포팅메뉴얼🛠️

팀 이름	SSAFY 7기 자율 프로젝트 A604

목차

1. 프로젝트 기술 스택

Front-end

- Unity 2021.3.9.f1
- VisualStudio 2019
- Plastic

Back-end

- JAVA 11
- Spring Boot 2.7.4
- Spring Security
- Hibernate
- JPA
- Gradle 7.5
- JWT 0.9.1
- MapStruct 1.5.2

Database

- Redis 7.0.4
- MySQL

Infra

- NginX
- AWS EC2 Ubuntu 20.04 LTS
- Docker 20.10.18
- Docker Compose 1.29.2
- Jenkins

2. MySQL 설정

스키마 생성

```
create database gardenary default charset utf8mb4;
```

MYSQL 계정 생성 (8.0.28 기준)

```
create user 'ssafy'@'localhost' identified by 'ssafy';
grant all privileges on gardenary.* to 'ssafy'@'localhost' with grant option;
```

---

3. Redis 설치

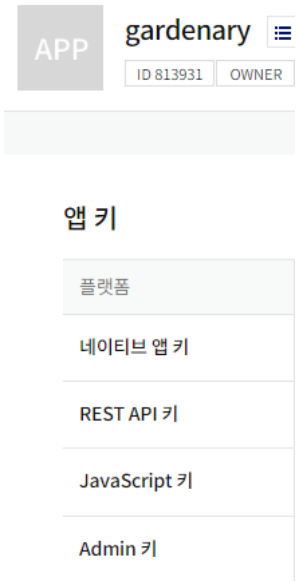
```
docker pull redis
docker run --name some-redis -d -p 6379:6379 redis
```

---

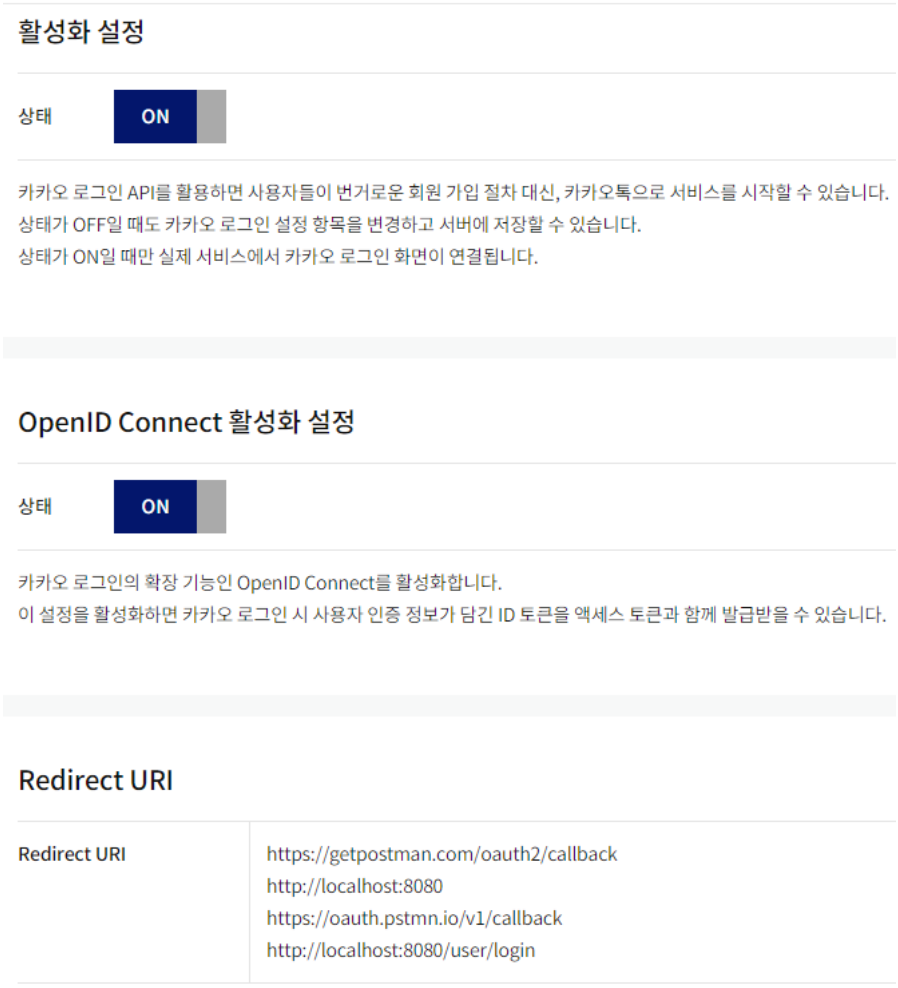
4. 외부 서비스 설정

카카오 소셜 로그인 설정

1. 애플리케이션을 등록 후 발급 받은 키 사용



2. 카카오 로그인 활성화, OpenID Connect 활성화, Redirect URI 설정



5. 배포

ssh 프로토콜로 서버에 접속 및 비밀번호 설정

```
sudo ssh -i keypair.pem ubuntu@k7a604.p.ssafy.io
sudo passwd root
sudo passwd ubuntu
```

도커 설치

1. 먼저 설치에 필요한 사전 업데이트 및 설치를 진행한다

```
sudo apt-get update
sudo apt install git
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

아래 명령어의 `arch=아키텍처` 에는 자신의 환경에 맞는 아키텍쳐를 적어줘야한다.

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

2. 도커 설치 및 실행

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

```
sudo systemctl enable docker && sudo service docker start
```

도커 컴포즈 설치

설치 및 권한 수정

- 도커 컴포즈는 여러개의 컨테이너의 실행과 관리를 할 수 있게 해준다.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/doc
sudo chmod +x /usr/local/bin/docker-compose
docker-compose -v
```

Jenkins 설치 및 설정

1. EC2 인스턴스에 접속
2. 젠킨스 도커 이미지 다운

```
sudo docker pull jenkins/jenkins:lts
```

3. 젠킨스 폴더 만들기

```
mkdir jenkins_build
cd jenkins_build
```

4. 젠킨스 컨테이너 실행 위한 Dockerfile 작성

```
cat > Dockerfile
```

```
FROM jenkins/jenkins:lts

USER root

# install docker
RUN apt-get update && \
    apt-get -y install apt-transport-https \
        ca-certificates \
        curl \
        gnupg2 \
        zip \
        unzip \
        software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
    apt-get update && \
    apt-get -y install docker-ce
```

5. 컨테이너 설정 위한 docker-compose 작성

```
cat > docker-compose.yml
```

```
version: '3.7'
services:
  jenkins:
    build:
      context: .
    container_name: jenkins
    user: root
    privileged: true
```

```
    ports:
      - 3333:8080
      - 50000:50000
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
```

6. docker compose up

```
sudo docker-compose up
```

7. ********** 사이에 있는 비밀번호 ctrl+c
8. 젠킨스 접속

```
http://k7a604.p.ssafy.io:3333
```

9. 저장했던 비밀번호 입력 > 왼쪽 박스(instal...) 클릭 > 설치 후, 계정 만들기

`계정명: 프로젝트이름`
`암호: 비밀번호 (임의)`
`암호확인: 비밀번호 (임의)`
`이름 : 프로젝트이름`
`이메일주소: 이메일주소`

10. 다음 화면에선 그냥 Save and Finish 누른다

젠킨스 설정

1. 플러그인 설치 왼쪽 메뉴의 `Jenkins 관리` > 플러그인 관리 > 설치 가능 > gitlab를 설치한다.
2. Gitlab 연동을 위한 Credentials 생성 jenkins 관리 > Manage Credentials > Stores scoped to Jenkins의 Domains (global) 클릭 > Add Credentials 클릭 > Username with password 클릭 후, 깃랩 이메일과 비밀번호를 넣어주고 생성한다.
3. Gradle, JDK 설정
   - Gradle: Jenkins 관리 > Global Tool Configuration > Gradle의 Add Gradle > 현 프로젝트에서는 Gradle7.5 사용 > Save
   - JDK: JDK11은 컨테이너에 따로 설치를 먼저 해줘야 함 (JDK9까지 있기때문) docker ps > jenkins 컨테이너 id확인 > docker exec -itu 0 컨테이너id /bin/sh > sudo apt-get update > apt-get install openjdk-11-jdk > java --version(버전확인) 젠킨스에서 name은 jdk11, java_home은 /usr/lib/jvm/java-11-openjdk-amd64 입력
4. 젠킨스 안에서 도커 컴포즈 설정을 위해 추가 작업을 해야 함
   - 젠킨스 컨테이너 접속

   ```
   docker exec -it 컨테이너ID /bin/sh
   ```

   - docker-compose 설치 및 권한 부여

   ```
   curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/dock
   chmod +x /usr/local/bin/docker-compose
   ```

백엔드 CI/CD

1. jenkins 왼쪽 메뉴의 '새로운 item' > 이름 입력후 Pipeline 클릭, Ok 클릭
2. Webhook 설정을 하기 위해 깃랩 저장소로 이동해서 Settings > Webhooks 클릭
3. 깃랩 해당 페이지의 URL 입력칸에 Jenkins의 Build Triggers에 나와있는 주소를 적는다.(
   (ex. http://k7a604.p.ssafy.io:3333/project/이름)
4. Build Triggers > Build when... 체크 > 고급 > Secret token의 Generate 클릭 후, 토큰 복사 > 깃랩 Webhooks 페이지의 Secret token에 붙여넣기
5. 깃랩 Webhooks의 Trigger의 Push events(체크)에 배포 브랜치 이름을 적는다. (ex. be/dev)
6. Add Webhook 클릭 > Test 진행 > 젠킨스의 Stage View를 보면 테스트 성공 여부가 보인다.

7. 젠킨스의 Pipeline > Definition의 pipeline script > 밑에 파이프라인 작성

```
pipeline {
    agent any
    tools {gradle "gradle7.5"}
    stages {
        stage('Prepare') {
            steps {
                echo 'Clonning Repository'
                git url: '깃랩주소',//깃랩주소
                    branch: 'backend/dev',//브런치이름
                    credentialsId: '크레덴셜아이디'//credentialsId
            }
            post {
                success {
                    echo 'Successfully Cloned Repository'
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
        stage('Copy Properties') {
            steps {
                echo 'Copy Properties'
                sh 'cp ./properties/application-db.yml garden-be/src/main/resources'//gitIgnore파일(properties)를 서버 실행할 때마다 자동으로
                sh 'cp ./properties/application-encrypt.yml garden-be/src/main/resources'
                sh 'cp ./properties/application-oauth.yml garden-be/src/main/resources'
                sh 'cp ./properties/application-social.yml garden-be/src/main/resources'
            }
            post {
                success {
                    echo 'Successfully Copied'
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
        stage('Build Gradle') {
            steps {
                echo 'Build Gradle'
                dir ('./garden-be') {//디렉토리 프로젝트에 맞춰서 변경
                    sh """
                        gradle clean build --exclude-task test
                    """
                }
            }
            post {
                success {
                    echo 'Successfully Builded'
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
        stage('Copy Jar') {
            steps {
                echo 'Copy Jar'
                sh 'rm back/backend-0.0.1-SNAPSHOT.jar || true'//디렉토리 프로젝트에 맞춰서 변경
                sh 'mv ./garden-be/build/libs/garden-0.0.1-SNAPSHOT.jar ./back'//디렉토리 프로젝트에 맞춰서 변경
            }
            post {
                success {
                    echo 'Successfully Copied'
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
        stage('Compose Down') {
            steps {
                echo 'Down Docker'
                sh 'docker-compose -f docker-compose-back.yml down -v'
                echo 'docker rmi start...'
                // sh 'docker stop -f $(docker ps -a -q -f name=nginx)'
                // sh 'docker rm -f $(docker ps -a -q -f name=nginx)'
                sh 'docker rmi -f com_backend'//실행 전 backend 도커 이미지 및 컨테이너 삭제
                sh 'docker rmi -f com_nginx'//실행 전 nginx 도커 이미지 및 컨테이너 삭제
            }
            post {
                success {
                    echo 'Successfully Build Down'
```

```
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
        stage('Compose Up') {
            steps {
                echo 'Push Docker'
                sh 'docker-compose -f docker-compose-back.yml up -d'
            }
            post {
                success {
                    echo 'Successfully Up'
                }
                failure {
                    error 'This pipeline stops here...'
                }
            }
        }
    }
}
```

- gitIgnore되는 properties.yml 파일들은 jenkins_home/workspace/Garden-Back에 properties폴더에 넣어놨음
- jenkins_home/workspace/Garden-Back에 docker-compose-back.yml을 작성해놨음 (mysql, redis, nginx(certbot)관련 설정을 함)

```
version: '3.7'
services:
  db:
    image: mysql:8.0.28
    expose:
      - "MYSQL포트"
    container_name: db
    volumes:
      - /home/ubuntu/deploy/db/conf.d:/etc/mysql/conf.d
      - /home/ubuntu/deploy/db/db/data:/var/lib/mysql
      - /home/ubuntu/deploy/db/db/initdb.d:/docker-entrypoint-initdb.d
    environment:
      MYSQL_DATABASE: 프로젝트DB이름
      MYSQL_ROOT_PASSWORD: "비밀번호"
  nginx:
    container_name: nginx
    build:
      dockerfile: Dockerfile
      context: ./nginx
    image: com_nginx
    expose:
      - "80"
      - "443"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/Garden-Back/back:/back
      - /home/ubuntu/deploy/certbot:/certbot
  backend:
    container_name: backend
    restart: on-failure
    build:
      dockerfile: Dockerfile
      context: ./back
    image: com_backend
    expose:
      - "8080"
    #ports:
    #  - "8080:8080"
    environment:
      SERVER_PORT: 8080
      SPRING_DATASOURCE_URL: jdbc:mysql://db:MYSQL포트/프로젝트DB이름?serverTimezone=Asia/Seoul&useLegacyDatetimeCode=false&useUnicode=tru
      SPRING_DATASOURCE_USERNAME: 유저이름
      SPRING_DATASOURCE_PASSWORD: "비밀번호"
      SPRING_REDIS_HOST: redis_boot
      SPRING_REDIS_PORT: 레디스포트
    depends_on:
      - db
      - redis_boot
  redis_boot:
    image: redis:alpine
    command: redis-server --port 레디스포트
    container_name: redis_boot
    hostname: redis_boot
```

```
    labels:
      - "name=redis"
      - "mode=standalone"
    expose:
      - "레디스포트"
    volumes:
      - /home/ubuntu/deploy/redis:/data
networks:
  default:
    name: com_net
    external: true
```

- Garden-Back에 nginx 폴더에 Dockerfile, default.conf 파일을 만듦

```
FROM nginx:stable-alpine

COPY ./default.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
upstream backend {
    server backend:8080;
}

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

    server_name juso.p.ssafy.io;

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

    location /.well-known/acme-challenge/ {
        root /certbot;
    }
}

server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    server_name  juso.p.ssafy.io;
    access_log   /var/log/nginx/nginx.vhost.access.log;
    error_log    /var/log/nginx/nginx.vhost.error.log;

    ssl                  on;
    ssl_certificate      /certbot/etc/live/juso.p.ssafy.io/fullchain.pem;
    ssl_certificate_key  /certbot/etc/live/juso.p.ssafy.io/privkey.pem;

    location /api {
        proxy_pass http://backend;
        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;
    }

}
```

- Garden-Back에 Dockerfile을 만들어놨음

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

CertBot 이용 SSL 인증서

1. mkdir makecertbot 생성 (단순히 certbot 생성 작업할 디렉토리)
2. 해당 디렉토리에서 cat > docker-compose.yml

```
version: "3.3"
services:
  nginx:
    image: nginx:latest
    volumes:
        - ./nginx/conf.d:/etc/nginx/conf.d
        - ./nginx/log:/var/log/nginx
        - ./www:/var/www/html
    ports:
        - 80:80
  certbot:
    restart: "no"
    depends_on:
      - nginx
    image: certbot/certbot
    container_name: certbot
    volumes:
      - ./certbot/etc:/etc/letsencrypt
      - ./certbot/var:/var/lib/letsencrypt
      - ./www:/var/www/html
    command: certonly --webroot --webroot-path=/var/www/html --email 이메일주소 --agree-tos --no-eff-email --force-renewal -d
juso.p.ssafy.io
```

3. 도커 컴포즈

```
docker-compose up
```

- Successfully received certificate가 나오면 정상적으로 설치가 된 것
- 실패하면 1시간 후 혹은 1주일 후에 생성할 수 있음 (동일 주소 요청으로 주 50회 제한. 본 프로젝트에서는 첫 시도부터 안돼서 1주일 후에 발급받음. 기다릴 시간이 없으면 다른 발급 기관에서 발급받기를 추천)

4. docker-compose-back.yml 파일의 내용에 맞춰 생성된 certbot 폴더를 옮겨주기

---

6. 기타 설정 파일

application.yml

```
server:
  port: 8080
  servlet:
    context-path: /api
    encoding:
      charset: UTF-8
      enabled: true
      force: true
spring:
  profiles:
    include: db, encrypt, oauth
  redis:
    host: localhost
    port: 6379
logging:
  file:
    name: logs/app.log
  pattern:
    console: "[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %clr([%M]){cyan}  %clr(%-5level) %clr(%logger{36}){magenta} - %msg%n"
    file: "[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] [%M] %-5level %logger{36} - %msg%n"
  logback:
    rollingpolicy:
      file-name-pattern: "logs/app.%d{yyyy-MM-dd}.%i.log"
      max-file-size: 10MB
  level:
    root: WARN
    org.hibernate.type.descriptor.sql: trace
    com.gardenary: debug
response:
  success: success
  fail: fail
const:
  tree-size: 15
  flower-size: 59
  question-size: 100
  exp-levelup: 100
  exp-tree: 25
  exp-flower: 50
  nickname-size: 55
  content-size: 1000
  item-size: 111
```

application-db.yml

spring.datasource.username, spring.datasource.password 설정 필요

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/gardenary?serverTimezone=Asia/Seoul&useLegacyDatetimeCode=false&useUnicode=true&characterEncoding
    username:
    password:
  jpa:
    show-sql: true
    hibernate:
      naming:
        physical-strategy: org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
        implicit-strategy: org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
      ddl-auto: update
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    generate-ddl: true
    properties:
      hibernate:
        show_sql: true
        format_sql: true
      jdbc:
        time_zone: UTC
  main:
    allow-bean-definition-overriding: true
```

application-encrypt.yml

spring.encrypt.encrypt-key, spring.encrypt.encrypt-iv 설정 필요

```
spring:
  encrypt:
    encrypt-key:
    encrypt-iv:
```

application-auth.tml

jwt.secret-key, jwt.access-token-expire-time, jwt.refresh-token-expire-time 설정 필요

```
jwt:
  secret-key:
  access-token-expire-time:
  refresh-token-expire-time:
```

7. 최종 EC2 폴더 구조

```
- /home/ubuntu
  └ /deploy/certbot/... (CertBot 키)
  └ /jenkins_build
    └ docker-compose.yml (jenkins)
      └ /jenkins_home/workspace/Garden-Back
        └ /garden-be ( 깃랩 클론 파일)
        └ docker-compose-back.yml (backend)
        └ /back
          └ Dockerfile
          └ garden-0.0.1-SNAPSHOT.jar
        └ /nginx
          └ default.conf
          └ Dockerfile
        └ /properties
          └ application-db.yml, application-encrypt.yml, application-oauth.yml
```