포팅 매뉴얼

🔎 Stacks

Management Tool

`Jira` `Mattermost` `Discord` `Notion` `GitLab` `Figma`

IDE

`Vscode(1.8.6)` `Intellij(2023.3.2)`

Infra

`Amazon S3(1.12.652)` `Nginx(1.18.0)` `Docker(25.0.1)` `Ubuntu(20.04.6)` `EC2`

Frontend

`HTML5` `CSS` `JS` `React(18.2.0)` `Axios(1.6.7)` `React-Router-Dom(6.21.2)` `Node(20.10)` `React-Dom(18.2.0)` `Zustand(4.5.0)` `@tanstack/react-query(5.18.1)` `Prettier(3.2.5)` `Styled-Components(6.1.8)`

Backend

`Java(17)` `Spring boot(3.2.1)` `JPA` `jwt(0.12.4)` `QueryDsl(5.0.0)` `Mariadb(10.11.6)` `Redis(7.2.4)` `Nurigo(4.3.0)`

📥 Build & Distribute

Backend Build

application.properties

```
server.port={Sevrver Port}
server.servlet.context-path={Server context-path}
spring.datasource.driver-class-name={DB Driver}
spring.datasource.url={DB URL}
spring.datasource.username={DB username}
spring.datasource.password={DB password}

# Hibernate의 DDL 자동 생성 기능 비활성화
spring.jpa.hibernate.ddl-auto=none

# SQL 쿼리의 가독성 증진
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.highlight_sql=true

# Hibernate가 생성하는 SQL 쿼리를 출력
spring.jpa.properties.hibernate.show-sql=true

# Hibernate의 SQL 매개 변수 바인딩 정보를 디버그 수준으로 로깅
logging.level.org.hibernate.type.descriptor.sql=DEBUG
logging.level.org.hibernate.SQL=DEBUG

# 로깅 구성 설정 파일
logging.config=classpath:logback-spring.xml

# 콘솔 출력 시 ANSI 색상 출력 활성화
spring.output.ansi.enabled=always

# Hibernate의 물리적 네이밍 전략 설정
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingS
```

```
trategyStandardImpl

# Spring 웹 리소스 핸들러가 기본적으로 제공하는 자원 매핑 비활성화
# 404 Error 처리를 위함
spring.web.resources.add-mappings=false

spring.data.redis.host={Redis의 DNS 혹은 URL}
spring data.redis.password={Redis의 비밀번호}
spring.data.redis.token.port={RefreshToken Redis Port}
spring.data.redis.association.port={연동 계정 인증번호 Redis Port}
spring.data.redis.certification.port={전화번호 인증번호 Redis Port}
spring.data.redis.sse.port={SSE 채널 등록 Redis Port}

sms.api.key={nurigo SMS API KEY}
sms.api.secret={nurigo SMS SECRET KEY}
sms.number={전송하는 전화번호}

jwt.issuer={JWT 발급자(issuer) 지정}
jwt.secretKey={JWT의 서명에 사용되는 비밀 키 지정}
jwt.refresh-token={JWT RefreshToken에 사용되는 키 지정}
jwt.claim-name={JWT의 페이로드에 사용되는 claim의 이름 지정}

cookie.age={Cookie의 유효 기간 설정}

spring.servlet.multipart.enabled={Spring의 Multipart 요청 처리 활성화 여부}
spring.servlet.multipart.max-file-size={업로드되는 각 파일의 최대 크기 지정}
spring.servlet.multipart.max-request-size={요청의 최대 크기 설정}

cloud.aws.s3.bucket={Amazon S3 버킷 이름 설정}
cloud.aws.s3.region.static={Amazon S3 버킷의 리전 설정}
cloud.aws.s3.credentials.accessKey={Amazon S3 액세스 키 설정}
cloud.aws.s3.credentials.secretKey={Amazon S3 시크릿 키 설정}

image.prefix={이미지 경로의 접두사 지정}
image.size.profile={프로필 이미지의 크기 설정}
image.size.report={정기 보고 이미지의 크기 설정}

map.range={지도 기반에서 집사를 찾는 지도 범위 설정}

code.phone.start={전화번호 인증번호의 시작 값 설정}
code.phone.end={전화번호 인증번호의 종료 값 설정}
code.association.time={연동 계정 인증번호의 유효 시간 설정}
code.association.start={연동 계정 인증번호의 시작 값 설정}
code.association.end={연동 계정 인증번호의 종료 값 설정}

default.timeout={SSE의 기본 Timeout 시간 설정}

date.to.minute={집사의 알림 회신 시간 계산을 위한 설정 값}
```

IDE 및 환경설정

```
1. jdk 17 다운로드 및 환경변수 설정
2. git clone 후 backend 폴더를 Intellij에서 Open하여 가져오기
3. Intellij - File - Project Structure - Project에서 SDK를 17버전으로 맞추기
4. Intellij - File - Settings - Gradle에서 Gradle JVM을 [1]에서 추가한 환경변수로 지정
```

```
5. Intellij 우측 Gradle 클릭 후 새로고침
6. Intellij - Run - BackendApplication으로 실행
```

Dockerfile (Back)

```
FROM docker
COPY --from=docker/buildx-bin:latest /buildx /usr/libexec/docker/cli-plugins/docker-buildx

FROM openjdk:17-jdk
EXPOSE 443
ADD ./build/libs/*.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Frontend Build

.env

```
REACT_APP_JAVASCRIPT_KEY={Kakao Map API의 JavaScript Key}
```

IDE 및 환경설정

```
1. Node.js 20.10 다운로드 및 환경변수 설정
2. git clone 후 frontend 폴더를 vscode에서 Open하여 가져오기
3. npm install -g yarn
4. yarn install
5. frontend 폴더 상위 경로에 .env 파일 위치시키기
6. yarn start
```

Dockerfile (Front)

```
FROM nginx:latest

RUN mkdir /app

WORKDIR /app

RUN mkdir ./build

ADD ./build ./build

RUN rm /etc/nginx/conf.d/default.conf

COPY ./nginx.conf /etc/nginx/conf.d

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

배포 파일 폴더 구조

```
backend
├ .gradle
├ Dockerfile
├ gradle
├ gradlew
├ gradlew.bat
├ settings.gradle
└ src
│  ├ main
│  │  ├ java.com.a407.back
│  │  │   ├ BackendApplication.java
│  │  │   ├ config
│  │  │   │  ├ constants
│  │  │   │  ├ jwt
│  │  │   │  └ redis
│  │  │   ├ controller
│  │  │   ├ domain
│  │  │   ├ dto
│  │  │   │  ├ association
│  │  │   │  ├ auth
│  │  │   │  ├ board
│  │  │   │  ├ comment
│  │  │   │  ├ match
│  │  │   │  ├ notification
│  │  │   │  ├ review
│  │  │   │  ├ room
│  │  │   │  ├ user
│  │  │   │  ├ util
│  │  │   │  └ zipsa
│  │  │   ├ exception
│  │  │   └ model
│  │  │      ├ repo
│  │  │      └ service
│  │  └ resources
│  └ test.java.com.a407.back
│      ├ BackApplicationTests.java
│      ├ controller
│      └ security
└ build.gradle
```

```
frontend
├ public
├ src
│  ├ apis
│  │  ├ api
│  │  └ utils
│  ├ assets
│  │  ├ fonts
│  │  └ styles
│  ├ components
│  │  ├ boards
│  │  ├ common
│  │  ├ filter
│  │  └ zipsamypage
│  ├ constants
│  ├ utils
│  ├ pages
│  │  ├ boards
│  │  ├ connect
│  │  ├ createRoomFunnel
│  │  ├ error
│  │  ├ filterFunnel
│  │  ├ home
│  │  ├ login
│  │  ├ mapFunnel
│  │  ├ myPage
│  │  ├ notify
│  │  ├ registerFunnel
│  │  ├ report
│  │  ├ reserve
│  │  ├ result
│  │  ├ userMyPage
│  │  ├ zipsaMyPage
│  │  └ zipsaRoom
│  ├ App.js
│  ├ index.css
│  └ index.css
├ README.md
├ .env
├ .prettierrc
├ Dockerfile
├ nginx.conf
├ package-lock.json
├ package.json
└ yarn.lock
```

⚙️ Server Settings

MobaXterm

- MobaXterm을 통해 EC2 서버에 접속

```
1. 좌측 상단의 Session - SSH 클릭
2. 필요 정보 입력
    - 1. Remote Host: EC2 Domain 입력
    - 2. Specify username: 체크 후 ubuntu 입력
3. Advanced SSH settings 탭 클릭
4. 필요 정보 입력
    - 1. Use private key 체크 후 .pem 파일 첨부
5. Bookmark settings - Session name에서 원하는 서버 이름 입력
```

Server Default Setting

- 한국 표준시로 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

- 패키지 목록 업데이트 및 패키지 업데이트

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

Docker Setting

- Docker 설치 전 필요한 패키지 설치

```
sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-propert
ies-common
```

- amd / arm 확인

```
dpkg -s libc6 | grep Arch
```

- 위에 해당하는 계열로 Docker 레포지토리 등록
  임의로 amd / arm ⇒ ver로 작성

```
sudo add-apt-repository "deb [arch=ver64] https://download.docker.com/linux/ubuntu$(lsb_relea
se -cs) stable"
```

- 패키지 리스트 갱신

```
sudo apt-get -y update
```

- Docker 패키지 설치

```
sudo apt-get -y install docker-ce docker-ce-cli containerd.io
```

- Docker 서비스 재시작

```
sudo service docker restart
exit
```

SSL Setting

- nginx를 설치

```
sudo apt-get -y install nginx
```

- CertBot 다운로드

```
sudo snap install --classic certbot
```

- SSL 인증서 발급

```
sudo nginx --nginx -d my.domain.com
이후에 이메일 주소 입력
```

NginX Setting

- NginX HTTP 방화벽 허용

```
sudo ufw app list
sudo ufw allow 'Nginx HTTP'
sudo ufw enable
```

- NginX 재시작 및 상태 확인

```
sudo service nginx restart
sudo service nginx status
```

- default 설정 편집
  - proxy_pass 경로를 지정하기 위해 url 추가

```
sudo vim /etc/nginx/sites-enabled/default
====> 아래 내용 추가
include /etc/nginx/conf.d/service-url.inc;
include /etc/nginx/conf.d/client-url.inc;
====> location / 안에 설정
proxy_pass $client_url;
====> location /api 안에 설정
proxy_pass $service_url;
```

- service-url.inc 추가

```
sudo vim /etc/nginx/conf.d/service-url.inc
====> 아래 내용 추가
set $service_url http://127.0.0.1:8081;
```

- client-url.inc 추가

```
sudo vim /etc/nginx/conf.d/client-url.inc
====> 아래 내용 추가
set $client_url http://127.0.0.1:3000;
```

- 추가 후 nginx 재시작

MariaDB Setting

- MariaDB 설치(Docker)

```
sudo docker pull mariadb:latest
```

- MariaDB 컨테이너 실행

```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD={비밀번호} -v /var/lib/mysql:/var/lib/mysql --name mariadb mariadb
```

- MariaDB 컨테이너 접속

```
docker exec -it mariadb /bin/bash
```

- 데이터베이스 접속

```
mariadb -u root -p
====> 패스워드 입력
```

- 데이터베이스 생성

```
create database 데이터베이스이름;
```

Redis Setting

- Redis Image 다운로드

```
docker pull redis:latest
```

- Redis Docker Container 생성

```
sudo docker run -d -p {port}:6379 --name token-redis --network bridge redis:latest
sudo docker run -d -p {port}:6379 --name association-redis --network bridge redis:latest
sudo docker run -d -p {port}:6379 --name certification-redis --network bridge redis:latest
sudo docker run -d -p {port}:6379 --name sse-redis --network bridge redis:latest
```

- Redis 접속

```
docker exec -i -t {Redis Container명} redis-cli
```

- Redis 동작 확인

```
ping
====> 답: PONG
```

- Redis 비밀번호 지정

```
config set requirepass {Redis password}
```

- 이후 Redis 접속 시 입력

```
AUTH '{Redis password}'
```

Jenkins Default Setting

- Java 설치

```
sudo apt install openjdk-17-jdk
```

- Jenkins 저장소 Key 다운로드

```
wget -q -O - https://pkg.jenkins.io.debian/jenkins-ci.org.key | sudo apt-key add -
```

- sources.list.d에 jenkins.list 추가

```
echo deb http://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt sources.list.d/jenki
ns.lst
```

- Key 등록

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys FCEF32E745F2C3D5
```

- apt update

```
sudo apt update
```

- Jenkins 설치

```
sudo apt install jenkins
```

- Jenkins 서버 포트 번호 변경

```
sudo vi /etc/default/jenkins
====> 아래 원하는 포트 번호 입력
HTTP_PORT=8080
```

- Jenkins 서비스 재시작

```
sudo service jenkins restart
```

- Jenkins 포트 방화벽 설정

```
sudo ufw allow 8080
sudo ufw enable
```

- Jenkins pipeline에서 sudo를 사용할 수 있도록 설정

```
sudo visudo
====> 아래 코드 추가
jenkins ALL=(ALL) NOPASSWD: ALL
```

Docker Hub Setting

- Docker Hub Token 발급

```
1. 우측 상단의 Sign in 버튼을 클릭하여 로그인
2. 우측 상단의 계정명을 클릭하여 Account Settings 클릭
3. New Access Token - Read,Write,Delete 권한을 가진 Token 발급
4. Token 값 저장
```

- Docker Hub Repository 생성

```
1. 상단의 Repositories - Create repository 클릭
2. Visibility 지정 후 Create 클릭
```

Jenkins Pipeline Setting

- 플러그인 설치
  - Jenkins 관리 - Plugins - Available Plugins - 선택 후 Install without restart 클릭

```
SSH Agent

Docker
Docker Commons
Docker Pipeline
Docker API

Generic Webhook Trigger

GitLab
GitLab API,
GitLab Authentication

NodeJS

Mattermost Notification
```

- Docker Hub Credential 등록
  - Jenkins 관리 - Credentials - global - Add Credentials - Create

```
Kind: Username with password
Username: Docker Hub에서 사용하는 ID
Password: Docker Hub에서 사용하는 Token 값
ID: Credential에 대한 별칭
```

- GitLab Credential 등록
  - Jenkins 관리 - Credentials - global - Add Credentials - Create

```
Kind: Username with password
Username: GitLab 계정 아이디 입력
Password: GitLab 계정 비밀번호
ID: Credential에 대한 별칭
```

- Ubuntu Credential 등록

    - Jenkins 관리 - Plugins - Available Plugins - SSH Agent
    - Jenkins 관리 - Credentials - global - Add Credentials - Create

```
Kind: Username with private key
ID: Credential에 대한 별칭
Username: SSH 원격 서버 호스트에서 사용하는 계정명(ubuntu)
====> Enter directly - Add 클릭
.pem 키의 내용을 메모장을 읽어 복사 후 Key에 붙여넣은 후 Create
```

- application.properties 등록
    - Jenkins 관리 - Credentials - global - Add Credentials - Create

```
Kind: Secret file
File: application.properties 첨부
ID: Credential에 대한 별칭
```

- .env 등록
    - Jenkins 관리 - Credentials - global - Add Credentials - Create

```
Kind: Secret file
File: .env 첨부
ID: Credential에 대한 별칭
```

- Item 추가

```
1. 새로운 Item 클릭
2. Pipeline 클릭 후 OK
3. Configure - General - GitLab Connection 선택
4. Build Triggers의 Build when a change is pushed to GitLab 체크
```

- Gradle 추가
    - Jenkins 관리 - Tools

```
name: gradle
Install automatically 체크
프로젝트 버전에 맞는 Gradle 선택 후 Save
```

- Node.js 추가
    - Jenkins 관리 - Tools

```
Name: Node.js 환경에 대한 이름
Version: 빌드하려는 Node.js 버전 선택 후 Save
```

- Node.js 빌드 시 사용하는 환경변수 설정
    - Jenkins 관리 -System
    - Global properties

```
1. Environment variables 체크
2, CI, false 환경변수 추가 - 저장
```

🪣 S3 Setting

IAM 설정

- 사용자 생성

```
1. 사용자 이름 지정
2. 권한 옵션에서 직접 정책 연결
3. 검색 창에 AmazonS3FullAccess 체크
4. 사용자 생성
```

- 액세스 키 발급

```
1. 생성한 사용자선택
2. 액세스 키 만들기 선택
3. 기타 선택
4. 액세스 키 생성
5. 액세스 키와 비밀 액세스 키 저장
```

- S3 버킷 생성

```
1. ACL 활성화
2. 모든 퍼블릭 액세스 차단 해제
3. 주의사항 체크
4. 버킷 만들기
```

- ACL 설정

```
1. 생성된 버킷 오픈
2. 권한 목록으로 이동
3. ACL(액세스 제어 목록) 편집
4. 모든 사람(퍼블릭 엑세스)에서 읽기에 체크
5. 주의사항 체크
6. 변경 사항 저장
```

Lambda 설정

- Lambda 세부 설정

```
1. 함수 생성
2. 새로 작성
3. 함수 이름 설정
4. 런타임 Node.js 20.x
5. 아키텍처 x86_64
6. 함수 생성
```

- 트리거 추가

```
1. 트리거 추가 선택
2. 소스 선택
3. S3 선택
4. 생성한 Bucket 선택
5. 주의사항 체크
6. 추가
```

- aws-sdk 업로드 파일 생성

```
1. 빈 폴더를 생성
2. npm init으로 npm 설정
3. npm i @aws-sdk/client-s3 입력
4. 생성된 node_modules, package-lock.json, package.json zip으로 압축
```

- aws-sdk 계층 생성

```
1. 계층 탭으로 이동
2. 계층 생성
3. 이름 설정
4. aws-sdk zip파일 업로드
5. 호환 아키텍처 x86_64 체크
6. 호환 런타임 Node.js 20.x 선택
7. 생성
```

- sharp 업로드 파일 다운로드

```
1. https://github.com/pH200/sharp-layer 접속
2. release-x64.zip 다운로드
```

- sharp 계층 생성

```
1. 계층 탭으로 이동
2. 계층 생성
3. 이름 설정
4. sharp zip파일 업로드
5. 호환 아키텍처 x86_64 체크
6. 호환 런타임 Node.js 20.x 선택
7. 생성
```

- 계층 추가

```
1. Lambda 함수 코드 탭의 마지막으로 이동
2. 계층 정보에서 [Add a layer] 선택
3. 사용자 지정 계층 선택
4. 위에서 생성한 aws-sdk 계층 선택
5. 추가
6. 다시 반복을 하여 sharp 계층 추가
```

- index.mjs에 코드 작성

```
// dependencies
const { S3Client, GetObjectCommand, PutObjectCommand } = require('@aws-sdk/client-s3');
const { Readable } = require('stream');
const sharp = require('sharp');
const util = require('util');


// create S3 client
const s3 = new S3Client();

const bucketName = "{자신의 Bucket Name}";

// define the handler function
exports.handler = async (event, context) => {
```

```
    console.log("Reading options from event:\n", util.inspect(event, { depth: 5 }));
    const srcBucket = event.Records[0].s3.bucket.name;

    const srcKey = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    const dstBucket = srcBucket;
    const dstKey = srcKey;

    const typeMatch = srcKey.match(/\.([^.]*)$/);
    if (!typeMatch) {
        console.log("Could not determine the image type.");
        return;
    }

    const imageType = typeMatch[1].toLowerCase();
    if (imageType != "jpg" && imageType != "png") {
        console.log(`Unsupported image type: ${imageType}`);
        return;
    }

    try {
        const params = {
            Bucket: srcBucket,
            Key: srcKey
        };
        var response = await s3.send(new GetObjectCommand(params));
        var stream = response.Body;

        if(!response.Metadata.size){
            console.log("Size Error")
            return;
        }

        if (stream instanceof Readable) {
            var content_buffer = Buffer.concat(await stream.toArray());

        } else {
            throw new Error('Unknown object stream type');
        }


    } catch (error) {
        console.log(error);
        return;
    }

    if (response.Metadata.size) {

        const size = parseInt(response.Metadata.size);

        try {
            var output_buffer = await sharp(content_buffer).resize(size, size, { fit: sharp.fit
        } catch (error) {
            console.log(error);
            return;
        }

        try {
            const destparams = {
```

```
                Bucket: dstBucket,
                Key: dstKey,
                Body: output_buffer,
                ContentType: "image",
                ACL: 'public-read'
            };
            const putResult = await s3.send(new PutObjectCommand(destparams));
        } catch (error) {
            console.log(error);
            return;
        }

        console.log('Successfully resized ' + srcBucket + '/' + srcKey +
            ' and uploaded to ' + dstBucket + '/' + dstKey);
    }
};
```

- 스프링 부트에서 S3에 이미지 업로드

```
1. Metadata에 Size 속성에 리사이즈 할 이미지의 사이즈를 추가
2. ACL pulblicRead 추가
```

💽 DataGrip Connection

```
# EC2에 MariaDB 컨테이너 구동 후 연결
1. DataGrip 접속
2. DataGrip - File - Data Sources... - +버튼 누른 후 MariaDB 선택
3. 필요 정보 입력
    - 1. Host: MariaDB가 구동되고 있는 컨테이너의 이름으로 docker inspect {컨테이너명} 한 후에 출력되는 I
PV4 Address 입력
  - 2. Port: MariaDB가 구동되고 있는 컨테이너와 포트포워딩 된 로컬 Port
  - 3. User: username 입력
  - 4. Password: password 입력
4. SSH/SSL 클릭 - Use SSH tunnel 체크
5. SSH Configuration 옆 ... 클릭 - 좌측 상단 + 클릭 후 필요 정보 입력
  - 1. Host: EC2 Domain 명 입력
  - 2. Username: ubuntu 입력
  - 3. Authentication type: Key pair 선택
  - 4. Private key file: .pem 파일 첨부
  - 5. Parse config file ~/.ssh/config 체크 - Test Connection 클릭 후 OK
6. Test Connection 클릭 후 OK
```

- dump 설정 시

```
1. database 우클릭 - New - Query Console
2. ddl.sql을 DataGrip에서 open 후 Query Console에 붙여넣기
3. 좌측 상단의 Execute로 모든 ddl 실행
4. dump.sql을 DataGrip에서 open 후 Query Console에 붙여넣기
5. 좌측 상단의 Execute로 모든 dump 실행
```

📜 API Setting

Nurigo(Coolsms)

- Nurigo(Coolsms) 설정

```
# Backend
1. 실명 인증 후 발신번호 자동 등록
2. API Key 생성
3. API Key, API Secret 저장
```

Kakao Map API

- Kakao Map API 설정

```
# Frontend
1. public에 해당 코드 설정
    - REACT_APP_JAVASCRIPT_KEY는 env 파일에 있는 환경 변수
<script
      type="text/javascript"
      src="//dapi.kakao.com/v2/maps/sdk.js?appkey=%{REACT_APP_JAVASCRIPT_KEY}%&libraries=clus
terer,services"
></script>
```

🎬 Deployment Command

Pipeline (Back)

```
pipeline {
    agent any

    environment {
        imageName = "{backend Image}"
        registryCredential = '{Docker Hub Credential}'

        releaseServerAccount = '{EC2 username}'
        releaseServerUri = '{EC2 Domain}'

        deployColor = ''
        beforeColor = ''
        nginxConfigFile = '/etc/nginx/conf.d/service-url.inc'
    }

    tools {
        gradle 'gradle'
    }

    stages {
        stage('clone'){
            steps{
                git branch: '{backend branch}',
                    credentialsId: '{gitlab Credential}',
                    url: {GitLab 주소}
                dir('backend') {
                                    // 프로젝트에 application.properties를 추가
```

```
                    withCredentials([file(credentialsId: '{application.properties Credentia
l}', variable: 'properties')]) {
                    script {
                        sh 'cp -f $properties ./src/main/resources/application.properties'
                        }
                    }
                }
            }
        }
        stage('Jar Build') {
            steps {
                dir ('backend') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                }
            }
        }
        stage('Trigger') {
            steps {
                script {
                    def containerNames = sh(script: 'docker ps --format \'{{.Names}}\'', retu
rnStdout: true).trim()
                    def targetContainerName = "backend-{second color name}"
                    def isContainerFound = containerNames.split().contains(targetContainerNam
e)
                    if(isContainerFound) {
                        deployColor = '{first color name}'
                        beforeColor = '{second color name}'
                    } else {
                        deployColor = '{second color name}'
                        beforeColor = '{first color name}'
                    }
                  }
               }
           }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('backend/') {
                    script {
                        docker.withRegistry('', registryCredential) {
                            sh "docker buildx create --use --name mybuilder"
                            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $im
ageName-$deployColor:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $im
ageName-$deployColor:latest --push ."
                        }
                    }
                }
            }
        }
        stage('DockerHub Pull') {
            steps {
                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerU
ri 'sudo docker pull $imageName-$deployColor:latest'"
                }
            }
        }
```

```
        stage('Deploy') {
            steps {
                script {
                    if(deployColor == '{first color name}') {
                        BEFORE_COLOR = "{second color name}"
                        AFTER_COLOR = "{first color name}"
                        BEFORE_PORT = "{second color port}"
                        AFTER_PORT = "{first color port}"
                    } else {
                        BEFORE_COLOR = "{first color name}"
                        AFTER_COLOR = "{second color name}"
                        BEFORE_PORT = "{first color port}"
                        AFTER_PORT = "{second color port}"
                    }
                    sh "sudo docker run -i -e TZ=Asia/Seoul --name backend-$deployColor -p $A
FTER_PORT:{backend Docker port} -d $imageName-$deployColor:latest"
                }
            }
        }
        stage('Service Check') {
            steps {
                script {
                    sshagent(credentials: ['{Ubuntu Credential}']) {
                        def deployedServerUrl = "http://${deployedServerUrl}:"
                        deployedServerUrl += "$AFTER_PORT/api/actuator/health"

                        for (int i=1; i<=10; i++) {
                            def response = sh(script: "curl -s ${deployedServerUrl}", returnS
tatus: true)

                            if (response == 0) {
                                sh(script: "curl -d '{\"text\":\"back 빌드 성공: (<https://${de
ployedServerUrl}/api/actuator/health|FEBI TEST>)\"}' -H 'Content-Type: application/json' -X P
OST {mattermost url}")
                                break
                            }

                            if (i == 10) {
                                sh(script: "curl -d '{\"text\":\"back 빌드 실패\"}' -H 'Content
-Type: application/json' -X POST {mattermost url}")
                                error "서버가 UP 상태가 되지 않아 빌드 실패로 처리됨"
                            }
                            sleep 5
                        }
                    }
                }
            }
        }
        stage('Before Service Stop & Swap container') {
            steps {
                script {
                    sshagent(credentials: ['{Ubuntu Credential}']) {
                        sh "docker stop backend-${beforeColor}"
                        sh "docker rm backend-${beforeColor}"
                        sh "docker rmi ${imageName}-${beforeColor}"
                        sh "sudo /usr/bin/sed -i 's/${BEFORE_PORT}/${AFTER_PORT}/' $nginxConf
igFile"
                        sh "sudo /usr/sbin/nginx -s reload"
```

```
                }
            }
        }
    }
}
post {
    success {
        script {
            mattermostSend (color: 'good',
            message: "back 빌드 성공"
            )
        }
    }
    failure {
        script {
            mattermostSend (color: 'danger',
            message: "back 빌드 실패"
            )
        }
    }
}
```

Pipeline (Front)

```
pipeline {
    agent any
    tools {nodejs "node.js"}

    environment {
        imageName = "{frontend Image}"
        registryCredential = '{Docker Hub Credential}'
                dockerImage = ''

        releaseServerAccount = '{EC2 username}'
        releaseServerUri = '{EC2 Domain}'

        releasePort = '{front port}'
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: '{frontend branch}',
                    credentialsId: '{gitlab Credential}',
                    url: {GitLab 주소}
                dir('frontend') {
                                        // 프로젝트에 .env를 추가
                    withCredentials([file(credentialsId: '{.env Credential}', variable: 'en
v')]) {
                    script {
                        sh 'cp -f $env ./.env'
                        }
                    }
                }
```

```
            }
        }
        stage('Node Build') {
            steps {
                dir ('frontend') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('frontend') {
                    script {
                        docker.withRegistry('', registryCredential) {
                            sh "docker buildx create --use --name mybuilder"
                            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $im
ageName:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $im
ageName:latest --push ."
                        }
                    }
                }
            }
        }
        stage('Before Service Stop') {
            steps {
                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh '''
                    if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseS
erverUri "docker ps -aq --filter ancestor=$imageName:latest"`"; then
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri
"docker stop $(docker ps -aq --filter ancestor=$imageName:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri
"docker rm -f $(docker ps -aq --filter ancestor=$imageName:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri
"docker rmi $imageName:latest"
                    fi
                    '''
                }
            }
        }
        stage('DockerHub Pull') {
            steps {
                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerU
ri 'sudo docker pull $imageName:latest'"
                }
            }
        }
        stage('Service Start') {
            steps {
                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerU
ri "sudo docker run -i -e TZ=Asia/Seoul --name frontend -p {front local port}:$releasePort -d
$imageName:latest"
                    '''
```

```
                }
            }
        }
    }
    post {
        success {
            script {
                mattermostSend (color: 'good',
                message: "front 빌드 성공"
                )
            }
        }
        failure {
            script {
                mattermostSend (color: 'danger',
                message: "front 빌드 실패"
                )
            }
        }
    }
}
```

📁 Files Ignored

backend

```
### Application.properties ###
**/src/main/resources/*.properties

### generated ###
**/src/main/generated

### application.log ###
**/*.gz
**/*.tmp
```

frontend

```
# dependencies
/node_modules
/.pnp
.pnp.js

# testing
/coverage

# production
/build

# misc
.DS_Store
.env
.env.local
.env.development.local
```

```
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

⚠ Caution

Trouble Shooting

Build Failure

```
[backend]
1. EC2에 conatiner가 실행되었지만 바로 Exited되었는지 확인
    - docker logs {컨테이너 명}을 통해 어떤 Error가 발생되었는지 확인 후 해결
    - docker stop {컨테이너 명}, docker rm {컨테이너 명}, docker rmi {이미지 명}
    - 이후 다시 Build 진행
2. container 및 image가 pull 되어지지 않았는지 확인
    - Jenkins - 해당 Item - Status - Stage View에서 가장 최신 Build 선택 - Console Output
    - Error 원인 확인 후 해결

[frontend]
1. EC2에 conatiner가 새로 실행되었지만 바로 Exited되었는지 확인
    - docker logs {컨테이너 명}을 통해 어떤 Error가 발생되었는지 확인 후 해결
2. 새로 만들어지지 않았는지 확인
    - Jenkins - 해당 Item - Status - Stage View에서 가장 최신 Build 선택 - Console Output
    - Error 원인 확인 후 해결
```

SSE

```
# 만약 EC2 환경에서 SSE 연결이 정상적으로 작동하지 않는다면
sudo vim /etc/nginx/sites-enabled/default
====> 아래 내용 추가
{location 경로} {
    proxy_set_header Connection '';
    proxy_http_version 1.1;
}
```

Redis

```
# 만약 EC2 환경에서 Redis 관련 오류가 발생한다면
1. docker ps -> application.properties에 설정된 port와 일치하는지 확인
2. docker exec -i -t {컨테이너 명} redis-cli -> ping 시 PONG이 나오지 않는다면
3. docker stop {컨테이너 명}, docker rm {컨테이너 명}, 컨테이너 재실행
4. 서버가 정상적이라면 config set requirepass {Redis password}
```

Token

```
# 만약 서비스를 이용하던 중 로그인 혹은 여타 기능이 동작하지 않을 때
1. 로그를 확인하여 Redis와의 연결이 잘 이루어졌는지 확인
2. 만약 Redis 오류가 발생했다면 이동
3. 개발자 도구 - Application - Cookies - {EC2 Domain}에 Authorization, RefreshToken이 존재하는지 확인
4. 존재한다면 삭제 후 재로그인
5. 계속 오류가 발생한다면 캐시 삭제 후 재접속
```

Logging

Monitoring

```
# nginx error.log 추적
tail -f /var/log/nginx/error.log

# nginx access.log 추적
tail -f /var/log/nginx/access.log

# 현재 구동중인 docker container
docker ps

# 모든 docker container 확인
docker ps -a

# 현재 pull 받은 docker image 확인
docker image ls

# docker container log 추적
docker logs -f {컨테이너 명}
```