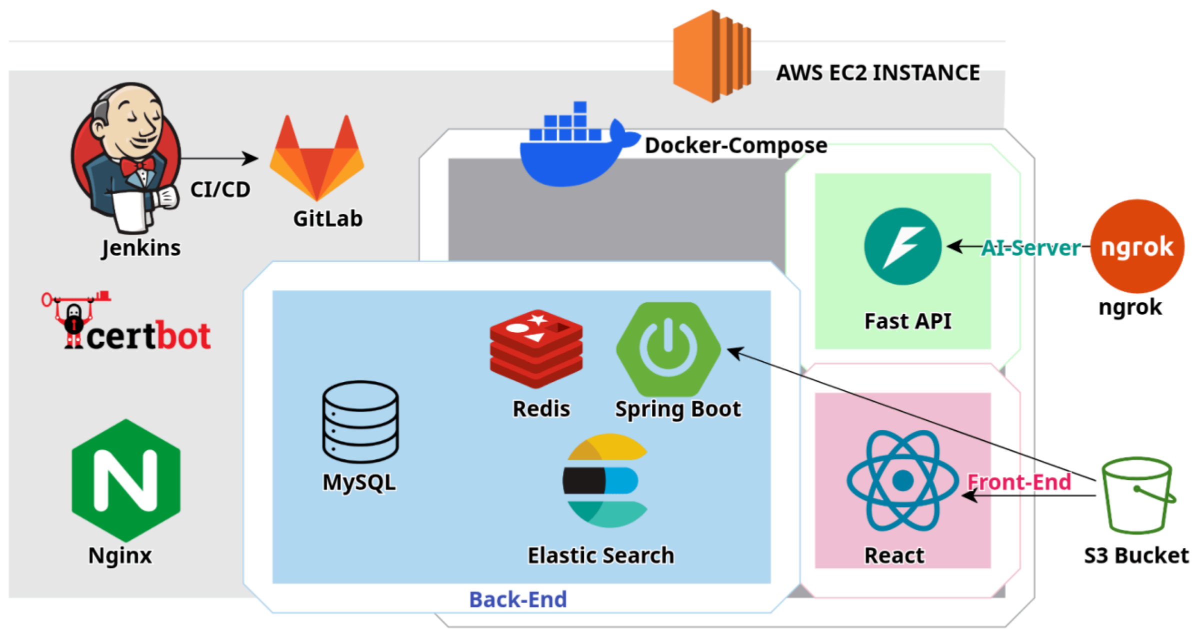
포팅 매뉴얼

포팅 매뉴얼

1. 기술 스택



Frontend

- **Framework**: React 19.1.0
- **Build Tool**: Vite 7.0.4
- **Package Manager**: npm
- **Language**: JavaScript
- **Additional Libraries**:
    - React Router DOM 7.7.0
    - Axios 1.11.0
    - Lucide React 0.525.0
    - JWT Decode 4.0.0

Backend

- **Framework**: Spring Boot 3.5.3
- **Language**: Java 21
- **Build Tool**: Gradle 8.5
- **Database**: MySQL
- **Cache**: Redis 7
- **Search Engine**: Elasticsearch 8.15.0
- **Additional Dependencies**:
    - Spring Data JPA
    - Spring Security
    - Spring Data Redis
    - Spring Data Elasticsearch
    - JWT (0.11.5)
    - AWS S3 Integration
    - SpringDoc OpenAPI

AI

- **Framework**: FastAPI
- **Language**: Python 3.11
- **Container**: Docker
- **AI Libraries**:
    - FastAPI >= 0.100.0
    - Uvicorn (ASGI server)
    - Sentence Transformers >= 2.0.0
    - Scikit-learn >= 1.2.0
    - PyTorch >= 1.9.0
    - Transformers >= 4.21.0
    - Pydantic >= 2.0.0

CI/CD

- **Jenkins**: Pipeline 기반 자동 배포
- **Docker**: 컨테이너화
- **Docker Compose**: 서비스 오케스트레이션

2. 환경 변수

.env 파일 설정

```
# JWT 시크릿 키
JWT_SECRET=MXRrYXRqZGZrZGxkaHM3d21kbnRtZGdrcmxmM3ZsZmZuY
XZsZmZuYTIwOA

# AWS 설정
AWS_ACCESS_KEY_ID=AKIA4WDKGAMDLEONOS4L
AWS_SECRET_ACCESS_KEY=RS+Ifz+aLXgGnwnrWVwN9hjifAI/Of923qrRD
QKh
AWS_REGION=ap-northeast-2
```

Docker Compose 환경 변수

- `DATA_PATH=/app/data`
- `TRANSFORMERS_CACHE=/app/models`
- `HF_HOME=/app/models`
- `SPRING_PROFILES_ACTIVE=prod`

3. 설정 파일

Backend (application.yml)

- **데이터베이스**: MySQL (i13D208.p.ssafy.io:3306)
- **Redis**: movie-redis:6379
- **Elasticsearch**: local-elasticsearch:9200
- **FastAPI**: http://movie-fastapi:8000
- **메일 서버**: Gmail SMTP
- **S3 버킷**: feelroom-1

- **API Keys**: KOBIS, TMDB

```
# src/main/resources/application.yml

spring:
  datasource:
    url: jdbc:mysql://i13D208.p.ssafy.io:3306/movie_recommendation_db?us
eSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF8
    username: root # 운영 db 계정
    password: gongtong208gumi
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: none # 애플리케이션 시작 시 스키마 변경을 하지 않음 (테이블 수동
생성 필수)
    show-sql: true # 실행되는 SQL 쿼리를 콘솔에 표시
    properties:
      hibernate:
        format_sql: true # SQL 쿼리 포맷팅
    open-in-view: false # Lazy 로딩 관련 설정, 성능을 위해 false 권장
  data:
    redis:
      url: redis://${SPRING_REDIS_HOST:localhost}:${SPRING_REDIS_PORT:6
379}
    elasticsearch:
      repositories:
        enabled: true
  elasticsearch:
    uris: http://${ELASTICSEARCH_HOST:localhost}:${ELASTICSEARCH_PO
RT:9200}
    connection-timeout: 5s
    socket-timeout: 30s
  jackson:
    serialization:
      write-dates-as-timestamps: false
    deserialization:
      fail-on-unknown-properties: false
    time-zone: Asia/Seoul
  mail:
    host: smtp.gmail.com  # Gmail SMTP 서버
    port: 587
    username: heyfeelroom@gmail.com  # 발송용 Gmail 주소
    password: mphidnjprvsgwenv  # Gmail 앱 비밀번호 (2단계 인증 필요)
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
            required: true
          connectiontimeout: 5000
          timeout: 3000
          writetimeout: 5000

s3:
  bucket:
    name: feelroom-1

kobis:
  api:
    key: b2cbd6969cf7b74aafd33ee81bf134e5
    base-url: http://www.kobis.or.kr/kobisopenapi/webservice/rest/boxoffic
e/searchDailyBoxOfficeList.json

tmdb:
  api:
    key: 3c3356b9a0daa1dbb9e21cda8b7b5d30 # 본인의 TMDB API 키로 변경
    base-url: https://api.themoviedb.org/3
    call-delay-ms: 200 # 각 API 호출 간 지연 시간 (밀리초), 예: 200ms
  image:
    base-url: https://image.tmdb.org/t/p/original # 포스터 이미지의 기본 URL
jwt:
  secret:
    key: MXRrYXRqZGZrZGxkaHM3d21kbnRtZGdrcmxmM3ZsZmZuYXZsZmZ
uYTIwOA # 256비트 이상의 시크릿 키
```

```
  token:
    access-expiration-time: 3600000
logging:
  level:
    org.hibernate.SQL: debug # Hibernate가 생성하는 SQL 쿼리 로그
    com.d208.feelroom: INFO # 개발자 정의 패키지의 로깅 레벨
    org.springframework.jdbc.core.JdbcTemplate: DEBUG
    org.springframework.jdbc.core.StatementCreatorUtils: DEBUG
    com.d208.feelroom.service.JsonMovieImportService: DEBUG
    root: INFO
server:
  forward-headers-strategy: framework

springdoc:
  swagger-ui:
    operations-sorter: alpha
    tags-sorter: alpha
    servers:
      - url: https://i13d208.p.ssafy.io
        description: Production Server

management:
  endpoints:
    web:
      exposure:
        include: health,info,caches
  health:
    elasticsearch:
      enabled: false  # 이 줄 추가!

elasticsearch:
  index:
    movie: movie_index
    user: user_index
  matching:
    similarity-threshold: 0.7
    #date-diff-days: 90
fastapi:
  url: ${FASTAPI_URL:http://localhost:8000}  # 환경변수 우선, 없으면 기본값
  recommend-endpoint: ${FASTAPI_RECOMMEND_ENDPOINT:/recommen
d}
recommendation:
  api:
    base-url: ${FASTAPI_URL:http://localhost:8000}

spring.data.elasticsearch.index-and-mapping.skip-creating-indices: true
```

```
//build.gradle

plugins {
    id 'java'
    id 'org.springframework.boot' version '3.5.3'
    id 'io.spring.dependency-management' version '1.1.7'
}

group = 'com.d208'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-redi
s'  // Redis 의존성 추가
    implementation 'org.springframework.boot:spring-boot-starter-cache'
// 캐시 의존성 추가
    implementation 'org.springframework.boot:spring-boot-starter-data-elas
ticsearch'
    implementation 'co.elastic.clients:elasticsearch-java:8.15.0'
    implementation 'org.elasticsearch.client:elasticsearch-rest-client:8.15.0'
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.
8.6'
    implementation 'org.springframework.boot:spring-boot-starter-mail'
    implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
    implementation 'io.awspring.cloud:spring-cloud-aws-starter-s3:3.0.3'
    implementation 'software.amazon.awssdk:s3:2.21.29'
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'
    runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.mysql:mysql-connector-j'
    annotationProcessor 'org.springframework.boot:spring-boot-configurati
on-processor'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
    testImplementation 'junit:junit:4.13.2'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}

tasks.named('test') {
    useJUnitPlatform()
}

tasks.withType(JavaCompile) {
    options.compilerArgs << "-parameters"
}
```

Frontend (React)

- **API URL**: http://localhost:8081
- **Build Output**: dist 폴더
- **Serve Port**: 3000

```
//vite.config.js

import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    host: '0.0.0.0',
    port: 5173,
    allowedHosts: 'all'
  }
})
```

```
//package.json

{
  "name": "vite-project",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite --host 0.0.0.0",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "axios": "^1.11.0",
    "jwt-decode": "^4.0.0",
    "lucide-react": "^0.525.0",
    "prop-types": "^15.8.1",
    "react": "^19.1.0",
    "react-dom": "^19.1.0",
    "react-router-dom": "^7.7.0"
  },
  "devDependencies": {
    "@eslint/js": "^9.30.1",
    "@types/react": "^19.1.8",
    "@types/react-dom": "^19.1.6",
    "@vitejs/plugin-react": "^4.6.0",
    "eslint": "^9.30.1",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.20",
    "globals": "^16.3.0",
    "vite": "^7.0.4"
  }
}
```

```
//eslint.config.js

import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import { defineConfig, globalIgnores } from 'eslint/config'

export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{js,jsx}'],
    extends: [
      js.configs.recommended,
      reactHooks.configs['recommended-latest'],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    rules: {
      'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
    },
  },
])
```

AI (FastAPI)

- **FastAPI URL**: http://movie-fastapi:8000 (컨테이너 간 통신)
- **외부 접근**: http://localhost:8000
- **API 문서**: http://localhost:8000/docs
- **환경 변수**:
  - `DATA_PATH=/app/data`
  - `TRANSFORMERS_CACHE=/app/models`
  - `HF_HOME=/app/models`
- **주요 엔드포인트**:
  - `/api/v1/recommendations/user` - 사용자 기반 추천
  - `/api/v1/recommendations/new_user` - 신규 사용자 추천
  - `/api/v1/recommendations/feed` - 피드 추천
  - `/api/v1/keywordSearch` - 키워드 검색
  - `/api/v1/reviews/tags/recommend` - 태그 기반 추천

```
#requirements.txt

# FastAPI and web server
fastapi>=0.100.0
uvicorn[standard]>=0.23.0
```

```
python-multipart>=0.0.6

# Machine learning and NLP - using compatible versions
sentence-transformers>=2.0.0
scikit-learn>=1.2.0
numpy>=1.21.0
scipy>=1.9.0
torch>=1.9.0
transformers>=4.21.0

# Pydantic for data validation
pydantic>=2.0.0
pydantic-settings>=2.0.0

# Testing (optional for production)
pytest>=7.0.0
httpx>=0.24.0
```

4. EC2 포트 설정

시스템 정보

- **OS**: Ubuntu 22.04.5 LTS (Jammy)
- **Kernel**: Linux 6.8.0-1024-aws

현재 사용 중인 포트

- **443**: HTTPS (Nginx SSL)
- **80**: HTTP (Nginx - HTTPS 리다이렉트)
- **3000**: Frontend (React) - Nginx 프록시
- **8081**: Backend (Spring Boot) - Nginx 프록시
- **8000**: AI Service (FastAPI) - Nginx 프록시
- **8080**: Jenkins (Docker 컨테이너)
- **6379**: Redis (localhost 바인딩)
- **9200**: Elasticsearch

- **3306**: MySQL (외부 접근)

Nginx 프록시 설정

- **Domain**: i13d208.p.ssafy.io
- **SSL**: Let's Encrypt 인증서
- **API 라우팅**:
  - `/api/` → Backend (8081)
  - `/docs`, `/redoc` → FastAPI (8000)
  - AI 추천 엔드포인트들 → FastAPI (8000)
  - 나머지 → Frontend (3000)

```
# /etc/nginx/sites-available/default

server {
    listen 80;
    server_name i13d208.p.ssafy.io;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name i13d208.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/i13d208.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i13d208.p.ssafy.io/privkey.pem;

# FastAPI Swagger 문서 경로들 (API 경로보다 먼저 처리)
    location /docs {
        proxy_pass http://localhost:8000/docs;
        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;
    }

    location /redoc {
        proxy_pass http://localhost:8000/redoc;
        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;
    }

# 이거 추가!
    location /openapi.json {
        proxy_pass http://localhost:8000/openapi.json;
        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;
    }


    # API와 Swagger를 먼저 처리 (더 구체적인 경로가 먼저)
    location /api/ {
        proxy_pass http://localhost:8081/api/;
        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;
        # 핵심: 모든 헤더 전달 (Authorization 포함)
        proxy_pass_request_headers on;
        proxy_set_header Authorization $http_authorization;

        # CORS 설정
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELET
E, OPTIONS, PATCH' always;
        add_header 'Access-Control-Allow-Headers' 'Accept,Authorization,Ca
che-Control,Content-Type,DNT,If-Modified-Since,Keep-Alive,Origin,User-A
gent,X-Requested-With' always;

        # OPTIONS 요청 처리 (preflight)
        if ($request_method = OPTIONS) {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DEL
ETE, OPTIONS, PATCH';
            add_header 'Access-Control-Allow-Headers' 'Accept,Authorization,
Cache-Control,Content-Type,DNT,If-Modified-Since,Keep-Alive,Origin,User
-Agent,X-Requested-With';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain; charset=utf-8';
            add_header 'Content-Length' 0;
            return 204;
        }
    }


    location /swagger-ui/ {
        proxy_pass http://localhost:8081/swagger-ui/;
        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;
    }

    location /swagger/ {
        proxy_pass http://localhost:8081/swagger/;
        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;
    }

location /api/v1/reviews/tags/recommend {
        proxy_pass http://localhost:8000/api/v1/reviews/tags/recommend;
        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;
    }
    location /api/v1/keywordSearch {
        proxy_pass http://localhost:8000/api/v1/keywordSearch;
        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;
    }

location /api/v1/recommendations/user {
        proxy_pass http://localhost:8000/api/v1/recommendations/user;
        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;
    }

    location /api/v1/recommendations/new_user {
        proxy_pass http://localhost:8000/api/v1/recommendations/new_user;
        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;
    }

    location /api/v1/recommendations/feed {
        proxy_pass http://localhost:8000/api/v1/recommendations/feed;
        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;
    }



    location /v3/api-docs {
        proxy_pass http://localhost:8081/v3/api-docs;
        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;
    }

    # 나머지 모든 요청은 프론트엔드로 (가장 마지막에)
    location / {

        # GeoIP를 이용한 국가별 접속 차단
        if ($allowed_country = no) {
            return 403;
        }

        proxy_pass http://localhost:3000;
        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;
    }
}
```

보안그룹 설정 (AWS)

다음 포트들이 열려있어야 합니다:

- **22**: SSH 접근
- **80, 443**: HTTP/HTTPS 접근 (Nginx)
- **8080**: Jenkins 접근
- **8081**: Backend API 접근 (외부에서 직접 접근하는 경우)
- **8000**: FastAPI 접근 (외부에서 직접 접근하는 경우)
- **3306**: MySQL 접근 (외부 DB 클라이언트 사용 시)

5. 도커 설정

Docker Compose 구성

- **Redis**: redis:7-alpine
- **Elasticsearch**: 8.15.0 (Nori 플러그인 포함)

- **FastAPI**: Python 3.11-slim 기반
- **Backend**: Gradle 8.5-jdk21-alpine 기반 멀티스테이지 빌드
- **Frontend**: Node 20-alpine 기반

```
#docker-compose.yml

version: '3.8'

services:
  redis:
    image: redis:7-alpine
    container_name: movie-redis
    ports:
      - "127.0.0.1:6379:6379"
    command: redis-server
    volumes:
      - redis_data:/data
    networks:
      - movie-network
    restart: unless-stopped

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.15.0
    container_name: local-elasticsearch
    ports:
     - "9200:9200"
    environment:
      - discovery.type=single-node
      - ES_JAVA_OPTS=-Xms512m -Xmx1g
      - xpack.security.enabled=false
    networks:
      - movie-network
    restart: unless-stopped
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data

  fastapi:
    build:
      context: ./ai/fastapi
      dockerfile: Dockerfile
    container_name: movie-fastapi
    ports:
      - "8000:8000"
    volumes:
      - /home/ubuntu/ai-data:/app/data:ro
      - /home/ubuntu/ai-models:/app/models
    environment:
      - DATA_PATH=/app/data
      - TRANSFORMERS_CACHE=/app/models
      - HF_HOME=/app/models
    networks:
      - movie-network
    restart: unless-stopped

  backend:
    build:
      context: ./backend/feelroom
      dockerfile: Dockerfile
    container_name: movie-backend
    environment:
      - SPRING_DATASOURCE_URL=jdbc:mysql://i13D208.p.ssafy.io:3306/m
ovie_recommendation_db
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=gongtong208gumi
      - SPRING_REDIS_HOST=movie-redis
      - SPRING_REDIS_PORT=6379
      - ELASTICSEARCH_HOST=local-elasticsearch
      - ELASTICSEARCH_PORT=9200
      - FASTAPI_URL=http://movie-fastapi:8000
      - JWT_SECRET=${JWT_SECRET}
      - SPRING_PROFILES_ACTIVE=prod
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - AWS_REGION=${AWS_REGION}
    ports:
      - "8081:8080"
    depends_on:
      - redis
      - elasticsearch
      - fastapi
    networks:
      - movie-network
    restart: unless-stopped

  frontend:
    build:
      context: ./frontend/feelroom
      dockerfile: Dockerfile
    container_name: movie-frontend
    ports:
      - "3000:3000"
    environment:
      - REACT_APP_API_URL=http://localhost:8081
    depends_on:
      - backend
    networks:
      - movie-network
    restart: unless-stopped

volumes:
  redis_data:
  elasticsearch_data:

networks:
  movie-network:
    driver: bridge
```

```
#backend/feelroom/Dockerfile

FROM gradle:8.5-jdk21-alpine AS build

WORKDIR /app
```

```
# 전체 프로젝트 복사 (feelroom 폴더 내용)
COPY . .

# Gradle Wrapper 권한 설정 (있다면)
RUN chmod +x gradlew || true

# 애플리케이션 빌드
RUN gradle clean build --no-daemon -x test

# 실행 단계
FROM openjdk:21-jdk-slim

WORKDIR /app

# 타임존 설정 추가 시작
RUN apt-get update && apt-get install -y tzdata
ENV TZ=Asia/Seoul
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/tim
ezone
# 타임존 설정 추가 끝

# 빌드된 JAR 파일 복사 (구체적인 파일명 지정)
COPY --from=build /app/build/libs/feelroom-0.0.1-SNAPSHOT.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]
```

```
#frontend/feelroom/Dockerfile

FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
RUN npm install -g serve
```

```
EXPOSE 3000
CMD ["serve", "-s", "dist", "-l", "3000"]
```

```
#ai/fastapi/Dockerfile

FROM python:3.11-slim

WORKDIR /app

# 필요한 시스템 패키지 설치
RUN apt-get update && apt-get install -y \
    gcc g++ \
    && rm -rf /var/lib/apt/lists/*

# requirements.txt 복사 및 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 전체 프로젝트 복사 (순서 중요!)
COPY . /app

# 데이터와 모델 디렉토리 생성 (마운트 포인트)
RUN mkdir -p /app/data /app/models

# 환경변수 설정
ENV DATA_PATH=/app/data
ENV TRANSFORMERS_CACHE=/app/models
ENV HF_HOME=/app/models

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Elasticsearch 컨테이너 실행

```
#ec2에서 컨테이너 수동으로 실행
docker run -d \
  --name local-elasticsearch \
  -p 9200:9200 \
  -e "discovery.type=single-node" \
  -e "ES_JAVA_OPTS=-Xms512m -Xmx1g" \
  -e "xpack.security.enabled=false" \
  docker.elastic.co/elasticsearch/elasticsearch:8.15.0
```

네트워크

- **네트워크 이름**: movie-network
- **드라이버**: bridge

볼륨

- `redis_data`: Redis 데이터 영구 저장
- `elasticsearch_data`: Elasticsearch 데이터 영구 저장
- `/home/ubuntu/ai-data`: AI 데이터 (호스트 마운트)
- `/home/ubuntu/ai-models`: AI 모델 (호스트 마운트)

6. 젠킨스 설치

현재 Jenkins 설정

- **버전**: jenkins/jenkins:lts (Docker 컨테이너)
- **포트**: 8080 (외부 접근 가능)
- **실행 방법**: Docker 컨테이너로 실행 중
- **볼륨**: jenkins_data

Jenkins 설치 방법

```
# Jenkins를 Docker로 설치
docker run -d \
  --name jenkins \
  -p 8080:8080 \
  -p 50000:50000 \
  -v jenkins_data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  jenkins/jenkins:lts

# Jenkins에 Docker 권한 부여
docker exec -u 0 jenkins apt-get update
docker exec -u 0 jenkins apt-get install -y docker.io
docker exec -u 0 jenkins usermod -aG docker jenkins
docker restart jenkins
```

```
# Jenkinsfile

pipeline {
    agent {
        label 'built-in'
    }

    environment {
        COMPOSE_PROJECT_NAME = 'spring-react-app'
        // Jenkins가 Docker 컨테이너로 실행되므로 컨테이너 이름 사용
        ELASTICSEARCH_URL = 'local-elasticsearch:9200'
    }

    stages {
        stage('Checkout') {
            steps {
                echo 'Checking out code from GitLab...'
                checkout scm
            }
        }

        stage('Build Frontend') {
            steps {
                echo 'Building React frontend...'
                dir('frontend/feelroom') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }
        stage('Build Backend') {
            steps {
                echo 'Building Spring Boot backend...'
                dir('backend/feelroom') {
                    sh 'chmod +x gradlew'
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage('Stop Current Services') {
            steps {
                echo 'Stopping current services (except Elasticsearch)...'
                sh '''
                    # 특정 컨테이너들만 중지 (Elasticsearch 제외)
                    docker stop movie-redis movie-backend movie-frontend || true
                    docker rm movie-redis movie-backend movie-frontend || true
                '''
            }
        }

        stage('Setup Elasticsearch') {
            steps {
                echo 'Setting up Elasticsearch with Nori...'
                sh '''
                    # Elasticsearch가 실행중인지 확인
                    if ! docker ps | grep -q local-elasticsearch; then
                        echo "Starting Elasticsearch..."
                        docker-compose up -d elasticsearch
                        sleep 30

                        # Nori 플러그인 설치
                        echo "Installing Nori plugin..."
                        docker exec local-elasticsearch bin/elasticsearch-plugin inst
all analysis-nori || true
                        docker restart local-elasticsearch
                        sleep 30
                    else
                        echo "Elasticsearch already running"

                        # Nori 플러그인 확인 및 설치
                        if ! docker exec local-elasticsearch bin/elasticsearch-plugin l
ist | grep -q analysis-nori; then
                            echo "Installing Nori plugin..."
                            docker exec local-elasticsearch bin/elasticsearch-plugin i
nstall analysis-nori || true
                            docker restart local-elasticsearch
                            sleep 30
                        fi
                    fi

                    # Jenkins를 같은 네트워크에 연결 (이미 연결되어 있다면 에러 무시)
                    docker network connect spring-react-app_movie-network jenki
ns || true

                    # Elasticsearch 연결 테스트 (Docker 네트워크 사용)
                    echo "Testing Elasticsearch connection via Docker network..."
                    curl -v "http://${ELASTICSEARCH_URL}/" || echo "Connection t
est failed, but continuing..."

                    # 인덱스 존재 확인 및 생성
                    echo "Checking if movies index exists..."
                    INDEX_EXISTS=$(curl -s -o /dev/null -w "%{http_code}" "htt
p://${ELASTICSEARCH_URL}/movies" || echo "404")

                    if [ "$INDEX_EXISTS" = "404" ]; then
                        echo "Creating movies index..."
                        curl -X PUT "http://${ELASTICSEARCH_URL}/movies" \
                             -H 'Content-Type: application/json' \
                             -d '{
                               "settings": {
                                 "analysis": {
                                   "analyzer": {
                                     "nori_analyzer": {
                                       "type": "nori",
                                       "decompound_mode": "mixed"
                                     }
                                   }
                                 }
                               },
                               "mappings": {
                                 "properties": {
                                   "movieId": { "type": "integer" },
                                   "title": {
                                     "type": "text",
                                     "analyzer": "nori_analyzer",
                                     "fields": {
                                       "exact": { "type": "keyword" }
                                     }
                                   },
                                   "overview": {
                                     "type": "text",
                                     "analyzer": "nori_analyzer"
                                   },
                                   "releaseDate": { "type": "date" },
                                   "voteAverage": { "type": "float" },
                                   "voteCount": { "type": "integer" },
                                   "runtime": { "type": "integer" },
                                   "posterUrl": { "type": "keyword" },
                                   "tmdbId": { "type": "integer" },
                                   "genres": { "type": "keyword" },
                                   "actors": { "type": "keyword" },
                                   "directors": { "type": "keyword" }
                                 }
                               }
                             }'
                        echo ""
                        echo "Index creation completed"
                    else
                        echo "Movies index already exists (HTTP $INDEX_EXISTS)"
                    fi

                    # Elasticsearch 상태 확인
```

```
                echo "Checking Elasticsearch status..."
                curl -s "http://${ELASTICSEARCH_URL}/_cluster/health?pretty"
|| echo "Status check failed"
                echo "Checking plugins..."
                docker exec local-elasticsearch bin/elasticsearch-plugin list
             '''
          }
       }


       stage('Deploy') {
   steps {
       echo 'Starting services...'
       sh '''
          # 특정 컨테이너들 완전 정리 (Elasticsearch 제외)
          echo "Stopping and removing containers..."
          docker stop movie-redis movie-fastapi movie-backend movie-front
end || true
          docker rm movie-redis movie-fastapi movie-backend movie-fronten
d || true

          # 혹시 남아있는 dangling 컨테이너들 정리
          docker container prune -f

          # 서비스 시작 (강제 리빌드 및 재생성)
          echo "Starting services with force recreate..."
          docker-compose up -d --build --force-recreate redis fastapi backen
d frontend

          # 컨테이너 상태 확인
          echo "Checking container status..."
          docker ps --filter "name=movie-"
       '''
   }
}

       stage('Health Check') {
          steps {
                echo 'Checking if services are running...'
                sh '''
                    sleep 30
                    docker-compose ps

                    # Elasticsearch 인덱스 확인
                    echo "Checking Elasticsearch indices..."
                    curl -s "http://${ELASTICSEARCH_URL}/_cat/indices?v" || echo
"Index check failed"

                    # 서비스 상태 확인
                    echo "Checking service connectivity..."
                    echo "Backend status:"
                    curl -f http://localhost:8081/actuator/health || echo "Backend n
ot ready yet"
                    echo "Frontend status:"
                    curl -f http://localhost:3000 || echo "Frontend not ready yet"
                '''
            }
        }
    }

    post {
        success {
            echo 'Deployment successful! 🎉'
        }
        failure {
            echo 'Deployment failed! 😞'
            sh '''
                echo "=== Docker Compose Logs ==="
                docker-compose logs --tail=50
                echo "=== Docker Network Info ==="
                docker network ls
                docker network inspect spring-react-app_movie-network || true
                echo "=== Container Status ==="
                docker ps -a
            '''
        }
        always {
            echo 'Cleaning up...'
            sh 'docker system prune -f || true'
        }
    }
}
```

Jenkins Pipeline 설정

- **프로젝트명**: spring-react-app
- **트리거**: GitLab 웹훅 (Push events, Merge request events)
- **빌드 단계**:
    1. Checkout (GitLab)
    2. Build Frontend (npm install & build)
    3. Build Backend (Gradle clean build)
    4. Stop Current Services
    5. Setup Elasticsearch (Nori 플러그인)
    6. Deploy (Docker Compose)
    7. Health Check

GitLab 웹훅 설정

```
웹훅 URL: http://i13d208.p.ssafy.io:8080/project/feelroom
트리거: Push events, Merge request events
시크릿 토큰: Jenkins에서 생성된 토큰 사용
```

Jenkins 권한 설정

```
# Jenkins 컨테이너에 Docker 접근 권한 부여
docker exec -u 0 jenkins apt-get update
docker exec -u 0 jenkins apt-get install -y docker.io
docker exec -u 0 jenkins usermod -aG docker jenkins
```

```
# Jenkins 재시작
docker restart jenkins
```

7. 추가 인프라 설정

MySQL 설치 및 설정

```
# MySQL 설치
sudo apt update
sudo apt install mysql-server

# MySQL 보안 설정
sudo mysql_secure_installation

# MySQL 설정
sudo mysql -u root -p
```

스키마 생성

```
-- DDL 파일을 사용하여 스키마 생성
mysql -u root -p < ddl.sql
# /backend/feelroom/src/main/resources 위치
```

Nginx 설치 및 SSL 설정

```
# Nginx 설치
sudo apt install nginx

# Let's Encrypt 인증서 설치
sudo apt install certbot python3-certbot-nginx

# SSL 인증서 발급
sudo certbot --nginx -d i13d208.p.ssafy.io
```

8. 추천 서버 실행 가이드

AI 추천 프로젝트 링크

제출용_추천서버.zip
https://drive.google.com/file/d/12ebHtxF96b_OePOZ79MDqzg0UIWJruWG/view?usp=drive_link

프로젝트 구조

```
recommendation-server/
├── data/              # ratings.csv, links.csv, *.pkl 등 모든 원본 데이터
├── models/            # 학습 후 모델 저장 폴더
├── train.py           # 영화 추천 학습 스크립트
├── train_review.py    # 리뷰 추천 학습 스크립트
├── main.py            # 실시간 추천 API 서버
└── requirements.txt   # 필요한 모든 라이브러리 목록
```

개발 환경 설정

1. **Conda 가상환경 생성**

   ```
   conda create -n ai_env python=3.10 -y
   ```

2. **가상환경 활성화**

   ```
   conda activate ai_env
   ```

3. **필요한 라이브러리 설치:** `recommendation-server` 폴더로 이동한 뒤, 아래 명령어를 실행

   ```
   pip install -r requirements.txt
   ```

   (만약 `scikit-surprise` 설치 오류가 발생하면, `conda install -c conda-forge scikit-surprise` 를 먼저 실행한 뒤 다시 시도합니다.)

4. **VS Code 인터프리터 설정:**
   - `Ctrl + Shift + P` → `Python: Select Interpreter` 검색
   - 목록에서 `(ai_env)` 라고 표시된 Conda 환경을 선택합니다

모델 학습

1. **영화 추천 모델 학습:** `(ai_env)`가 활성화된 터미널에서 아래 명령어를 실행

    ```
    python train.py
    ```

실행 완료 후 `./models` 폴더 안에 `.pkl`과 `.json` 파일들이 생성된 것을 확인

API 서버 실행

1. **서버 실행:** `(ai_env)`가 활성화된 터미널에서 아래 명령어를 실행합니다.

    ```
    uvicorn main:app --reload
    ```

2. **실행 확인:** 터미널에 `Uvicorn running on http://127.0.0.1:8000` 메시지가 보이면 **이 터미널은 끄지 말고 그대로 두세요.**

로컬 서버 외부 연결

1. `ngrok` 실행

    ```
    ngrok http 8000
    ```

2. `Forwarding` 부분에 있는 **`https://...ngrok-free.app`** 주소를 복사(추천 서버의 외부주소)
3. `AI/fastapi/main.py` 파일에서 `MOVIE_SERVER_URL` 변수에 해당 주소를 입력.
    - **주의:** 주소 마지막에 `/`가 붙지 않도록 주의해주세요!
    - (예: `https://...ngrok-free.app` O, `https://...ngrok-free.app/` X)

실행 확인

- 웹 브라우저를 열고 주소창에 아래 주소를 입력합니다.
- **http://127.0.0.1:8000/docs**
- 여기서 API를 직접 테스트할 수 있습니다.