# Gitlab 소스 클론 이후 빌드 및 배포 매뉴얼

작성 - 9기 C107 손효민, 연주원

## 목차

# Gitlab 소스 클론 이후 빌드 및 배포 매뉴얼

## 1. 사용한 JVM, 웹 서버, WAS 제품 등의 종류와 설정값, 버전 (IDE버전 포함) 기재

### 1-1. 종류

- 웹서버 : Nginx

### 1-2. 기술 스택

| 구분 | 기술스택 | 상세내용 | 버전 |
|---|---|---|---|
| 공통 | 형상관리 | Gitlab | - |
| | 이슈관리 | Jira | - |
| | 커뮤니케이션 | Mattermost | 5.3.1 |
| | | Notion | 2.0.49 |
| | | discord | 1.0.55 |
| | | kakaotalk | - |
| | 기타 편의 툴 | Postman | 10.16 |
| | | Termius | 8.0.2 |
| | | MobaXterm | 23.2 |
| | UI/UX | Figma | - |
| | OS | Windows | 10 |
| Data | Python | | 3.11 |
| | IDE | Pycharm | 2023.2.1 |
| Back-End | DB | MySQL | 8.0.33 |
| | | (Docker Image) MySQL | 8.0.29 |
| | | JPA | - |
| | Java | OpenJDK | 11.0.15 |
| | | Spring Boot | 2.4.5 |
| | | Spring 내장 tomcat | 2.4.5 |
| | Build | Gradle | 8.2.1 |
| | IDE | IntelliJ IDEA | 2023.1.4 |
| Front-End | React | React | 18.2.0 |
| | | Redux | 8.1.1 |

| | | | |
|---|---|---|---|
| | | Redux - toolkit | 1.9.5 |
| | | React-router-dom | 6.14.1 |
| | | React-Query | 3.39.3 |
| | Node.js | 18.16.1 | |
| | Axios | 1.4.0 | |
| | Chart.js | 4.4.0 | |
| | Material UI | 5.14.8 | |
| | react-kakao-maps | | |
| | IDE | Visual Studio Code | 1.80.1 |
| Server | 서버 | Ubuntu | 20.04.6 LTS |
| | 플랫폼 | Docker | 24.0.4 |
| | | Docker Compose | 2.20.2 |
| | 배포 | Docker Desktop | 4.21.1 |
| | | Jenkins | 2.414.2 |

## 2. 배포 매뉴얼

### 1) Git clone 및 배포 과정

A. 로컬 컴퓨터에서 Git clone을 받습니다.

```
https://lab.ssafy.com/s09-bigdata-recom-sub2/S09P22C107.git
```

B. AWS EC2 서버에 MySQL DB를 배포합니다.

i. 도커를 이용하여 MySQL을 다음과 같이 실행합니다. 백엔드 배포 전에 반드시 선행되어야 합니다.

ii. 3-5. MySQL 의 단계의 명령어를 실행합니다.

C. 로컬 컴퓨터에서 프론트엔드 프로젝트를 빌드하고 AWS EC2 서버에 배포합니다.

iii. Git clone을 통해 받은 프로젝트의 frontend 디렉토리의 루트 경로에서 3-2. Frontend 의 단계의 명령어를 실행합니다.

iv. Nginx와 react가 함께 배포됩니다.

D. 로컬 컴퓨터에서 백엔드 프로젝트를 빌드하고 AWS EC2 서버에 배포합니다.

i. 프로젝트 폴더 내에 있는 backend 디렉토리의 루트 경로에서 3-3. Backend의 단계의 명령어를 실행합니다.

ii. Backend 경로에 Dockerfile이 있습니다. 이를 이용하여 Docker Container를 통해 프론트엔드를 배포할 준비를 합니다.

E. Backend gradle 의존성

```
plugins {
	id 'java'
	id 'org.springframework.boot' version '2.7.15'
	id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}

group = 'com.ssafy'
version = '0.0.1-SNAPSHOT'

java {
	sourceCompatibility = '11'
}

configurations {
	compileOnly {
		extendsFrom annotationProcessor
	}
}

repositories {
	mavenCentral()
}

dependencies {
	implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
	implementation 'org.springframework.boot:spring-boot-starter-web'
	implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
	implementation 'io.jsonwebtoken:jjwt-impl:0.11.2'
	implementation 'io.jsonwebtoken:jjwt-jackson:0.11.2'
	compileOnly 'org.projectlombok:lombok'
	developmentOnly 'org.springframework.boot:spring-boot-devtools'
	runtimeOnly 'com.mysql:mysql-connector-j'
	annotationProcessor 'org.projectlombok:lombok'
	testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

tasks.named('test') {
	useJUnitPlatform()
}
```

F. AWS EC2의 Nginx 설정과 ssl 인증서 발급 및 적용

- “2. Nginx 설정과 SSL 인증서 발급 및 적용” 단계의 명령어를 실행합니다.
- Nginx 를 다운받고, letsencrypt를 설치하고, 인증서 발급 후 /etc/nginx/sites-available에서 default파일에 nginx 설정을 해준 후, nginx 재시작 명령어를 입력해줍니다.
- 이렇게 실행을 마친다면, http로 80포트 접근시, 443 포트(https)로 리다이렉트 됩니다. 그리고 백엔드 url을 /api/**로 분기처리할 수 있습니다. https://도메인주소 로 접근하면 배포한 웹 페이지에 접속할 수 있게 됩니다.

## 3. 빌드 시 사용되는 환경 변수 등의 주요 내용 상세 기재

### 3-1. 개요

SHERPA 서비스의 배포 환경 및 흐름으로는 팀원들이 개발 완료한 프로젝트를 Gitlab에 push 후 develop branch에 merge 하면 젠킨스에서 Front-end, Back-end를 빌드합니다.

젠킨스에서는 각 프로젝트를 빌드한 후에 Docker 이미지를 만들고, 이를 Docker Hub에 push한 후, 이로부터 서비스에 필요한 이미지를 받아와 컨테이너로 띄웁니다.

서버의 경우에 SSAFY에서 지원받은 AWS EC2 인스턴스로 인프라를 구축하였습니다. 또한 Nginx는 리버스 프록시 서버로 구성했습니다. 그래서 8080포트는 Spring Boot 서버, 3000 포트는 React 서버, 8000 포트는 FastAPI 서버로 설정하여 Load Balancing이 가능하도록 구축하였습니다.

### 3-2. Frontend

- Frontend 프로젝트 내에 Dockerfile 생성 (src나 build 있는 곳)

```
# nginx 이미지를 사용
FROM nginx

# root 에 app 폴더를 생성
RUN mkdir /app

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 3000 포트 오픈
EXPOSE 3000

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

- Frontend 프로젝트 내에 nginx.conf 파일 생성 (src 폴더나 build 있는 곳)

```
server {
    listen 3000;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

3-3. BackEnd

- BackEnd 프로젝트 내에 Dockerfile 생성 (src나 build, greadlew 있는 곳)

  ■ 프로젝트 배포 시 Docker Hub 계정 아이디는 jwyeon, 레포지토리명은 sherpa를 사용했습니다.

```
FROM openjdk:11-jdk

# 컨테이너가 포트 8080 에서 듣도록 설정
EXPOSE 8080

# JAR_FILE 변수 정의
ARG JAR_FILE=./build/libs/adrec-0.0.1-SNAPSHOT.jar

# JAR 파일 메인 디렉토리에 복사
COPY ${JAR_FILE} app.jar

# 시스템 진입점 정의
ENTRYPOINT ["java","-jar","/app.jar"]

ENV TZ=Asia/Seoul
RUN apt-get install -y tzdata
```

3-4. Docker 설치

- Docker 및 Docker Compose 설치

  ■ Docker를 설치합니다.

```
# 오래된 버전 삭제
sudo apt-get remove docker docker-engine docker.io containerd runc

# repository 설정
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg lsb-release
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

# Docker Engine 설치
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io

# 설치 완료 후 버전 확인
$ docker --version
```

- Docker Compose를 설치합니다.

```
# 최신 버전 설정 후 다운로드
$ sudo apt install jq

$ VERSION=$(curl --silent
https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)

$ DESTINATION=/usr/local/bin/docker-compose

$ sudo curl -L
https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-
$(uname -s)-$(uname -m) -o $DESTINATION

$ sudo chmod +x $DESTINATION

# 버전 정보 확인
$ sudo docker-compose --version
```

- 포트 열기 (방화벽 끄기, 나중에 관련 포트 방화벽 허용하기)

```
# 방화벽 끄기
ufw disable

ufw allow ssh
ufw allow 80/tcp
ufw allow 443/tcp
ufw allow 3478/tcp
ufw allow 3478/udp
ufw allow 40000:57000/tcp
ufw allow 40000:57000/udp
ufw allow 57001:65535/tcp
ufw allow 57001:65535/udp
ufw enable
```

1. Letsencrypt 인증서를 받았기 때문에 letsencrypt 의 certificate type으로 명시하고, 인증서 받을 때 작성했던 이메일 주소를 넣었습니다.

2. Nginx 설정과 SSL 인증서 발급 및 적용

- Nginx 설치

```
# nginx 설치
$ sudo apt install nginx

# nginx 상태 확인
$ sudo systemctl status nginx

# nginx 실행 시작
$ sudo systemctl start nginx
```

- SSL 설치

```
# let's Encrypt 설치
sudo apt-get install letsecrypt

# Certbot 설치
sudo apt-get install python3-certbot-nginx

# Certbot 동작
sudo certbot --nginx
```

- AWS EC2 서버의 nginx 설정 파일 – default

```
# nginx 환경 설정
sudo vi /etc/nginx/sites-available/sherpa.conf
```

- Nginx 파일에 SSL 인증서 설정 및 리버스 프록시 구축

```
server {
  listen 80;
  server_name j9c107.p.ssafy.io;
  return 301 https://j9c107.p.ssafy.io$request_uri;


}
server {
  listen 443 ssl http2;
  server_name j9c107.p.ssafy.io;

  # ssl 인증서 적용하기
  ssl_certificate /etc/letsencrypt/live/j9c107.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/j9c107.p.ssafy.io/privkey.pem;
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

  location / {
    proxy_pass http://localhost:3000;
  }

  location /api { # location 이후 특정 url을 처리하는 방법을 정의
    proxy_pass http://localhost:8080; # Request에 대해 어디로 리다이렉트하는지
    proxy_redirect off;
    charset utf-8;

    proxy_http_version 1.1;
    proxy_set_header Connection "upgrade";
    proxy_set_header Upgrade $http_upgrade;
    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;
    proxy_set_header X-NginX-Proxy true;
```

■ Nginx 재시작

Nginx 설정파일을 수정할 경우, nginx의 재시작이 필수입니다.

```
# 저희 프로젝트에서는 default 로 했습니다.
$sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-
enabled/[파일명]

# 다음 명령어에서 successful 이 뜨면 nginx 를 실행할 수 있다.
$sudo nginx -t

# nginx 실행 재시작
$ sudo systemctl restart nginx

# nginx 상태 확인
$ sudo systemctl status nginx
```

2. 기본 포트인 상태에서 정상 접속을 확인한 후 포트 번호를 변경해줘야 합니다.

3-5. MySQL

- Docker 로 MySQL의 이미지를 가져온다

```
$ docker pull mysql:8.0.29
```

- Redis 를 Docker container로 실행합니다. Volume name은 mysql-container로 지정해줬습니다. 또한 Redis의 비밀번호로 c107adrec을 설정했습니다.

```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=<비밀번호> --name <도커 컨테이너 이
름>        mysql:8.0.29        --character-set-server=utf8mb4        --collation-
server=utf8mb4_unicode_ci
```

## 4.배포 시 특이사항

- 젠킨스를 이용한 자동 배포

```
pipeline {
    agent any  // 사용 가능한 에이전트에서 이 파이프라인 또는 해당 단계를 실행

    tools {nodejs "NodeJS 18.16.1"}

    environment {
        DOCKER_IMAGE_NAME_BE = 'jwyeon/sherpa_be' // 도커 허브 레포지토리
        DOCKER_IMAGE_NAME_FE = 'jwyeon/sherpa_fe' // 도커 허브 레포지토리
        DOCKER_IMAGE_NAME_DATA = 'jwyeon/sherpa_data' // 도커 허브 레포지토리
        DOCKER_IMAGE_BE = ''
        DOCKER_IMAGE_FE = ''
        DOCKER_IMAGE_DATA = ''
    }

    stages {
        stage('Prepare') {
            steps {
                sh 'echo "Clonning Repository"'
                git branch: 'develop',
                credentialsId: 'gitlabId',
                url: 'https://lab.ssafy.com/s09-bigdata-recom-sub2/S09P22C107.git'
            }
            post {
                success {
                    sh 'echo "Successfully Cloned Repository"'
                }
                failure {
                    sh 'echo "Fail Cloned Repository"'
                }
            }
        }
        stage('Build Gradle') {
```

5. DB 접속 정보 등 프로젝트에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록

- 백엔드 프로젝트의 application.properties의 DB 관련 설정

```
# jpa

spring.jpa.hibernate.ddl-auto=update

spring.jpa.hibernate.naming.physical-strategy                                    =
org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl



# log

logging.level.com.ssafy.adrec=debug



# account

spring.profiles.include=account



# database

spring.datasource.url=jdbc:mysql://j9c107.p.ssafy.io:3306/adrec?serverTimezone=
UTC&characterEncoding=UTF-8

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.hikari.username=c107

spring.datasource.hikari.password=c107adrec



# jwt

jwt.salt=ssafysecret

jwt.secret=VlwEyVBsYt9V7zq57TejMnVUyzblYcfPQye08f7MGVA9XkHa
```

- 백엔드 프로젝트의 application.properties의 server address 관련 설정

```
#it will be set build date by gradle. if this value is @build.date@, front-
end is development mode
build.date=@build.date@
server.port=8080
server.address=0.0.0.0
server.servlet.contextPath=/api
# Charset of HTTP requests and responses. Added to the "Content-Type"
header if not set explicitly.
server.servlet.encoding.charset=UTF-8
# Enable http encoding support.
server.servlet.encoding.enabled=true
# Force the encoding to the configured charset on HTTP requests and re-
sponses.
server.servlet.encoding.force=true
```