**B303 포팅 매뉴얼 (빌드, 배포)**

팀명 : Wow

프로젝트명 : 키우고팜

**백엔드(BE)**

Java: OpenJDK 17, Spring Boot: 3.2.3 (내장 웹서버 사용), Gradle: 8.5, JPA: 3.2.4

Lombok, Spring Security: 6.2, OAuth2 JWT: 6.2, Flask

**프론트엔드(FE)**

Vue.js: 3.4.21, Node.js: 20.11.0, npm: 10.2.4, Tailwind CSS: 3.4.1,JavaScript: ES6

**데이터베이스(DB)**

MySQL: 8.0, Redis: 3.2, MongoDB: 4.2

**DevOps 및 추가 도구**

Jenkins: 2.426.3, Nexus Repository, Docker: 19.03.12

**웹서버 및 기타 기술**

Nginx 1.24.0

**개발 환경**

IntelliJ IDEA: 2023.3.4 (Ultimate Edition), Visual Studio Code: 1.86

**Application.yml**

jwt:  
 secretKey: jfdashlghiopewrgjweirquyptih4554dfkjhertiuyasdf  
 access:  
 header: Access  
 expiration: 1800000000  
 refresh:  
 header: Refresh  
 expiration: 604800  
  
spring:  
 datasource:  
 url: jdbc:mysql://j10b303.p.ssafy.io:3306/member?serverTimezone=Asia/Seoul  
 username: root  
 password: ssafy  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 hikari:  
 pool-name: jpa-hikari-pool  
 maximum-pool-size: 20  
 jdbc-url: ${spring.datasource.url}  
 username: ${spring.datasource.username}  
 password: ${spring.datasource.password}  
 driver-class-name: ${spring.datasource.driver-class-name}  
 data-source-properties:  
 rewriteBatchedStatements: true  
  
 jpa:  
 show-sql: truehibernate:  
 ddl-auto: none  
 properties:  
 hibernate:  
 format\_sql: true open-in-view: false security:  
 oauth2:  
 client:  
 registration:  
 google:  
 clientId: client id input  
 clientSecret: client secret input  
 scope:  
 - email  
 - profile  
 naver:  
 clientId: client id input  
 clientSecret: client secret input  
 clientAuthenticationMethod: post  
 authorizationGrantType: authorization\_code  
 redirectUri: "http://localhost:8080/{action}/oauth2/code/{registrationId}"  
 scope:  
 - nickname  
 - email  
 - profile\_image  
 clientName: Naver  
 kakao:  
 clientId: 20dc23e573cbebb6e44b57c46021b332  
 clientAuthenticationMethod: client\_secret\_post  
 authorizationGrantType: authorization\_code  
 redirectUri: "http://j10b303.p.ssafy.io:8080/{action}/oauth2/code/{registrationId}"  
 scope:  
 - profile\_nickname  
 - profile\_image  
 - account\_email  
 clientName: Kakao  
 provider:  
 naver:  
 authorizationUri: https://nid.naver.com/oauth2.0/authorize  
 tokenUri: https://nid.naver.com/oauth2.0/token  
 userInfoUri: https://openapi.naver.com/v1/nid/me  
 userNameAttribute: response  
 kakao:  
 authorizationUri: https://kauth.kakao.com/oauth/authorize  
 tokenUri: https://kauth.kakao.com/oauth/token  
 userInfoUri: https://kapi.kakao.com/v2/user/me  
 userNameAttribute: id

**Nginx 설정 파일**

default.conf 파일

**server** {

**listen** 80 default\_server;

**listen** [::]:80 default\_server;

**root** /app/dist;

**index** index.html;

*#location / {*

*# First attempt to serve request as file, then*

*# as directory, then fall back to displaying a 404.*

*# try\_files $uri $uri/ =404;*

*#}*

**location** / {

**proxy\_pass** http://j10b303.p.ssafy.io:5173;

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

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

**proxy\_set\_header** Host $http\_host;

}

**location** /api {

**rewrite** ^/api(/.\*)$ $1 break;

**proxy\_pass** http://j10b303.p.ssafy.io:8080;

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

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

**proxy\_set\_header** Host $http\_host;

**proxy\_redirect** off;

}

**location** /recommendapi {

**rewrite** ^/recommendapi(/.\*)$ $1 break;

**proxy\_pass** http://j10b303.p.ssafy.io:8082;

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

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

**proxy\_set\_header** Host $http\_host;

**proxy\_redirect** off;

}

**location** /recommendapi {

**rewrite** ^/recommendapi(/.\*)$ $1 break;

**proxy\_pass** http://j10b303.p.ssafy.io:8082;

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

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

**proxy\_set\_header** Host $http\_host;

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

**proxy\_redirect** off;

}

**location** /test {

**proxy\_pass** http://localhost:9000;

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

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

**proxy\_set\_header** Host $http\_host;

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

**proxy\_redirect** off;

}

**listen** 443 ssl; *# managed by Certbot*

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

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

}

**server** {

**if** ($host = j10b303.p.ssafy.io) {

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

} *# managed by Certbot*

**listen** 80;

**server\_name** j10b303.p.ssafy.io;

**return** 404;

}

**Front Dockerfile 설정**

**FROM** node:20.11.0

**WORKDIR** /app

**COPY** package.json .

**RUN** npm install

**COPY** . .

**CMD** ["npm", "start"]

**Backend Dockerfile 설정**

**FROM** bellsoft/liberica-openjdk-alpine:17

**CMD** ["./gradlew", "clean", "build"]

**VOLUME** /tmp

**ARG** JAR\_FILE=build/libs/\*.jar

**COPY** **${**JAR\_FILE**}** app.jar

**EXPOSE** 8080

**ENTRYPOINT** ["java","-jar","/app.jar"]

**Flask Dockerfile 설정**

**FROM** python:3.12.0

**WORKDIR** /app

**COPY** requirements.txt .

**RUN** pip install --no-cache-dir -r requirements.txt

**COPY** . .

**EXPOSE** 8082

**CMD** ["python", "app.py"]

**Frontend Jenkins PipeLine 설정**

pipeline {

agent any

environment {

CONTAINER\_NAME = "rhythm-frontend-container"

IMAGE\_NAME = "rhythm-frontend-image"

}

stages {

stage('Git Clone') {

steps {

git branch: 'master', credentialsId: 'gitlabToken', url: 'https://lab.ssafy.com/s10-bigdata-recom-sub2/S10P22B303.git'

}

}

stage('Build') {

steps {

dir("./FrontEnd") {

nodejs(nodeJSInstallationName: 'NodeJS 20.11.0') {

sh 'npm install && npm run build'

}

}

}

}

stage('Docker delete') {

steps {

script {

*// 컨테이너가 존재하면 삭제합니다.*

sh "docker stop ${CONTAINER\_NAME} || true"

sh "docker rm -f ${CONTAINER\_NAME} || true"

*// 이미지가 존재하면 삭제합니다.*

sh "docker image rm ${IMAGE\_NAME} || true"

}

}

post {

success {

echo "docker delete Success"

}

failure {

echo "docker delete Fail"

}

}

}

stage('Dockerizing') {

steps {

sh 'echo "Image Build Start"'

sh """

docker build -t ${IMAGE\_NAME} ./FrontEnd

"""

}

post {

success {

echo "Build Docker Image Success"

}

failure {

echo "Build Docker Image Fail"

}

}

}

stage('Deploy') {

steps {

sh "docker run --name ${CONTAINER\_NAME} -d -p 5173:5173 ${IMAGE\_NAME}"

}

post {

success {

echo 'Deploy Success'

}

failure {

echo 'Deploy Failed'

}

}

}

}

}

**Backend Jenkins PipeLine 설정**

pipeline {

agent any

environment {

CONTAINER\_NAME = "rhythm-backend-container"

IMAGE\_NAME = "rhythm-backend-image"

}

stages {

stage('Git Clone') {

steps {

git branch: 'master', credentialsId: 'gitlabToken', url: 'https://lab.ssafy.com/s10-bigdata-recom-sub2/S10P22B303.git'

}

}

stage('Build') {

steps {

sh '''

cd ./BackEnd

chmod +x ./gradlew

./gradlew clean build -x test

'''

}

}

stage('Docker delete') {

steps {

script {

**try** {

*// 컨테이너가 존재하면 삭제합니다.*

sh "docker stop ${CONTAINER\_NAME}"

sh "docker rm -f ${CONTAINER\_NAME}"

} **catch** (Exception e) {

*// 컨테이너가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.*

echo "Docker container ${CONTAINER\_NAME} does not exist. Skipping deletion."

}

**try** {

*// 이미지가 존재하면 삭제합니다.*

sh "docker image rm ${IMAGE\_NAME}"

} **catch** (Exception e) {

*// 이미지가 존재하지 않는 경우 에러가 발생할 수 있으므로, 에러를 무시합니다.*

echo "Docker image ${IMAGE\_NAME} does not exist. Skipping deletion."

}

}

}

post {

success {

sh 'echo "docker delete Success"'

}

failure {

sh 'echo "docker delete Fail"'

}

}

}

stage('Dockerizing'){

steps{

sh 'echo " Image Bulid Start"'

sh """

cd ./BackEnd

docker build -t ${IMAGE\_NAME} .

"""

}

post {

success {

sh 'echo "Bulid Docker Image Success"'

}

failure {

sh 'echo "Bulid Docker Image Fail"'

}

}

}

stage('Deploy') {

steps {

sh "docker run --name ${CONTAINER\_NAME} -d -p 8080:8080 ${IMAGE\_NAME}"

}

post {

success {

echo 'deploy success'

}

failure {

echo 'deploy failed'

}

}

}

}

}

**Flask Jenkins PipeLine 설정**

pipeline {

agent any

environment {

PATH = "$PATH:/usr/local/bin" *// Python 바이너리 경로 추가*

CONTAINER\_NAME = "flask-container"

IMAGE\_NAME = "flask-image"

}

stages {

stage('Git Clone') {

steps {

git branch: 'master', credentialsId: 'gitlabToken', url: 'https://lab.ssafy.com/s10-bigdata-recom-sub2/S10P22B303.git'

}

}

stage('Build Docker Image') {

steps {

dir("/Flask") { *// Dockerfile이 있는 경로로 이동*

*// Docker 빌드 명령 실행*

}

}

}

stage('Docker delete') {

steps {

script {

*// 컨테이너가 존재하면 삭제합니다.*

sh "docker stop ${CONTAINER\_NAME} || true"

sh "docker rm -f ${CONTAINER\_NAME} || true"

*// 이미지가 존재하면 삭제합니다.*

sh "docker image rm ${IMAGE\_NAME} || true"

}

}

post {

success {

echo "docker delete Success"

}

failure {

echo "docker delete Fail"

}

}

}

stage('Dockerizing') {

steps {

sh 'echo "Image Build Start"'

sh """

docker build -t ${IMAGE\_NAME} ./Recommend

"""

}

post {

success {

echo "Build Docker Image Success"

}

failure {

echo "Build Docker Image Fail"

}

}

}

stage('Deploy') {

steps {

sh "docker run --name ${CONTAINER\_NAME} -d -p 8082:8082 ${IMAGE\_NAME}"

}

post {

success {

echo 'Deploy Success'

}

failure {

echo 'Deploy Failed'

}

}

}

}

}