/IP: i10c104.p.ssafy.io
- 사용자: root
- 암호: fubao82493
- 포트: 3306
```

## 7. Files ignored

### Django

/backend/my_settins.py

```
# my_settings.py
SECRET_KEY = '장고 프로젝트 시크릿키'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # mysqlclient librarly 설치
        'NAME': 테이블 명,
        'USER': user name,
        'PASSWORD': 비밀번호,
        'HOST': mariadb,
        'PORT': '' # 안적으면 기본 포트 번호(3306)
    }
}

WEATHER_API_KEY= 'API 인증키'
```

## 8. Settings or Tips

- 본 문서는 배포 서버에 이미 docker가 설치되어있음을 가정한다.
- 해당 배포는 jenkins와 GitLab을 이용해 기준 repository에 commit&push 시 자동으로 배포 될 수 있도록 지원하고자 한다. (dockerFile, jenkinsFile 작성 필수)
- jenkins 파일 내에 docker를 run하는 과정에서 Django 서버와 Flask의 서버는 mariadb docker와 연동 될 수 있도록 link 옵션을 걸어두는 것에 주의한다.