포팅매뉴얼

0. 버전 정보

Skill	Version
Python	3.10
FastAPI	0.115.12
SpringBoot	3.4.4
Tomcat	10.1.39
JVM(JDK)	17
intellij IDEA	2024.3.1.1
MySQL	8.0.41
Jenkins	2.492.3
nginx	1.27.5
ubuntu	20.04
docker	28.1.1
docker compose	2.35.1
Node.js	22.15.1
Yarn	1.22.22
Next.js	15.3.1
React	19.0.0
TypeScript	5.8.3

1. 포트 설정

EC2 서버

서비스	포트	설명
SSH	22	원격 접속
Nginx	80, 443	웹 서버 HTTP, HTTPS
MySQL	3306	개발용 데이터베이스 접근
프론트엔드	3000	Next.js 배포용 서버 (Blue-Green 배포)
프론트엔드	3001	Next.js 개발용 서버
백엔드	8080	Spring Boot 배포용 서버 (Blue-Green 배포)
백엔드	8081	Spring Boot 개발용 서버
Jenkins	8080:8080 50000:50000	CI/CD 관리

GCP 서버

서비스	포트	설명
SSH	22	원격 접속
Nginx	80, 443	웹 서버 HTTP, HTTPS
FASTAPI	8001	AI기능을 위한 서버
Jenkins	8080:8080 50000:50000	CI/CD 관리

2. 네트워크 설정

```
docker network create app-network
```

3. Jenkins 컨테이너 성

3-1. EC2 서버

설정 파일 작성

1. docker-compose.yml

```
services:
  jenkins:
    # image: jenkins/jenkins:lts
    build:
      context: ./jenkins
    user: root
    container_name: jenkins
    ports:
      - "8080:8080"
      - "50000:50000"
    group_add:
      - "${DOCKER_GID}"
    volumes:
      - jenkins_jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/ubuntu/nginx/upstreams:/etc/nginx/upstreams
      - /home/ubuntu/logs:/logs
    command: >
      bash -c "apt-get update && apt-get install -y docker.io && /usr/bin/tini
    environment:
      - TZ=Asia/Seoul
    dns:
      - 1.1.1.1
      - 8.8.8.8
    restart: unless-stopped

volumes:
  jenkins_jenkins_home:
      external: true
  es_data:

networks:
  default:
    external: true
    name: app-network
```

2. .env 파일

```
DOCKER_GID=999
```

3. Dockerfile 작성
   - docker compose 설치용

```
FROM jenkins/jenkins:lts

USER root

# Docker 설치
RUN apt-get update && \
    apt-get install -y docker.io && \
    getent group docker || groupadd docker && \
    usermod -aG docker jenkins

# Docker Compose v2 설치
RUN mkdir -p /usr/libexec/docker/cli-plugins && \
    curl -SL https://github.com/docker/compose/releases/download/v2.35.1
    chmod +x /usr/libexec/docker/cli-plugins/docker-compose

# PATH 등록
ENV PATH="/usr/libexec/docker/cli-plugins:$PATH"

USER jenkins
```

컨테이너 띄우기

```
docker compose up -d --build
```

3-2. GCP 서버

설정 파일 작성

1. docker-compose.yml

```
services:
  jenkins:
    image: jenkins/jenkins:lts
    user: root
    container_name: jenkins
    ports:
      - "8080:8080"
      - "50000:50000"
    group_add:
      - "${DOCKER_GID}"
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      # - /home/ubuntu/nginx/upstreams:/etc/nginx/upstreams
    command: >
      bash -c "apt-get update && apt-get install -y docker.io && /usr/bin/tini
    environment:
      - TZ=Asia/Seoul
    dns:
      - 1.1.1.1
      - 8.8.8.8
    restart: unless-stopped

volumes:
  jenkins_home:

networks:
  default:
    external: true
    name: app-network
```

2. .env 파일

```
DOCKER_GID=999
```

컨테이너 띄우기

```
docker compose up -d --build
```

4. docker-compose.yml 파일

4-1. EC2 서버

개발용 (docker-compose.dev.yml)

```
services:
  frontend:
    build:
      context: ./FRONTEND
      dockerfile: Dockerfile
    # 빌드한 이미지의 이름 설정
    image: ${NEXTJS_DEV_IMAGE_NAME}
    container_name: ${NEXTJS_DEV_CONTAINER_NAME}
    restart: always
    expose:
      - "${NEXTJS_DEV_CONTAINER_PORT}"
    # 운영환경 명시 : 항상 production으로 설정(성능 저하 방지)
    environment:
    - NODE_ENV=production
    - PORT=${NEXTJS_DEV_CONTAINER_PORT}
    - HOSTNAME=0.0.0.0
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3001"]
      interval: 30s # 검사 간 간격
      timeout: 10s # 검사 최대 실행 시간
      retries: 3 # 실패 시 재시도 횟수 (모두 실패 시 unhealthy)
      start_period: 30s # 컨테이너 시작 후 검사 시작까지 대기 시간

  backend:
    # Dockerfile로 백엔드 프로젝트를 이미지로 빌드
    build:
      context: ./BACKEND
      dockerfile: Dockerfile
```

```
    # 빌드한 이미지의 이름 설정정
    image: ${SPRING_DEV_IMAGE_NAME}
    container_name: ${SPRING_DEV_CONTAINER_NAME}
    restart: always
    environment:
      - SERVER_PORT=${SPRING_DEV_CONTAINER_PORT}
      - LOG_FILENAME=dev.log
    expose:
      - "${SPRING_DEV_CONTAINER_PORT}"
    volumes:
      - /home/ubuntu/aws:/app/aws
      - /home/ubuntu/logs:/logs
    depends_on:
      mysql:
        condition: service_healthy   # 에러 방지를 위해 mysql healthcheck 통과 후 시

  mysql:
    image: mysql:8.0.41-oracle
    container_name: ${MYSQL_CONTAINER_NAME}
    restart: always
    env_file:
      - ./mysql/.env.mysql.dev
    ports:
      - "${MYSQL_CONTAINER_PORT}:${MYSQL_CONTAINER_PORT}"
    volumes:
      - /home/ubuntu/docker-mysql/filmmoa_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping"]
      interval: 5s # 5초 간격으로 체크
      retries: 10 # 10번까지 재시도

networks:
  default:
    external: true
    name: app-network
```

배포용 (블루/그린 무중단 배포, docker-compose.master.blue.yml/ docker-compose.master.green.yml)

```
services:
  frontend-master-blue:
    build:
      context: ./FRONTEND
      dockerfile: Dockerfile
    # 빌드한 이미지의 이름 설정
    image: ${NEXTJS_MASTER_BLUE
    container_name: ${NEXTJS_MAS
    restart: always
    expose:
      - "${NEXTJS_MASTER_BLUE_CO
    # 운영환경 명시 : 항상 production으
    environment:
    - NODE_ENV=production
    - PORT=${NEXTJS_MASTER_BLU
    - HOSTNAME=0.0.0.0
    healthcheck:
      test: ["CMD", "curl", "-f", "http://
      interval: 30s # 검사 간 간격
      timeout: 10s # 검사 최대 실행 시간
      retries: 3 # 실패 시 재시도 횟수 (모
      start_period: 30s # 컨테이너 시작

  backend-master-blue:
    # Dockerfile로 백엔드 프로젝트를 이
    build:
      context: ./BACKEND
      dockerfile: Dockerfile
    # 빌드한 이미지의 이름 설정정
    image: ${SPRING_MASTER_BLUE
    container_name: ${SPRING_MAS
    restart: always
    environment:
      - SERVER_PORT=${SPRING_MA
      - LOG_FILENAME=master.log
    expose:
      - "${SPRING_MASTER_BLUE_CO
```

```
services:
  frontend-master-green:
    build:
      context: ./FRONTEND
      dockerfile: Dockerfile
    # 빌드한 이미지의 이름 설정
    image: ${NEXTJS_MASTER_GREE
    container_name: ${NEXTJS_MAS
    restart: always
    expose:
      - "${NEXTJS_MASTER_GREEN_C
    # 운영환경 명시 : 항상 production으
    environment:
    - NODE_ENV=production
    - PORT=${NEXTJS_MASTER_GRE
    - HOSTNAME=0.0.0.0
    healthcheck:
      test: ["CMD", "curl", "-f", "http://
      interval: 30s # 검사 간 간격
      timeout: 10s # 검사 최대 실행 시간
      retries: 3 # 실패 시 재시도 횟수 (모
      start_period: 30s # 컨테이너 시작

  backend-master-green:
    # Dockerfile로 백엔드 프로젝트를 이
    build:
      context: ./BACKEND
      dockerfile: Dockerfile
    # 빌드한 이미지의 이름 설정정
    image: ${SPRING_MASTER_GREE
    container_name: ${SPRING_MAS
    restart: always
    environment:
      - SERVER_PORT=${SPRING_MA
      - LOG_FILENAME=master.log
    expose:
      - "${SPRING_MASTER_GREEN_C
```

```
    volumes:
      - /home/ubuntu/aws:/app/aws
      - /home/ubuntu/logs:/logs

networks:
  default:
    external: true
    name: app-network
```

```
    volumes:
      - /home/ubuntu/aws:/app/aws
      - /home/ubuntu/logs:/logs

networks:
  default:
    external: true
    name: app-network
```

AI 기능용 docker-compose.ai.yml

```
services:
  ai-dev-app:
    build:
      context: ./AI
      dockerfile: Dockerfile
    image: ${FASTAPI_DEV_IMAGE_NAME}:latest
    container_name: ${FASTAPI_DEV_CONTAINER_NAME}
    environment:
      - PORT=${FASTAPI_DEV_CONTAINER_PORT}   # FastAPI에 전달
    expose:
      - "${FASTAPI_DEV_CONTAINER_PORT}"
    restart: always

networks:
  default:
    external: true
    name: app-network
```

4-2. GCP 서버

FastAPI(AI 얼굴분류 기능) docker-compose.ai.dev.yml

```
services:
```

```
  ai-dev-app:
    build:
      context: ./AI
      dockerfile: Dockerfile
    image: ${FASTAPI_DEV_IMAGE_NAME}:latest
    container_name: ${FASTAPI_DEV_CONTAINER_NAME}
    environment:
      - PORT=${FASTAPI_DEV_CONTAINER_PORT}   # FastAPI에 전달
    expose:
      - "${FASTAPI_DEV_CONTAINER_PORT}"
    restart: always
networks:
  default:
    external: true
    name: app-network
```

5. Jenkins 환경변수 및 Excute Shell 설정

5-1. EC2 서버

Job 구성

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	
✓	☀	filmmoa	5 hr 51 min #365	—	2 min 7 sec	▷
✓	☀	filmmoa-master	3 hr 47 min #64	1 day 23 hr #59	2 min 14 sec	▷
✗	☁	filmmoa-rollback	8 days 0 hr #5	1 day 23 hr #6	21 sec	▷
✓	☀	filmmoa-switch-traffic	3 hr 44 min #38	—	1 sec	▷

Jenkins 환경 변수

```
--------- EC2 Server ---------
# 공통
DOCKER_NETWORK_NAME=app-network
DOCKER_COMPOSE_DEV_FILE=docker-compose.dev.yml
DOCKER_COMPOSE_MASTER_FILE=docker-compose.master.yml
DOCKER_COMPOSE_MASTER_BLUE_FILE=docker-compose.master.blue.yml
```

```
DOCKER_COMPOSE_MASTER_GREEN_FILE=docker-compose.master.green.y

# dev - nextjs
NEXTJS_DEV_SERVICE_NAME=frontend-dev
NEXTJS_DEV_IMAGE_NAME=nextjs-dev
NEXTJS_DEV_CONTAINER_NAME=nextjs-dev-container
NEXTJS_DEV_CONTAINER_PORT=3001
NEXTJS_DEV_HOST_PORT=3001

# dev - spring
SPRING_DEV_SERVICE_NAME=backend-dev
SPRING_DEV_IMAGE_NAME=spring-dev
SPRING_DEV_CONTAINER_NAME=spring-dev-container
SPRING_DEV_CONTAINER_PORT=8081
SPRING_DEV_HOST_PORT=8081

# master - nextjs
NEXTJS_MASTER_SERVICE_NAME=frontend-master
NEXTJS_MASTER_IMAGE_NAME=nextjs-master
NEXTJS_MASTER_CONTAINER_NAME=nextjs-master-container
NEXTJS_MASTER_CONTAINER_PORT=3000
NEXTJS_MASTER_HOST_PORT=3000
SPRING_MASTER_HOST_PORT=8080

# master - spring
SPRING_MASTER_SERVICE_NAME=backend-master
SPRING_MASTER_IMAGE_NAME=spring-master
SPRING_MASTER_CONTAINER_NAME=spring-master-container
SPRING_MASTER_CONTAINER_PORT=8080

# master(blue) - nextjs
NEXTJS_MASTER_BLUE_SERVICE_NAME=frontend-master-blue
NEXTJS_MASTER_BLUE_IMAGE_NAME=nextjs-master:blue-
NEXTJS_MASTER_BLUE_CONTAINER_NAME=nextjs-master:blue-container
NEXTJS_MASTER_BLUE_CONTAINER_PORT=3000
NEXTJS_MASTER_BLUE_HOST_PORT=3000

# master(blue) - spring
```

```
SPRING_MASTER_BLUE_SERVICE_NAME=backend-master-blue
SPRING_MASTER_BLUE_IMAGE_NAME=spring-master:blue-
SPRING_MASTER_BLUE_CONTAINER_NAME=spring-master-blue-container
SPRING_MASTER_BLUE_CONTAINER_PORT=8080

# master(green) - nextjs
NEXTJS_MASTER_GREEN_SERVICE_NAME=frontend-master-green
NEXTJS_MASTER_GREEN_IMAGE_NAME=nextjs-master:green-
NEXTJS_MASTER_GREEN_CONTAINER_NAME=nextjs-master-green-containe
NEXTJS_MASTER_GREEN_CONTAINER_PORT=3000
NEXTJS_MASTER_GREEN_HOST_PORT=3000

# master(green) - spring
SPRING_MASTER_GREEN_SERVICE_NAME=backend-master-green
SPRING_MASTER_GREEN_IMAGE_NAME=spring-master:green-
SPRING_MASTER_GREEN_CONTAINER_NAME=spring-master-green-containe
SPRING_MASTER_GREEN_CONTAINER_PORT=8080

# mysql
MYSQL_CONTAINER_PORT=3306
MYSQL_CONTAINER_NAME=mysql-dev-container



--------- GCP Server ---------
# 공통
DOCKER_NETWORK_NAME=app-network
DOCKER_COMPOSE_DEV_FILE=docker-compose.ai.dev.yml
DOCKER_COMPOSE_MASTER_FILE=docker-compose.ai.master.yml

FASTAPI_DEV_IMAGE_NAME=fastapi-dev
FASTAPI_DEV_CONTAINER_NAME=fastapi-dev-container
FASTAPI_DEV_CONTAINER_PORT=8001

FASTAPI_MASTER_IMAGE_NAME=fastapi-master
FASTAPI_MASTER_CONTAINER_NAME=fastapi-master-container
FASTAPI_MASTER_CONTAINER_PORT=8000
```

Secret 파일

1. .env.dev (개발용 Next.js 서버 환경 설정 파일)

```
NEXT_PUBLIC_API_URL=dev_api_url

# 카카오
NEXT_PUBLIC_KAKAO_CLIENT_ID=kakao_client_id
NEXT_PUBLIC_KAKAO_REDIRECT_URI=dev_kakao_redirect_uri
```

2. .env.prod (배포용 Next.js 서버 환경 설정 파일)

```
NEXT_PUBLIC_API_URL=master_api_url

# 카카오
NEXT_PUBLIC_KAKAO_CLIENT_ID=kakao_client_id
NEXT_PUBLIC_KAKAO_REDIRECT_URI=master_kakao_redirect_uri

NEXT_PUBLIC_GOOGLE_TAG_MANAGER_ID=google_tag_manager_id
```

3. private_key.pem (cloudfront 접근용 키)

```
-----BEGIN RSA PRIVATE KEY-----
....
private_key
....
-----END RSA PRIVATE KEY-----
```

4. application-secret.yml (배포용 springboot 서버 환경 설정 파일)

```
BUCKET_NAME: bucket_name

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://{mysql_container_name}:3306/{db_name}?serverTime
    username: mysql_username
    password: mysql_password
```

```
  jpa:
    database-platform: org.hibernate.dialect.MySQLDialect
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        use_sql_comments: true
        dialect: org.hibernate.dialect.MySQLDialect

  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: kakao_client_id
            client-secret: kakao_client_secret
            redirect-uri: kakao_redirect_uri
            authorization-grant-type: grant_type
            client-authentication-method: kakao_client_authentication_method
            client-name: kakao_client_name
            scope:
              - account_email
        provider:
          kakao:
            authorization-uri: kakao_authorization_uri
            token-uri: kakao_token_uri
            user-info-uri: kakao_user_info_uri
            user-name-attribute: id

kakao:
    logout-redirect-uri: kakao_logout_redirect_uri

jwt:
  secret: jwt_secret
  access:
    expiration: jwt_access_expiration
```

```
  refresh:
    expiration: jwt_refresh_expiration

cloud:
  aws:
    credentials:
      access-key: aws_access_key
      secret-key: aws_secret_key
    region:
      static: cloudfront_region_static
    s3:
      bucket: s3_bucket_name

cloudfront:
  domain: aws_cloudfront_domain
  keyPairId: aws_cloudfront_key_pair_id
  privateKeyPath: aws_cloudfront_private_key_path

cookie:
  domain: .film-moa.com
  path: /
  same-site: Strict
  http-only: true
  secure: ${COOKIE_SECURE:true}
```

5. application-secret-dev.yml (개발용 springboot 서버 환경 설정 파일)

```
BUCKET_NAME: bucket_name

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://{mysql_container_name}:3306/{db_name}?serverTime
    username: mysql_username
    password: mysql_password

  jpa:
    database-platform: org.hibernate.dialect.MySQLDialect
```

```
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        use_sql_comments: true
        dialect: org.hibernate.dialect.MySQLDialect

  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: kakao_client_id
            client-secret: kakao_client_secret
            redirect-uri: kakao_redirect_uri
            authorization-grant-type: grant_type
            client-authentication-method: kakao_client_authentication_method
            client-name: kakao_client_name
            scope:
              - account_email
        provider:
          kakao:
            authorization-uri: kakao_authorization_uri
            token-uri: kakao_token_uri
            user-info-uri: kakao_user_info_uri
            user-name-attribute: id

kakao:
    logout-redirect-uri: kakao_logout_redirect_uri

jwt:
  secret: jwt_secret
  access:
    expiration: jwt_access_expiration
  refresh:
    expiration: jwt_refresh_expiration
```

```
cloud:
  aws:
    credentials:
      access-key: aws_access_key
      secret-key: aws_secret_key
    region:
      static: cloudfront_region_static
    s3:
      bucket: s3_bucket_name

cloudfront:
  domain: aws_cloudfront_domain
  keyPairId: aws_cloudfront_key_pair_id
  privateKeyPath: aws_cloudfront_private_key_path

cookie:
  domain: .film-moa.com
  path: /
  same-site: Strict
  http-only: true
  secure: ${COOKIE_SECURE:true}
```

6. .env.mysql.dev (개발용 mysql 서버 환경 설정)

```
MYSQL_ROOT_PASSWORD=mysql_root_password
MYSQL_DATABASE=mysql_database
MYSQL_USER=mysql_user
MYSQL_PASSWORD=mysql_password
```

Jenkins Excute Shell

1. filmmoa (개발용 서버 CI/CD 구축용 Job)

```
#!/bin/bash

# 경로 생성
```

```
mkdir -p ./BACKEND/src/main/resources/aws/cloudfront
mkdir -p ./mysql

echo "시크릿파일 복사 및 권한 활성화..."
cp $SPRING_SECRET ./BACKEND/src/main/resources/application-secret.yml
cp $CLOUDFRONT_KEY ./BACKEND/src/main/resources/aws/cloudfront/priva
cp $NEXTJS_ENV_DEV_FILE ./FRONTEND/.env.dev
cp $NEXTJS_ENV_PROD_FILE ./FRONTEND/.env.prod
cp $MYSQL_ENV_DEV_FILE ./mysql/.env.mysql.dev
chmod 644 ./BACKEND/src/main/resources/application-secret.yml
chmod 644 ./BACKEND/src/main/resources/aws/cloudfront/private_key.pem
chmod 644 ./FRONTEND/.env.dev
chmod 644 ./FRONTEND/.env.prod
chmod 644 ./mysql/.env.mysql.dev

echo "💡 개발용 .env 적용..."
cp ./FRONTEND/.env.dev ./FRONTEND/.env

# 기존 컨테이너·로컬 이미지·orphan 컨테이너 정리 (볼륨·외부 네트워크 유지)
echo "🧹 기존 컨테이너 및 이미지 정리..."
docker compose -f $DOCKER_COMPOSE_DEV_FILE down --rmi local --remov

# external 네트워크(app-network) 존재 확인 및 생성
echo "🔄 Docker 네트워크 확인..."
if ! docker network ls | grep -q "app-network"; then
    echo "🚀 네트워크(app-network) 생성..."
    docker network create app-network
fi

# 서비스 빌드 및 재시작
echo "서비스 재빌드 및 재시작..."
docker compose -f $DOCKER_COMPOSE_DEV_FILE up -d --build

docker exec nginx nginx -s reload

# 시크릿 파일 삭제
echo "🧹 시크릿 파일 삭제..."
```

```
rm -f "./BACKEND/src/main/resources/application-secret.yml"
rm -f "./mysql/.env.mysql.dev"
```

2. filmmoa-master (배포용 서버 CI/CD 구축용 Job)

```
#!/bin/bash

# 경로 생성
mkdir -p ./BACKEND/src/main/resources/aws/cloudfront
echo "시크릿파일 복사 및 권한 활성화..."
cp $SPRING_SECRET ./BACKEND/src/main/resources/application-secret.yml
cp $CLOUDFRONT_KEY ./BACKEND/src/main/resources/aws/cloudfront/priva
cp $NEXTJS_ENV_DEV_FILE ./FRONTEND/.env.dev
cp $NEXTJS_ENV_PROD_FILE ./FRONTEND/.env.prod
chmod 644 ./BACKEND/src/main/resources/application-secret.yml
chmod 644 ./BACKEND/src/main/resources/aws/cloudfront/private_key.pem
chmod 644 ./FRONTEND/.env.dev
chmod 644 ./FRONTEND/.env.prod

echo "💡 배포용 .env 적용..."
cp ./FRONTEND/.env.prod ./FRONTEND/.env

# 1. 다음 슬롯 결정 ————————————————————————————————————————
if grep -q "nextjs-master-blue-container" /etc/nginx/upstreams/upstream-fro
  NEXT_SLOT="green"
  COMPOSE_FILE=$DOCKER_COMPOSE_MASTER_GREEN_FILE
  NEXTJS_CONTAINER=$NEXTJS_MASTER_GREEN_CONTAINER_NAME
  SPRING_CONTAINER=$SPRING_MASTER_GREEN_CONTAINER_NAME
else
  NEXT_SLOT="blue"
  COMPOSE_FILE=$DOCKER_COMPOSE_MASTER_BLUE_FILE
  NEXTJS_CONTAINER=$NEXTJS_MASTER_BLUE_CONTAINER_NAME
  SPRING_CONTAINER=$SPRING_MASTER_BLUE_CONTAINER_NAME
fi

echo "🎯 다음 슬롯: $NEXT_SLOT ($COMPOSE_FILE)"
echo "🎯 컨테이너: $NEXTJS_CONTAINER ,$SPRING_CONTAINER"
```

```
# 2. Docker 네트워크 확인 ──────────────────────────────────────────
echo "🔄 Docker 네트워크 확인..."
if ! docker network ls | grep -q "app-network"; then
    echo "🚀 네트워크(app-network) 생성..."
    docker network create app-network
fi


# 3. 다음 슬롯에 컨테이너 up & build ──────────────────────────────────
echo "🚀 새 버전 컨테이너 빌드 및 기동..."
docker compose -f "$COMPOSE_FILE" up -d --build


# 4. Health Check ──────────────────────────────────────────────────
echo "🩺 헬스체크 중..."
for i in {1..10}; do
  if curl -sf http://$NEXTJS_CONTAINER:3000 && curl -sf http://$SPRING_CON
    echo "✅ 헬스체크 통과"
    break
  fi
  echo "⏳ 대기중 ($i/10)..."
  sleep 10
  if [ $i -eq 10 ]; then
    echo "❌ 헬스체크 실패. 이전 상태 유지."
    docker compose -f "$COMPOSE_FILE" down
    exit 1
  fi
done


# 5. 시크릿 파일 삭제 ──────────────────────────────────────────────────
echo "🧹 시크릿 파일 삭제..."
rm -f "./BACKEND/src/main/resources/application-secret.yml"
```

3. filmmoa-rollback (배포용 서버 빌드 실패 시 이전 이미지 rollback용 Job)

```
#!/bin/bash

# 현재 슬롯 파악
if grep -q "nextjs-master-blue-container" /etc/nginx/upstreams/upstream-fro
  SRC="blue"
  OLD="green"
else
  SRC="green"
  OLD="blue"
fi

echo "🔁 롤백 실행: $SRC → $OLD"

# 1. 이전 슬롯 컨테이너가 살아있는지 확인
echo "🧪 롤백 슬롯 컨테이너 상태 확인..."
if docker ps -q -f name=spring-master-${OLD}-container | grep -q .; then
  echo "✅ 이전 슬롯 컨테이너가 이미 실행 중임. 재기동 생략"
else
  echo "🟡 이전 슬롯 컨테이너가 꺼져 있음. 재기동 시도 중..."
  cd ../filmmoa-master
  docker compose -f docker-compose.master.${OLD}.yml up -d --no-build
  sleep 5  # nginx가 연결할 수 있도록 약간의 텀
fi

# 2. nginx upstream 설정 교체 (이제 안전)
echo "🔄 트래픽 전환: $SRC → $OLD"
cp /etc/nginx/upstreams/${OLD}-frontend.conf /etc/nginx/upstreams/upstrea
cp /etc/nginx/upstreams/${OLD}-backend.conf  /etc/nginx/upstreams/upstrea

# 3. nginx reload
docker exec nginx nginx -s reload


echo "롤백 완료"
```

4. filmmoa-switch-traffic (blue/green 무중단 배포 구현 시 nginx 설정파일 수정을 위한 Job)

```
#!/bin/bash

if grep -q "nextjs-master-blue-container" /etc/nginx/upstreams/upstream-fro
  SRC="green"
  OLD="blue"
else
  SRC="blue"
  OLD="green"
fi

# 현재 트래픽 방향을 전환
echo "🔄 트래픽 전환: $OLD → $SRC"
cp /etc/nginx/upstreams/${SRC}-frontend.conf /etc/nginx/upstreams/upstrea
cp /etc/nginx/upstreams/${SRC}-backend.conf  /etc/nginx/upstreams/upstrea

docker exec nginx nginx -s reload

# 3. 실제 컨테이너가 살아있는지 검증
echo "🧪 ${OLD} 컨테이너 상태 확인..."
if docker ps -q -f name=spring-master-${OLD}-container | grep -q .; then
  echo "🧹 이전 슬롯 종료: ${OLD}"
  cd ../filmmoa-master
  docker compose -f docker-compose.master.${OLD}.yml down
else
  echo "⚠️ 이전 슬롯(${OLD}) 컨테이너가 이미 종료되어 있음. down 생략."
fi
```

5-2. GCP 서버

환경 변수

```
# 공통
DOCKER_NETWORK_NAME=app-network
DOCKER_COMPOSE_DEV_FILE=docker-compose.ai.dev.yml
DOCKER_COMPOSE_MASTER_FILE=docker-compose.ai.master.yml

FASTAPI_DEV_IMAGE_NAME=fastapi-dev
FASTAPI_DEV_CONTAINER_NAME=fastapi-dev-container
FASTAPI_DEV_CONTAINER_PORT=8001
```

Jenkins ExcuteShell

```
#!/bin/bash

# 기존 컨테이너·로컬 이미지·orphan 컨테이너 정리 (볼륨·외부 네트워크 유지)
echo "🧹 기존 컨테이너 및 이미지 정리..."
docker compose -f $DOCKER_COMPOSE_DEV_FILE down --rmi local --remov

# external 네트워크(app-network) 존재 확인 및 생성
echo "🔄 Docker 네트워크 확인..."
if ! docker network ls | grep -q "app-network"; then
    echo "🚀 네트워크(app-network) 생성..."
    docker network create app-network
fi

# 서비스 빌드 및 재시작
echo "서비스 재빌드 및 재시작..."
docker compose -f $DOCKER_COMPOSE_DEV_FILE up -d --build
```

6. Nginx 설정

6-1. EC2 서버

파일 구조

```
/home/ubuntu/infra/nginx
├── conf.d/
│   ├── dev.conf              # 개발용 서버 리버스프록시 설정 파일
│   └── master.conf           # 배포용 서버 리버스프록시 설정 파일
├── upstreams/                # blue/green 무중단 배포 구현용
│   ├── blue-backend.conf     # blue 배포 시 upstream-backend로 붙여넣을 파일
│   ├── blue-frontend.conf    # blue 배포 시 upstream-frontend로 붙여넣을 파일
│   ├── green-backend.conf    # green 배포 시 upstream-backend로 붙여넣을 파
│   ├── green-frontend.conf   # green 배포 시 upstream-frontend로 붙여넣을 파
│   ├── upstream-backend.conf  # 현재 nginx의 backend 트래픽 방향 설정 파일
│   └── upstream-frontend.conf # 현재 nginx의 frontend 트래픽 방향 설정 파일
└── nginx.conf                # nginx 기본 설정 파일
```

설정 파일 작성

1. dev.conf

```
upstream frontend-dev {
  server nextjs-dev-container:3001;
}

upstream backend-dev {
  server spring-dev-container:8081;
}

# 1. HTTP(80포트) 요청은 모두 HTTPS로 리다이렉트
server {
  listen 80;
  server_name dev.film-moa.com;

  return 308 https://$host$request_uri;
}

# 2. HTTPS(443포트) 요청을 처리하는 서버 블록
server {
  listen 443 ssl;
```

```
    server_name dev.film-moa.com;

    client_max_body_size 20M; # 최대 업로드 크기 설정

    ssl_certificate /etc/letsencrypt/live/dev.film-moa.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dev.film-moa.com/privkey.pem;

    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
      proxy_pass http://frontend-dev;
      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/ {
      proxy_pass http://backend-dev;
      client_max_body_size 20M; # 최대 업로드 크기 설정
      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;
    }
  }
```

2. master.conf

```
include /etc/nginx/upstreams/upstream-frontend.conf;
include /etc/nginx/upstreams/upstream-backend.conf;

  # 1. HTTP(80포트) 요청은 모두 HTTPS로 리다이렉트
  server {
    listen 80;
    server_name www.film-moa.com film-moa.com;
```

```
  return 308 https://$host$request_uri;
}

# 2. HTTPS(443포트) 요청을 처리하는 서버 블록
server {
  listen 443 ssl;
  server_name www.film-moa.com film-moa.com;

  client_max_body_size 20M; # 최대 업로드 크기 설정

  ssl_certificate /etc/letsencrypt/live/www.film-moa.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/www.film-moa.com/privkey.pem;

  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

  location / {
    # proxy_pass http://frontend-master;
    proxy_pass http://frontend_app;
    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/ {
    # proxy_pass http://backend-master;
    proxy_pass http://backend_app;
    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;
  }
}
```

3. blue-backend.conf

```
upstream backend_app  {
  server spring-master-blue-container:8080;
}
```

4. blue-frontend.conf

```
upstream frontend_app  {
  server nextjs-master-blue-container:3000;
}
```

5. green-backend.conf

```
upstream backend_app  {
  server spring-master-green-container:8080;
}
```

6. green-frontend.conf

```
upstream frontend_app  {
  server nextjs-master-green-container:3000;
}
```

7. upstream-backend.conf, upstream-backend.conf
   - blue, green 배포 순서에 따라 덮어씌워짐
   - **blue 차례**

     blue-backend.conf → upstream-backend.conf

     blue-frontend.conf → upstream-frontend.conf
   - **green 차례**

     green-backend.conf → upstream-backend.conf

     green-frontend.conf → upstream-frontend.conf

Nginx 컨테이너 구동

```
docker run -d \
  --name nginx \
  --network app-network \
  -p 80:80 \
  -p 443:443 \
  -v /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  nginx:latest
```

6-2. GCP 서버

파일 구조

```
/home/ubuntu/infra/nginx
├── conf.d/
│   └── dev.conf          # FASTAPI 서버 리버스프록시 설정 파일
└── nginx.conf            # nginx 기본 설정 파일
```

설정 파일 작성

1. dev.conf

```
  upstream ai-dev {
    server fastapi-dev-container:8001;
  }

  # 1. HTTP(80포트) 요청은 모두 HTTPS로 리다이렉트
  server {
    listen 80;
    server_name ai-dev.film-moa.com;

    return 308 https://$host$request_uri;
  }
```

```
  # 2. HTTPS(443포트) 요청을 처리하는 서버 블록
  server {
    listen 443 ssl;
    server_name ai-dev.film-moa.com;

    client_max_body_size 100M; # 최대 업로드 크기 설정

    ssl_certificate /etc/letsencrypt/live/ai-dev.film-moa.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ai-dev.film-moa.com/privkey.pem;

    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
      proxy_pass http://ai-dev;
      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;
    }
  }
```

Nginx 컨테이너 구동

```
docker run -d \
  --name nginx \
  --network app-network \
  -p 80:80 \
  -p 443:443 \
  -v /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  nginx:latest
```

7. 사용한 외부 서비스 정보

Kakao 로그인

- **방식**: OAuth 2.0 Authorization Code Grant
- **Client ID**: `.env` 파일의 `NEXT_PUBLIC_KAKAO_CLIENT_ID`에 저장
- **Redirect URI**: `https://film-moa.com/oauth/kakao/callback`
- **주의사항**
  - 배포 도메인(`https://film-moa.com`)은 Kakao Developer Console에 반드시 등록되어야 함
  - 로컬 개발 시에는 `http://localhost:3000/oauth/kakao/callback` 등 별도 설정 필요

Amazon EC2

- **역할**: Jenkins, Nginx, Spring Boot, Next.js 컨테이너가 배포된 주 서버
- **접속 방식**: PEM 키를 통한 SSH (`ubuntu@<public-ip>`)
- **보안 그룹 설정**: 80, 443, 22 포트 허용

Amazon RDS

- **DBMS**: MySQL 8.41
- **계정 정보**: Jenkins Credentials로 주입

Amazon S3

- **용도**: 이미지, 동영상 등 미디어 업로드 저장소
- **접근 방식**: 백엔드 서버가 AWS SDK를 통해 Signed URL 발급
- **버킷 정책**: 비공개 (private), 모든 접근은 Signed URL을 통해 수행

Amazon CloudFront

- **역할**: S3 미디어 파일의 CDN 배포
- **도메인**
  - 개발용: `d3t4tucldvkzm6.cloudfront.net`
  - 배포용: `d2w650bgmlbl7n.cloudfront.net`
- **Signed URL 사용**

- **주의사항**
  - 프론트엔드에서는 직접 접근하지 않고, 백엔드에서 Signed URL을 받아 렌더링

GCP 인프라 (부가적인 AI 기능 용도)

- **역할**: AI 기능 서버
- **서버 위치**: Google Cloud Compute Engine
- **접속 방식**: SSH 접속 / Docker 기반으로 서버 운영
- **연동 방식**: API 통신을 통해 메인 서버(Spring)에서 해당 서버로 요청

---

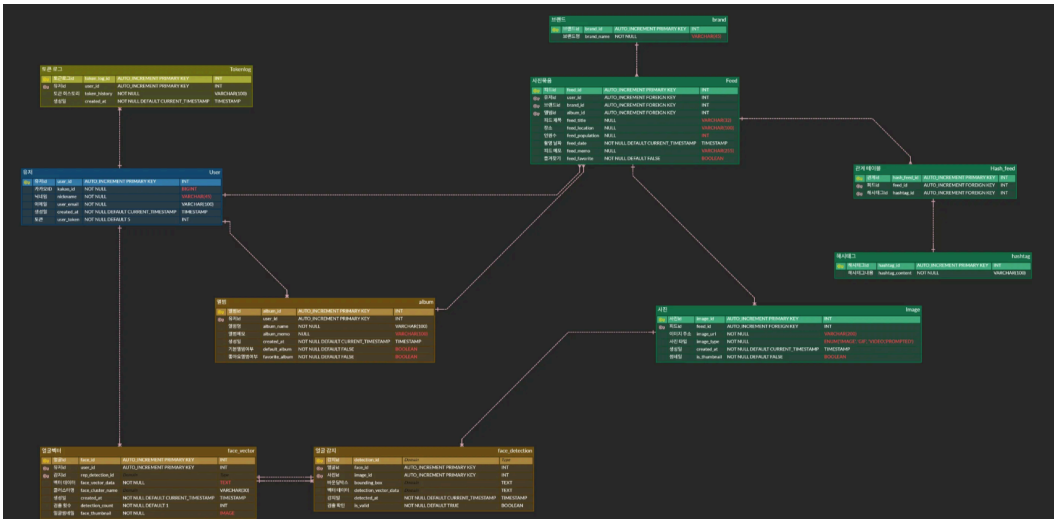
8. ERD & DB 접속 정보

- DB 설정 정보

항목	값
DBMS	MySQL
JDBC URL	`jdbc:mysql://mysql-dev-container:3306/filmmoa?serverTimezone=Asia/Seoul&characterEncoding=UTF-8`
사용자명	`ro...`
비밀번호	`r...`
Driver	`com.mysql.cj.jdbc.Driver`
Hibernate Dialect	`org.hibernate.dialect.MySQLDialect`
ddl-auto	`update`

- 개발용 서버 MYSQL 환경 변수

환경 변수명	설명
`MYSQL_ROOT_PASSWORD`	MySQL root 계정 비밀번호
`MYSQL_DATABASE`	생성할 기본 DB 이름 (filmmoa)
`MYSQL_USER`	일반 사용자 계정명
`MYSQL_PASSWORD`	일반 사용자 비밀번호



9. DB 덤프 파일

대전_B208_PIXX_Dump20250521.zip

10. 기타 특이사항

- jenkins credentials파일들은 Git에 포함되지 않으며, Jenkins 환경변수 파일로 Excute Shell을 통해 경로 상에 배치시킨 후 `docker compose up -d —build`
- Nginx 컨테이너 1개를 별도로 띄우고, 외부 포트(80, 443)를 열어주도록 설정하였으며, 개발용 및 배포용 Spring, Next.js 컨테이너를 내부포트만 열어 리버스프록시 설정함
- LoadBalancing 용 Nginx 설정 및 Let’s Encrypt로 https 설정함
- `conf.d/dev.conf`, `conf.d/prod.conf` 두 개로 개발용, 배포용 분리함