Porting Manual

Porting Manual

📋1. 클론 이후 빌드 및 배포할 수 있는 작업 문서

1️⃣ 사용한 JVM, 웹서버, WAS 제품 등의 종류와 설정값, 버전(IDE버전 포함)

배포 서버

- Ubuntu 20.04
- Nginx
- Nodejs 14
- Docker-compose 3.2

DB

- MySQL 8.0.27
- Redis 7.0.0

Front-end

- React 18.0.0
- React bootstrap 5.1.3

Back-end

- java : 11
- Spring Boot
  - Spring boot : “2.6.6”
  - azul 17

IDE

- IntelliJ IDEA 2021.3.1
- Visual Studio Code : 1.67.2
- MySQL Workbench : 8.0.27

2️⃣ DB 접속 정보 등 프로젝트(ERD)에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록

MySQL

```
datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://[domain]:3333/[schema name]?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: [username]
    password: [password]
```

Redis

```
cache:
    type: redis
  redis:
    host: "[domain]"
    port: 8180
    password: [password]
```

application.yml

```
spring:
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://[domain]:3333/[schema name]?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: [username]
    password: [password]

  jpa:
    hibernate:
      ddl-auto:
      jdbc:
        time_zone: Asia/Seoul
    properties:
      hibernate:
        show_sql: false
        format_sql: false

  cache:
    type: redis
  redis:
    host: "[Host IP]"
    port: 8180
    password: [password]

  jwt:
    header: Authorization
    secret: [scret]
    token-validity-in-seconds: 86400

  output:
    ansi:
      enabled: always

server:
  port: 8081
```

🌐2. 배포

1️⃣ Front-end 배포

Front-end 빌드 및 배포

1. ec2 setting

```
$ sudo apt-get update

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

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

2. docker 설치

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

3. front docker file

```
# node 14버전으로 이미지 생성
FROM node:14 as build-stage
# 작업 디렉토리
WORKDIR var/jenkins_home/workspace/fessaffron/front/bwadrigo

# package.json,파일을 컨테이너 작업공간에 복사
COPY package*.json ./
# 의존성 설치
RUN npm install
# 코드전체 컨테이너로 복사
COPY . .
# dist파일 생성
RUN npm run build

#nginx 베이스이미지 설치및 연결
FROM nginx:stable-alpine as production-stage
# 만들어진파일 디렉토리로 복사
COPY --from=build-stage /var/jenkins_home/workspace/fessaffron/front/bwadrigo/build /usr/share/nginx/html
EXPOSE 80

# nginx 백그라운드 실행
CMD ["nginx", "-g","daemon off;"]

#test
```

4. jenkins 설치 (8081 포트)

```
sudo docker run -d -u root --restart always --name jenkins \
-p 8001:8080 -p 50000:50000 \
-v $PWD/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/bin/docker:/usr/bin/docker \
jenkins/jenkins
```

5. jenkins 설정 (**Execute shell)**

```
docker image prune -a --force
 mkdir -p /var/jenkins_home/images_tar
 cd /var/jenkins_home/workspace/fessaffron/front/bwadrigo
 docker build -t fessaffron-client .
 docker save fessaffron-client > /var/jenkins_home/images_tar/fessaffron-client.tar

 ls /var/jenkins_home/images_tar
```

6. jenkisn ssh 쉘에서 접근 (**Execute shell script on remote host using ssh)**

- credential 에 ssh 인증정보 생성 후

```
ls /home/ubuntu/jenkins_home/images_tar

 sudo docker load < /home/ubuntu/jenkins_home/images_tar/fessaffron-client.tar

 if (sudo docker ps | grep "fessaffron-client"); then sudo docker stop fessaffron-client; fi

sudo docker run -it -d --rm -p 80:80 -p 443:443 -v /home/ubuntu/certbot/conf:/etc/letsencrypt/ -v /home/ubuntu/certbot/www:/var/www/cer
 echo "Run client"

 sudo docker ps
```

Nginx 설정과 ssl 인증서 발급 및 적용

1. Certbot 설치

- /home/ubuntu 에 certbot 디렉토리를 생성하고 conf와 www 디렉토리를 생성
- 디렉토리와 컨테이너 연동

```
cd
sudo mkdir certbot
cd certbot
sudo mkdir conf www logs

sudo docker pull certbot/certbot
sudo docker run -it --rm --name certbot -p 80:80\\
             -v "/home/ubuntu/certbot/conf:/etc/letsencrypt" \\
             -v "/home/ubuntu/certbot/log:/var/log/letsencrypt" \\
             -v "/home/ubuntu/certbot/www:/var/www/certbot" \\
             certbot/certbot certonly
```

2. nginx 설정

- sudo docker exec -it [fessaffron의 docker id] bin/sh 입력
- cd /etc/nginx/conf.d/default.conf 파일 수정

```
server {
   listen 80;
   server_name k6s1041.p.ssafy.io;
   location / {
     return 301 https://$host$request_uri;
   }
 }

 server {
   listen 443 ssl;
   server_name k6s1041.p.ssafy.io;
   access_log /var/log/nginx/access.log;
   error_log /var/log/nginx/error.log;

   ssl_certificate /etc/letsencrypt/live/k6s1041.p.ssafy.io/fullchain.pem;
   ssl_certificate_key /etc/letsencrypt/live/k6s1041.p.ssafy.io/privkey.pem;
   ssl_protocols TLSv1 TLSv1.1 TLSv1.2 SSLv3;
   ssl_ciphers ALL;

   location / {
     root /usr/share/nginx/html;
     index index.html index.html;
     proxy_redirect off;
     charset utf-8;
     try_files $uri $uri/ /index.html;

     proxy_http_version 1.1;
     proxy_set_header Upgrade $http_upgrade;
     proxy_set_header Connection "upgrade";
     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_set_header X-Nginx-Proxy true;
   }
    location /v1 {
        proxy_pass http://k6s1041.p.ssafy.io:8083;
    }
 }
```

3. 명령어

```
# 알파인 기반
nginx -s reload
```

2️⃣ Back-end 배포

Back-end 배포

- Docker 설치

```
$ sudo apt-get update
$ sudo apt-get -y upgrade
$ curl -fsSL https://get.docker.com/ | sudo sh

$ docker --version
```

→ 여기서 끝내면 K6S1041 처럼 도커 실행할 때 sudo docker 방식이 됨

- 현재 유저에게 docker를 사용할 수 있는 권한 부여

```
$ sudo usermod -aG docker $USER
$ sudo service docker restart

# 재로그인
$ sudo su
$ sudo su ubuntu

$ docker ps
```

Jenkins 설치

```
$ mkdir compose && cd compose #compose 관리 폴더
$ mkdir jenkins-compose && cd jenkins-compose # jenkins-compose 폴더 생성
$ mkdir jenkins-dockerfile && cd jenkins-dockerfile # dockerfile을 저장할 폴더 생성
$ vim Dockerfile # dockerfile 생성
```

```
FROM jenkins/jenkins:lts

USER root
RUN apt-get update &&\
    apt-get upgrade -y &&\
    apt-get install -y openssh-client
```

```
$ cd .. # jenkins-compose로 이동
$ vim docker-compose.yml # jenkins 관련 docker-compose.yml 파일 생성
```

```
version: "3"
services:
  jenkins:
    container_name: jenkins-compose
    build:
      context: jenkins-dockerfile
      dockerfile: Dockerfile
    user: root
    ports:
      - 8000:8080
      - 9090:50000
    volumes:
      - /home/ubuntu/compose/jenkins-compose/jenkins:/var/jenkins_home
      - /home/ubuntu/compose/jenkins-compose/.ssh:/root/.ssh
```

```
# 컨테이너 경로와 공유할 폴더 생성
$ mkdir jenkins
$ mkdir .ssh

# docker-compose up : 이미지를 빌드하고 컨테이너를 실행
# -d : 백그라운드 실행
$ docker-compose up --build -d


$ docker image ls
$ docker ps
$ docker logs jenkins-compose
# 젠킨스 비밀번호 확인
```

Spring 배포 준비

```
$ cd ..
$ mkdir ssaffron-compose && cd ssaffron-compose
$ mkdir ssaffron-dockerfile && cd ssaffron-dockerfile
$ vim Dockerfile
```

```
FROM openjdk:11-jdk

ENTRYPOINT java -jar /deploy/business-0.0.1-SNAPSHOT.jar

EXPOSE 8081
```

```
$ cd ..
$ vim docker-compose.yml
```

```
version: "3"
services:
  spring:
    container_name: ssaffron-compose
    build:
      context: ssaffron-dockerfile
      dockerfile: Dockerfile
    ports:
      - 8081:8081
    volumes:
      - /home/ubuntu/compose/jenkins-compose/jenkins/workspace/business/back/business/build/libs:/deploy
```

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

$ docker ps
```

자동 배포

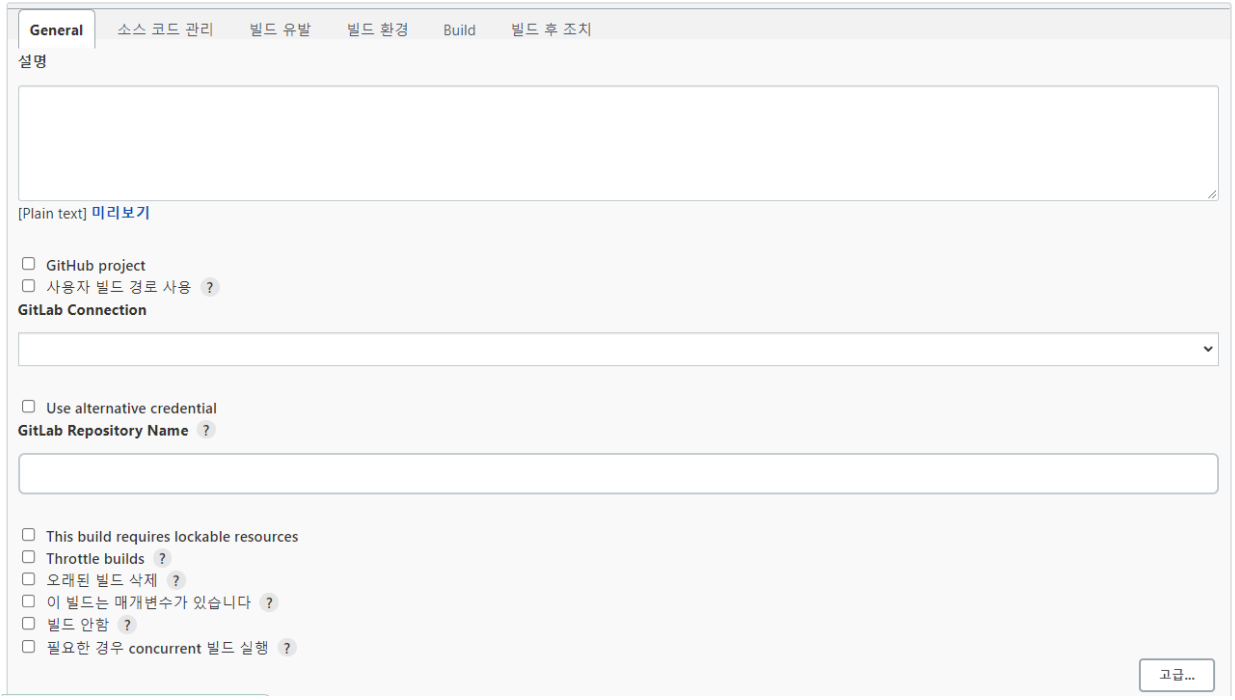
```
$ docker exec -it jenkins-compose bash

$ ssh-keygen -t rsa
$ cat /root/.ssh/id_rsa.pub
# ssh-rsa부터 뒤에 root@~ 까지 모두 복사
$ exit # Ctrl + D
$ vim ~/.ssh/authorized_keys
# 복사한 거 붙여넣기
```

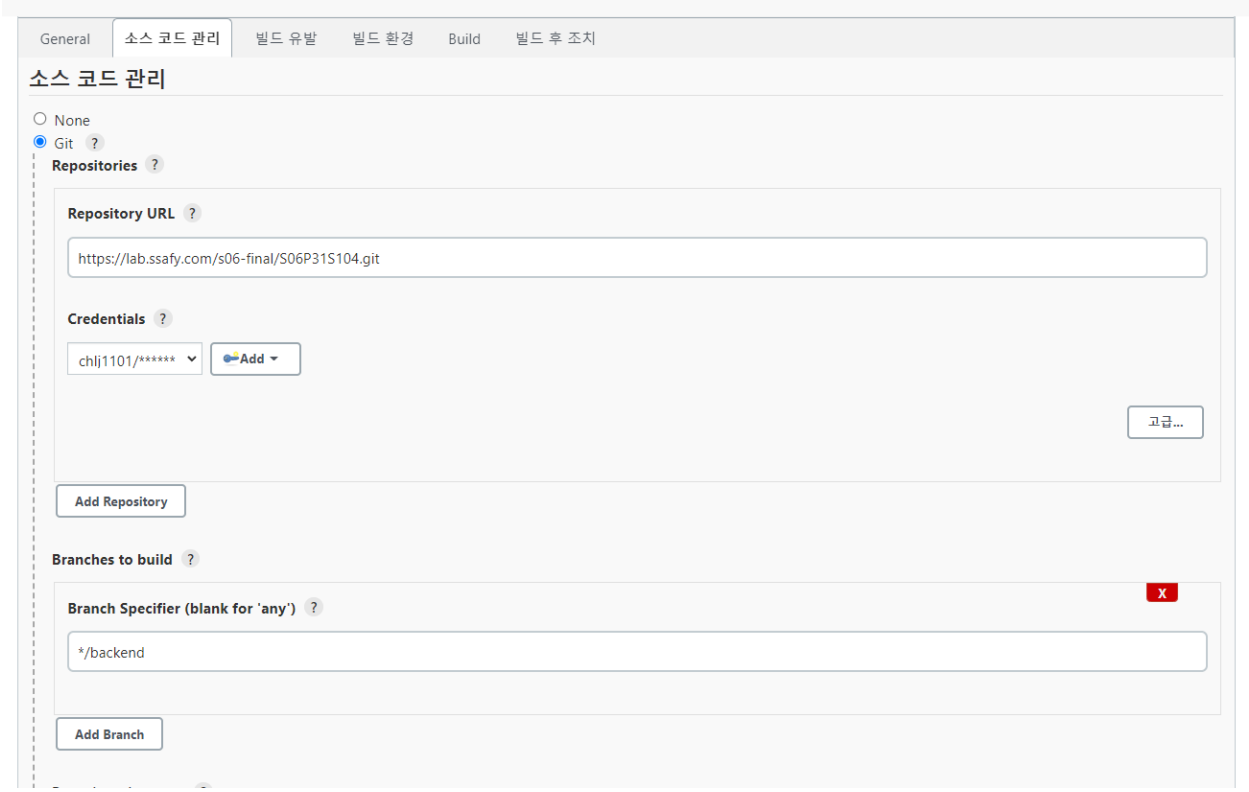
```
$ docker exec -it jenkins-compose bash
$ apt install iproute2
# 이 명령어가 안 되면
# apt update 혹은 apt upgrade
$ ssh ubuntu@$(/sbin/ip route | awk '/default/ { print $3 }')
# `$(/sbin/ip route | awk '/default/ { print $3 }')` 명령어는
# 도커 컨테이너 내부에서 ec2 로컬로 접속할 수 있는 주소를 출력한다.
$ exit # Ctrl + D
```

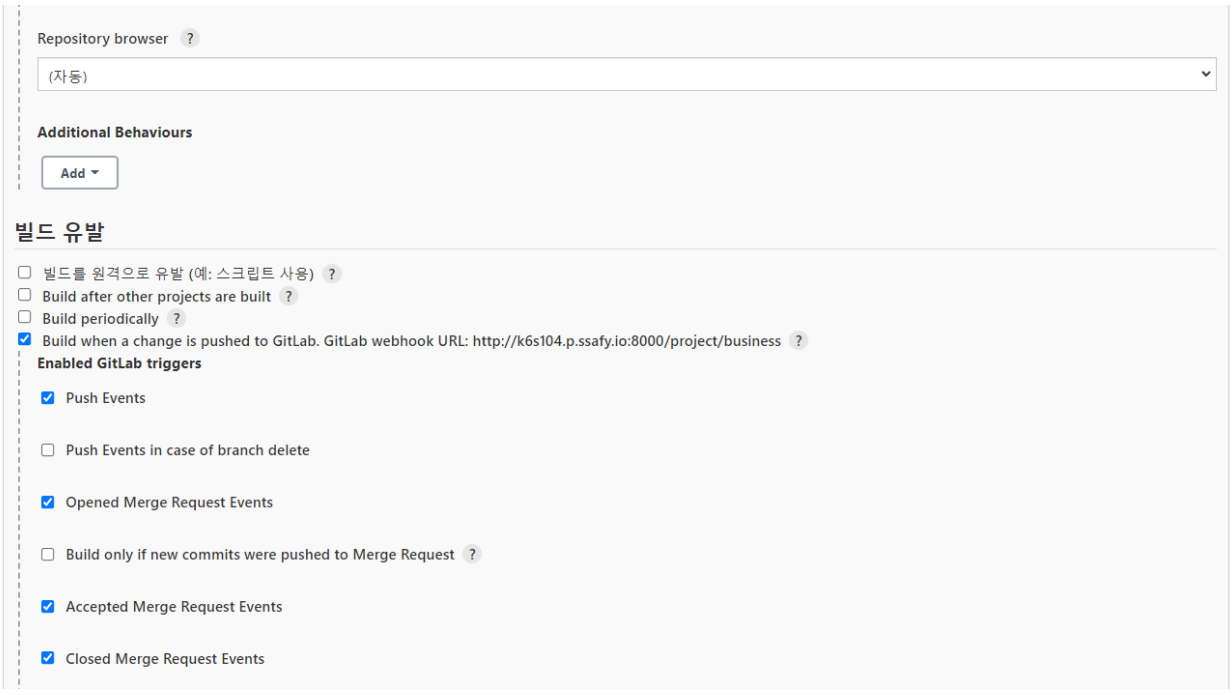
빌드 테스트

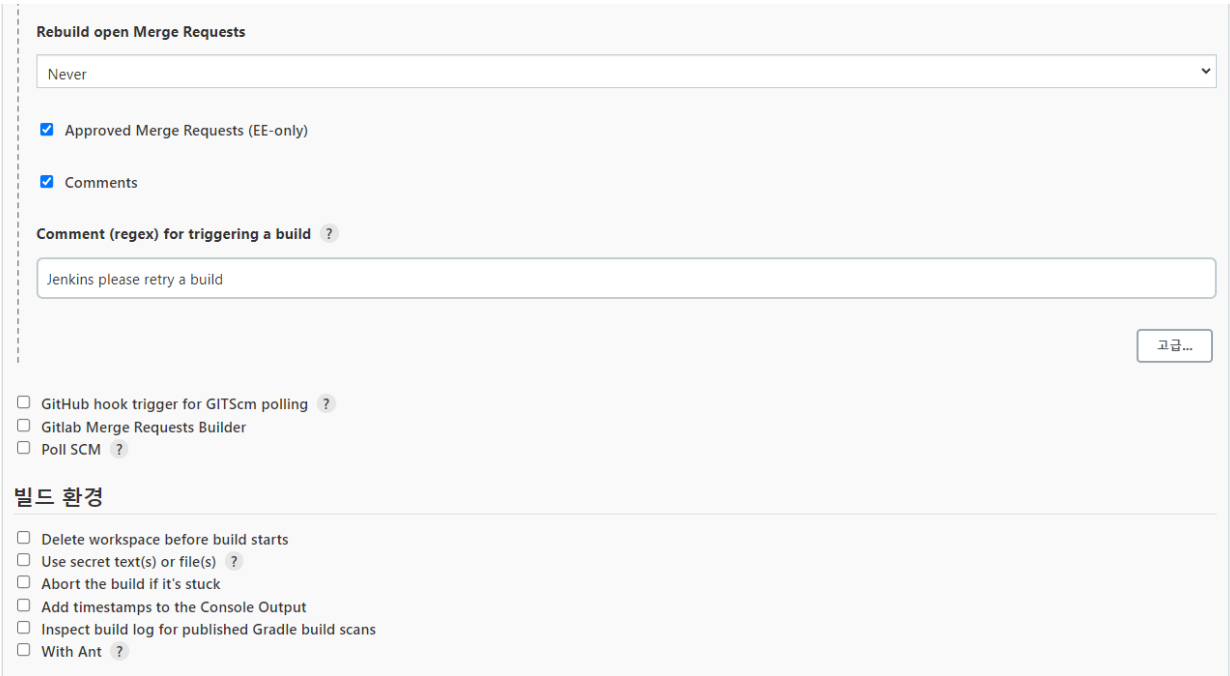
```
$ ssh -t -t ubuntu@$(/sbin/ip route | awk '/default/ { print $3 }') <<EOF
> cd /home/ubuntu/compose/ssaffron-compose
> docker-compose build --no-cache
> docker-compose up -d
> exit
> EOF

$ docker ps
```

젠킨스에서 CI/CD 설정



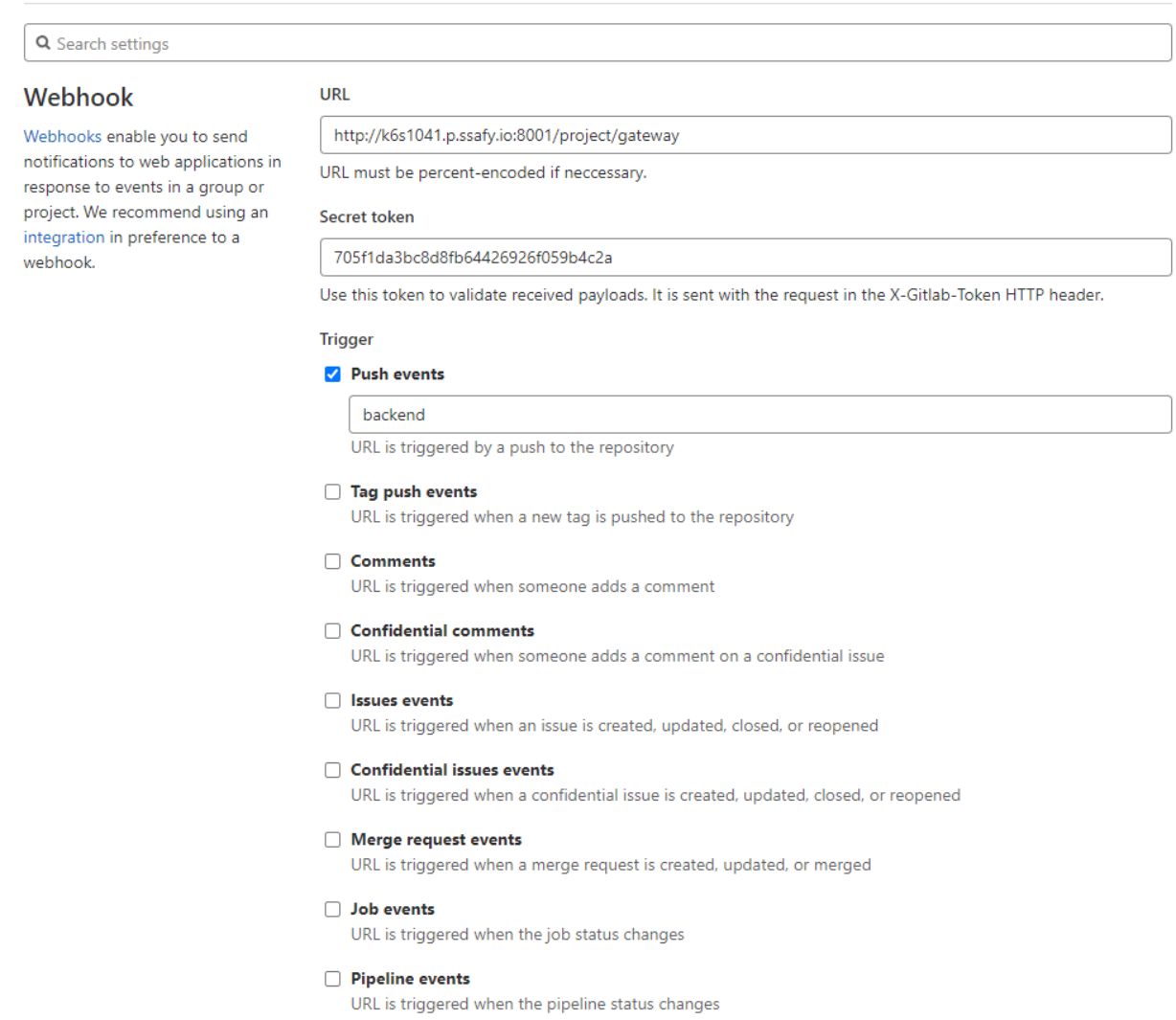

General
소스 코드 관리
빌드 유발
빌드 환경
Build
빌드 후 조치
소스 코드 관리
None
Git
Repositories
Repository URL
https://lab.ssafy.com/s06-final/S06P31S104.git
Credentials
chlj1101/******
Add
고급...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/backend
Add Branch

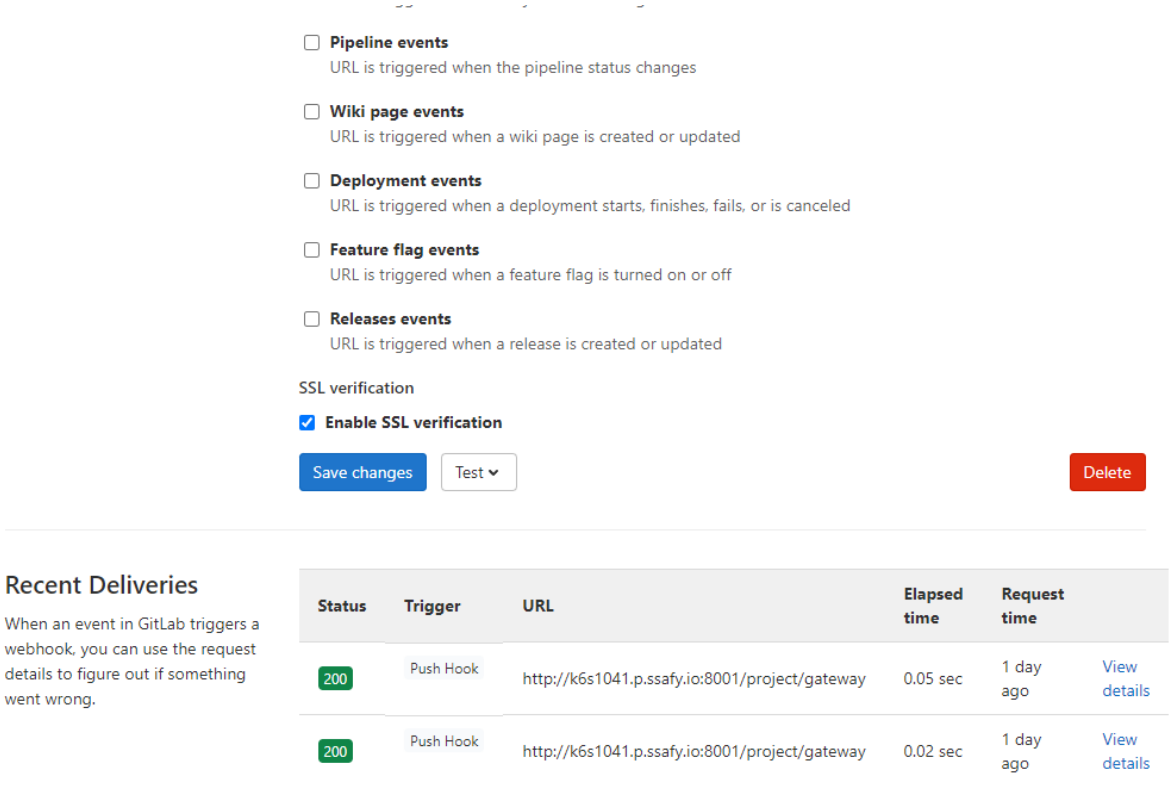

Repository browser
(자동)
Additional Behaviours
Add
빌드 유발
빌드를 원격으로 유발 (예: 스크립트 사용)
Build after other projects are built
Build periodically
Build when a change is pushed to GitLab. GitLab webhook URL: http://k6s104.p.ssafy.io:8000/project/business
Enabled GitLab triggers
Push Events
Push Events in case 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
고급...
GitHub hook trigger for GITScm polling
Gitlab Merge Requests Builder
Poll SCM
빌드 환경
Delete workspace before build starts
Use secret text(s) or file(s)
Abort the build if it's stuck
Add timestamps to the Console Output
Inspect build log for published Gradle build scans
With Ant


Build
Execute shell
Command
pwd
cd back/business
ls -al
chmod +x gradlew
./gradlew clean build
ssh -t -t ubuntu@$(/sbin/ip route | awk '/default/ { print $3 }') <<EOF
cd /home/ubuntu/compose/ssaffron-compose
docker-compose build --no-cache
docker-compose up -d
exit
EOF
See the list of available environment variables
고급...
Add build step
빌드 후 조치
빌드 후 조치 추가

깃 랩 설정


Search settings
Webhook
Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.
URL
http://k6s1041.p.ssafy.io:8001/project/gateway
URL must be percent-encoded if neccessary.
Secret token
705f1da3bc8d8fb64426926f059b4c2a
Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.
Trigger
Push events
backend
URL is triggered by a push to the repository
Tag push events
URL is triggered when a new tag is pushed to the repository
Comments
URL is triggered when someone adds a comment
Confidential comments
URL is triggered when someone adds a comment on a confidential issue
Issues events
URL is triggered when an issue is created, updated, closed, or reopened
Confidential issues events
URL is triggered when a confidential issue is created, updated, closed, or reopened
Merge request events
URL is triggered when a merge request is created, updated, or merged
Job events
URL is triggered when the job status changes
Pipeline events
URL is triggered when the pipeline status changes



Recent Deliveries

When an event in GitLab triggers a webhook, you can use the request details to figure out if something went wrong.

Status	Trigger	URL	Elapsed time	Request time	
200	Push Hook	http://k6s1041.p.ssafy.io:8001/project/gateway	0.05 sec	1 day ago	View details
200	Push Hook	http://k6s1041.p.ssafy.io:8001/project/gateway	0.02 sec	1 day ago	View details