포팅매뉴얼

목차

---

1. 개발환경

1. Front-End
    a. Visual Studio Code `1.74.3`
    b. TypeScript `4.9.5`
    c. React `18.2.0`
    d. Redux `4.2.1`
    e. Tailwind CSS
    f. Three.js
    g. Blender

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`

h. Spring WebSocket

3. Elastic Search
    a. Elastic Search `7.16.1`

4. DataBase
    a. MySQL `8.0.32`
    b. S3

5. RabbitMQ
    a. RabbitMQ `management`

6. FastAPI
    a. Python `3.9.10`
    b. FastAPI `0.95.1`
    c. OpenCV `4.7.0.72`
    d. OpenAI `0.27.4`
    e. Python `3.6.5`
    f. tensorflow `1.12.0`
    g. Keras `2.2.4`

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/Worldy?allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&serverTimezone=Asia/Seoul

# DB Username, Password
spring.datasource.username=root
spring.datasource.password=A507WorldyA507

# JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.open-in-view=false

# log
```

```
logging.level.com.ssafy.worldy=trace

# JWT
jwt.header=Authorization
jwt.secret=c2lsdmVybmluZS10ZWNoLXNwcmluZy1ib290LWp3dC10dXRvcmlhbC1zZWNyZXQtc2lsdmVybmluZS10ZWNoLXNwcmluZy1ib290LWp3dC10dXRvcmlhbC1zZWNyZXQK
jwt.token-validity-in-seconds=86400

# kakao login
kakao.client.id=01f7a5ec071370f48c515e47f8e79b5c

# RabbitMQ
spring.rabbitmq.host=k8a507.p.ssafy.io
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

# Redis
spring.redis.host=k8a507.p.ssafy.io
spring.redis.port=6379

# FastAPI url
fastapi.exchange-rate.url=https://k8a507.p.ssafy.io/crawling/exchange_rate/
fastapi.hidden-catch.url=https://k8a507.p.ssafy.io/hidden_catch/
```

.env

```
REACT_APP_BASE_URL=https://k8a507.p.ssafy.io/api
REACT_APP_BASE_URL_SHORTER=https://k8a507.p.ssafy.io
REACT_APP_KAKAO_KEY=c6f82fc3b98485b2394889a33664e793
```

crawling/config.py

```
# OPEN AI API KEY
OPEN_AI_API_KEY = "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"

# OWM API KEY
OWM_API_KEY = "e3e3fac54ada85280f858e626598f203"

# MYSQL URL
MYSQL_URL = "mysql"

# MYSQL USER
MYSQL_USER = "root"

# MYSQL PASSWORD
MYSQL_PASSWORD = "A507WorldyA507"

# MYSQL DB
MYSQL_DB = "Worldy"
```

hidden_catch/config.py & pconv_model/config.py

```
# AWS ACCESS KEY ID
AWS_ACCESS_KEY_ID ="AKIA5LR33YUVXC3XRBVF"

# AWS SECRET ACCESS KEY
AWS_SECRET_ACCESS_KEY = "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"

# AWS REGION
AWS_DEFAULT_REGION = "ap-northeast-2"

# BUCKET NAME
BUCKET_NAME = "worldy-soft"

# MYSQL URL
MYSQL_URL = "mysql"

# MYSQL USER
MYSQL_USER = "root"
```

```
# MYSQL PASSWORD
MYSQL_PASSWORD = "A507WorldyA507"

# MYSQL DB
MYSQL_DB = "Worldy"
```

---

3. 배포 환경 설정

초기 세팅

EC2 접속

```
ssh -i K8A507T.pem ubuntu@k8a507.p.ssafy.io
```

Docker 설치

- apt package index 업데이트

```
sudo apt-get update
```

- Repostory 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 k8a507.p.ssafy.io:3000;
        server k8a507.p.ssafy.io:3001;
}

upstream backend {
        server k8a507.p.ssafy.io:9090;
        server k8a507.p.ssafy.io:9091;
}

upstream crawling {
        server k8a507.p.ssafy.io:8000;
        server k8a507.p.ssafy.io:8001;
}

upstream hidden {
        server k8a507.p.ssafy.io:8002;
        server k8a507.p.ssafy.io:8003;
}

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

     server_name k8a507.p.ssafy.io;

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

server {
     listen 443 ssl;
     server_name k8a507.p.ssafy.io;
     server_tokens off;

     ssl_certificate /etc/letsencrypt/live/k8a507.p.ssafy.io/fullchain.pem;
     ssl_certificate_key /etc/letsencrypt/live/k8a507.p.ssafy.io/privkey.pem;
     #include /etc/letsencrypt/options-ssl-nginx.conf;
     #ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

     location / {
             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;
     }

     location /api {
             proxy_pass http://backend;
             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 /mq/ {
             add_header 'Access-Control-Allow-Origin' '*';
             add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
             proxy_pass http://k8a507.p.ssafy.io:15672/;
             error_page 405 = $uri;
     }

     location /es/ {
             add_header 'Access-Control-Allow-Origin' '*';
             add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
             proxy_pass http://k8a507.p.ssafy.io:9200/;
     }

     location /crawling {
             add_header 'Access-Control-Allow-Origin' '*';
             add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
             proxy_pass http://crawling;
     }
```

```
    location /hidden_catch {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Headers' 'X-Requested-With, Content-Type';
            proxy_pass http://hidden;
    }
}
```

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: mysql
    restart : always
    environment:
      MYSQL_DATABASE: Worldy
      MYSQL_ROOT_PASSWORD: A507WorldyA507
    ports:
      - 3306:3306
    volumes:
      - ./mysql/db:/var/lib/mysql
      - ./mysql/initdb.d:/docker-entrypoint-initdb.d
```

⚙️ DataBase Name : Worldy

⚙️ UserName : root

⚙️ Password : A507WorldyA507

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 : Worldy

⚙️ Password : A507WorldyA507

⚙️ URL : http://15.164.245.146:8080/

Jenkins Docker Compose 실행

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

Jenkins 플러그인 설치

DashBoard > Manager JenKins > Plugin Manager > Imstalled 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

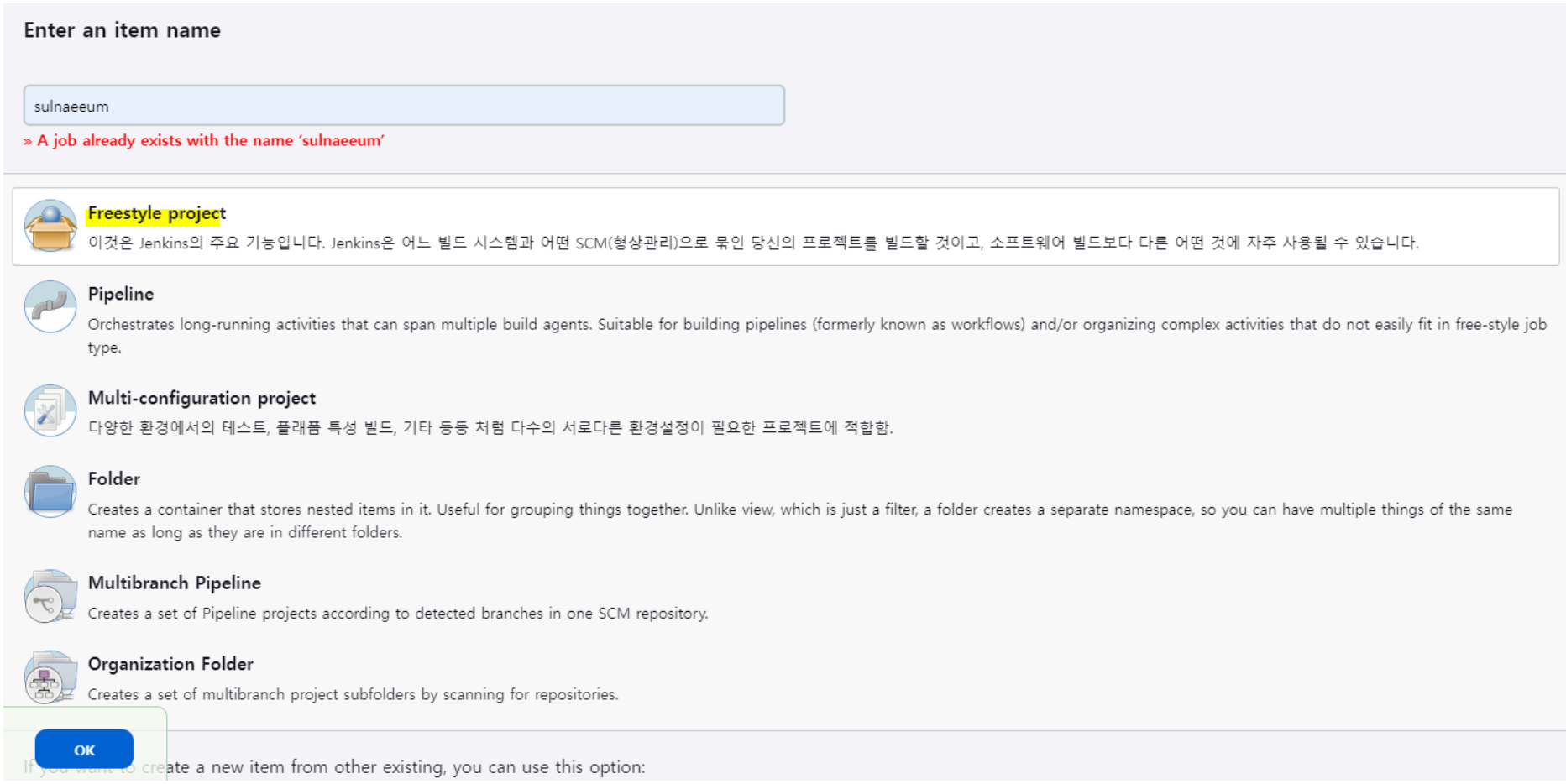
```
sudo chmod +x /usr/local/bin/docker-compose
```

- Version check

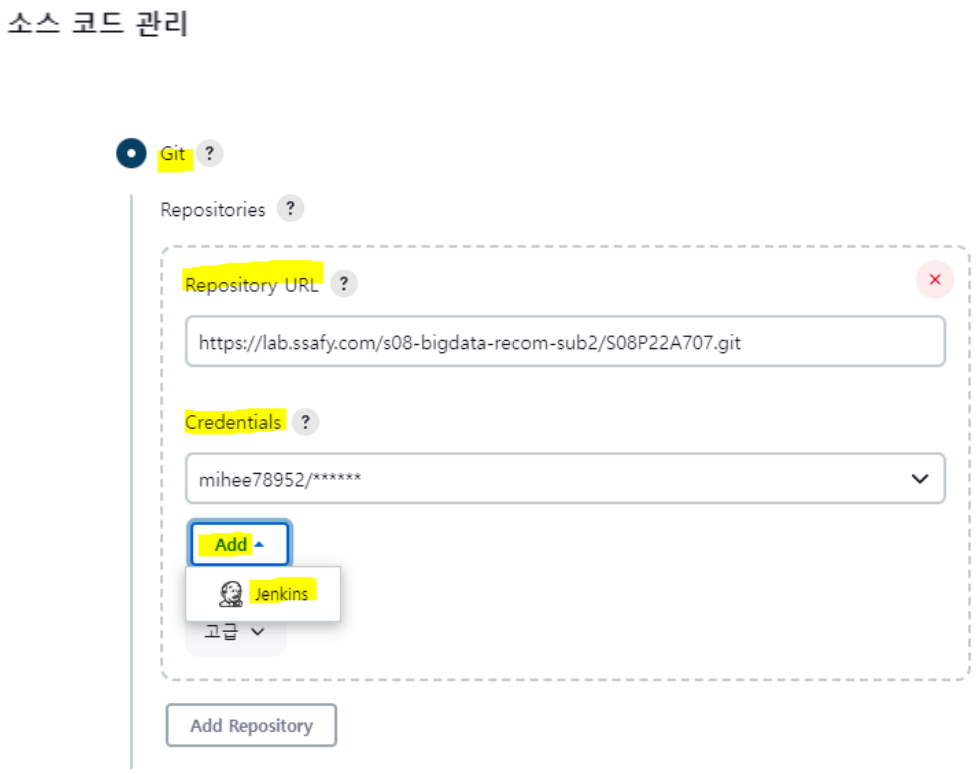
```
docker-compose version
```

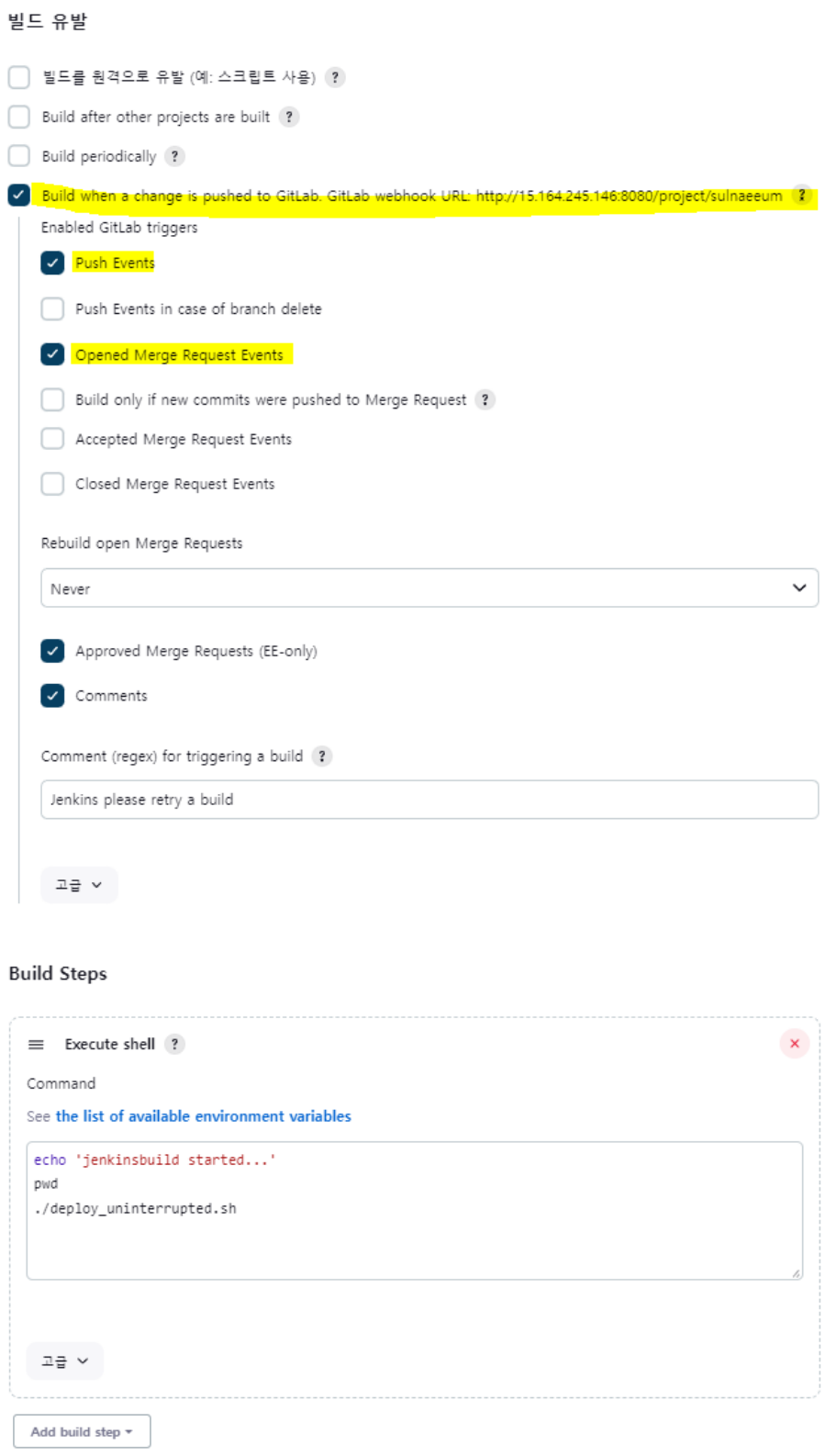
JenKins 환경 설정

- item 생성

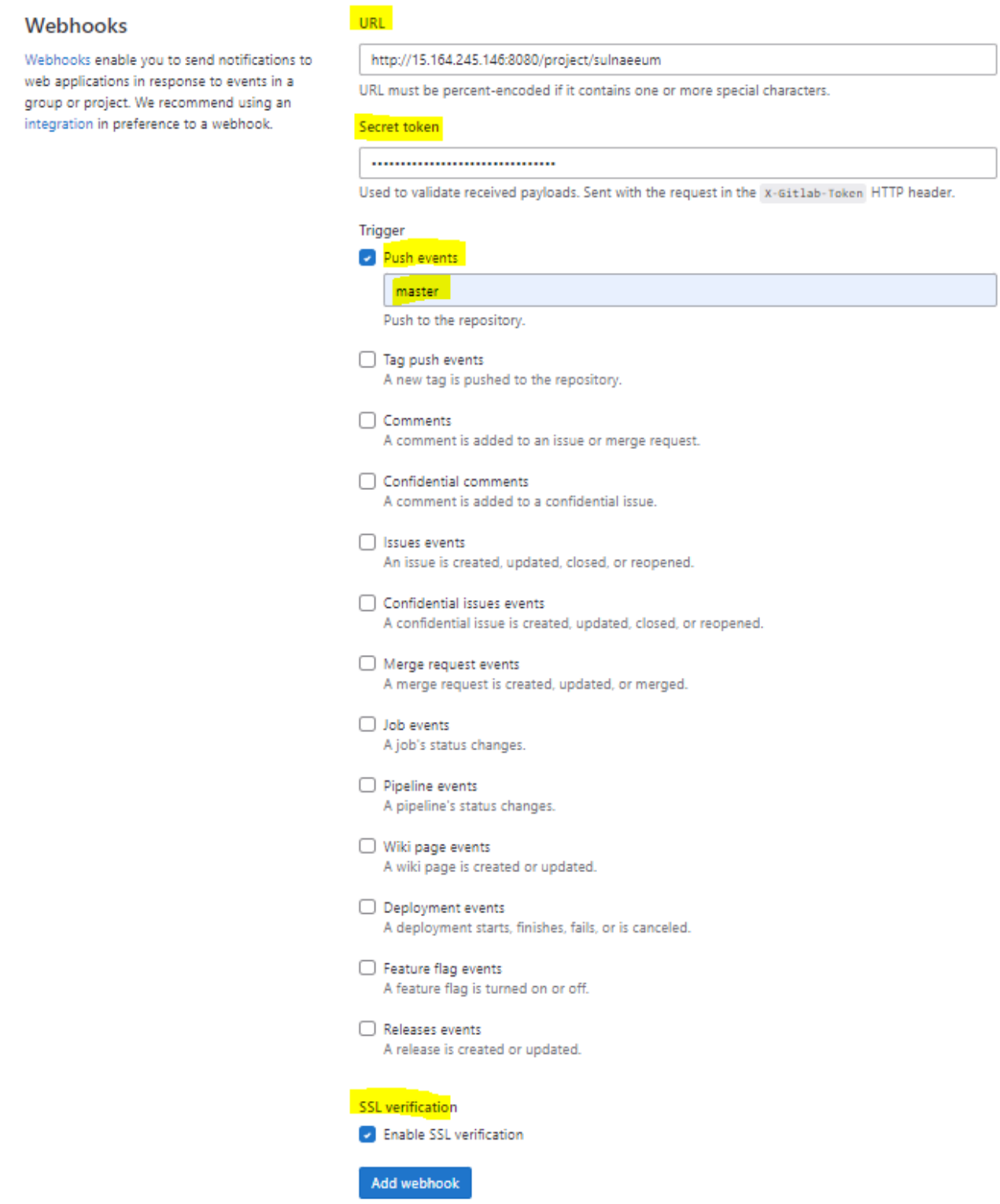


- 생성한 item의 Configure
  - 연결할 Git URL & 비밀번호 및 기타 작성
  - 빌드 유발 설정
  - 빌드 시 실행될 코드 shell script 작성





Git Webhooks 설정



BackEnd & FrontEnd & Flask 블루 그린 방식 무중단 배포

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 k8a507.p.ssafy.io:3000;
        server k8a507.p.ssafy.io:3001;
}

upstream backend {
        server k8a507.p.ssafy.io:9090;
        server k8a507.p.ssafy.io:9091;
}

upstream crawling {
        server k8a507.p.ssafy.io:8000;
        server k8a507.p.ssafy.io:8001;
}

upstream hidden {
        server k8a507.p.ssafy.io:8002;
```

```
        server k8a507.p.ssafy.io:8003;
}
```

SpringBoot & React & Flsk Dockerfile 생성

Dockerfile (springboot)

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

Dockerfile (React)

```
FROM node:18.13.0
# 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 (FastAPI - crawling)

```
FROM python:3.9.10

WORKDIR /app/

COPY ./* /app/
COPY ./requirements.txt /app/

RUN apt-get update && \
    apt-get install -y gnupg wget curl unzip --no-install-recommends && \
    wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - && \
    echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google.list && \
    apt-get update -y && \
    apt-get install -y google-chrome-stable && \
    CHROMEVER=$(google-chrome --product-version | grep -o "[^\.]*\.[^\.]*\.[^\.]*") && \
    DRIVERVER=$(curl -s "https://chromedriver.storage.googleapis.com/LATEST_RELEASE_$CHROMEVER") && \
    wget -q --continue -P /chromedriver "http://chromedriver.storage.googleapis.com/$DRIVERVER/chromedriver_linux64.zip" && \
    unzip /chromedriver/chromedriver* -d /chromedriver

RUN pip install --upgrade pip
RUN pip3 install --upgrade --no-deps --force-reinstall -r requirements.txt

CMD uvicorn --host=0.0.0.0 --port 8000 main:app
```

Dockerfile (FastAPI - hidden_catch)

```
FROM python:3.9.10

WORKDIR /app/

COPY ./ /app/
```

```
COPY ./requirements.txt /app/

RUN ls --recursive /app/
RUN apt-get update
RUN apt-get -y install libgl1-mesa-glx
RUN pip install --upgrade pip
RUN pip3 install --upgrade --no-deps --force-reinstall -r requirements.txt

CMD uvicorn --host=0.0.0.0 --port 8000 main:app
```

Dockerfile (FastAPI - pconv_model)

```
FROM python:3.6.5

WORKDIR /app/

COPY ./ /app/
COPY ./requirements.txt /app/

RUN ls --recursive /app/
RUN pip install --upgrade pip
RUN pip3 install --upgrade --no-deps --force-reinstall -r requirements.txt

CMD uvicorn --host=0.0.0.0 --port 8000 main:app
```

SpringBoot & React & Flsk Docker Compose file 파일 생성

docker-compose.blue.yml

```
version: '3.3'
services:
        springboot:
                build:
                        context: ./BE/worldy
                container_name: springboot_blue
                ports:
                        - 9090:9090
        react:
                build: ./FE/worldy
                container_name: react_blue
                ports:
                        - 3000:3000
        crawling:
                build: ./BE/crawling
                container_name: crawling_blue
                ports:
                        - 8000:8000
        hidden:
                build: ./BE/hidden_catch
                container_name: hidden_blue
                ports:
                        - 8002:8000
        ai:
                build: ./BE/pconv_model
                container_name: ai_blue
                ports:
                        - 8004:8000
networks:
        default:
                external:
                        name: ubuntu_default

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

docker-compose.green.yml

```
version: '3.3'
services:
        springboot:
                build:
                        context: ./BE/worldy
                container_name: springboot_green
                ports:
                        - 9091:9090
        react:
                build: ./FE/worldy
                container_name: react_green
                ports:
                        - 3001:3000
        crawling:
                build: ./BE/crawling
                container_name: crawling_green
                ports:
                        - 8001:8000
        hidden:
                build: ./BE/hidden_catch
                container_name: hidden_green
                ports:
                        - 8003:8000
        ai:
                build: ./BE/pconv_model
                container_name: ai_green
                ports:
                        - 8005:8000
networks:
        default:
                external:
                        name: ubuntu_default

# 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/worldy
chmod +x gradlew
./gradlew build
cd ../../

DOCKER_APP_NAME=worldy

EXIST_BLUE=$(docker-compose -p ${DOCKER_APP_NAME}-blue -f docker-compose.blue.yml ps | grep Up)

if [ -z "$EXIST_BLUE" ]; then
        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
        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

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
docker-compose -p ${DOCKER_APP_NAME}-${BEFORE_COMPOSE_COLOR} -f docker-compose.${BEFORE_COMPOSE_COLOR}.yml down
docker rmi worldy-${BEFORE_COMPOSE_COLOR}_springboot
docker rmi worldy-${BEFORE_COMPOSE_COLOR}_react
docker rmi worldy-${BEFORE_COMPOSE_COLOR}_crawling
docker rmi worldy-${BEFORE_COMPOSE_COLOR}_hidden
docker rmi worldy-${BEFORE_COMPOSE_COLOR}_ai
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

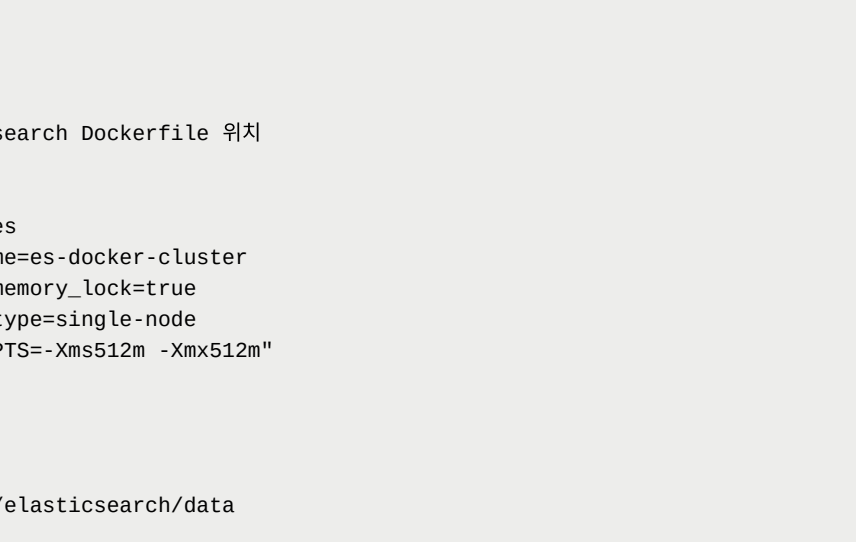
```
version: '3'

services:e
        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



RabbitMQ

실시간으로 게임 매칭을 시도하는 사람들을 mmr과 레벨을 기준으로 나눠준다.

RabbitMQ Docker Compose file 생성

> docker-compose.yml

```
version: '3'
services:
        rabbitmq:
                image: rabbitmq:management
                container_name: rabbitmq
                ports:
                        - 5672:5672
                        - 15672:15672
                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
```

RabbitMQ Queue & Exchange & Binding 설정

```
# Queue 생성
rabbitmqadmin declare exchange name=sulnaeeum.exchange type=direct

# Exchange 생성
rabbitmqadmin declare queue name=sulnaeeum.queue

# Queue&Exchange Binding 설정
rabbitmqadmin declare binding source="sulnaeeum.exchange" destination_type="queue" destination="sulnaeeum.queue" routing_key="sulnaeeum.key"
```

그 외 명령어

```
# 서버 용량 확인
df -h

# 실행중인 컨테이너
sudo docker ps -a

# 다운받은 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
```

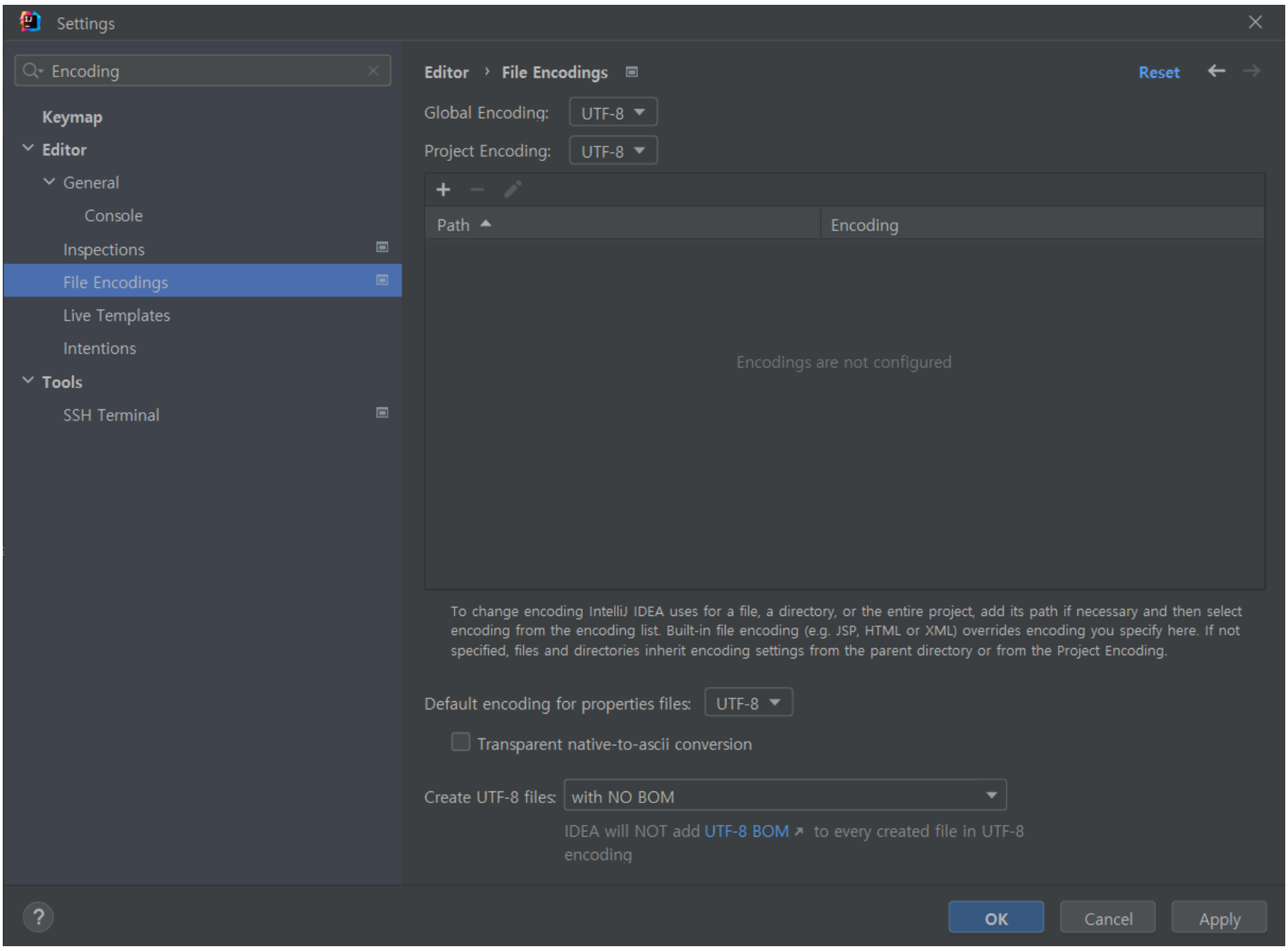
---

4. IntelliJ 환경 설정

UTF-8 설정

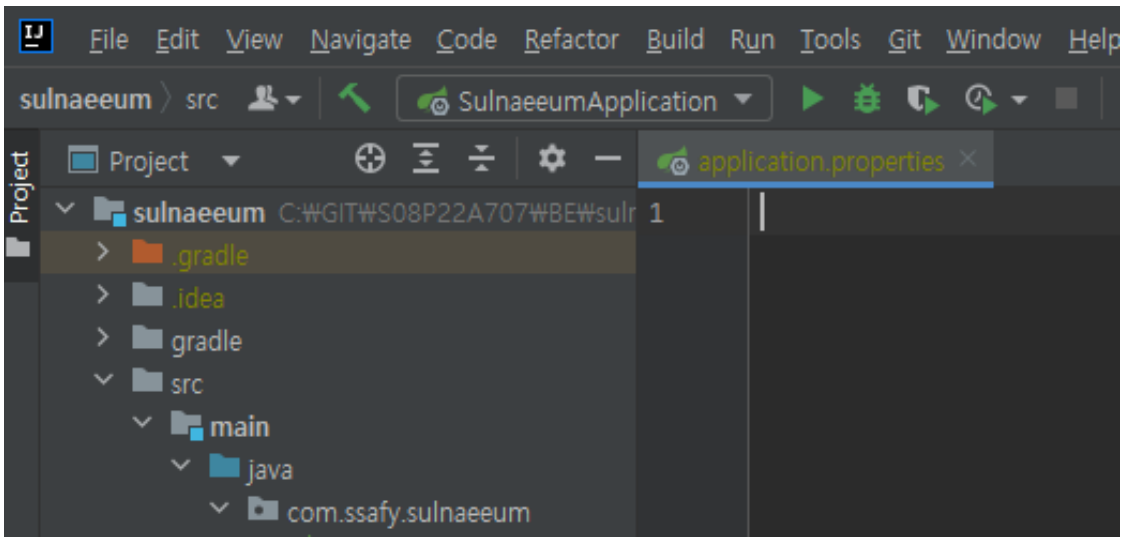
1) File → Settings → File Encodings

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

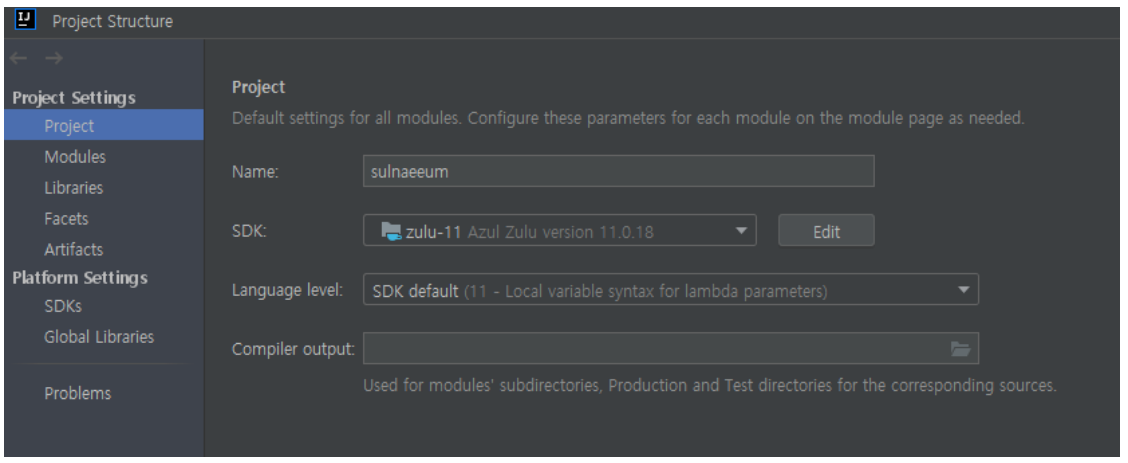


JAVA 11버전 설정

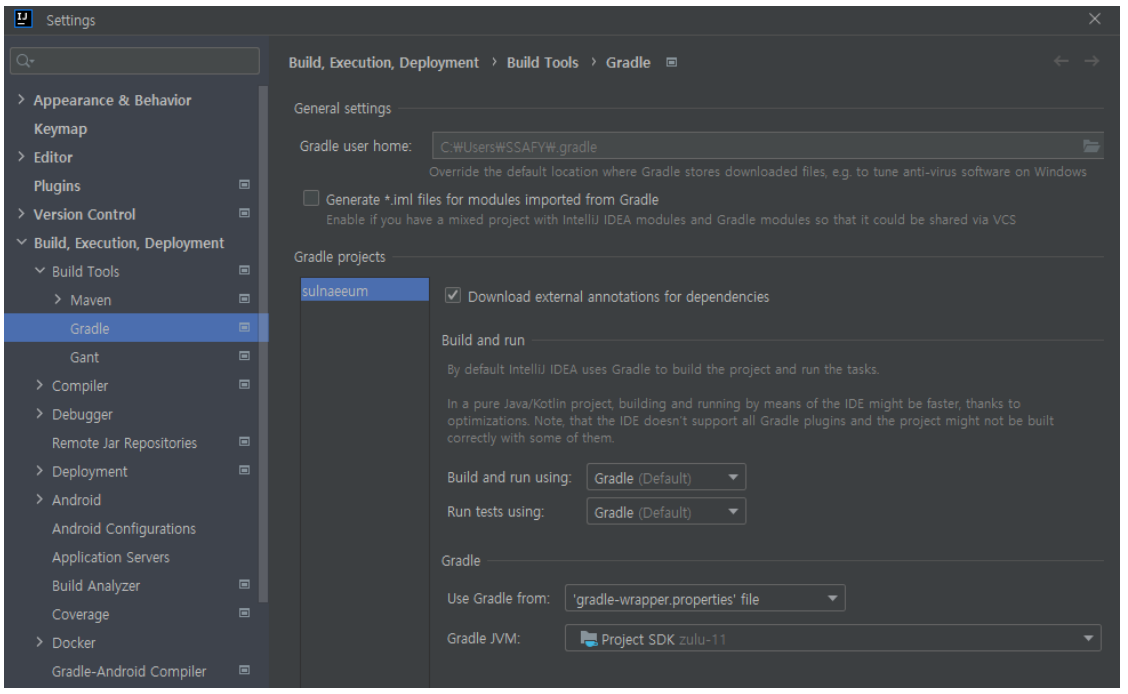
1) File → Project Structure 클릭



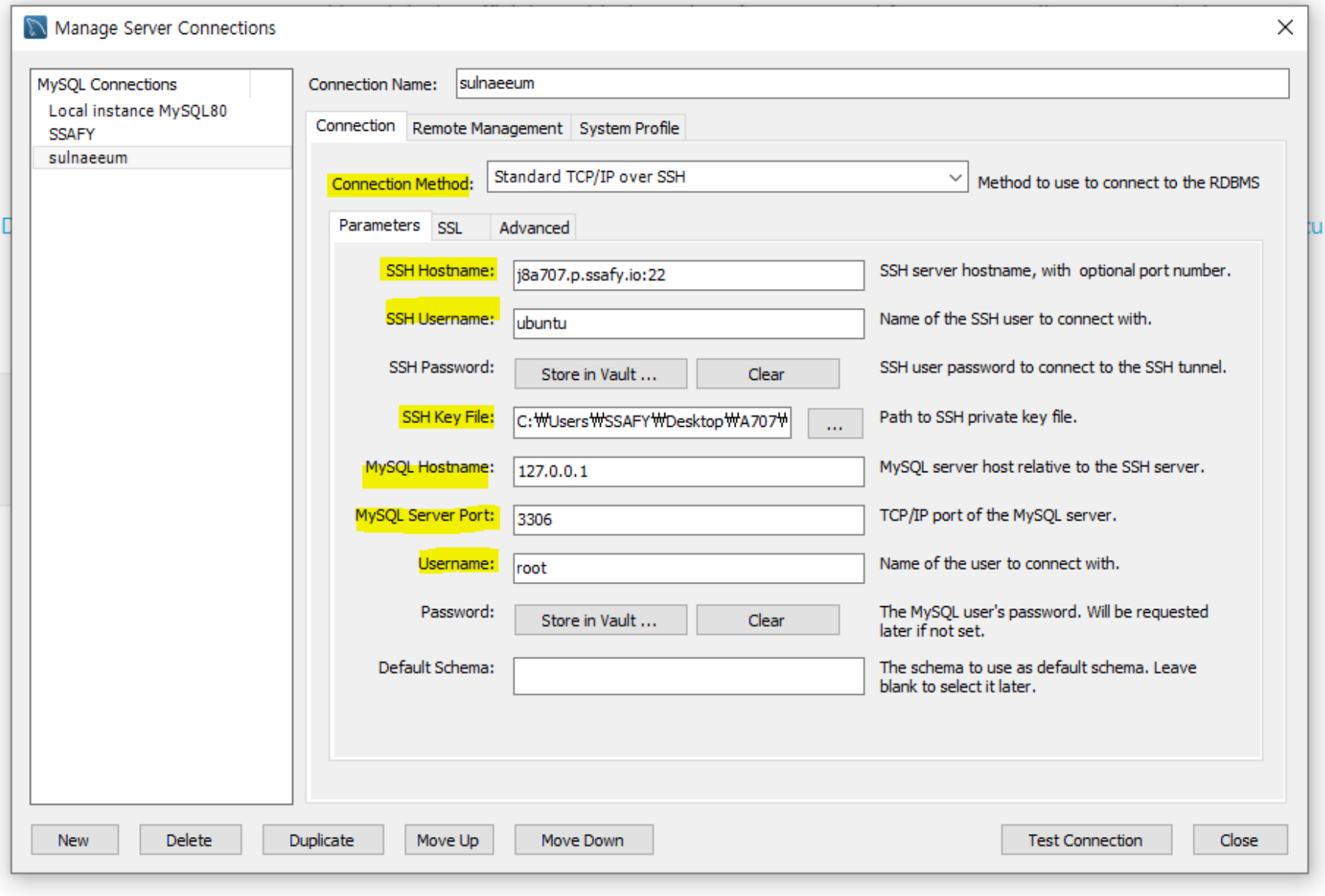
2) SDK를 위에서 설치했던 zulu-11로 설정 & Language level을 SDK default로 설정



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 = 01f7a5ec071370f48c515e47f8e79b5c
```

> .env

```
REACT_APP_KAKAO_KEY=c6f82fc3b98485b2394889a33664e793
```