쓸#

금융 통합 네트워크 플랫폼 서비스

빌드 및 배포

프로젝트 기간 : 2022.10.11 ~ 11.21

삼성SW청년아카데미 서울캠퍼스 7기

민경욱 이주영 김찬영 정건우 이지나 채윤선

1. 기술스택

구분	기술스택	상세내용	버전
공통	형상관리	Gitlab	-
	이슈관리	Jira	-
	커뮤니케이션	Mattermost,Notion	-
BackEnd	DB	MySQL	5.7
		JPA	-
	Kotlin	-	1.3.61
	Spring	Spring	5.3.6
		SpringBoot	2.4.5
		Spring Security	-
	python	-	3.7
	Data	selenium	4.4.3
		mysql-connector-python	8.0.31
		numpy	1.21.6
		pandas	1.3.5
		pandas-datareader	0.10.0
		SQLAlchemy	1.4.42
	IDE	Pycharm pro	2022.1.4
	kafka	zookeeper	3.7
		kafka	3.1.0
	IDE	InteliJ	2022.2.1
	클라우드스토리지	AWS S3	-
	Build	Gradle	7.3.2

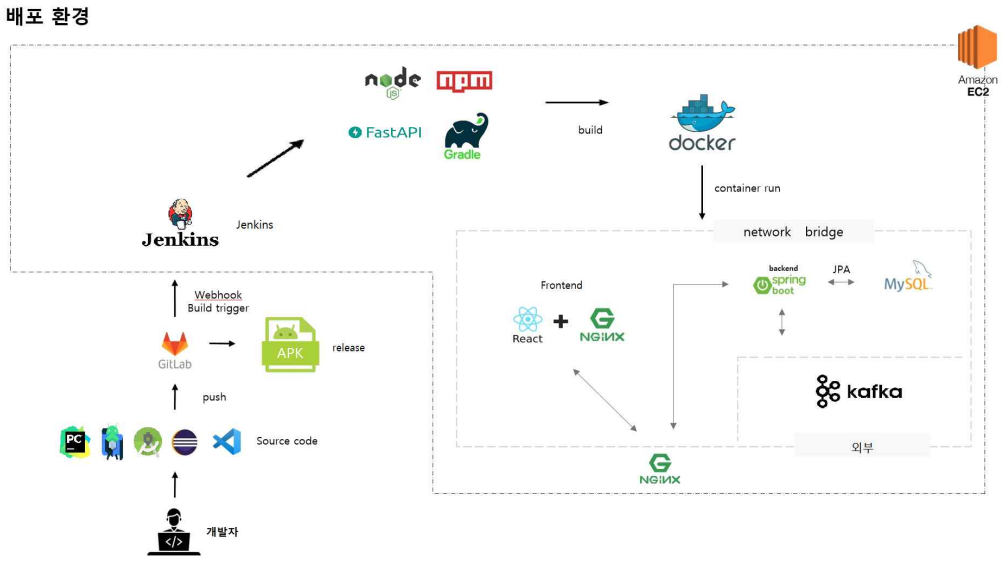
	APIDocs	Swagger2	3.0.0
FrontEnd	React	React	8.0.2
	mui-material		5.9.0
	axios		0.14.0
	IDE	Visual Studio Code	1.63.2
Android	Android compose	compose	1.2.1
		hilt	2.44
		retrofit2	2.9.0
		lottie-compose	5.2.
	IDE	Android Studio	Dolphin | 2021.3.1
Server	서버	AWSEC2	-
	플랫폼	Ubuntu	20.04.3 LTS
	배포	Docker	20.10.17
		Jenkins	-

2. 상세내용

□ 개요

아래 그림은 **쏠#** 서비스의 배포 환경 및 CI/CD 배포 흐름도입니다. 팀원들이 각자 작성한 프로젝트를 GitLab에 push 하면, Jenkins가 자동으로 FrontEnd, BackEnd, FastApi를 빌드하게 됩니다. 각 프로젝트를 빌드 한 후에는 Docker 이미지를 만들고 이를 Docker Hub 에 push 한 후, 이로부터 서비스에 필요한 이미지를 받아와 컨테이너로 띄웁니다.

<CI/CD 배포환경>



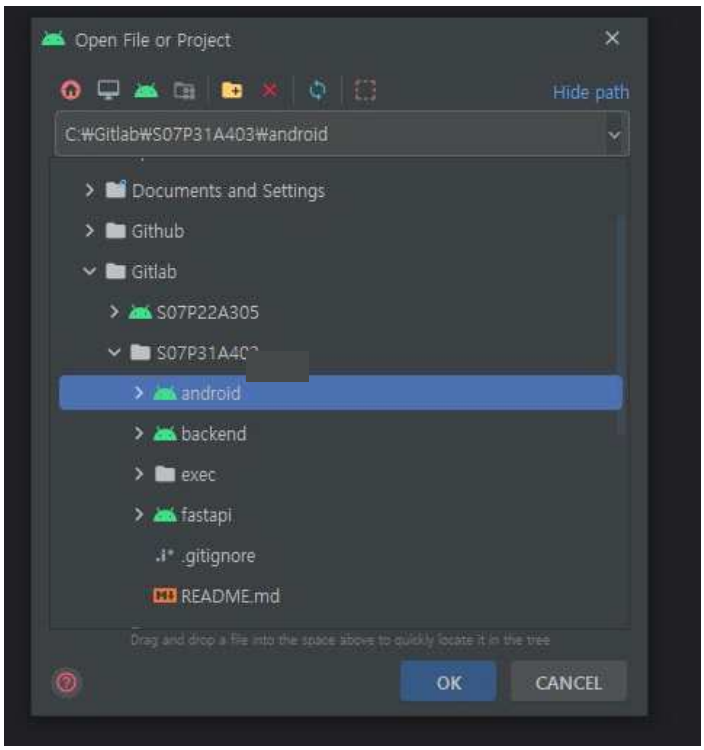
서버의 경우 SSAFY에서 지원받은 AWS EC2 싱글 인스턴스로 인프라를 구축하였습니다. 이때, 추후 서비스화를 위해 Nginx는 리버스 프록시 서버로 설정하였습니다. 또한 8080포트를 Backend 서버로 설정, 8082포트를 Data 서버로 설정하여 Load Balancing이 가능하도록 구축하였습니다.

어플리케이션의 경우, 개발자가 각자 GitLab에 push 하면 배포 담당자가 이를 릴리즈 버전으로 apk화 하여 배포합니다.

□ Android

◦ 사용할 IDE에서 android 폴더를 open 합니다.

<import Project – Android Studio>



□ FrontEnd

◦ Docker 이미지 생성을 위한 Dockerfile (해당 파일은 프로젝트 내에 이미 작성되어 있습니다)

```
# Dockerfile
FROM node:16.16.0 as build-stage
WORKDIR /var/jenkins_home/workspace/BuildFrontend2/frontend2
COPY package*.json ./
RUN npm i -y
COPY . .
EXPOSE 3000
CMD [ "npm", "start"]
```

◦ Jenkins에서 docker 이미지 생성을 위한 명령어

```
cd front
docker build -t react .

docker save react > /var/jenkins_home/images_tar/react.tar

ls /var/jenkins_home/images_tar
```

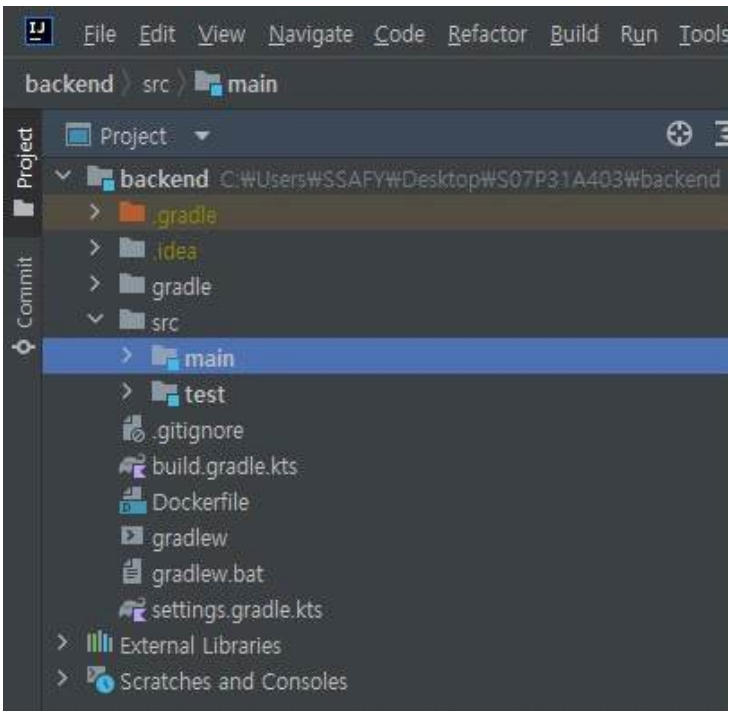
◦ Jenkins에서 이미지 컨테이너 실행

```
sudo docker load < /jenkins/images_tar/react.tar

if (sudo docker ps | grep "react") then sudo docker stop react; fi

sudo docker run -it -d --rm -p 3000:3000 --name react react
echo "Run frontend"
```

□ BackEnd

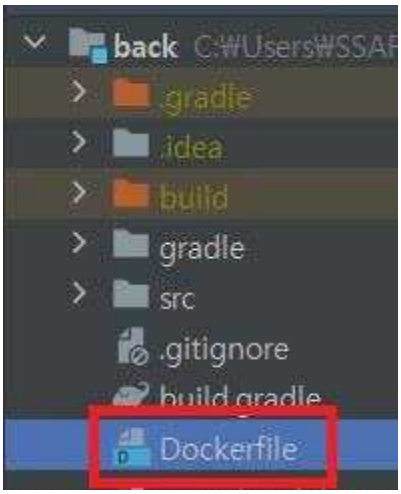
◦ 사용할 IDE에서 backend 폴더를 Gradle로 import 합니다.

<import Gradle Project – InteliJ>



◦ Docker 이미지 생성을 위한 Dockerfile 작성 (해당 내용은 프로젝트 내에 이미 작성 되어 있습니다)

```
FROM openjdk:8-jdk-alpine
EXPOSE 8080
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} spring.jar
ENTRYPOINT ["java","-jar","spring.jar"]
```



◦ Gradle 빌드

```
gradlew clean build
```

◦ Jenkins 에서 docker 이미지 생성

```
cd back
docker build -t spring .
docker save spring > /var/jenkins_home/images_tar/spring.jar
ls /var/jenkins_home/images_tar
```

◦ mysql 컨테이너 실행

```
docker run -p 3306:3306 –e MYSQL_ROOT_PASSWORD=BigDataDDP77!
-e TZ=Asia/eoul –d mysql:latest
```

◦ 이미지 컨테이너 실행

```
sudo docker load < /jenkins/images_tar/spring.jar

if (sudo docker ps | grep "spring"); then sudo docker stop spring;
fi

sudo docker run -it -d --rm -p 8080:8080  --name spring spring
echo "Run backend"
```

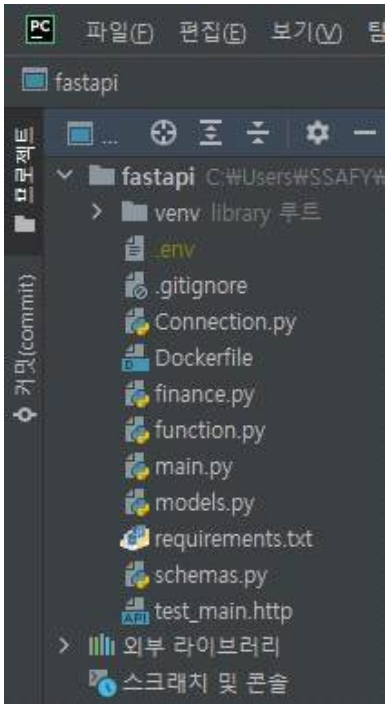
◦ 서버 컨테이너 로그 확인

```
docker logs <containerID>
```

□ Fast API

◦ 사용할 IDE에서 fast api 폴더를 import 합니다.

<impor Project – Pycharm>



◦ Docker 이미지 생성을 위한 Dockerfile 작성 (해당 내용은 프로젝트 내에 이미 작성 되어 있습니다)

```
FROM python:3.7
WORKDIR /var/jenkins_home/workspace/data/fastapi
COPY requirements.txt ./
RUN pip install --no-cache-dir --upgrade -r /var/jenkins_home/workspace/data/fastapi/requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8082"]
```

◦ 가상환경 설치

```
python -m venv venv
source venv/Scripts/activate
pip install –r requirements.txt
```

◦ Jenkins 에서 docker 이미지 생성

```
cd back
docker build -t data .
docker save data > /var/jenkins_home/images_tar/data.jar
ls /var/jenkins_home/images_tar
```

◦ 이미지 컨테이너 실행

```
sudo docker load < /jenkins/images_tar/data.jar

if (sudo docker ps | grep "data"); then sudo docker stop data; fi

sudo docker run -it -d --rm -p 8080:8080  --name data data
echo "Run backend"
```

◦ 서버 컨테이너 로그 확인

```
docker logs <containerID>
```

3. 특이사항

□ 개요

쏠# 서비스는 Docker 이미지 컨테이너를 기반으로 서비스를 배포하고 있습니다. 서비스에 문제가 발생 시, 아래 명령어를 확인하여 상태를 확인 할 수 있습니다. BackEnd, FrontEnd, Mysql 프로젝트의 상태를 확인하기 위해선 각 컨테이너의 로그를 확인하는 명령어를 사용하여 log 확인이 가능합니다.

□ Nginx

◦ 상태 확인

```
sudo service nginx status
```

◦ 재실행

```
sudo service nginx restart
```

□ Docker

◦ 도커 compose 파일

```
version: '3'

networks:
  kafka-net:
    driver: bridge

services:
  jenkins:
      image: jenkins/jenkins:lts
      container_name: jenkins
      volumes:
          - /var/run/docker.sock:/var/run/docker.sock
```

```
            - /jenkins:/var/jenkins_home
        ports:
            - "9090:8080"
        privileged: true
        user: root
        environment:
                - TZ=Asia/Seoul

  zookeeper:
    image: bitnami/zookeeper:3.7
    container_name: zookeeper
    networks:
      - kafka-net
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
  kafka:
    image: bitnami/kafka:3
    container_name: kafka
    networks:
      - kafka-net
    ports:
      - '9093:9093'
    environment:
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - ALLOW_PLAINTEXT_LISTENER=yes
      -
KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CLIENT:PLAINTEXT,
EXTERNAL:PLAINTEXT
      - KAFKA_CFG_LISTENERS=CLIENT://:9092,EXTERNAL://:9093
      -
KAFKA_CFG_ADVERTISED_LISTENERS=CLIENT://kafka:9092,EXTERNAL://
k7a403.p.ssafy.io:9093
```

```
    - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=CLIENT
  depends_on:
    - zookeeper
```

◦ 컨테이너 확인

```
sudo docker ps -a
```

◦ 서버 컨테이너 로그 확인

```
docker logs <containerID>
```

◦ 컨테이너 재실행

```
sudo docker restart <containerID>
```

◦ 컨테이너 삭제

```
sudo docker rm <containerID>
```

◦ 이미지 삭제

```
sudo docker rmi <imageID>
```

4. 프로퍼티 정의

□ MySQL

◦ MySQL Docker 컨테이너에서 DB 스키마를 생성해두면 SpringBoot 구동 시 자동으로 Table 생성됩니다.

◦ Spring application.properties DB 관련 설정

```
spring.jpa.hibernate.naming.implicit-strategy=org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
spring.data.web.pageable.one-indexed-parameters=true
spring.datasource.url=jdbc:mysql://[web-address]:3306/finance_db?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDateTimeBehavior=convertToNull&rewriteBatchedStatements=true
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.hikari.username=root
spring.datasource.hikari.password=wkdbf77!
```

◦ 계정 생성

```
create user 'root'@'%' identified by 'wkdbf77!';
grant all privileges on *.* to 'root'@'%' with grant option; flush privileges;
```

◦ 스키마 생성

```
create database if not exists finance_db collate utf8mb4_general_ci;
```

□ Nginx

◦ 환경설정 – etc/nginx/nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log  /var/log/nginx/error.log warn;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 1024;
}

http {
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        upstream spring {
                server k7a403.p.ssafy.io:8080;
        }
        upstream data_server {
                server k7a403.p.ssafy.io:8082;
        }
```

```
        upstream home {
                server 172.17.0.1:3000;
        }
        upstream react {
                ip_hash;
                server 172.17.0.1:3001;
        }


    server {
        #listen 80;
        server_name k7a403.p.ssafy.io;
        underscores_in_headers on;

                # /api 경로로 오는 요청을 백엔드 upstream 의 /api 경로로 포워딩
        location /api {
            proxy_pass          http://spring/api;

            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
            proxy_pass_request_headers on;

        }

        # /data 경로로 오는 요청을 백엔드 upstream 의 /data 경로로 포워딩
        location /data {
            proxy_pass          http://data_server/data;

            proxy_http_version 1.1;
```

```
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;

        }

        location / {
            proxy_pass          http://172.17.0.1:3000/;

            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        proxy_connect_timeout 300s;
        proxy_read_timeout 600s;
        proxy_send_timeout 600s;
            proxy_buffer_size           128k;
    proxy_buffers                   4 256k;
    proxy_busy_buffers_size     256k;

        }

                # / 경로로 오는 요청을 프론트엔드 upstream 의 /
경로로 포워딩
        #location / {
        #       proxy_pass              http://home/;
        #       proxy_http_version 1.1;
        #       proxy_set_header Upgrade $http_upgrade;
        #       proxy_set_header Connection 'upgrade';
        #       proxy_set_header Host $host;
        #       proxy_cache_bypass $http_upgrade;
        #}
```

```
            #root /jenkins/workspace/BuildFrontend2/frontend2/public;
            #index index.html index.htm;

            #location / {
            #           try_files $uri $uri/ /index.html;
            #}


        listen 443 ssl; # managed by Certbot
        ssl_certificate /etc/letsencrypt/live/k7a403.p.ssafy.io/fullchain.pem;
# managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/k7a403.p.ssafy.io/privkey.pem; # managed by
Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf; # managed by
Certbot
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by
Certbot

}


            ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping
SSLv3, ref: POODLE
            ssl_prefer_server_ciphers on;

            access_log /var/log/nginx/access.log;
            error_log /var/log/nginx/error.log;

            gzip on;

            include /etc/nginx/conf.d/*.conf;
            include /etc/nginx/sites-enabled/*;
```

```
    server {
    if ($host = k7a403.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        server_name k7a403.p.ssafy.io;
    listen 80;
    return 404; # managed by Certbot


}}
```