





# 포딩 매뉴얼

## 목차

1. 개발환경
2. 설정파일 및 환경 변수 정보
3. 배포 환경 설정
  - a. 초기 세팅
  - b. SSL
  - c. Nginx
  - d. MySQL
  - e. Jenkins
  - f. BackEnd & FrontEnd & Flask 무중단 배포
  - g. Elastic Search
  - h. RabbitMQ
  - i. 그 외 명령어
4. IntelliJ 환경 설정
5. MySQL WorkBench 설정
6. 외부 서비스

## 1. 개발환경

1. Front-End
  - a. Visual Studio Code 1.74.3
  - b. React 18.2.0
  - c. Redux 4.2.1
  - d. Tailwind CSS
2. Back-End
  - a. IntelliJ IDEA 2022.3.1
  - b. SpringBoot Gradle 2.7.9
  - c. JAVA 11
  - d. Spring Security
  - e. Spring Data JPA
  - f. Swagger Doc 3
  - g. JWT 0.11.5
3. Elastic Search
  - a. Elastic Search 7.16.1

#### 4. DataBase

a. MySQL 8.0.32

#### 5. RabbitMQ

a. RabbitMQ management

#### 6. Flask

a. Python 3.9.10

b. Flask 2.2.3

#### 7. CI/CD

a. Ubuntu 20.04 LTS

b. Docker 23.0.1

c. Docker Compose 1.25.0

d. Jenkins lts

e. Nginx 1.15-alpine

f. SSL

## 2. 설정파일 및 환경 변수 정보

| application.properties

```
server.port=9090

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

# DB Source URL
spring.datasource.url=jdbc:mysql://mysql:3306/sulnaeeum?allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&serverTimezone=Asia/

# DB Username, Password
spring.datasource.username=root
spring.datasource.password=A707sulnaeeumA707

# true 설정 시 JPA 쿼리문 확인 가능
spring.jpa.show-sql=true

# DDL(Create, Alter, Drop) 정의 시 DB의 고유 기능 사용 가능
spring.jpa.hibernate.ddl-auto=update

# JPA의 구현체인 Hibernate가 동작하면서 발생한 SQL의 가독성 높여줌
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.open-in-view=false

logging.level.com.ssafy.sulnaeeum=trace

# Swagger
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/
springdoc.swagger-ui.disable-swagger-default-url=true

# JWT
jwt.header=Authorization
jwt.secret=c2lsdmVybmluZS10ZWNoLXNwcmLuZy1ib290LWp3dC10dXRvcmlhbmC1zZWNYZXQtc2lsdmVybmluZS10ZWNoLXNwcmLuZy1ib290LWp3dC10dXRvcmlhbmC1zZWNY
jwt.token-validity-in-seconds=86400

# Kakao key
kakao.client.id = 8ffe34463577f1799ebd2b1d8b64c61d

# custom proxy header
server.forward-headers-strategy=FRAMEWORK

# RabbitMQ
```

```
spring.rabbitmq.host=j8a707.p.ssafy.io
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

| .env

```
NEXT_PUBLIC_CNU_CHECK_KEY=YKWU2pC8Iz64fL%2BabC3qn0dREZFS7p0o60cGvt3hnNd4Wjj8RKNwv2E2vXyl8QeRF3PmWe%2Fo911rMdUIufwiPw%3D%3D
NEXT_PUBLIC_KAKAO_JS_KEY=c53a68cb92fe2142f2f3e12d799dff15
NEXT_PUBLIC_KAKAO_LOGIN_KEY=8ffe34463577f1799ebd2b1d8b64c61d
NEXT_PUBLIC_KAKAO_REDIRECT_URI='http://j8a707.p.ssafy.io/api/user/kakao/callback'
```

## 3. 배포 환경 설정

### 초기 세팅

#### EC2 접속

```
ssh -i J8A707T.pem ubuntu@j8a707.p.ssafy.io
```

#### Docker 설치

- apt package index 업데이트

```
sudo apt-get update
```

- Repository 등록을 위한 필요 패키지 설치

```
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

- Docker 공식 GPG-Key 추가

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

- Docker Repository 추가

```
echo \
    "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

#### Docker Engine 설치

- Docker engine 설치

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

- Version check

```
docker --version
```

## Docker Compose 설치

- Docker Compose 설치

```
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
```

- docker-compose에 권한을 부여

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

- Version check

```
docker-compose version
```

## SSL

### SSL 인증키 발급

- letsencrypt설치

```
sudo apt-get update  
sudo apt-get install letsencrypt -y
```

- SSL 인증서 발급

```
sudo letsencrypt certonly --standalone --register-unsafely-without-email -d j8a707.p.ssafy.io
```

- 키 확인

📁 /etc/letsencrypt/live/j8a707.p.ssafy.io

👉 fullchain.pem

👉 privkey.pem

## Nginx

### Nginx Docker Compose file 생성

| docker-compose.yml

```
version: '3'  
  
services:  
  nginx:  
    container_name: nginx  
    image: nginx:1.15-alpine  
    restart: always  
    ports:  
      - 80:80  
      - 443:443  
    volumes:  
      - ../data/nginx/conf.d:/etc/nginx/conf.d  
      - ../data/certbot/conf:/etc/letsencrypt
```

./data/certbot/conf 아래에

📁 /etc/letsencrypt/live/{domain}/

👉 fullchain.pem

👉 chain.pem

두 키가 가리키는 원본 키 파일을 ./data/certbot/conf/live/{domain}/ 경로에 COPY하기

## Nginx conf 설정

### | app.conf 생성

```
# 무중단 배포를 위한 로드밸런싱 upstream 서버 설정
upstream frontend {
    server j8a707.p.ssafy.io:3000;
    server j8a707.p.ssafy.io:3001;
}

upstream backend {
    server j8a707.p.ssafy.io:9090;
    server j8a707.p.ssafy.io:9091;
}

upstream flask {
    server j8a707.p.ssafy.io:5000;
    server j8a707.p.ssafy.io:5001;
}

server {
    listen 80;
    listen [::]:80;

    server_name j8a707.p.ssafy.io;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name j8a707.p.ssafy.io;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/j8a707.p.ssafy.io/fullchain1.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8a707.p.ssafy.io/privkey1.pem;
    #include /etc/letsencrypt/options-ssl-nginx.conf;
    #ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # front 설정
    location / {
        #proxy_pass http://j8a707.p.ssafy.io:3000; # :3000
        proxy_pass http://frontend;
        proxy_set_header Host $host:$http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # backend 설정
    location /api {
        #add_header 'Access-Control-Allow-Origin' '*'; # CORS
        #proxy_pass http://j8a707.p.ssafy.io:9090/api; # :9090/api
        proxy_pass http://backend;
        #proxy_set_header X-Forwarded-Prefix /api/;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $server_name;
        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 Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
    }

    location ~ ^/(swagger-ui|v3|csrf) {

```

```

        proxy_pass http://j8a707.p.ssafy.io:9090;
        #proxy_pass http://swagger;
        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;
    }

    # elasticsearch api를 사용하기 위해 추가 설정
    location /es/ {
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
        proxy_pass http://j8a707.p.ssafy.io:9200/;
    }

    # rabbitMQ를 사용하기 위해 추가 설정
    location /mq/ {
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
        proxy_pass http://j8a707.p.ssafy.io:15672/;
        error_page 405 = $uri;
    }

    # flask 서버를 사용하기 위한 추가 설정
    location /flask/{
        add_header 'Access-Control-Allow-Origin' '*';
        add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
        proxy_pass http://flask/;
    }
}

```

## Nginx Docker Compose 실행

```
docker-compose up -d
```

## MySQL

### MySQL Docker Compose file 생성

| docker-compose.yml

```

version: "3"
services:
  mysql:
    image: mysql:8.0

    container_name: mysqlldom
    restart : always
    environment:
      MYSQL_DATABASE: sulnaeeum
      MYSQL_ROOT_PASSWORD: A707sulnaeeumA707
    ports:
      - 3306:3306
    volumes:
      - ./mysql/db:/var/lib/mysql
      - ./mysql/initdb.d:/docker-entrypoint-initdb.d

```

- ⚙️ DataBase Name : sulnaeeum
- ⚙️ UserName : root
- ⚙️ Password : A707sulnaeeumA707

### MySQL Docker Compose 실행

```
docker-compose up -d
```

## Jenkins

### Jenkins Docker Compose file 생성

| docker-compose.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:
      - 8080:8080
    privileged: true
    user: root
```

⚙ ID : sulnaeeum

⚙ Password : a707sulnaeeum

⚙ URL : <http://15.164.245.146:8080/>

### Jenkins Docker Compose 실행

```
docker-compose up -d
```

### Jenkins 플러그인 설치

DashBoard > Manager JenKins > Plugin Manager > Installed plugins

- GitLab & Docker plugin 설치

### JenKins 컨테이너 안에 Docker 설치

- Jenkins 컨테이너 접속

```
docker exec -it jenkins /bin/bash
```

- Docker 설치

```
apt-get update -y

apt-get install -y

apt-get install docker.io -y

# version check
docker -v
```

### JenKins 컨테이너 안에 Docker Compose 설치

- Jenkins 컨테이너 접속



```
docker exec -it jenkins /bin/bash
```

- Docker Compose 설치

```
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
```

- docker-compose에 권한을 부여

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

- Version check


```
docker-compose version
```

## JenKins 환경 설정


- item 생성

Enter an item name


> A job already exists with the name 'sulnaeeum'

 **Freestyle project**


이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

 **Pipeline**


Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **Multi-configuration project**


다양한 환경에서의 테스트, 플러그인 특정 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

 **Folder**

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**

Creates a set of Pipeline projects according to detected branches in one SCM repository.

 **Organization Folder**

Creates a set of multibranch project subfolders by scanning for repositories.

OK

If you want to create a new item from other existing, you can use this option:

- 생성한 item의 Configure
  - 연결할 Git URL & 비밀번호 및 기타 작성
  - 빌드 유발 설정
  - 빌드 시 실행될 코드 shell script 작성

소스 코드 관리

**Git** ?

Repositories ?

Repository URL ?

https://lab.ssfy.com/s08-bigdata-recom-sub2/S08P22A707.git

Credentials ?

mihee78952/\*\*\*\*\*

Add

Jenkins

고급

Add Repository

#### 빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://15.164.245.146:8080/project/sulnaeum ?

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request ?

☐ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

☒ Approved Merge Requests (EE-only)

☒ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급

#### Build Steps

Execute shell ?

Command

See [the list of available environment variables](#)

```
echo "jenkinsbuild started..."
pwd
./deploy_uninterrupted.sh
```

고급

Add build step

## Git Webhooks 설정

## Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

### URL

`http://15.164.245.146:8080/project/sulnaeeum`

URL must be percent-encoded if it contains one or more special characters.

### Secret token

.....

Used to validate received payloads. Sent with the request in the `X-GitLab-Token` HTTP header.

### Trigger

#### ☒ Push events

matter

Push to the repository.

#### ☐ Tag push events

A new tag is pushed to the repository.

#### ☐ Comments

A comment is added to an issue or merge request.

#### ☐ Confidential comments

A comment is added to a confidential issue.

#### ☐ Issues events

An issue is created, updated, closed, or reopened.

#### ☐ Confidential issues events

A confidential issue is created, updated, closed, or reopened.

#### ☐ Merge request events

A merge request is created, updated, or merged.

#### ☐ Job events

A job's status changes.

#### ☐ Pipeline events

A pipeline's status changes.

#### ☐ Wiki page events

A wiki page is created or updated.

#### ☐ Deployment events

A deployment starts, finishes, fails, or is canceled.

#### ☐ Feature flag events

A feature flag is turned on or off.

#### ☐ Releases events

A release is created or updated.

### SSL verification

☒ Enable SSL verification

Add webhook

## BackEnd & FrontEnd & Flask 무중단 배포

### 블루-그린 방식의 무중단 배포 구현

- 신규 서버가 배포 완료 상태가 되기 까지 기존 서버다 동작하다, 신규 서버가 준비되면 로드 밸런서의 방향을 변경하는 방식
- 배포 속도가 빠르며, 장애가 발생했을 때 로드 밸런서가 기존 서버를 가리키면 되기 때문에 롤백이 쉬움
- 추가적인 서버로 인한 비용이 단점

### Nginx 로드 밸런싱

두 개의 포트를 변경하며 사용할 수 있도록, nginx의 upstream을 활용하여 로드밸런싱 적용

#### app.conf 추가된 설정

```
# 무중단 배포를 위한 로드밸런싱 upstream 서버 설정
upstream frontend {
    server j8a707.p.ssafy.io:3000;
    server j8a707.p.ssafy.io:3001;
}

upstream backend {
```

```

server j8a707.p.ssafy.io:9090;
server j8a707.p.ssafy.io:9091;
}

upstream flask {
server j8a707.p.ssafy.io:5000;
server j8a707.p.ssafy.io:5001;
}

```

## SpringBoot & React & Flsk Dockerfile 생성

### Dockerfile (springboot)

```

FROM openjdk:11-jdk
ARG JAR_FILE=build/libs/*.jar
ARG JAR_FILE=/BE/sulnaeeum/build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]

```

### Dockerfile (React)

```

FROM node:19.3.0
# 경로 설정하기
WORKDIR /app

# package.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json .

# 명령어 실행 (의존성 설치)
RUN npm install
# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .

# 빌드
RUN npm run build

# npm start 스크립트 실행
CMD ["npm", "run", "start"]

```

### Dockerfile (Flask)

```

FROM python:3.9.10

WORKDIR /app
COPY . /app

RUN /usr/local/bin/python -m pip install --upgrade pip
RUN pip3 install --upgrade --no-deps --force-reinstall -r requirements.txt

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

```

## SpringBoot & React & Flsk Docker Compose file 파일 생성

### docker-compose.blue.yml

```

version: '3'

services:
  springboot:
    build:
      context: ../BE/sulnaeeum
    ports:
      - 9090:9090
  react:
    build: ../Front/sulnaeeum

```

```

        ports:
            - 3000:3000

    flask:
        build: ./BE/data/Flask
        ports:
            - 5000:5000

# mysql의 network인 ubuntu_default network 사용
networks:
    default:
        external:
            name: ubuntu_default

```

## docker-compose.green.yml

```

version: '3'

services:
    springboot:
        build:
            context: ./BE/sulnaeeum
            container_name: springboot
        ports:
            - 9091:9090

    react:
        build: ./Front/sulnaeeum
        container_name: react

        ports:
            - 3001:3000

    flask:
        build: ./BE/data/Flask
        container_name: flask
        ports:
            - 5001:5000

# mysql의 network인 ubuntu_default network 사용
networks:
    default:
        external:
            name: ubuntu_default

```

## Jenkins 기반 자동 배포화 될 시, 실행될 .sh 파일 생성

### deploy\_uninterrupted.sh

```

echo '빌드 시작'
cd BE/sulnaeeum
chmod +x gradlew
./gradlew build
cd ../../

DOCKER_APP_NAME=sulnaeeum

# Blue 를 기준으로 현재 떠있는 컨테이너를 체크한다.
EXIST_BLUE=$(docker-compose -p ${DOCKER_APP_NAME}-blue -f docker-compose.blue.yml ps | grep Up)

# 컨테이너 스위칭
if [ -z "$EXIST_BLUE" ]; then
    # 컨테이너 실행
    echo "blue up"
    docker-compose -p ${DOCKER_APP_NAME}-blue -f docker-compose.blue.yml up -d

    BEFORE_COMPOSE_COLOR=green
    AFTER_COMPOSE_COLOR=blue
else
    # 컨테이너 실행
    echo "green up"
    docker-compose -p ${DOCKER_APP_NAME}-green -f docker-compose.green.yml up -d

    BEFORE_COMPOSE_COLOR=blue
    AFTER_COMPOSE_COLOR=green
fi

sleep 20

```

```
# 새로운 컨테이너가 제대로 뒀는지 확인
EXIST_AFTER=$(docker-compose -p ${DOCKER_APP_NAME}-${AFTER_COMPOSE_COLOR} -f docker-compose.${AFTER_COMPOSE_COLOR}.yml ps | grep Up)
if [ -n "$EXIST_AFTER" ]; then
    # 이전 컨테이너 종료
    docker-compose -p ${DOCKER_APP_NAME}-${BEFORE_COMPOSE_COLOR} -f docker-compose.${BEFORE_COMPOSE_COLOR}.yml down
    # 이전 이미지 지우기
    docker rmi sulnaeeum-${BEFORE_COMPOSE_COLOR}_flask
    docker rmi sulnaeeum-${BEFORE_COMPOSE_COLOR}_springboot
    docker rmi sulnaeeum-${BEFORE_COMPOSE_COLOR}_react

    echo "$BEFORE_COMPOSE_COLOR down"
fi
```

## Elastic Search

### Elastic Search Dockerfile 생성

#### Dockerfile

```
FROM elasticsearch:7.16.1

# elasticsearch안에서 nori 설치하기
RUN bin/elasticsearch-plugin install analysis-nori
```

### Elastic Search Docker Compose file 생성

#### docker-compose.yml

```
version: '3'

services:
  elasticsearch:
    build: ./ # elasticsearch Dockerfile 위치
    container_name: es
    environment:
      - node.name=es
      - cluster.name=es-docker-cluster
      - bootstrap.memory_lock=true
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ports:
      - 9200:9200
      - 9300:9300
    volumes:
      - /usr/share/elasticsearch/data
```

### Elastic Search Docker Compose 실행

```
docker-compose up -d
```

### Elastic Search Check

← → ↺ j8a707.p.ssafy.io/es/

```
{
  "name" : "es",
  "cluster_name" : "es-docker-cluster",
  "cluster_uuid" : "r9Dw9W5RSamFA9kcjJPtxQ",
  "version" : {
    "number" : "7.16.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "5b38441b16b1ebb16a27c107a4c3865776e20c53",
    "build_date" : "2021-12-11T00:29:38.865893768Z",
    "build_snapshot" : false,
    "lucene_version" : "8.10.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

## RabbitMQ

### RabbitMQ Docker Compose file 생성

| docker-compose.yml

```
version: '3'
services:
  rabbitmq:
    image: rabbitmq:management
    container_name: rabbitmq
    ports:
      - 5672:5672
    environment:
      RABBITMQ_ERLANG_COOKIE: "RabbitMQ-My-Cookies"
      RABBITMQ_DEFAULT_USER: "guest"
      RABBITMQ_DEFAULT_PASS: "guest"

networks:
  default:
    external:
      name: ubuntu_default
```

### RabbitMQ Docker Compose 실행

```
docker-compose up -d
```

## 그 외 명령어

```
# 서버 용량 확인
df -h

# 실행중인 컨테이너
sudo docker ps -a

# 다운로드 이미지 목록
sudo docker image ls

# 이미지 생성
sudo docker build -t {이미지이름} .

# 이미지 삭제
sudo docker rmi {이미지이름}

# 네트워크
sudo docker network create
sudo docker network connect {network_name} {container_name}

# 네트워크 구성 확인
docker network ls

# 컨테이너 생성 & 실행
```

```

sudo docker run --name={컨테이너이름} {hostPort}:{containerPort} {이미지이름}:{버전}

# 컨테이너 중지
docker stop containername

# 컨테이너 삭제
sudo docker rm {컨테이너아이디}

# 컨테이너 접속
docker exec -it containername bash|sh

# 컨테이너 ip 등 정보 확인
docker container inspect test

# 도커 컴포즈 실행
docker-compose up -d

# 도커 컴포즈 stop & restart
docker-compose stop
docker-compose restart

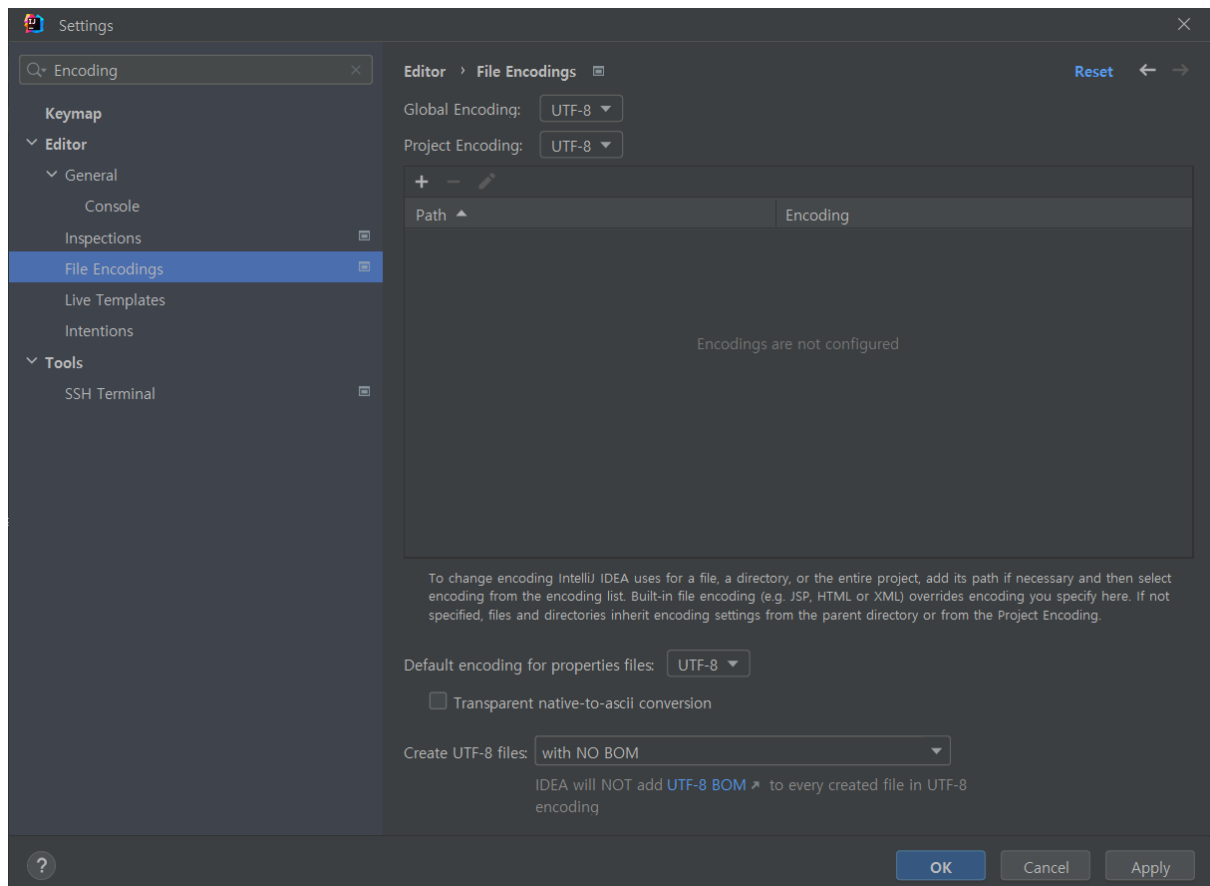
```

## 4. IntelliJ 환경 설정

### UTF-8 설정

1) File → Settings → File Encodings

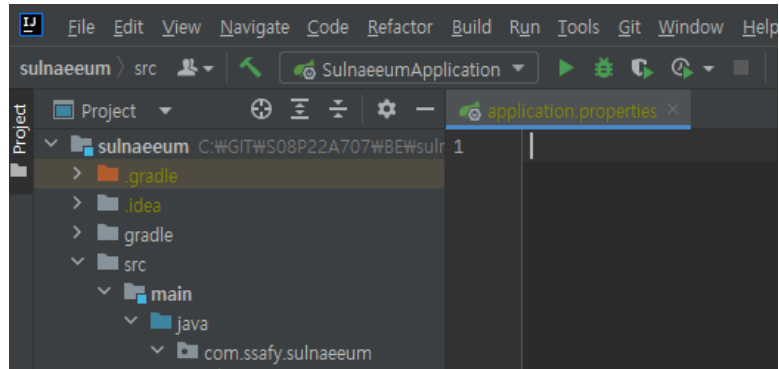
Global Encoding Project Encoding Default encoding for properties files 를 UTF-8로 변경



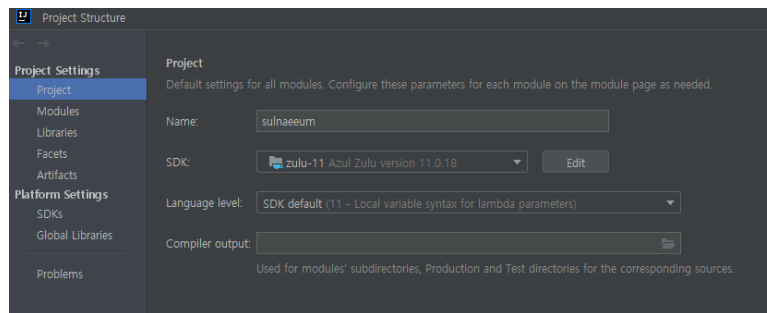
### JAVA 11버전 설정

1) File → Project Structure 클릭

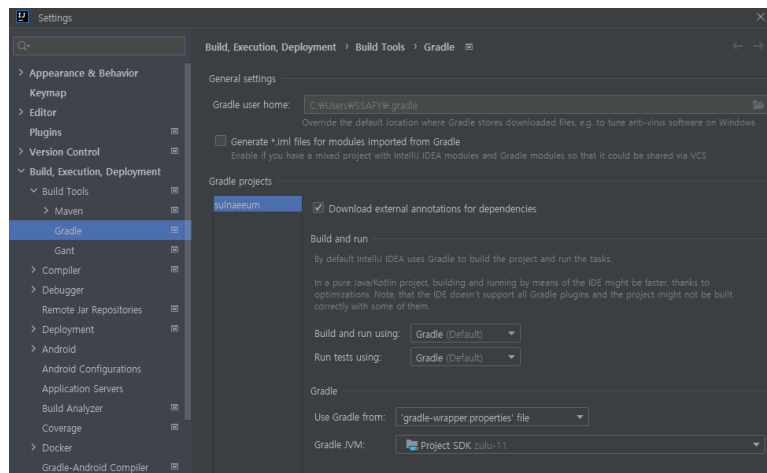




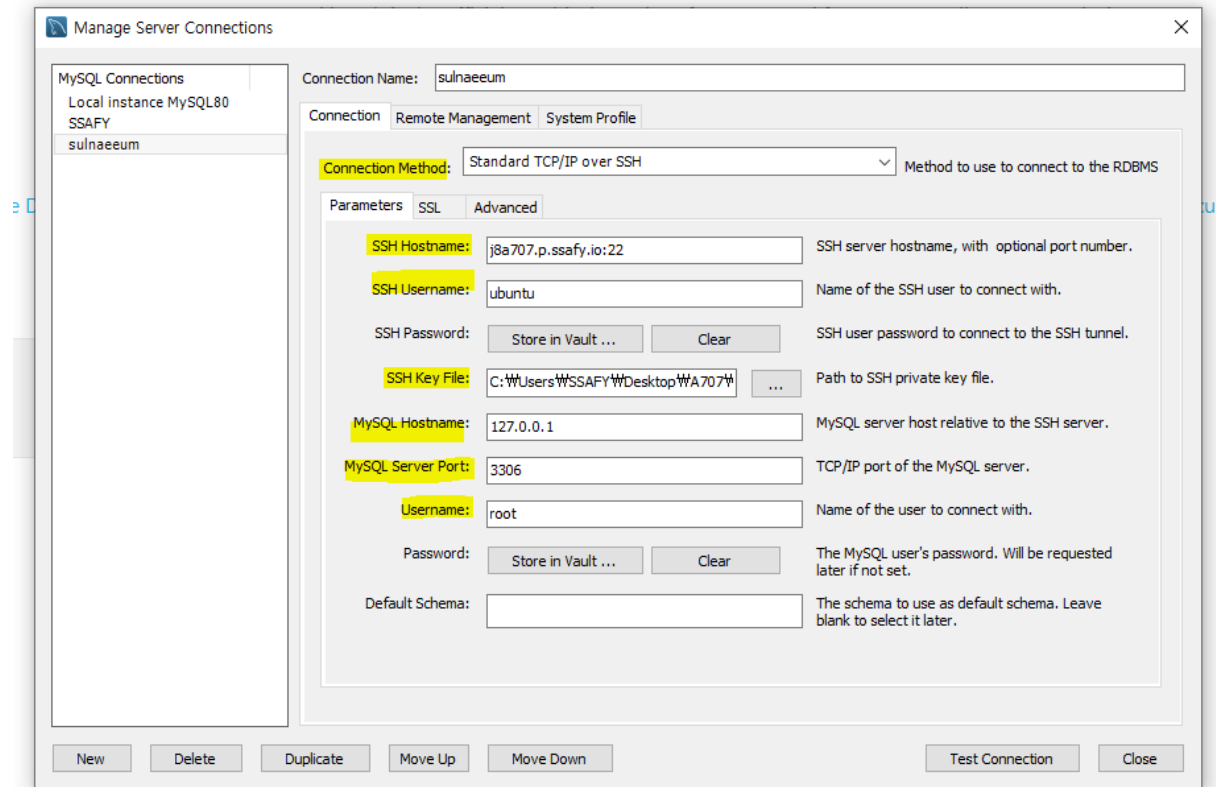
2) SDK를 위에서 설치했던 zulu-11로 설정 & Language level을 SDK default로 설정



3) File → Settings → Build, Extensions, Deployment → Gradle → Gradle JVM을 JDK 11로 설정



## 5. MySQL WorkBench 설정



SSH Key File 📁 EC2 pem 키

Test Connection 누른 후, MySQL 비밀번호 입력하면 Connection 성공

## 6. 외부 서비스

### Kakao Login & Message 서비스

| application.properties

```
# Kakao key
kakao.client.id = 8ffe34463577f1799ebd2b1d8b64c61d
```

| .env

```
NEXT_PUBLIC_KAKAO_JS_KEY=c53a68cb92fe2142f2f3e12d799dff15
NEXT_PUBLIC_KAKAO_LOGIN_KEY=8ffe34463577f1799ebd2b1d8b64c61d
NEXT_PUBLIC_KAKAO_REDIRECT_URI='http://j8a707.p.ssafy.io/api/user/kakao/callback'
```

### 사업자 인증

| .env

```
NEXT_PUBLIC_CNU_CHECK_KEY=YKWu2pC8Iz64fL%2BabC3qn0dREZFS7p0o60cGvt3hnNd4Wjj8RKNwv2E2vXyl8QeRF3PmWe%2Fo911rMdUIuFiPw%3D%3D
```