

EZDG 포팅 매뉴얼

서버사양

CPU	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
RAM	16GB
SSD	320GB SSD
운영체제	Ubuntu 20.04.6 LTS

도커설치

1. Set up Docker's apt repository.

```
# 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/keyrings/docker.asc
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.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Install the Docker packages.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

3. docker compose 설치

```
curl -SL https://github.com/docker/compose/releases/download/v2.30.3/docker-compose-
linux-x86_64 -o /usr/local/bin/docker-compose
```

4. docker compose 권한부여

```
sudo chmod +x /usr/local/bin/docker-compose
```

5. docker-compose 가 제대로 설치되었는지 확인

```
docker-compose --version
# 또는
docker compose version
```

Dockerfile

Jenkins

파일 경로	home/ubuntu/docker-files/jenkins/Dockerfile
-------	---------------------------------------------

```
FROM jenkins/jenkins
USER root
RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-
properties-common && \
    curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian
$(lsb_release -cs) stable" && \
    apt-get update && \
    apt-get install -y docker-ce-cli iputils-ping netcat-openbsd && \
    apt-get clean
RUN groupadd -f docker
RUN usermod -aG docker jenkins
# Docker Compose 설치
RUN curl -L "https://github.com/docker/compose/releases/download/$(curl -s
https://api.github.com/repos/docker/compose/releases/latest | grep -oP '(?<="tag_name":
")[^"]*)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose && \
    chmod +x /usr/local/bin/docker-compose
USER jenkins
```

Ezdg-guide

Jenkins 파이프라인 동작 시 사용합니다.

파일 경로	git repository ezdg-guide 디렉토리 최상단
-------	------------------------------------

```
# 가져올 이미지 정의
FROM node:20.15.0 as build-stage
# 경로 설정
WORKDIR /app
# package.json 워킹 디렉토리에 복사
COPY package*.json ./
# 의존성 설치
RUN npm install
# 결과물 복사
COPY . .
# 빌드하기
RUN npm run build
```

```
# 포트 노출
EXPOSE 3000
# 실행 명령어 정의
CMD ["npm", "start"]
```

Ezdg-auto

Jenkins 파이프라인 동작 시 사용합니다.

파일 경로	git repository ezdg-auto 디렉토리 최상단
-------	-----------------------------------

```
FROM openjdk:17-jdk
LABEL maintainer="gkrhf@naver.com"
ARG JAR_FILE=build/libs/ezdg-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} ezdg-auto.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/ezdg-auto.jar"]
```

Ezdg-api-server

Jenkins 파이프라인 동작 시 사용합니다.

파일 경로	git repository ezdg-api-server 디렉토리 최상단
-------	-----------------------------------------

```
FROM openjdk:17-jdk
ARG JAR_FILE=build/libs/ezdg_api_server-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} ezdg-api-server.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/ezdg-api-server.jar"]
```

Ezdg-croller

Jenkins 파이프라인 동작 시 사용합니다.

파일 경로	git repository ezdg-croller 최상단
-------	---------------------------------

```
FROM python:3.12.7
# 필수 라이브러리 설치
RUN apt update && apt install -y wget unzip libxpm4 libxrender1 libgtk2.0-0 libnss3
libgconf-2-4 fonts-liberation
# Chrome 설치
RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb \
    && apt install -y ./google-chrome-stable_current_amd64.deb \
    && rm ./google-chrome-stable_current_amd64.deb # 설치 후 .deb 파일 제거
# ChromeDriver 버전을 Chrome 버전과 일치시키기 위한 설정
# RUN CHROME_DRIVER_VERSION=$(curl -sS
https://chromedriver.storage.googleapis.com/LATEST_RELEASE) && \
#   wget -O /tmp/chromedriver.zip
https://chromedriver.storage.googleapis.com/$CHROME_DRIVER_VERSION/chromedriver_linux64.
zip && \
#   unzip /tmp/chromedriver.zip -d /usr/local/bin/ && \
#   rm /tmp/chromedriver.zip # 설치 후 .zip 파일 제거
# 환경 설정
ENV DISPLAY=:99
# 작업 디렉토리 설정
WORKDIR /app
# 필요한 패키지 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# 애플리케이션 코드 복사
COPY . .
# uvicorn을 통해 FastAPI 앱 실행
CMD ["uvicorn", "Server:app", "--host", "0.0.0.0", "--port", "8000"]
```

NGINX

설정파일을 작성합니다

파일 경로	home/ubuntu/docker-files/default.conf
-------	---------------------------------------

```
server {
    listen 80;
    server_name k11d201.p.ssafy.io;

    location /mongodb/ {
        proxy_pass http://mongodb:27017; # MongoDB 컨테이너 이름을 사용하여 내부
        # 네트워크에서 접근
        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;
    }

    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
}
server {
    listen 443 ssl;
    server_name k11d201.p.ssafy.io;

    charset utf-8;
    client_max_body_size 10M;
    ssl_certificate /etc/letsencrypt/live/k11d201.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k11d201.p.ssafy.io/privkey.pem;
    # SSL settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers HIGH:!aNULL:!MD5;
    location / {
        proxy_pass http://ezdg-guide:3000/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /admin/ {
        #rewrite ^/admin/(.*)$ /$1 break;
        #proxy_pass http://ezdg-auto:8080/admin;
        proxy_pass http://ezdg-auto:8080/;
        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_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
    location /croller/ {
        #rewrite ^/croller/(.*)$ /$1 break;
```

```

    proxy_pass http://ezdg-croller:8000/;
    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_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
location /api/ {
    #rewrite ^/api/(.*)$ /$1 break;
    proxy_pass http://ezdg-api-server:8080/;
    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_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
}

```


docker-compose.yml

Jenkins 용 docker-compose.yml

파일 경로	home/ubuntu/docker-files/jenkins/docker-compose.yml
-------	-----------------------------------------------------

```
version: "3"
services:
  jenkins:
    build:
      context: .
      dockerfile: /home/ubuntu/docker-files/jenkins/Dockerfile
    container_name: jenkins-ezdg
    restart: always
    ports:
      - "9091:8080"
    volumes:
      - /home/ubuntu/docker-files/jenkins-data:/var/jenkins_home
      - /home/ubuntu/docker-files:/docker-files
      - /var/run/docker.sock:/var/run/docker.sock
    user: root
```

home/ubuntu/docker-files/jenkins 경로로 이동합니다.

실행

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

중지

```
docker-compose down
```

Jenkins 외 docker-compose.yml

파일 경로	home/ubuntu/docker-files/docker-compose.yml
-------	---------------------------------------------

```
version: "3"
services:
  ezdg-auto:
    image: ezdg-auto # 빌드된 Spring Boot 이미지 이름
    container_name: ezdg-auto
    depends_on:
      - mongodb # mongodb 서비스가 먼저 시작되도록 설정
      - mysql
    ports:
      - "8080:8080" # Spring Boot 포트 매핑
    volumes:
      - /home/ubuntu/docker-files/files:/app/files
      - /home/ubuntu/release/ezdg-api-server/src/main/java/com/openmind/ezdg_api_server:/app/files/api-server
      - /home/ubuntu/release/ezdg-javalib/src/main/java/com/openmind/ezdg:/app/files/javalib
      - /home/ubuntu/release_commit.sh:/app/files/release_commit.sh

  ezdg-api-server:
    image: ezdg-api-server
    container_name: ezdg-api-server
    depends_on:
      - mongodb
    environment:
      SPRING_DATA_MONGODB_URI:
mongodb://admin:openmindd201mongo@mongodb:27017/data?authSource=admin
    ports:
      - "8081:8080"

  ezdg-croller:
    image: ezdg-croller
    container_name: ezdg-croller
    ports:
      - "8000:8000"

  ezdg-guide:
    image: ezdg-guide
    container_name: ezdg-guide
    ports:
      - "3000:3000"

  mysql:
    image: mysql
    container_name: mysql-ezdg
    restart: always
    environment:
      MYSQL_DATABASE: ezdg
      MYSQL_ROOT_PASSWORD: openmind1!
      MYSQL_USER: ezdg
      MYSQL_PASSWORD: ezdg
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    volumes:
      - /home/ubuntu/docker-files/mysql-data:/var/lib/mysql
```

```

redis:
  container_name: redis-ezdg
  image: redis
nginx:
  image: nginx
  container_name: nginx-ezdg
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - /home/ubuntu/docker-files/default.conf:/etc/nginx/conf.d/default.conf
    - /etc/letsencrypt/live/k11d201.p.ssafy.io:/etc/letsencrypt/live/k11d201.p.ssafy.io
    - /etc/letsencrypt/archive/k11d201.p.ssafy.io:/etc/letsencrypt/archive/k11d201.p.ssafy.io
    - /etc/letsencrypt/renewal:/etc/letsencrypt/renewal
  depends_on:
    - ezdg-auto
    - ezdg-croller
    - ezdg-guide
    - ezdg-api-server
mongodb:
  image: mongo:latest
  container_name: mongodb-ezdg # 컨테이너 이름
  # ports:
  #   #- "27017:27017" # 호스트의 27017 포트를 컨테이너의 27017 포트에 매핑
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin # 관리자 계정 이름
    MONGO_INITDB_ROOT_PASSWORD: openmindd201mongo # 관리자 계정 비밀번호
  ports:
    - "27017:27017"
  volumes:
    - mongo_data:/data/db # 데이터가 영구적으로 저장될 볼륨
volumes:
  mongo_data:
    driver: local # 로컬 볼륨을 사용하여 데이터 유지

```

home/ubuntu/docker-files 경로로 이동합니다.

실행

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

중지

```
docker-compose down
```

Jenkins 설치 및 실행

젠킨스 대시보드 접속을 위해 포트를 허용합니다(9091)

```
sudo ufw allow 9091 /tcp
sudo ufw reload
sudo ufw status
```

- docker-compose.yml 파트를 참고하여 젠킨스 컨테이너를 실행합니다.
- 아래 명령어 실행 후 초기 패스워드를 복사합니다.

```
sudo docker logs Jenkins
```

- <http://k11d201.p.ssafy.io:9091> 젠킨스 대시보드에 접속해서 초기 비밀번호를 입력합니다.
- 추천 플러그인 설치 후 어드민 계정을 생성합니다.
- jenkins 관리 → plugins → Available plugins 로 이동해 아래 plugin 들을 추가로 설치합니다
 - NodeJS Plugin
 - Gitlab
- jenkins 관리 → Tools 로 이동 후 아래와 같이 nodejs 설정을 추가합니다

NodeJS Installations

NodeJS installations ^ Edited

Add NodeJS

NodeJS

Name

NodeJS 20.15.0

☒ Install automatically ?

Install from nodejs.org

Version

NodeJS 20.15.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

☐ Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax "packageName@version"

Global npm packages refresh hours

Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

72

Credential 추가

- Jenkins 관리 → Credential 로 이동
- global 도메인 클릭 → Add Credential

gitlab access token

kind	Username with password
scope	Global (Jenkins, nodes, items, all child items, etc)
username	깃랩계정 ex) user1234@gmail.com
password	gitlab 에서발급받은 accessToken
id	jenkins-acces-token

2. ssh key

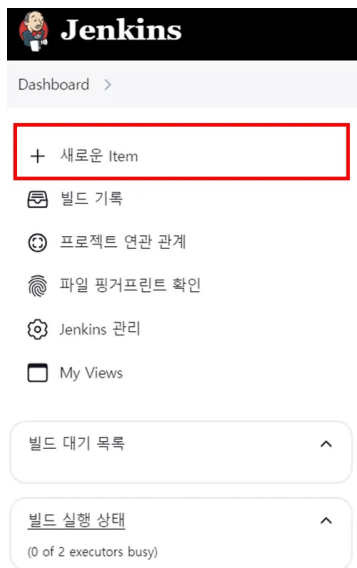
kind	SSH Username with private key
scope	Global (Jenkins, nodes, items, all child items, etc)
id	ssh-key
username	ubuntu
private key	add 클릭 후 pem 파일을 메모장으로 열어서 복사-붙여넣기

Jenkins pipeline

- gitlab webhook 과 연동해 gitlab 에 새로운 코드가 push 됐을 때 서버에 자동으로 배포되도록 합니다.
- 총 5 개의 아이템이 있고 등록 과정은 아래와 같습니다.

1. jenkins item 등록





1.1. jenkins 대시보드 → 새로운 Item



1.2. 아이템 이름 등록 및 파이프라인 선택

Enter an item name

Select an item

-  **Freestyle project**
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
-  **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
-  **Multi-configuration project**
다양한 환경에서의 테스트, 플랫폼 특성 빌드, 기타 등등처럼 다수의 서로 다른 환경설정이 필요한 프로젝트에 적합함.
-  **Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

1.3. 구성

- gitlab webhook URL 은 gitlab Webhook 에서 사용됩니다.

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k11d201.p.ssafy.io:9091/project/ezdg-auto> ?

Enabled GitLab triggers

- ☒ Push Events ?
- ☐ Push Events in case of branch delete ?
- ☒ Opened Merge Request Events ?
- ☐ Build only if new commits were pushed to Merge Request ?
- ☐ Accepted Merge Request Events ?
- ☐ Closed Merge Request Events ?

Rebuild open Merge Requests ?

Never

- ☒ Approved Merge Requests (EE-only) ?
- ☒ Comments ?

Comment (regex) for triggering a build ?

Jenkins please retry a build

secret token 은 gitlab webhook 등록 시 사용됩니다.

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

Generate

1.4.pipeline

ezdg-auto

```
pipeline {
    agent any
    environment {
        // 환경 변수 설정
        AUTO_IMAGE_NAME = "ezdg-auto" // 도커 이미지 이름
        AUTO_CONTAINER_NAME = "ezdg-auto" // 도커 컨테이너 이름
    }
    stages {
        stage('Clean Workspace') {
            steps {
                deleteDir() // Jenkins 작업 공간을 초기화
            }
        }
        stage('Clone Repository') {
            steps {
                script {
                    // 오래된 브랜치 삭제
                    sh 'git remote prune origin || true'
                    // sh 'rm -rf ezdg-auto' // 기존 디렉토리 제거
                    // sh 'git clone -b dev-auto https://lab.ssafy.com/s11-final/S11P31D201.git ezdg-auto'
                    // GitLab 에서 deploy 브랜치에 있는 프로젝트를 클론
                    git branch: 'dev-auto', credentialsId: 'jenkins-access-token', url:
                    'https://lab.ssafy.com/s11-final/S11P31D201.git'
                }
            }
        }
        stage('Replace application.yml') {
            steps {
                script {
                    sh 'cp /docker-files/ezdg-auto/application.yml ezdg-
                    auto/src/main/resources/application.yml'
                }
            }
        }
        stage('Build Auto Project') {
            steps {
                // 프로젝트 빌드
                // backend 디렉토리에서 아래 명령어 실행
                dir('ezdg-auto'){
                    sh '''
                        chmod +x ./gradlew
                        ./gradlew clean build -x test
                    '''
                    // -x test 로 테스트 진행 X
                }
            }
        }
        stage('Build Docker Image') {
```

```

    steps {
        // Docker 이미지 빌드
        // Dockerfile 이 /backend 디렉토리에 있어야 한다.
        // 빌드한 백엔드 jar 파일로 도커 이미지를 생성하는 것
        dir('ezdg-auto'){
            sh "docker build -t ${AUTO_IMAGE_NAME} ."
        }
    }
}
stage('Stop and Remove Old Container') {
    steps {
        script {
            // 기존 컨테이너 중지 및 삭제
            // 만약 ezdg-auto 라는 이름의 컨테이너가 존재한다면 아래 코드 실행
            // 도커에서는 컨테이너 이름이 유니크 값이라 중복되면 안됩니다.
            def containerExists = sh(script: "docker ps -a --filter
\"name=${AUTO_CONTAINER_NAME}\" --format \"{{.Names}}\"", returnStdout: true).trim()

            if (containerExists) {
                sh "docker stop ${AUTO_CONTAINER_NAME}" // Stop the container
                sh "docker rm ${AUTO_CONTAINER_NAME}" // Remove the container
            } else {
                echo "Container ${AUTO_CONTAINER_NAME} does not exist."
            }
        }
    }
}
stage('Run New Container') {
    steps {
        script {
            // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
            sh '''
docker-compose -f /docker-files/docker-compose.yml stop ezdg-auto
docker-compose -f /docker-files/docker-compose.yml rm -f ezdg-auto
docker-compose -f /docker-files/docker-compose.yml up -d --build ezdg-
auto
            '''
        }
    }
}
}
post {
    success {
        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed.'
    }
}
}
}

```

ezdg-api-server

```
pipeline {
  agent any
  environment {
    // 환경 변수 설정
    IMAGE_NAME = "ezdg-api-server" // 도커 이미지 이름
    CONTAINER_NAME = "ezdg-api-server" // 도커 컨테이너 이름
  }
  stages {
    stage('Clean Workspace') {
      steps {
        deleteDir() // Jenkins 작업 공간 전체를 초기화
      }
    }
    stage('Clone Repository') {
      steps {
        script {
          // 오래된 브랜치 삭제
          sh 'git remote prune origin || true'

          // GitLab 에서 deploy 브랜치에 있는 프로젝트를 클론
          // git branch: 'dev-api-server', credentialsId: 'jenkins-access-token',
          url: 'https://lab.ssafy.com/s11-final/S11P31D201.git'
          git branch: 'master', credentialsId: 'jenkins-access-token', url:
          'https://lab.ssafy.com/s11-final/S11P31D201.git'
        }
      }
    }
    stage('Build Auto Project') {
      steps {
        // 프로젝트 빌드
        // backend 디렉토리에서 아래 명령어 실행
        dir('ezdg-api-server'){
          sh '''
            chmod +x ./gradlew
            ./gradlew clean build -x test
            ...
            //-x test 로 테스트 진행 X
          '''
        }
      }
    }
    stage('Build Docker Image') {
      steps {
        // Docker 이미지 빌드
        dir('ezdg-api-server'){
          sh "docker build -t ${IMAGE_NAME} ."
        }
      }
    }
    stage('Stop and Remove Old Container') {
      steps {
        script {
          // 기존 컨테이너 중지 및 삭제
          def containerExists = sh(script: "docker ps -a --filter
          \"name=${CONTAINER_NAME}\" --format \"{{.Names}}\"", returnStdout: true).trim()
```

```

        if (containerExists) {
            sh "docker stop ${CONTAINER_NAME}" // Stop the container
            sh "docker rm ${CONTAINER_NAME}"    // Remove the container
        } else {
            echo "Container ${CONTAINER_NAME} does not exist."
        }
    }
}
}
stage('Run New Container') {
    steps {
        script {
            // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
            sh '''
server          docker-compose -f /docker-files/docker-compose.yml stop ezdg-api-
server          docker-compose -f /docker-files/docker-compose.yml rm -f ezdg-api-
api-server      docker-compose -f /docker-files/docker-compose.yml up -d --build ezdg-
                ...
            '''
        }
    }
}
}
post {
    success {
        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed.'
    }
}
}
}

```

ezdg-croller

```
pipeline {
  agent any
  environment {
    // 환경 변수 설정
    IMAGE_NAME = "ezdg-croller" // 도커 이미지 이름
    CONTAINER_NAME = "ezdg-croller" // 도커 컨테이너 이름
  }
  stages {
    stage('Clean Workspace') {
      steps {
        deleteDir() // Jenkins 작업 공간 전체를 초기화
      }
    }
    stage('Clone Repository') {
      steps {
        script {
          // 오래된 브랜치 삭제
          sh 'git remote prune origin || true'

          // GitLab 에서 deploy 브랜치에 있는 프로젝트를 클론
          git branch: 'dev-croller', credentialsId: 'jenkins-access-token', url:
'https://lab.ssafty.com/s11-final/S11P31D201.git'
        }
      }
    }
    stage('Build Docker Image') {
      steps {
        // Docker 이미지 빌드
        dir('ezdg-croller'){
          sh "docker build -t ${IMAGE_NAME} ."
        }
      }
    }
    stage('Stop and Remove Old Container') {
      steps {
        script {
          // 기존 컨테이너 중지 및 삭제
          def containerExists = sh(script: "docker ps -a --filter
\"name=${CONTAINER_NAME}\" --format \"{{.Names}}\"", returnStdout: true).trim()

          if (containerExists) {
            sh "docker stop ${CONTAINER_NAME}" // Stop the container
            sh "docker rm ${CONTAINER_NAME}" // Remove the container
          } else {
            echo "Container ${CONTAINER_NAME} does not exist."
          }
        }
      }
    }
    stage('Run New Container') {
      steps {
        script {
          // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
          sh '''
docker-compose -f /docker-files/docker-compose.yml stop ezdg-croller
docker-compose -f /docker-files/docker-compose.yml rm -f ezdg-croller
'''
        }
      }
    }
  }
}
```

```
docker-compose -f /docker-files/docker-compose.yml up -d --build ezdg-
croller
    ...
  }
}
}
post {
  success {
    echo 'Pipeline completed successfully!'
  }
  failure {
    echo 'Pipeline failed.'
  }
}
}
```

ezdg-guide

```
pipeline {
  agent any
  environment {
    IMAGE_NAME = "ezdg-guide"
    CONTAINER_NAME = "ezdg-guide"
  }
  stages {
    stage('Clone Repository') {
      steps {
        script {
          // 오래된 브랜치 삭제
          sh 'git remote prune origin || true'
        }

        // GitLab 에서 프로젝트를 클론
        git branch: 'dev-guide', credentialsId: 'jenkins-access-token', url:
'https://lab.ssafy.com/s11-final/S11P31D201.git'
      }
    }

    stage('input .env') {
      steps {
        script {
          sh 'cp /docker-files/ezdg-guide/.env ezdg-guide/.env'
        }
      }
    }

    stage('Stop and Remove Old Container') {
      steps {
        script {
          script {
            def containerExistsFront = sh(script: "docker ps -a --filter
\"name=${CONTAINER_NAME}\" --format \"{{.Names}}\"", returnStdout: true).trim()
            if (containerExistsFront) {
              sh "docker stop ${CONTAINER_NAME}"
              sh "docker rm ${CONTAINER_NAME}"
            } else {
              echo "Container ${CONTAINER_NAME} does not exist."
            }
          }
        }
      }
    }

    stage('Build Docker Image') {
      steps {
        dir('ezdg-guide'){
          sh "docker build -t ${IMAGE_NAME} ."
        }
      }
    }

    stage('Run New Container') {
      steps {
        script {
          // 새 이미지로 컨테이너 실행, 불필요 이미지 제거
          sh '''
```

```
        docker-compose -f /docker-files/docker-compose.yml stop ezdg-guide
        docker-compose -f /docker-files/docker-compose.yml rm -f ezdg-guide
        docker-compose -f /docker-files/docker-compose.yml up -d --build ezdg-
guide
        ...
    }
}
}
post {
    success {
        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed.'
    }
}
}
```


ezdg-deploy

```
def TAG_NAME = ''
pipeline {
    agent any
    environment {
        TAG_REGEX = '^v\\d+\\.\\.\\d+\\.\\.\\d+$' // 원하는 태그 형식 (예: v1.0.0)
    }
    stages {
        stage('Debug') {
            steps {
                script {
                    echo "gitlabBranch: ${env.gitlabBranch}"
                    echo "gitlabActionType: ${env.gitlabActionType}"
                    echo "GIT_BRANCH: ${env.GIT_BRANCH}"
                    echo "TAG_NAME: ${TAG_NAME}"
                }
            }
        }

        stage('Prepare') {
            steps {
                script {
                    TAG_NAME = env.gitlabBranch.replace('refs/tags/', '')
                }
            }
        }

        stage('Checkout') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11P31D201.git',
                    credentialsId: 'jenkins-access-token',
                    branch: "master",
                    changelog: false,
                    poll: false
            }
        }

        stage('Release') {
            when {
                expression {
                    return TAG_NAME =~ /${TAG_REGEX}/
                }
            }
            steps {
                echo "릴리즈 준비 중 - 태그: ${TAG_NAME}"
                // 빌드, 패키징, 배포 등의 릴리즈 작업 수행
                dir('ezdg-javalib') {
                    withEnv(["TAG_NAME=${TAG_NAME}"]) {
                        sh '''
                        chmod +x ./gradlew

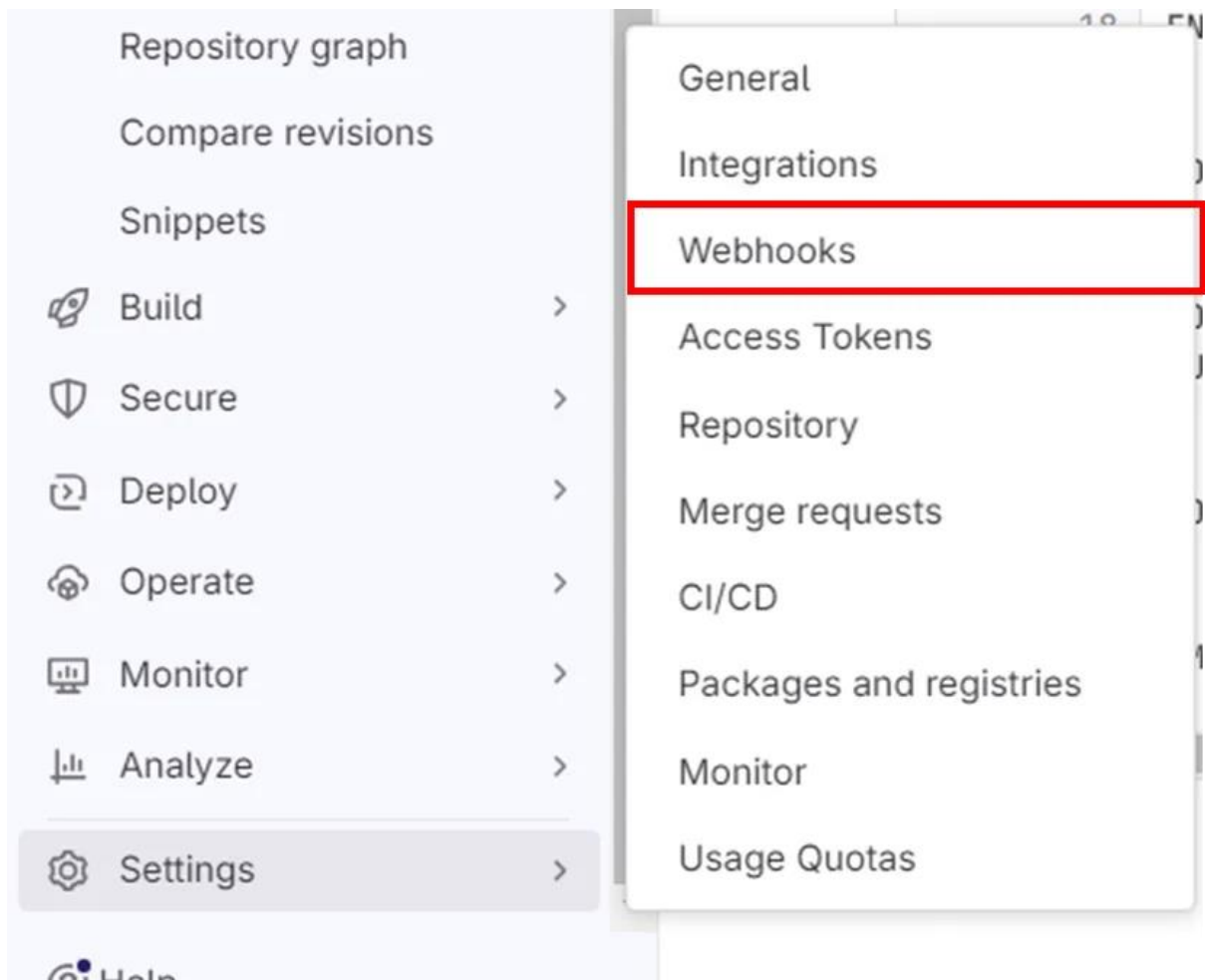
                        ./gradlew clean build \
                        -
                        PrepositoryUrl=https://lab.ssafy.com/api/v4/projects/823781/packages/maven \
                        -PrepositoryUsername=kyoungmop \
                        -PrepositoryPassword=gldt-3pDCB-ixBg9KcyRyYzXs \
                        -Pversion=${TAG_NAME}

                        ./gradlew publish \
                    '''
                    }
                }
            }
        }
    }
}
```

```
-  
PrepositoryUrl=https://lab.ssafy.com/api/v4/projects/823781/packages/maven \  
-PrepositoryUsername=kyoungmop \  
-PrepositoryPassword=gldt-3pDCB-ixBg9KcyRyYzXs \  
-Pversion=$TAG_NAME  
...  
}  
}  
}  
}  
}  
}
```

gitlab webhook 등록

2.1. gitlab → settings - webhooks



2.2. 새로운 webhook 등록

- 젠킨스 아이템에 있는 URL, Secret token 을 입력합니다.
- trigger 로 Push events 를 선택하고 regular expression 을 아래와 같이 입력합니다
 - ezdg-deploy 는 tag push event 를 선택합니다.

젠킨스아이템	정규표현식
dev-auto	^dev-auto\$
dev-api-server	^dev-api-server\$
dev-croller	^dev-croller\$
dev-guide	^dev-guide\$

URL

URL must be percent-encoded if it contains one or more special characters.

☒ Show full URL

☐ Mask portions of URL

Do not show sensitive data such as tokens in the UI.

Custom headers </> 0

Add custom header

No custom headers configured.

Name (optional)

Description (optional)

Secret token

Used to validate received payloads. Sent with the request in the `X-GitLab-Token` HTTP header.

Trigger

☒ Push events

☐ All branches

☐ Wildcard pattern

☒ Regular expression

Regular expressions such as `^(feature|hotfix)/` are supported.

자바 라이브러리 배포

- ezdg-javalib 배포 프로세스는 다음과 같습니다.

1. 관리자페이지에서 배포하기 클릭
2. webhook(release_script_hook) 호출
3. release_script_hook.sh 이 실행됨
 - 생성된 api-server 코드 javalib 코드를 커밋 후 master branch 에 push
 - version.txt 에 명시된 버전으로 git tag 등록 후 gitlab 으로 push
4. tag push 이벤트를 감지해 jenkins pipe 라인 동작
 - gradle publish 를 통해 라이브러리 배포

webhook

- 호스트 OS 에서 Webhook 리스너를 설정하여, dev-auto 서버가 Webhook 을 호출하면 호스트 OS 에서 스크립트를 실행하게 합니다.

1. 호스트 OS 에 Webhook 리스너 설치

```
sudo apt install webhook
```

2. Webhook 설정 파일: `/home/username/release_script_hook.json` 에 설정을 추가합니다.

```
3.  
4. [  
5.   {  
6.     "id": "release-script",  
7.     "execute-command": "/home/ubuntu/release_commit.sh",  
8.     "command-working-directory": "/home/ubuntu",  
9.     "response-message": "Script executed successfully!"  
10.  }  
11. ]  
12.
```

3. 스크립트 실행권한 부여

- webhook 리스너를 실행하는 사용자와 스크립트가 실행되는 위치의 접근 권한을 동일하게 설정합니다.

```
chmod +x /home/ubuntu/release_commit.sh # 스크립트에 실행 권한 부여
```

4. Webhook 리스너 시작:

```
webhook -hooks /home/ubuntu/release_script_hook.json -port 9000
```

Webhook 백그라운드에서 계속 실행시키기

1. Webhook 실행 명령어 작성

`nohup`을 사용하여 Webhook 을 백그라운드에서 실행하려면 다음과 같이 명령어를 작성합니다:

```
bash
코드 복사
nohup /usr/local/bin/webhook -hooks /home/ubuntu/release_script_hook.json -port 9000 > /home/username/webhook.log 2>&1 &
```

- `nohup`: 터미널 세션이 종료되더라도 명령어가 계속 실행되도록 합니다.

- `/usr/local/bin/webhook`: Webhook 실행 파일의 경로입니다. `which webhook` 명령으로 정확한 경로를 확인할 수 있습니다.

- `-hooks /home/ubuntu/release_script_hook.json`: Webhook 설정 파일 경로입니다. 이 파일에 Webhook 실행에 필요한 설정을 지정해야 합니다.

- `-port 9000`: Webhook 이 수신할 포트를 지정합니다. 필요에 따라 다른 포트를 지정할 수 있습니다.

- `> /home/ubuntu/webhook.log 2>&1`: 로그 출력을 `webhook.log` 파일로 리다이렉트합니다.

- `2>&1`: 표준 오류(stderr)도 표준 출력(stdout)으로 리다이렉트하여 모든 로그가 `webhook.log`에 기록되도록 합니다.

- `&`: 마지막에 `&`를 붙여 백그라운드에서 실행되도록 합니다.

Shell Script

파일 경로	home/ubuntu/release_commit.sh
-------	-------------------------------

```
#!/bin/bash
cd /home/ubuntu/release
VERSION=$(cat /home/ubuntu/version.txt)
if [[ -z "$VERSION" ]]; then
    exit 1
fi
git checkout master
git add .
git commit -m ":rocket: $(date '+%Y-%m-%d %H:%M:%S') deploy"
git tag "$VERSION"
git push origin master
git push origin "$VERSION"
# 기존 버전을 major, minor, patch 로 분리
IFS='.' read -r major minor patch <<< "${VERSION//v/}"
# minor 버전을 증가시키고 patch 버전은 0 으로 초기화
NEW_VERSION="v$major.$((minor + 1)).0"
# version.txt 파일에 새 버전 저장
echo "$NEW_VERSION" > /home/ubuntu/version.txt
```

version.txt

- v{major}.{minor}.{patch}
- 다음에 배포될 ezdg java 라이브러리의 버전이 기록되어있습니다.
- release_commit.sh 동작 시 minor 버전이 1 증가합니다.

파일 경로	home/ubuntu/version.txt
-------	-------------------------

v1.0.0