포팅 매뉴얼

**-프로젝트 사용 도구**

|  |  |
| --- | --- |
| 이슈 관리 | Jira |
| 형상 관리 | Gitlab |
| 커뮤니케이션 | Mattermost, notion |
| 와이어프레임 작성 및 디자인 | Figma |
| UCC 제작 | Movavi |
| CI/CD | Jenkins |

**-개발 환경**

-Frontend

Node.js: 18.16.1

React: 2.0.0

-Backend

Java: openjdk11

Mysql: 8.0.33

Spring Boot: 2.7.15

Redis: 7.2.1

**-배포 환경**

서버: AWS EC2 Ubuntu 20.04 (LTS)

-외부 서비스

Google OAuth

Kakao OAuth

**-아키텍처 구성도**

**1) EC2 서버에 설치할 기술들**

**①Docker**

-Docker 설치

sudo apt update

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

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

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

sudo apt update

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

-Docker-compose 설치

sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

docker-compose --version

df -h

- Docker container로 생성할 App 들

Container를 생성할 때 버전과 빌드 환경을 설정하는 설명서 역할을 하는 docker-compose.yml 파일은, 서버의 /app/config 경로에 각각 backend, frontend, db폴더에 저장합니다. 이 파일들은 Jenkins에서 Docker image 빌드 시 Jenkins workspace에 복사하여 사용합니다.

①백엔드 이미지-Spring Boot (8080 포트)

-/app/config/backend/Dockerfile

\* application.yml 파일에서 profile에 따라 다른 property set이 적용되는 경우,

Dockerfile에 USE\_PROFILE ENV를 추가해야 합니다!

FROM gradle:8.2.1-jdk11

WORKDIR /vh

ENV USE\_PROFILE prod

COPY ./build/libs/vh-0.0.1-SNAPSHOT.jar vh.jar

ENTRYPOINT ["java","-Dspring.profiles.active=${USE\_PROFILE}", "-jar", "vh.jar"]

-/app/config/backend/docker-compose.yml

\*여기에도 environment에 spring profiles 설정 추가

version: "3.8"

services:

backend:

env\_file:

- "./vh/.env"

environment:

- "SPRING\_PROFILES\_ACTIVE=prod"

- "TZ=Asia/Seoul"

build:

context: ./vh

dockerfile: Dockerfile

restart: always

ports:

- 8080:8080

container\_name: backend

networks:

- deploy

redis:

image: redis:latest

container\_name: redis

hostname: redis

command: redis-server --requirepass melon1\* --port 6379

volumes:

- redis-volume:/data

environment:

- REDIS\_PASSWORD=melon1\*

ports:

- 6379:6379

restart: always

networks:

- deploy

networks:

deploy:

external: true

volumes:

redis-volume:

②프론트엔드 이미지-React (3000 포트)

-/app/config/frontend/Dockerfile

FROM node:18.16.1

WORKDIR ./frontend

COPY package.json ./

RUN npm install

COPY . ./

RUN npm run build

EXPOSE 3000

-/app/config/frontend/docker-compose.yml

version: "3.8"

services:

frontend:

restart: always

command: npm start

container\_name: frontend

environment:

- "TZ=Asia/Seoul"

build:

context: .

dockerfile: Dockerfile

ports:

- "3000:3000"

stdin\_open: true

networks:

- deploy

networks:

deploy:

external: true

③데이터베이스-Mysql, Redis (Mysql-3306 포트, Redis-6379)

-Mysql: docker-compose.yml

version: "3.8"

services:

db:

image: mysql:8.0.33

command: "--lower\_case\_table\_names=1"

container\_name: db

environment:

MYSQL\_ROOT\_PASSWORD: melon1\*

MYSQL\_USER: it

MYSQL\_PASSWORD: melon1\*

MYSQL\_DATABASE: it

MYSQL\_CHARACTER\_SET\_SERVER: utf8mb4

MYSQL\_COLLATION\_SERVER: utf8mb4\_unicode\_ci

TZ: Asia/Seoul

ports:

- 3306:3306

volumes:

- db:/var/lib/mysql

networks:

- deploy

networks:

deploy:

external: true

volumes:

db:

-Redis: 백엔드 docker-compose.yml에 포함

\*기타) docker image 생성 시 필요한 .env 파일-.env 파일은 gitignore이므로 서버에 저장 후 필요 시 복사해서 이용

**②Jenkins**

Jenkins를 통해 CI/CD를 실행합니다. 이번 프로젝트에서는 Gitlab Webhook과 Jenkins를 연결하여,

ⅰ. Gitlab Webhook 대상 브랜치에 Push/Merge가 실행되면

ⅱ. Jenkins는 Gitlab으로부터 프로젝트를 clone한 후,

ⅲ. 서버 내에 저장된 Dockerfile과 docker-compose.yml을 Jenkins workspace로 복사해 온 후

ⅳ. 빌드하여 docker image를 만듭니다.

Jenkins pipeline으로 이 워크플로우는 하나의 파이프라인으로 모델링될 수 있습니다.

-설치

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt update

sudo apt install jenkins

sudo ufw disable

sudo vi /lib/systemd/system/jenkins.service

sudo systemctl daemon-reload

sudo systemctl restart jenkins

sudo cat /var/lib/jenkins/secrets/initialAdminPassword jenkins-plugin-cli --plugins ssh-agent:333.v878b\_53c89511

-파이프라인 생성

ⅰ. 프론트엔드 파이프라인

pipeline {

agent any

options {

timeout(time: 1, unit: 'HOURS')

}

environment

{

CREDENTIAL\_ID = 'vh' #jenkins credentials에 설정된 id

SOURCE\_CODE\_URL = 'https://lab.ssafy.com/s09-bigdata-dist-sub2/S09P22A504.git' #ssafy git url

RELEASE\_BRANCH = 'dev' #gitlab webhook 브랜치 이름

}

stages {

stage('git clone') {

steps {

git url: "$SOURCE\_CODE\_URL",

branch: "$RELEASE\_BRANCH",

credentialsId: "$CREDENTIAL\_ID"

sh "ls -al"

}

}

stage('set frontend environment') {

steps {

sh "cp /app/config/frontend/.env /var/lib/jenkins/workspace/vh\_frontend/frontend/.env"

sh "cp /app/config/frontend/Dockerfile /var/lib/jenkins/workspace/vh\_frontend/frontend/Dockerfile"

sh "cp /app/config/frontend/docker-compose.yml /var/lib/jenkins/workspace/vh\_frontend/frontend/docker-compose.yml"

}

}

stage('down container') {

steps {

dir("./frontend"){

sh "docker-compose -f docker-compose.yml down --rmi all"

}

}

// steps{

// }

}

stage('build docker') {

steps {

dir("./frontend"){

sh "docker-compose -f docker-compose.yml build --no-cache"

}

}

}

stage('up container') {

steps {

dir("./frontend"){

sh "docker-compose -f docker-compose.yml up -d"

}

}

}

}

}

ⅱ. 백엔드 파이프라인

pipeline {

agent any

environment {

CREDENTIAL\_ID = 'vh'

SOURCE\_CODE\_URL = 'https://lab.ssafy.com/s09-bigdata-dist-sub2/S09P22A504.git'

RELEASE\_BRANCH = 'dev'

}

stages {

stage('git clone') {

steps {

git url: "$SOURCE\_CODE\_URL",

branch: "$RELEASE\_BRANCH",

credentialsId: "$CREDENTIAL\_ID"

sh "ls -al"

}

}

stage('set backend environment'){

steps{

dir("./backend/vh"){

sh '''

cp /app/config/backend/.env /var/lib/jenkins/workspace/vh\_backend/backend/vh/.env

cp /app/config/backend/Dockerfile /var/lib/jenkins/workspace/vh\_backend/backend/vh/Dockerfile

cp /app/config/backend/docker-compose.yml /var/lib/jenkins/workspace/vh\_backend/backend/docker-compose.yml'''

sh "chmod +x ./gradlew"

sh "./gradlew clean"

sh "./gradlew build -x test"

}

}

}

stage('down container') {

steps {

dir("./backend"){

sh "docker-compose -f docker-compose.yml down --rmi all"

}

}

}

stage('build docker') {

steps {

dir("./backend"){

sh "docker-compose -f docker-compose.yml build --no-cache"

}

}

}

stage('up container') {

steps {

dir("./backend"){

sh "docker-compose -f docker-compose.yml up -d"

}

}

}

}

}

**③Nginx**

-설치

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install nginx

-Certbot 설치

sudo snap install core

sudo snap refresh core

sudo apt remove certbot

sudo snap install --classic certbot

ln -s /snap/bin/certbot /usr/bin/certbot

sudo ln -s /snap/bin/certbot /usr/bin/certbot

sudo certbot --nginx

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

-nginx 설정

Nginx 설치 후 /etc/nginx/conf.d/default.conf를 다음과 같이 설정합니다.

\*default.conf는 nginx.conf를 통해 include 된 서버 설정 관련 파일입니다.

upstream frontend {

server j9a504.p.ssafy.io:3000;

}

upstream backend {

server j9a504.p.ssafy.io:8080;

}

server {

listen 80;

server\_name j9a504.p.ssafy.io;

location /api {

proxy\_pass http://j9a504.p.ssafy.io:8080;

proxy\_set\_header Host $host;

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

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

proxy\_set\_header X-Forwarded-Proto $scheme;

}

location / {

return 301 https://$host$request\_uri;

}

}

server{

listen 443 ssl; # managed by Certbot

ssl\_certificate /etc/letsencrypt/live/j9a504.p.ssafy.io/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/j9a504.p.ssafy.io/privkey.pem;

server\_name j9a504.p.ssafy.io;

location / {

proxy\_pass http://j9a504.p.ssafy.io:3000;

proxy\_redirect default;

proxy\_set\_header Host $http\_host;

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

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

proxy\_set\_header X-Forwarded-Proto $scheme;

}

location /api {

proxy\_pass http://j9a504.p.ssafy.io:8080;

proxy\_set\_header Host $host;

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

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

proxy\_set\_header X-Forwarded-Proto $scheme;

}

include /etc/letsencrypt/options-ssl-nginx.conf;

ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem;

}

-BackEnd .env

accessKey=AKIA5JDRDAAY6JFUKGQ5

secretKey=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

region=ap-northeast-2

stack\_auto=false

bucket=visiblehand-bucket

SPRING\_DATASOURCE\_USERNAME=root

SPRING\_DATASOURCE\_PASSWORD=melon1\*

-FrontEnd .env

REACT\_APP\_HTTP\_URL = http://j9a504.p.ssafy.io

REACT\_APP\_GOOGLE\_CLIENT\_ID =

627892398327-lne8q8lf9jkbju5glga1vm9q03vktpjr.apps.googleusercontent.com

REACT\_APP\_GOOGLE\_REDIRECT\_URI = http://j9a504.p.ssafy.io/auth/google

REACT\_APP\_KAKAO\_CLIENT\_ID = 239963a7ee5248741ccce5709bed01cf

REACT\_APP\_KAKAO\_REDIRECT\_URI = http://j9a504.p.ssafy.io/auth/kakao