자율 프로젝트 포팅 매뉴얼

소유자	CJ Lee
태그	

[개발환경 세팅]

- Android

Android Studio	Dolphin (2021.3.1)
Android SDK	minSDK = 21, targetSDK = 34, compileSDK = 34
Java	JDK 1.8

- Backend

Spring boot	3.3.5
Swagger(Spring doc)	2.0.2
S3	1.12.547
jdk	17
mysql	9.1.0
ORM	JPA(Hibernate)
intellij	2024.2.3(ultimate)

- Server

EC2	Ubuntu 20.04LTS
Nginx	1.27.2(Ubuntu)
Jenkins	2.479.1
docker	27.3.1

1. Docker, Docker-compose 설치

Docker와 Docker Compose 설치

```
sudo apt update && sudo apt upgrade

curl -fsSL https://get.docker.com -o install-docker.sh
chmod 711 install-docker.sh
./install-docker.sh

docker –v && docker compose version
```

2. Nginx 설치 및 세팅

2-1. Nginx, letsencrypt, certbot docker-compose.yml 로 구성

```
services:
  nginx:
    container_name: workalone-nginx
    image: nginx
    restart: unless-stopped
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; done & ngi
    volumes:
      - ./nginx:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - "80:80"
      - "443:443"
    networks:
      - workalone-network
  certbot:
    container_name: workalone-certbot
    image: certbot/certbot
    restart: unless-stopped
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait
    networks:
      - workalone-network

networks:
  workalone-network:
    external: true
```

2-2. Nginx Conf 수정

경로 : etc/nginx/conf.d/default.conf

```
server {
        listen 80;
        server_name k11s201.p.ssafy.io;

        location /.well-known/acme-challenge/ {
                root /var/www/certbot;
        }

        location / {
                return 301 https://$host$request_uri;
        }
```

```
}

server {
        listen 443 ssl;
        server_name k11s201.p.ssafy.io;

        ssl_certificate /etc/letsencrypt/live/k11s201.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/k11s201.p.ssafy.io/privkey.pem;

        deny 209.236.125.59;

        location / {
                proxy_pass http://workaloneApp:8080;

                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;

                root /usr/share/nginx/html;
                index index.html index.htm;
        }
}
```

2-3. Nginx 혹은 Certbot을 따로 실행할 수 있음

```
//nginx 시작
sudo docker compose up -d nginx
//certbot 시작
sudo docker compose up -d certbot
```

3. Jenkins 설치 및 실행

3-1. Jenkins 폴더 만들기

```
|Dockerfile
|docker-compose.jenkins.yml
|jenkins-data
```

3-2. Jenkins 설치를 위한 DockerFile 생성

```
FROM jenkins/jenkins:lts
USER root

RUN apt-get update && \
    apt-get upgrade && \
    curl -fsSL "https://get.docker.com" -o dockerSetter.sh && \
    chmod 711 dockerSetter.sh && \
    ./dockerSetter.sh
```

3-3. docker-compose.jenkins.yml 생성

```
services:
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: jenkins
```

```
    image: jenkins/jenkins:lts
    restart: always
    user: root
    ports:
      - "9090:8080"
      - "50000:50000"
    volumes:
      - /home/ubuntu/jenkins-data:/var/jenkins_home
      - /var/run:/var/run:ro
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      TZ: "Asiz/Seoul"
```

3-4. 젠킨스 컨테이너 실행

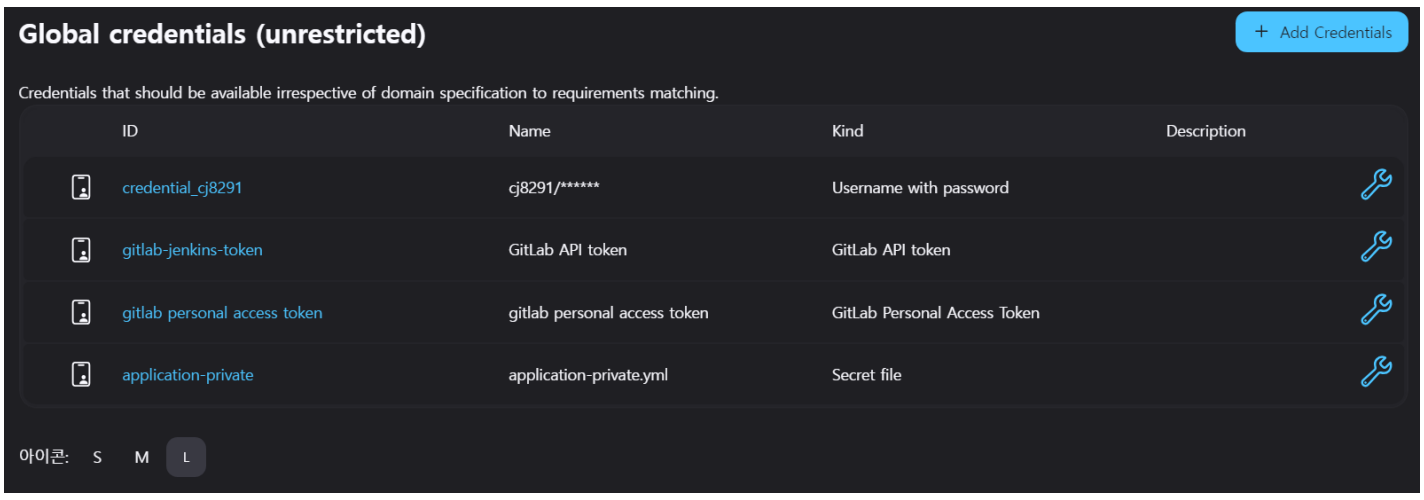
```
// docker compose 실행
docker compose -f docker-compose.jenkins.yml up -d --build
```

3-5. 젠킨스 접속 및 플러그인 설치

`http://k11s201.p.ssafy.io:9090`

→ initialAdminPassword는 docker logs ${젠킨스 컨테이너 이름} 을 통해 나오는 password를 입력

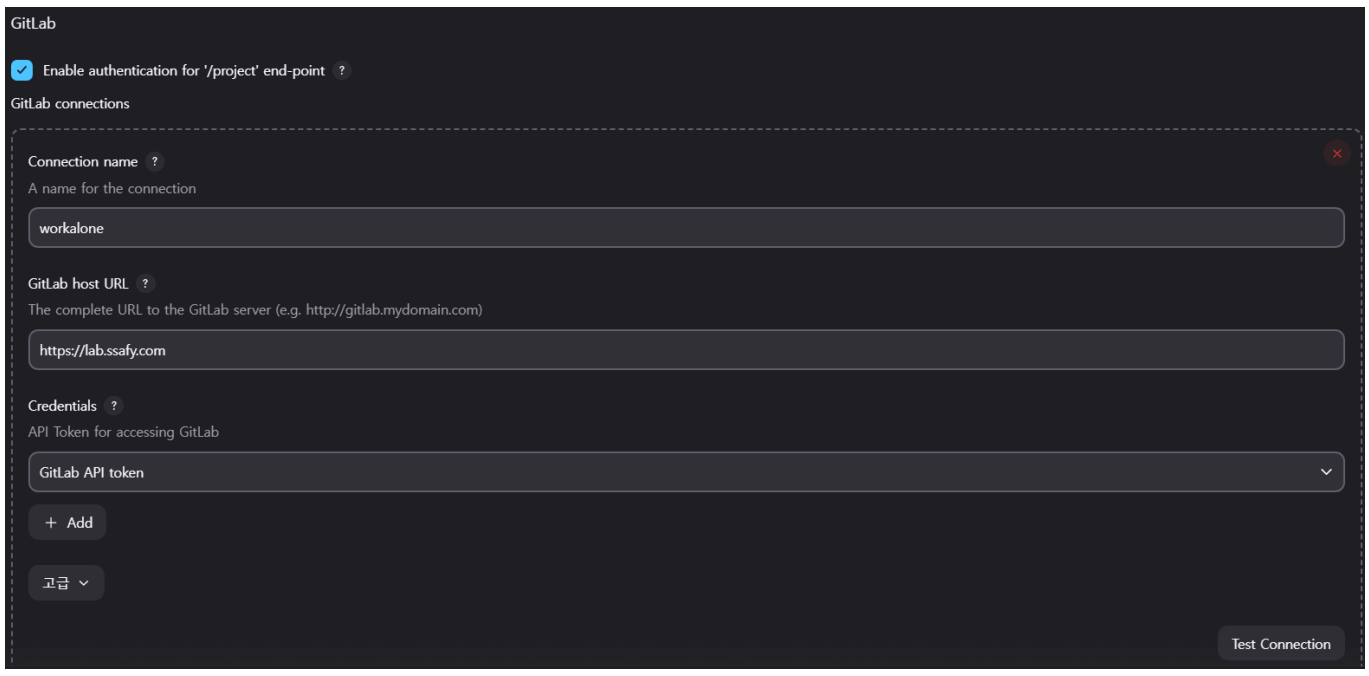
3-6. Credentials 추가



▼ credential_cj8291 ⇒ UserName, password credentials

```
stage('Checkout Application Git Branch') {
    steps {
        git credentialsId: 'credential_cj8291',
        url:
        'https://lab.ssafy.com/s11-final/S11P31S201.git',
        branch: 'develop'
    }
    post {
        failure {
            echo 'Repository clone failure !'
        }
        success {
            echo 'Repository clone success !'
        }
    }
}
```

▼ gitlab-jenkins-token ⇒ GitLab API token (프로젝트 관련 토큰)



▼ **gitlab personal access token (ID 관련 토큰)**

▼ **application-private.yml**

- **backend 민감한 정보 저장용** application-private.yml 파일

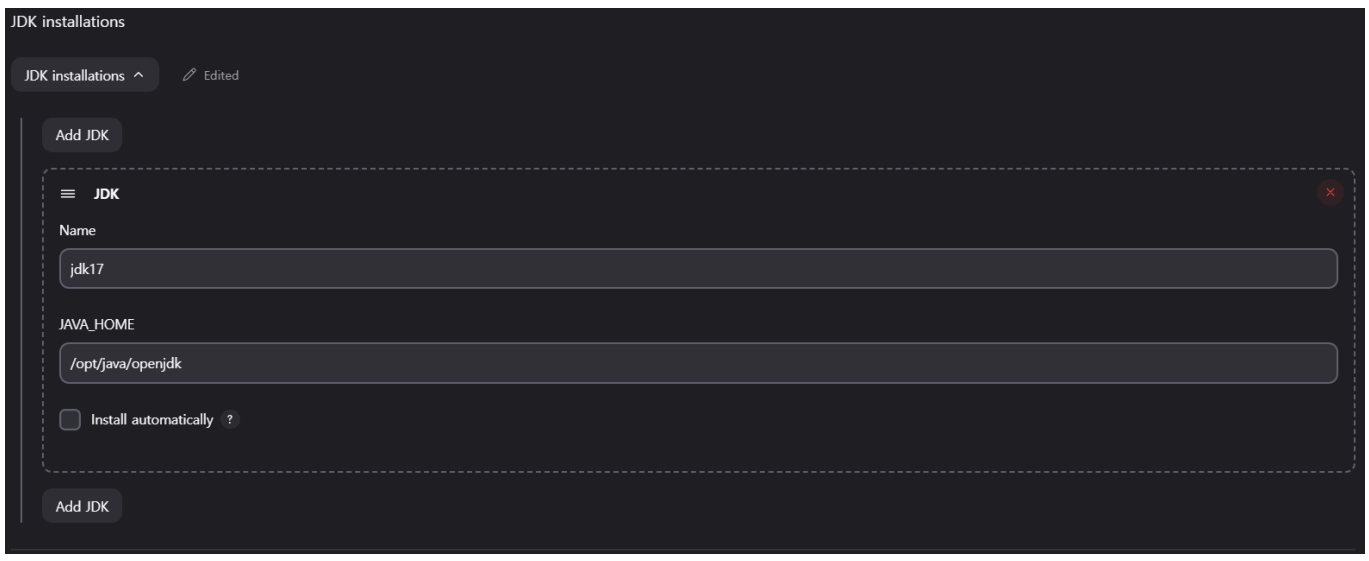
```
stage('spring application prviate setting'){
    steps{
        dir(path:'WorkAlone_BackEnd/src/main/resources'){
            sh 'cp $APPLICATION_PRIVATE application-private.yml'
        }
    }
}
```

3-7. Tools 설정

- jenkins container에서 사용할 jdk17, gradle 설치

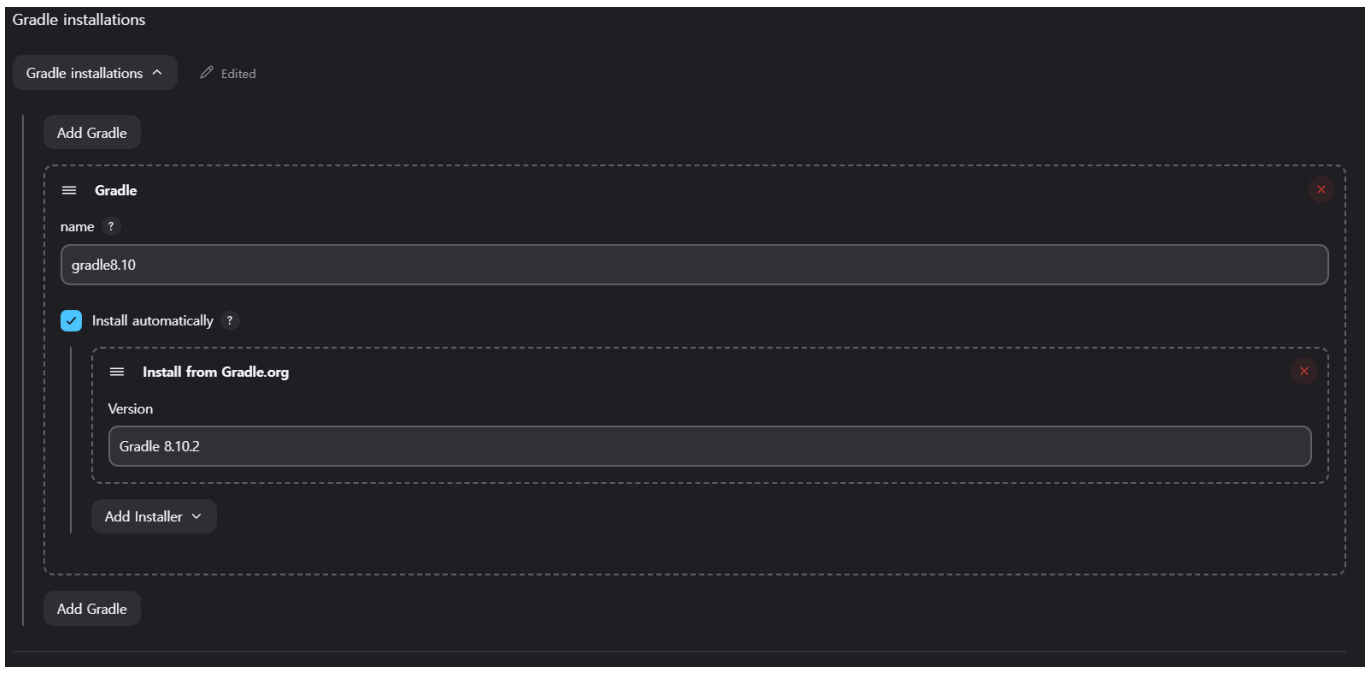
▼ **jdk17**

Jenkins에서 기본적으로 제공하는 jdk를 활용



▼ **gradle8.10**

Spring Boot 에서 활용하는 gradle version 과 일치



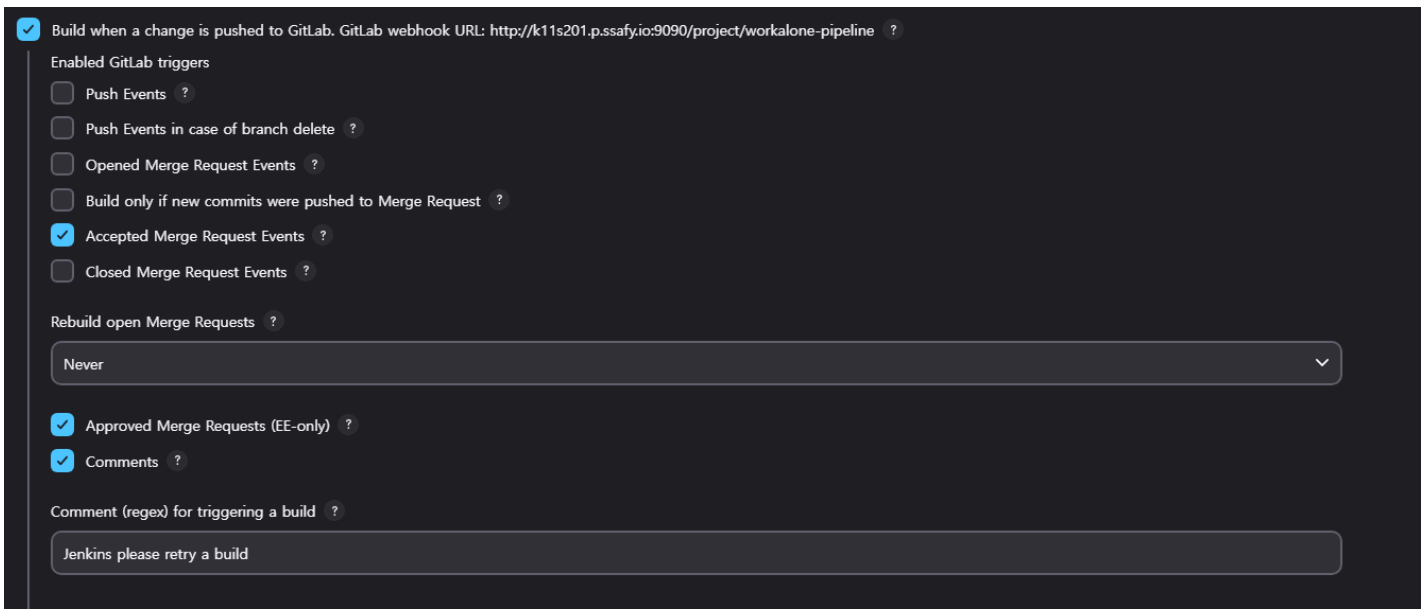
3-8. Pipeline 설정

```
pipeline {
    agent any

    environment {
        APPLICATION_PRIVATE = credentials('application-private')
    }

    tools {
        jdk ("jdk17")
        gradle ("gradle8.10")
    }

    stages {
        stage('Checkout Application Git Branch') {
            steps {
                git credentialsId: 'credential_cj8291',
                url:
                'https://lab.ssafy.com/s11-final/S11P31S201.git',
                branch: 'develop'
            }
            post {
                failure {
                    echo 'Repository clone failure !'
                }
                success {
                    echo 'Repository clone success !'
                }
            }
        }
        stage('spring application prviate setting'){
            steps{
                dir(path:'WorkAlone_BackEnd/src/main/resources'){
                    sh 'cp $APPLICATION_PRIVATE application-private.yml'
                }
            }
        }
        stage('gardle Jar Build') {
            steps {
                dir(path: 'WorkAlone_BackEnd/') {
```

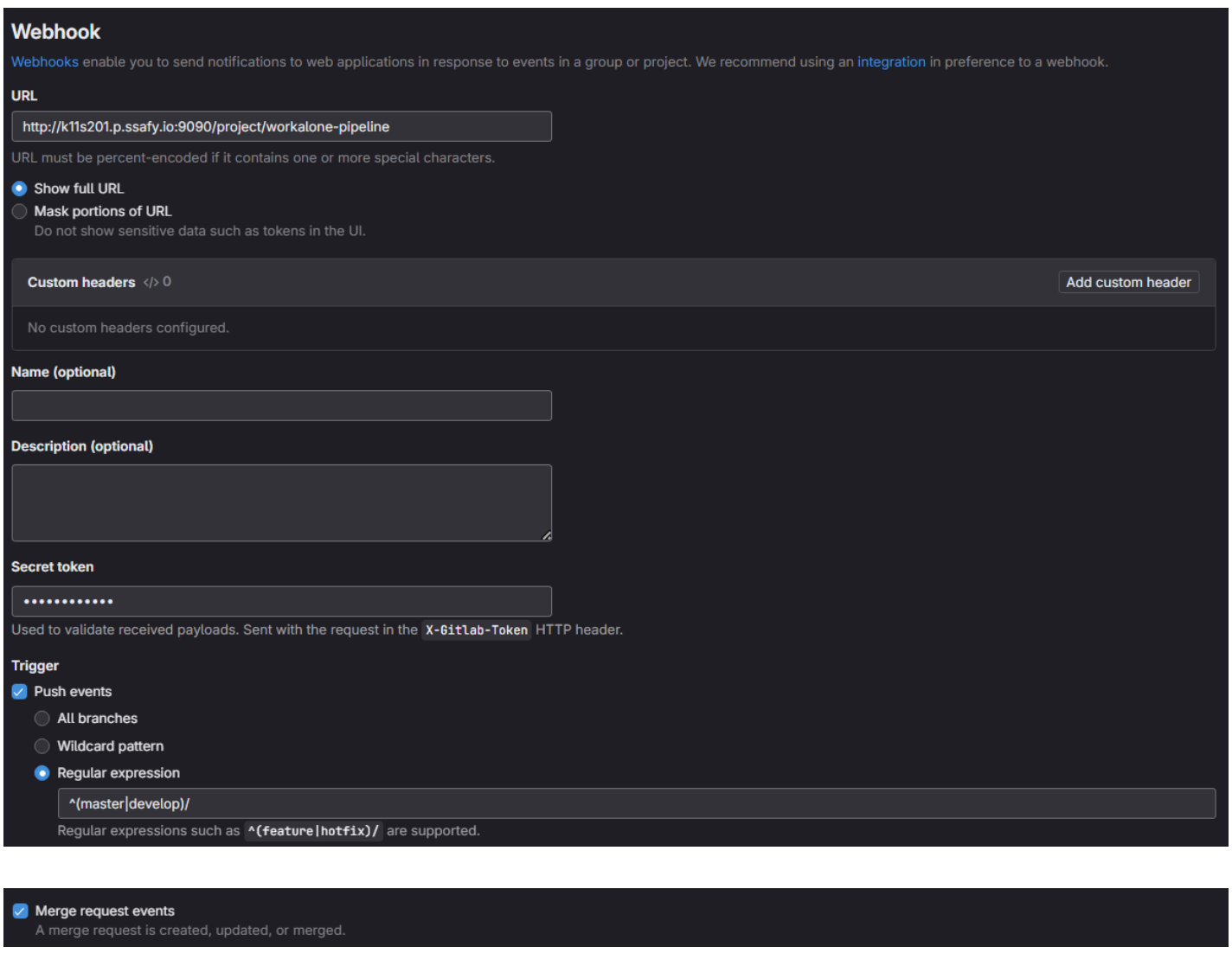
```
                    sh 'chmod +x gradlew'
                    sh './gradlew clean bootjar -Dspring.profiles.active=prod -x test'
                }
            }
            post {
                    failure {
                      echo 'Gradle jar build failure !'
                    }
                    success {
                      echo 'Gradle jar build success !'
                    }
            }
        }

        stage('Clean Up Old Images') {
            steps {
                script {
                    sh '''
                    docker stop $(docker ps -q --filter "ancestor=workalone:latest") || t
                    docker rm $(docker ps -a -q --filter "ancestor=workalone:latest") ||
                    docker rmi workalone:latest || true
                    '''
                }
            }
        }

        stage('build docker') {
            steps {
                dir(path: 'WorkAlone_BackEnd/'){
                    sh "docker build -t workalone:latest --build-arg SPRING_PROFILES_ACT
                }
                dir(path: 'WorkAlone_BackEnd/'){
                    sh "docker compose -f docker-compose.prod.yml up -d --force-recreate"
                }
            }
            post {
                    failure {
                      echo 'Docker build or run failure !'
                    }
                    success {
                      echo 'Docker build and run success !'
                    }
            }
        }
    }
}
```

3-9. Jenkins Webhook 설정

- Merge를 진행할 때 backend label을 설정해야 Jenkins에서 CI / CD가 이루어 질 수 있도록 설정



3-10. Gitlab Webhook 설정



- gitlab의 master 혹은 develop에 merge request 될 댸 webhook을 이용하여 자동 CI / CD 설정

4. CI/CD를 위한 Docker File 및 Docker-compose.yml 설정

Backend Server(Spring) - DockerFile

```
# 1단계: Gradle 빌드
FROM gradle:8.10.2-jdk17 AS build
WORKDIR /app
COPY . .
ARG SPRING_PROFILES_ACTIVE
RUN ./gradlew clean build

# 2단계: 빌드된 애플리케이션 실행
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} workalone_backend.jar
ENV SPRING_PROFILES_ACTIVE=${SPRING_PROFILES_ACTIVE}
CMD ["java", "-Dspring.profiles.active=${SPRING_PROFILES_ACTIVE}", "-jar", "-Duser.timezo
```

docker-compose.yml (local환경에서 실행하는 compose 파일)

```
services:
  workalone:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: workaloneApp
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://workaloneDB:3306/workalone?serverTimezone=Asia/
      SPRING_DATASOURCE_USERNAME: swallaby
      SPRING_DATASOURCE_PASSWORD: swallaby1234!
      SPRING_PROFILES_ACTIVE: local # 어차피 local과 prod 의 compose 파일 구성이 다르니 상관 없
    depends_on:
      workalone_db:
        condition: service_healthy

  workalone_db:
    image: mysql
    container_name: workaloneDB
    ports:
      - "3307:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ssafyssafyfighting
      MYSQL_DATABASE: workalone
      MYSQL_USER: swallaby
      MYSQL_PASSWORD: swallaby1234!
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
      interval: 10s
      retries: 3
```

docker-compose.prod.yml (production 환경에서 실행하는 compose 파일)

```
services:
  workalone:
    image: workalone:latest
    container_name: workaloneApp
    expose:
      - "8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://workaloneDB:3306/workalone?serverTimezone=As
```

```
ia/Seoul&characterEncoding=UTF-8
      SPRING_DATASOURCE_USERNAME: swallaby
      SPRING_DATASOURCE_PASSWORD: swallaby1234!
      SPRING_PROFILES_ACTIVE: prod
    networks:
      - workalone-network

networks:
  workalone-network:
    name: workalone-network
    driver: bridge
```

5. 서버에서 사용하는 환경변수

- Jenkins에서 credentials로 등록하는 private한 변수값들

application-private.yml(Spring boot)

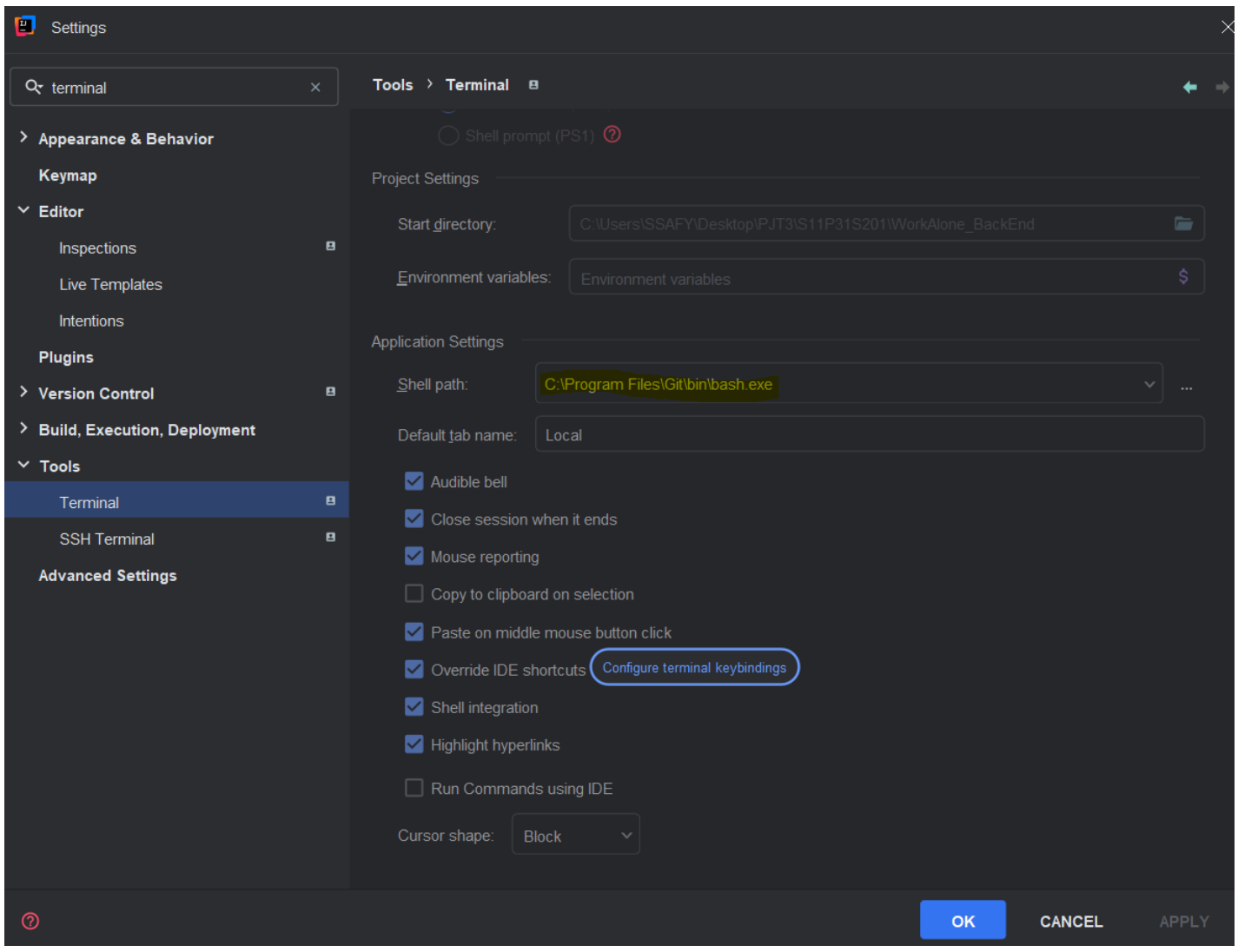
```yaml
spring:
  cloud:
    aws:
      credentials:
        access-key: "AKIA6GBMCJZ6TDZSNOVR"
        secret-key: "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"
      s3: #버킷이름
        bucket: "work-alone"
      region: #S3 지역
        static: "ap-northeast-2"
      stack:
        auto: false
```

사용 포트

컨테이너	포트 번호
nginx	80, 443
jenkins	9090, 50000
spring	8080 (EXPOSE)
mysql	3306 (EXPOSE)

로컬 환경에서 백엔드 실행

1. **IntelliJ terminal ⇒ git-bash 로 변경**
    - **[File - Setting - Terminal]**



2. **Docker Desktop 구동**
3. **git-bash 터미널 창**

```
./play.sh
```

4. **구동 종료 시**

```
docker compose down
```