# 가물가물

## 빌드 및 배포

---

프로젝트 기간 : 2022.08.29 ~ 10.07


삼성SW청년아카데미 서울캠퍼스 7기

민경욱 이주영 김찬영 정건우 이지나 채윤선

# 1. 기술스택

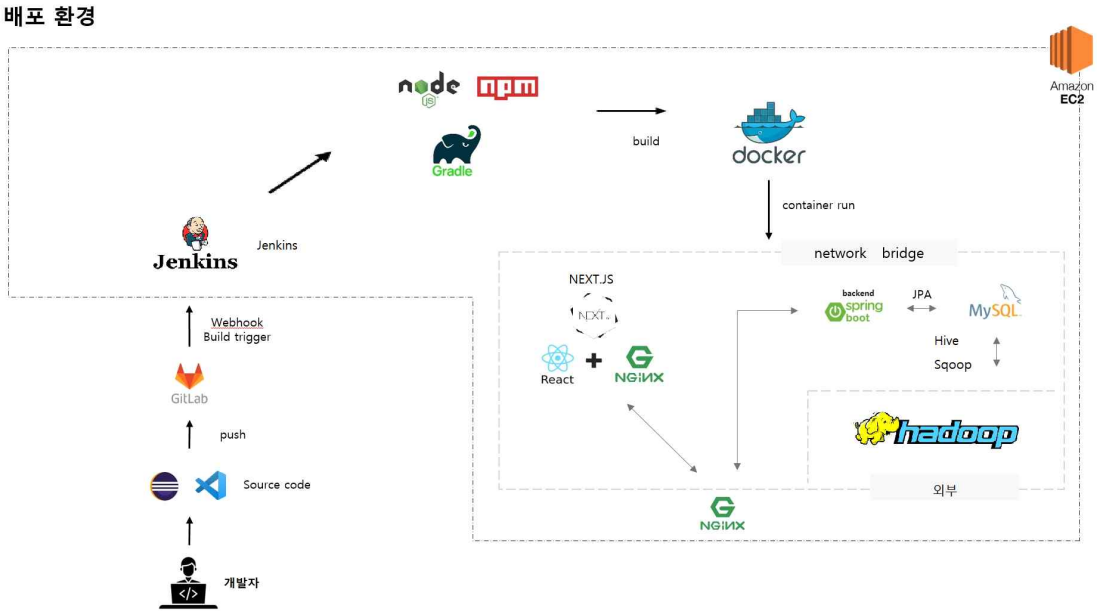
| 구분 | 기술스택 | 상세내용 | 버전 |
|---|---|---|---|
| 공통 | 형상관리 | Gitlab | - |
| | 이슈관리 | Jira | - |
| | 커뮤니케이션 | Mattermost, Notion | - |
| BackEnd | DB | MySQL | 5.7 |
| | | JPA | - |
| | | Hadoop | 3.3.1 |
| | | Hive | 3.1.2 |
| | | Sqoop | 1.4.7 |
| | Java | Zulu | 8.33.0.1 |
| | Spring | Spring | 5.3.6 |
| | | Spring Boot | 2.4.5 |
| | | Spring Security | - |
| | IDE | InteliJ | 2022.2.1 |
| | 클라우드 스토리지 | AWS S3 | - |
| | Build | Gradle | 7.3.2 |
| | API Docs | Swagger2 | 3.0.0 |
| FrontEnd | React | React | 8.0.2 |
| | Next | next.js | 12.2.5 |
| | | next-pwa | 5.5.2 |
| | mui-material | | 5.9.0 |
| | axios | | 0.14.0 |
| | apex-charts | | 3.35.5 |
| | IDE | Visual Studio Code | 1.63.2 |
| Server | 서버 | AWS EC2 | - |
| | 플랫폼 | Ubuntu | 20.04.3 LTS |
| | 배포 | Docker | 20.10.17 |
| | | Jenkins | - |

# 2. 상세내용

## □ 개요

아래 그림은 **가물가물** 서비스의 배포 환경 및 CI/CD 배포 흐름도입니다. 팀원들이 각자 작성한 프로젝트를 GitLab에 push 하면, Jenkins가 자동으로 FrontEnd, BackEnd를 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. 를 이용하여 8080포트를 Backend 서버로 설정하여 Load Balancing이 가능하도록 구축하였습니다.

## □ FrontEnd

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

```
# Dockerfile
FROM node:16.15.0 as build-stage
WORKDIR /var/jenkins_home/workspace/frontend/front
COPY package*.json ./
RUN npm i -y
COPY . .
# COPY /deploy_conf/nginx.conf /etc/nginx/conf.d/default.conf
RUN npm run build
CMD [ "npm", "run", "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
```

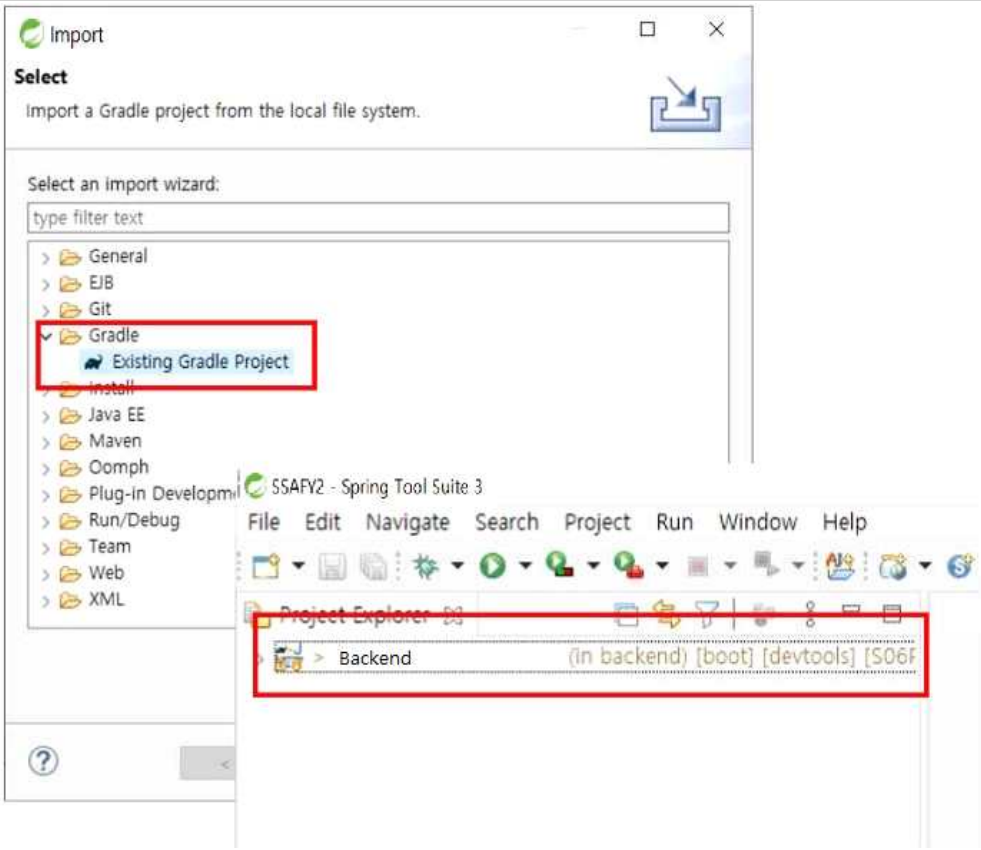
◦ Jenkins에서 이미지 컨테이너 실행

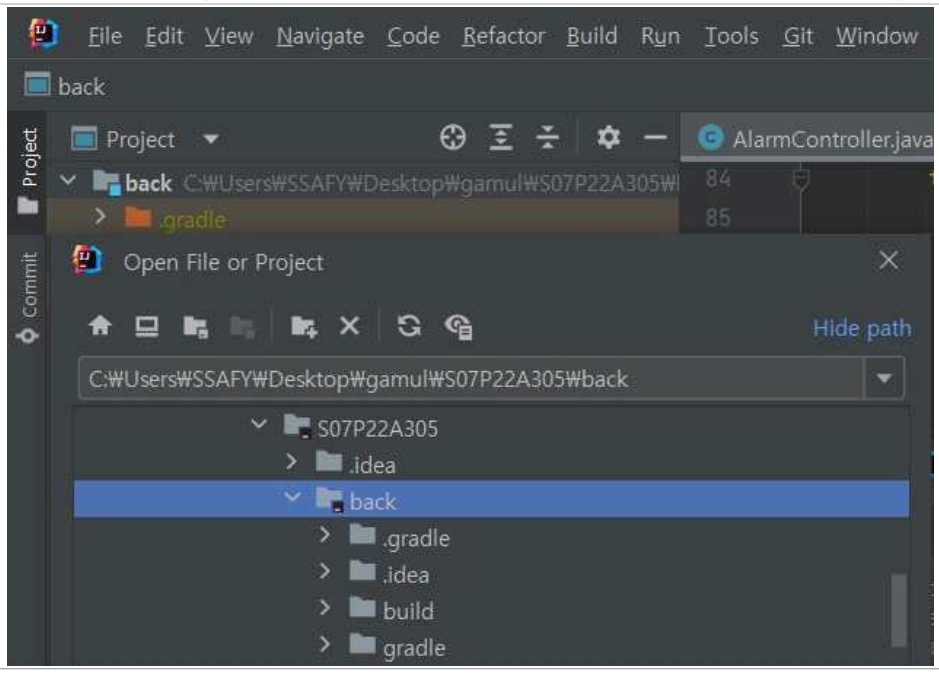
```
sudo docker load < /jenkins/images_tar/react.tar

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 – 위 STS, 아래 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>
```

# 3. 특이사항

## □ 개요

**가물가물** 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

◦ 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/gamul_db?useUnicode=true&cha**racter Encoding**=utf8&serverTimezone=Asia/Seoul&zeroDateTimeBehavior=convertToNull&re**writeB atchedStatements**=true
**spring.datasource.driver-class-name**=com.mysql.cj.jdbc.Driver
**spring.datasource.hikari.username**=root
**spring.datasource.hikari.password**=BigDataDDP77!

◦ 계정 생성

```
create user 'root'@'%' identified by 'BigDataDDP77!';
grant all privileges on *.* to 'root'@'%' with grant option; flush privileges;
```

◦ 스키마 생성

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

## □ Nginx

◦ 환경설정 - etc/nginx/sites-available/test.conf

```
user www-data;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;
events {
    worker_connections 1024;
}
http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
        # 백엔드 upstream 설정
    upstream spring {
        server j7a305.p.ssafy.io:8080;
    }

        # 프론트엔드 upstream 설정
    upstream react {
        server 172.17.0.1:3000;
    }

    server {
        #listen 80;
        server_name j7a305.p.ssafy.io;

                # /api 경로로 오는 요청을 백엔드 upstream 의
/api 경로로 포워딩
        location /api/v1 {
```

```
            proxy_pass              http://spring/api/v1;

            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;

        }

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

            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;

        }

                # / 경로로 오는 요청을 프론트엔드 upstream 의 / 경로로 포워딩
        location / {
            proxy_pass              http://react/;
            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;
        }
```

```
        listen 443 ssl; # managed by Certbot
        ssl_certificate /etc/letsencrypt/live/j7a305.p.ssafy.io/fullchain.pem;
# managed by Certbot
      ssl_certificate_key
 /etc/letsencrypt/live/j7a305.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

}

        log_format main '$remote_addr - $remote_user [$time_local]
"$request" '
                        '$status $body_bytes_sent "$http_referer" '
                        '"$http_user_agent" "$http_x_forwarded_for"';
        access_log /var/log/nginx/access.log main;

        sendfile on;
        keepalive_timeout 65;
        # include /etc/nginx/conf.d/*.conf;



        server {
        if ($host = j7a305.p.ssafy.io) {
            return 301 https://$host$request_uri;
        } # managed by Certbot



            server_name j7a305.p.ssafy.io;
        listen 80;
        return 404; # managed by Certbot
```

```
}}
```