포팅 매뉴얼

🔎Stacks

Management Tool

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

IDE

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

Infra

`NCP Object Storage` `Nginx(1.18.0)` `Docker(25.0.1)` `Ubuntu(20.04.6)` `EC2`

Frontend

`HTML5` `CSS` `TypeScript(5)` `Next.js(14.1.3)` `React(18)` `React DOM(18)` `Zustand(4.5.2)` `SASS(1.71.1)` `Node.js(20.12.0)` `ESlint(8.57)` `prettier`

Backend

`Java(17)` `SpringBoot(3.2.3)` `JPA` `QueryDSL(5.0.0)` `Redis(7.2.4)` `MySQL(8.3.0)`

Data

`Python(3.9)` `Jupyter notebook` `FastAPI` `Pandas` `Scikit-learn` `SQLAlchmy` `pymysql`

`ElasticSearch(8.12.2)` `Kibana(8.12.2)` `Logstash(8.12.2)` `Zookeeper` `Kafka`

📦Build & Distribute

Backend Build

application.properties

```
server.servlet.context-path={Server context-path}
server.port={Sevrver Port}

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.defer-datasource-initialization=true
#spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

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

spring.datasource.driver-class-name={DB Driver}
spring.datasource.url={DB URL}
spring.datasource.username={DB username}
spring.datasource.password={DB password}

#JWT SETTING
jwt.salt={JWT_SALT}
#Access Token Expiretime
jwt.access-token.expiretime={JWT_ACCESS_TOKEN_EXPIRETIME}
#Refresh Token Expiretime
jwt.refresh-token.expiretime={JWT_REFRESH_TOKEN_EXPIRETIME}

naver.client_id={NAVER_CLIENT_ID}
naver.redirect_uri={NAVER_REDIRECT_URI}
naver.client_secret={NAVER_CLIENT_SECRET}
naver.state={NAVER_STATE}
naver.auth_base_url={NAVER_AUTH_BASE_URL}
naver.info_base_url={NAVER_INFO_BASE_URL}

google.client_id={GOOGLE_CLIENT_ID}
```

```
google.redirect_uri={GOOGLE_REDIRECT_URI}
google.client_secret={GOOGLE_CLIENT_SECRET}
google.auth_base_url={GOOGLE_AUTH_BASE_URL}

kakao.client_id={KAKAO_CLIENT_ID}
kakao.redirect_uri={KAKAO_REDIRECT_URI}
kakao.auth_base_url={KAKAO_AUTH_BASE_URL}

cloud.aws.credentials.access-key={IMAGE_ACCESS_KEY}
cloud.aws.credentials.secret-key={IMAGE_SECRET_KEY}
cloud.aws.region.static={IMAGE_REGION}
cloud.aws.stack.auto=false
cloud.aws.endpoint={IMAGE_ENDPOINT}
cloud.aws.bucket.name={IMAGE_BUCKET}
cloud.aws.folder.custom={IMAGE_CUSTOM}

spring.data.redis.host={REDIS_HOST}
spring.data.redis.port={REDIS_PORT}
spring.data.redis.count.prefix={REDIS_COUNT_PREFIX}
spring.data.redis.match.prefix={REDIS_MATCH_PREFIX}
spring.data.redis.password={REDIS_PASSWORD}

spring.data.kafka.host={KAFKA_HOST}
spring.data.kafka.port={KAFKA_PORT}

spring.elasticsearch.username={ELASTIC_USER}
spring.elasticsearch.password={ELASTIC_PASSWORD}
spring.elasticsearch.host={ELASTIC_URL}
spring.elasticsearch.port={ELASTIC_PORT}

spring.fast.url={FAST_URL}
spring.api.url={API_URL}

google.iss1={GOOGLE_ISS1}
google.iss2={GOOGLE_ISS2}
google.jwks={GOOGLE_JWKS}

kakao.iss={KAKAO_ISS}
```

IDE 및 환경설정

```
1. jdk 17 다운로드 및 환경변수 설정
2. git clone 후 backend/api 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 - ApiApplication으로 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

```
NEXT_PUBLIC_BASE_URL={SERVICE_URL}
NEXT_PUBLIC_OAUTH_GOOGLE_URL={GOOGLE_URL}
NEXT_PUBLIC_OAUTH_NAVER_URL={NAVER_URL}
NEXT_PUBLIC_OAUTH_KAKAO_URL={KAKAO_URL}
```

IDE 및 환경설정

```
1. Node.js 20 이상 다운로드 및 환경변수 설정
2. git clone 후 frontend 폴더를 vscode에서 Open하여 가져오기
3. npm install
4. frontend 폴더 상위 경로에 .env 파일 위치시키기
```

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;"]
```

Nginx.conf

```
server {
    listen 80;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

next.config.js

```
const nextConfig = {
  distDir: 'build',
  output: 'export',
  trailingSlash: true,
};

module.exports = nextConfig;
```

ELK & Kafka Build

.env

```
ELASTIC_VERSION={ElK Version}

ELASTIC_PASSWORD={ElasticSearch Password}

LOGSTASH_INTERNAL_PASSWORD={Logstash Password}

KIBANA_SYSTEM_PASSWORD={Kibana Password}

KAFKA_PORT={Kafka_Port}
KAFKA_SUB_PORT={Kafka_Sub_Port}

KIBANA_PORT={Kibana_Port}

LOGSTASH_PORT={Logstash_Port}

ELASTIC_PORT_1={ElasticSearch_port_1}
ELASTIC_PORT_2={ElasticSearch_port_2}

VIEW_LOG={ElasticSearch Index name}

HEART_LOG={ElasticSearch Index name}

ZOOKEEPER_PORT={Zookeeper_Port}
ZOOKEEPER_CLIENT_PORT={Zookeeper_Client_Port}
ZOOKEEPER_TICK_TIME={Zookeeper_Tick_time}

SERVER_IP={Server_Domain}
```

FastAPI Build

.env

```
HOST={EC2 HOST}
DB_NAME={DB Database name}
PORT={DB Port}
DB_USER={DB username}
PASSSWORD={DB password}
```

⚙️ 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:{port};
```

- client-url.inc 추가

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

- 추가 후 nginx 재시작

MySQL Setting

- MySQL 설치(Docker)

```
sudo docker pull mysql:latest
```

- MySQL Container 실행

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

- MySQL 컨테이너 접속

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

- 데이터베이스 접속

```
mysql -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 redis 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
```

🪫 NCP Setting

Object Storage 설정

- 서비스 신청 및 버킷 생성

```
1. Object Storage > Subscription
2. 상품 이용 신청 클릭
3. Bucket Management > + 버킷 생성 클릭
4. 버킷 이름, 설정, 권한을 입력 후 생성
```

- 액세스 키 발급

```
1. 우측 상단의 본인 클릭 > 계정 관리 클릭
2. 비밀번호(and OTP) 입력
3. 인증키 관리 탭 클릭
4. 신규 API 인증키 생성 클릭
```

💽 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 IPV4 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
```

🎬 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/api') {
                    withCredentials([file(credentialsId: '{application.properties Credentia
l}', variable: 'properties')]) {
                    script {
                        sh 'cp -f $properties ./src/main/resources/application.properties'
                        }
                    }
            }
        }
        stage('Jar Build & Trigger') {
            steps {
                dir ('backend/api') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                    }

                    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}'
                        }
                        echo "${deployColor}"
                    }
            }
        }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('backend/api/') {
                    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 & Deploy') {
            steps {

                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerU
ri 'sudo docker pull $imageName-$deployColor:latest'"
                }

                script {
                    if(deployColor == '{first color name}') {
                        BEFORE_PORT = "{second color port}"
                        AFTER_PORT = "{first color port}"
                    } else {
                        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: (<${deployedSe
rverUrl}|DALKAK TEST>)\"}' -H 'Content-Type: application/json' -X POST {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 {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
im()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).
trim()
                mattermostSend (color: 'good',
                message: "back 빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n[<${env.BUILD_URL}|Jenkins(backend)>]"
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
im()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).
trim()
                mattermostSend (color: 'danger',
                message: "back 빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n[<${env.BUILD_URL}|Jenkins(backend)>]"
                )
            }
        }
    }
}
```

Pipeline (Front)

```
pipeline {
    agent any
    tools {
        nodejs "nodejs"
    }

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

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

        deployColor = ''
        beforeColor = ''
        nginxConfigFile = '/etc/nginx/conf.d/client-url.inc'
```

```
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: '{frontend branch}',
                    credentialsId: '{gitlab Credential}',
                    url: '{GitLab 주소}'
                dir('frontend') {
                    withCredentials([file(credentialsId: '{.env Credential}', variable: 'en
v')]) {
                    script {
                        sh 'cp -f $env ./.env.production'
                        }
                    }
                }
            }
        }
        stage('Node Build & Trigger') {
            steps {
                dir ('frontend') {
                    sh 'npm install'
                    sh 'npm run build'
                }
                script {
                    def containerNames = sh(script: 'docker ps --format \'{{.Names}}\'', retu
rnStdout: true).trim()
                    def targetContainerName = "frontend-{second color name}"
                    def isContainerFound = containerNames.split().contains(targetContainerNam
e)
                    if(isContainerFound) {
                        deployColor = '{first color name}'
                        beforeColor = '{second color name}'
                    } else {
                        deployColor = '{first color name}'
                        beforeColor = '{second color name}'
                    }
                    echo "${deployColor}"
                }
            }
        }
        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 $im
ageName-$deployColor:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platform linux/amd64,linux/arm64 -t $im
ageName-$deployColor:latest --push ."
                        }
                    }
                }
            }
        }
        stage('DockerHub Pull & Deploy') {
            steps {
```

```
                sshagent(credentials: ['{Ubuntu Credential}']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerU
ri 'sudo docker pull $imageName-$deployColor:latest'"
                }

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

                        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\":\"front 빌드 성공: (<${deployedS
erverUrl}|DALKAK TEST>)\"}' -H 'Content-Type: application/json' -X POST {mattermost url}")
                                break
                            }

                            if (i == 10) {
                                sh(script: "curl -d '{\"text\":\"front 빌드 실패\"}' -H 'Conten
t-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 "sudo /usr/bin/sed -i 's/${BEFORE_PORT}/${AFTER_PORT}/' $nginxConf
igFile"
                        sh "sudo /usr/sbin/nginx -s reload"

                        def containers = sh(script: 'docker ps', returnStdout: true).trim()
                        if(containers.contains("frontend-${beforeColor}")) {
                            sh "docker stop frontend-${beforeColor}"
                            sh "docker rm frontend-${beforeColor}"
                            sh "docker rmi ${imageName}-${beforeColor}"
```

```
                        } else {
                            echo "frontend-${beforeColor} 컨테이너가 존재하지 않습니다."
                        }

                    }
                }
            }
        }
    }
    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
im()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).
trim()
                mattermostSend (color: 'good',
                message: "front 빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n[<${env.BUILD_URL}|Jenkins(frontend)>]"
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
im()
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).
trim()
                mattermostSend (color: 'danger',
                message: "front 빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n[<${env.BUILD_URL}|Jenkins(frontend)>]"
                )
            }
        }
    }
}
```

Pipeline (elk & kafka)

```
pipeline {
    agent any

    environment {
        imageName = "{elk Image}"

        releaseServerAccount = '{EC2 username}'
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: '{elk branch}',
                    credentialsId: '{gitlab Credential}',
                    url: '{GitLab 주소}'
                dir('dalkak-elkk') {
```

```
                    withCredentials([file(credentialsId: '{.env Credential}', variable: 'en
v')]) {
                    script {
                        sh 'cp -f $env ./.env'
                        }
                    }
                }
            }
        }
        stage('before docker close') {
            steps {
                sh 'docker stop kafka'
                sh 'docker rm kafka'

                sh 'docker stop zookeeper'
                sh 'docker rm zookeeper'

                sh 'docker stop dalkak-elkk_logstash_1'
                sh 'docker rm dalkak-elkk_logstash_1'

                sh 'docker stop dalkak-elkk_kibana_1'
                sh 'docker rm dalkak-elkk_kibana_1'

                sh 'docker stop dalkak-elkk_elasticsearch_1'
                sh 'docker rm dalkak-elkk_elasticsearch_1'
            }
        }
        stage('Docker deploy') {
            steps {
                dir('dalkak-elkk') {
                    script {
                        sh 'docker-compose build'
                        sh 'docker-compose up -d'
                    }
                }
            }
        }
        stage('Docker setup') {
            steps {
                dir('dalkak-elkk') {
                    script {
                        sh 'sudo chmod +x /var/lib/jenkins/workspace/ELKK/dalkak-elkk/setup/e
ntrypoint.sh'
                        for(int i=0; i<2; i++) {
                            sleep 5
                            sh 'sudo docker-compose up setup'
                        }
                    }
                }
            }
        }
    }
}
```

Pipeline (fastApi)

```
pipeline {
    agent any

    tools {
        gradle 'gradle'
    }

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

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

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

        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('Image Build & DockerHub Push') {
            steps {
                dir('dalkak_fast') {
                    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('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 dalkak-fast -p {fastApi local port}:{fastApi D
ocker port} -d $imageName:latest"
                    '''
                }
            }
        }
    }
}
```

⚠ 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

[ELK + Kafka]
1. EC2에 container들이 실행되었지만, 동작하지 않는 경우
    - setup container가 정상적으로 실행되어서 비밀번호가 입력되었는지 확인
    - logstash와 kafka의 log를 중점적으로 확인
```

```
2. Build되지 않은 경우
    - Jenkins - 해당 Item - Status - Stage View에서 가장 최신 Build 선택 - Console Output - Error 원인 확인 후 해결

[FastAPI]
1. EC2에 conatiner가 새로 실행되었지만 바로 Exited되었는지 확인
    - docker logs {컨테이너 명}을 통해 어떤 Error가 발생되었는지 확인 후 해결
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
```

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
 - 만약 이미 password가 적용되어 있다면 AUTH '{Redis password}' 입력, pass가 나올 시 통과
3. docker stop {컨테이너 명}, docker rm {컨테이너 명}, 컨테이너 재실행
4. 서버가 정상적이라면 config set requirepass {Redis password}
```

ELK

```
# 만약 로그를 출력했지만 elasticsearch에 적용되지 않은 경우
1. docker ps -> docker logs {logstash 컨테이너 명}
    - Error 확인, 없다면 다음 단계
2. docker ps -> docker logs {kafka컨테이너 명}
 - Error 확인, 없다면 다음 단계
3. Kibana Port를 확인하고 접속해서 로그가 입력되었는지 확인
    - 보내는 형식이 logstash에서 정상적으로 정제되는지 확인
    - index의 mapping이 의도한대로 이루어져있는지 확인
```

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}
```

```
# docker container 내부 확인
docker exec -it {컨테이너 명} bash
```