



포팅 메뉴얼

1. 개발 환경 및 기술 스택
2. 설정 파일 및 환경 변수 정보
3. 빌드 및 배포
 - 1) 배포용 서버에 Docker + Docker compose 설치
 - 2) Jenkins 컨테이너 실행
 - 3) docker compose를 통한 실행
 - 5) NGINX 설정
4. 외부 서비스 및 정보
5. DB dump 설명

1. 개발 환경 및 기술 스택

- Frontend
 - React 17.0.0
 - TypeScript 5.2.2
 - Vite 5.2.0
 - Axios 1.6.8
 - ReactQuery
 - Zustand
 - Styled Components
 - StompJs
 - SockJs
 - ChartJs
 - D3
 - FCM
 - ReactSpring

- SonarQube Scanner
- ESLint Config Airbnb
- Prettier
- Backend
 - JAVA 17.0.9 LTS
 - Spring Boot 3.2.4
 - Lombok
 - OpenAPI (Swagger 3.0)
 - OAUTH 2.0
 - Gradle 8.7
 - Spring Security
 - JWT
 - Spring Data JPA
 - QueryDSL
 - WebFlux
 - WebSocket
 - STOMP
 - Kafka
 - Kstreams (Kafka Streams)
 - Firebase Storage
 - FCM
 - Jasypt
 - Java Mail Sender
 - Flyway
 - Geotools
 - Python 3.8.10
 - FastAPI 0.110.1
- DB

- MySQL 8.0.36
- Redis 7.2.4
- MongoDB 4.0.26
- Infra
 - AWS EC2
 - AWS LightSail
 - Ubuntu 20.04.6 LTS
 - Nginx 1.25.5
 - Docker 26.1.0
 - Docker Compose 2.26.1
 - Jenkins 2.458
 - SonarQube 10.5
 - Portainer 2.19.5
- Data
 - Selenium 4.19.0
 - Hadoop 3.2.4
 - Spark 3.2.1
- IDE
 - IntelliJ Ultimate 2023.3.6
 - Vscode 1.85.1
 - WebStorm 2024.1.2

2. 설정 파일 및 환경 변수 정보

1. Spring boot

- application-local.yml (로컬용)

```
server:  
  base-url: localhost  
  fastapi-url: http://localhost:8001
```

```

spring:
  servlet:
    # file 업로드 관련 세팅 (명시적으로 설정 안할 시 Spring boot는
    multipart:
      max-file-size: 10MB # 최대 파일 크기
      max-request-size: 10MB # 최대 요청 크기

  jpa:
    open-in-view: false
    defer-datasource-initialization: false
    generate-ddl: false
    hibernate:
      ddl-auto: none # ddl 자동 작성 여부
    properties:
      hibernate:
        format_sql: true # 하이버네이트가 실행한 SQL
        use_sql_comments: true
        show_sql: true # 하이버네이트가 실행한 SQL
      jdbc:
        batch_size: 100 # insert/update 쿼리
        default_batch_fetch_size: 100

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver # DB 드라이버
    url: jdbc:mysql://${server.base-url}:3306/nowdoboss?useSSL=false
    username: ENC(/u0+12qNJvB7yNP4g9pITQ==)
    password: ENC(8Z02KjT7dEI3sWWMRNGIfg==)

# data_테이블명.sql 관련 실행 setting (더미데이터)
# sql:
#   init:
#     mode: always
#     data-locations:
#       - 'classpath:/db/dummy/FOREIGN_KEY_CHECKS_0.sql'
#       - 'classpath:/db/dummy/db_dump.sql'
#       - 'classpath:/db/dummy/db_dump1.sql'
#       - 'classpath:/db/dummy/db_dump2.sql'

```

```
#          - 'classpath:/db/dummy/FOREIGN_KEY_CHECKS_1.sql'

# NoSQL setting
data:
  # Redis setting
  redis:
    host: ${server.base-url}
    port: 6379

  # MongoDB setting
  mongodb:
    auto-index-creation: true
    host: localhost
    port: 27017
    database: nowdoboss

# Java Mail Sender setting (Google Mail)
mail:
  host: smtp.gmail.com
  port: 587
  username: ENC(9zcJxzMJjdfsV9707BJQvAtUCYHyAV12FmzBkDqp
  password: ENC(bad9dzuo4Yv1McadA8E0Sn13JEE1/6UeyvsknQfC
  properties:
    mail:
      smtp:
        auth: true
        starttls:
          enable: true

# Kafka setting
kafka:
  bootstrap-servers: ${server.base-url}:9092, ${server.b
  consumer:
    group-id: NowDoBossGroup
    auto-offset-reset: earliest
    key-deserializer: org.apache.kafka.common.serializat
    value-deserializer: org.springframework.kafka.support
```

```

        properties:
            spring:
                json:
                    trusted:
                        packages: '*'
        producer:
            key-serializer: org.apache.kafka.common.serialization.
            value-serializer: org.springframework.kafka.support.

# flyway setting
flyway:
    enabled: false # Flyway 비활성화

mvc:
    async:
        request-timeout: 3000000

# log 관리
logging:
    level:
        org.hibernate:
            type.descriptor.sql: trace
            org.hibernate.SQLQuery: debug

# jwt setting
jwt:
    accessKey: ENC(s0vsxAuXm1zr6DjkUVLr1IS+MdTML5ar0QyKWar2q
    refreshKey: ENC(gE+DTAeZ7HD32Rd6u5HcfJu+bg4kgdULucqZWVJC
    accessExpiration: PT420M # 420분 (PT420M)
    refreshExpiration: PT10080M # 10080분 (7일) (PT10080M)

# firebase setting
app:
    firebase-configuration-file: classpath:serviceAccountKey
    firebase-bucket: ENC(kEjE5eNrPFCEldmd4V0t0mXnXI2qkPVNJIn
    firebase-project-id: nowdoboss

# OAUTH2.0 Setting

```

```

oauth:
  kakao:
    client-id: ENC(BXCwf0NtueMIhhv565w/js6mYPVy/ydnEixfuH4
    client-secret: ENC(Vt6Pop34W0MzRbM/vmr0uqASBu2KA78aqSh
    redirect-uri: http://${server.base-url}:5173/member/lo
    scope:
      - profile_nickname
      - profile_image
      - account_email
      - name

  naver:
    client-id: ENC(+VjRqdSmB6m1Jv11ZJGJRW+bTtY4tJNh123J0sH
    client-secret: ENC(0BeyGmHNDHLJW5hudtEfpEpq+0hFiGa0)
    redirect_uri: http://${server.base-url}:5173/member/lo
    scope:
      - nickname
      - name
      - email
      - profile_image

  google:
    client_id: ENC(cDBVfDvLY03EnW9oNtZ9XqgVTS1otEDtR+jbK/4
    client_secret: ENC(FEwnh24Bb7QzhYBqVbWowEN+fJkgt0qVchd
    redirect_uri: http://${server.base-url}:5173/member/lo
    scope:
      - profile
      - email

```

- application-dev.yml (배포용)

```

server:
  base-url: k10c208.p.ssafy.io
  https-url: https://k10c208.p.ssafy.io
  fastapi-url: http://13.124.23.220:8000

spring:
  servlet:

```

```

# file 업로드 관련 세팅 (명시적으로 설정 안할 시 Spring boot는
multipart:
  max-file-size: 10MB # 최대 파일 크기
  max-request-size: 10MB # 최대 요청 크기

jpa:
  open-in-view: false
  defer-datasource-initialization: false # flyway 관련
  generate-ddl: false
  hibernate:
    ddl-auto: none # ddl 자동 작성 여부
  properties:
    hibernate:
      format_sql: true # 하이버네이트가 실행한 SQL
      use_sql_comments: true
      show_sql: true # 하이버네이트가 실행한 SQL
    jdbc:
      batch_size: 100 # insert/update 쿼리
      default_batch_fetch_size: 100

datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver # DB 드라이버
  url: jdbc:mysql://${server.base-url}:3306/nowdoboss?useSSL=false
  username: ENC(CIv9gIlHdev0RVfvISnt1Q==)
  password: ENC(8JDguUr+RtTPfR+4re7EluKRxKLKmaLb)

# NoSQL setting
data:
  # Redis setting
  redis:
    host: ${server.base-url}
    port: 6379

  # MongoDB setting
  mongodb:
    host: ${server.base-url}
    port: 27017
    database: nowdoboss

```



```

# Java Mail Sender setting (Google Mail)
mail:
  host: smtp.gmail.com
  port: 587
  username: ENC(9zcJxzMJjdfsV9707BJQvAtUCYHyAV12FmzBkDqp
  password: ENC(bad9dzuo4Yv1McadA8E0Sn13JEE1/6UeyvsknQfC
  properties:
    mail:
      smtp:
        auth: true
        starttls:
          enable: true

# Kafka setting
kafka:
  bootstrap-servers: kafka-1:29092, kafka-2:29093, kafka
  consumer:
    group-id: NowDoBossGroup
    auto-offset-reset: earliest
    key-deserializer: org.apache.kafka.common.serializat
    value-deserializer: org.springframework.kafka.support
    properties:
      spring:
        json:
          trusted:
            packages: '*'
  producer:
    key-serializer: org.apache.kafka.common.serialization
    value-serializer: org.springframework.kafka.support.

# flyway setting
flyway:
  enabled: true
  locations: classpath:db/migration
  baseline-on-migrate: true
  out-of-order: false

```

```

mvc:
  async:
    request-timeout: 3000000

# log 관리
logging:
  level:
    org.hibernate:
      type.descriptor.sql: trace
      org.hibernate.SQLQuery: debug

# jwt setting
jwt:
  accessKey: ENC(s0vsxAuXm1zr6DjkUVLr1IS+MdTML5ar0QyKWar2q
  refreshKey: ENC(gE+DTAeZ7HD32Rd6u5HcfJu+bg4kgdULucqZWVJC
  accessExpiration: PT420M # 420분 (PT420M)
  refreshExpiration: PT10080M # 10080분 (7일) (PT10080M)

# firebase setting
app:
  firebase-configuration-file: classpath:serviceAccountKey
  firebase-bucket: ENC(kEjE5eNrPFCEldmd4V0t0mXnXI2qkPVNJIn
  firebase-project-id: nowdoboss

# OAUTH2.0 Setting
oauth:
  kakao:
    client-id: ENC(Er9vs53W0nigwpkw3PMICW6PwM0fN0m1xbjy9AI
    client-secret: ENC(15fElLNo9a/0jmRDVVdspIQYQQemm1JLA6Z
    redirect-uri: ${server.https-url}/member/loading/kakao
    scope:
      - profile_nickname
      - profile_image
      - account_email
#      - name

  naver:

```

```

client-id: ENC(+VjRqdSmB6m1Jv11ZJGJRW+bTtY4tJNh123J0sH
client-secret: ENC(0BeyGmHNDHLJW5hudtEfpEpq+0hFiGa0)
redirect-uri: ${server.https-url}/member/loading/naver
scope:
  - nickname
  - name
  - email
  - profile_image

google:
  client_id: ENC(cDBVfDvLY03EnW9oNtZ9XqgVTS1otEDtR+jbK/4
  client_secret: ENC(FEwnh24Bb7QzhYBqVbWowEN+fJkgt0qVchd
  redirect-uri: ${server.https-url}/member/loading/googl
  scope:
    - profile
    - email

```

2. React

- .env(로컬용)

```

# API URL settings for PJT
VITE_REACT_API_URL=http://localhost:8080/api/v1

# 카카오 맵 API 키
VITE_REACT_APP_KAKAOMAP_API_KEY=23e6ac283713cfc6d7967f4616

# 웹소켓 URL 설정
VITE_REACT_WS_URL=ws://localhost:8080/ws

# 클라이언트 기본 URL 설정
BASE_URL=/

# Firebase 설정
VITE_REACT_FIREBASE_API_KEY=AIzaSyB15gX6R0z46ojhJThVJhdNdG
VITE_REACT_FIREBASE_MESSAGING_SENDER_ID=88151658182
VITE_REACT_FIREBASE_APP_ID=1:88151658182:web:12ecd86a23bff
VITE_REACT_FIREBASE_MEASUREMENT_ID=G-6340LC2GSM

```

```
VITE_REACT_FIREBASE_VAPID_KEY=BHyCqnVZHrtKUz3je8uIGip0kfhq
```

- .env-dev(배포용)

```
# API URL 설정
VITE_REACT_API_URL=https://k10c208.p.ssafy.io/api/v1

# 카카오 맵 API 키
VITE_REACT_APP_KAKAOMAP_API_KEY=e55da734c04c78fcc069c3dab6

# 웹소켓 URL 설정
VITE_REACT_WS_URL=wss://k10c208.p.ssafy.io/ws

# 클라이언트 URL 설정
BASE_URL=/

# Firebase 설정
VITE_REACT_FIREBASE_API_KEY=AIzaSyB15gX6R0z46ojhJThVJhdNdg
VITE_REACT_FIREBASE_MESSAGING_SENDER_ID=88151658182
VITE_REACT_FIREBASE_APP_ID=1:88151658182:web:12ecd86a23bff
VITE_REACT_FIREBASE_MEASUREMENT_ID=G-6340LC2GSM

VITE_REACT_FIREBASE_VAPID_KEY=BHyCqnVZHrtKUz3je8uIGip0kfhq
```

3. 빌드 및 배포

1) 배포용 서버에 Docker + Docker compose 설치

1. 배포용 서버에 해당 VI 편집기를 이용하여 쉘 스크립트를 작성한 뒤 실행시킵니다.

install_docker.sh

```
#!/bin/bash

# 기존 도커 패키지 제거 (이전 버전이 설치된 경우)
sudo apt-get remove docker docker-engine docker.io containerd

# 필수 패키지 설치
```

```

sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates c

# 도커 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sud

# 도커 저장소 추가
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-ar

# 도커 설치
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# 도커 컴포즈 최신 버전 다운로드
sudo curl -L "https://github.com/docker/compose/releases/late

# 실행 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 도커 사용자 그룹에 현재 사용자 추가
sudo usermod -aG docker $USER
newgrp docker
sudo service docker restart

# 설치 확인
docker --version
docker-compose --version

```

2) Jenkins 컨테이너 실행

1. 배포용 서버에 해당 VI 편집기를 이용하여 쉘 스크립트를 작성한 뒤 실행시킵니다.
2. 젠킨스가 실행되면 <해당 도메인>:8888로 접속하여 설정을 합니다.

install_jenkins.sh

```

#!/bin/bash

# Jenkins 폴더 생성
JENKINS_DIR="./jenkins"

```

```

if [ ! -d "$JENKINS_DIR" ]; then
    mkdir "$JENKINS_DIR"
fi

# Docker Compose 실행
docker-compose up -d

# Jenkins 컨테이너가 완전히 실행될 때까지 대기
sudo sleep 60

# Jenkins 폴더로 이동
cd ./jenkins

# Jenkins 폴더가 완전히 생성될 때까지 대기
sudo sleep 60

# update center에 필요한 CA 파일 다운로드
UPDATE_CENTER_DIR="./update-center-rootCAs"
if [ ! -d "$UPDATE_CENTER_DIR" ]; then
    mkdir "$UPDATE_CENTER_DIR"
fi

sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-c

# Jenkins 설정 파일 수정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#

# Jenkins 재시작 (필수)
docker restart jenkins

```

3) docker compose를 통한 실행

1. Jenkins를 통해 각종 docker-compose.yml 파일의 컨테이너들을 실행시킵니다.
2. Jenkins 파이프라인을 구성한 뒤 GitLab Webhook 및 Generic Webhook Trigger를 연동하여 개발자가 GitLab에 merge할 때마다 Jenkins 파이프라인이 실행됩니다.
 - 백엔드 서버 (Spring Boot) 관련 배포

SpringBootServer.Dockerfile

```

# OpenJDK 17 이미지를 베이스로 사용
FROM openjdk:17-jdk-slim

# 애플리케이션을 빌드할 소스 코드 및 리소스 복사
COPY . /app

# 작업 디렉토리 설정
WORKDIR /app

# Gradle Wrapper에 실행 권한 부여
RUN chmod +x ./gradlew

# Spring Boot 애플리케이션 빌드
RUN ./gradlew clean bootJar

# JAR 파일을 /app 디렉토리로 복사
RUN cp build/libs/*.jar /app/app.jar

RUN mkdir /app/videos

# Spring Boot 애플리케이션 실행을 위한 명령 설정
ENTRYPOINT ["java", "-Dspring.profiles.active=dev", "-jar", ".

```

docker-compose-springboot.yml

```

version: "3.8" # Docker Compose 파일 버전을 지정합니다. "3.8"은 사

services:
  nowdoboss_backend_springboot:
    container_name: nowdoboss_backend_springboot # 컨테이너의
    build: # 도커 이미지 빌드 관련 설정입니다.
      context: . # Dockerfile이 위치한 디렉토리 경로입니다. (현재 d
      dockerfile: SpringBootServer.Dockerfile # 사용할 Dockerf
    image: nowdoboss_backend_springboot_img # 빌드된 이미지의 이
    restart: always # 컨테이너가 항상 재시작되도록 설정합니다.
    ports:
      - "8080:8080" # 호스트의 8080 포트를 컨테이너의 8080 포트에
    environment:

```

```

- TZ=Asia/Seoul # 해당 백엔드 서버의 시간을 아시아/서울로 설정함
- jasypt.encryptor.key=ssafy # 환경 변수로 jasypt 암호화 키
networks:
- nowdoboss_net # 사용할 네트워크를 지정합니다.

```

```

networks:
  nowdoboss_net: # 사용할 네트워크를 정의합니다.
    name: nowdoboss_net
    driver: bridge

```

Jenkinsfile-springBootServer

```

pipeline {
  agent any // 이 파이프라인이 실행될 Jenkins 에이전트를 지정합니다
  triggers {
    GenericTrigger(
      genericVariables: [
        [key: 'USER_NAME', value: '$.user.name', expression: true],
        [key: 'IF_MERGED', value: '$.object_attribute', expression: true],
        [key: 'BASE_BRANCH', value: '$.object_attribute', expression: true],
        [key: 'LABEL', value: '$.labels[*].title', expression: true]
      ],
      causeString: 'Triggered by GitLab Merge Request by',
      token: 'backend-springboot',
      printContributedVariables: true,
      printPostContent: true,
      regexpFilterText: '$IF_MERGED $BASE_BRANCH $LABEL',
      regexpFilterExpression: '(?=.*merged)(?=.*develop)'
    )
  }

  stages {
    stage('Setup') {
      steps {
        script {
          // 빌드 설명 설정
          currentBuild.description = "Merge request"
        }
      }
    }
  }
}

```



```

COPY package*.json ./

# 종속성 설치
RUN npm install

# 나머지 애플리케이션 코드 복사
COPY . .

# .env 파일 변경
COPY .env-dev .env

# 프론트엔드 코드 빌드
RUN npm run build

```

Nginx.Dockerfile

```

# nginx의 최신 이미지를 기반으로 합니다.
FROM nginx:latest

# nginx 설정 파일을 컨테이너의 적절한 위치에 복사합니다.
COPY nginx.conf /etc/nginx/conf.d/default.conf

```

docker-compose-frontend.yml

```

version: "3.8" # Docker Compose 파일 버전을 지정합니다. "3.8"은 사

services:
  nowdoboss_react:
    container_name: nowdoboss_react # 컨테이너의 이름을 설정합니다
    build: # 도커 이미지 빌드 관련 설정입니다.
      context: . # Dockerfile이 위치한 디렉토리 경로입니다.
      dockerfile: ReactServer.Dockerfile # 사용할 Dockerfile의
    image: nowdoboss_frontend_img # 빌드된 이미지의 이름을 설정합니
    volumes:
      - react_build:/usr/src/app/dist # 빌드 결과물을 볼륨에 저장
    entrypoint: ["true"] # Docker 이미지가 빌드되고 볼륨이 준비된 :

  nowdoboss_frontend:

```

```

    container_name: nowdoboss_frontend
    build:
      context: ../CICD/Nginx
      dockerfile: Nginx.Dockerfile
    volumes:
      - react_build:/usr/share/nginx/html # Nginx가 빌드된 정적
      - ./certbot/conf:/etc/letsencrypt
      - ./certbot/www:/var/www/certbot
    ports:
      - "80:80"
      - "443:443"
    networks:
      - nowdoboss_net
    depends_on:
      - nowdoboss_react
    # command: "nginx -g 'daemon off;'" # Nginx를 포그라운드 모
    # command: "/bin/sh -c 'while ;; do sleep 6h & wait $${!}'"

nowdoboss_certbot:
  container_name: nowdoboss_certbot
  image: certbot/certbot
  volumes:
    - ./certbot/conf:/etc/letsencrypt
    - ./certbot/www:/var/www/certbot
  # entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do ce
  entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do cert
  depends_on:
    - nowdoboss_frontend

volumes:
  react_build: # 공유 볼륨 선언

networks:
  nowdoboss_net: # 사용할 네트워크를 정의합니다.
    name: nowdoboss_net
    driver: bridge

```

Jenkinsfile-ReactServer

```

pipeline {
    agent any // 이 파이프라인이 실행될 Jenkins 에이전트를 지정합니다
    triggers {
        GenericTrigger(
            genericVariables: [
                [key: 'USER_NAME', value: '$.user.name', expression: true],
                [key: 'IF_MERGED', value: '$.object_attribute', expression: true],
                [key: 'BASE_BRANCH', value: '$.object_attribute', expression: true],
                [key: 'LABEL', value: '$.labels[*].title', expression: true]
            ],
            causeString: 'Triggered by GitLab Merge Request by',
            token: 'frontend',
            printContributedVariables: true,
            printPostContent: true,
            regexpFilterText: '$IF_MERGED $BASE_BRANCH $LABEL',
            regexpFilterExpression: '(?=.*merged)(?=.*develop'
        )
    }
    tools {
        nodejs '20.11.1'
    }

    stages {
        stage('Setup') {
            steps {
                script {
                    // 빌드 설명 설정
                    currentBuild.description = "Merge request"
                }
            }
        }

        stage('Deploy with Docker Compose') { // 'Deploy with
            steps {
                script {
                    echo "certbot 컨테이너 실행 상태 확인 중..."
                    def isCertBotRunning = sh(script: "docker
                    echo "certbot 실행 상태: ${isCertBotRunning

```

```

        if (isCertBotRunning == "") {
            echo "Certbot dockerfile 빌드..."
            sh "docker-compose -f ./FrontEnd/docker-c
        }

        // React Nginx 컨테이너를 종료하고 관련 볼륨을
        sh "docker-compose -f ./FrontEnd/docker-c
        // React Nginx 컨테이너를 재빌드하고 백그라운드0
        sh "docker-compose -f ./FrontEnd/docker-c

        // React 컨테이너가 빌드 작업을 완료한 후에는 종
        sh "docker-compose -f ./FrontEnd/docker-c
    }
}

stage('Cleanup') {
    steps {
        script {
            // nowdoboss_react 컨테이너를 종료하고 삭제합니
            sh "docker stop nowdoboss_react || true"
            sh "docker rm nowdoboss_react || true" //
        }
    }
}

stage('SonarQube Analysis - ReactServer') {
    steps {
        dir('FrontEnd') {
            withSonarQubeEnv('SonarQube Server') {
                sh 'npm install'
                sh 'npm run sonarqube'
            }
        }
    }
}
}

```

- 백엔드 서버 (FastAPI) 배포용

FastApiServer.Dockerfile

```
# 공식 Python 런타임 이미지를 사용합니다
FROM python:3.10-slim

# 설치할 수 있는 Java 버전 확인
RUN apt-get update && apt-cache search openjdk

# Java와 필요한 도구를 설치합니다
RUN apt-get update && apt-get install -y \
    openjdk-17-jdk \
    procps \
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# JAVA_HOME 환경 변수 설정
ENV JAVA_HOME /usr/lib/jvm/java-17-openjdk-amd64

# 컨테이너 내에서 작업 디렉토리를 설정합니다
WORKDIR /app

# 현재 디렉토리의 내용을 컨테이너 내의 /app 디렉토리로 복사합니다
COPY . /app

# requirements.txt에 명시된 필요한 패키지를 설치합니다
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션을 실행하는 명령어를 정의합니다
CMD ["uvicorn", "server:app", "--host", "0.0.0.0", "--port",
```

FastApiServer.Dockerfile

```
# 공식 Python 런타임 이미지를 사용합니다
FROM python:3.10-slim

# 설치할 수 있는 Java 버전 확인
```

```

RUN apt-get update && apt-cache search openjdk

# Java와 필요한 도구를 설치합니다
RUN apt-get update && apt-get install -y \
    openjdk-17-jdk \
    procps \
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# JAVA_HOME 환경 변수 설정
ENV JAVA_HOME /usr/lib/jvm/java-17-openjdk-amd64

# 컨테이너 내에서 작업 디렉토리를 설정합니다
WORKDIR /app

# 현재 디렉토리의 내용을 컨테이너 내의 /app 디렉토리로 복사합니다
COPY . /app

# requirements.txt에 명시된 필요한 패키지를 설치합니다
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션을 실행하는 명령어를 정의합니다
CMD ["uvicorn", "server:app", "--host", "0.0.0.0", "--port",

```

docker-compose-fastapi.yml

```

version: "3.8" # Docker Compose 파일 버전을 지정합니다. "3.8"은 사

services:
  nowdoboss_backend_fastapi:
    container_name: nowdoboss_backend_fastapi # 컨테이너의 이름
    build: # 도커 이미지 빌드 관련 설정입니다.
      context: . # Dockerfile이 위치한 디렉토리 경로입니다. (현재 d
      dockerfile: FastApiServer.Dockerfile # 사용할 Dockerfile
    image: nowdoboss_backend_fastapi_img # 빌드된 이미지의 이름을
    restart: always # 컨테이너가 항상 재시작되도록 설정합니다.
    ports:

```

```

- "8000:8000" # 호스트의 8000 포트를 컨테이너의 8000 포트에 매핑
networks:
- nowdoboss_data_net # 사용할 네트워크를 지정합니다.

```

```

networks:
  nowdoboss_data_net: # 사용할 네트워크를 정의합니다. 여기서는 기본

```

Jenkinsfile-FastApiServer

```

pipeline {
  agent any // 이 파이프라인이 실행될 Jenkins 에이전트를 지정합니다
  triggers {
    GenericTrigger(
      genericVariables: [
        [key: 'USER_NAME', value: '$.user.name', expression: true],
        [key: 'IF_MERGED', value: '$.object_attribute', expression: true],
        [key: 'BASE_BRANCH', value: '$.object_attribute', expression: true],
        [key: 'LABEL', value: '$.labels[*].title', expression: true]
      ],
      causeString: 'Triggered by GitLab Merge Request by',
      token: 'backend-fastapi',
      printContributedVariables: true,
      printPostContent: true,
      regexpFilterText: '$IF_MERGED $BASE_BRANCH $LABEL',
      regexpFilterExpression: '(?=.merged)(?=.develop)'
    )
  }

  stages {
    stage('Setup') {
      steps {
        script {
          // 빌드 설명 설정
          currentBuild.description = "Merge request"
        }
      }
    }
  }
}

```



```

        stage('Deploy with Docker Compose') { // 'Deploy with Docker Compose'
            steps {
                script {
                    // 이전 실행에서 사용된 컨테이너 및 네트워크 정리
                    sh "docker-compose -f ./BackEnd/FastApiSe

                    // 새로운 푸시에 대한 스크립트 실행
                    sh "docker-compose -f ./BackEnd/FastApiSe
                }
            }
        }
    }
}

```

- 데이터 프로세싱 관련 (Hadoop + Spark) 배포용

hd-spark-base.Dockerfile

```

# 우분투 베이스 이미지 사용
FROM ubuntu:latest

# 필수 패키지 설치
RUN apt-get update && apt-get upgrade -y
RUN apt-get install -y curl openssh-server rsync wget vim ipu

# python 명령어를 python3로 링크
RUN ln -s /usr/bin/python3 /usr/bin/python

# 하둡 다운로드 및 설치
RUN wget http://mirror.navercorp.com/apache/hadoop/common/had
    && tar zxvf hadoop-3.2.4.tar.gz \
    && rm hadoop-3.2.4.tar.gz \
    && mv hadoop-3.2.4 /usr/local/hadoop

# Spark 다운로드 및 설치
RUN wget https://archive.apache.org/dist/spark/spark-3.2.1/sp
    && tar zxvf spark-3.2.1-bin-hadoop3.2.tgz \
    && rm spark-3.2.1-bin-hadoop3.2.tgz \

```

```

    && mv spark-3.2.1-bin-hadoop3.2 /usr/local/spark

# 가상 환경 생성 및 활성화, PySpark 및 필요 라이브러리 설치
RUN python3 -m venv /opt/venv
RUN /opt/venv/bin/pip install --upgrade pip
RUN /opt/venv/bin/pip install pyspark

# 환경변수 설정
ENV JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
ENV HADOOP_HOME=/usr/local/hadoop
ENV SPARK_HOME=/usr/local/spark
ENV PYSPARK_PYTHON=/opt/venv/bin/python
ENV PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$JAVA_HOME/

# 디렉토리 생성
RUN mkdir -p /usr/local/bin /usr/local/bin/master /usr/local/

# 공통적으로 사용되는 하둡 설정 파일(.xml)과 및 쉘 스크립트 복사
COPY hadoop/common/core-site.xml $HADOOP_HOME/etc/hadoop/core
COPY hadoop/common/mapred-site.xml $HADOOP_HOME/etc/hadoop/ma
COPY hadoop/common/yarn-site.xml $HADOOP_HOME/etc/hadoop/yarn
COPY hadoop/common/setup-hadoop.sh /usr/local/bin/setup-hadoo
COPY hadoop/common/init-ssh-keys.sh /usr/local/bin/init-ssh-k
COPY hadoop/common/collect-ssh-keys.sh /usr/local/bin/collect
COPY hadoop/common/update-hosts.sh /usr/local/bin/update-host

# 하둡 마스터 노드 설정 파일 및 스크립트 복사
COPY hadoop/master/hdfs-site.xml /usr/local/bin/master/hdfs-s
COPY hadoop/master/setup-master-hadoop-env.sh /usr/local/bin/

# 하둡 워커 노드 설정 파일 및 스크립트 복사
COPY hadoop/worker/hdfs-site.xml /usr/local/bin/worker/hdfs-s
COPY hadoop/worker/setup-worker-hadoop-env.sh /usr/local/bin/

# 각 노드내에 스파크 설정 파일 및 스파크 관련 쉘 스크립트 복사
COPY spark/spark-env.sh $SPARK_HOME/conf/spark-env.sh
COPY spark/spark-defaults.conf $SPARK_HOME/conf/spark-default
COPY spark/start-master.sh /usr/local/bin/start-master.sh

```

```

COPY spark/start-slave.sh /usr/local/bin/start-slave.sh
COPY spark/start-history-server.sh /usr/local/bin/start-histo
COPY spark/create-hdfs-log-dir.sh /usr/local/bin/create-hdfs-

# 쉘 스크립트 실행 권한 부여 및 실행
RUN chmod +x /usr/local/bin/setup-hadoop.sh
RUN chmod +x /usr/local/bin/init-ssh-keys.sh /usr/local/bin/c
RUN chmod +x /usr/local/bin/update-hosts.sh
RUN chmod +x /usr/local/bin/start-master.sh
RUN chmod +x /usr/local/bin/start-slave.sh
RUN chmod +x /usr/local/bin/start-history-server.sh
RUN chmod +x /usr/local/bin/create-hdfs-log-dir.sh
RUN chmod +x /usr/local/bin/master/setup-master-hadoop-env.sh
RUN chmod +x /usr/local/bin/worker/setup-worker-hadoop-env.sh

# SSH 구성
RUN service ssh start

# 포트 설정
EXPOSE 9870 8088 19888 7077 8080 18080

# SSH 데몬 실행
CMD ["/usr/sbin/sshd", "-D"]

```

docker-compose-hadoopSpark.yml

```

version: "3.8" # Docker Compose 파일 버전을 지정합니다. "3.8"은
services:
  master1:
    build:
      context: .
      dockerfile: hd-spark-base.Dockerfile
    container_name: master1
    hostname: master1
    ports:
      - "9870:9870" # Hadoop NameNode Web UI

```

```

- "8088:8088"      # Hadoop ResourceManager Web UI
- "19888:19888"    # Hadoop MapReduce JobHistory Server W
- "7077:7077"      # Spark Master Port
- "8870:8080"      # Spark Master Web UI
- "18080:18080"    # Spark History Server

```

volumes:

```
- shared_keys:/shared_keys
```

networks:

```
nowdoboss_data_net: # 사용할 네트워크를 지정합니다.
```

```
  ipv4_address: 172.24.48.100
```

command: >

```

/bin/bash -c "
/usr/local/bin/setup-hadoop.sh &&
/usr/local/bin/update-hosts.sh &&
/usr/local/bin/init-ssh-keys.sh &&
/usr/local/bin/collect-ssh-keys.sh 3 &&
/usr/local/bin/master/setup-master-hadoop-env.sh &&
/usr/local/bin/create-hdfs-log-dir.sh &&
/usr/local/bin/start-master.sh &&
/usr/local/bin/start-history-server.sh
"

```

worker1:

build:

```
context: .
```

```
dockerfile: hd-spark-base.Dockerfile
```

container_name: worker1

hostname: worker1

volumes:

```
- shared_keys:/shared_keys
```

networks:

```
nowdoboss_data_net: # 사용할 네트워크를 지정합니다.
```

```
  ipv4_address: 172.24.48.101
```

command: >

```

/bin/bash -c "
/usr/local/bin/setup-hadoop.sh &&
/usr/local/bin/update-hosts.sh &&
/usr/local/bin/init-ssh-keys.sh &&

```

```

        /usr/local/bin/collect-ssh-keys.sh 5 &&
        /usr/local/bin/worker/setup-worker-hadoop-env.sh &&
        /usr/local/bin/start-slave.sh
    "

worker2:
  build:
    context: .
    dockerfile: hd-spark-base.Dockerfile
  container_name: worker2
  hostname: worker2
  volumes:
    - shared_keys:/shared_keys
  networks:
    nowdoboss_data_net: # 사용할 네트워크를 지정합니다.
      ipv4_address: 172.24.48.102
  command: >
    /bin/bash -c "
    /usr/local/bin/setup-hadoop.sh &&
    /usr/local/bin/update-hosts.sh &&
    /usr/local/bin/init-ssh-keys.sh &&
    /usr/local/bin/collect-ssh-keys.sh 7 &&
    /usr/local/bin/worker/setup-worker-hadoop-env.sh &&
    /usr/local/bin/start-slave.sh
    "

networks:
  nowdoboss_data_net: # 사용할 네트워크를 정의합니다.
    name: nowdoboss_data_net
    driver: bridge
    ipam:
      config:
        - subnet: 172.24.48.0/20

volumes:
  shared_keys:

```

Jenkinsfile-DataProcessing

```

pipeline {
    agent any // 이 파이프라인이 실행될 Jenkins 에이전트를 지정합니다
    triggers {
        GenericTrigger(
            genericVariables: [
                [key: 'USER_NAME', value: '$.user.name', expr
                [key: 'IF_MERGED', value: '$.object_attribute
                [key: 'BASE_BRANCH', value: '$.object_attribu
                [key: 'LABEL', value: '$.labels[*].title', ex
            ],
            causeString: 'Triggered by GitLab Merge Request b
            token: 'dataProcessing',
            printContributedVariables: true,
            printPostContent: true,
            regexpFilterText: '$IF_MERGED $BASE_BRANCH $LABEL
            regexpFilterExpression: '(?=.merged)(?=.develop
        )
    }

    stages {
        stage('Setup') {
            steps {
                script {
                    // 빌드 설명 설정
                    currentBuild.description = "Merge request
                }
            }
        }

        stage('Build and Deploy Hadoop And Spark') {
            steps {
                script {
                    echo "하둡 + 스파크 (masater1) 컨테이너 구성 실행
                    def isHadoopRunning_master1 = sh(script:
                    echo "Kafka-1 실행 상태: ${isHadoopRunning_

                    echo "하둡 + 스파크 (worker1) 컨테이너 구성 실행
                    def isHadoopRunning_worker1 = sh(script:

```

```
echo "Kafka-1 실행 상태: ${isHadoopRunning_}"

echo "하둡 + 스파크 (worker2) 컨테이너 구성 실행"
def isHadoopRunning_worker2 = sh(script:
echo "Kafka-1 실행 상태: ${isHadoopRunning_}

if (isHadoopRunning_master1 == "" || isHa
    echo "하둡 + 스파크 dockerfile 빌드..."
    sh "docker-compose -f ./CICD/DataProc
}

}

}

}
```

5) NGINX 설정

nginx.conf

```
# HTTP 서버 설정
# server {
#     listen 80;
#     server_name k10c208.p.ssafy.io;

#     location /.well-known/acme-challenge/ {
#         root /var/www/certbot;
#         try_files $uri $uri/ =404;
#     }

#     # 프론트엔드 설정
#     location / {
#         root /usr/share/nginx/html;
#         index index.html;
#         try_files $uri $uri/ /index.html;
#     }

#     # 백엔드 프록시 설정
#     location /api {
```

```

#         proxy_pass http://k10c208.p.ssafy.io:8080;
#         proxy_set_header Host $host;
#         proxy_set_header X-Real-IP $remote_addr;
#         proxy_set_header X-Forwarded-For $proxy_add_x_forwa
#         proxy_set_header X-Forwarded-Proto $scheme;
#     }
# }

# HTTP 리다이렉션을 HTTPS로
server {
    listen 80;
    server_name k10c208.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
        try_files $uri $uri/ =404;
    }

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

# HTTPS 서버 설정
server {
    listen 443 ssl;
    server_name k10c208.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/k10c208.p.ssafy.io/
    ssl_certificate_key /etc/letsencrypt/live/k10c208.p.ssafy

    # 프론트엔드
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    # 백엔드 API 프록시
    location /api {

```



```

        proxy_pass http://k10c208.p.ssafy.io:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # 백엔드 웹소켓 프록시
    location /ws {
        proxy_pass http://k10c208.p.ssafy.io:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}

```

4. 외부 서비스 및 정보

- 카카오 소셜 로그인 정보



동의항목

카카오 로그인으로 서비스를 시작할 때 동의받는 항목을 설정합니다. 미리보기를 통해 사용자에게 보여질 화면을 확인할 수 있습니다.
권한이 필요한 동의항목은 심사 신청 후 사용할 수 있습니다.

개인정보 동의항목 심사 신청

개인정보

| 항목 이름 | ID | 상태 | |
|------------|------------------|--------------|----|
| 닉네임 | profile_nickname | ● 필수 동의 | 설정 |
| 프로필 사진 | profile_image | ● 필수 동의 | 설정 |
| 카카오계정(이메일) | account_email | ● 필수 동의 [수집] | 설정 |

• 네이버 소셜 로그인 정보

애플리케이션 이름

NowDoBoss

- 네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요.
- 40자 이내의 영문, 한글, 숫자, 공백문자, 쉼표(,), "/" , "-" , "_" , 만 입력 가능합니다.

카테고리

생산성

사용 API

선택하세요.

네이버 로그인

제공 정보 선택(이용자 식별자는 기본 정보로 제공)

필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보만을 선택해야 합니다.

| 권한 | 필수 | 추가 |
|------------|-------------------------------------|--------------------------|
| 회원이름 | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 연락처 이메일 주소 | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 별명 | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 프로필 사진 | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 성별 | <input type="checkbox"/> | <input type="checkbox"/> |
| 생일 | <input type="checkbox"/> | <input type="checkbox"/> |
| 연령대 | <input type="checkbox"/> | <input type="checkbox"/> |
| 출생연도 | <input type="checkbox"/> | <input type="checkbox"/> |
| 휴대전화번호 | <input type="checkbox"/> | <input type="checkbox"/> |

- 구글 소셜 로그인 정보

Google Cloud

NowDoBoss

리소스, 문서, 제품 등 검색(/)

검색

사용자 인증 정보

+

사용자 인증 정보 만들기

삭제

↶

삭제된 사용자 인증 정보 복원

사용 설정한 API에 액세스하려면 사용자 인증 정보를 만드세요. 자세히 알아보기

API 키

이름

생성일

재만사항

작업

표시할 API 키가 없습니다.

OAuth 2.0 클라이언트 ID

이름

생성일

유형

클라이언트 ID

작업

NowDoBoss Social Login (OAUTH 2.0)

2024. 4. 18.

웹 애플리케이션

758022711678-1k3m...

서비스 계정

이메일

이름

작업

표시할 서비스 계정이 없습니다.

포팅 메뉴얼

35

- 웹 애플리케이션의 클라이언트 ID

 삭제

이름 *

NowDoBoss Social Login (OAUTH 2.0)

OAuth 2.0 클라이언트의 이름입니다. 이 이름은 콘솔에서 클라이언트를 식별하는 용도로만 사용되며 최종 사용자에게 표시되지 않습니다.



아래에 추가한 URI의 도메인이 [승인된 도메인](#)으로 [OAuth 동의 화면](#)에 자동으로 추가됩니다.

5. DB dump 설명

- 개발 초기 ~ 중기 단계에서는 JPA를 이용하여 테이블들을 자동으로 DB와 매핑하여 Spring boot 서버 내에서 자동으로 생성 되게끔 설정하였습니다.
- 개발 중기 단계에서는 해당 테이블 관련 설정 및 DB 덤프 데이터를 init하여 서버 실행시 자동으로 데이터가 생성되게끔 하였습니다.
- 참고사항으로 DB 덤프 데이터 init시 양이 많기 때문에 파일을 나눠서 설정하였습니다.
 - 사용된 DB dump 파일
(BackEnd/SpringBootServer/src/main/resources/db/dummy 폴더내에 저장)
 - db_dump.sql
 - db_dump1.sql
 - db_dump2.sql
 - FOREIGN_KEY_CHEKCS_0.sql (외래키 제약 해제)
 - FOREIGN_KEY_CHEKCS_1.sql (외래키 제약 설정)
- 개발 후기 단계 및 프로덕션 상태에서는 JPA를 이용한 DDL 자동 생성 및 삭제를 사용하지 않고 flyway를 사용한 DB 마이그레이션 작업을 실시하였습니다.
 - 사용된 DB dump 파일
(BackEnd/SpringBootServer/src/main/resources/db/migration 폴더내에

저장)

- V1__Initial_schema.sql (배포용 초기 DB 테이블 및 연관관계 세팅)
- V2__Add_stratup_support_table.sql (창업지원 관련 테이블 추가)
- V3__Add_share_table.sql (카카오톡 공유하기 관련 테이블 추가)
- 프로젝트 exec 폴더내에 배포서버에 적용한 db_dump 파일 전체가 있습니다.