🚥

포팅 매뉴얼

⚙️ 프로젝트 설정

1. 프로젝트 기술 스택

FRONT END

react	18.2.0
node.js	18.15.0 LTS
vite	4.1.0
typescript	4.9.3
three.js	0.150.1
react-three-fiber	8.12.0
blender	3.4.1
react query	3.39.3
react-router-dom	6.8.2
tailwindcss	3.2.7
recoil	0.7.7

BACK END

java	11.0.18
springboot	2.7.9
gradle	Openjdk 11.0.18+10
swagger	org.springdoc:springdoc-openapi-starter-webmvc-ui:2.0.0
MySQL	8.0.30
Redis	
Python	3.10.9
jUnit	
Flask	
Django	
Fast API	

2. EC2 설정 시나리오

1) 사용 포트

구분	포트 번호
Front-end	5173
Back-end	8080
Sub-Back-end	9090
MySQL	3306
Redis	6379
Django	8000

2) 방화벽 설정

1. 현재 방화벽 설정 확인

```
$ sudo ufw status
```

2. 방화벽 설정

```
$ sudo ufw allow ssh
$ sudo ufw enable
```

3) Docker 설치

Ubuntu에 도커 설치

```
$ sudo apt-get update

# 필수 패키지 설치
$ sudo apt-get install \
       apt-transport-https \
       ca-certificates \
       curl \
       gnupg \
       lsb-release

# GPG Key 인증
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# docker repository 등록
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

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

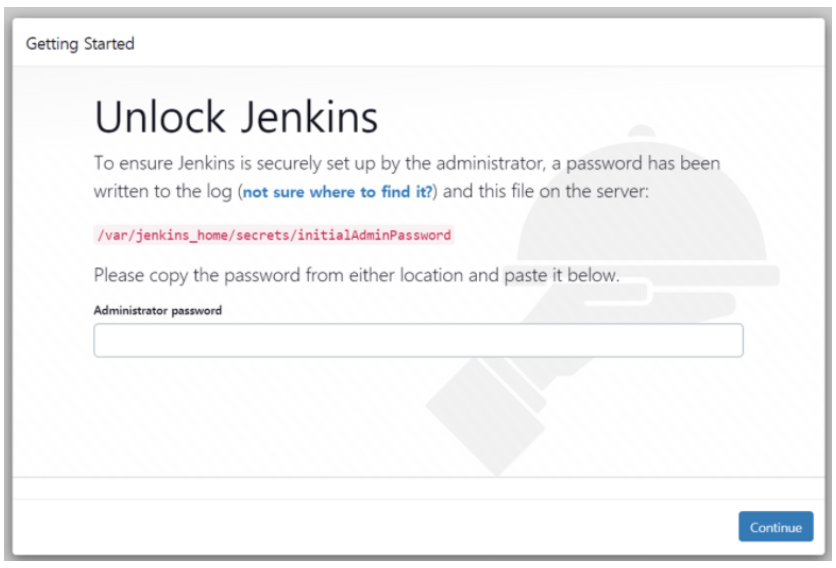
```
# 도커 확인
$ sudo service docker status
```

4) Jenkins 설치 및 설정

1. jenkins 이미지 설치 및 실행

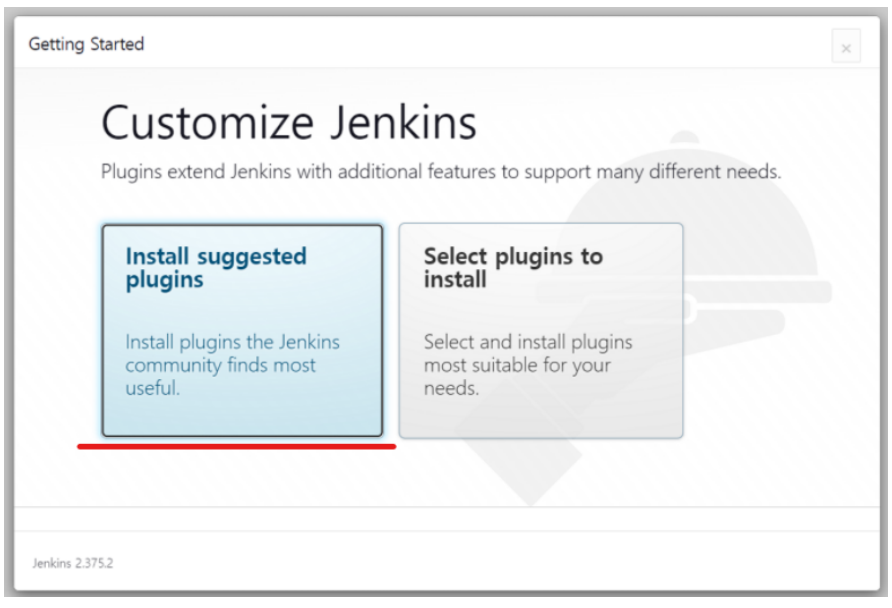
```
# 설치 및 실행
$ sudo docker run -u 0 -d -p 9090:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/jenkins:lts-jdk11

# 재시작
$ sudo docker restart jenkins

# 이미지 설치 확인
$ sudo docker images
```

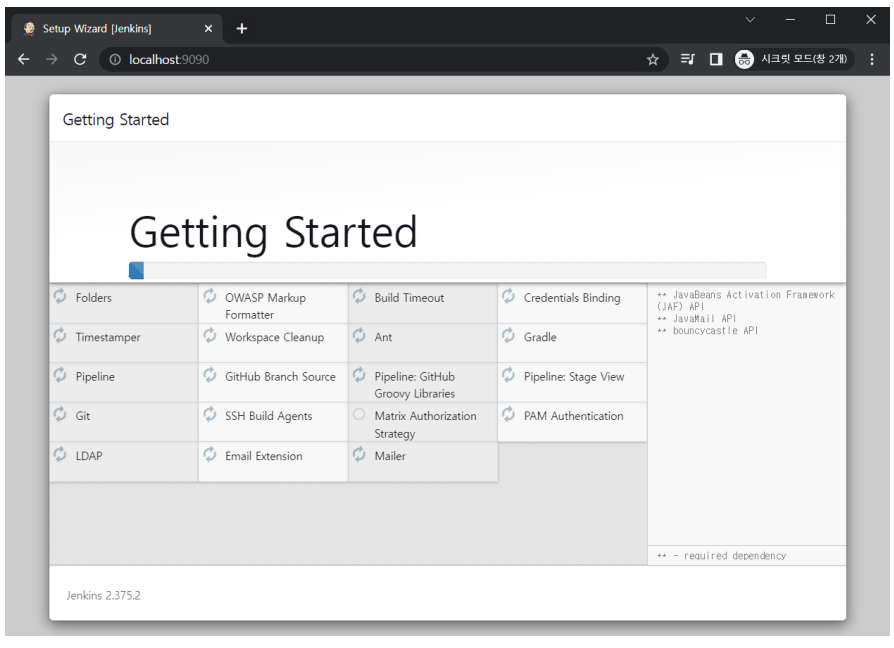
2. jenkins 접속 (서버 주소)




```
# 비밀번호 확인 방법
$ sudo docker logs jenkin
```

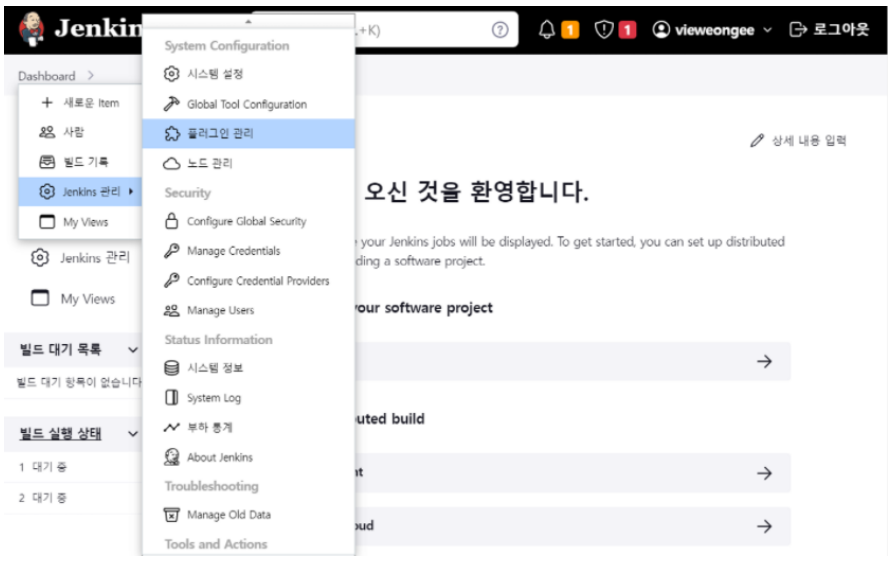
3. 기본 설치

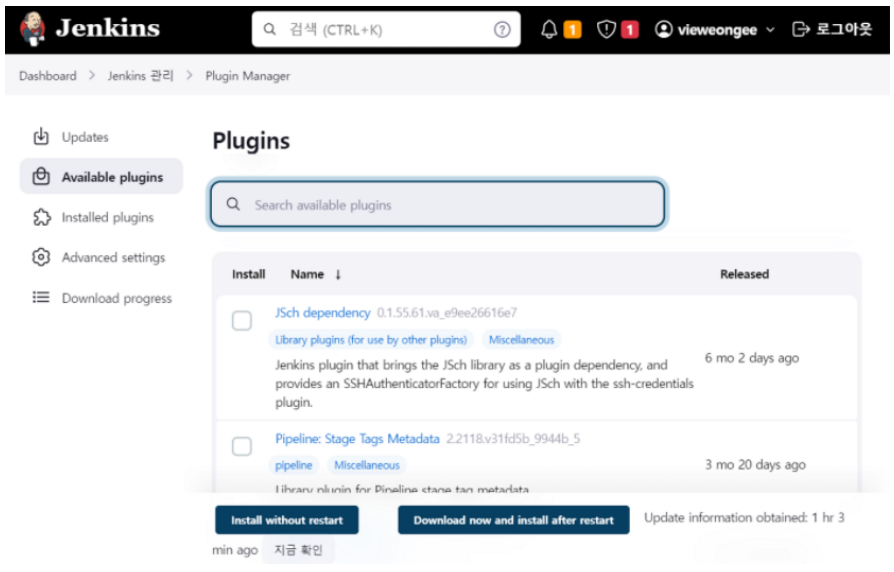




4. jenkins 플러그인 설치

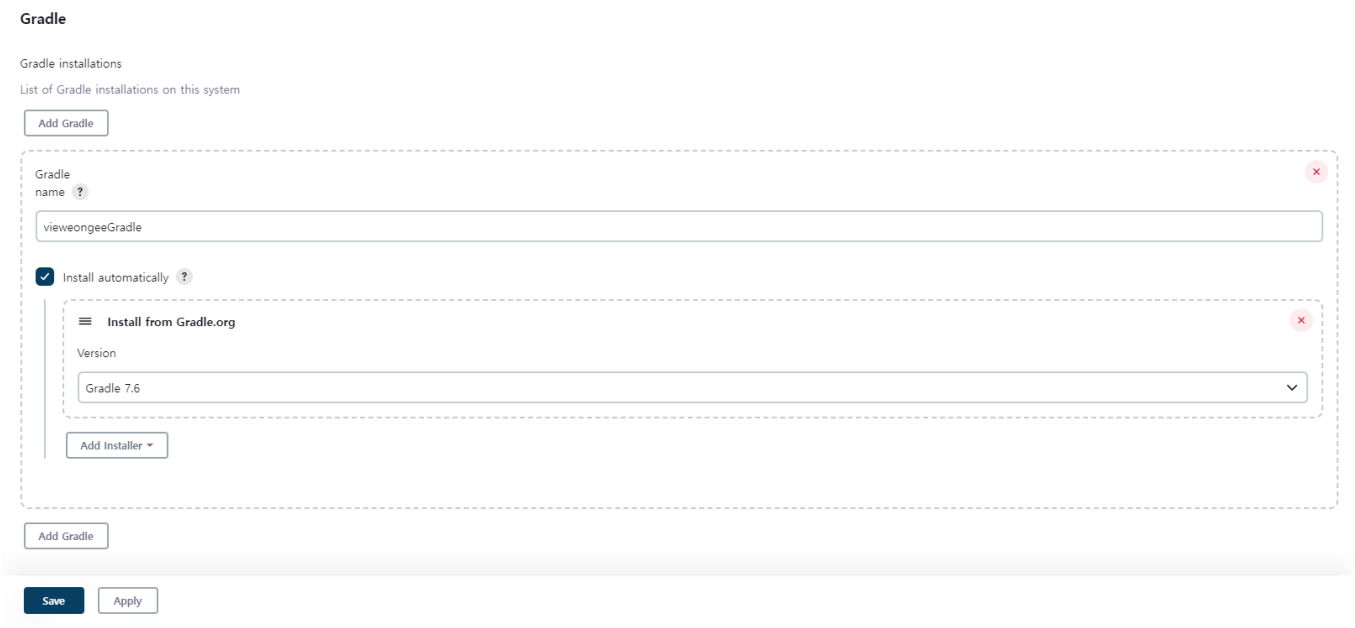
   Dashboard → Jenkins 관리 → 플러그인 관리 → Available plugins





- **Gitlab 관련 항목 설치**
  - Gitlab, Generic Webhook Trigger, Gitlab API, Gitlab Authentication 설치
- **Docker 관련 항목 설치**
  - Docker, Docker Commons, Docker Pipeline, Docker API 설치
- **백엔드에서 Gradle을 사용하였다면 Gradle Plugin도 설치**

5. Gradle을 사용하는 경우
   - Jenkins 관리 → Global Tool Configuration → 다음과 같이 사용한 Gradle 버전을 맞추고 추가하기



6. 젠킨스 컨테이너 안에 도커 설치

```
# 젠킨스 컨테이너 실행
$ docker exec -it jenkins bash

# 도커 설치
$ apt-get update

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

$ apt-get install docker.io -y

# 도커 확인
$ sudo service docker status
```

5) Nginx 설치

```
# 설치
$ sudo apt-get install nginx
```

```
# 설치 확인 및 버전 확인
$ nginx -v

# Nginx 설정은 SSL 인증서 발급 후에 하기
```

6) SSL 인증서 발급

```
$ sudo apt-get install letsencrypt

# 만약 nginx를 사용중이라면 중지
$ sudo systemctl stop nginx

# 인증서 발급
sudo letsencrypt certonly --standalone -d j8a705.p.ssafy.io

# 이메일 쓰고 Agree
# 뉴스레터 no
```

7) Nginx 설정

```
$ sudo vi /etc/nginx/conf.d/default.conf
$ sudo vi /etc/nginx/sites-available/default
```

```
server {
    location /{
        proxy_connect_timeout      90;
        proxy_send_timeout         90;
        proxy_read_timeout         90;
        proxy_pass http://localhost:5173;
    }

    location /api {
        proxy_connect_timeout      90;
        proxy_send_timeout         90;
        proxy_read_timeout         90;
        proxy_pass http://localhost:8080/api;
    }

    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/j8a705.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8a705.p.ssafy.io/privkey.pem;
}

server {
    if ($host = j8a705.p.ssafy.io) {
        return 301 https://$host$request_uri;
    }

        listen 80;
        server_name j8a705.p.ssafy.io;
    return 404;
}
```

```
# nginx 실행
$ sudo systemctl start nginx

# 실행 확인
$ sudo systemctl status nginx
```

3. 배포

1) Dockerfile 및 Nginx 설정

(1) Backend - Dockerfile

```
# openjdk11로 실행
FROM openjdk:11-jdk

# 해당 경로의 모든 jar파일을 변수로 담기
ARG JAR_FILE=build/libs/*.jar

# 빌드된 jar파일을 api.jar파일 이라는 이름으로 생성
COPY ${JAR_FILE} app.jar

# 컨테이너가 리스닝할 포트
EXPOSE 8080

# 환경 변수 설정
ENV TZ=Asia/Seoul

# 컨테이너를 실행할 때 실행할 커맨드
ENTRYPOINT ["java","-jar","app.jar"]
```

(2) Frontend - Dockerfile

```
# node.js로 빌드 ( base로 사용할 Image name )
FROM node:18 as build-stage

# 경로 설정
WORKDIR /app

# ADD <복사할 파일 경로> <이미지에서 파일이 위치할 경로>
ADD . .

# 의존성 설치
RUN npm install

# 빌드 -> dist 폴더 생성됨
RUN npm run build

# nginx로 실행
FROM nginx:stable-alpine as production-stage

# 컨테이너가 리스닝할 포트
EXPOSE 5173

# nginx.conf를 default.conf로 복사
COPY  ./nginx/nginx.conf /etc/nginx/conf.d/default.conf

# /app/dist를 /usr/share/nginx/html로 복사
COPY --from=build-stage /app/dist /usr/share/nginx/html

# 컨테이너를 실행할 때 실행할 커맨드
# CMD : 해당 이미지로 컨테이너 실행 시, 실행 명령어를 설정하지 않았을 때만 이 명령어 실행
CMD ["nginx", "-g", "daemon off;"]
```

(3) Frontend - nginx.conf

```
server {
    # 포트 번호
    listen 5173;

    # 경로 설정
    location / {
        # index 파일이 있는 경로
        root    /usr/share/nginx/html;

        # index 파일로 지정할 파일 설정
        index   index.html index.htm;
        # 요청한 주소의 uri를 무시하고 index.html 파일을 제공
        try_files $uri $uri/ /index.html;
    }
}
```

4. Jenkins & GitLab 연동

(1) jenkins에서 프로젝트 생성

Dashboard → 새로운 Item → 프로젝트 이름 입력 → Freestyle project

(2) 소스 코드 관리

소스 코드 관리 > Git을 선택하고 git clone 주소 입력

→ Credentials 아래의 Add 버튼을 클릭해서 깃 아이디와 비밀번호 저장

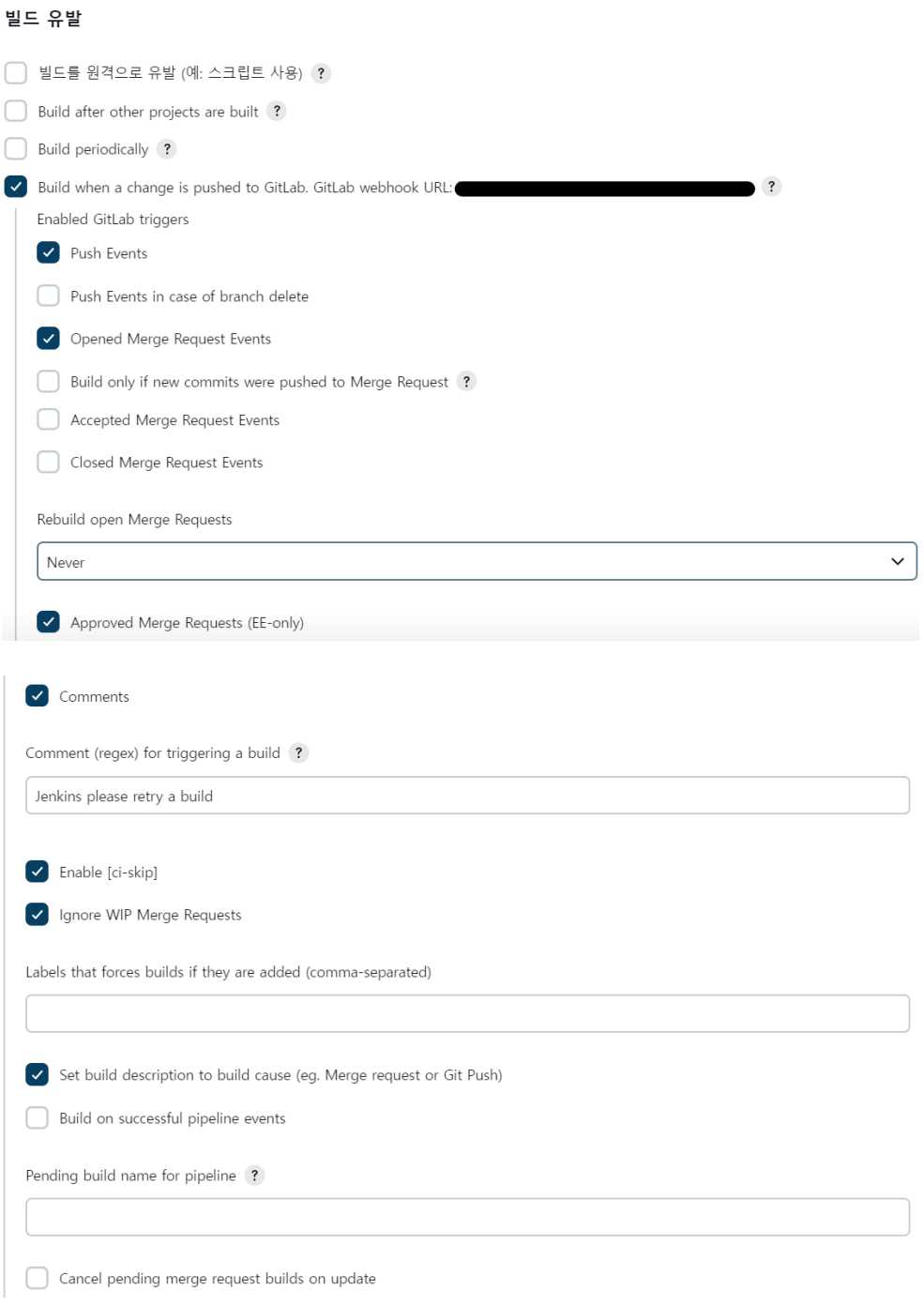
→ 저장한 credential을 클릭했을 때 에러메세지가 뜨지 않으면 정상 접근 연동에 성공

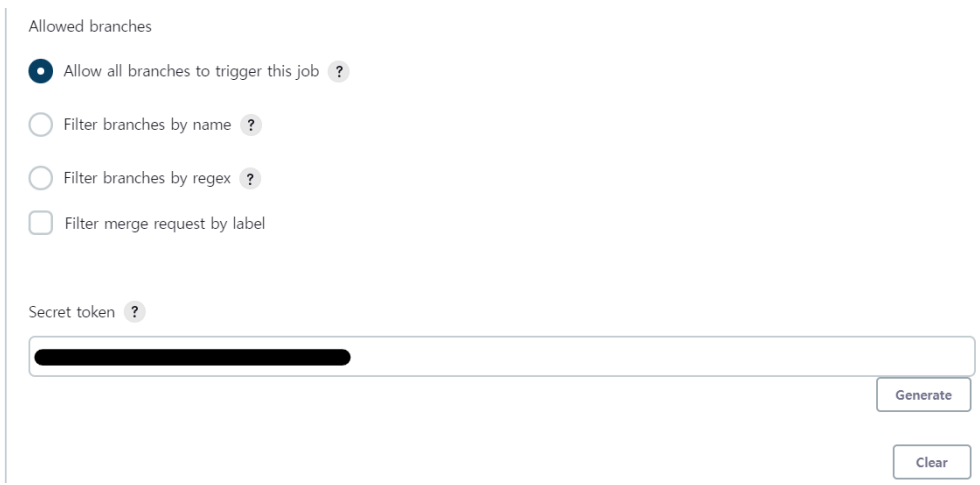
(3) 빌드 유발

- `Build when a change is pushed to GitLab` 체크

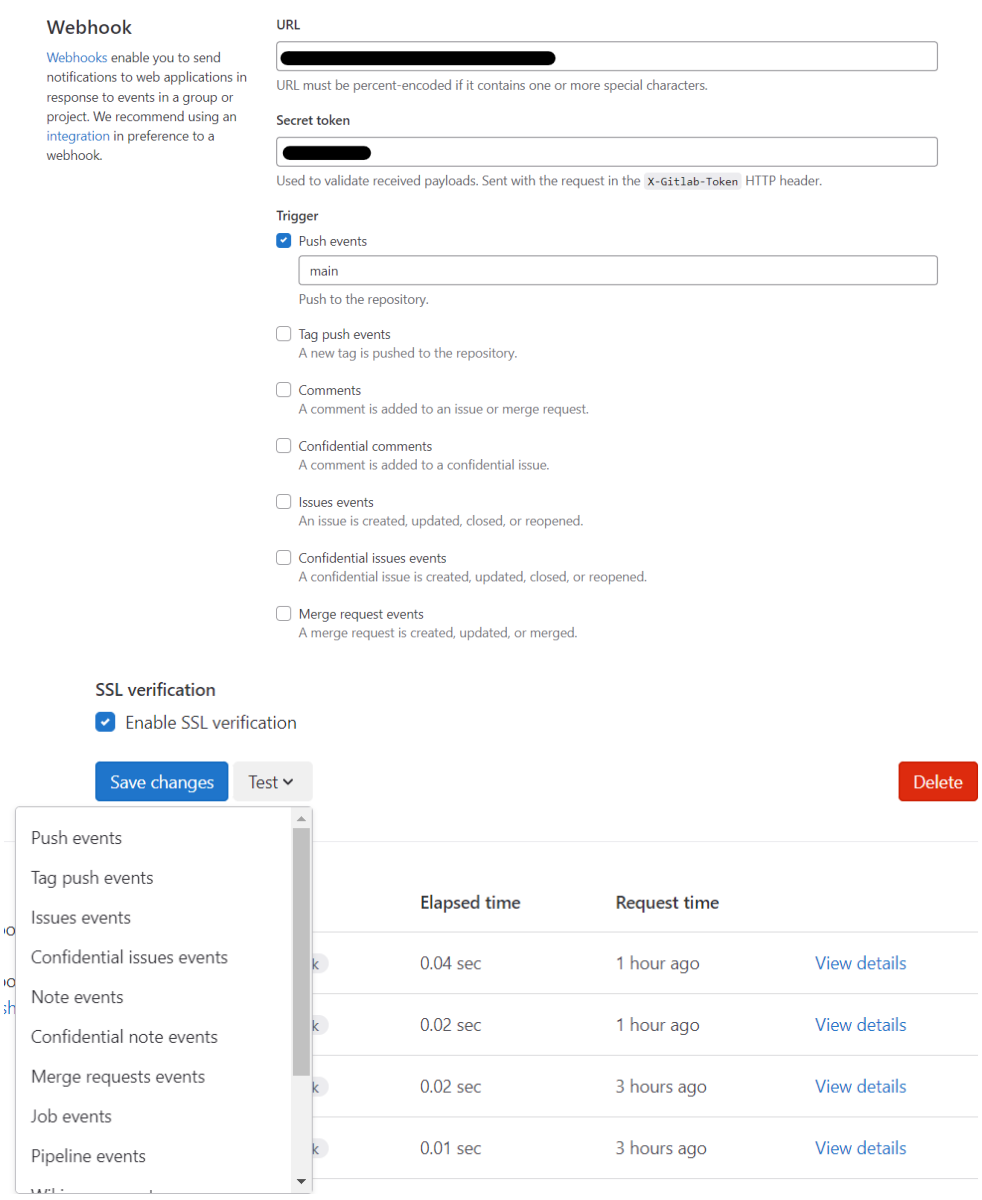
(4) Webhook 설정

1. Jenkins project 선택 > 구성 > 빌드 유발
    - 꼭 저장 눌러주기!



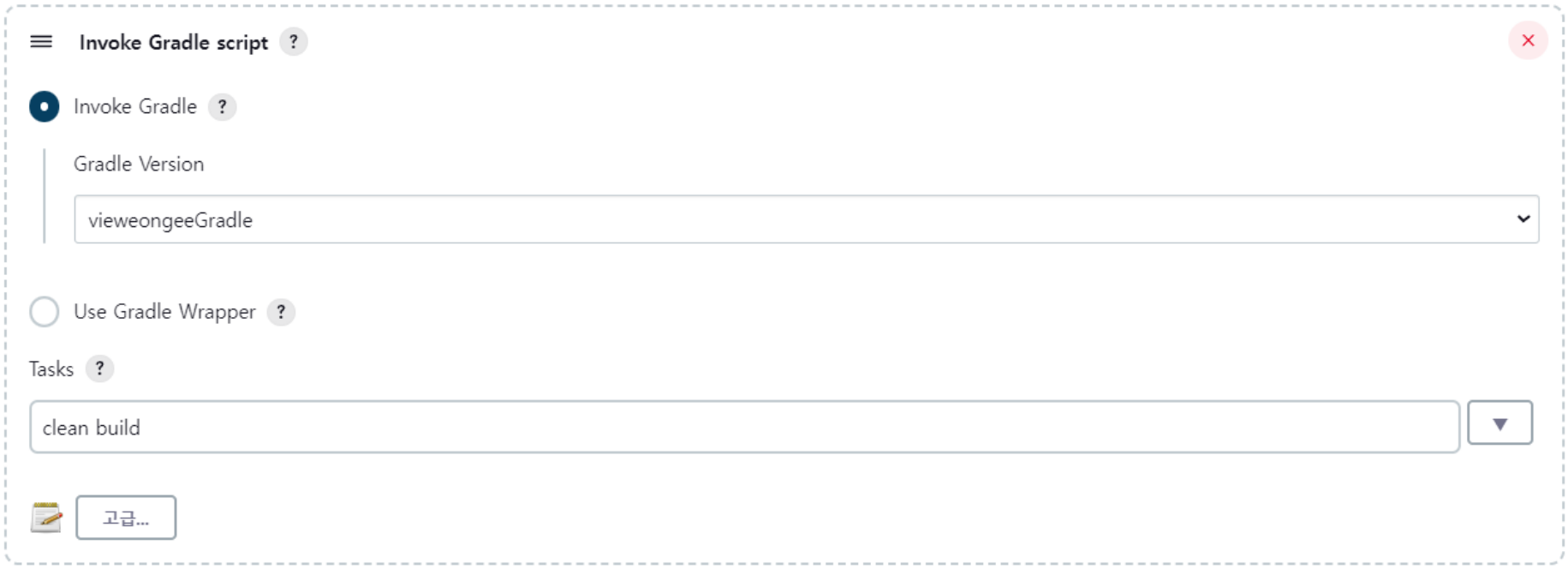


2. GitLab Settings > Webhooks



3. GitLab Webhooks의 URL과 Secret token에 Jenkins의 URL과 Secret token을 작성한다.
4. GitLab에서 main branch에 push할 때마다 자동 빌드 되게 설정해준다.
5. 저장 후, Test버튼을 눌러 원하는 테스트를 해볼 수 있다.

(5) Build Steps



(6) Execute Shell

```
# frontend 컨테이너 생성
docker build -t omz_frontend:latest ./Development/FE/omz

# 이미 실행 중인 frontend 컨테이너가 있다면 중단하기
if (docker ps | grep omz_frontend) then docker stop omz_frontend; fi

# frontend 컨테이너 실행
docker run -d --rm --name omz_frontend-p 3000:3000 omz_frontend

# backend 컨테이너 생성
docker build -t omz_backend:latest ./Development/BE/omz

# 이미 실행 중인 backend 컨테이너가 있다면 중단하기
if (docker ps | grep omz_backend) then docker stop omz_backend; fi

# backend 컨테이너 실행
docker run -d --rm --name omz_backend-p 8080:8080 omz_backend

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

(7) 저장 후 지금 빌드

빌드 추이로 확인 가능