

늘봄 (Neulbom)

삼성SW청년아카데미 서울캠퍼스 6기

Gitlab 소스 클론 이후 빌드 및 배포 과정 정리

A104팀 늘봄

손형선, 유민상, 정현정, 허은아, 황수진

목 차

1. 사용한 기술 스택
2. 빌드 과정
3. 배포 과정
4. AI 서버

1. 사용한 기술 스택

1) 이슈 관리 : Jira

2) 형상 관리 : Gitlab

3) 빌드/배포 관리 : Jenkins

4) 커뮤니케이션 : Mattermost, Notion, Discord

5) 개발 환경

A. 운영체제 : Windows10

B. IDE

가) STS *3.9.14 RELEASE*

나) Visual Studio Code *1.64.2*

다) 안드로이드 스튜디오 *2021.2.1.14*

라) UI/UX : Figma

C. 데이터베이스 : MySQL Workbench *8.0.27*

D. 서버 : AWS EC2

가) Ubuntu *20.04 LTS*

나) Docker *20.10.14*

다) Nginx *1.18.0*

라) Flask *2.1.2*

E. 저장소 : AWS S3, Firebase

6) 세부 사항

A. Backend

가) Java *Open-JDK zulu 1.8*

나) Spring Boot *3.9.14.RELEASE*

다) Gradle *7.4.1*

라) JPA *2.5.6*

마) JDBC *2.5.6*

마) lombok *1.18.22*

바) Swagger2 *2.9.2*

사) JWT *0.9.1*

아) OAuth2, Spring Security *2.5.6*

자) Sockjs *1.1.2*

차) stomp *2.3.3*

카) Python *3.8.10*

B. Frontend

- 가) ReactNative *0.64.3*
- 나) redux *4.2.0*
- 다) axios *0.27.2*
- 라) react-dom *17.0.1*
- 마) react-redux *8.0.1*
- 바) socketjs-client *1.6.0*
- 사) chocolaty *1.1.0*
- 아) android *11.0*
- 자) styled-components *5.3.5*

2. 빌드 과정 – Jenkins 세팅

Gitlab과 연동된 Jenkins의 관리페이지로 접속합니다. (<https://k6a104.p.ssafy.io:8080>)

1) jenkins 플러그인 설치

jenkins 관리 → 플러그인 관리에서 필요한 플러그인 설치 (설치가능 탭에서 Search 가능)

- Blue Ocean을 검색 후 검색된 모든 플러그인
- gitlab을 검색하고 검색된 모든 플러그인
- Google Play Android Publisher Plugin 플러그인

2) jenkins에 gitlab 연결

jenkins 관리 → 시스템 설정

GitLab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

neulbom

A name for the connection

Gitlab host URL

https://lab.ssafy.com/

The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

Credentials

GitLab API token Add

API Token for accessing GitLab

고급...

Test Connection

삭제

추가

- 시스템 설정 후 내리면 GitLab 부분
- Connection name → 원하는 아무 이름이나 작성
- Gitlab host URL에 gitlab 주소 작성
→ https://xxx.xxxxxx.com 까지만 기입
- Credentials → Add → jenkins를 눌러 새로 생성

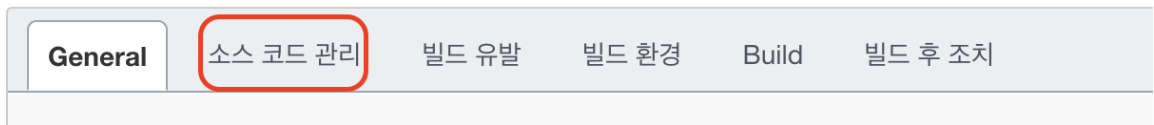
- 위 화면과 같은 화면이 나오면 Kind → GitLab API token으로 변경
- API tokens는 gitlab에서 발급받은 personal token 입력
 - personal token 발급 방법
 - gitlab으로 가서 User Settings → Access Tokens

- 원하는 token name과 만료 기간을 정해주고 api에 체크한 후
Create personal access token을 누름 → 생성 완료
→ 화면 상단에 token 발급된 거 확인 가능
- API tokens에 발급한 token 값 입력해주고
- ID → gitlab에서 사용하는 아이디 기입 (@뒤를 제외한 아이디만!)
- Add하면 추가 완료
- Credentials에 방금 만든 것으로 설정해주면 완료 → Apply 및 저장도 해주기

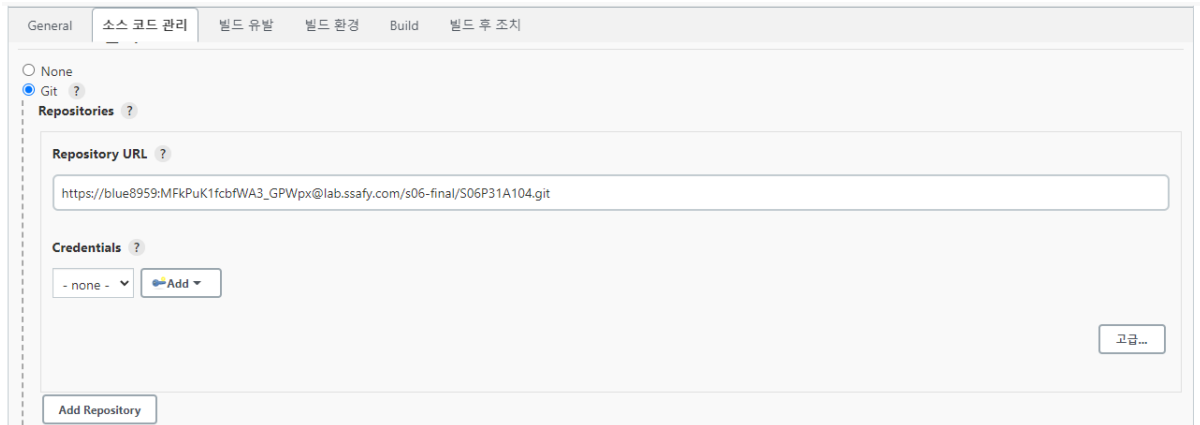
3) jenkins에 프로젝트 연결 후 기본 설정하기

- dashboard → 새로운 Item
- Enter an item name에 프로젝트 이름 작성 (아무거나 상관없음)
- Freestyle project 선택 후 완료

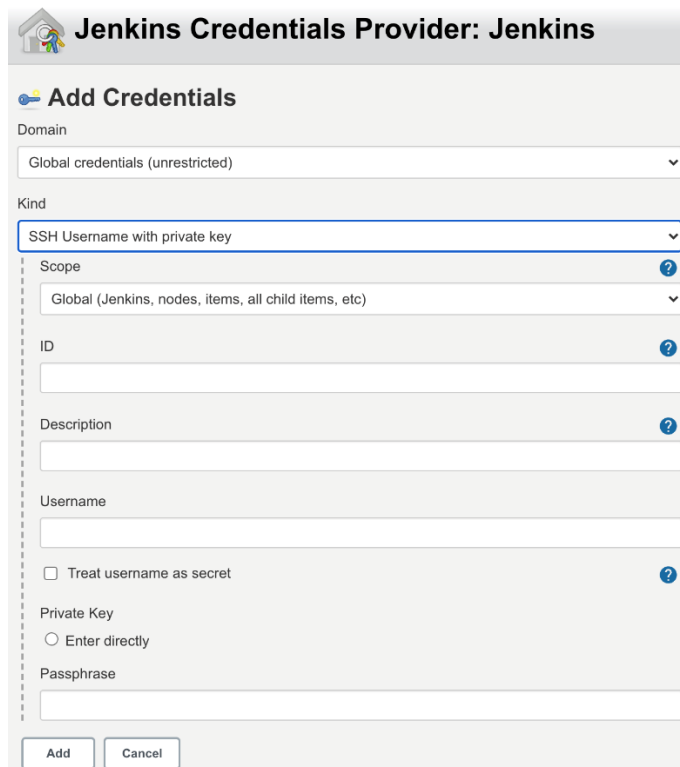
4) 프로젝트 git 연결하기



- 프로젝트로 들어가서 구성 → 소스코드 관리 탭에서 Git을 선택



- Repository URL에 git project clone 주소 작성
 - Failed to connect to repository ... ERROR 나는 경우
 - https://gitlabID:PersonalAccessToken@Repository주소로 작성
 - PersonalAccessToken 아까 위에서 발급 받은 것
 - Credentials → Add → Jenkins



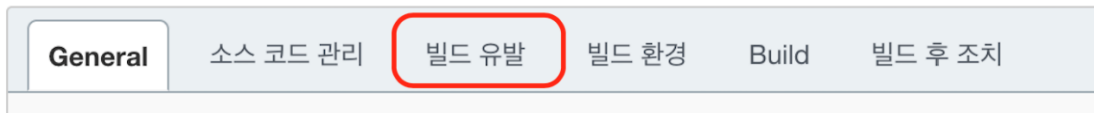
- kind → SSH Username with private key
 - ID → gitlab ID (@뒤 제외)
 - Username → gitlab에 로그인하는 email 형식
 - Private key → Enter directly → Add
 - 입력하는 곳에 ssh key를 입력해야함
 - gitlab으로 가서 User Settings → SSH keys에서 key를 입력해야 함
 - 이때 입력하는 키는 배포할 서버에서 생성 가능
 - 서버 접속 후
 - `ssh-keygen -t rsa -C "gitlabID(이메일형식)"`
 - `cat home/ubuntu/.ssh/id_rsa.pub` 열고 출력된 key 값 전체 복사
 - gitlab으로 돌아와서 ssh key 입력해 주고 타이틀, 유효기간 설정해주면
키 생성 완료 → 이 key를 이제 jenkins에 입력
- Branches to build에 배포될 브랜치 설정

Branches to build ?

Branch Specifier (blank for 'any') ?

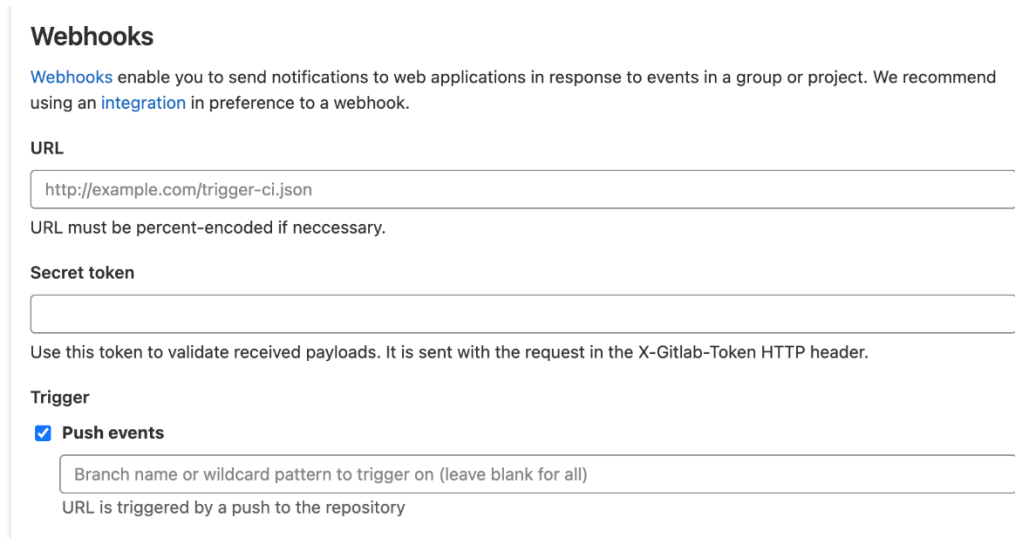
3. 빌드 과정

1) 프로젝트 빌드 기본설정



General 소스 코드 관리 **빌드 유발** 빌드 환경 Build 빌드 후 조치

- 구성 → 빌드 유발
- Build when a change is pushed to GitLab~~~ 체크 → 고급 → 맨 아래 generate 클릭
- generate까지 생성되면 key가 하나 생성되는데 복사해서 잘 가지고 있기!
- jenkins 설정 apply → 저장
- gitlab으로 가서 배포하고자 하는 프로젝트에서 Settings → Webhooks



Webhooks

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

URL must be percent-encoded if neccessary.

Secret token

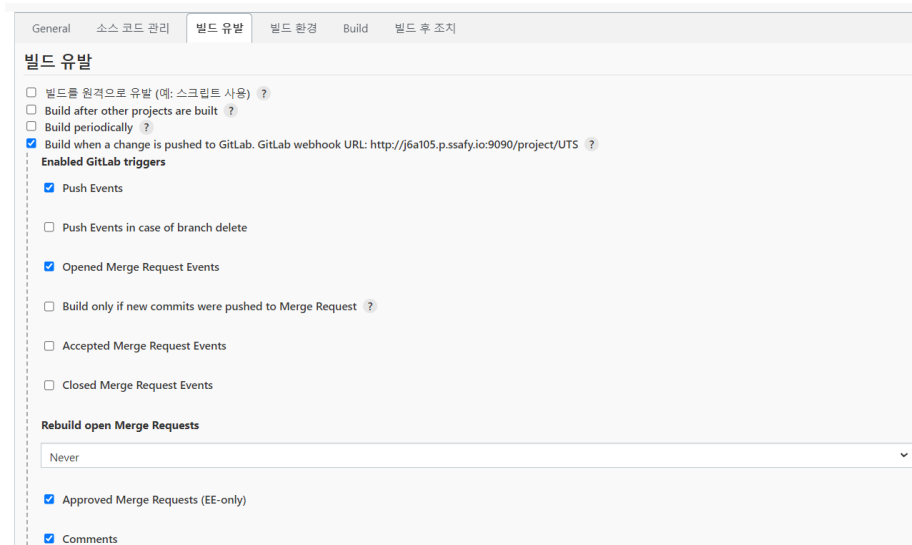
Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

URL is triggered by a push to the repository

- URL에 jenkins 프로젝트 url을 입력 (빌드유발 탭에서 확인 가능)



General 소스 코드 관리 **빌드 유발** 빌드 환경 Build 빌드 후 조치

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: ?

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

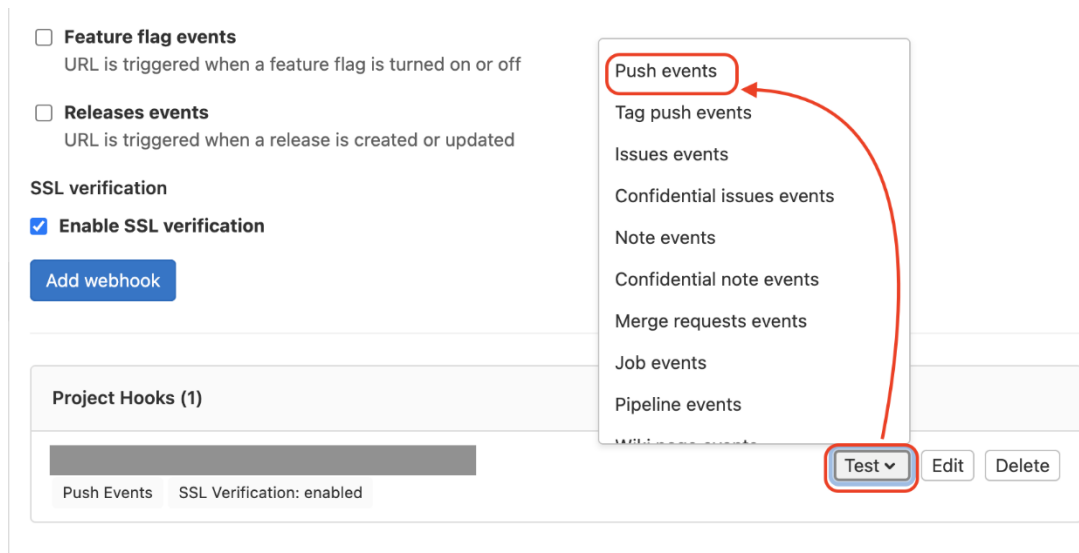
Rebuild open Merge Requests

☒ Approved Merge Requests (EE-only)

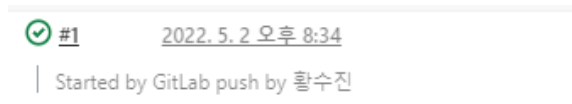
☒ Comments

- secret token에다가 아까 복사해둔 key를 입력 후 push events에는 푸시가 일어날 브랜치 입력
- 이제 아래의 Add webhook을 클릭
- hook이 생성되면 테스트에서 push events 설정하고 test

- 상단에 Hook executed successfully : HTTP 200 뜨면 성공



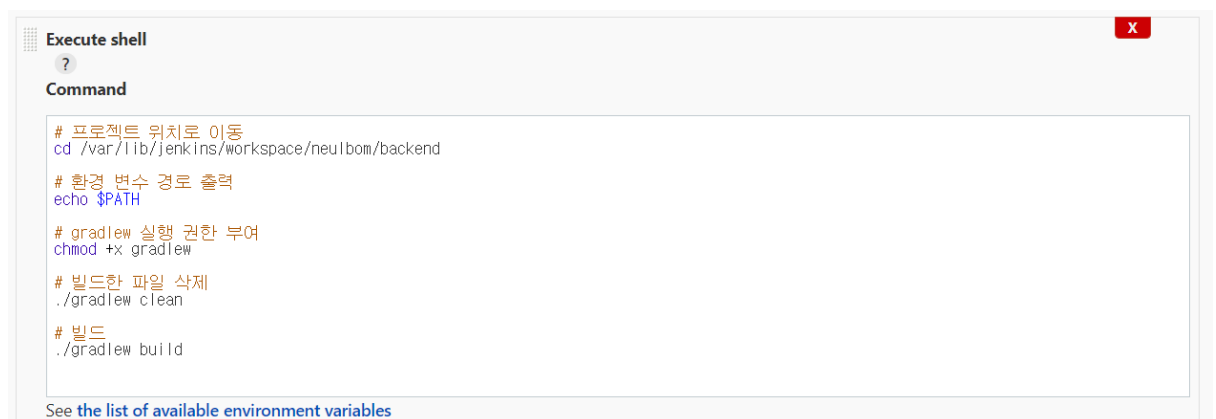
- jenkins로 가면 test build가 진행중이고 성공 여부를 확인 할 수 있음
(이때 실패하면 위에 설정 다 제대로 따라했는지 확인해보기)



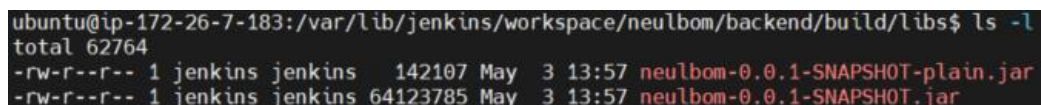
2) 생성한 프로젝트 빌드와 배포 설정



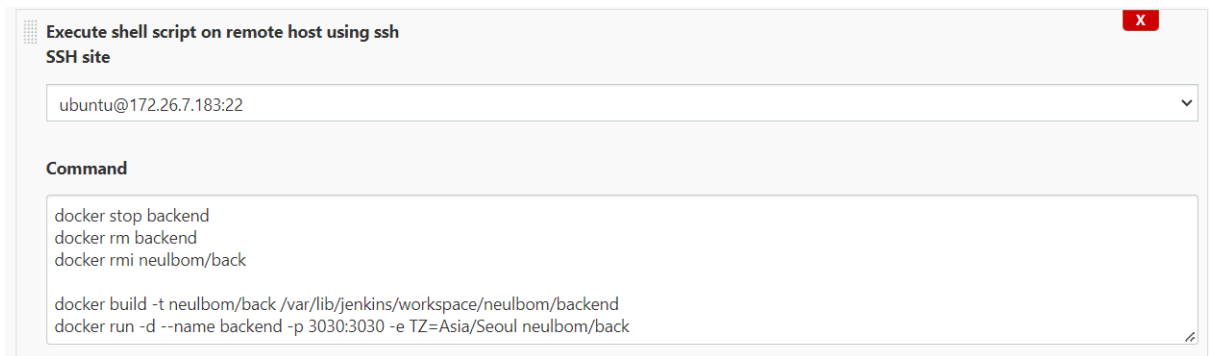
- 구성 → Build
- Execute shell에 명령어 입력



- build를 진행하면



jenkins - 프로젝트에 jar 파일 생성



- 실행중인 백엔드 도커 컨테이너를 멈춘 뒤 제거, 해당 도커 이미지도 제거
- 백엔드 프로젝트 위치에서 도커 이미지를 빌드 후 실행

* backend Dockerfile

```

FROM openjdk:8-jdk-alpine

# SpringBoot 프로젝트 경로/build/~/ .jar
ARG JAR_FILE=build/libs/neulbom-0.0.1-SNAPSHOT.jar

# app.jar로 복사
COPY ${JAR_FILE} app.jar

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

```

- 이제 저장하고 Build Now를 클릭하면 build가 생성되고 자동으로 배포 진행됨

3) Nginx 설정

- /etc/nginx/nginx.conf 파일 수정

```

server {
    server_name k6a104.p.ssafy.io;

    listen 443 ssl default_server ; # managed by Certbot
    listen [::]:443 ssl default_server;

    ssl_certificate /etc/letsencrypt/live/k6a104.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/k6a104.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot


    location /api/ {
        proxy_pass http://k6a104.p.ssafy.io:3030;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $http_host;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_http_version 1.1;
        proxy_ssl_server_name on;
    }
}

```

- https 설정 위해 변경

```
server {
    listen 80;
    listen [::]:80;

    server_name k6a104.p.ssafy.io;

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

- 먼저 80포트로 들어올 경우(http로 접속할 경우) https로 반환
- 그 후 443포트로 들어올 경우 (https로 접속할 경우) **/etc/letsencrypt**에 있는 SSL 키값들을 통해 인증을 진행
- 만약 도메인주소/api 경로로 접근할 경우 SpringBoot 프로젝트가 실행중인 3030포트로 연결

4. AI 서버

1) 환경 세팅

EC2에 AI 모델을 다운 받은 후 프로젝트에 맞게 코드 수정

2) Docker 설정

```
FROM python:3.8-slim

COPY . /app

RUN pip install virtualenv
CMD ["virtualenv", "venv"]
CMD ["cd", "venv/Scripts"]
CMD ["activate"]

RUN pip3 install flask
RUN pip3 install flask_restx
RUN pip install -r app/requirements.txt

WORKDIR /app

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

3) Docker 빌드 후 실행

```
docker build -t neulbom/flask test
```

```
docker run -d --name flask -p 5000:5000 -e TZ=Asia/Seoul neulbom/flask
```