포팅 메뉴얼

목차

1. 개발환경

1.1 Frontend

Next.js 및 관련 라이브러리

- next 14.2.5
- react ^18
- react-dom ^18
- cookies-next ^4.2.1
- swr ^2.2.5

스타일링 및 UI 라이브러리

- tailwindcss ^3.4.1
- framer-motion ^11.3.8
- swiper ^11.1.5

유틸리티 및 데이터 라이브러리

- dayjs ^1.11.12
- html-react-parser ^5.1.12
- react-hook-form ^7.52.1

차트 및 시각화 라이브러리

- chart.js ^4.4.3
- react-chartjs-2 ^5.2.0
- chartjs-plugin-datalabels ^2.2.0

웹 소켓 및 스트리밍

- @stomp/stompjs ^7.0.0
- sockjs-client ^1.6.1
- openvidu-browser ^2.30.1

Google Cloud API

- @google-cloud/speech ^6.7.0
- @google-cloud/text-to-speech ^5.3.0

개발 도구 및 형식화

- typescript ^5
- eslint ^8
- eslint-config-next 14.2.5
- eslint-config-prettier ^9.1.0
- @types/node ^20
- @types/react ^18
- @types/react-dom ^18
- @types/sockjs-client ^1.5.4
- postcss ^8

SVG 처리 라이브러리

- @svgr/webpack ^8.1.0

디지털 잉크 처리 라이브러리

- iink-ts ^1.0.5

1.2 Backend

자바

- Java OpenJDK 17
- Spring Boot 3.3.1
- Spring Dependency Management 1.1.5
    - Spring Data JPA 3.3.1
    - Spring Security 3.3.1
    - Websocket 3.3.1
    - Validation 3.3.1
    - Lombok 1.18.34
- Gradle 8.8

JSON 및 XML 처리

- jsoup 1.15.3
- org.json 20231013

JWT

- io.jsonwebtoken 0.11.5

데이터베이스

- MySQL 8.0.33

WebRTC

- io.openvidu 2.30.0

Amazon S3

- com.amazonaws 1.12.765

SpringDoc

- org.springdoc 2.6.0

1.3 Server

- Ubuntu 20.04.6 LTS
- Nginx 1.27.0
- Docker 27.1.1
- Docker Compose 2.29.1
- Jenkins 2.471

1.4 Database

- MySQL 9.0.1

1.5 IDE

- Visual Studio Code 1.90.2
- IntelliJ IDEA 2024.1.4

1.6 형상 / 이슈관리

- Gitlab
- Jira

1.7 기타 툴

- Postman 11.8.0

2. 환경변수

2.1 Frontend

> **NEXT_PUBLIC_BASE_URL**
> **NEXT_PUBLIC_APPLICATION_KEY**
> **NEXT_PUBLIC_HMAC_KEY**
> **NEXT_GOOGLE_API_PATH**

2.2 Backend

가독성이 좋은 application.yml 을 작성하여 환경변수를 관리

```
spring:
  config:
    import: optional:file:.env[.properties]
  application:
    name: koala_back
  datasource:
    url: ${SPRING_DATASOURCE_URL}
    username: ${SPRING_DATASOURCE_USERNAME}
    password: ${SPRING_DATASOURCE_PASSWORD}
    driver-class-name: ${SPRING_DATASOURCE_DRIVER_CLASS_NAME}
    hikari:
      connection-timeout: 15000
      maximum-pool-size: 10
      max-lifetime: 240000
      leak-detection-threshold: 10000
  jpa:
    database-platform: org.hibernate.dialect.MySQLDialect
    hibernate:
      ddl-auto: update
  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB
      max-request-size: 10MB
  task:
    scheduling:
```

```
      pool:
        size: 10
      shutdown:
        await-termination: true
        await-termination-period: 60s

logging:
  level:
    org:
      springframework: DEBUG

jwt:
  secret: ${JWT_SECRET}
  access-token-ms: ${JWT_ACCESS_TOKEN_MS}
  refresh-token-ms: ${JWT_REFRESH_TOKEN_MS}

openvidu:
  url: ${OPENVIDU_URL}
  secret: ${OPENVIDU_SECRET}

cloud:
  aws:
    credentials:
      access-key: ${S3_ACCESS_KEY_ID}
      secret-key: ${S3_SECRET_ACCESS_KEY}
    region:
      static: us-east-1
    stack:
      auto: false

openai:
  api:
    key: ${OPENAI_API_KEY}

springdoc:
  swagger-ui:
    path: /swagger-ui.html

gemini:
  baseurl: ${GEMINI_BASE_URL}

googleai:
  api:
    key: ${GEMINI_API_KEY}
```

2.3 민감 환경변수 관리

2.3.1 Frontend

> **Jenkins Pipeline의 workspace에 위치한 프로젝트 Git Repository에서 .env , secret.json(google api key) 수동 저장 및 관리 (.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)**

```
# 경로
/home/ubuntu/jenkins-data/workspace/koala/frontend

### koala는 Jenkins Pipeline 이름
```

2.3.2 Backend

> **Jenkins Pipeline의 workspace에 위치한 프로젝트 Git Repository에서 .env 수동 저장 및 관리 (.gitignore에 추가하여 GitLab에 푸시되는 일이 없도록 함)**

```
# 경로
/home/ubuntu/jenkins-data/workspace/koala/backend/koala_back

### koala는 Jenkins Pipeline 이름
```

3. EC2 세팅

3.1 Docker 설치

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyr
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Docker 패키지 설치(최신 버전)
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plug

# Docker Engine 설치 성공 확인
sudo docker run hello-world
```

3.2 MySQL(Docker) 설치

```
# MySQL Docker 이미지 다운로드
## 버전 명시하지 않으면 최신버전으로 다운로드
$sudo docker pull mysql

# MySQL Docker 컨테이너 생성 및 실행
$sudo docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> -d
```

3.3 Nginx 설치 및 Reverse Proxy 세팅

3.3.1 nginx 이미지 다운로드

```
$docker pull nginx:lates
```

3.3.2 nginx.conf 파일 작성

```
* ec2 인스턴스 /home/ubuntu/nginx.conf 경로에 존재

events { }

http {
        # DDoS 방어 설정
    limit_req_zone $binary_remote_addr zone=ddos_req:50m rate=20r/s;

    upstream frontend {
        server frontend:5000;
    }

    upstream backend {
        server backend:8080;
    }


    server {
        listen 80;
        server_name ko-ala.site;


        location /.well-known/acme-challenge/ {
              allow all;
            root /var/www/certbot;
        }
        # http 요청 https로 리다이렉션
        location / {
            return 301 https://$host$request_uri;
        }
    }
```

```
    server {
        listen 443 ssl;
        server_name ko-ala.site;

                # ssl 인증서 관련 부분
                # ssl 인증서 받기 전에는 이부분을 주석 처리해야 오류가 안남.
        ssl_certificate /etc/letsencrypt/live/ko-ala.site/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/ko-ala.site/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        location / {
            proxy_pass http://frontend/;
            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;
            limit_req zone=ddos_req burst=10;

        }

        location /api/lecture-chat/connections {
            proxy_pass http://backend/api/lecture-chat/connections;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "Upgrade";
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }


        location /api/ {
            proxy_pass http://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;

        }




    }
}
```

3.3.3 docker-compose-prod.yml 파일 작성

- docker-compose-prod.yml은 jenkins와 연동된 gitlab 레포지토리 안에 존재한다.

```yaml
version: '3'
services:
  backend:
    container_name: back-server
    build:
      context: ./backend/koala_back
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    env_file:
      - ./backend/koala_back/.env
    environment:
      - TZ=Asia/Seoul
    networks:
      - koala

  frontend:
    container_name: front-client
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    env_file:
      - ./frontend/.env
    environment:
      - TZ=Asia/Seoul
    volumes:
      - /home/ubuntu/jenkins-data/workspace/koala/frontend/secret.json:/app/s
    networks:
      - koala

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf
      - /home/ubuntu/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/certbot/www:/var/www/certbot
    networks:
      - koala

    # certbot 이미지는 도커 컴포즈를 실행하면 자동으로 다운해준다.
  certbot:
    image: certbot/certbot
```

```
    volumes:
      - /home/ubuntu/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep

networks:
  koala:
```

3.3.4 SSL 인증서 받기

```
# ec2 인스턴스

# ssl 인증서를 저장할 경로를 만들어 준다.
$ mkdir -p certbot/conf
$ mkdir -p certbot/www

$ curl -L https://raw.githubusercontent.com/wmnnd/nginx-certbot/master/init-l
$ chmod +x init-letsencrypt.sh
$ vi init-letsencrypt.sh // 도메인, 이메일, 디렉토리 수정

domains=(ko-ala.site)
rsa_key_size=4096
data_path="./certbot"
email="" # Adding a valid address is strongly recommended
staging=0 # Set to 1 if you're testing your setup to avoid hitting request li
-> 위 부분 처럼 도메인, 경로 지정 후 저장

$ sudo ./init-letsencrypt.sh // script를 실행하여 인증서 발급
```

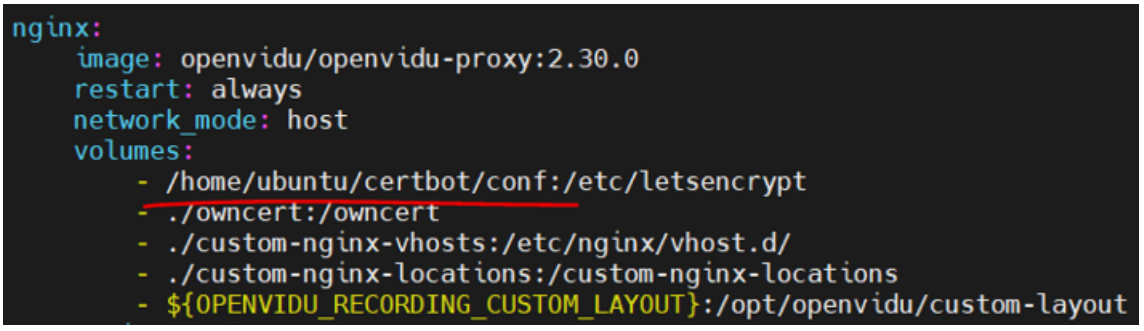
3.4 OPENVIDU 온 프레미스 배포

```
# ec2 인스턴스

# /opt 디렉토리로 이동
$ cd /opt

# 스크립트 실행
$ curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_la

# openvidu 디렉토리 이동
$ cd openvidu

# .env 파일 편집
$ vi .env

# DOMAIN_OR_PUBLIC_IP, OPENVIDU_SECRET 설정
# CERTIFICATE_TYPE=letsencrypt
```

```
# openvidu http, https 포트를 수정해 주어야 백, 프론트를 띄운 nginx 포트와 충돌이 일어나
# HTTP_PORT=8082
# HTTPS_PORT=8443

# openvidu 실행
$ ./openvidu start
```

3.4.1 OPENVIDU SSL 인증서 경로 수정

```
$ cd opt/openvidu
$ sudo vi docker-compose.yml
```



→ 기존에 발급 받은 ssl 인증서가 저장되어 있는 root로 변경해 준다.

3.5 EC2 Port

Port 번호	내용
22	SSH
80	HTTP (HTTPS로 redirect)
443	HTTPS
3306	MySQL
3478	Openvidu STUN/TURN
5000	Frontend
8080	Backend
8081	Jenkins
8443	Openvidu
8888	Openvidu Kurento Media
40000-57000	Openvidu Kurento Media
57001-65535	Openvidu TURN

4. CI/CD 구축

4.1 Jenkins Dockerfile, Docker in Docker 방식

```
# ec2 인스턴스 /home/ubuntu
```

```
# jenkins dockerfile을 저장할 디렉토리 생성
$ mkdir -p jenkins-dockerfile

# /home/ubuntu/jenkins-dockerfile 루트에 Dockerfile 생성

FROM jenkins/jenkins:2.471-jdk17

USER root

COPY docker_install.sh /docker_install.sh
RUN chmod +x /docker_install.sh
RUN /docker_install.sh

RUN groupadd -f docker
RUN usermod -aG docker jenkins

USER jenkins

# 동일 루트에 docker_install.sh 생성
#!/bin/sh
apt-get update && \
apt-get -y install apt-transport-https \
  ca-certificates \
  curl \
  gnupg2 \
  zip \
  unzip \
  software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")
add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo
$(lsb_release -cs) \
stable" && \
apt-get update && \
apt-get -y install docker-ce
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-co
chmod +x /usr/local/bin/docker-compose
```

4.2 Jenkins docker-compose.yml

```
# ec2 인스턴스 /home/ubuntu

# jenkins docker-compose 파일을 저장할 디렉토리 생성
$ mkdir -p jenkins

# Docker 볼륨 폴더 권한 설정
$ mkdir -p jenkins-data
$ sudo chown 1000 /home/ubuntu/jenkins-data/
```

```
# /home/ubuntu/jenkins 루트에 docker-compose.yml 생성

version: "3.2"
services:
    jenkins:
       container_name: jenkinscicd
       build:
          context: /home/ubuntu/jenkins-dockerfile
          dockerfile: Dockerfile
       restart: unless-stopped
       ports:
         - 8081:8081
         - 50000:50000
       environment:
         - JENKINS_OPTS=--httpPort=8081
         - TZ=Asia/Seoul
       volumes:
         - /home/ubuntu/jenkins-data:/var/jenkins_home
         - /home/ubuntu/jenkins/.ssh:/root/.ssh
         - /var/run/docker.sock:/var/run/docker.sock

 # /home/ubuntu/jenkins
 # 젠킨스 컨테이터 실행
 $ sudo docker-compose up -d --build
```

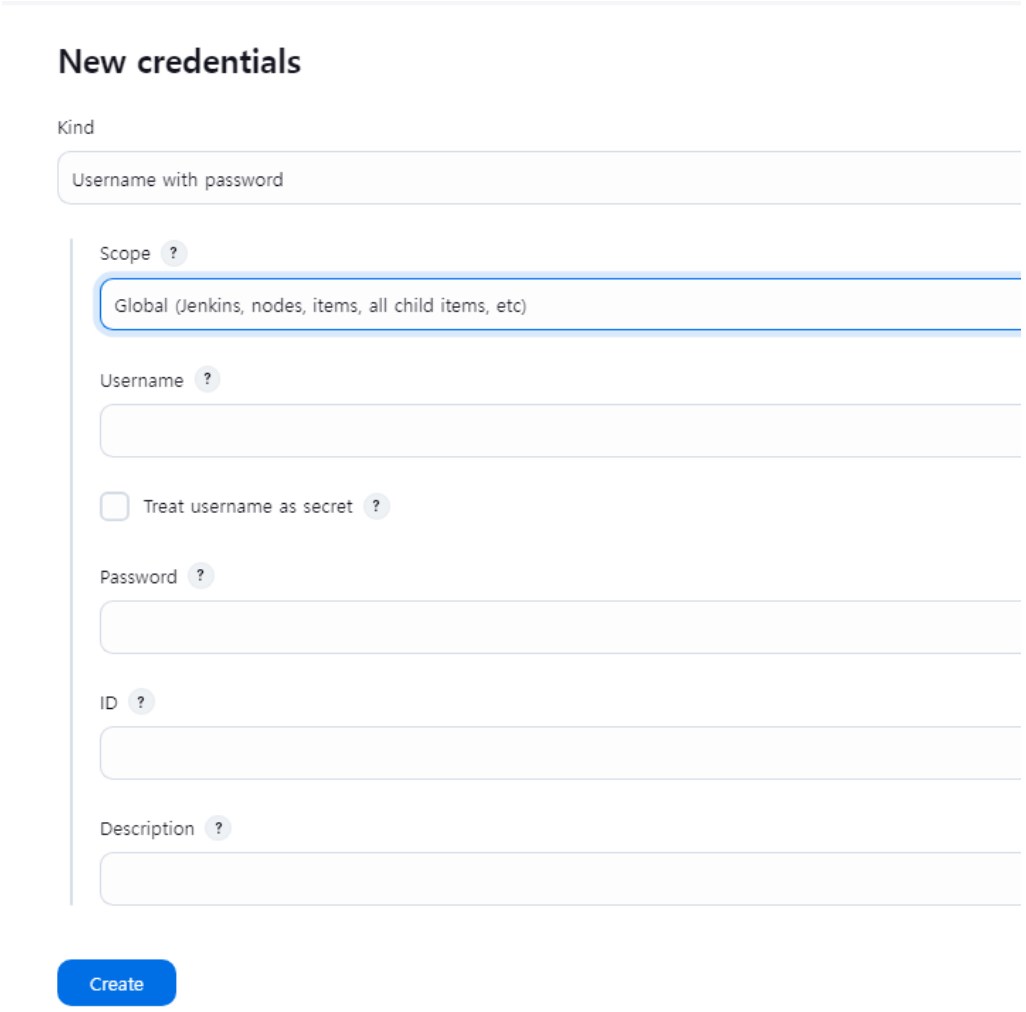
4.3 Jenkins docker 권한 설정

```
# jenkins 접속 전 /var/run/docker.sock 에 대한 권한을 설정해줘야 함.
# 초기 권한이 소유자와 그룹 모두 root였기 때문에 이제 그룹을 root에서 docker로 변경

# jenkins 컨테이너 접속
$ sudo docker exec -it -u root jenkinscicd /bin/bash

# 그룹 변경
$ sudo chown root:docker /var/run/docker.sock
$ exit
```

4.4 Jenkins 설정

4.4.1 GitLab Credentials 설정

1. 아이디 → “Credentials” 클릭
2. “Store : System” → “(global)” → “+ Add Credentials” 클릭
3. Username: GitLab ID
4. Password: GitLab Personal Access Tokens

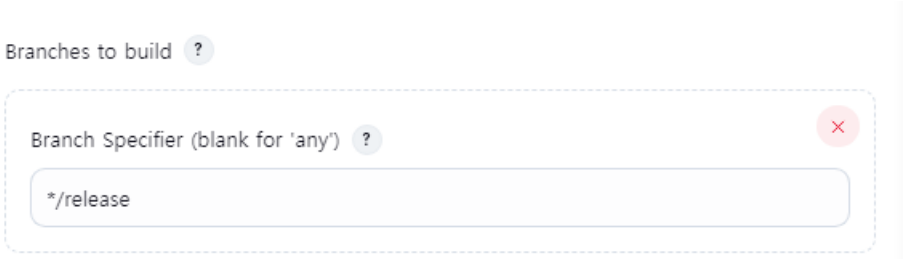


4.4.2 Jenkins Item 생성

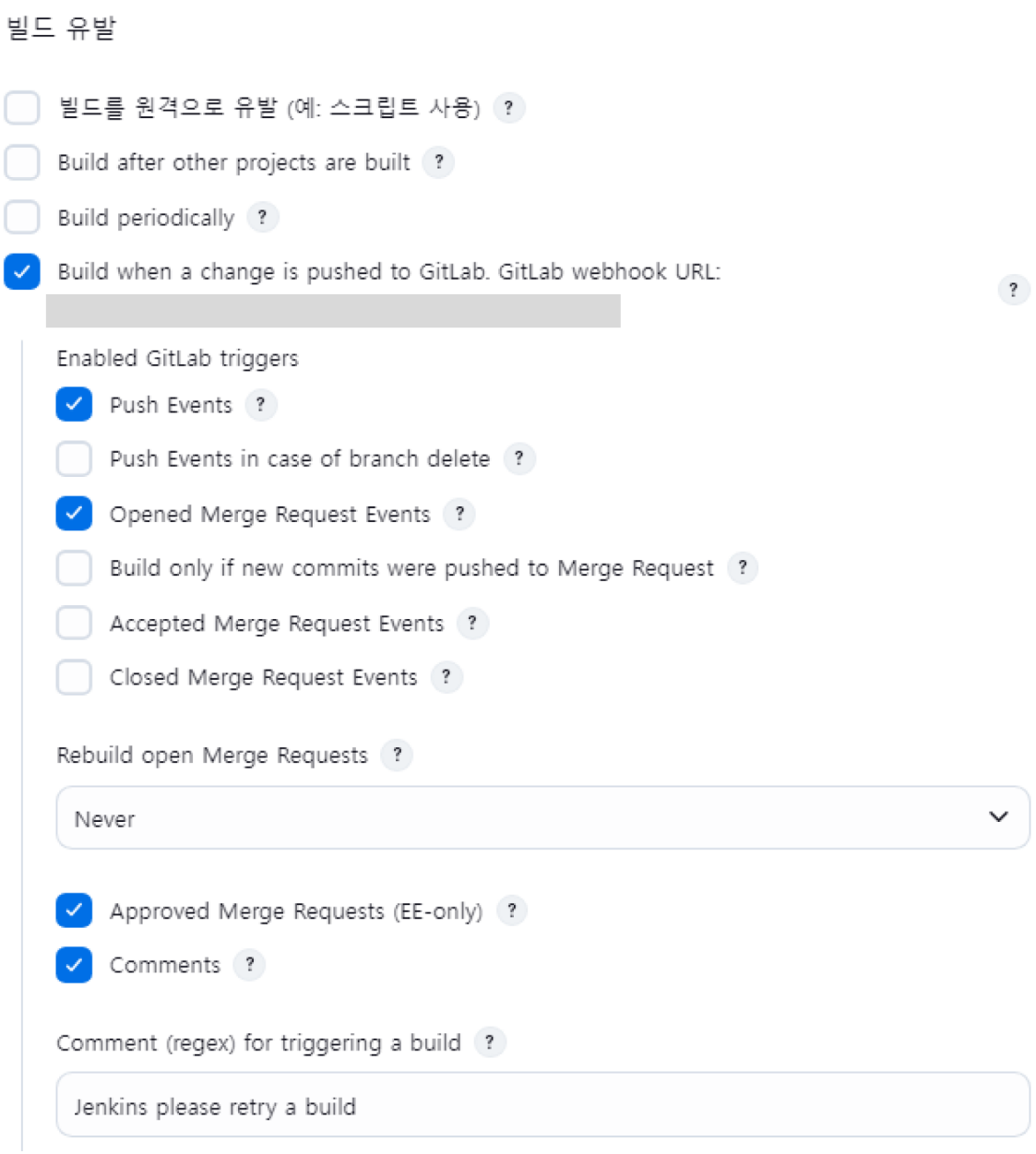
1. “새로운 Item” 클릭
2. “Enter an item name”에 임의 Item 이름 입력 → “Freestyle project” 클릭

3. “구성” 클릭
4. “소스 코드 관리”에 연동할 Repository URL 입력
5. 등록해둔 Credentials 설정

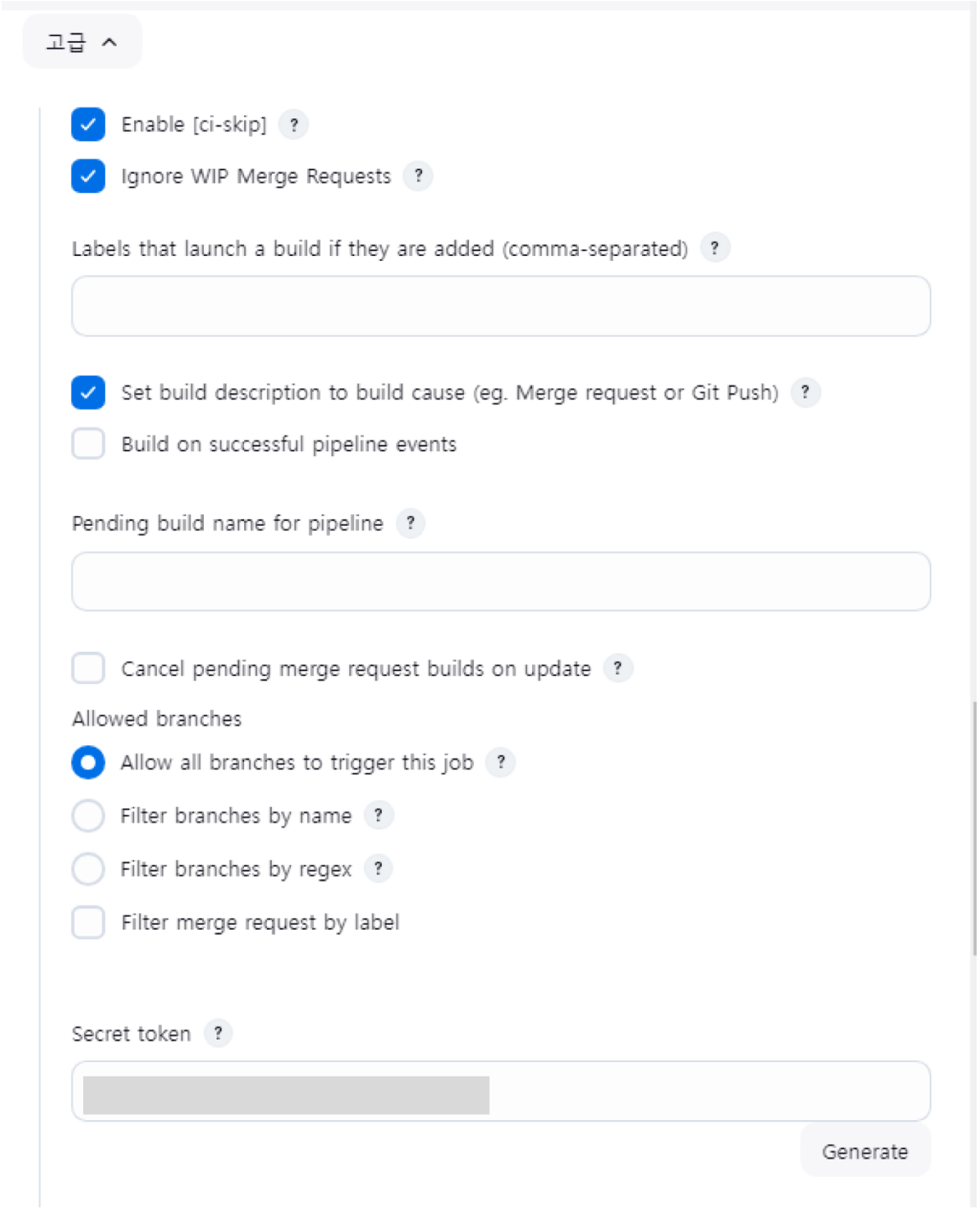
6. 연동할 브랜치 입력



7. “빌드 유발”에서 트리거 설정
8. GitLab webhook URL 기록해두기

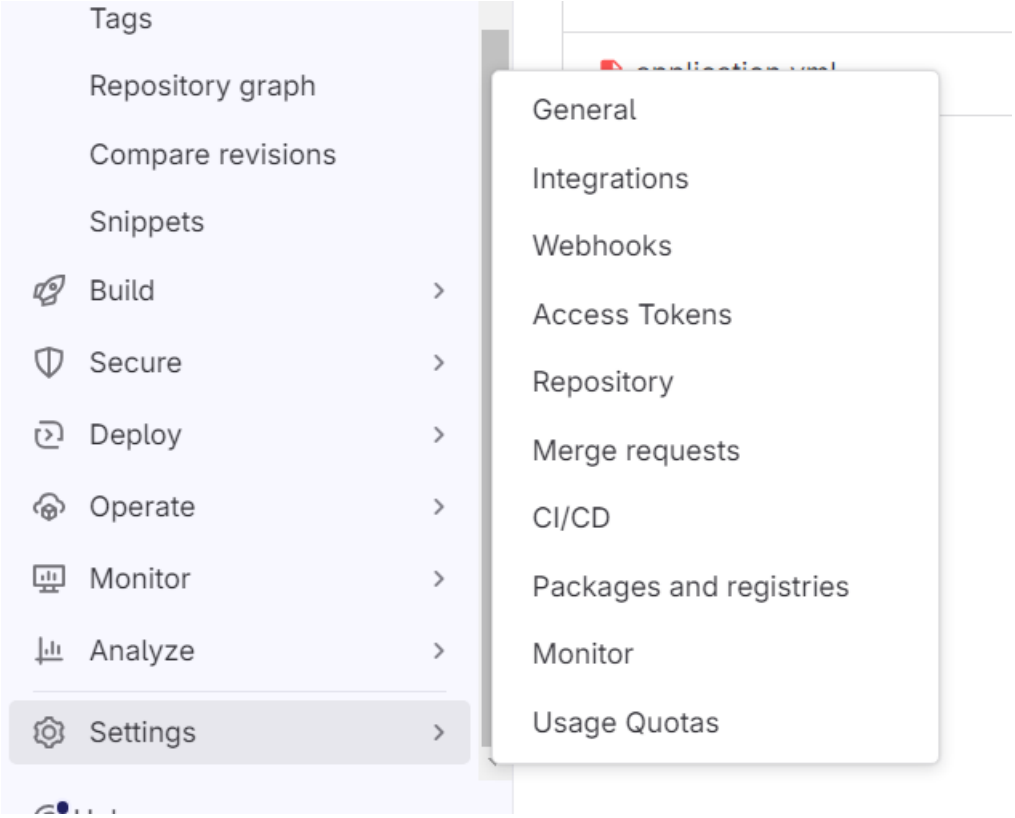


9. “빌드유발” 안에 “고급” 클릭
10. Secret Token Generate & 기록해두기

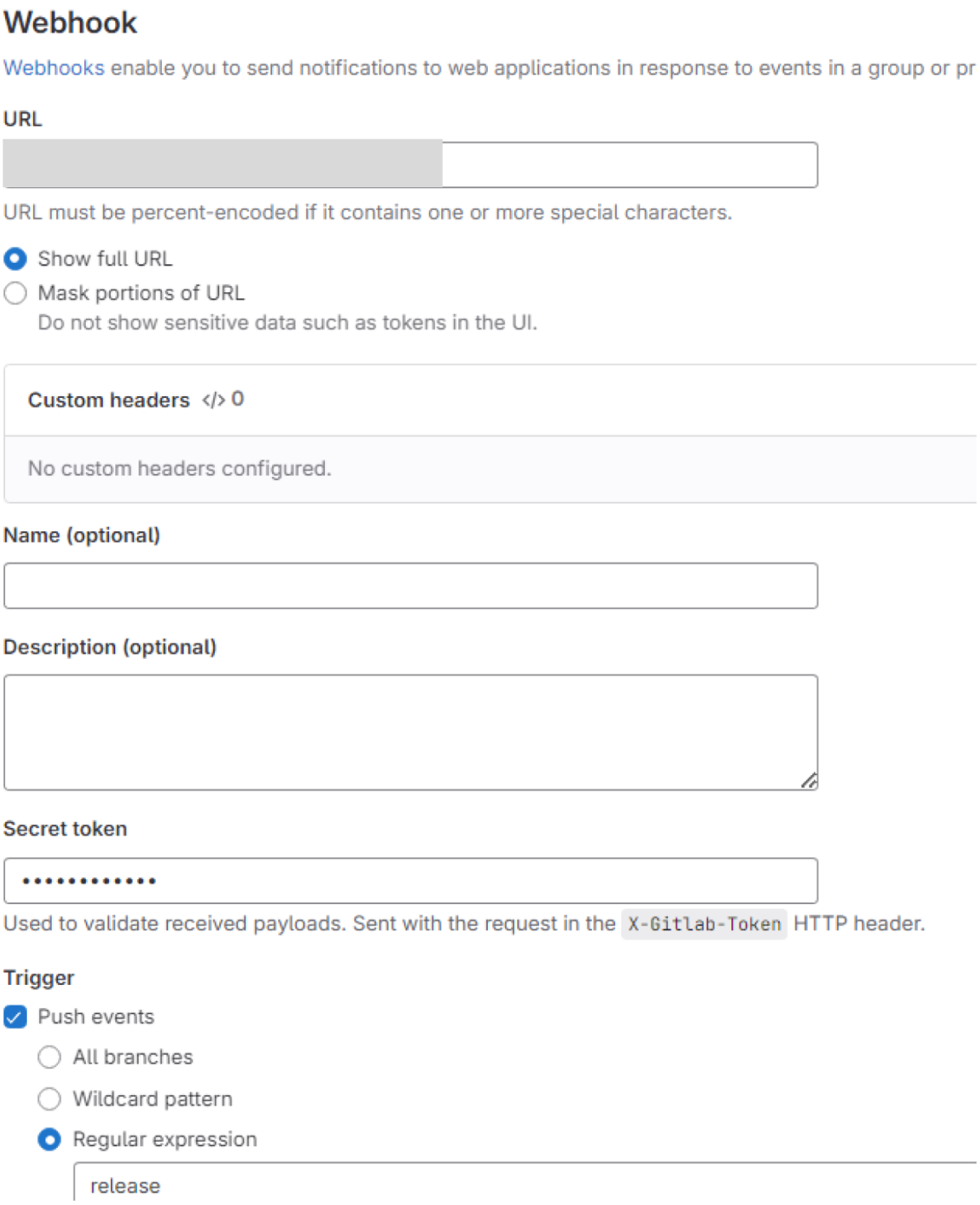


4.4.3 GitLab Webhook 설정

1. 프로젝트 GitLab → “Settings” → “Webhooks” 클릭



2. “URL”에 사전에 복사해놓은 Jenkins URL 입력
3. “Secret token”에 사전에 복사해놓은 Secret token 입력
4. “Trigger” Push events 클릭 후 “Regular expression” 에 연결 브랜치 입력



4.4.4 빌드 후 배포 명령어 설정

1. 만들어 둔 아이템 “구성” 클릭
2. “Build Steps”에서 Execute shell 작성 후 저장

4.4.5 빌드 및 배포

> **상기 WebHook 설정한 브랜치로 푸시 및 MergeRequest**

4.5 Jenkins 연동 브랜치 배포 파일 구성

4.5.1 루트 경로 파일구조

```
S11P12A502/
│
├── .idea/                      # IDE 관련 설정 디렉토리 (IntelliJ 등)
│
├── backend/                    # 백엔드 소스 코드가 위치한 디렉토리
│
├── frontend/                   # 프론트엔드 소스 코드가 위치한 디렉토리
│
├── docker-compose-prod.yml     # 프로덕션 환경용 Docker Compose 파일
│
├── README.md                   # 프로젝트에 대한 설명과 정보를 담은 파일
│
└── start-prod.sh               # 프로덕션 환경에서 실행할 스크립트 파일
```

4.5.2 start-prod.sh 빌드 후 실행되는 파일

```
docker-compose -f docker-compose-prod.yml down

docker rmi -f $(docker images koala_backend:latest -q) || true
docker rmi -f $(docker images koala_frontend:latest -q) || true
```

```
docker-compose -f docker-compose-prod.yml pull

COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose

docker rmi -f $(docker images -f "dangling=true" -q) || true
```

4.5.3 docker-compose-prod.yml 파일

```
version: '3'
services:
  backend:
    container_name: back-server
    build:
      context: ./backend/koala_back
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    env_file:
      - ./backend/koala_back/.env
    environment:
      - TZ=Asia/Seoul
    networks:
      - koala

  frontend:
    container_name: front-client
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    env_file:
      - ./frontend/.env
    environment:
      - TZ=Asia/Seoul
    volumes:
      - /home/ubuntu/jenkins-data/workspace/koala/frontend/secret.json:/app/s
    networks:
      - koala

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/nginx.conf:/etc/nginx/nginx.conf
```

```
      - /home/ubuntu/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/certbot/www:/var/www/certbot
    networks:
      - koala

  certbot:
    image: certbot/certbot
    volumes:
      - /home/ubuntu/certbot/conf:/etc/letsencrypt
      - /home/ubuntu/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep

networks:
  koala:
```

5. Front, Back 배포 Dockerfile

5.1 Frontend

5.1.1 frontend 루트 경로 파일구조

```
frontend/
│
├── .next/                      # Next.js에서 빌드된 파일들이 저장되는 디렉토리
│
├── node_modules/               # 프로젝트의 NPM 패키지들이 저장되는 디렉토리
│
├── public/                     # 정적 파일(이미지, 폰트 등)이 위치한 디렉토리
│
├── src/                        # 소스 코드가 위치한 디렉토리
│
├── .dockerignore               # Docker 빌드 시 제외할 파일 및 디렉토리 목록
├── .eslintrc.json              # ESLint 설정 파일
├── .gitignore                  # Git에서 추적하지 않을 파일 및 디렉토리 목록
├── .prettierrc                 # Prettier 코드 스타일 설정 파일
├── Dockerfile                  # Docker 이미지를 빌드하기 위한 설정 파일
├── next.config.mjs             # Next.js 설정 파일
├── next-env.d.ts               # Next.js에서 환경 변수를 정의하는 TypeScript 파일
├── package.json                # 프로젝트 메타정보 및 의존성을 정의하는 파일
├── package-lock.json           # NPM 패키지의 의존성 트리를 잠그는 파일
├── postcss.config.mjs          # PostCSS 설정 파일
├── README.md                   # 프로젝트에 대한 설명과 정보를 담은 파일
├── tailwind.config.ts          # Tailwind CSS 설정 파일
└── tsconfig.json               # TypeScript 컴파일러 설정 파일
```

5.1.2 frontend Dockerfile

1. 경량화하여 이미지를 만들도록 설정함.
2. “next.config” 파일 수정

```
const nextConfig={
....
output: 'standalone',
....
}
```

```
# Dockerfile

FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies based on the preferred package manager
COPY package*.json ./
RUN npm install --force
RUN npm install sharp
RUN rm -rf ./.next/cache

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npm run build

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV=production
ENV PORT 5000

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public/
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs
```

```
EXPOSE 5000

CMD ["node", "server.js"]
```

5.1.3 .dockerignore 파일

```
node_modules
```

5.2 Backend

5.2.1 backend 루트 경로 파일구조

```
backend/
└── koala_back/
    │
    ├── .gradle/                # Gradle 빌드 도구와 관련된 캐시 및 파일이 저장되는 디
    │
    ├── .idea/                  # IDE(IntelliJ 등) 설정 디렉토리
    │
    ├── build/                  # 빌드된 파일들이 저장되는 디렉토리
    │
    ├── gradle/                 # Gradle Wrapper 관련 파일들이 위치한 디렉토리
    │
    ├── sql/                    # SQL 스크립트 파일들이 저장된 디렉토리
    │
    ├── src/                    # 애플리케이션의 소스 코드가 위치한 디렉토리
    │
    ├── .env                    # 환경 변수를 정의한 파일
    │
    ├── .gitignore              # Git에서 추적하지 않을 파일 및 디렉토리 목록
    │
    ├── build.gradle            # Gradle 빌드 스크립트 파일
    │
    ├── Dockerfile              # Docker 이미지를 빌드하기 위한 설정 파일
    │
    ├── gradlew                 # Gradle Wrapper 실행 파일 (Unix/Linux/Mac용)
    │
    ├── gradlew.bat             # Gradle Wrapper 실행 파일 (Windows용)
    │
    ├── passportKey.txt         # Passport 키를 저장하는 파일
    │
    ├── settings.gradle         # Gradle 설정 파일
    │
    └── README.md               # 프로젝트에 대한 설명과 정보를 담은 파일
```

5.2.2 backend Dockerfile

```
FROM bellsoft/liberica-openjdk-alpine:17 AS builder

COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

FROM bellsoft/liberica-openjdk-alpine:17
COPY --from=builder build/libs/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app.jar"]
```