포팅 매뉴얼

선택	인프라 및 개발환경

개발 환경

Frontend

- React19
- TypeScript
- Vite

Backend

- Java17
  - Java OpenJDK
  - Spring Boot3
    - Spring Data JPA
    - Spring Data Redis
    - Lombok
  - Spring Websocket
  - Gradle
  - Socket-io

AI

- FastAPI
- Uvicorn
- Tensorflow/Keras
- PyTorch + Transformers

DB

- Redis

UI/UX

- Figma

Server 배포 환경

- AWS EC2
- Docker
- Docker Compose
- Docker Hub
- Nginx

Collaboration

형상관리

- GitLab

커뮤니케이션

- Mattermost
- Notion

이슈관리

- Jira

환경 변수 설정

Backend

application.yml

```
# Socket.IO 및 Docker 통신 설정
server:
  port: 8080

# Socket.IO 서버 설정
socketio:
  server:
    host: ${SOCKETIO_HOST:0.0.0.0}
    port: ${SOCKETIO_PORT:9092}

# Redis 설정
spring:
  # 기본 활성 프로필 지정: 로컬 개발 시 'dev'를 자동으로 활성화 (매번 IntelliJ에서 설정할 필요 없음)
  profiles:
    active: dev  # 로컬에서 아무 설정 없이 실행하면 'dev' 프로필이 기본으로 적용됩니
```

```
다. 배포 시 환경 변수로 'production'으로 덮어쓰세요.
  application:
    name: common-backend
  data:
      timeout: 3000ms
      lettuce:
        pool:
          max-active: 10
          max-idle: 10
          min-idle: 2
          max-wait: 3000ms

# 필터 설정
filter:
  use-redis: ${USE_REDIS:true}
  session-ttl-hours: 24

# 로깅 설정
logging:
  level:
    backend.SSAFY_PTJ2: DEBUG
    # JMX/RMI 로그 레벨 조정 (TRACE/DEBUG → WARN)
    javax.management: WARN
    sun.rmi: WARN
    com.sun.jmx: WARN
  pattern:
    console: "%d{dd HH:mm:ss} - %msg%n"
```

application.yml - dev

```
# 로컬 개발
spring.config.activate.on-profile: dev

socketio:
```

```
  server:
    host: localhost

spring:
  application:
    name: common-backend
  data:
    redis:
      host: localhost
      port: ${REDIS_PORT:6380}
# AI 통신 클라이언트 설정값
ai:
  image:
    base-url: "http://localhost:8001"
  text:
    base-url: "http://localhost:8002"
  timeout-ms:  # HTTP 요청 관련 타임아웃
    connect: 150000
    read: 25000
    write: 25000
#  retry:
#    max-attempts: 3
#    backoff-ms: 200
#    jitter: 0.3
  upload:
    max-bytes: 10485760  # 10MB

logging:
  level:
    root: DEBUG
  pattern: "[%d{dd'일' HH:mm:ss}][%M]⇒ %m%n"
```

application.yml - prod

```
spring.config.activate.on-profile: production

socketio:
  server:
    host: 0.0.0.0

spring:
  data:
    redis:
      host: redis  # Docker 컨테이너 간 통신: 서비스명으로 접근
      port: 6379

# AI 컨테이너 설정 (컨테이너 간 네트워크 통신)
ai:
  image:
    base-url: "http://fastapi-ai-image-service:8000"    # Docker Compose 서비스명:내부포트
  text:
    base-url: "http://fastapi-ai-text-service:8000"     # Docker Compose 서비스명:내부포트

filter:
  use-redis: true

logging:
  level:
    root: DEBUG
```

Frontend

.env → dev

```
VITE_SOCKET_IO_BASE=http://localhost:9092/
```

배포 환경 설정

초기 세팅

1. **EC2 접속**

   ```
   ssh -i J13A207T.pem ubuntu@j13a207.p.ssafy.io
   ```

2. **Docker & Docker Engine 설치**
3. **Docker Compose 설치**

Docker 컨테이너 생성

> Docker Compose 파일 실행 시 자동 생성
> Nginx, AI 이미지, AI 텍스트, Spring, Redis, Jenkins

- **docker ps 결과**

```
ubuntu@ip-172-26-1-215:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED        STATUS                    PORTS                                                                                  NAMES
db5e4496153a   nginx:alpine                   "/docker-entrypoint.…"   18 hours ago   Up 18 hours               0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   main-nginx
00657cf18870   main-ec2-fastapi-ai-image-service   "uvicorn app.main:ap…"   18 hours ago   Up 18 hours               127.0.0.1:8001->8000/tcp                                                               main-ai-image
9c2358bbb31b   main-ec2-common-backend        "sh -c 'java $JAVA_O…"   18 hours ago   Up 18 hours (unhealthy)   127.0.0.1:8080->8080/tcp, 0.0.0.0:9092->9092/tcp                                       main-backend
d352fcf10e2a   main-ec2-fastapi-ai-text-service    "uvicorn app.main:ap…"   18 hours ago   Up 18 hours               127.0.0.1:8002->8000/tcp                                                               main-ai-text
45df1e590e52   redis:7-alpine                 "docker-entrypoint.s…"   18 hours ago   Up 18 hours               127.0.0.1:6380->6379/tcp                                                               main-redis
d598e5630bf5   jenkins/jenkins:lts            "/usr/bin/tini -- /u…"   11 days ago    Up 11 days                0.0.0.0:50000->50000/tcp, [::]:50000->50000/tcp, 0.0.0.0:9090->8080/tcp, [::]:9090->8080/tcp   main-jenkins
```

Docker Compose 파일

fastapi-ai-image-service

```
fastapi-ai-image-service:
  build:
    context: ../../Service/ai-image-service
    dockerfile: Dockerfile
  container_name: main-ai-image
```

```
  ports:
    - "127.0.0.1:8001:8000" # ← 호스트 8001 → 컨테이너 8000
  environment:
    - MODEL_PATH=/app/models/image/student_ft_best.keras
    - CLASSMAP_PATH=/app/models/image/class_mapping.json
    - THRESH_PATH=/app/models/image/thresholds.json
    - REDIS_URL=redis://redis:6379
    - LOG_LEVEL=INFO
  volumes:
    - /home/ubuntu/app/models:/app/models
    - /home/ubuntu/app/logs/ai-image:/app/logs
  depends_on:
    - redis
  networks:
    - main-network
  restart: unless-stopped
  deploy:
    resources:
      limits:
        memory: 2G
        cpus: '2.0'
```

fastapi-ai-text-service

```
  fastapi-ai-text-service:
    build:
      context: ../../Service/ai-text-service
      dockerfile: Dockerfile
    container_name: main-ai-text
    ports:
      - "127.0.0.1:8002:8000" # ← 호스트 8002 → 컨테이너 8000
    environment:
      - AI_MODEL_PATH=/app/models
      - REDIS_URL=redis://redis:6379
```

```
    - LOG_LEVEL=INFO
  volumes:
    - /home/ubuntu/app/models:/app/models
    - /home/ubuntu/app/logs/ai-text:/app/logs
  depends_on:
    - redis
  networks:
    - main-network
  restart: unless-stopped
  deploy:
    resources:
      limits:
        memory: 2G
        cpus: '2.0'
```

redis

```
 redis:
  image: redis:7-alpine
  container_name: main-redis
  command: redis-server --appendonly yes --maxmemory 512mb --maxmem
ory-policy allkeys-lru
  ports:
    - "127.0.0.1:6380:6379"
  volumes:
    - redis-data:/data
  networks:
    - main-network
  restart: unless-stopped
```

common-backend

```
  common-backend:
    build:
      context: ../../Service/common-backend
      dockerfile: Dockerfile
    container_name: main-backend
    ports:
      - "127.0.0.1:8080:8080"
      - "0.0.0.0:9092:9092"
    environment:
      - FASTAPI_IMAGE_URL=http://fastapi-ai-image-service:8000  # 컨테이너명
이랑 일치
      - FASTAPI_TEXT_URL=http://fastapi-ai-text-service:8000  # 컨테이너명이랑
일치
      - SPRING_PROFILES_ACTIVE=production
      - REDIS_HOST=redis
      - REDIS_PORT=6380
      - USE_REDIS=true
      - SOCKETIO_HOST=0.0.0.0
      - SOCKETIO_PORT=9092
    depends_on:
      - redis
    networks:
      - main-network
    restart: unless-stopped
```

nginx

```
  nginx:
    image: nginx:alpine
    container_name: main-nginx
    ports:
      - "80:80"
      - "443:443"
      - "9090:9090"
```

```
volumes:
  - ./logs/nginx:/var/log/nginx
  - extension-downloads:/var/www/html/downloads
  - ./extension-web:/var/www/html/extension
  - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro  # nginx.conf 마운트
  # SSL 인증서 볼륨
  - /etc/letsencrypt:/etc/letsencrypt:ro
  - /var/www/certbot:/var/www/certbot
depends_on:
  - fastapi-ai-image-service
  - fastapi-ai-text-service
  - common-backend
networks:
  - main-network
restart: unless-stopped
```

Nginx 설치 + SSL 인증키 발급

1. **Nginx 설치**

   $ `sudo apt update && sudo apt upgrade`

   $ `sudo apt install nginx`

   $ `sudo service nginx start`

2. **Encrypt, Certbot 설치**

   $ `sudo apt-get install letsencrypt`

   $ `sudo apt-get install certbot python3-certbot-nginx`

3. **SSL 인증서 발급**

```
# Certbot 동작 (nginx 중지하고 해야함)
$ sudo systemctl stop nginx
```

```
# Nginx 상태 확인 & 80번 포트 확인
$ sudo service nginx status
$ netstat -na | grep '80/*LISTEN'

# SSL 인증서 발급
$ sudo certbot --nginx
$ sudo letsencrypt certonly --standalone -d j13a207.p.ssafy.io

# 설치한 인증서 확인 및 위치 확인
$ sudo certbot certificates

# nginx 설정 적용
# nginx 재시작
$ sudo service nginx restart
$ sudo systemctl reload nginx
```

Nginx conf 설정

```
events {
    worker_connections 1024;
    use epoll;
    multi_accept on;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # WebSocket 업그레이드 매핑
    map $http_upgrade $connection_upgrade { default upgrade; '' close; }

    # 로그 포맷
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
```

```
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for" '
                    'rt=$request_time uct="$upstream_connect_time" '
                    'uht="$upstream_header_time" urt="$upstream_response_time"';

    access_log /var/log/nginx/access.log main;
    error_log /var/log/nginx/error.log warn;

    # 성능 기본
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    client_max_body_size 100M;

    # Gzip
    gzip on;
    gzip_vary on;
    gzip_min_length 1000;
    gzip_types
        text/plain
        text/css
        text/xml
        text/javascript
        application/json
        application/javascript
        application/xml+rss
        application/atom+xml
        image/svg+xml;

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
    limit_req_zone $binary_remote_addr zone=general:10m rate=50r/s;

    # Upstreams
```

```
    upstream ai_image_backend { server fastapi-ai-image-service:8000; keepa
live 16; }
    upstream ai_text_backend  { server fastapi-ai-text-service:8000;  keepalive
16; }
    upstream common_backend   { server common-backend:8080;          keep
alive 16; }
    upstream socketio_backend { server common-backend:9092;          keepali
ve 16; }
    upstream redis_backend    { server redis:6379; }
    upstream jenkins_backend  { server localhost:9090;             keepalive 8; }

    # HTTP → HTTPS 리디렉션 서버
    server {
        listen 80;
        server_name j13a207.p.ssafy.io;

        # Let's Encrypt 인증을 위한 경로
        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
            try_files $uri =404;
        }

        # 나머지 모든 요청을 HTTPS로 리디렉션
        location / {
            return 301 https://$server_name$request_uri;
        }
    }

    # HTTPS 서버
    server {
        listen 443 ssl http2;
        server_name j13a207.p.ssafy.io;

        # SSL 인증서 설정
        ssl_certificate /etc/letsencrypt/live/j13a207.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/j13a207.p.ssafy.io/privkey.pem;
```

```
        # SSL 보안 설정
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-G
CM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384;
        ssl_prefer_server_ciphers off;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 1d;
        ssl_stapling on;
        ssl_stapling_verify on;

        # 보안 헤더 (HTTPS 강화)
        add_header X-Frame-Options DENY always;
        add_header X-Content-Type-Options nosniff always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header Referrer-Policy "strict-origin-when-cross-origin" always;
        add_header Strict-Transport-Security "max-age=31536000; includeSubD
omains" always;
        add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'u
nsafe-inline' 'unsafe-eval'; style-src 'self' 'unsafe-inline'; img-src 'self' data:
https:; connect-src 'self' wss: https:; font-src 'self' data:;" always;

        # ----- API -----
        location /api/ {
            limit_req zone=api burst=50 nodelay;

            proxy_pass http://common_backend/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;

            proxy_set_header Connection "";
            proxy_http_version 1.1;

            proxy_connect_timeout 10s;
```

```
        proxy_send_timeout    90s;
        proxy_read_timeout    90s;

        proxy_cache_bypass 1;
        proxy_no_cache   1;
    }

    # ----- WebSocket -----
    location /ws/ {
        proxy_pass http://socketio_backend/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade    $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        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;

        proxy_connect_timeout 5s;
        proxy_send_timeout   300s;
        proxy_read_timeout   300s;
        proxy_buffering off;
    }

    # ----- Socket.IO -----
    location /socket.io/ {
        proxy_pass http://socketio_backend/socket.io/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade    $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        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;

        proxy_connect_timeout 5s;
```

```
        proxy_send_timeout   300s;
        proxy_read_timeout   300s;
        proxy_buffering off;
    }

    # ----- Health -----
    location /health {
        limit_req zone=general burst=10 nodelay;

        proxy_pass http://common_backend/health;
        proxy_set_header Host      $host;
        proxy_set_header X-Real-IP $remote_addr;

        proxy_connect_timeout 2s;
        proxy_send_timeout    5s;
        proxy_read_timeout    5s;
    }

    # ----- Extension -----
    location /extension/ {
        limit_req zone=general burst=10 nodelay;
        alias /var/www/html/extension/;
        try_files $uri $uri/ /extension/index.html;

        location ~* \.html$ {
            expires -1;
            add_header Cache-Control "no-cache, no-store, must-revalidate";
        }
    }

    location /downloads/ {
        limit_req zone=general burst=5 nodelay;
        alias /var/www/html/downloads/;

        location ~* \.zip$ {
            add_header Content-Disposition "attachment";
```

```
            add_header Content-Type "application/zip";
            expires 1h;
            add_header Cache-Control "public";
        }
    }

    location /static/ {
        limit_req zone=general burst=50 nodelay;
        root /var/www/html;
        expires 1d;
        add_header Cache-Control "public, immutable";
    }

    # ----- Root -----
    location / {
        limit_req zone=general burst=20 nodelay;
        default_type application/json;
        return 200 '{"status":"OK","service":"AI Extension Main Server","timest
amp":"$time_iso8601"}';
    }

    # Errors
    error_page 404 /404.html;
    error_page 500 502 503 504 /50x.html;

    location = /404.html { root /var/www/html; internal; }
    location = /50x.html { root /var/www/html; internal; }
  }

  # HTTPS Jenkins 서버 (포트 9090)
  server {
    listen 9090 ssl http2;
    server_name j13a207.p.ssafy.io;

    # SSL 인증서 설정 (메인 서버와 동일)
    ssl_certificate /etc/letsencrypt/live/j13a207.p.ssafy.io/fullchain.pem;
```

```
        ssl_certificate_key /etc/letsencrypt/live/j13a207.p.ssafy.io/privkey.pem;

        # SSL 보안 설정
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-G
CM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384;
        ssl_prefer_server_ciphers off;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 1d;

        # 보안 헤더
        add_header X-Frame-Options SAMEORIGIN always;
        add_header X-Content-Type-Options nosniff always;
        add_header Strict-Transport-Security "max-age=31536000; includeSubD
omains" always;

        # Jenkins 프록시
        location / {
            proxy_pass http://jenkins_backend;
            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;

            proxy_connect_timeout 10s;
            proxy_send_timeout    300s;
            proxy_read_timeout    300s;

            # Jenkins 특화 설정
            proxy_redirect http://localhost:9090/ https://$host:9090/;
        }
    }
}
```

Jenkins 설정

: 특정 브랜치를 추적하여 자동 배포가 진행되도록 한다.

매인 Jenkinsfile

- develop 브랜치에 Merge 이벤트가 발생 시 배포 진행

```
// jenkinsfile (프로젝트 루트)
pipeline {
    agent any

    environment {
        GITLAB_REPO = 'https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A207.git'
        MAIN_DEPLOY_PATH = '/home/ubuntu/app'
    }

    stages {
        stage('Prepare Environment') {
            steps {
                script {
                    echo "🔧 환경 준비 중..."

                    sh '''
                        # 필수 디렉토리 생성
                        mkdir -p ${MAIN_DEPLOY_PATH}/{models/{text,image},logs/{fastapi,nginx},extension-downloads}
                        chmod -R 755 ${MAIN_DEPLOY_PATH}

                        # Docker 버전 확인
                        docker --version
                        docker-compose --version || echo "Docker Compose 설치 필요"

                        echo "✅ 환경 준비 완료"
                    '''
```

```
            }
        }
    }

    stage('Checkout') {
        steps {
            git branch: 'develop',
                credentialsId: 'gitlab-token',
                url: "${GITLAB_REPO}"
        }
    }

    stage('Detect Changes') {
        steps {
            script {
                def changes = sh(
                    script: 'git diff --name-only HEAD~1 HEAD || echo "all"',
                    returnStdout: true
                ).trim().split('\n')

                // 실제 소스코드 경로 기반 변경 감지
                env.DEPLOY_MAIN = changes.any {
                    it.startsWith('Service/') ||          // FastAPI 소스코드 변경
                    it.startsWith('Infra/main-ec2/')          // 메인 EC2 인프라 변경
                } ? 'true' : 'false'

                env.DEPLOY_EXTENSION = changes.any {
                    it.startsWith('Front-End/SSAFY_PJT2/') ||    // 크롬 익스텐션 소스
코드 변경
                    it.startsWith('Infra/main-ec2/')           // 메인 EC2 인프라 변경 시
에도 익스텐션 재빌드
                } ? 'true' : 'false'

                echo "🔍 변경 감지 결과:"
                echo "  • 메인 EC2 배포: ${env.DEPLOY_MAIN}"
                echo "  • 크롬 익스텐션 배포: ${env.DEPLOY_EXTENSION}"
```

```
            }
        }
    }

    stage('Deploy Main EC2 (Local)') {
        when {
            anyOf {
                allOf {
                    branch 'develop'  // develop 브랜치에서
                    anyOf {
                        environment name: 'DEPLOY_MAIN', value: 'true'  // 변경사항
이 있거나
                        expression { currentBuild.getBuildCauses('hudson.model.Cau
se$UserIdCause').size() > 0 }  // 수동 빌드
                    }
                }
                expression { currentBuild.getBuildCauses('hudson.model.Cause
$UserIdCause').size() > 0 }  // Jenkins에서 수동 빌드 시 항상 실행
            }
        }
        steps {
            script {
                echo "🚀 메인 EC2 로컬 배포 시작..."

                withCredentials([
                    file(credentialsId: 'main-env-file', variable: 'MAIN_ENV')
                ]) {
                    sh '''
                        # === 강화된 디렉토리 생성 ===
                        echo "📁 필수 디렉토리 강제 생성 중..."
                        mkdir -p ${MAIN_DEPLOY_PATH}/{models/{text,image},logs/
{fastapi,nginx},extension-downloads,data}
                        chmod -R 755 ${MAIN_DEPLOY_PATH}

                        # 디렉토리 존재 확인
                        echo "📋 디렉토리 확인:"
```

```
ls -la ${MAIN_DEPLOY_PATH}/

# models 디렉토리 강제 생성
if [ ! -d "${MAIN_DEPLOY_PATH}/models" ]; then
    echo "⚠️ models 디렉토리 없음 - 강제 생성"
    mkdir -p ${MAIN_DEPLOY_PATH}/models/{text,image}
    chmod -R 755 ${MAIN_DEPLOY_PATH}/models
fi

# === 파일 복사 (git 구조와 동일하게) ===
# 복사 전 기존 파일들을 정리
rm -rf ${MAIN_DEPLOY_PATH}/{Service,Front-End,Infra}

# 메인 EC2에 필요한 폴더들을 내부 파일까지 완전히 복사
cp -r Service ${MAIN_DEPLOY_PATH}/
cp -r Front-End ${MAIN_DEPLOY_PATH}/
cp -r Infra ${MAIN_DEPLOY_PATH}/
cp $MAIN_ENV ${MAIN_DEPLOY_PATH}/.env

cd ${MAIN_DEPLOY_PATH}
export $(cat .env | xargs)

# === AI 모델 파일 복사 ===
echo "📥 AI 모델 파일 복사 중..."

# 모델 서브디렉토리 생성
mkdir -p models/text models/image
chmod -R 755 models/

if [ -d "Service/ai-text-service/models" ]; then
    cp -r Service/ai-text-service/models/* models/text/ || true
    echo "✅ Text 모델 복사 완료"
    ls -la models/text/
else
    echo "⚠️ Text 모델 소스 디렉토리 없음"
fi
```

```
                    if [ -d "Service/ai-image-service/models" ]; then
                        cp -r Service/ai-image-service/models/* models/image/ ||
true
                        echo "✅ Image 모델 복사 완료"
                        ls -la models/image/
                    else
                        echo "⚠️ Image 모델 소스 디렉토리 없음"
                    fi

                    # === 안전한 모델 파일 생성 ===
                    echo "📥 기본 모델 설정 중..."
                    if [ -d "models" ]; then
                        echo "기본 모델 파일" > models/latest
                        echo "✅ models/latest 파일 생성 완료"
                    else
                        echo "❌ models 디렉토리 생성 실패!"
                        exit 1
                    fi

                    # === Docker 서비스 재시작 ===
                    echo "🔄 메인 서비스 재시작 중..."
                    docker-compose -f Infra/main-ec2/docker-compose.main.y
ml down || true
                    docker-compose -f Infra/main-ec2/docker-compose.main.y
ml up -d --build

                    echo "✅ 메인 EC2 배포 완료"
                '''
            }
        }
    }

    stage('Health Check') {
        steps {
```

```
            script {
                echo "🏥 헬스체크 수행 중..."

                // 메인 EC2 헬스체크
                sh '''
                    if [ "${DEPLOY_MAIN}" = "true" ]; then
                        echo "🔍 메인 EC2 FastAPI 상태 확인..."
                        sleep 45

                        # FastAPI 헬스체크
                        for i in {1..10}; do
                            if curl -f http://localhost:8000/health > /dev/null 2>&1; then
                                echo "✅ 메인 EC2 FastAPI 정상 동작"
                                break
                            fi
                            echo "⏳ FastAPI 시작 대기 중... ($i/10)"
                            sleep 10
                        done
                    fi
                '''
            }
        }
    }
}

post {
    success {
        echo '🎉 모든 배포가 성공적으로 완료되었습니다!'
        script {
            // 성공 시 상태 확인
            sh '''
                echo "📊 최종 서비스 상태:"
                docker ps --format "table {{.Names}}\\t{{.Status}}\\t{{.Ports}}"
            '''
        }
    }
```

```
        failure {
            echo '❌ 배포 실패! 로그를 확인하세요.'
            script {
                // 실패 시 로그 수집
                sh '''
                    echo "💥 메인 EC2 로그:"
                    docker-compose -f ${MAIN_DEPLOY_PATH}/Infra/main-ec2/dock
er-compose.main.yml logs --tail=50 || true
                '''

            }
        }

        always {
            // 정리 작업
            sh '''
                # 환경변수 파일 보안 삭제
                rm -f ${MAIN_DEPLOY_PATH}/.env || true

                # 미사용 Docker 이미지 정리 (디스크 공간 확보)
                docker image prune -f || true
            '''
        }
    }
}
```

AI 이미지 Jenkinsfile

- deploy-image 브랜치에 Merge 이벤트가 발생 시 배포 진행

```
pipeline {
  agent any
  environment {
    GITLAB_REPO  = 'https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A207.
```

```
git'
    BRANCH_NAME  = 'deploy-image'
    SERVICE_NAME = 'ai-image-service'
    SERVICE_PORT = '8001'
    CONTAINER_PORT = '8000'
    CREDENTIALS_ID = 'gitlab-token'
    DEPLOY_PATH  = '/home/ubuntu/services/image'
  }

  stages {
    stage('Checkout') {
      steps {
        echo "🔄 ${SERVICE_NAME} 소스 체크아웃 중..."
        git branch: "${BRANCH_NAME}", credentialsId: "${CREDENTIALS_ID}", u
rl: "${GITLAB_REPO}"
      }
    }

    stage('Prepare deploy dir') {
      steps {
        echo "📁 배포 디렉토리 준비 중..."
        sh '''
          set -e
          mkdir -p ${DEPLOY_PATH}
          chmod -R 755 ${DEPLOY_PATH}

          # 기존 파일 백업 및 정리
          rm -rf ${DEPLOY_PATH}/backup || true
          if [ -d "${DEPLOY_PATH}/app" ]; then
            mv ${DEPLOY_PATH}/app ${DEPLOY_PATH}/backup
          fi

          # 새 디렉토리 생성
          mkdir -p ${DEPLOY_PATH}/app

          # 내용만 복사 (중요)
```

```
        cp -r Service/ai-image-service/. ${DEPLOY_PATH}/app/
        chmod -R 755 ${DEPLOY_PATH}/app

        # 빠른 경로 확인
        test -f ${DEPLOY_PATH}/app/Dockerfile
        test -f ${DEPLOY_PATH}/app/requirements.txt

        echo "✅ 배포 디렉토리 준비 완료"
      '''
    }
  }

  stage('Build image') {
    steps {
      echo "🔨 ${SERVICE_NAME} Docker 이미지 빌드 중..."
      script { env.SHA = sh(script: 'git rev-parse --short HEAD', returnStdout: tr
ue).trim() }
      dir("${DEPLOY_PATH}/app") {
        sh '''
          set -e
          docker stop ${SERVICE_NAME} || true
          docker rm ${SERVICE_NAME} || true
          docker rmi ${SERVICE_NAME}:latest || true
          docker build -t ${SERVICE_NAME}:latest -t ${SERVICE_NAME}:${SHA}
.
          echo "✅ Docker 이미지 빌드 완료"
        '''
      }
    }
  }

  stage('Run container') {
    when {
      anyOf {
        branch ${BRANCH_NAME}  // deploy-image 브랜치에 merge 시 자동 배포
        expression { currentBuild.getBuildCauses('hudson.model.Cause$UserId
```

```
Cause').size() > 0 }  // Jenkins에서 수동 빌드 시
      }
    }
    steps {
      echo "🚀 ${SERVICE_NAME} 컨테이너 배포 중..."
      dir("${DEPLOY_PATH}/app") {
        sh '''
          set -e
          # .env 없으면 생성
          [ -f .env ] || cat > .env <<EOF
MODEL_PATH=./models/student_ft_best.keras
CLASSMAP_PATH=./models/class_mapping.json
THRESH_PATH=./models/thresholds.json
IMG_SIZE=192
TOPK=3
ALLOWED_MIME=image/jpeg,image/png,image/webp,image/avif
EOF
          docker run -d \
            --name ${SERVICE_NAME} \
            --restart unless-stopped \
            -p ${SERVICE_PORT}:${CONTAINER_PORT} \
            --env-file .env \
            ${SERVICE_NAME}:latest

          echo "✅ 컨테이너 실행 트리거 완료"
        '''
      }
    }
  }

  // ✅ 텍스트 서비스에서 사용한 Health Check 방식으로 변경
  stage('Health Check') {
    steps {
      echo "🏥 ${SERVICE_NAME} 헬스체크 수행 중..."
      sh '''
        set -eu
```

```
        echo "⏳ 서비스 시작 대기 중..."
        # 짧은 워밍업(필요 시 조정)
        sleep 10

        # 컨테이너가 running 상태인지 확인 (curl 대신 docker 상태로 체크)
        if docker ps --filter "name=${SERVICE_NAME}" --filter "status=runnin
g" --format "{{.Names}}" | grep -q "^${SERVICE_NAME}$"; then
          echo "✅ ${SERVICE_NAME} 컨테이너 정상 실행 중!"

          # Docker HEALTHCHECK를 정의해두었다면 상태도 함께 출력(없으면 none)
          health_status=$(docker inspect --format='{{.State.Health.Status}}' ${S
ERVICE_NAME} 2>/dev/null || echo "none")
          echo "Docker 헬스체크 상태: $health_status"

          echo "🌐 서비스 URL: http://$(curl -s ifconfig.me):${SERVICE_PORT}"
          echo "✅ 배포 완료 - 컨테이너가 정상 실행 중입니다"
          exit 0
        else
          echo "❌ 컨테이너 실행 실패"
          echo "---- ${SERVICE_NAME} 최근 로그 ----"
          docker logs ${SERVICE_NAME} --tail=100 || true
          exit 1
        fi
      '''
    }
  }
 }

 post {
  success {
   echo "🎉 ${SERVICE_NAME} 배포 성공!"
   sh '''
    echo "📊 배포 완료 상태:"
    docker ps --filter name=${SERVICE_NAME} --format "table {{.Names}}\\t
{{.Status}}\\t{{.Ports}}"
    echo "🌐 서비스 접근: http://$(curl -s ifconfig.me):${SERVICE_PORT}"
```

```
      '''
    }
    failure {
      echo "❌ ${SERVICE_NAME} 배포 실패!"
      sh '''
        echo "💥 실패 로그 수집 중..."
        if docker ps -a --filter name=${SERVICE_NAME} --format "{{.Names}}" |
grep -q "^${SERVICE_NAME}$"; then
          echo "🔍 ${SERVICE_NAME} 컨테이너 로그:"
          docker logs ${SERVICE_NAME} --tail=200 || true
        fi
        echo "🔍 시스템 리소스 상태:"
        df -h
        free -h
        echo "🔍 Docker 상태:"
        docker ps -a
      '''
    }
    always {
      sh '''
        echo "🧹 정리 작업 수행 중..."
        # (선택) .env 정리 — 이미지 서비스는 유지가 필요하면 주석 처리
        # rm -f ${DEPLOY_PATH}/app/.env || true

        docker image prune -f || true
        echo "✅ 정리 완료"
      '''
    }
  }
}
```

AI 텍스트 Jenkinsfile

- deploy-text 브랜치에 Merge 이벤트가 발생 시 배포 진행

```
pipeline {
    agent any

    environment {
        GITLAB_REPO = 'https://lab.ssafy.com/s13-bigdata-dist-sub1/S13P21A207.git'
        BRANCH_NAME = 'deploy-text'
        SERVICE_NAME = 'ai-text-service'
        SERVICE_PATH = 'Service/ai-text-service'
        DEPLOY_PATH = '/home/ubuntu/services/text'
        SERVICE_PORT = '8002'
    }

    stages {
        stage('Checkout') {
            steps {
                echo "🔄 ${SERVICE_NAME} 소스 체크아웃 중..."
                git branch: "${BRANCH_NAME}",
                    credentialsId: 'gitlab-token',
                    url: "${GITLAB_REPO}"
            }
        }

        stage('Prepare Deploy Directory') {
            steps {
                echo "📁 배포 디렉토리 준비 중..."
                sh '''
                    # 배포 디렉토리 생성
                    mkdir -p ${DEPLOY_PATH}
                    chmod -R 755 ${DEPLOY_PATH}

                    # 기존 파일 백업 및 정리
                    if [ -d "${DEPLOY_PATH}/backup" ]; then
                        rm -rf ${DEPLOY_PATH}/backup
                    fi
```

```
            if [ -d "${DEPLOY_PATH}/app" ]; then
                mv ${DEPLOY_PATH}/app ${DEPLOY_PATH}/backup
            fi

            echo "✅ 배포 디렉토리 준비 완료"
        '''
    }
}

stage('Copy Service Files') {
    steps {
        echo "📦 ${SERVICE_NAME} 파일 복사 중..."
        sh '''
            # 서비스 파일 복사
            cp -r ${SERVICE_PATH} ${DEPLOY_PATH}/app

            # 권한 설정
            chmod -R 755 ${DEPLOY_PATH}/app

            # 복사된 파일 확인
            echo "📋 복사된 파일 목록:"
            ls -la ${DEPLOY_PATH}/app/

            echo "✅ 파일 복사 완료"
        '''
    }
}

stage('Build Docker text') {
    steps {
        echo "🔨 ${SERVICE_NAME} Docker 이미지 빌드 중..."
        sh '''
            cd ${DEPLOY_PATH}/app

            # 기존 컨테이너 중지 및 제거
```

```
                docker stop ${SERVICE_NAME} || true
                docker rm ${SERVICE_NAME} || true

                # 기존 이미지 제거
                docker rmi ${SERVICE_NAME}:latest || true

                # Docker 이미지 빌드
                docker build -t ${SERVICE_NAME}:latest .

                echo "✅ Docker 이미지 빌드 완료"
            '''
        }
    }

    stage('Deploy Container') {
        when {
            anyOf {
                branch ${BRANCH_NAME}  // deploy-text 브랜치에 merge 시 자동 배포
                expression { currentBuild.getBuildCauses('hudson.model.Cause$UserIdCause').size() > 0 }  // Jenkins에서 수동 빌드 시
            }
        }
        steps {
            echo "🚀 ${SERVICE_NAME} 컨테이너 배포 중..."
            sh '''
                cd ${DEPLOY_PATH}/app

                # 환경변수 파일 생성 (필요시)
                if [ ! -f .env ]; then
                    cp .env.example .env || echo "SERVICE_NAME=${SERVICE_NAME}" > .env
                fi

                # Docker 컨테이너 실행
                docker run -d \
```

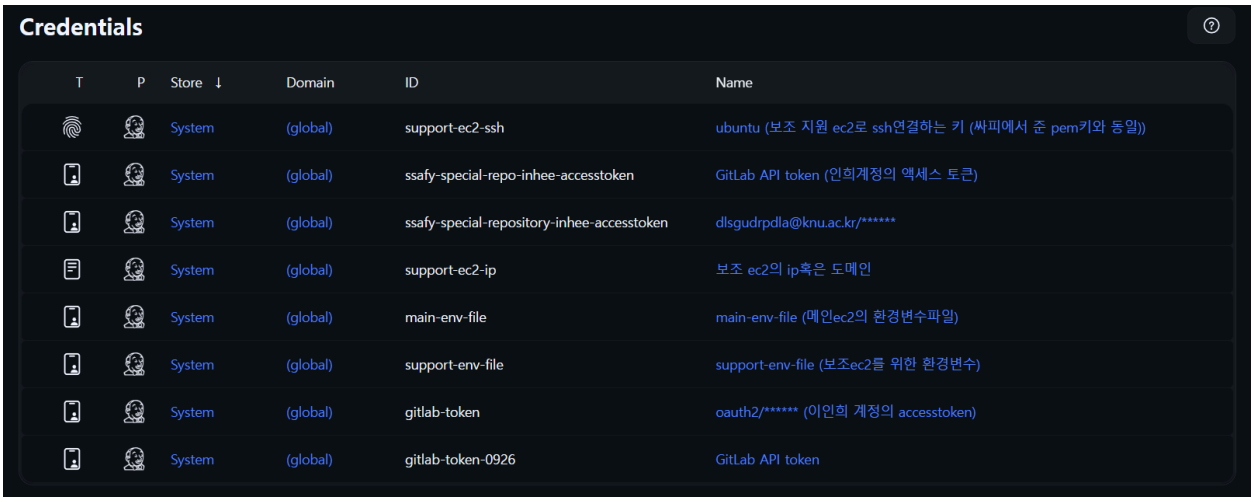
```
                --name ${SERVICE_NAME} \
                --restart unless-stopped \
                -p ${SERVICE_PORT}:8000 \
                --env-file .env \
                ${SERVICE_NAME}:latest

            echo "✅ 컨테이너 배포 완료"
          '''
        }
      }

      stage('Health Check') {
        steps {
          echo "🏥 ${SERVICE_NAME} 헬스체크 수행 중..."
          sh '''
            echo "⏳ 서비스 시작 대기 중..."
            sleep 30

            # Docker 컨테이너 상태 확인으로 헬스체크 대체
            if docker ps --filter name=${SERVICE_NAME} --filter status=runni
ng | grep -q ${SERVICE_NAME}; then
              echo "✅ ${SERVICE_NAME} 컨테이너 정상 실행 중!"

              # Docker 헬스체크 상태 확인 (있는 경우)
              health_status=$(docker inspect --format='{{.State.Health.Statu
s}}' ${SERVICE_NAME} 2>/dev/null || echo "none")
              echo "Docker 헬스체크 상태: $health_status"

              echo "🌐 서비스 URL: http://$(curl -s ifconfig.me):${SERVICE_P
ORT}"
              echo "✅ 배포 완료 - 컨테이너가 정상 실행 중입니다"
              exit 0
            else
              echo "❌ 컨테이너 실행 실패"
              docker logs ${SERVICE_NAME} --tail=50
              exit 1
```

```
                fi
            '''
        }
    }
}

post {
    success {
        echo "🎉 ${SERVICE_NAME} 배포 성공!"
        sh '''
            echo "📊 배포 완료 상태:"
            docker ps --filter name=${SERVICE_NAME} --format "table {{.Name
s}}\\t{{.Status}}\\t{{.Ports}}"
            echo "🌐 서비스 접근: http://$(curl -s ifconfig.me):${SERVICE_POR
T}"
        '''
    }

    failure {
        echo "❌ ${SERVICE_NAME} 배포 실패!"
        sh '''
            echo "💥 실패 로그 수집 중..."

            # Docker 로그 출력
            if docker ps -a --filter name=${SERVICE_NAME} --format {{.Name
s}} | grep -q ${SERVICE_NAME}; then
                echo "🔍 ${SERVICE_NAME} 컨테이너 로그:"
                docker logs ${SERVICE_NAME} --tail=100
            fi

            # 시스템 상태 확인
            echo "🔍 시스템 리소스 상태:"
            df -h
            free -h

            echo "🔍 Docker 상태:"
```

```
            docker ps -a
        '''
    }

    always {
        sh '''
            # 정리 작업
            echo "🧹 정리 작업 수행 중..."

            # 환경변수 파일 보안 삭제
            if [ -f "${DEPLOY_PATH}/app/.env" ]; then
                rm -f ${DEPLOY_PATH}/app/.env
            fi

            # 미사용 Docker 이미지 정리
            docker image prune -f || true

            echo "✅ 정리 완료"
        '''
    }
  }
}
```

Credentials 관리

빌드에 필요한 env 파일들을 저장해두고 배포 시 파일을 옮겨 서버에 올린다.



Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	support-ec2-ssh	ubuntu (보조 지원 ec2로 ssh연결하는 키 (싸피에서 준 pem키와 동일))
		System	(global)	ssafy-special-repo-inhee-accesstoken	GitLab API token (인희계정의 액세스 토큰)
		System	(global)	ssafy-special-repository-inhee-accesstoken	dlsgudrpdla@knu.ac.kr/******
		System	(global)	support-ec2-ip	보조 ec2의 ip혹은 도메인
		System	(global)	main-env-file	main-env-file (메인ec2의 환경변수파일)
		System	(global)	support-env-file	support-env-file (보조ec2를 위한 환경변수)
		System	(global)	gitlab-token	oauth2/****** (이인희 계정의 accesstoken)
		System	(global)	gitlab-token-0926	GitLab API token

- **GitLab**: gitlab의 프로젝트를 clone 해오기 위한 credentials
  - `gitlab-token-0926` : gitlab API 토큰
- **.env prod:** token 값 등 주요 값들 저장
  - `main-env-file` : 파일 형태

Gitlab 웹훅 설정

- 배포 : develop 브랜치
- AI 이미지 : deploy-image 브랜치
- AI 텍스트 : deploy-text 브랜치

젠킨스 플러그인 추가 설치

- GitLab
- SSH Agent
- Pipeline Graph View

배포 위한 파일 생성

Backend - Spring

1. **Main Dockerfile 생성**

```
FROM python:3.11-slim

ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    PIP_NO_CACHE_DIR=1 \
    TF_CPP_MIN_LOG_LEVEL=2

WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN python -m pip install --upgrade pip setuptools wheel
RUN pip install --no-cache-dir -r requirements.txt

COPY app ./app
COPY models ./models
COPY tests ./tests

# 텍스트 모델 환경변수
ENV MODEL_PATH=./models/final_multilabel_model/model.safetensors \
    CONFIG_PATH=./models/final_multilabel_model/config.json \
    TOKENIZER_PATH=./models/final_multilabel_model \
    VOCAB_PATH=./models/final_multilabel_model/vocab.txt \
    SPECIAL_TOKENS_PATH=./models/final_multilabel_model/special_token
s_map.json

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

2. **docker-compose.yml 파일 생성**

```
# Infra/main-ec2/docker-compose.main.yml
# version: '3.8'
#

services:
 # ================================================================
 # Jenkins 서비스 제거 (jenkins-standalone.sh 으로 사전 별도 실행 필요)
 # ================================================================

 fastapi-ai-image-service:
  build:
   context: ../../Service/ai-image-service
   dockerfile: Dockerfile
  container_name: main-ai-image
  ports:
   - "127.0.0.1:8001:8000" # ← 호스트 8001 → 컨테이너 8000
  environment:
   - MODEL_PATH=/app/models/image/student_ft_best.keras
   - CLASSMAP_PATH=/app/models/image/class_mapping.json
   - THRESH_PATH=/app/models/image/thresholds.json
   - REDIS_URL=redis://redis:6379
   - LOG_LEVEL=INFO
  volumes:
   - /home/ubuntu/app/models:/app/models
   - /home/ubuntu/app/logs/ai-image:/app/logs
  depends_on:
   - redis
  networks:
   - main-network
  restart: unless-stopped
  deploy:
   resources:
```

```
        limits:
          memory: 2G
          cpus: '2.0'

  fastapi-ai-text-service:
    build:
      context: ../../Service/ai-text-service
      dockerfile: Dockerfile
    container_name: main-ai-text
    ports:
      - "127.0.0.1:8002:8000" # ← 호스트 8002 → 컨테이너 8000
    environment:
      - AI_MODEL_PATH=/app/models
      - REDIS_URL=redis://redis:6379
      - LOG_LEVEL=INFO
    volumes:
      - /home/ubuntu/app/models:/app/models
      - /home/ubuntu/app/logs/ai-text:/app/logs
    depends_on:
      - redis
    networks:
      - main-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 2G
          cpus: '2.0'

  redis:
    image: redis:7-alpine
    container_name: main-redis
    command: redis-server --appendonly yes --maxmemory 512mb --max
memory-policy allkeys-lru
    ports:
      - "127.0.0.1:6380:6379"
```

```
    volumes:
      - redis-data:/data
    networks:
      - main-network
    restart: unless-stopped

  # extension-builder:
  #   build:
  #     context: ../../Front-End/SSAFY_PJT2    # Front-End 서비스 경로 (git 구조와 동일)
  #     dockerfile: Dockerfile                 # Front-End/SSAFY_PJT2/Dockerfile 사용
  #   container_name: main-extension-builder
  #   volumes:
  #     - extension-downloads:/var/www/html/downloads
  #     - ./extension-web:/var/www/html       # nginx에서 서빙할 정적 파일
  #   networks:
  #     - main-network
  #   restart: "no"                          # 빌드 완료 후 자동 종료
  #

  common-backend:
    build:
      context: ../../Service/common-backend
      dockerfile: Dockerfile
    container_name: main-backend
    ports:
      - "127.0.0.1:8080:8080"
      - "0.0.0.0:9092:9092"
    environment:
      - FASTAPI_IMAGE_URL=http://fastapi-ai-image-service:8000  # 컨테이너명이랑 일치
      - FASTAPI_TEXT_URL=http://fastapi-ai-text-service:8000  # 컨테이너명이랑 일치
      - SPRING_PROFILES_ACTIVE=production
      - REDIS_HOST=redis
```

```
      - REDIS_PORT=6380
      - USE_REDIS=true
      - SOCKETIO_HOST=0.0.0.0
      - SOCKETIO_PORT=9092
    depends_on:
      - redis
    networks:
      - main-network
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    container_name: main-nginx
    ports:
      - "80:80"
      - "443:443"
      - "9090:9090"
    volumes:
      - ./logs/nginx:/var/log/nginx
      - extension-downloads:/var/www/html/downloads
      - ./extension-web:/var/www/html/extension
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro  # nginx.conf 마운트
      # SSL 인증서 볼륨
      - /etc/letsencrypt:/etc/letsencrypt:ro
      - /var/www/certbot:/var/www/certbot
    depends_on:
      - fastapi-ai-image-service
      - fastapi-ai-text-service
      - common-backend
    networks:
      - main-network
    restart: unless-stopped

volumes:
  redis-data:
  extension-downloads:
```

```
networks:
  main-network:
    driver: bridge
```

3. **AI 이미지 Dockerfile**

```
# ai-image-service/Dockerfile
FROM python:3.11-slim

# 기본 환경
ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    PIP_NO_CACHE_DIR=1 \
    TF_CPP_MIN_LOG_LEVEL=2

WORKDIR /app

# Pillow 등 이미지 디코드 필수 라이브러리 (AVIF 지원 포함)
RUN apt-get update && apt-get install -y --no-install-recommends \
    libjpeg62-turbo-dev zlib1g-dev libpng-dev libavif-dev \
    && rm -rf /var/lib/apt/lists/*

# 파이썬 패키지
COPY requirements.txt .
RUN python -m pip install --upgrade pip setuptools wheel
RUN pip install --no-cache-dir -r requirements.txt

# 앱 소스 & 모델(간단히 이미지 안에 포함; 운영에서는 볼륨 마운트 권장)
COPY app ./app
COPY models ./models

# 기본 환경값
ENV MODEL_PATH=./models/student_ft_best.keras \
```

```
    CLASSMAP_PATH=./models/class_mapping.json \
    THRESH_PATH=./models/thresholds.json \
    IMG_SIZE=192 \
    TOPK=3 \
    ALLOWED_MIME="image/jpeg,image/png,image/webp,image/avif"

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

4. **AI 텍스트 Dockerfile**

```
FROM python:3.11-slim

ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    PIP_NO_CACHE_DIR=1 \
    TF_CPP_MIN_LOG_LEVEL=2

WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN python -m pip install --upgrade pip setuptools wheel
RUN pip install --no-cache-dir -r requirements.txt

COPY app ./app
COPY models ./models
COPY tests ./tests

# 텍스트 모델 환경변수
ENV MODEL_PATH=./models/final_multilabel_model/model.safetensors \
    CONFIG_PATH=./models/final_multilabel_model/config.json \
```

```
    TOKENIZER_PATH=./models/final_multilabel_model \
    VOCAB_PATH=./models/final_multilabel_model/vocab.txt \
    SPECIAL_TOKENS_PATH=./models/final_multilabel_model/special_token
s_map.json

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