AGORA

역삼역 인력 사무소

목 차

✓빌드 및 배포

✓외부 서비스

✓ 빌드 및 배포

1. 개발 환경

Client

- NodeJS 18.14.0 LTS
- ReactJS 18.2.0
- Axios 1.2.4
- Recoil 0.7.6
- Styled-component 5.3.6

Server

- Openjdk 11
- Spring Boot 2.7.7
- Spring Security 2.7.7 starter
- Spring Data JPA 2.7.7 starter
- Swagger 3.0.0
- jjwt 0.9.1
- QueryDSL 1.0.10

1. 개발 환경

Database

- MYSQL 8.0
- Redis 7.0

Infra

- Docker 20.10.23
- Jenkins 2.375.2
- Nginx 1.18.0
- AWS
- GCP

2. 설정 파일

Server

- application-prod.yml server/src/main/resources
- gcp-account-file.json server/src/main/resources

Infra(AWS)

- custom.conf /etc/nginx/conf.d/custom.conf

Infra(GCP)

- .env /opt/openvidu/.env

- gcp-account-file.json 파일은 gcp에서 제공됩니다.

3. 환경 변수

Server

spring.config.active	
on-profile	실행 환경
spring.redis	
host	redis host
port	redis port
password	redis password
spring.gcp	
config.file	gcp 설정 파일
project.id	gcp project id
bucket.id	저장되는 bucket id
dir.name	이미지 저장 폴더
mvc.pathmatch	
matching-strategy	swagger 방식
datasource	
url	mysql url
username	mysql username
password	mysql password
driver-class-name	사용할 driver 종류

Server

spring.security.oauth2	
client.registration	
google	
client-id	google client id
client-secret	google secret
redirect-uri	google login redirect uri
scope	google로부터 받을 정보
server.port	실행할 port 번호
openvidu.hostname	openvidu hostname
openvidu.secret	openvidu secret

Infra (custom.conf)

```
location /api {
        proxy_pass (backend port url)
}
# client 에서 요청은 /api/oauth/google로 요청
location /api/oauth/google {
        proxy_pass
        http://localhost:8081/oauth2/authorization/google
}

location /api/oauth2/callback/google {
        proxy_pass
        http://localhost:8081/oauth2/callback/google
}

server {
        listen 80;
        server_name domain name;
        return 301 https://$server_name$request_uri;
}
```

Infra(openvidu .env)

DOMAIN_OR_PUBLIC_IP	프로젝트 domain name
OPENVIDU_SECRET	openvidu secret
CERTIFICATE_TYPE	인증서 타입 선택
LETSENCRYPT_EMAIL	letsencrypt 선택 시 유효한 이메일 작성

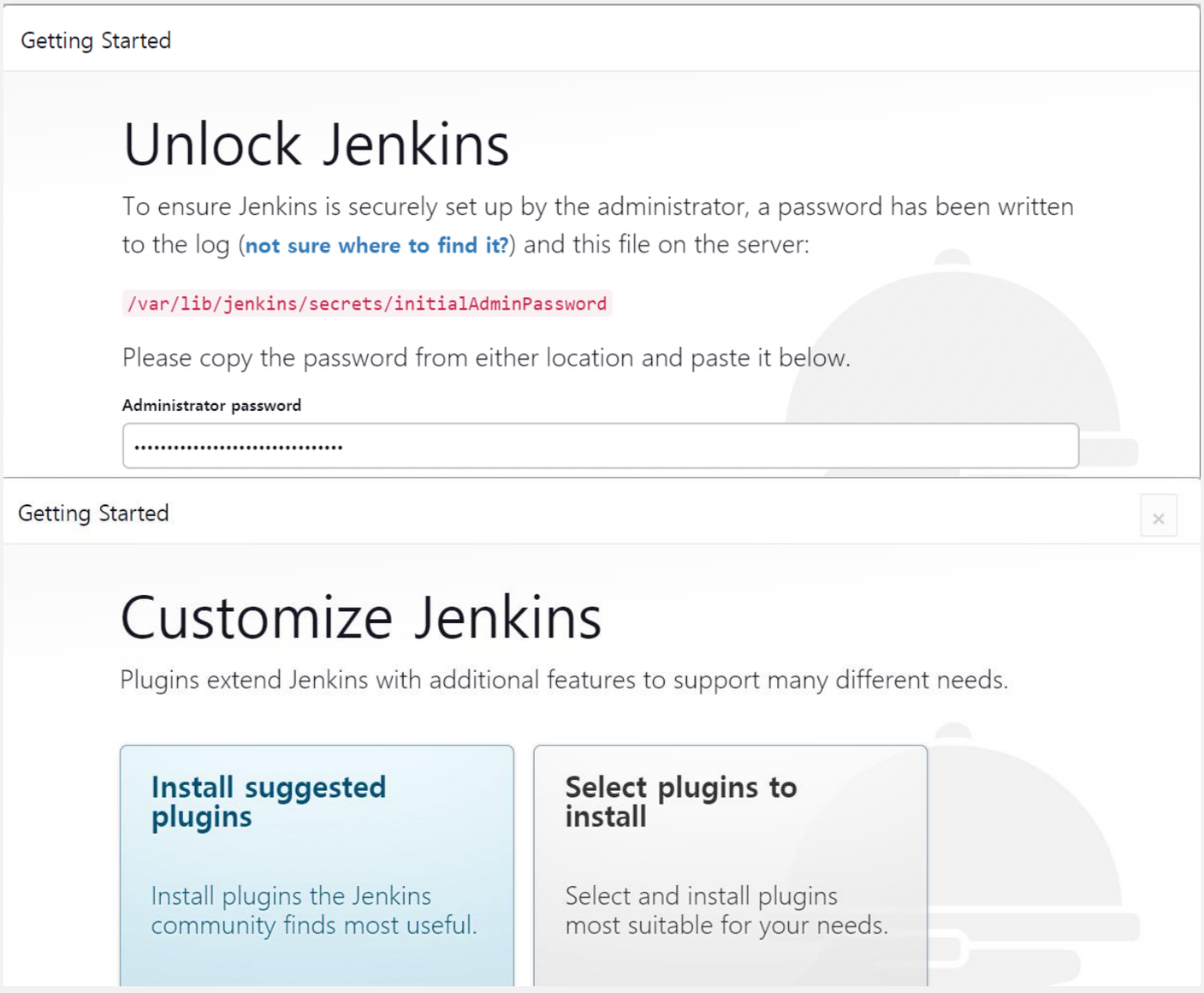
4. Jenkins

설치

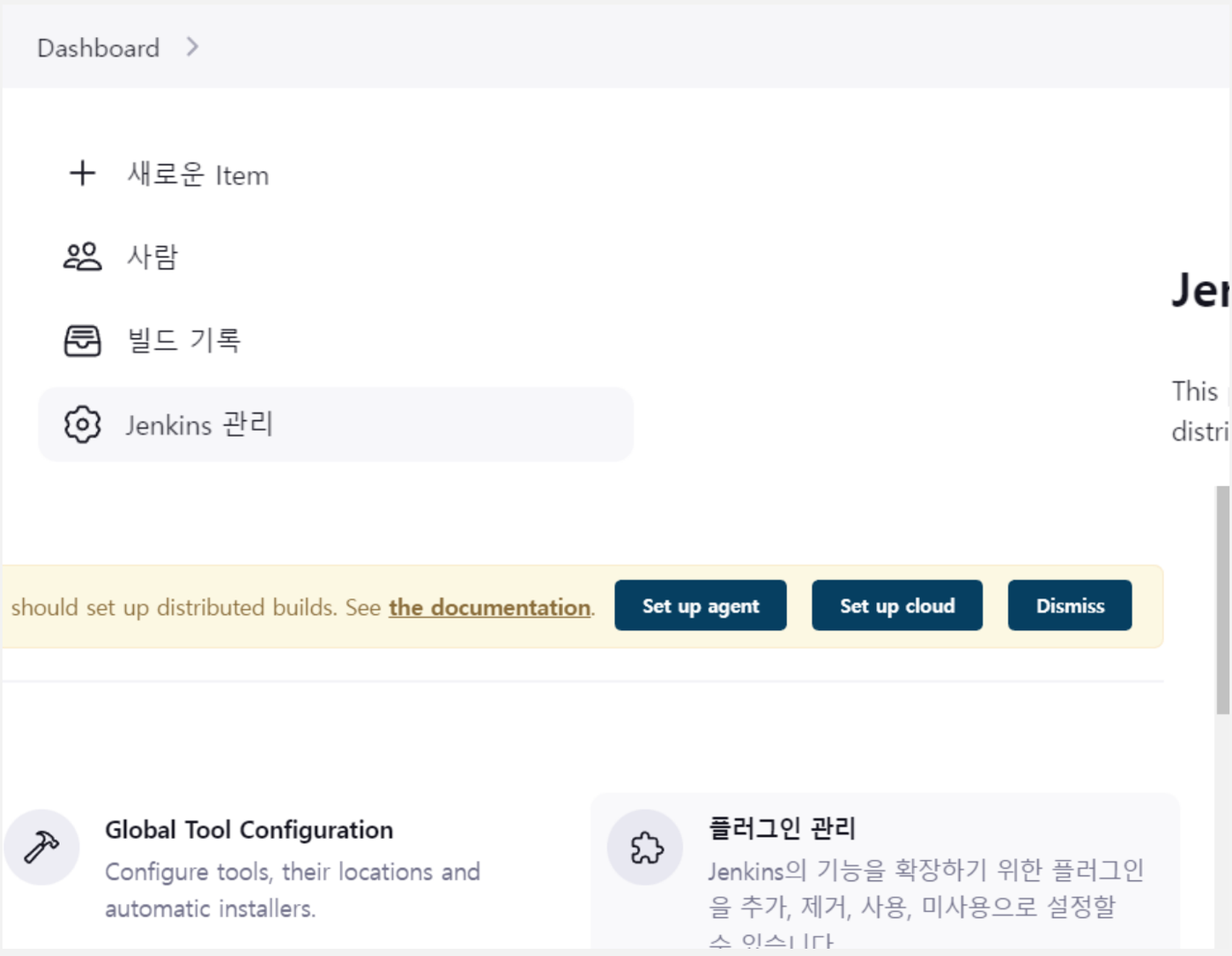
```
sudo apt-get update

sudo apt-get install openjdk-11-jdk –y

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins
```

- 공식문서에 따른 설치 명령어입니다.

설정



- 설치 후 domain:8080 으로 접속
- sudo cat /var/lib~~ 명령어로 password 확인
- 필수 플러그인 설치
- 설치 완료 후 Jenkins 접속 password 설정



- Jenkins 관리 -> 플러그인 관리 -> available plugins
- 필요한 플러그인 설치
- docker와 gitlab을 검색하여 연관된 플러그인 모두 설치

5. Docker

설치

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -m 0755 -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

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

- 공식문서에 따른 설치 명령어입니다.

설정

sudo chmod 666 /var/run/docker.sock

- docker 권한 변경하여 sudo 키워드를 넣지 않아도 됩니다.

6. Nginx

설치

sudo apt update –y

sudo apt install nginx –y

SSL인증서 설치

sudo apt remove certbot

sudo snap install --classic certbot

sudo certbot --nginx

7. Openvidu

설치

sudo su

cd /opt

curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash

8. Database

MYSQL 설치

```
docker run –name mysql –d –p 3306:3306 –e
MYSQL_ROOT_PASSWORD=PW mysql:8
```

- docker run 명령어로 mysql을 컨테이너에서 실행

Redis 설치

```
docker run --name redis --network name -p 6379:6379 -d redis
--requirepass PASSWORD
```

- docker run 명령어로 redis 를 컨테이너에서 실행
- --requirepass로 password 필수 지정

9. CI/CD

Client Dockerfile

```
FROM node:14
ENV REACT_APP_SERVER_BASE_URL (backend base url)
WORKDIR /app
COPY client/package*.json ./
RUN npm install
COPY client/. .
RUN npm run build

FROM nginx:alpine
COPY --from=0 /app/build /usr/share/nginx/html
COPY client/nginx/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]

# client/nginx/nginx.conf
server {
  listen 3000;
  location / {
    root /usr/share/nginx/html;
    index index.html;
    try_files $uri $uri/ /index.html;
  }
}
```

Client Dockerfile 설명

- ENV로 컨테이너 실행 시 추가할 환경 변수 추가
- jenkins에서 프로젝트 클론 후 빌드되기 때문에 client/폴더로 파일 접근
- 빌드된 파일 컨테이너 nginx 위치로 이동
- client nginx.conf 파일에 빌드된 페이지 경로 설정

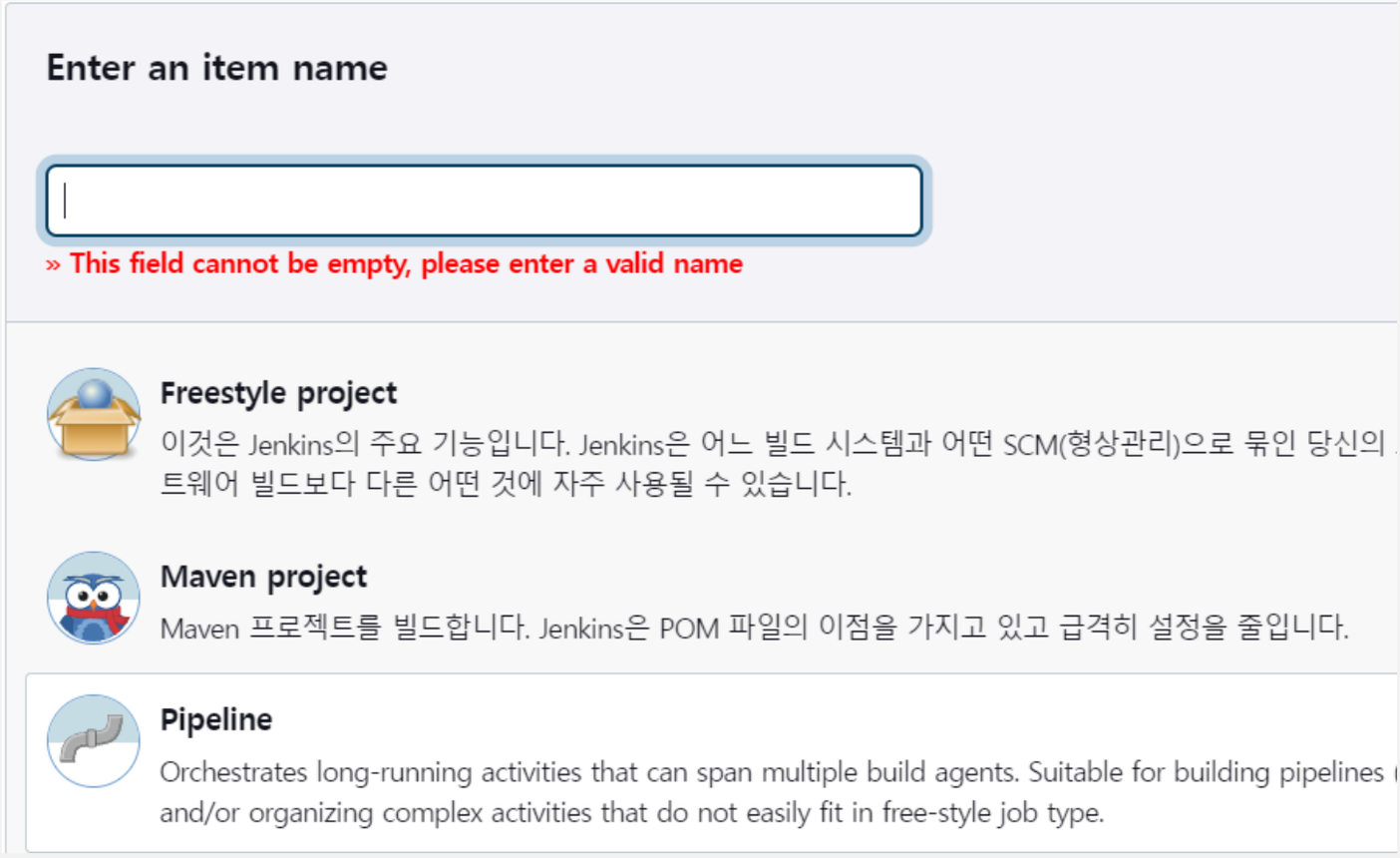
Server Dockerfile

```
FROM adoptopenjdk:11-hotspot AS builder
ENV USE_PROFILE dev
…
COPY server/gradlew .
COPY server/gradle gradle
COPY server/build.gradle .
COPY server/settings.gradle .
COPY server/src src
RUN chmod +x ./gradlew
RUN ./gradlew clean compileQuerydsl
RUN ./gradlew clean bootJar

FROM adoptopenjdk:11-hotspot
COPY --from=builder build/libs/*.jar app.jar

EXPOSE 8081
ENTRYPOINT ["java","-jar", “ENV 설정” "/app.jar"]
```

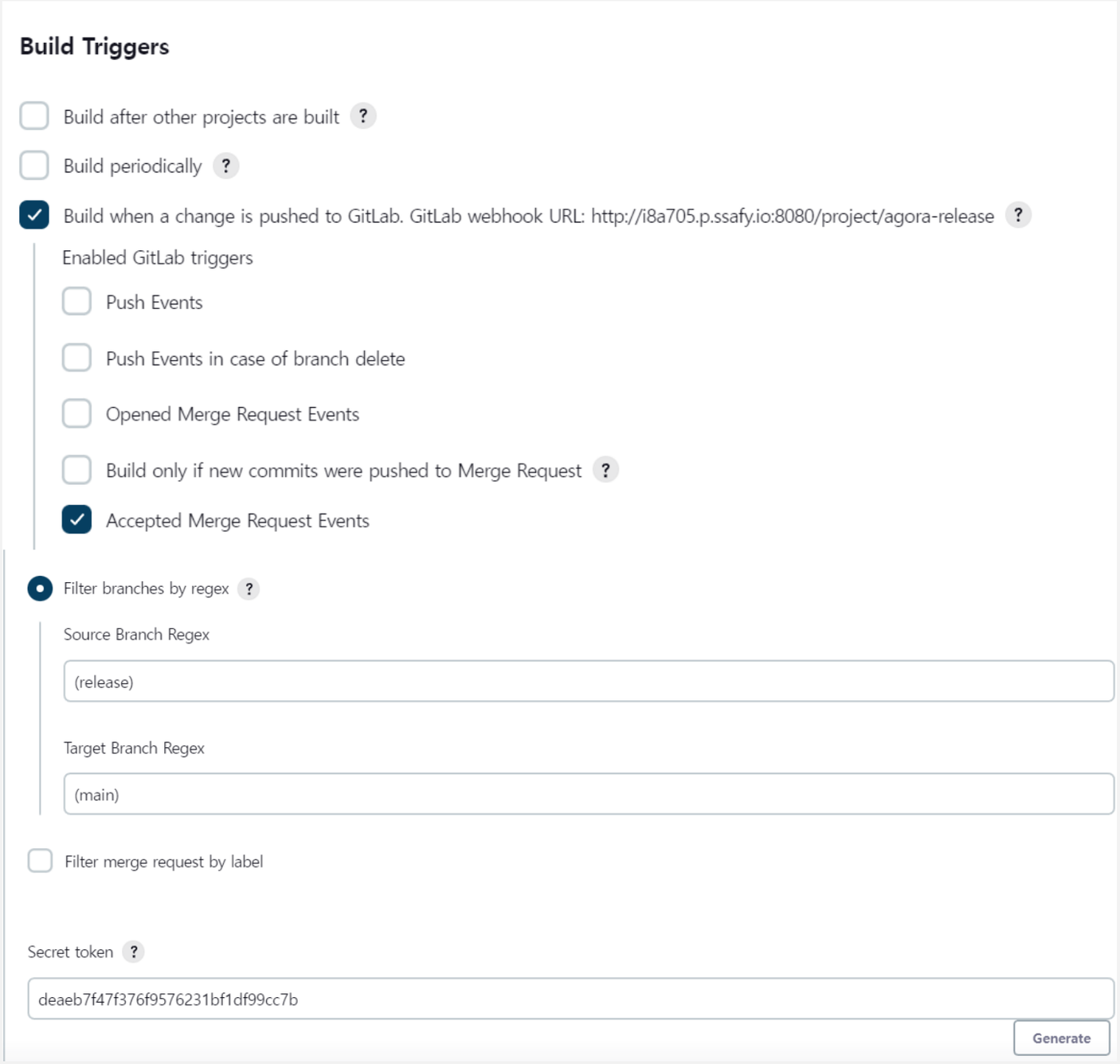
- 프로젝트 빌드 후 jar 파일 복사
- 환경변수 설정 후 ENTRYPOINT 에 추가

(예시 :"-Dspring.profiles.active=${USE_PROFILE}“)

Jenkins Build (CI)



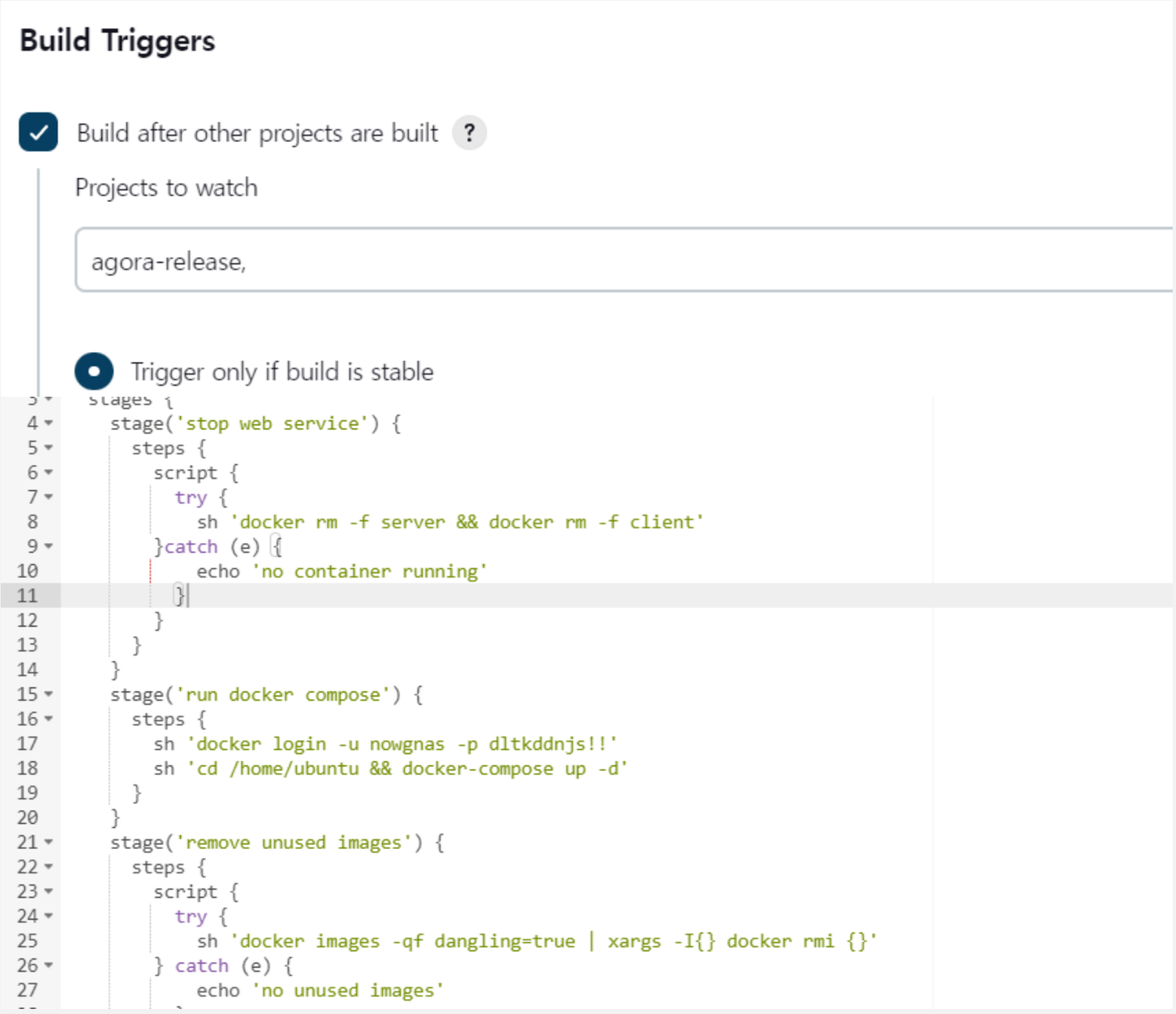
- Jenkins pipeline 선택하여 프로젝트 생성



- Merge가 되었을 때 빌드가 동작하도록 설정
- 정규식으로 source 브랜치와 target 브랜치 지정

```
pipeline {
  agent any
  stages {
    stage('repository clone') {
      steps {
        git branch: 'release', credentialsId: 'gitlab', url: 'https://lab.ssafy.com/s08-webmobile1-sub2/S08P12A705.git'
      }
    }
    stage('stop web service') {
      steps {
        script {
          try {
            sh 'docker rm -f backend && docker rm -f client'
          }catch (e) {
            echo 'no container running'
          }
        }
      }
    }
    stage('parallel build') {
      parallel {
        stage('client build') {
          steps {
            sh 'docker build -f client/Dockerfile -t nowgnas/agora:client .'
          }
        }
        stage('server build') {
          steps {
            sh 'docker build -f server/Dockerfile -t nowgnas/agora:backend .'
          }
        }
      }
    }
    stage('push build images') {
      steps {
        sh 'docker login -u nowgnas -p dltkddnjs!!'
        sh 'docker push nowgnas/agora:backend'
        sh 'docker push nowgnas/agora:client '
      }
    }
  }
```

- jenkins에서 release 브랜치를 clone 해서 client와 server를 빌드
- clien와 server 이미지 병렬 빌드
- 빌드된 이미지 docker hub에 push



- 앞선 빌드가 끝나면 AWS의 docker compose 파일로 서비스 실행
- docker compos로 docker hub의 이미지 pull 한 후 실행
- docker-compose.yml 위치는 AWS의 /home/ubuntu

AWS docker-compose.yml

```
version: '3'
services:
  client:
    image: n[redacted]agora:client
    container_name: client
    ports:
      - 3000:3000
    environment:
      REACT_APP_SERVER_BASE_URL: https://i8a705.p.ssafy.io/api
    networks:
      - AGORA
  backend:
    image: r[redacted]/agora:backend
    container_name: backend
    ports:
      - 8081:8081
    environment:
      USE_PROFILE: prod
      HOST_NAME: a[redacted]
      SCHEMA: b[redacted]
      USERNAME: r[redacted]
      PASSWORD: d[redacted]
      JWT_SECRET: l[redacted]
      REDISHOST: r[redacted]
      REDISPASS: r[redacted]
      MEETING_HOST: https://[redacted]
      MEETING_SECRET: ss[redacted]
    networks:
      - AGORA
networks:
  AGORA:
    external:
      name: AGORA
```

✓ 외부 서비스

1. Google Login

1. Google Login

- 사용자 인증 정보 선택, Oauth 클라이언트 ID 만들기 선택
- 프로젝트 이름 설정, redirect uri를 지정
- 현재 프로젝트의 redirect uri는 nginx에서 설정한 location에 따른 경로

2. Kakao Login

- redirect uri를 지정
- 동의 받을 필수 항목 지정
- client id 와 client secret 확인

*Naver 로그인은 페이지 인증 이슈로 설명 추가하지 않았습니다.