|  |
| --- |
|  |

|  |
| --- |
|  |
| **포팅 메뉴얼** |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Sasatech - Book DoDum** |



목차

[I. 개요 2](#_Toc96072459)

[1. 프로젝트 사용 도구 2](#_Toc96072461)

[3. 개발환경 2](#_Toc96072462)

[4. 외부 서비스 2](#_Toc96072463)

[5. Gitgnore 처리한 핵심 키들 3](#_Toc96072464)

[II. 빌드 3](#_Toc96072465)

[1. 환경변수 형태 3](#_Toc96072466)

[2. 빌드하기 5](#_Toc96072467)

[3. 배포하기 5](#_Toc96072468)

[4. 서비스 이용 방법 7](#_Toc96072469)

[가) PAPAGO 7](#_Toc96072470)

[나) KakaoMap 8](#_Toc96072471)

[다) Dall-e2 8](#_Toc96072472)

[라) 도서관 정보나루 8](#_Toc96072472)

# 개요

## 프로젝트 사용 도구

이슈 관리 : JIRA

형상 관리 : Gitlab

커뮤니케이션 : Notion, discord

디자인 : Figma

UCC : 모바비

CI/CD : Jenkins, Docker

## 개발환경

VS Code : 1.74.2,

IntelliJ : 17.0.5+1-b653.23 amd64

JDK : 11.0.11

Node.js : 16.14.2

SERVER : AWS EC2 Ubuntu 20.04.3 LTS

DB : MySQL

## 외부 서비스

Papago(back) : application.yml

Dall-E 2(front) : .env /REACT\_APP\_CANGE\_API\_KEY

KakaoMap(front): .env /REACT\_APP\_KAKAO\_REST\_API\_KEY

도서관 정보 나루 : .env /REACT\_APP\_LIBRARY\_API\_KEY

## Gitgnore 처리한 핵심 키들

Vue : .env.loval (최상단 위치)

Spring : application.yml, serviceAccountKey.json

(\src\main\resources, 또는 classPath에 위치)

Flask : .env (최상단 위치)

# 빌드

## 환경변수 형태

.env.local :

// FIREBASE RTDB 프론트 키

VUE\_APP\_FIREBASE\_APIKEY=

VUE\_APP\_FIREBASE\_AUTHDOMAIN=

VUE\_APP\_FIREBASE\_DATABASEURL=

VUE\_APP\_FIREBASE\_PROJECTID=

VUE\_APP\_FIREBASE\_STORAGEBUCKET=

VUE\_APP\_FIREBASE\_MESSAGING\_SENDERID=

VUE\_APP\_FIREBASE\_APPID=

VUE\_APP\_FIREBASE\_MEASUREMENTID=

// YOUTUBE DATA API KEY

VUE\_APP\_YOUTUBE\_API\_KEY=

// 카카오 공유 서비스 키

VUE\_APP\_KAKAO\_API\_KEY=

.env

# api URL

REACT\_APP\_API\_URL

REACT\_APP\_API\_FRONT\_URL

#

# bigdata api URL

REACT\_APP\_DATA\_API\_URL

# 도서관 정보나루 API key

REACT\_APP\_LIBRARY\_API\_URL

REACT\_APP\_LIBRARY\_API\_KEY

# 이미지 api

REACT\_APP\_CHANGE\_API\_URL

REACT\_APP\_CHANGE\_API\_KEY

# 카카오 맵 api

REACT\_APP\_KAKAO\_MAP\_KEY

REACT\_APP\_KAKAO\_REST\_API\_KEY

.application.yml

#MySQL(배포)

#JWT 시크릿 키

Jwt:

Secret: : 시크릿 키

#papago 키

Papago:

clientId:

secret :

## 빌드하기

1) Front

npm i

npm run build

2) Back-spring

Gradle 실행

Bootjar 실행

3) Back-flask

pip install -r requirements.txt

서버 구동 위치에서 python app.py

## 배포하기

## AWS 도커 올리기

sudo apt update

sudo apt-get install -y ca-certificates \

curl \

software-properties-common \

apt-transport-https \

gnupg \

lsb-release

## gpg 키 다운로드

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

## Docker 설치

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io docker-compose

## 젠킨스 설치 - aws환경

## Vim docker-copose.yml

version: '3'

services:

jenkins:

image: jenkins/jenkins:lts

container\_name: jenkins

volumes:

- /var/run/docker.sock:/var/run/docker.sock

- /jenkins:/var/jenkins\_home

ports:

- "9090:8080"

privileged: true

user: root

## 컨테이너 생성

## sudo docker-compose up –d

## 젠킨스 : 서버 공인ip:9090포트 접속

## sudo docker logs Jenkins 입력 후

## 

## 빨간 박스의 값을 시작 값에 추가

## Install suggested plugin클릭

## 계정 생성 후

## 

## 

## image

## image

## image

## Frontend/React, backend/spring, backend/Django 나눠서 프로젝트 만들어 놓음

## 젠킨스 프로젝트 생성

## image

## image

## image

## None을 클릭하면 Form이 나옴

## 

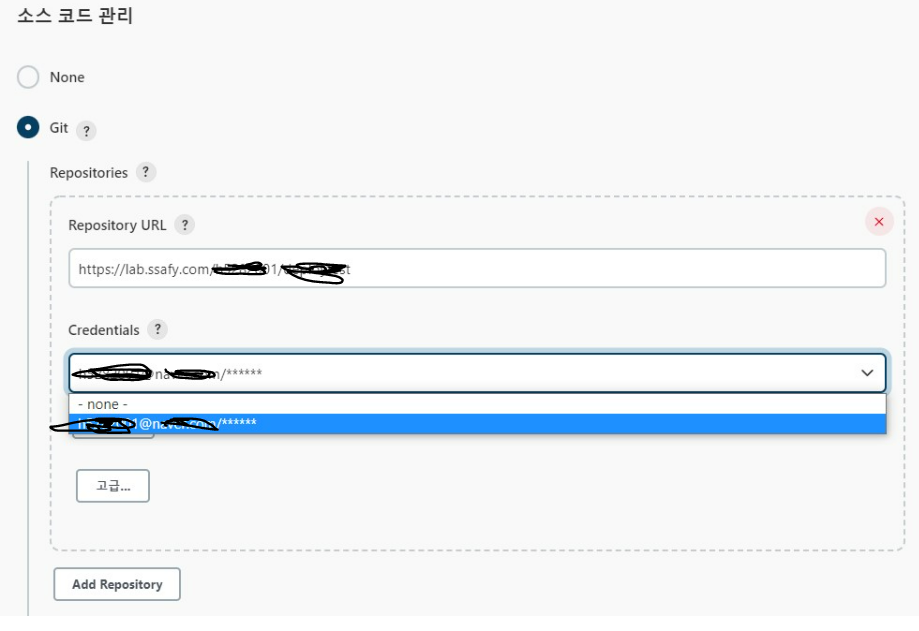
## Repository URL에는 싸피깃 레포지토리 URL입력

## 

## Credentials에서 add-> Jenkins 클릭

## 

* Username : 싸피깃 아이디
* Password : 싸피킷 비밀번호
* ID : Credential 구별할 아무 텍스트 입력하면 됩니다.



## 오류메세지 사라지면 성공

## image

## 빌드 유발 탭에서는 먼저 밑줄 친 체크박스 체크, 고급 클릭

## image

## 스크롤을 내려 Secret token을 찾아 Generate버튼을 누르면 토큰 생성

## Gitlab과 Webhook을 연결할 때 사용되니 저장

## image

## build탭에서 Add build step을 클릭 Execute Shell을 선택

## image

## 연결 테스트이기 때문에 pwd만 입력

## 저장 눌러 저장

## #깃랩 webhook연결

## 

## 

URL에는 http://배포서버공인IP:9090/project/생성한jenkins프로젝트이름/을 입력해줍니다.

Secret token에는 아까 위에서 젠킨스 프로젝트를 생성할 때 저장해둔 값을 입력합니다.

빌드 유발 Trigger으로, Push events로 설정 – Merge request events도 선택해도 됨

완료했다면 Add Webhook버튼을 눌러 생성

## image

## 빌드 테스트를 위해 생성된 WebHook에서 test에서 push events누른다.

## image

## 성공적으로 연결 확인(200)

## 젠킨스와 연결된 gitlab에 도커 이미지 빌드하기

## 젠킨스 컨테이너 안에 도커 설치 -> EC2에 설치한 과정과 동일

## 젠킨스 bash에 접근 – sudo docker exec –it Jenkins bash

젠킨스 컨테이너 내부에서 설치된 os을 체크하는 명령어 cat /etc/issue를 통해 os를 확인해보면 debian으로 나타나는 것을 확인할 수 있습니다.

기존 링크에서 제공한 방식은 ubuntu os에 대한 gpg 키를 다운로드 하는 것이기 때문에, 이를 debian으로 바꾸어주어야 합니다.

이를 바꾸지 않으면 패키지를 찾지 못하는 에러가 발생합니다.

기존 명령어에서 ubuntu로 되어있는 부분을 debian으로 바꾸어주면 해결

## 도커 파일 작성

## React Dockerfile

# build stage

FROM node:16.18.0 as build-stage

# # 웹캠에 접근할 수 있는 권한 설정

# RUN chmod 777 /dev/video0

# USER webcam

# 필요한 라이브러리 및 패키지 설치

# RUN apk add --no-cache v4l-utils

#

WORKDIR /app

COPY package\*.json ./

RUN npm install --force

COPY . .

RUN npm run build

# production stage

FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /app/build /usr/share/nginx/html

COPY --from=build-stage /app/deploy\_conf/nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

#test

## SpringBoot Dockerfile

# 사용할 base 이미지를 정의합니다.  
FROM openjdk:11.0.11-jdk-slim-buster  
  
# 작업 디렉토리를 설정합니다.  
WORKDIR /var/jenkins\_home/workspace/S08P22C104/backend/SpringFramework/bookdodum  
  
# 실시간 jenkins 테스트  
  
# Gradle 빌드 스크립트 파일을 복사합니다.  
COPY build.gradle .  
COPY gradlew .  
  
# Gradle 의존성을 복사합니다.  
COPY gradle gradle  
  
RUN chmod +x ./gradlew  
#test  
# 애플리케이션을 빌드합니다.  
RUN ./gradlew build --stacktrace  
  
# 빌드된 jar 파일을 복사합니다.  
COPY build/libs/bookdodum-0.0.1-SNAPSHOT.jar app.jar  
  
# 컨테이너 실행 시 실행할 명령어를 정의합니다.  
CMD ["java", "-jar", "app.jar"]

## Django Dockerfile

FROM python:3.9.11

WORKDIR /app

RUN pip install gunicorn

COPY requirements.txt ./

RUN pip install --upgrade pip

RUN python -m venv venv

RUN . venv/bin/activate

RUN pip install -r requirements.txt

COPY . .

RUN python manage.py migrate

RUN python manage.py loaddata books.json

# RUN python manage.py runserver

# CMD ["gunicorn", "testproject.wsgi", "--bind", "0.0.0.0:8082"]

CMD ["python", "manage.py", "runserver", "0.0.0.0:8082"]

## 젠킨스 Dockerfile이용 도커 이미지 올리기

## 

## 구성으로 이동

## -----Execute shell----------------------------------------------------------------------

## docker image prune -a --force

## mkdir -p /var/jenkins\_home/images\_tar

## # spring

## cd /var/jenkins\_home/workspace/S08P22C104/backend/SpringFramework/bookdodum

## ./gradlew clean build --stacktrace

## docker build --no-cache -t spring .

## cd /var/jenkins\_home/images\_tar

## rm spring.tar

## docker save spring > /var/jenkins\_home/images\_tar/spring.tar

## # django

## cd /var/jenkins\_home/workspace/S08P22C104/backend/Django

## docker build -t django .

## docker save django > /var/jenkins\_home/images\_tar/django.tar

## # react

## cd /var/jenkins\_home/workspace/S08P22C104/frontend

## docker build -t react .

## docker save react > /var/jenkins\_home/images\_tar/react.tar

## 저장 후 지금 빌드 클릭

## 

## 이렇게 저장되면 성공

## 젠킨스 SSH 명령어 전송을 통해 빌드한 도커 이미지로 컨테이너 실행하기

## SSH 연결 설정

## 

## jenkins관리를 클릭하고 이어서 시스템 설정을 클릭

## image

## Public over ssh 항목에 추가 버튼 클릭

## image

Name : 그냥 이름

Hostname : EC2 IP

Username : EC2 접속 계정 이름

## 작성 후 고급 버튼 클릭

## 

## Key에 EC2 .pem 키 값 입력

## 이후 Test Configuration 버튼 눌렀을 때 success가 나오면 성공

## 구성 /빌드 후 조치로 이동

## 

## image

## Exex command에 명령어 추가

## ------------------------------------------------------------------------------------------

## sudo docker load < /jenkins/images\_tar/react.tar

## sudo docker load < /jenkins/images\_tar/spring.tar

## sudo docker load < /jenkins/images\_tar/django.tar

## if (sudo docker ps | grep "react"); then sudo docker stop react; fi

## if (sudo docker ps | grep "spring"); then sudo docker stop spring; fi

## if (sudo docker ps | grep "django"); then sudo docker stop django; fi

## sudo docker run -it -d --rm --network testNetwork -p 8081:8081 --name spring spring

## #ssl인증서 사용 전

## sudo docker run –it d –rm network testNetwokr –p 80:80 –name react react

## #ssl 인증서 발급 후

## #sudo docker run -it -d --rm --network testNetwork -p 80:80 -p 443:443 -v ~/etc/letsencrypt/:/etc/letsencrypt/ --name react react

## sudo docker run -it -d --rm --network testNetwork -p 8000:8082 --name django django

## ------------------------------------------------------------------------------------------

**Nginx 설정**

cd /Jenkins/workspace/S08P22C104/frontend디렉토리로 이동 후

sudo mkdir deploy\_conf명령어로 디렉토리 생성

cd deploy\_conf를 이용해 이동

sudo vim nginx.conf명령어로 nginx.conf 파일 생성

(카메라와 gps기능이 있어 https 사용해야 함)

-----------nignx.conf--------------------------------------------------------------------

server {

listen 80;

server\_name dodum.xyz;

return 301 http://$server\_name$request\_uri;

}

server {

listen 443 ssl;

server\_name dodum.xyz;

ssl\_certificate /etc/letsencrypt/live/dodum.xyz/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/dodum.xyz/privkey.pem;

root /usr/share/nginx/html;

index index.html;

location / {

index index.html index.htm;

try\_files $uri $uri/ /index.html;

}

location /api {

proxy\_pass http://43.201.102.210:8081;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

location /ml {

proxy\_pass http://43.201.102.210:8000;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

}

-------------------------------------------------------------------------------------------------

Push 후 빌드 되면 react컨테이너 환경으로 이동해서 아래 명령어 입력

apk update

apk add vim //.conf파일 수정을 위해 설치

apk update

apk add certbot certbot-nginx //SSL인증서 무료 발급

nginx –s reload – nignx //변경사항 반영해서 실행

certbot –nginx –d {EC2도메인}

/etc/letsencrypt/live/{EC2 도메인}/fullchain.pem

/etc/letsencrypt/live//{EC2 도메인} /privkey.pem

이 두개 키를 메모장에 복사

exit해서 ec2환경으로 돌아가 /etc/letsencrypt live/{EC2 도메인} 디텍토리 안에 파일 생성해서 저장

sudo vim fullchain.pem //메모장에 복사해논 값

sudo vim privkey.pem

## Exex command에 명령어 ssl발급 후로 변경

## #sudo docker run -it -d --rm --network testNetwork -p 80:80 -p 443:443 -v ~/etc/letsencrypt/:/etc/letsencrypt/ --name react react

## -v ~/etc/letsencrypt/:/etc/letsencrypt/ : volume으로 ec2에 만들어 놓은 /etc/letsencrypt이하의 디렉토리들을 react 컨테이너 환경에 복사

## 서비스 이용 방법

## Papago

## Papago API 웹 사이트 -> 파파고 API 사용 신청 -> 애플리케이션 등록(API 이용 신청) -> API 키 생성

## -

## Kakao-Map

## KakaoDeveloper 로그인 → 내 애플리케이션 → 애플리케이션 추가하기

## 앱설정 → 요약정보 → 앱키 확인(REST API)

## Dall-E 2

* OpenAI API웹 사이트 → Personal → View API Keys
* Create new secret key 버튼 눌러서 api key 생성
* API 호출 50건 이상이면 key 만료

## 도서관 정보나루

* 도서관 정보나루 웹 사이트 → 마이페이지
* 인증키 승인 신청하기 → 인증키 받기