집에서 운동중

PORTING MANUAL

목 차

I. 빌드 및 배포

1. 개발 환경

Server : AWS EC2 Ubuntu 20.04 LTS

Visual Studio Code : 1.70.1

IntelliJ IDEA : 2022.1.3(Ultimate Edition) 11.0.15 + 10-b2043.56 amd64

JVM : 11.0.16+8-post-Ubuntu-0ubuntu120.04

Docker : 20.10.17

Node.js : 18.7.0

MySQL : 8.0.30-1.el8

Redis : 7.0.4

Nginx : 1.23.1

Jenkins : 2.346.2

Openvidu : 2.22.0

Teachable Machine : 2.0

2. 설정 파일 목록 (경로와 파일은 4~9번 배포 과정 중 생성 될 수 있습니다.)

React

- **.env** : /jenkins/workspace/frontend/frontend
- **Dockerfile** : /jenkins/workspace/frontend/frontend

Spring

- **application.properties** : /jenkins/workspace/backend/backend/src/main/resources
- **application-oauth.properties** : /jenkins/workspace/backend/backend/src/main/resources
- **application.properties** : /jenkins/workspace/backend/backend/src/test/resources
- **Dockerfile** : /jenkins/workspace/backend/backend
- **deploy.sh** : /jenkins/workspace/backend/backend

Docker

- **docker-compose.yml** : /home/ubuntu

Nginx

- **app.conf** : /home/ubuntu/nginx/conf.d

3. 설정 파일 및 환경 변수 정보

React

- .env

```
WDS_SOCKET_PORT=0

REACT_APP_BASE_URL= REST API(BACKEND) 요청 URL

REACT_APP_GOOGLE_CLIENT_ID= 구글 클라이언트 ID
REACT_APP_GOOGLE_CLIENT_SECRET= 구글 클라이언트 SECRET
REACT_APP_GOOGLE_REDIRECT_URI= 구글 리다이렉트 URI

REACT_APP_NAVER_CLIENT_ID= 네이버 클라이언트 ID
REACT_APP_NAVER_CLIENT_SECRET= 네이버 클라이언트 SECRET
REACT_APP_NAVER_REDIRECT_URI= 네이버 리다이렉트 URI

REACT_APP_KAKAO_CLIENT_ID= 카카오 클라이언트 ID
REACT_APP_KAKAO_CLIENT_SECRET= 카카오 클라이언트 SECRET
REACT_APP_NAVER_REDIRECT_URI= 네이버 리다이렉트 URI
```

- Dockerfile

```
FROM node:alpine
WORKDIR /usr/src/app
COPY ./package* /usr/src/app/
RUN npm install
COPY ./ /usr/src/app/
CMD ["npm","run","start"]
```

Spring

- application.properties(main)

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://{DB컨테이너이름}:3306/{SCHEME이름
}?useSSL=false&useUnicode=true&characterEncoding=utf8&allowPublicKey
Retrieval=true&serverTimezone=Asia/Seoul&useLegacyDatetimeCode=false
spring.datasource.username=root
spring.datasource.password=ROOT계정 PASSWORD

spring.jpa.show-sql=true

spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8
Dialect

spring.profiles.include=oauth

logging.level.root = info

spring.redis.host = REDIS컨테이너명
spring.redis.port = 6379
spring.redis.password = REDIS암호

spring.jpa.open-in-view=false
```

- **application-oauth.properties**

```
#Google
spring.security.oauth2.client.registration.google.client-id= 구글
클라이언트 ID
spring.security.oauth2.client.registration.google.client-secret=
구글 클라이언트 SECRET
spring.security.oauth2.client.registration.google.redirect-uri= 구글
리다이렉트 URI
spring.security.oauth2.client.registration.google.scope= profile,
email

#Naver
spring.security.oauth2.client.registration.naver.client-id= 네이버
클라이언트 ID
spring.security.oauth2.client.registration.naver.client-secret=
```

```
네이버 클라이언트 SECRET
spring.security.oauth2.client.registration.naver.redirect-uri=
네이버 리다이렉트 URI
spring.security.oauth2.client.registration.naver.authorization-
grant-type=authorization_code
spring.security.oauth2.client.registration.naver.scope=name,email
spring.security.oauth2.client.registration.naver.client-name=Naver
#Provider-Naver
spring.security.oauth2.client.provider.naver.authorization-
uri=https://nid.naver.com/oauth2.0/authorize
spring.security.oauth2.client.provider.naver.token-
uri=https://nid.naver.com/oauth2.0/token
spring.security.oauth2.client.provider.naver.user-info-
uri=https://openapi.naver.com/v1/nid/me
spring.security.oauth2.client.provider.naver.user-name-
attribute=response

#Kakao
spring.security.oauth2.client.registration.kakao.client-id= 카카오
클라이언트 ID
spring.security.oauth2.client.registration.kakao.client-secret=
카카오 클라이언트 SECRET
spring.security.oauth2.client.registration.kakao.redirect-uri=카카오
리다이렉트 URI
spring.security.oauth2.client.registration.kakao.client-
authentication-method=POST
spring.security.oauth2.client.registration.kakao.authorization-
grant-type = authorization_code
spring.security.oauth2.client.registration.kakao.scope=profile_nickn
ame, account_email
spring.security.oauth2.client.registration.kakao.client-
name=Kakao
#Provider-Kakao
spring.security.oauth2.client.provider.kakao.authorization-
uri=https://kauth.kakao.com/oauth/authorize
spring.security.oauth2.client.provider.kakao.token-
uri=https://kauth.kakao.com/oauth/token
spring.security.oauth2.client.provider.kakao.user-info-
uri=https://kapi.kakao.com/v2/user/me
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
```

- **application.properties(test)**

```
# Datasource
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1
spring.datasource.username=h2test
spring.datasource.password=h2test

# JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.open-in-view=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dial
ect

#Google
spring.security.oauth2.client.registration.google.client-id=test
spring.security.oauth2.client.registration.google.client-secret=test
spring.security.oauth2.client.registration.google.scope=profile,
email

spring.redis.host = REDIS컨테이너명
spring.redis.port = 6379
spring.redis.password = REDIS PASSWORD
```

- **Dockerfile**

```
FROM openjdk:11-jdk
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","-Duser.timezone=Asia/Seoul","/app.jar"]
```

- **deploy.sh**

```
echo '실행 시작'
echo 'git pull'

echo 'jar 파일 삭제'
rm build/libs/*.jar

echo '빌드 시작'
./gradlew build
```

```
echo '도커파일 이미지 빌드'
docker build -t springbootapp .

echo '컨테이너 중지'
docker stop springbootapp

echo '컨테이너 삭제'
docker rm springbootapp

echo '컨테이너 실행'
docker run -p 8080:8080 --name springbootapp --network ubuntu_default
  -d springbootapp
```

Docker

- **docker-compose.yml**

```
version: "3"
services:
    mydb:
      image: mysql
      container_name: 원하는 DB컨테이너명
      environment:
        MYSQL_DATABASE: 원하는 SCHEME명
        MYSQL_ROOT_PASSWORD: 원하는 ROOT 계정 PASSWORD
      volumes:
        - /mydb:/var/lib/mysql
      ports:
         - 3306:3306

    redis:
      container_name: redis
      image: redis
      ports:
        - 6379:6379
      command: redis-server --requirepass REDIS PASSWORD
      environment:
        - REDIS_REPLICATION_MODE=master

    nginx:
      image: nginx
      container_name: 원하는 컨테이너명
```

```
    ports:
      - 80:80
      - 443:443
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt
      - ./nginx/conf.d:/etc/nginx/conf.d

  jenkins:
    image: jenkins/jenkins:lts
    container_name: 원하는 컨테이너명
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - 9090:8080
    privileged: true
    user: root
```

Nginx

- **app.conf**

```
server {
    listen 80;
    server_name 서비스도메인 www.서비스도메인;

    return 301 https://$server_name$request_uri;
 }

server {
        listen 443 ssl;
        server_name 서비스도메인;
        access_log off;

        ssl_certificate
        /etc/letsencrypt/live/서비스도메인/fullchain.pem;
        ssl_certificate_key
        /etc/letsencrypt/live/서비스도메인/privkey.pem;

        location / {
        proxy_pass http://서비스도메인:3000;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Host $server_name;
```

```
            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 Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_redirect off;
            }

            location /api/ {
            proxy_pass http://서비스도메인:8080/;
            proxy_set_header Host $host;
            proxy_set_header X-Forwarded-Host $server_name;
            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 Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_redirect off;
            # add_header 'Access-Control-Allow-Origin' '*' always;
            # add_header 'Access-Control-Allow-Methods' 'GET, POST,
            PUT, DELETE, OPTIONS' always;
            # add_header 'Access-Control-Allow-Headers' 'content-type,
            authorization, x-requested-with' always;
        }
    }
```

4. Docker 설치

- **Docker 인스톨**

```
sudo apt-get update

sudo apt-get install ₩

ca-certificates ₩

curl ₩

gnupg ₩

lsb-release

sudo mkdir -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 update

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

- **Docker Compose 인스톨**

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

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

5. SSL 인증서 발급

```
sudo apt-get install letsencrypt

sudo letsencrypt certonly --standalone -d www제외한 도메인 이름

이메일 작성 후 Agree

뉴스레터 수신 여부 Yes/No

ssl_certificate /etc/letsencrypt/live/도메인이름/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/도메인이름/privkey.pem;
```

6. OpenVidu 배포

- **기존 OpenVidu가 있다면 삭제**

```
$ sudo docker ps -a
```

#openvidu, kurento media server등의 컨테이너가 존재한다면 삭제한다.

$ sudo docker rm <ID or Name>

#컨테이너 모두 삭제를 원할 경우

$ sudo docker rm $(docker ps -a)

이미지도 삭제

$ sudo docker rmi <ID or IMAGE>

이미지 전체 삭제를 원할 경우

$ sudo docker rmi $(docker images)

- **OpenVidu 설치**

관리자 권한

$ sudo su

openvidu가 설치되는 경로

$ cd /opt

openvidu on promises 설치

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

$ exit

- **OpenVidu 설정 및 SSL 인증서 적용**

$ cd /opt/openvidu

```
$ vi .env

DOMAIN_OR_PUBLIC_IP=<도메인 또는 public IP>
OPENVIDU_SECRET=MY_SECRET
CERTIFICATE_TYPE=letsencrypt
LETSENCRYPT_EMAIL=인증서 발급 시 입력한 이메일
HTTP_PORT=8442
HTTPS_PORT=8443
...

# ESC 입력 후 :wq! 로 저장 후 나가기
```

- **OpenVidu 포트 개방**

```
sudo apt update
sudo apt install netfilter-persistent
sudo apt install iptables-persistent

sudo service iptables start

sudo iptables -A INPUT -p udp --match multiport --dports 40000:65535 -j ACCEPT
sudo iptables -A INPUT -p tcp --match multiport --dports 40000:65535 -j ACCEPT
sudo iptables -I INPUT 1 -p tcp --dport 22 -j ACCEPT
sudo iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT
sudo iptables -I INPUT 1 -p tcp --dport 443 -j ACCEPT
sudo iptables -I INPUT 1 -p tcp --dport 3478 -j ACCEPT
sudo iptables -I INPUT 1 -p udp --dport 3478 -j ACCEPT
```

```
sudo iptables -I INPUT 1 -p tcp --dport 8082 -j ACCEPT
sudo iptables -I INPUT 1 -p tcp --dport 8443 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 8080 -j ACCEPT
sudo service iptables save
sudo service iptables restart
sudo netfilter-persistent save
sudo netfilter-persistent start
```

- **OpenVidu 서비스(관련 컨테이너) 실행**

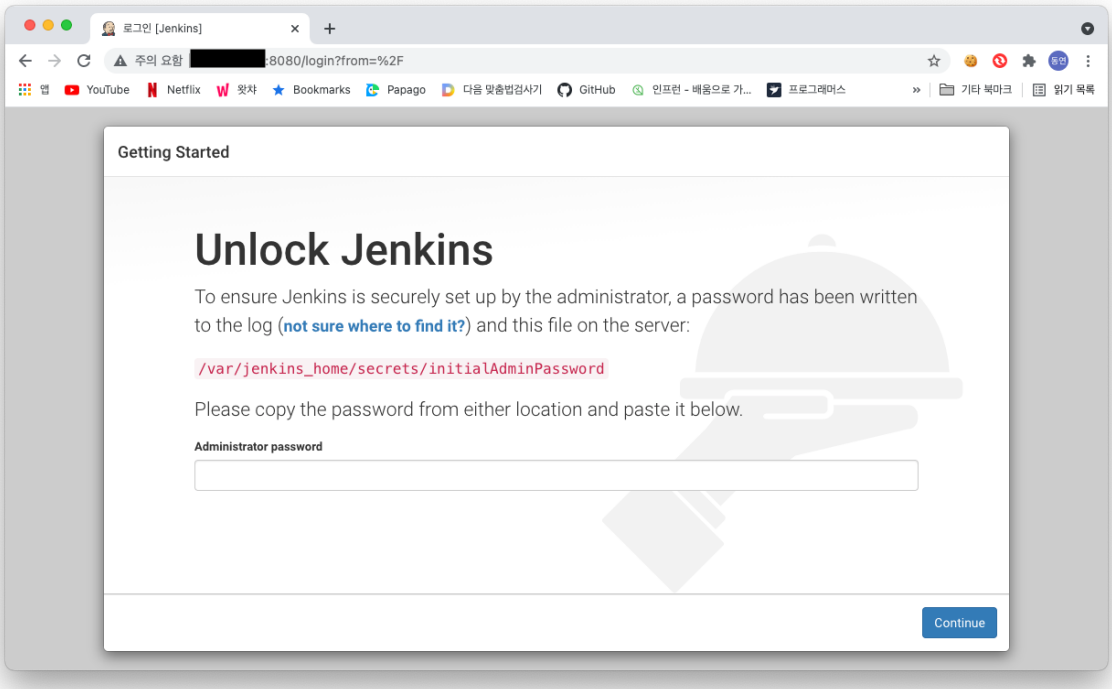
```
# /opt/openvidu 위치에서
$ sudo ./openvidu start

# 종료할 때는 같은 경로에서
$ ./openvidu stop
```

7. DB 및 Infra 배포

- **docker-compose 작성** : 본 문서의 I-2, I-3의 docker-compose.yml 참조
- **docker-compose 실행**

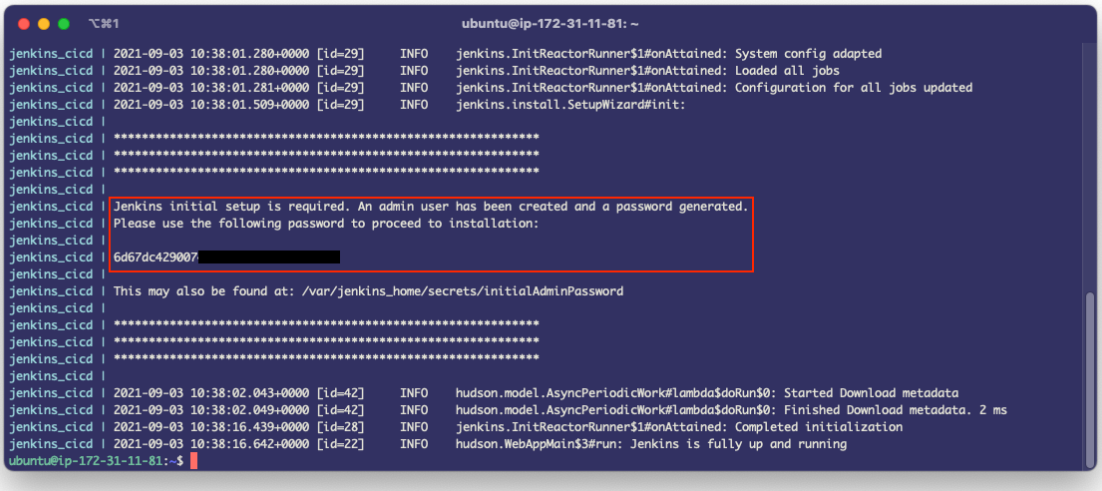
```
# /home/ubuntu (docker-compose.yml 경로에서)
$ sudo docker-compose up -build -d
```

- **Nginx 설정** : 본 문서의 I-2, I-3의 app.conf 참조
- **Jenkins 설정**

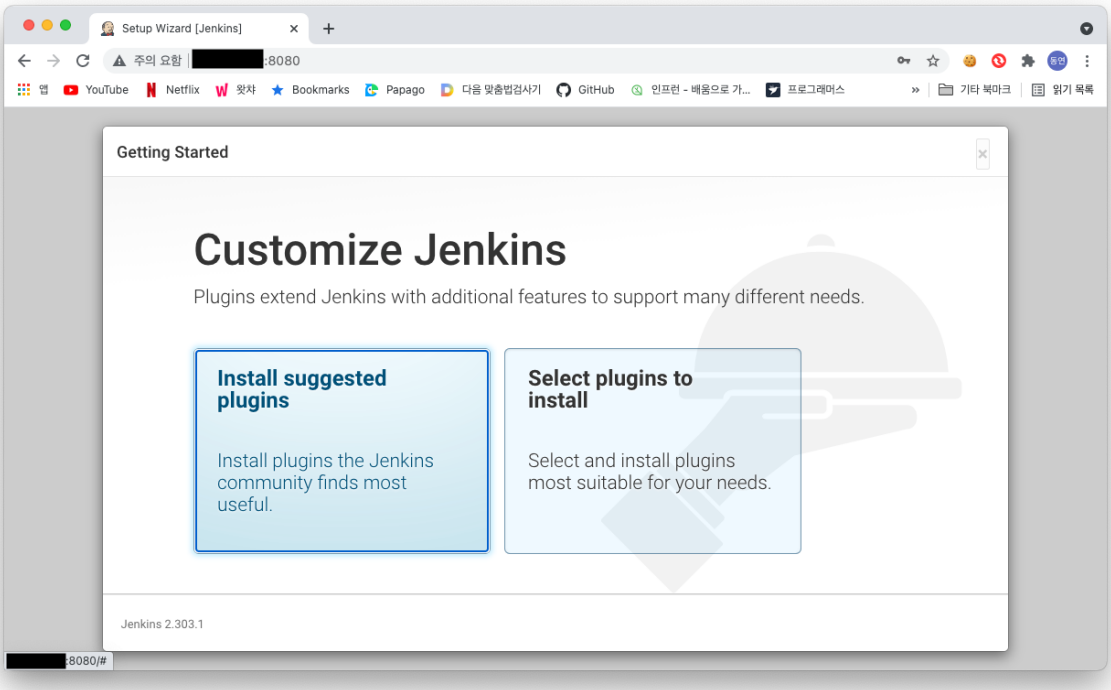


-

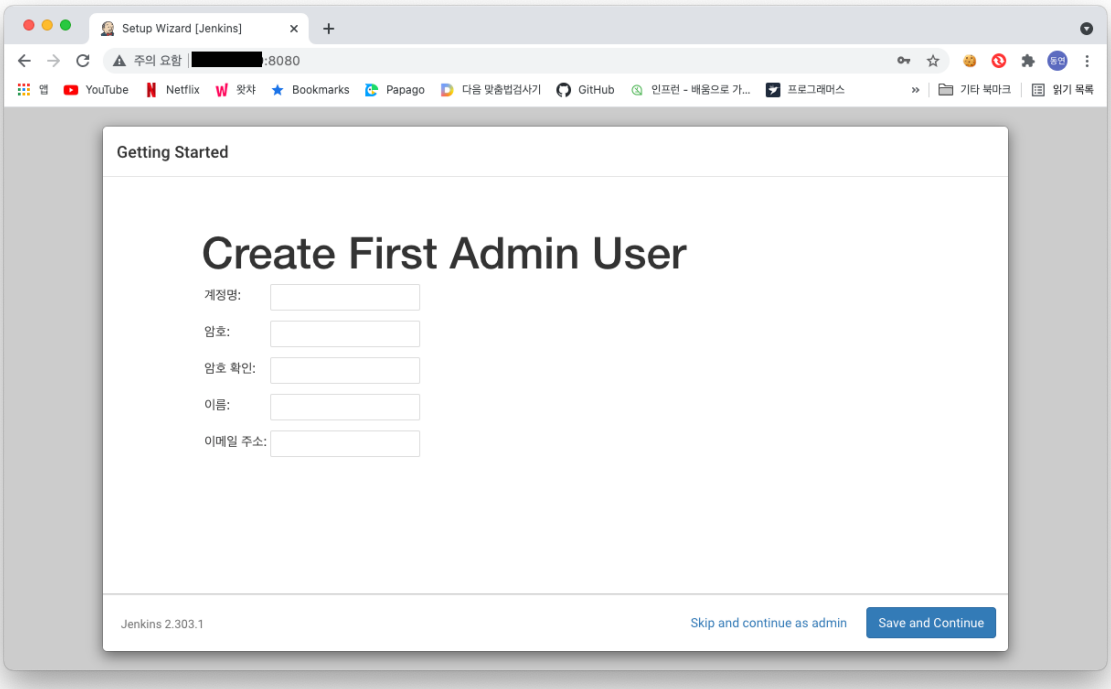
1. http://서비스도메인:9090 으로 접속하여 jenkins 페이지 진입



2. 서버 콘솔에서 sudo docker logs [Jenkins 컨테이너 이름]으로 Administrator password를 확인하고 입력합니다.

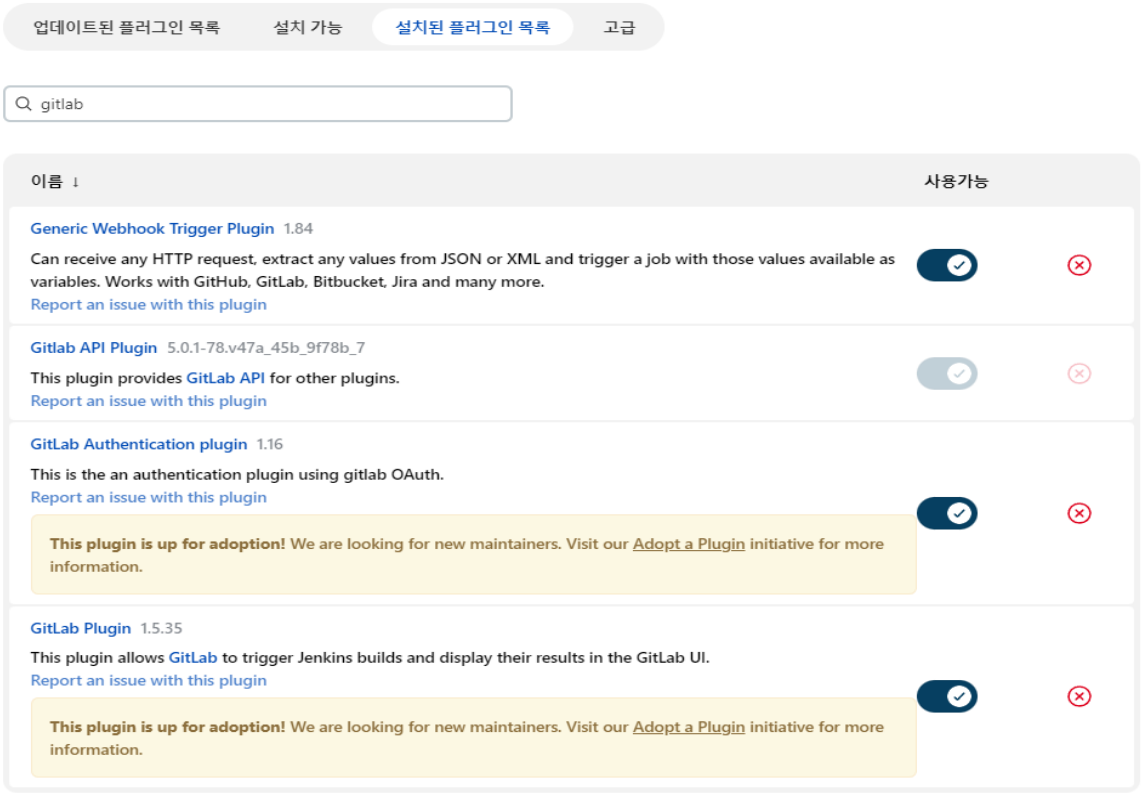


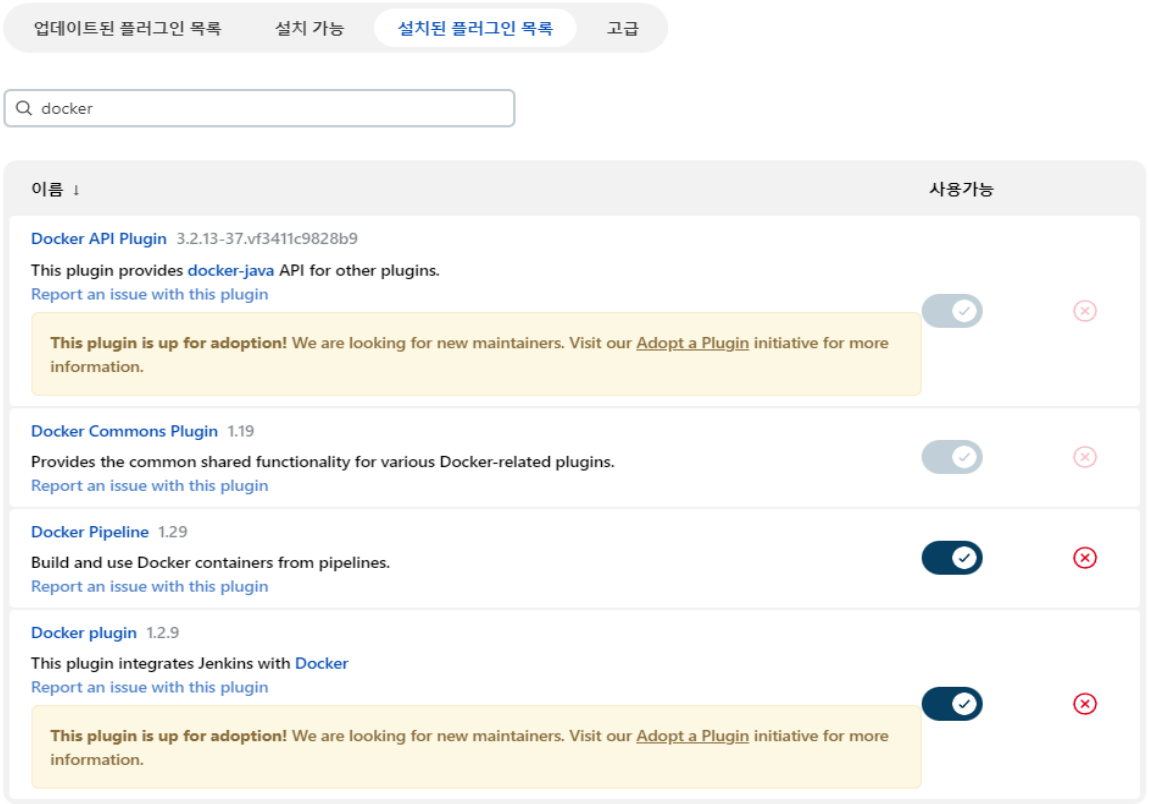
3. Install suggested plugins을 선택하여 플러그인들을 설치합니다.



4. 생성할 관리자 계정 정보를 입력하고 Save and Continue

5. Jenkins 접속 URL확인 후 Save and Finish





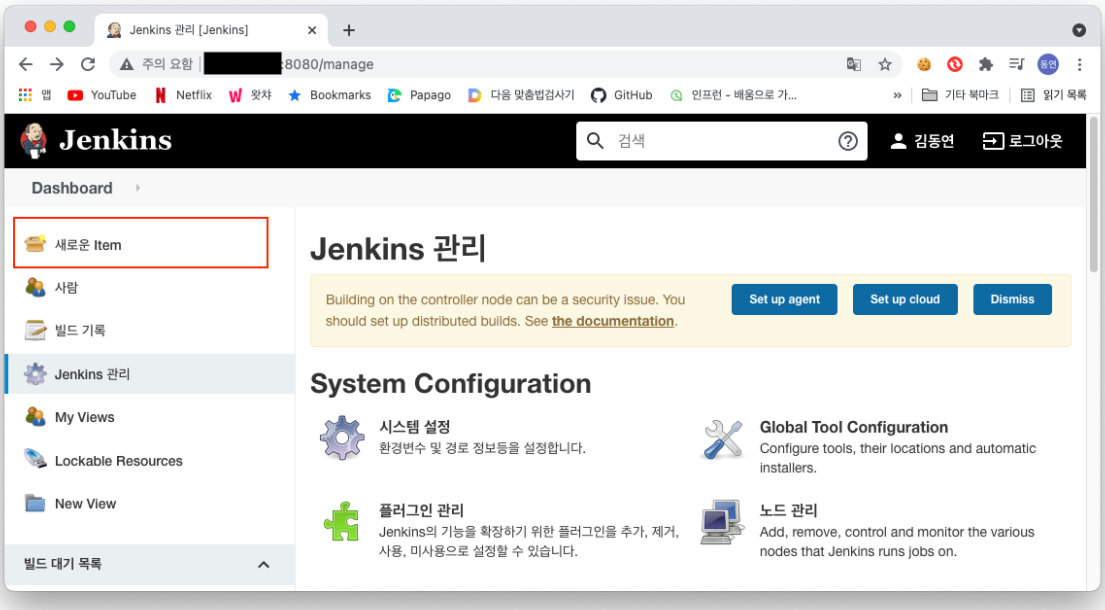
6. 메인 화면에서 DashBoard -> Manager JenKins -> Plugin Manager에서 gitlab, docker에 대해서 플러그인을 설치합니다.

7. 서버 콘솔로 돌아가 Jenkins 내부에 docker를 설치합니다.

```
$ sudo docker exec -it jenkins컨테이너명 /bin/bash
$ apt-get update -y
$ apt-get install -y
$ apt-get install docker.io -y
$ docker -v
```

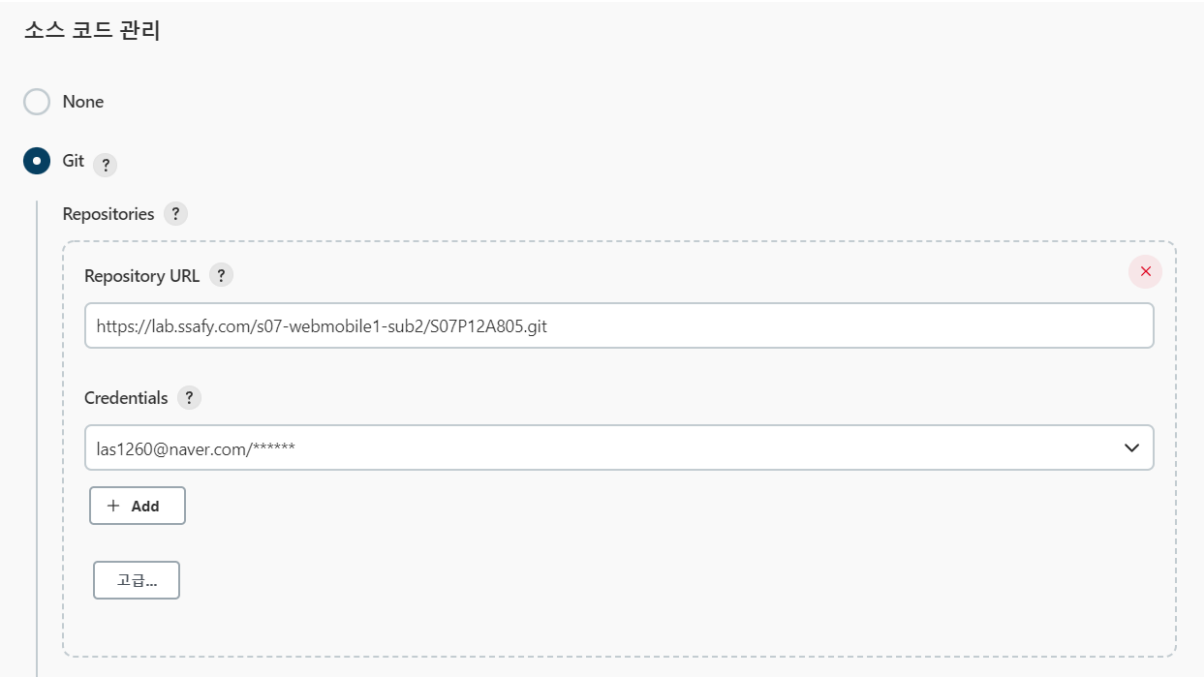
8. Backend CI/CD

- **application.properties 작성**: 본 문서의 I-2, I-3 applictaion.properties 참조
- [redacted] : 본 문서 I-2, I-3 application-oauth.properties 참조
  (본 문서의 II-1 소셜 로그인 부분을 먼저 진행하시는 것이 좋습니다.)
- **application.properties(test) 작성**: 본 문서의 I-2, I-3 application.properties(test) 참조
- **Dockerfile 작성**: 본 문서의 I-2, I-3 Backend(Spring) Dockerfile 참조
- **Jenkins Job 설정**



1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭

2. FreeStyle project를 선택하고 item name은 backend로 설정 후 OK



3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력

   Credentials의 Add를 클릭하고,

   Domain -> Global credentials

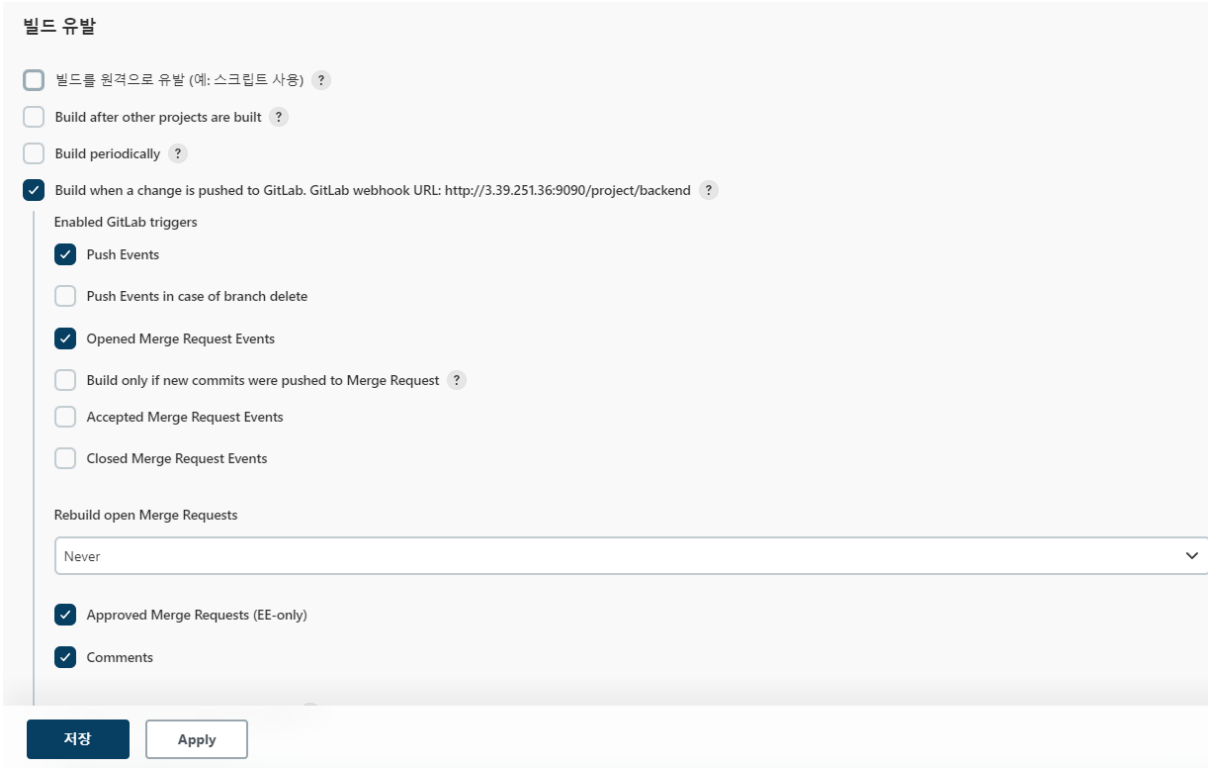
   Kind -> Username with password

   Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디

   Password -> Username에 작성한 Gitlab 계정 비밀번호

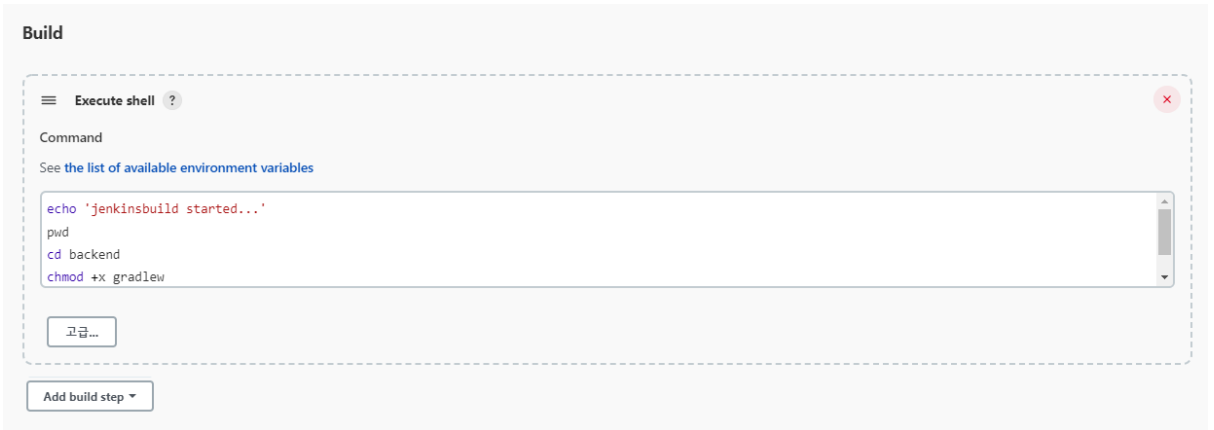
   를 순서대로 입력하고 드롭박스에서 생성된 Credential을 선택해준다.

4. backend 브랜치의 내용만을 받아와서 빌드하기 위한 설정을 해줍니다.



5. 빌드 유발을 다음과 같이 설정합니다.

   하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록해둡니다.



6. Build 탭에 Excute shell을 선택하고

   cd backend

   chmod +x gradlew

   ./deploy.sh

   을 작성해줍니다.

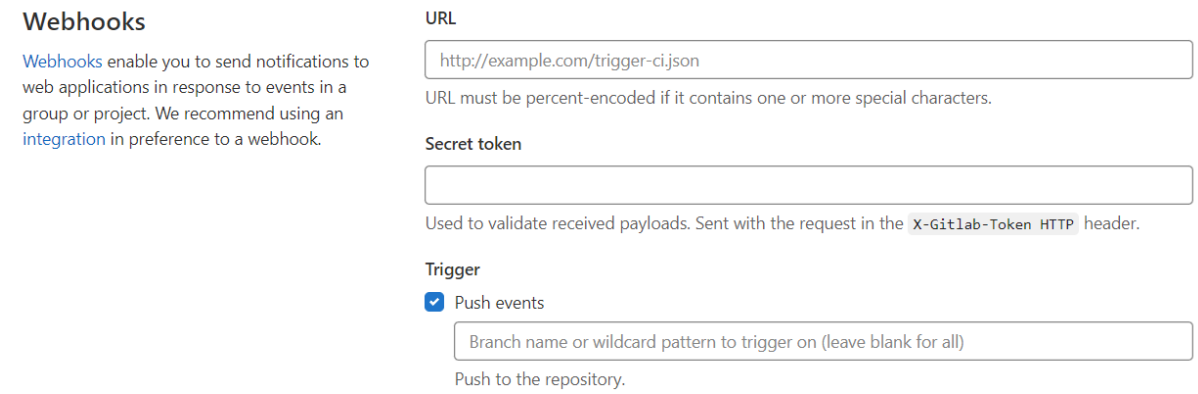
- **deploy.sh 작성**: 본 문서의 I-2, I-3의 deploy.sh 참조
- **jenkins에 sudo 권한 부여**

  $ sudo vim /etc/sudoers

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
jenkins ALL=(ALL) NOPASSWD: ALL
```

  해당 부분에 jenkins ALL=(ALL) NOPASSWD: ALL 작성

- **Gitlab Webhook 작성**



  Gitlab 프로젝트 Repository의 Setting > Webhooks 로 이동

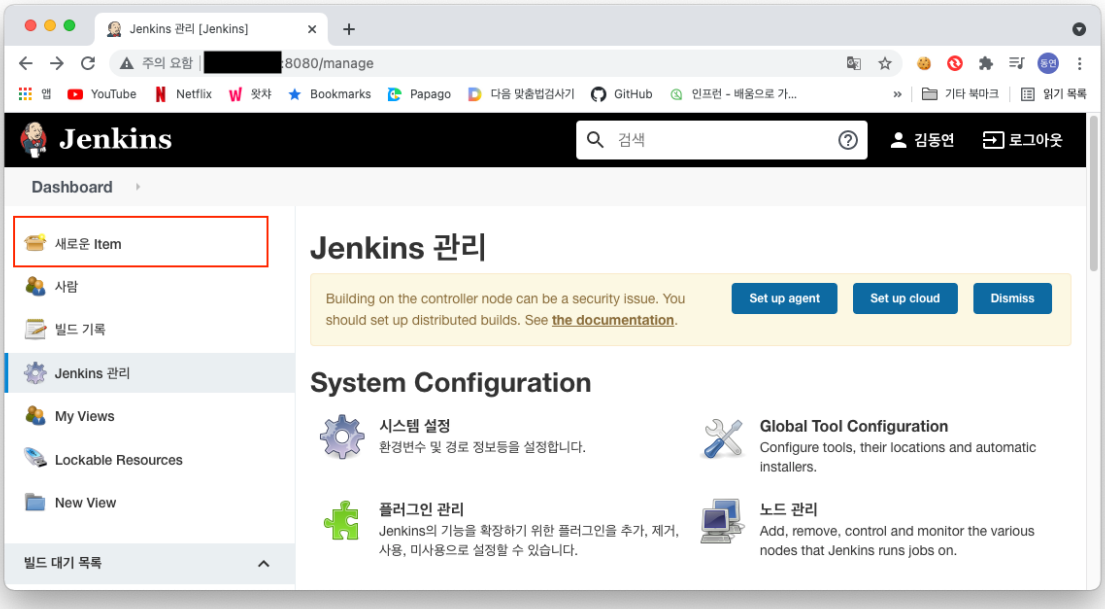
  URL에는 Jenkins의 빌드 유발의 Webhook URL을 입력

  Secret Token에는 빌드 유발에서 생성했던 Secret Token을 입력

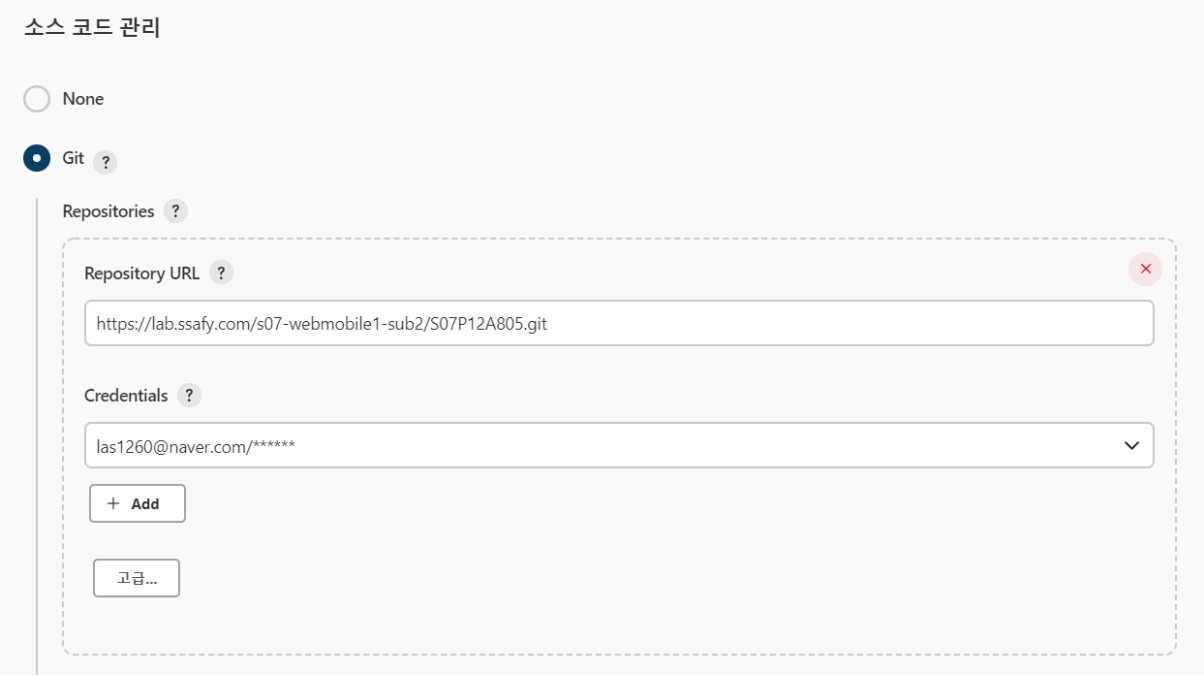
  Trigger의 Push events를 체크하고 backend 입력 후 Add webhook

9. Frontend CI/CD

- **.env 작성**: 본 문서의 I-2, I-3 .env 참조
- **Dockerfile 작성**: 본 문서의 I-2, I-3 Frontend(React) Dockerfile 참조
- **jenkins Job 설정**



1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭
2. FreeStyle project를 선택하고 item name은 frontend로 설정 후 OK



3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력

   Credentials의 Add를 클릭하고,

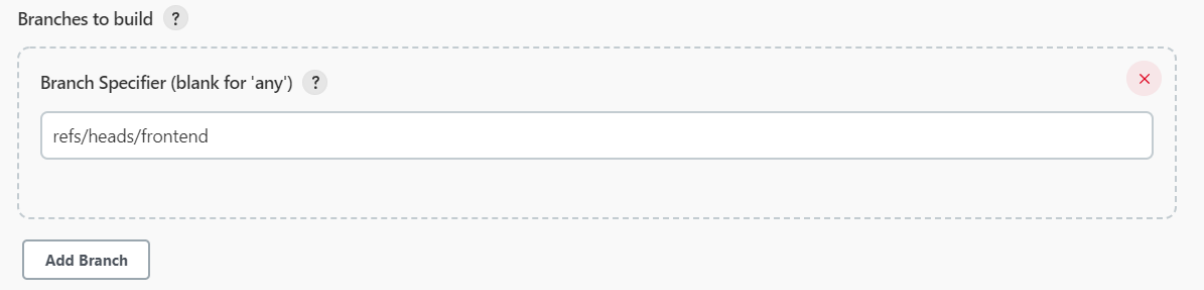
   Domain -> Global credentials

   Kind -> Username with password

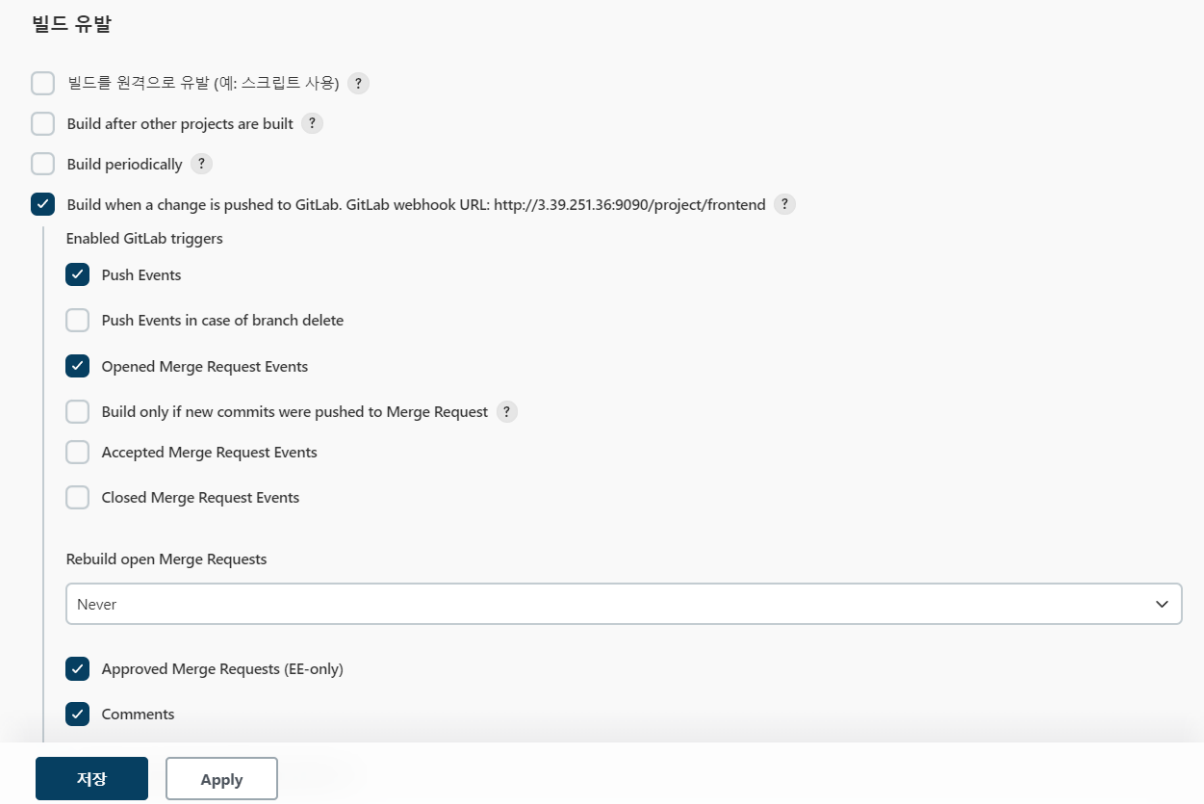
Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디

Password -> Username에 작성한 Gitlab 계정 비밀번호

를 순서대로 입력하고 드롭박스에서 생성된 Credential을 선택해준다.

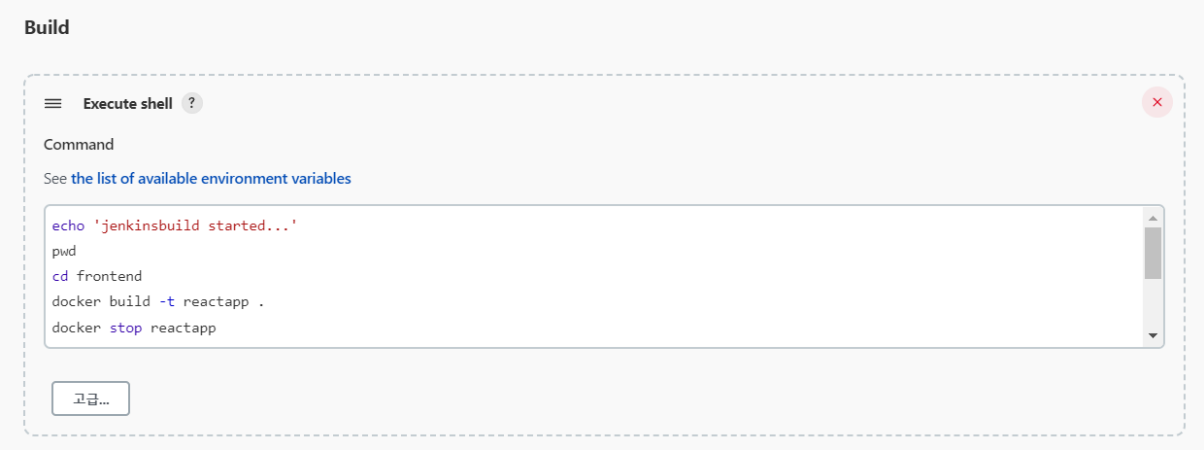


4. frontend 브랜치의 내용만을 받아와서 빌드하기 위한 설정



5. 빌드 유발을 다음과 같이 설정합니다.

하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록해둡니다.



6. Build 탭에 Excute shell을 선택하고

cd frontend

docker build -t reactapp .

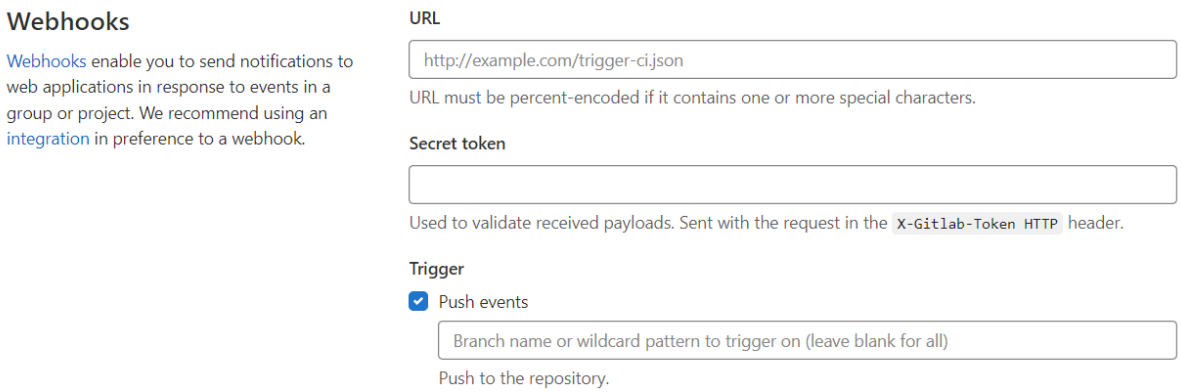
docker stop reactapp

docker rm reactapp

docker run -p 3000:3000 --name reactapp --network ubuntu_default -d reactapp

을 작성해줍니다.

- Gitlab Webhook 작성



Gitlab 프로젝트 Repository의 Setting > Webhooks 로 이동

URL에는 Jenkins의 빌드 유발의 Webhook URL을 입력

Secret Token에는 빌드 유발에서 생성했던 Secret Token을 입력

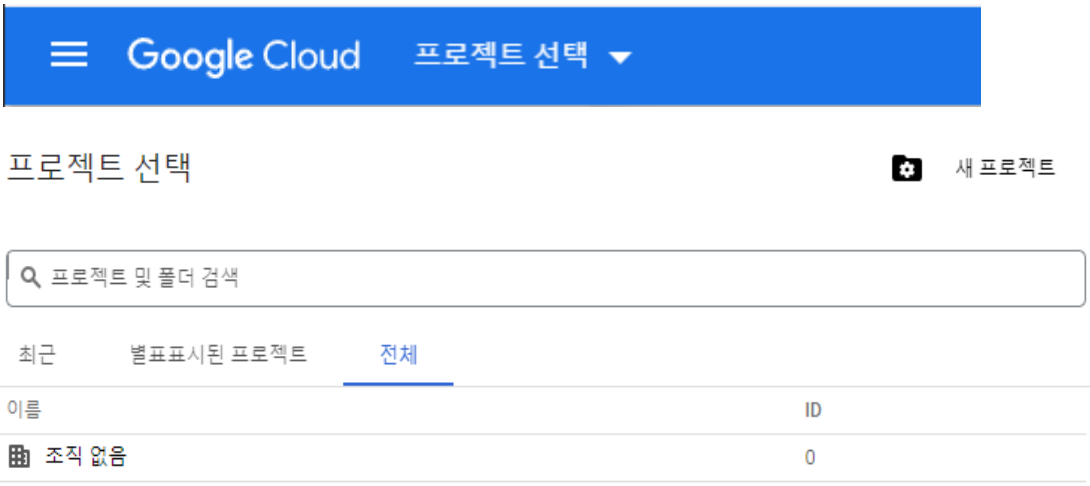
Trigger의 Push events를 체크하고 backend 입력 후 Add Webhook

II. 외부 서비스

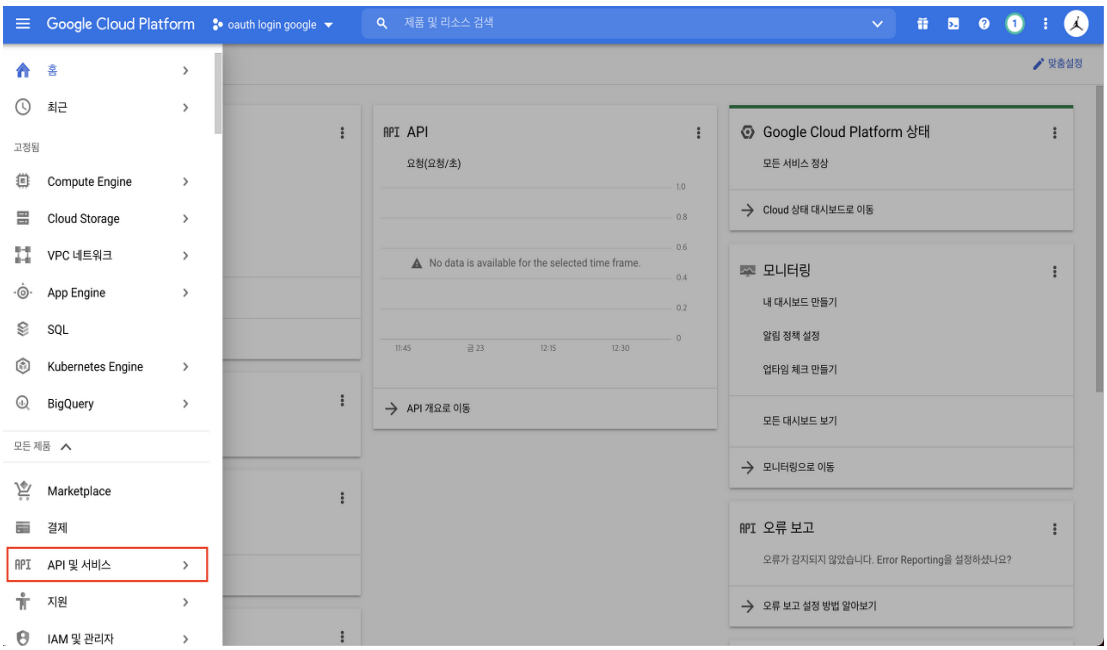
1. 소셜 로그인

- 구글(Google)

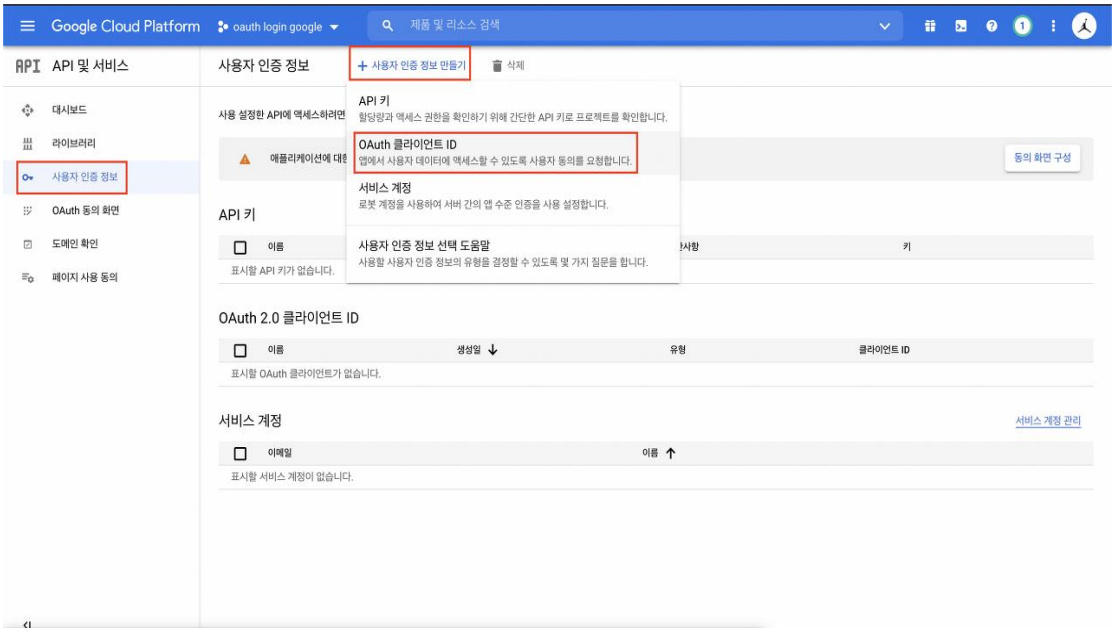
1. https://console.cloud.google.com/home/dashboard 로 접속한다.



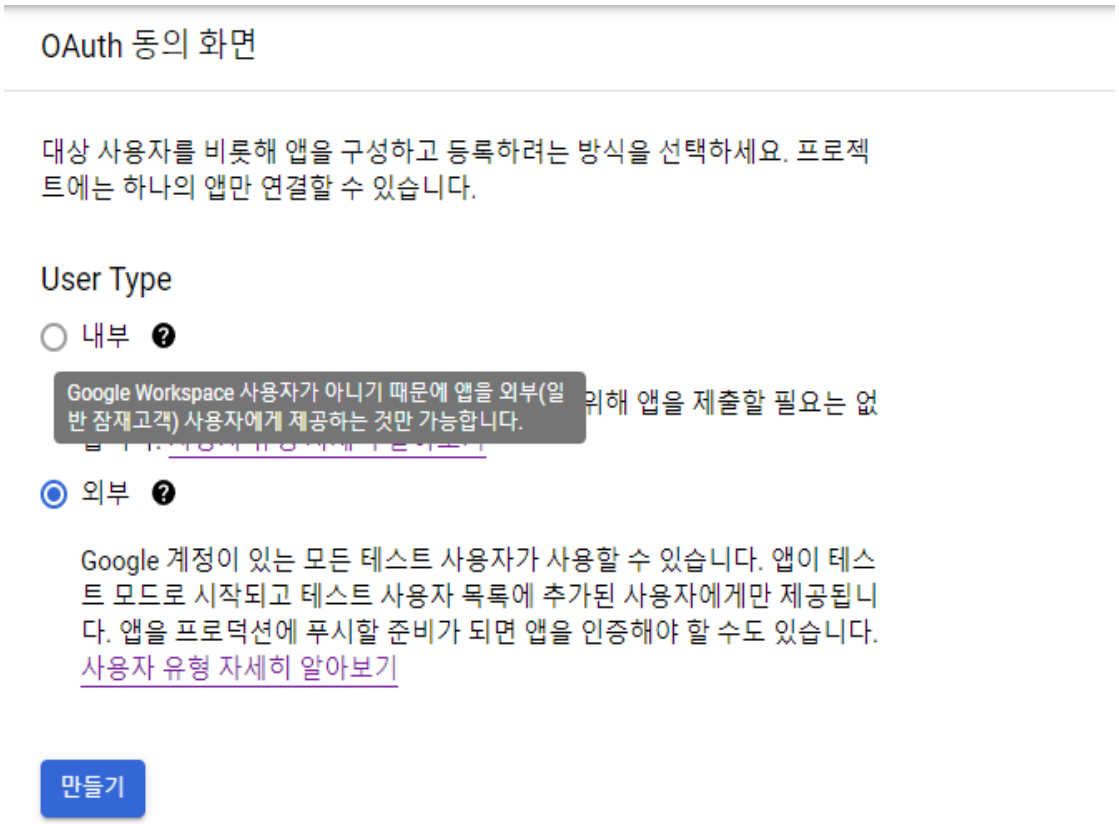
2. 프로젝트 선택을 클릭하고 모달 우측 상단의 새 프로젝트 클릭



3. API 및 서비스 선택

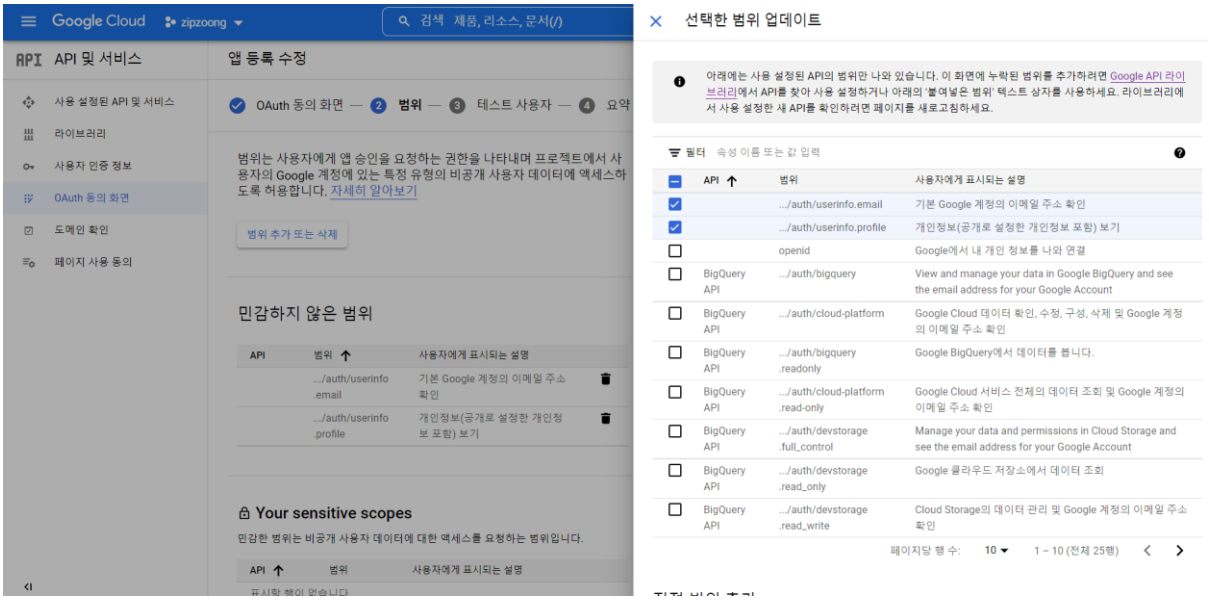


4. 사용자 인증정보 > 사용자 인증 정보 만들기 > Oauth 클라이언트 ID



5. 동의 화면 구성을 클릭하고 해당 화면에서 외부로 선택하여 외부 사용자들이 사용할 수 있도록 설정합니다.

6. 앱 정보를 입력하고, 개발자 연락처 정보에 이메일을 입력합니다.



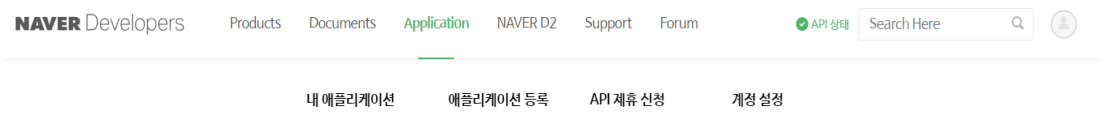
7. 범위 추가 또는 삭제 > email, profile 선택 후 저장

   민감하지 않은 범위에 추가된 것 확인

8. 사용자 인증정보 > 사용자 인증정보 만들기 -> Oauth 클라이언트 ID
   승인된 리디렉션 URI 작성

- 네이버(Naver)

1. https://developers.naver.com/main/ 로 접속한다



2. 상단의 Application > 애플리케이션 등록 클릭

3. 필요 정보를 기입합니다.

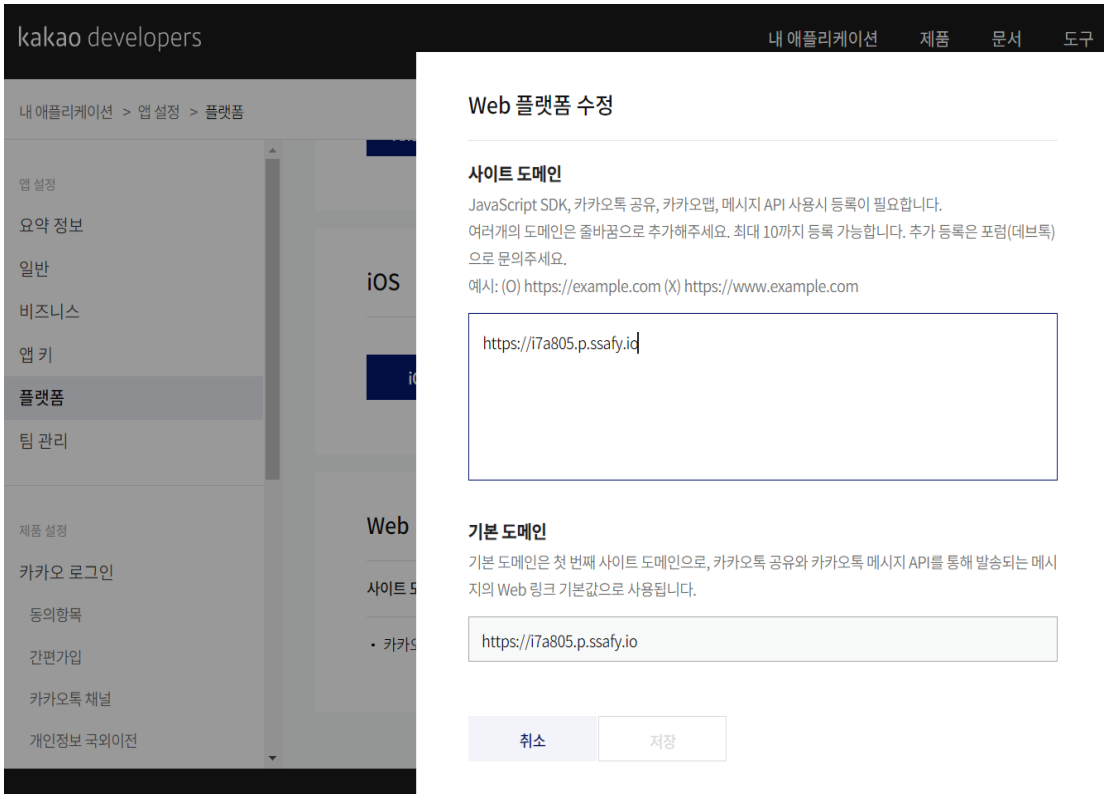
- **카카오(Kakao)**

1. https://developers.kakao.com/ 로 접속한다
2. 내 애플리케이션 > 애플리케이션 추가하기 > 앱 이름 입력 > 사업자명 입력 > 저장
3. 좌측 Nav 바에서

   앱 설정 > 요약 정보 > 앱 키 > REST API키

   제품 설정 > 카카오 로그인 > 보안 > Client Secret의 코드 발급

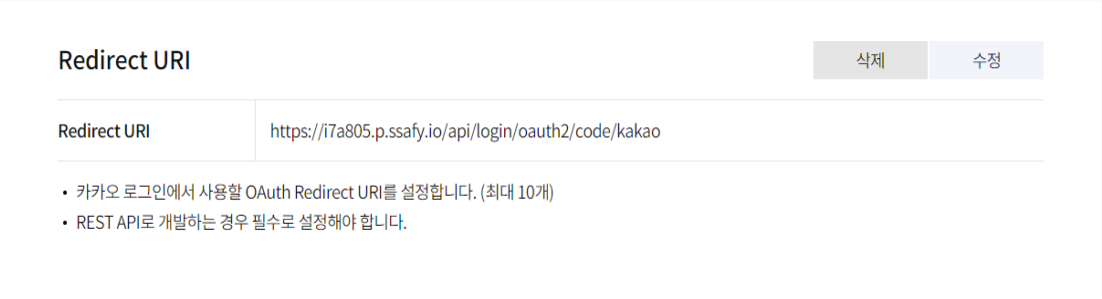
   기록해둡니다.



4. 좌측 Nav 바에서

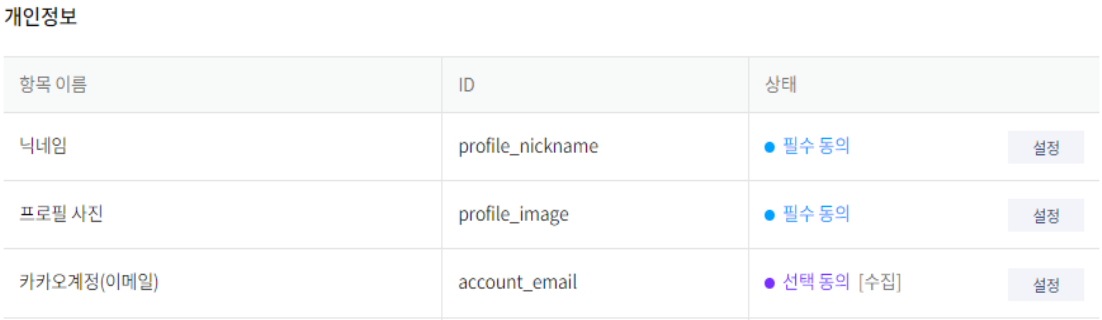
   앱 설정 > 플랫폼 > Web > Web 플랫폼 등록

   필요 정보를 기입합니다.



5. 제품 설정 > 카카오 로그인 클릭

Redirect URI를 입력해줍니다.

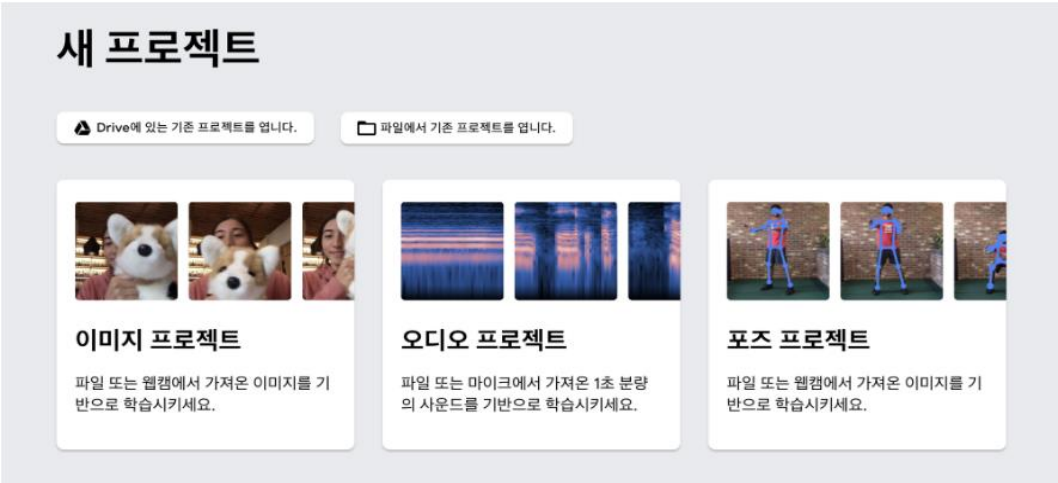


6. 제품 설정 > 카카오 로그인 > 동의항목에서 위처럼 설정합니다.

- **application-oauth.properties 작성**: 본 문서의 I-2, I-3의 Backend(Spring) application-oauth.properties를 참조

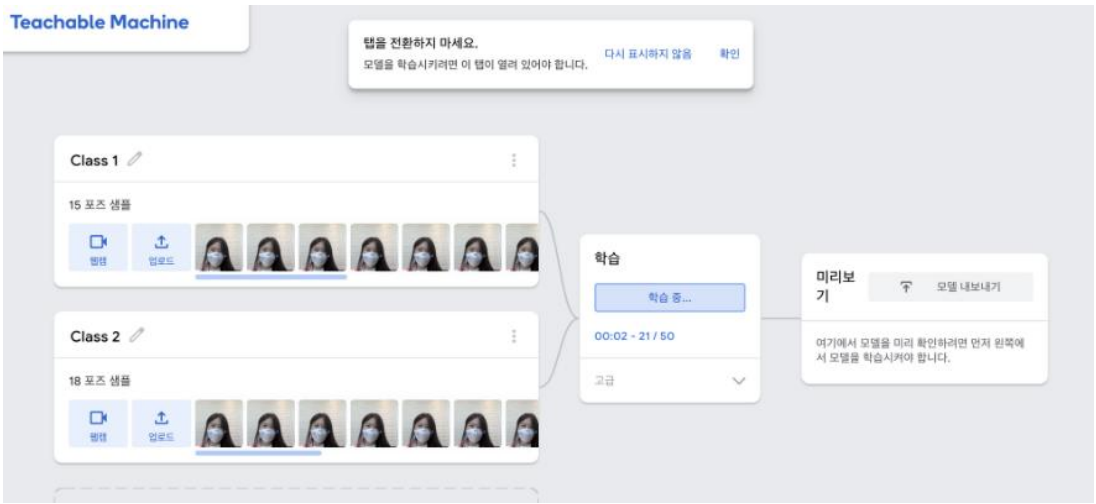
2. Teachable Machine

- **모델 학습**



1. Teachable Machine 홈페이지에서 포즈 프로젝트 선택

2. 모델의 클래스 촬영



3. 모델 학습시키기 버튼

4. 학습 완료 후 모델 내보내기

- 모델 학습

1. Tenserflow, Teachable Machine Pose 라이브러리 CDN으로 삽입

```
<script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@teachablemachine/pose@0.8/dist/teachablemachine-pose.min.js"></script>
```

2. 학습한 모델 url로 모델 load

```
model = await tmPose.load(modelURL, metadataURL);
```