

# 포팅 매뉴얼

≡ 날짜	10월 5일 7시
------	-----------

## 1. 개발환경

- [1.1. Frontend](#)
- [1.2. Backend](#)
- [1.3. Server](#)
- [1.4. Database](#)
- [1.5. UI/UX](#)
- [1.6. IDE](#)
- [1.7. 형상 / 이슈관리](#)
- [1.8. 기타 툴](#)

## 2. EC2 세팅

- [EC2 접속](#)
- [Docker Engine 설치](#)
- [Docker-Compose 설치](#)
- [환경 설정](#)
- [nginx.conf 파일 생성](#)
- [docker-compose 파일 작성](#)
- [docker-compose 실행](#)

## 3. SSL 인증서 발급 및 적용

- [init-letsencrypt 생성](#)
- [init-letsencrypt 실행](#)
- [nginx 재실행](#)

## 4. CI/CD 구축

- [Jenkins 환경설정](#)
  - [Credentials 설정](#)
  - [Tools 설정](#)
  - [System 설정](#)
- [Jenkins Pipeline 생성](#)
  - [Build Triggers 설정](#)
  - [Pipeline 설정](#)
  - [GitLab Webhook 생성](#)
- [Jenkinsfile, Dockerfile 작성](#)
  - [backend Jenkinsfile](#)
  - [backend Dockerfile](#)

[frontend Jenkinsfile](#)  
[frontend Dockerfile](#)  
[frontend react-nginx.conf](#)

## 1. 개발환경

### 1.1. Frontend

- React 18.2

### 1.2. Backend

- Java
  - Java OpenJDK 11.0.20
  - Spring Boot 2.7.15
    - Spring Data JPA 2.7.6
    - Spring Security 5.7.6
    - JUnit 4.13.2
    - Lombok 1.18.24
    - Swagger 3.0.0
  - Gradle 8.2.1

### 1.3. Server

- Ubuntu 20.04 LTS
- Docker 24.0.5
- Docker Compose version v2.20.2
- Nginx 1.18.0-0ubuntu1.4

### 1.4. Database

- MySQL8.0.33
- Redis
- H2 1.4.200

## 1.5. UI/UX

- Figma

## 1.6. IDE

- Visual Studio Code
- IntelliJ IDEA
- Dbeaver

## 1.7. 형상 / 이슈관리

- Gitlab
- Jira

## 1.8. 기타 툴

- Posrman
- Figma
- Notion

# 2. EC2 세팅

## EC2 접속

- I9A507T.pem 키가 있는 디렉토리에서 접속 명령어
- `ssh -i J9A607T.pem ubuntu@j9a607.p.ssafy.io`

## Docker Engine 설치

- 참고 : 도커 공식문서 <https://docs.docker.com/engine/install/ubuntu/>

```
# 도커 설치 되었는지 확인
docker version

# 구 버전 삭제 (Unable to locate package docker-engine)
sudo apt-get remove docker docker-engine docker.io containerd runc

# apt 패키지 업데이트
sudo apt-get update

# 도커 Apt 패키지 셋업
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
# Add the repository to Apt sources:
echo \
"deb [arch=$(dpkg --print-architecture)] signed-by=/etc/apt/keyrings/docker.gpg http
s://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME)" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Docker Engine 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-com
pose-plugin
```

## Docker-Compose 설치

- 참고 : 도커 공식 문서 <https://docs.docker.com/compose/install/linux/>

```
# apt 패키지 업데이트 후 최신버전 다운 (Ubuntu)
sudo apt-get update

# 이거 아닌 거 같다
sudo apt-get install docker-compose-plugin

# manually install (이건 거 같다)
DOCKER_CONFIG=${DOCKER_CONFIG:-$HOME/.docker}
mkdir -p $DOCKER_CONFIG/cli-plugins
curl -SL https://github.com/docker/compose/releases/download/v2.20.3/docker-compose-linux-
x86_64 -o $DOCKER_CONFIG/cli-plugins/docker-compose
```

```
# 버전확인
docker compose version
```

## 환경 설정

- 사용자 설정

```
# ssafy 계정 생성
sudo adduser ssafy

# docker 그룹에 ssafy 추가
sudo usermod -aG docker ssafy

# ssafy 계정으로 전환
su - ssafy
```

## nginx.conf 파일 생성

- home/ssafy/readed/proxy에 nginx.conf 생성
- ssl 인증 전에는 인증서 파일 경로, 개인키 파일 경로 주석처리

```
# nginx가 리버시 프록시 역할을 하도록 nginx 파일 설정
user nginx;
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;

    # upstream 설정은 docker-compose에서 설정한 서비스명 사용
    # docker-compose.yml에서 올라가는 컨테이너명으로 작성
    # 백엔드 upstream 설정
    upstream server {
        # http:// 붙이면 안 됨
        # 8081 포트를 열어줘야 됨
        server 3.34.135.50:8081;
        # 접속시 커넥션 유지 시간을 지정
        keepalive 1024;
    }
}
```

```

# 프론트엔드 upstream 설정
upstream client {
    server 3.34.135.50:3000;
}

server {
    # nginx를 통해 외부로 노출되는 port
    # http 80으로 진입해도 https 443로 리다이렉트
    listen 80;

    # 지정한 서버인증서에 포함된 도메인
    server_name haruman.site;
    server_tokens off;

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

    # certbot 설정파일
    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }
}

server {
    # default_server 필요 없음
    listen 443 ssl;

    # 인증서 파일 경로
    ssl_certificate /etc/letsencrypt/live/haruman.site/fullchain.pem;
    # 개인키 파일 경로
    ssl_certificate_key /etc/letsencrypt/live/haruman.site/privkey.pem;

    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # 블루 그린 포트 리다이렉팅을 위한 conf 파일
    # ./proxy/service-url.inc:/etc/nginx/conf.d/service-url.inc로 볼륨 마운트를 해줌
    include /etc/nginx/conf.d/service-url.inc;

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

    # / 경로로 오는 요청을 프론트엔드 upstream 의 / 경로로 포워딩
    location / {
        proxy_pass http://client;
        # 시간 넉넉하게

```

```

        proxy_connect_timeout 300s;
        proxy_read_timeout 600s;
        proxy_send_timeout 600s;
        proxy_buffers 8 16k;
        proxy_buffer_size 32k;
    }
}

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;
}

```

## docker-compose 파일 작성

- 도커 이미지 빌드 및 컨테이너 실행 자동화 설정의 편리성을 위해 docker-compose 파일 생성
- home/ssafy/readed에 docker-compose.yml 생성
- jenkins, nginx, certbot, redis 설치

```

version: "3.0"
services:
  jenkins:
    image: jenkins/jenkins:lts
    user: root
    ports:
      - 8080:8080
    volumes:
      - /jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock

  nginx:
    image: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./proxy/nginx.conf:/etc/nginx/nginx.conf
      - ./proxy/service-url.inc:/etc/nginx/conf.d/service-url.inc
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot

```

```

    depends_on:
      - jenkins
    command: "/bin/sh -c 'while ;; do sleep 6h & wait $$(!); nginx -s reload; done & nginx  
-g \"daemon off;\""

certbot:
  image: certbot/certbot
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h & wait  
$$(!); done;'"
  depends_on:
    - nginx

redis:
  image: redis:latest
  ports:
    - 6379:6379
  volumes:
    - ./redis/data:/data
    - ./redis/conf/redis.conf:/usr/local/conf/redis.conf
  labels:
    - "name=redis"
    - "mode=standalone"
  restart: always
  command: redis-server /usr/local/conf/redis.conf
  depends_on:
    - jenkins

```

## docker-compose 실행

- docker-compose.yml 파일이 위치한 경로에서 실행

```

# -d 는 백그라운드 실행
docker-compose up -d

# --build를 추가하면 기존이 삭제되어서 안됨
docker-compose up --build

```

- docker-compose 파일이 실행되면 jenkins 컨테이너와 nginx 컨테이너가 up 되어야 함

```

# docker 컨테이너 확인
docker ps -a

```



```
# docker 컨테이너 로그 확인
docker logs [컨테이너]

# docker 컨테이너 삭제 (컨테이너 삭제 후 이미지 삭제)
docker rm -f $(docker ps -qa)

# docker 이미지 삭제
docker image rm -f $(docker image ls -q)
```

### 3. SSL 인증서 발급 및 적용

#### init-letsencrypt 생성

- /home/ssafy/readed 경로에 [init-letsencrypt.sh](#) 생성

```
#!/bin/bash

if ! [ -x "$(command -v docker-compose)" ]; then
    echo 'Error: docker-compose is not installed.' >&2
    exit 1
fi

domains=(haruman.site)
rsa_key_size=4096
data_path="./data/certbot"
email="davidpoplar@naver.com"
staging=0 # Set to 1 if you're testing your setup to avoid hitting request limits

if [ -d "$data_path" ]; then
    read -p "Existing data found for $domains. Continue and replace existing certificate? (y/N) " decision
    if [ "$decision" != "Y" ] && [ "$decision" != "y" ]; then
        exit
    fi
fi

if [ ! -e "$data_path/conf/options-ssl-nginx.conf" ] || [ ! -e "$data_path/conf/ssl-dhparams.pem" ]; then
    echo "### Downloading recommended TLS parameters ..."
    mkdir -p "$data_path/conf"
    curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot_n
    ginx/_internal/tls_configs/options-ssl-nginx.conf > "$data_path/conf/options-ssl-nginx.con
    f"
    curl -s https://raw.githubusercontent.com/certbot/certbot/master/certbot/certbot/ssl-dhp
    arams.pem > "$data_path/conf/ssl-dhparams.pem"
fi
```

```

    echo
fi

echo "### Creating dummy certificate for $domains ..."
path="/etc/letsencrypt/live/$domains"
mkdir -p "$data_path/conf/live/$domains"
docker-compose run --rm --entrypoint "\
    openssl req -x509 -nodes -newkey rsa:$rsa_key_size -days 1\
    -keyout '$path/privkey.pem' \
    -out '$path/fullchain.pem' \
    -subj '/CN=localhost'" certbot
echo

echo "### Starting nginx ..."
docker-compose up --force-recreate -d nginx
echo

echo "### Deleting dummy certificate for $domains ..."
docker-compose run --rm --entrypoint "\
    rm -Rf /etc/letsencrypt/live/$domains && \
    rm -Rf /etc/letsencrypt/archive/$domains && \
    rm -Rf /etc/letsencrypt/renewal/$domains.conf" certbot
echo

echo "### Requesting Let's Encrypt certificate for $domains ..."
#Join $domains to -d args
domain_args=""
for domain in "${domains[@]}"; do
    domain_args="$domain_args -d $domain"
done

# Select appropriate email arg
case "$email" in
    "") email_arg="--register-unsafely-without-email" ;;
    *) email_arg="--email $email" ;;
esac

# Enable staging mode if needed
if [ $staging != "0" ]; then staging_arg="--staging"; fi

docker-compose run --rm --entrypoint "\
    certbot certonly --webroot -w /var/www/certbot \
    $staging_arg \
    $email_arg \
    $domain_args \
    --rsa-key-size $rsa_key_size \
    --agree-tos \
    --force-renewal" certbot
echo

```

```
echo "### Reloading nginx ..."  
docker-compose exec nginx nginx -s reload
```

## init-letsencrypt 실행

```
# 파일 실행 권한 변경  
chmod +x init-letsencrypt.sh  
  
# init-letsencrypt 셸스크립트 실행  
sudo ./init-letsencrypt.sh
```

- `chmod +x` : change mode 권한 변경 파일 실행 (x = execute)
- `init-letsencrypt` 실행 과정에서 `nginx`가 restart 된다.

## nginx 재실행

- 인증서 위치에 맞게 `docker` 볼륨 마운트
- `nginx.conf` 인증키 주석 없이 `docker-compose`로 재실행

```
# nginx 컨테이너 스탑  
docker-compose stop haruman_nginx_1  
  
# nginx 컨테이너 삭제  
docker rm haruman_nginx_1  
  
# 도커 컴포즈 다시 실행  
docker-compose up -d
```

## 4. CI/CD 구축

### Jenkins 환경설정















- 플러그인 설치
  - Gradle

- Docker
- Docker-compose
- Sonarqube

## Credentials 설정

- 토큰
  - Gitlab API token
  - GitLab Pipeline token
  - EC2 pemkey
- 주입 파일
  - application-secret.yml
  - application-develop.yml

## Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	application-secret	<a href="#">application-secret.yml</a>
		System	(global)	GitLab API token	<a href="#">GitLab API token</a>
		System	(global)	application-develop	<a href="#">application-develop.yml</a>
		System	(global)	SonarQube token	<a href="#">SonarQube token</a>
		System	(global)	EC2 pemkey	<a href="#">J9A607T.pem</a>
		System	(global)	application-production	<a href="#">application-production.yml</a>
		System	(global)	GitLab Pipeline token	<a href="#">davidpoplar@naver.com/*****</a>

## Tools 설정

- JDK
  - <https://corretto.aws/downloads/latest/amazon-corretto-11-x64-linux-jdk.tar.gz>

JDK ✕

Name

jdk-11

☒ Install automatically ?

☰ Extract \*.zip/\*.tar.gz ?

✕

Download URL for binary archive ?

https://corretto.aws/downloads/latest/amazon-corretto-11-x64-linux-jdk.tar.gz

Subdirectory of extracted archive ?

amazon-corretto-11.0.20.9.1-linux-x64

Advanced ▾

Add Installer ▾

- Gradle

Gradle ✕

name ?

gradle-8.2.1

☒ Install automatically ?

☰ Install from Gradle.org

✕

Version

Gradle 8.2.1 ▾

Add Installer ▾

- NodeJS

NodeJS

Name

node-18

☒ Install automatically

Install from nodejs.org

Version

NodeJS 18.17.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail
☐ Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax 'packageName@version'

Global npm packages refresh hours

Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

72

Add Installer

- Docker

Docker

Name

docker

☒ Install automatically

Download from docker.com

Docker version

latest

Add Installer

## System 설정

- SonarQube servers

Name

SonarQube-haruman

Server URL

Default is http://localhost:9000

https://sonarqube.ssafy.com/

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

SonarQube token

Add

Advanced

- Gitlab

Connection name

A name for the connection

haruman

GitLab host URL

The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)

https://lab.ssafy.com

Credentials

API Token for accessing GitLab

GitLab API token

Add

Advanced

Test Connection

- Mattermost

#### Global Mattermost Notifier Settings

Endpoint ?

https://meeting.ssafy.com/hooks/a97c5c8hw7rozgacwhxi6rurr

Channel ?

A607\_Jenkins

Icon to use ?

Build Server URL ?

http://3.34.135.50:8080/

Mattermost Custom Proxy Settings ▾

Test Connection

## Jenkins Pipeline 생성

### Build Triggers 설정

- webhook URL 저장 : http://3.38.252.22:8080/project/readed-backend-develop
- Advanced에서 Secret token generate 후 저장

▼ 캡처



### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: `http://3.38.252.22:8080/project/readed-backend-develop` ?

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

Advanced ^

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

a1e82457b2219afd6e01ff16a8125668

Generate

Clear

## Pipeline 설정

- Pipeline script from SCM으로 설정
- Repository URL 입력
- Credentials 선택

- Branches to build 입력 : 빌드할 브랜치명

- Script Path 입력 : Jenkinsfile 위치

▼ 캡처

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssafy.com/s09-webmobile1-sub2/S09P12A507.git

Credentials ?

loginonlyyy@gmail.com/\*\*\*\*\*

Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

be/develop

Add Branch

Repository browser ?

(Auto) ▾

Additional Behaviours

Add ▾

Script Path ?

backend/Jenkinsfile

☒ Lightweight checkout ?

## GitLab Webhook 생성

- URL : 젠킨스 파이프라인 webhook URL 입력
- Secret token : 젠킨스 파이프라인 generate한 Secret token 입력
- Trigger - Push events : trigger 시킬 브랜치명 설정

Q Search page

### Webhook

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**

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

☒ Show full URL  
☐ Mask portions of URL  
 Do not show sensitive data such as tokens in the UI.

**Secret token**

Used to validate received payloads. Sent with the request in the `X-6t1ab-Token` HTTP header.

**Trigger**

☒ Push events

☐ All branches

☒ Wildcard pattern

Wildcards such as `*-stable` or `production/*` are supported.

☐ Regular expression

☐ Tag push events  
A new tag is pushed to the repository.

☐ Comments  
A comment is added to an issue or merge request.

- 생길 수 있는 에러
  - 403 / 401
  - Secret token을 제대로 가져오지 않았거나 / URL 입력을 잘못했거나
- Test로 Webhook 테스트 가능

## Jenkinsfile, Dockerfile 작성

### backend Jenkinsfile

```
pipeline {
    agent any

    tools {
        gradle 'gradle-8.2.1'
        jdk 'jdk-11'
        dockerTool 'docker'
    }
}
```

```

stages {
  stage('Clear current directory') {
    steps {
      sh'''
        rm -rf *
      '''
    }
  }

  stage('Pull from GitLab') {
    steps {
      git url: 'https://lab.ssafty.com/s09-fintech-finance-sub2/S09P22A607.git',
        branch: 'be/develop',
        credentialsId: 'GitLab Pipeline token'
    }
  }

  stage('Apply application.yml files') {
    steps {
      withCredentials([file(credentialsId: 'application-secret', variable: 'secretFile'),
        file(credentialsId: 'application-develop', variable: 'developFile'),
        file(credentialsId: 'EC2 pemkey', variable: 'pemkey')]) {
        script {
          sh 'cp $secretFile backend/src/main/resources/application-secret.yml'
          sh 'cp $developFile backend/src/main/resources/application-develop.yml'
          sh 'cp $pemkey J9A607T.pem'
        }
      }
    }
  }

  stage('Build Backend') {
    steps {
      dir('backend') {
        sh'''
          gradle wrapper
          chmod +x gradlew
          ./gradlew clean build -x test --stacktrace
        '''
      }
    }
  }

  stage('Zero Downtime Deployment') {
    steps {
      sh'''

```

```

echo "\n***** Blue Health Check & Build *****"

# 8081 포트가 살아있다면 -> 8082 포트 그린에 배포
# 8081 포트가 죽어있다면 -> 8081 포트 블루에 배포
if curl -s "http://3.34.135.50:8081" > /dev/null
then
    deployment_target_port=8082
    deployment_target=green
    opposite_target=blue
else
    deployment_target_port=8081
    deployment_target=blue
    opposite_target=green
fi

# 해당 포트의 기존 이미지가 존재하면 삭제
echo delete existing Docker image
if docker image inspect server_${deployment_target} >/dev/null 2>&1; t
hen
    docker rmi server_${deployment_target}
    echo "${deployment_target}-image exists locally & removed"
else
    echo "${deployment_target}-image does not exist locally"
fi

# 백 빌드를 해서 타겟포트(호스트) 8081(컨테이너내부) 이미지로 만들어서
# 블루 그린 이름을 붙여서 도커 run
echo [BE] ${deployment_target} Build Docker Image
docker build -t server_${deployment_target} ./backend
echo [BE] ${deployment_target} Run Docker Container
docker run -dp ${deployment_target_port}:8081 --name haruman_server_
${deployment_target} server_${deployment_target}

echo "\n***** Continuous Health Check *****"

# 10초마다 헬스 체크 10번 -> 실패하면 그린 배포 멈춤
# 주로 2번 정도 돌아가면 성공함
for retry_count in $(seq 10)
do
    if curl -s "http://3.34.135.50:${deployment_target_port}" > /dev/nul
1
    then
        echo "Health check success 🟢"
        break
    fi

    if [ $retry_count -eq 10 ]
    then
        echo "Health check failed 🛑"
        exit 1
    fi

```

```

        echo "The server is not alive yet. Retry health check in 10 second
s..."
        sleep 10
    done

    echo "\n***** Nginx Proxy Redirect *****"

    # nginx 컨테이너 접속
    ssh -o StrictHostKeyChecking=no -i J9A607T.pem ubuntu@j9a607.p.ssafy.i
o /bin/bash

    # $service_url을 target_port로 바꾸는데 기존 값은 중요하지 않음
    docker exec haruman_nginx_1 bash -c "echo 'set \\\$service_url 3.34.13
5.50:${deployment_target_port};' > /etc/nginx/conf.d/service-url.inc ; service nginx reloa
d"

    echo "Switch the reverse proxy direction of nginx to [${deployment_tar
get}] http://3.34.135.50:${deployment_target_port}"

    echo "\n***** Kill the process on the opposite server *****"
    # 리다이렉트 후 다른 컨테이너가 존재하면 내림
    if docker container inspect haruman_server_${opposite_target} >/dev/nu
ll 2>&1; then
        echo "${opposite_target}-container exists locally & removed"
        docker stop haruman_server_${opposite_target}
        docker rm haruman_server_${opposite_target}
    else
        echo "${opposite_target}-container does not exist locally"
    fi

    '''
    }
}
}

post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: tru
e).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: tru
e).trim()
            def GIT_COMMIT_MSG = sh(script: 'git log -1 --pretty=%B ${GIT_COMMIT}', re
turnStdout: true).trim()
            mattermostSend(color: 'good', message: "✅ 빌드 & 배포 성공: ${env.JOB_NAME}
(<${env.BUILD_URL}|#${env.BUILD_NUMBER}>)\n브랜치: be/develop\n커밋 메시지: ${GIT_COMMIT_MSG}
by ${Author_ID}(${Author_Name})")
        }
    }
    failure {
        script {

```



```

        def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
        def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
        def GIT_COMMIT_MSG = sh(script: 'git log -1 --pretty=%B ${GIT_COMMIT}', returnStdout: true).trim()
        mattermostSend(color: 'danger', message: "❌ 빌드 & 배포 실패: ${env.JOB_NAME} (<${env.BUILD_URL}|#${env.BUILD_NUMBER}>)\n브랜치: be/develop\n커밋 메시지: ${GIT_COMMIT_MSG} by ${Author_ID}(${Author_Name})")
    }
}
}
}

```

## backend Dockerfile

```

FROM adoptopenjdk/openjdk11
ARG JAR_FILE=build/libs/*-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "-Dserver.port=8081", "-Dspring.profiles.active=develop", "/app.jar"]

```

## frontend Jenkinsfile

```

pipeline {
    agent any

    tools {
        nodejs 'node-16'
        dockerTool 'docker'
    }

    stages {
        stage('Clear current directory') {
            steps {
                sh'''
                    rm -rf *
                '''
            }
        }

        stage('Pull from GitLab') {
            steps {
                git url: 'https://lab.ssafty.com/s09-fintech-finance-sub2/S09P22A607.git',

```

```

        branch: 'fe/develop',
        credentialsId: 'GitLab Pipeline token'
    }
}

stage('Build Frontend') {
    steps {
        dir('frontend') {
            sh'''
                npm install -g yarn
                yarn install
                CI=false yarn build
            '''
        }
    }
}

stage('Delete existing Docker images and containers') {
    steps {
        sh'''
            if docker container inspect haruman_client >/dev/null 2>&1; then
                echo "container exists locally"
                docker stop haruman_client
                docker rm haruman_client
            else
                echo "container does not exist locally"
            fi
            if docker image inspect client >/dev/null 2>&1; then
                echo "Image exists locally"
                docker rmi client
            else
                echo "Image does not exist locally"
            fi
        '''
    }
}

stage('Build and Deploy Docker') {
    steps {
        dir('frontend') {
            sh'''
                echo [FE] Build Docker Image!
                docker build -t client .
                echo [FE] Run Docker Container!
                docker run -dp 3000:3000 --name haruman_client client
            '''
        }
    }
}

post {

```

```

        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                def GIT_COMMIT_MSG = sh(script: 'git log -1 --pretty=%B ${GIT_COMMIT}', returnStdout: true).trim()
                mattermostSend(color: 'good', message: "✅ 빌드 & 배포 성공: ${env.JOB_NAME} (<${env.BUILD_URL}|#${env.BUILD_NUMBER}>)\n브랜치: fe/develop\n커밋 메시지: ${GIT_COMMIT_MSG} by ${Author_ID}(${Author_Name})")
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
                def GIT_COMMIT_MSG = sh(script: 'git log -1 --pretty=%B ${GIT_COMMIT}', returnStdout: true).trim()
                mattermostSend(color: 'danger', message: "❌ 빌드 & 배포 실패: ${env.JOB_NAME} (<${env.BUILD_URL}|#${env.BUILD_NUMBER}>)\n브랜치: fe/develop\n커밋 메시지: ${GIT_COMMIT_MSG} by ${Author_ID}(${Author_Name})")
            }
        }
    }
}

```

## frontend Dockerfile

```

# nginx 이미지 pull
FROM nginx
# app 디렉토리 생성
WORKDIR /app
# workdir에 build 폴더 생성 /app/build
RUN mkdir ./build
# build에서 build 폴더로 이동
ADD ./build ./build
# nginx의 기본 설정을 삭제
RUN rm -rf /etc/nginx/nginx.conf
# nginx 설정 파일 복사
COPY ./react-nginx.conf /etc/nginx/nginx.conf
# 80포트 오픈하고 nginx 실행
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]

```

## frontend react-nginx.conf

- 빌드된 프론트엔드 프로젝트를 nginx로 연결

```
user  nginx;
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;

    server {
        listen 3000;
        listen [::]:3000;

        server_name _;

        location / {
            root   /app/build;
            index  index.html index.htm;
            try_files $uri /index.html;
        }

        error_page   500 502 503 504  /50x.html;

        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
}
```