# 여행가기쉬울지도 포팅매뉴얼

## 1. 서버아키텍쳐

## 2. 배포 환경/기술 스택

### 2.1 COMMON

- Server: Ubuntu 22.04.4 LTS
- Docker:  26.1.0
- jenkins: 2.440.3
- PostgreSql: 16.2
- Redis: 5.0.4

### 2.2 APP

- Kotlin : 1.9.23
- android: SDK(28~34) // Version(9~14)

### 2.3 BACK

- Django: 5.0.4
- numpy: 1.26.4
- pandas: 2.2.2
- xgboost: 2.0.3

### 2.4 DEVELOP TOOLS

- Visual Studio Code
- AWS EC2
- AWS S3
- Android Studio

## 3. 환경 변수 설정 파일

### 3.1 App

- buildgradle.kts에 작성

### 3.2 Back

```
# /var/jenkins_home/workspace/Backend/backend
```

```
# .env 파일생성
touch .env
# .env 파일 편집
vi .env
```

# 4. 서버 기본 설정

## 4.1 서버 시간 설정

```
sudo timedatectl set-timezone Asia/Seoul
```

## 4.2 패키지 목록 업데이트 및 패키지 업데이트

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

# 5. 방화벽 열기

```
sudo ufw allow
sudo ufw enable
sudo ufw reload
sudo ufw status

포트번호: 22,80,443
```

# 6. 필요한 리소스 설치

## 6.1 도커설치

### 6.1.1 도커 설치 전 필요한 패키지 설치

```
sudo apt-get -y install apt-transport-https ca-certificates
curl gnupg-agent software-properties-common
```

### 6.1.2 도커에 대한 GPG KEY 인증 진행.

! OK가 떴다면 정상적으로 등록된 것.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | s
udo apt-key add -
```

### 6.1.3 도커 저장소 등록

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release
```

### 6.1.4 패키지 리스트 갱신

```
sudo apt-get -y update
```

### 6.1.5 도커 패키지 설치

```
sudo apt-get -y install docker-ce docker-ce-cli containerd.io
```

### 6.1.6 도커 일반유저에게 권한부여

```
sudo usermod -aG docker ubuntu
```

### 6.1.7 도커 서비스 실행

```
sudo service docker restart
```

### 6.1.8 도커 compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/do
wnload/v2.21.0/docker-compose-$(uname -s)-$(uname -m)" -o /
usr/local/bin/docker-compose
```

## 6.2 Redis 설치(in Docker)

### 6.2.1 redis 네트워크 생성

```
docker network create redis-net
```

### 6.2.2 redis 이미지 가져오기

```
docker pull redis
```

### 6.2.3 redis 실행

```
docker run --name redis-server -p 6397:6397 --network redis-net -d redis redis-server --appen
donly yes --requirepass '비밀번호'
```

### 6.2.4 redis 서버 접속

```
docker exec -it redis-server redis-cli -a '비밀번호'
```

## 6.3 Postgresql 설치 (in docker)

### 6.3.1 PostgreSql 이미지 다운로드

```
docker pull postgres:16.2
```

### 6.3.2 PostgreSql 컨테이너 실행

```
docker run --name my-postgres-container -e POSTGRES_PASSWORD=mysecretpassword -d postgres
```

### 6.3.3 PostgresSQL 접속

```
docker exec -it 'postgres 컨테이너 이름' psql -U postgres
```

! 마지막 postgres는 기본관리자인 postgres 이다. 따라서 다른 사용자를 만들고 그 사용자 이름으로 접속할 수도 있다.

## 6.4 젠킨스 설치

### 6.4.1 저장소 설치

```
mkdir -p /var/jenkins_home
```

### 6.4.2 권한 설정

```
chown -R 1000:1000 /var/jenkins_home/
```

### 6.4.3 호스트 8888:컨테이너8080 매핑/ 포트50000을 도커 소켓 통신 위해 매핑

```
docker run --restart=on-failure --user='root' -p 8888:8080 -p 50000:50000 --env JAVA_HOME=/us
r/lib/jvm/java-17-openjdk-amd64 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.soc
k:/var/run/docker.sock -d --name jenkins jenkins/jenkins:lts
```

### 6.4.4 자바 및 데비안 설치

- 젠킨스에 자바 17버전 설치 및 환경 변수 입력(LTS로 컨테이너를 만들었기 때문에 자바17버전을 수동으로 설치해야 한다.) 그리고 젠킨스 OS는 데비안이기 때문에 위와 같은 도커 설치 명령으로는 설치가 되지 않고 데비안 키를 받아 설치해야한다.(젠킨스 컨테이너 안에서 실행)

```
#!/bin/bash

apt-get update
apt-get install openjdk-17-jdk -y

apt-get install -y \
apt-transport-https \
ca-certificates \
curl \
gnupg2 \
software-properties-common

curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -

add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/debian \
$(lsb_release -cs) \
stable"

apt-get update
apt-get install docker-ce docker-ce-cli containerd.io
```

### 6.4.5 젠킨스 파이프라인

```
pipeline {
    agent any
   // build start 알림이 필요할시 이자리에 stage 추가

        stage('Update repository') {
            steps {
                git branch: 'be/deploy', credentialsId: '본인credential', url: '클론받을 주소'
            }
        }
```

```
        stage('Build Docker image') {
            steps {
                dir("backend"){
                script {
                    sh'''
                         # 컨테이너가 실행 중인지 확인
                        if docker ps -a --filter "name=컨테이너 이름" --format '{{.Names}}' 2>/d
ev/null | grep -q '컨테이너 이름'; then
                            echo "컨테이너 '컨테이너 이름'가 실행 중입니다."
                            docker stop '컨테이너 이름'
                            docker rm '컨테이너 이름'
                        else
                            echo "컨테이너 '컨테이너 이름'가 실행 중이 아닙니다."
                        fi

                        # 이미지가 존재하는지 확인
                        if docker images -a --format '{{.Repository}}:{{.Tag}}' | grep -q '이
미지 이름'; then
                            echo "이미지 '이미지 이름'가 존재합니다."
                            docker rmi --force '이미지 이름'
                        else
                            echo "이미지 '이미지 이름'가 존재하지 않습니다."
                        fi

                    '''

                }
                }
            }
        }
        stage('Deploy') {
            steps {
                script {
                    // Docker 이미지를 실행하여 컨테이너 배포
                    sh '''
                     docker build -t '컨테이너 이름' /var/jenkins_home/workspace/Backend/backend
                     docker run -d -p 8087:8000 --name '컨테이너 이름' '이미지 이름'
                     ''' // 실제 Docker 이미지 이름으로 수정
                }
            }
         post {
                success {
                    // 알림 필요할시 script{} 작성
                }
                failure {
                    // 알림 필요할시 script{} 작성
                    }
                }
            }
    }
```

# 7. 도커를 활용한 django ci/cd

## 7.1 dockerfile 작성

- manage.py랑 같은 디렉토리에 ‘dockerfile’ 이름으로 파일 생성

```
# 설치할 파이썬 버전
FROM python:3.12.2-slim

# 파이썬 환경설정, 아래에 추가설명
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# 작업 디렉토리 설정
WORKDIR /app

# 소스 코드 추가
COPY . /app

# 필요한 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# 첫 배포 그리고 데이터베이스 변경사항이 있을 때 필요한 마이그래이션 수행
RUN python manage.py makemigrations

RUN python manage.py migrate

# 컨테이너 실행 시 실행할 명령어 설정
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

- PYTHONDONTWRITENBYTECODE 1: python이 ‘.pyc’ 파일을 생성하지 않도록 한다. 이로인해 ‘__pycache__’폴더에 저장되지 않는다. 그래서 디스크 사용이 줄어들고, 일부 환경에서 파일 시스템의 쓰기 작업을 최소화하는데 유용할 수 있다. 개발때 코드를 자주 변경하니까 이로인해 쌓이는cache파일을 방지할 수 있어서 편리하다.
- PYTHONUNBUFFERED 1 : Pythond의 stdout과 stderr의 버퍼링을 비활성화 한다. 도커 컨테이너에서 애플리케이션을 실행할 때 로그를 즉시 보기 위해 자주 사용되는 설정

## 7.2 환경세팅

### 7.2.1 배포, 개발 환경 세팅

- ‘__init__.py’ 설정

```
from split_settings.tools import include, optional
from decouple import config

# include base.py

include(
    'base.py',
    'logging.py',
    optional('local_setting.py')
)

if 'dev' == config('DJANGO_ENV'):
    include('dev.py')
elif 'prod' == config('DJANGO_ENV'):
    include('prod.py')
```

- 배포시 DEBUG = False 해야한다.

```
# settings/prod.py
print("RUNNING ON PRODUCTION ENVIRONMENT")
```

```
DEBUG = False
```

- 개발시는 DEBUG = True로 하는 것이 좋다.

```
# settings/dev.py
print("RUNNING ON DEVELOPMENT ENVIRONMENT")
DEBUG = True

ALLOWED_HOSTS = [
  '허용하고 싶은 호스트'
]
```

- django secret_key가 적혀있을 텐데 env 파일에 옮기고 아래처럼 사용하는것이 좋다.

```
from decouple import config

SECRET_KEY = config("SECRET_KEY")
```

- 정적데이터 ,동적 데이터 처리 설정

```
# settings/base.py
STATIC_URL = "/static/"
STATIC_ROOT = "/app/static/"

MEDIA_URL = "/media/"
MEDIA_ROOT = "/app/media/"
```

# 8. env 파일 내용

```
SECRET_KEY = 'your_django_secret_key'
DJANGO_ENV = '실행할 환경세팅파일'
DB_NAME = '데이터베이스 이름'
DB_USER = '데이터베이스 사용자'
DB_PASSWORD = '데이터베이스 비밀번호'
DB_HOST = '데이터베이스 호스트'
DB_PORT = '포트번호'

S3_BASE_URL = ''
GOOGLE_MAPS_API_KEY = ''
KAKAO_MAPS_API_KEY = ''
TMAP_API_KEY = ""

REDIS_LOCATION = ''
REDIS_PASSWORD = ''
```