포팅 메뉴얼

A402: Hoodies

삼성 청년 소프트웨어 아카데미 7기 서울캠퍼스

자율프로젝트 (2022/10/21~2022/11/21, 7주)

포팅 매뉴얼

담당 컨설턴트- **이상현**

이름	역할
오현규	팀장, FE
김무종	팀원, FE, DevOps, AI
이상현	팀원, FE
임재현	팀원, FE, AI
임재훈	팀원, BE
조준식	팀원, BE

<<목차>>

도입

싸피 교육 과정을 이수하면서, 싸피 과정 속에서 아쉬운 문제가 있었습니다.

첫 번째 문제는 싸피생만을 위한 다른 교육생들의 이야기를 들을 수 있는 채널이 없다는 점입니다. 매터모스트는 공적인 공간이라 교육생들이 자유롭게 이야기하기에는 제약이 따릅니다, 한편 오픈 카카오톡의 싸피 채널은 싸피 교육생 인증이 불가능 하여 싸피생 만의 커뮤니티라고 보기에는 힘들었습니다..

두 번째 문제는, 싸피 특성상 프로젝트 팀원들과 이야기를 주로 할 수밖에 없다는 점입니다. 2학기 프로젝트 내내 아쉬웠던 점 중 하나는 다른 반 사람들이나 같은 반이어도 다른 조에 소속된 사람들과 이야기를 많이 못해본 것입니다. 싸피 교육생들이 익명으로 이야기할 수 있

는 공간이 있다면, 좀 더 교류를 할 수 있지 않았을까요?

세 번째 문제는 매번 바뀌는 컨설턴트님, 실습코치님, 교육프로님들에 대한 정보가 부족하여 다소 낯설게 느껴질 수 있다는 점입니다. 이러한 점은 새로운 반 환경에 적응하는데 많은 시간을 쓰게 하는 한편, 프로젝트를 진행하는 데에 있어서 영향이 갈 수밖에 없습니다.

이를 싸피생에, 싸피생의, 싸피생을 위한 커뮤니티인 Hoodies를 통해 해결하고자 합니다.

1. 프로젝트 기술 스택

프로젝트 기술 스택

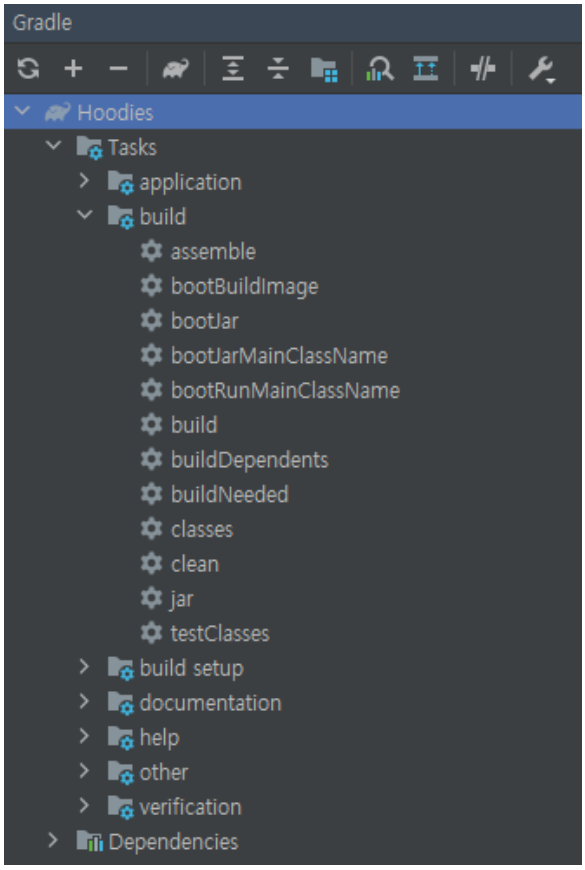
분류	이름	버전
이슈관리	Jira	
형상관리	GitLab	
CI/CD	Jenkins	
협업	Mattermost	
협업	Notion	
협업	Discord	
협업	Webex	
Operating System	Windows 10	10
IDE	IntelliJ IDEA CE	2022.1.3
IDE	Visual Studio Code	1.69.1
Database	MySql WorkBench	8.0 CE
Database	MongoDB	
Server	AWS EC2	
Back-end	JAVA 8	8
Back-end	Spring boot, Gradle	
Back-end	Lombok, Swagger	
Back-end	Spring data JPA	
Back-end	Spring Security, JWT	
Back-end	Mattermost API	
Back-end	Simple Magic	
Front-end	React	18.2.0
Front-end	React-router-dom	5.3.4
Front-end	React-scripts	5.0.1
Front-end	Styled-components	5.3.6
Front-end	@mui/icons-material	5.10.9
Front-end	@mui/material	5.10.10
Front-end	@material-ui/core	4.12.4
Front-end	@material-ui/icons	4.11.3
Front-end	@nivo/radar	0.80.0
Front-end	@nivo/core	0.80.0

분류	Aa 이름	버전
Front-end	@toast-ui/react-editor	3.2.2
Front-end	@toast-ui/editor-plugin-color-syntax	3.1.0
Artificial Intelligence	Flask	
Artificial Intelligence	Korean UnSmile Dataset	
Artificial Intelligence	Google vision API(safe search)	
AWS EC2	Nginx	
AWS EC2	Docker	
AWS EC2	Docker-compose	

2. 빌드 상세 내용

Spring build

- 우측의 Gradle → Hoodies → Tasks → build → bootJar를 더블 클릭
- build 폴더가 생성 후, 하단 libs 폴더 하위에 `Hoodies-0.0.1-SNAPSHOT.jar`가 빌드되었는지 확인



- **실행 방법 (백그라운드)**

```
nohup java -jar Hoodies-0.0.1-SNAPSHOT.jar &
```

Hate Speech Filter Server build

- **실행 방법**
  - 클론 받은 후, ai 디렉터리 안에 있는 HateSpeechDetector 폴더로 이동

- 도커 컨테이너를 통한 실행

```
docker ps -q --filter name=hoodies_ai_container | grep -q . && docker stop hoodies_ai_container && docker rm hoodies_ai_container

docker ps -a -q --filter name=hoodies_ai_container | grep -q . && docker rm hoodies_ai_container

docker build -t hoodies_ai_image .
docker run -d -p 8081:8081 --name hoodies_ai_container hoodies_ai_image
```

- 백그라운드 실행

```
pip3 install -r requirements.txt
# 의존성 주입

python3 Smilegate_Hate_Speech_Detector.py --noreload
# 백그라운드로 실행
```

Image Moderation Server build

- **실행 방법**
  - 클론 받은 후, contentFilter 안으로 이동
  - 도커 컨테이너를 통한 실행

```
cp /var/jenkins_home/winter-alliance-368105-8cf0b5f21e09.json .
# google vision API key file을 복사

docker ps -q --filter name=hoodies_filter_container | grep -q . && docker stop hoodies_filter_container && docker rm hoodies_filte

docker ps -a -q --filter name=hoodies_filter_container | grep -q . && docker rm hoodies_filter_container

docker build -t hoodies_filter_image .
docker run -d -p 8083:8083 --name hoodies_filter_container hoodies_filter_image
```

  - 백그라운드 실행

```
cp /var/jenkins_home/winter-alliance-368105-8cf0b5f21e09.json .
# google vision API key file을 복사

pip3 install -r requirements.txt
# 의존성 주입

python3 content_moderation.py --noreload
# 백그라운드로 실행
```

React build

- 빌드 방법 - clone 받은 /frontend/hoodies로 이동한 후, 다음 커멘드 입력

```
npm install --force
npm run build
```

- build 파일 생성되었는지 확인

3. 배포 특이사항

1) docker engine 설치

https://docs.docker.com/engine/install/ubuntu/

- Repository 설정

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

- GPG 키

```
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
```

- 도커 설치

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

2) Jenkins 설치

https://www.jenkins.io/download/

- 젠킨스 컨테이너 초기 설정

```
Dockerfile
# Dockerfile

FROM jenkins/jenkins:lts  # 젠킨스 lts 버전을 다운로드 받음

USER root  # 이후의 명령어의 사용자를 root로 설정함

# docker install
RUN apt-get update && \
 apt-get -y install apt-transport-https \
 ca-certificates \
 curl \
 gnupg2 \
 zip \
 unzip \
 software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
 add-apt-repository \
 "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
 $(lsb_release -cs) \
 stable" && \
 apt-get update && \
 apt-get -y install docker-ce

 # docker-compose install
 RUN curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin
    chmod +x /usr/local/bin/docker-compose && \
    ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

- 다음의 명령어로 외부 도커 환경과 젠킨스 컨테이너의 볼륨 연결해서 올려질 젠킨스 컨테이너 내부에서 호스트OS의 도커 데몬으로 접근할 수 있도록 설정

```
docker run -d -p 9090:8080 -v /var/run/docker.sock:/var/run/docker.sock -v /home/ubuntu/jenkins_home:/var/jenkins_home jenkins/jer
```

- 서버 url의 9090 포트로 접속하면, Jenkins 초기 화면 등장. `sudo cat /var/lib/jenkins/secrets/initialAdminPassword` 를 통해 초기 비밀번호 확인 후 입력
- install suggested plugins 선택하여 초기 플러그인 설치한 후, admin 계정을 생성한다.
- Docker, git, Node, Java, pip, python을 설치

3) 젠킨스와 Gitlab 연동(자동 배포)

- backend 프로젝트 생성
  - 필요한 플러그인을 설치 후, project 생성. branches to build는 merge request 또는 push event가 발생했을 경우 자동 빌드되는 브랜치를 의미. 이 프로젝트에서는 backend 브렌치로 설정했음
  - 무중단배포를 위해 Docker-compose를 사용했으며, 이를 제어하기 쉘스크립트를 사용, 이에 필요한 정보는 다음과 같음
  - `Docker-compose.left.yml`

```
version: "3"
services:
  hoodies_backend:
    build: .
    ports:
      - 8080:8080
```

  - `Docker-compose.right.yml`

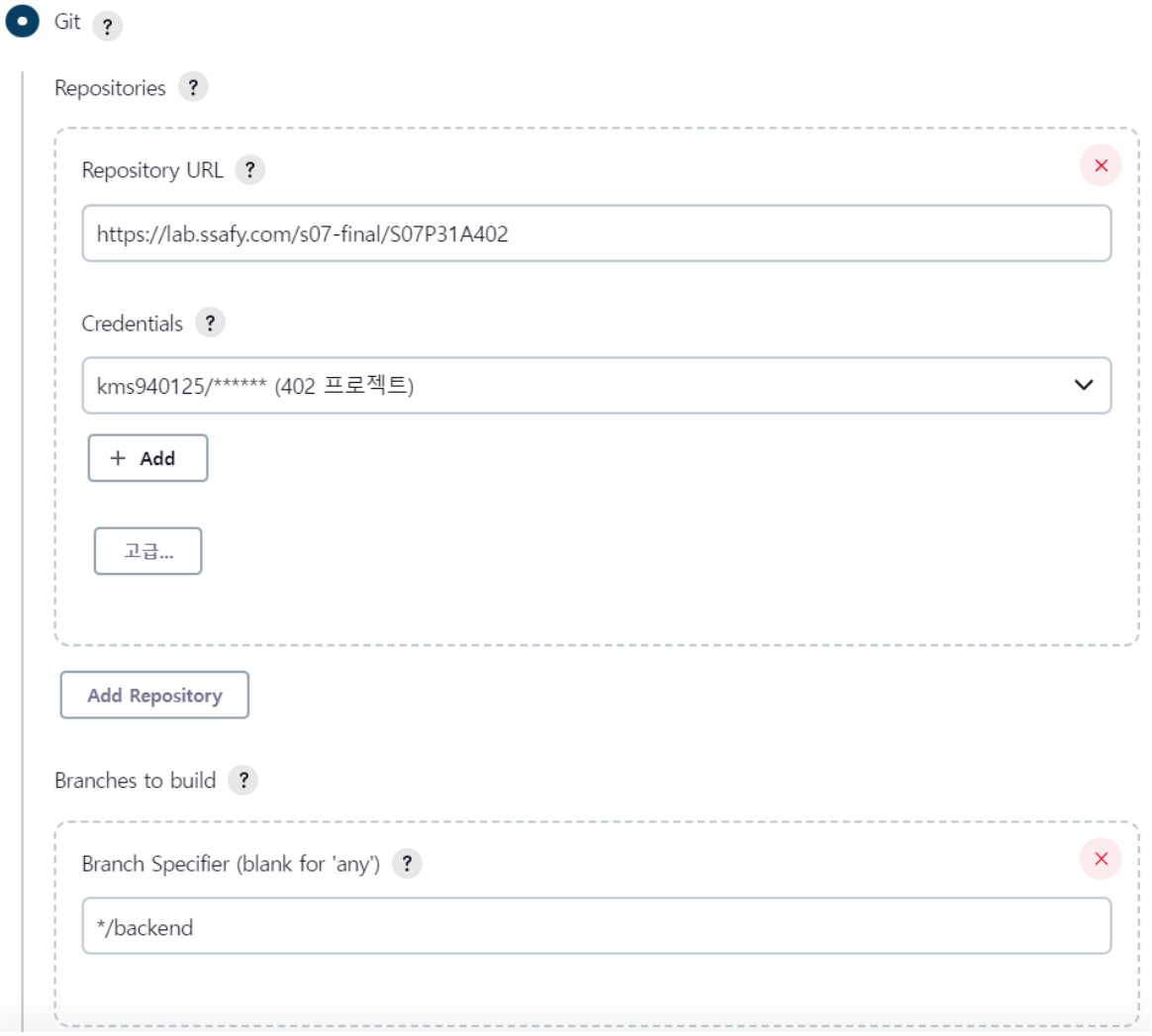
```
version: "3"
services:
  hoodies_backend:
    build: .
    ports:
      - 8082:8080
```

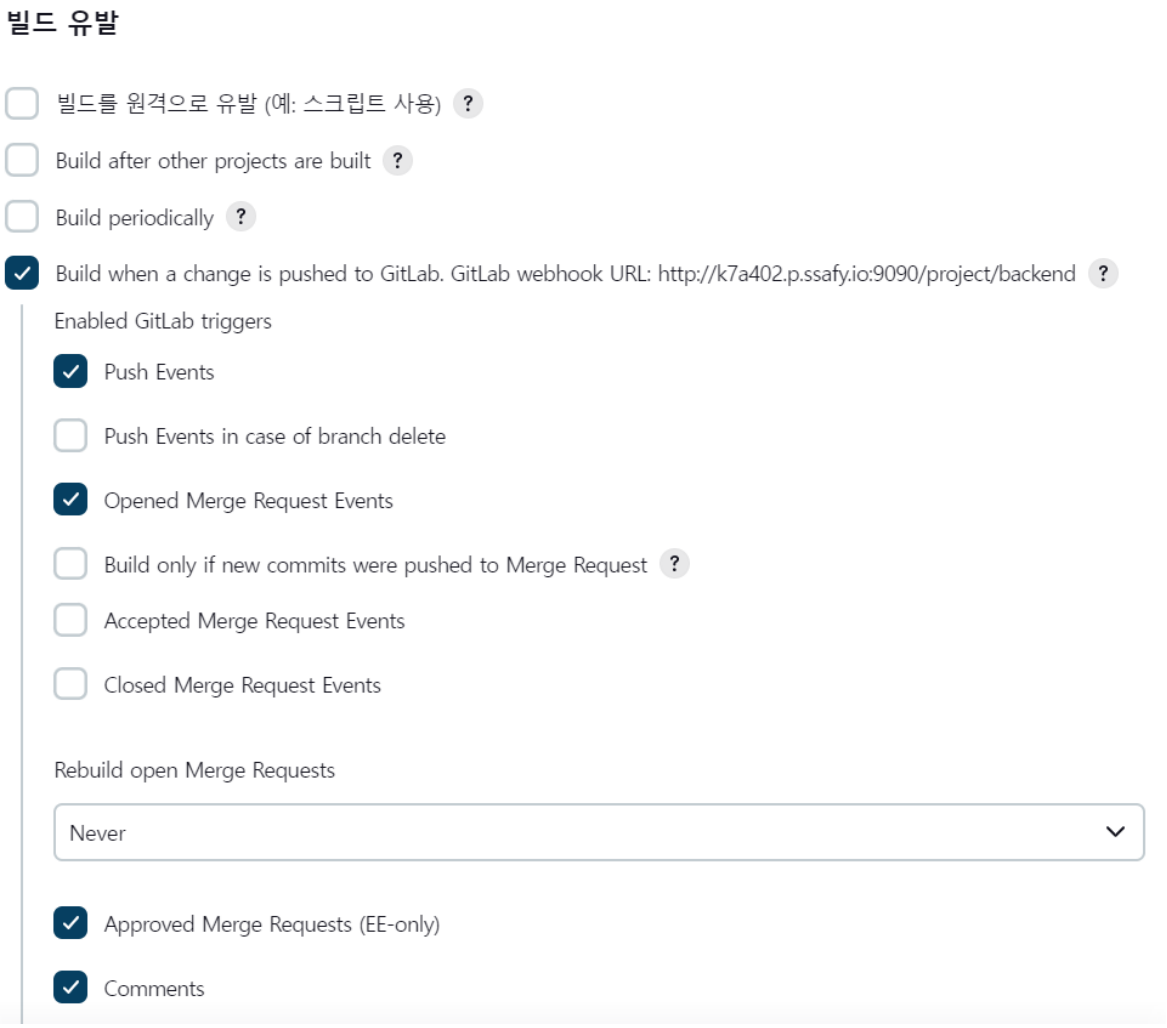
  - `scripts/dev.sh`

```
EXIST_A=$(docker-compose -p blink-a -f docker-compose.left.yml ps | grep blink-left)
if [ -z "${EXIST_A}" ] # -z는 문자열 길이가 0이면 true. A가 실행 중이지 않다는 의미.
then
        # B가 실행 중인 경우
        START_CONTAINER=left
        TERMINATE_CONTAINER=right
        START_PORT=8080
        TERMINATE_PORT=8082
else
        # A가 실행 중인 경우
        START_CONTAINER=right
        TERMINATE_CONTAINER=left
        START_PORT=8082
        TERMINATE_PORT=8080
fi


echo "###################### current container ${TERMINATE_CONTAINER} ########################"

# 실행해야하는 컨테이너 docker-compose로 실행. -p는 docker-compose 프로젝트에 이름을 부여
# -f는 docker-compose파일 경로를 지정
echo "################ blink-${START_CONTAINER} up ###################"
docker-compose -p blink-${START_CONTAINER} -f docker-compose.${START_CONTAINER}.yml up -d --build || exit 1

sleep 5 # 실행되었으면 5초 대기

echo "############### change nginx server port ################"
# sed 명령어를 이용해서 아까 지정해줬던 service-url.inc의 url값을 변경해줍니다.
# sed -i "s/기존문자열/변경할문자열" 파일경로 입니다.
# 종료되는 포트를 새로 시작되는 포트로 값을 변경해줍니다.
# -i.bak 는 백업파일을 만들겠다는 의미입니다.(그래야 변경값이 저장됨)
sed -i.bak "s/${TERMINATE_PORT}/${START_PORT}/" /etc/nginx/conf.d/service-url.inc
echo "################## ${TERMINATE_PORT} down and ${START_PORT} up ##################"

echo "################ nginx reload.. ########################"
service nginx reload
```

```
# 기존에 실행 중이었던 docker-compose는 종료시켜줍니다.
echo "##################### blink-${TERMINATE_CONTAINER} container down ##################"
docker-compose -p blink-${TERMINATE_CONTAINER} -f docker-compose.${TERMINATE_CONTAINER}.yml down

# 도커 쓸대없는 Image제거
docker rmi $(docker images -f "dangling=true" -q)

# # 도커 캐시 제거
# echo y | sudo docker system prune --volumes

echo "################ end of deployment #################"
```

- Execute shell을 다음과 같이 설정






```
cd ${WORKSPACE}/Hoodies
cp -r /var/jenkins_home/resources src/main/
chmod 744 gradlew
./gradlew build
chmod 755 scripts
chmod 755 scripts/dev.sh
./scripts/dev.sh
```

- frontend 프로젝트 생성
  - 이 프로젝트에서는 branches to build를 frontend로 설정
  - 무중단배포를 위해 Docker-compose를 사용했으며, 이를 제어하기 쉘스크립트를 사용, 이에 필요한 정보는 다음과 같음
  - `Docker-compose.blue.yml`

```
version: "3"
services:
  hoodies:
    build: .
    ports:
      - 3001:3000
```

  - `Docker-compose.green.yml`

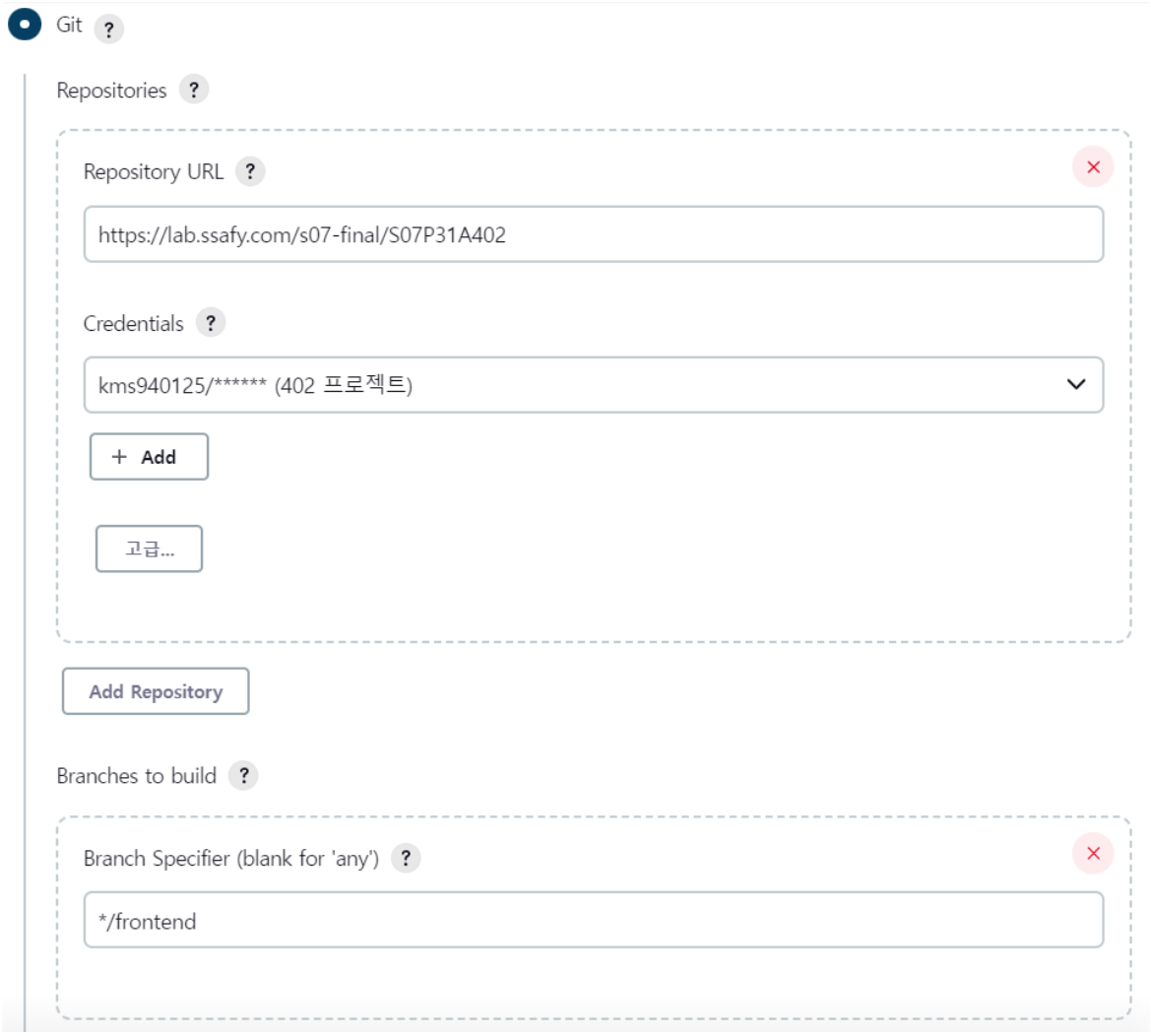
```
version: "3"
services:
  hoodies:
    build: .
```

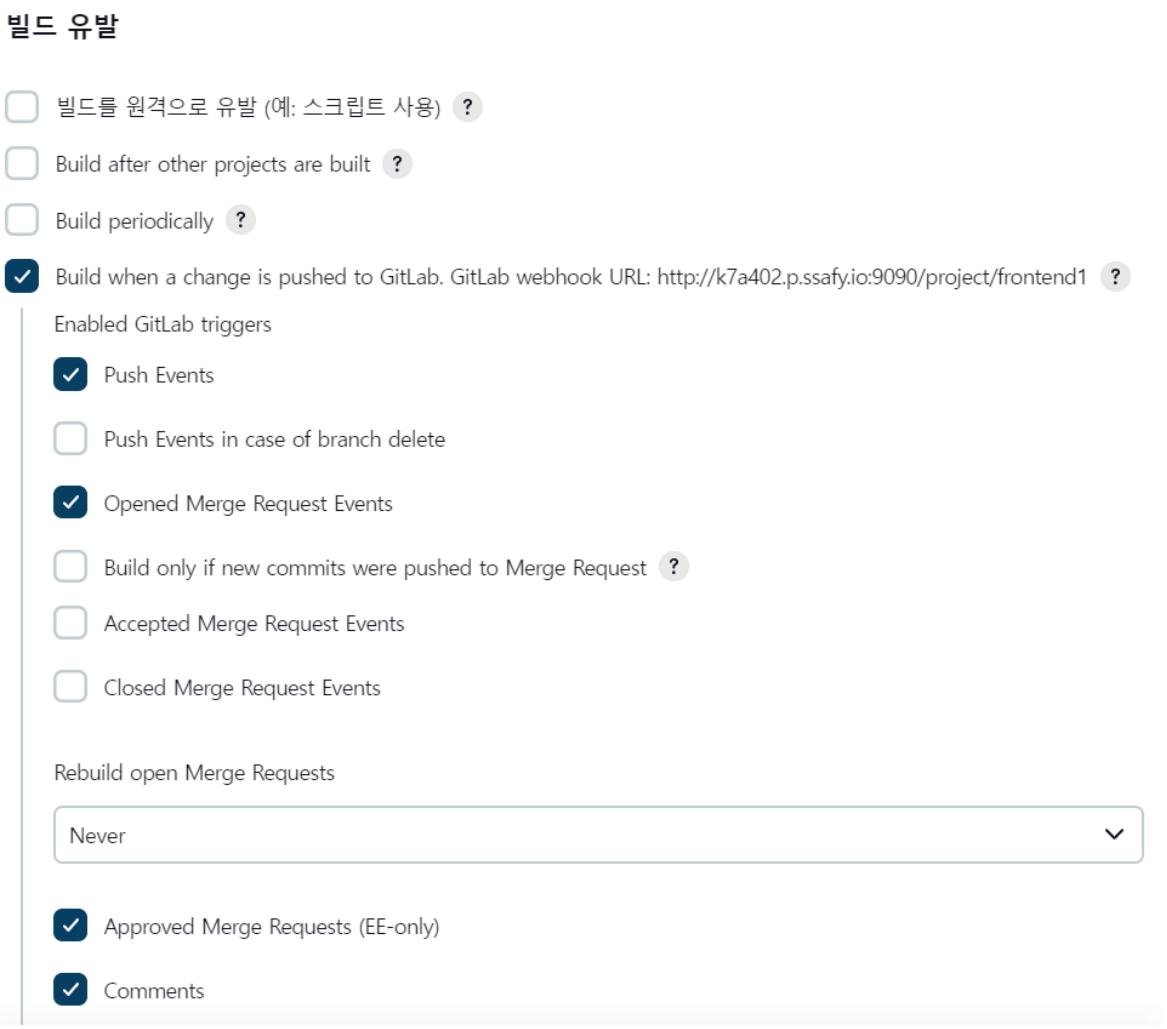
```
    ports:
      - 3000:3000
```

- `scripts/dev.sh`

```
EXIST_A=$(docker-compose -p blink-a -f docker-compose.green.yml ps | grep blink-green)
if [ -z "${EXIST_A}" ] # -z는 문자열 길이가 0이면 true. A가 실행 중이지 않다는 의미.
then
        # B가 실행 중인 경우
        START_CONTAINER=green
        TERMINATE_CONTAINER=blue
        START_PORT=3000
        TERMINATE_PORT=3001
else
        # A가 실행 중인 경우
        START_CONTAINER=blue
        TERMINATE_CONTAINER=green
        START_PORT=3001
        TERMINATE_PORT=3000
fi


echo "###################### current container ${TERMINATE_CONTAINER} ########################"

# 실행해야하는 컨테이너 docker-compose로 실행. -p는 docker-compose 프로젝트에 이름을 부여
# -f는 docker-compose파일 경로를 지정
echo "################ blink-${START_CONTAINER} up ###################"
docker-compose -p blink-${START_CONTAINER} -f docker-compose.${START_CONTAINER}.yml up -d --build || exit 1

sleep 5 # 실행되었으면 5초 대기

echo "############### change nginx server port ################"
# sed 명령어를 이용해서 아까 지정해줬던 service-url.inc의 url값을 변경해줍니다.
# sed -i "s/기존문자열/변경할문자열" 파일경로 입니다.
# 종료되는 포트를 새로 시작되는 포트로 값을 변경해줍니다.
# -i.bak 는 백업파일을 만들겠다는 의미입니다.(그래야 변경값이 저장됨)
sed -i.bak "s/${TERMINATE_PORT}/${START_PORT}/" /etc/nginx/conf.d/service-url.inc
echo "################## ${TERMINATE_PORT} down and ${START_PORT} up ##################"

echo "################ nginx reload.. ########################"
service nginx reload


# 기존에 실행 중이었던 docker-compose는 종료시켜줍니다.
echo "###################### blink-${TERMINATE_CONTAINER} container down ##################"
docker-compose -p blink-${TERMINATE_CONTAINER} -f docker-compose.${TERMINATE_CONTAINER}.yml down

# 도커 쓸대없는 Image제거
docker rmi $(docker images -f "dangling=true" -q)

# # 도커 캐시 제거
# echo y | sudo docker system prune --volumes

echo "################ end of deployment #################"
```

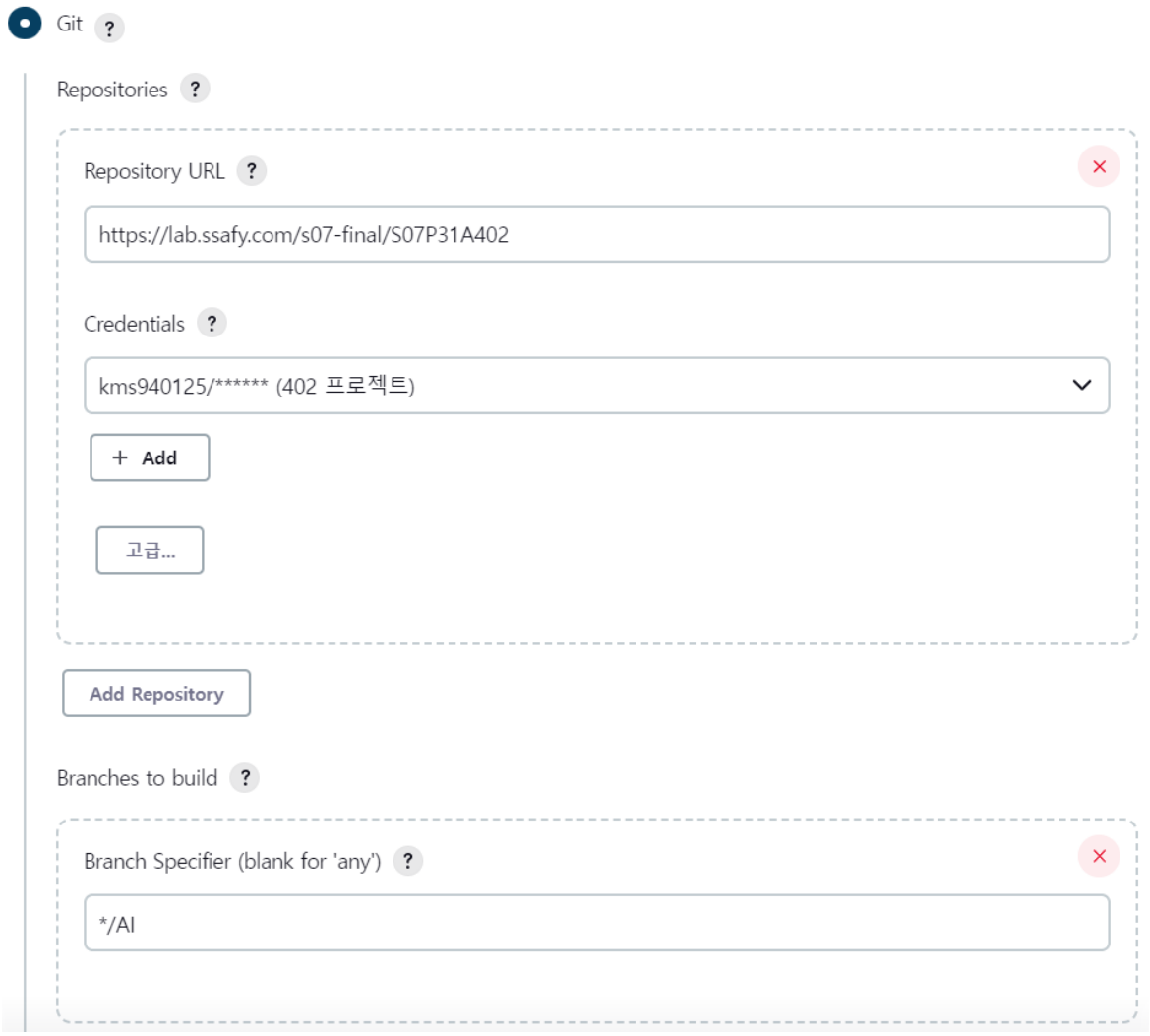
- Execute Shell을 다음과 같이 설정

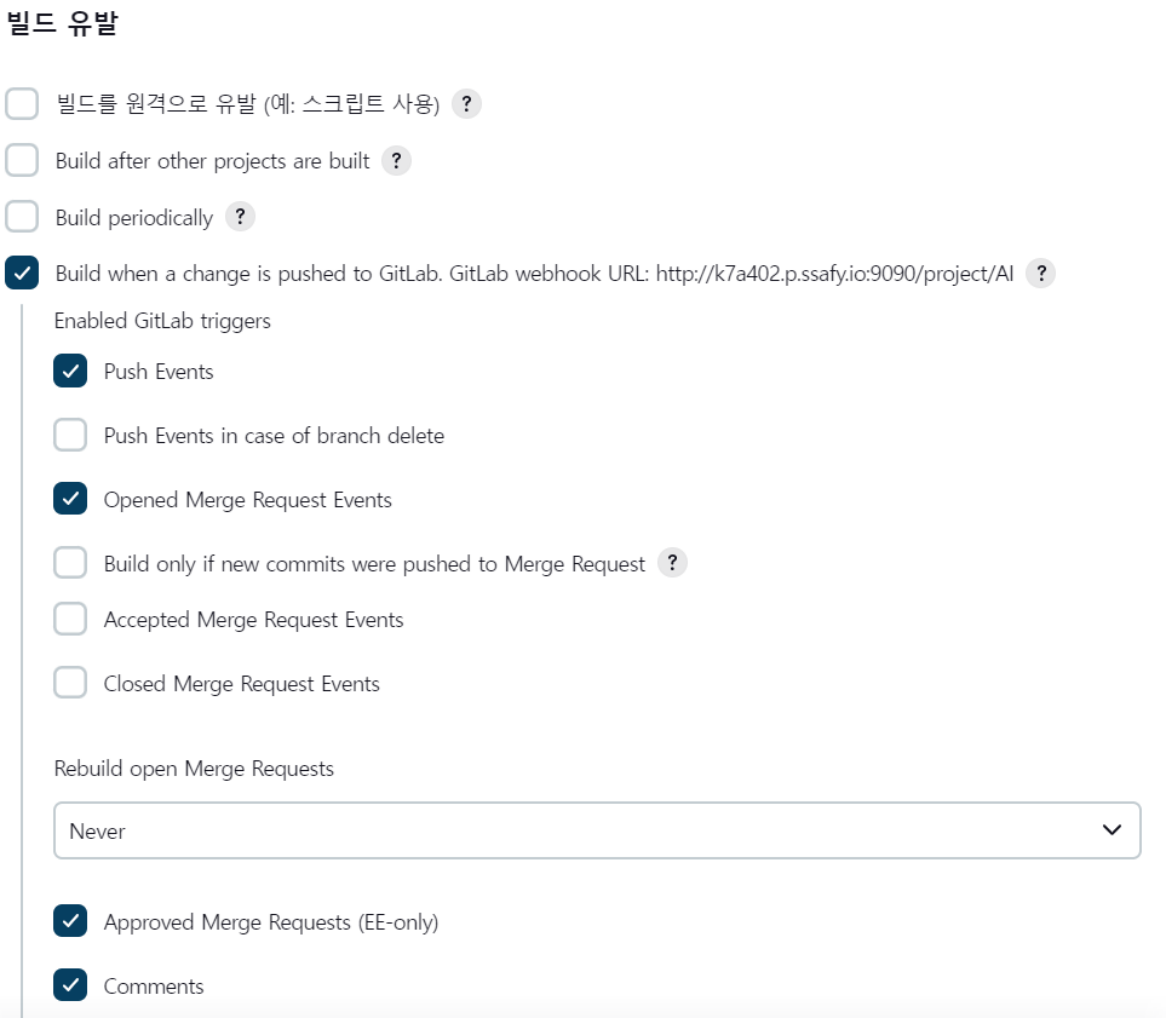

Git
Repositories
Repository URL
https://lab.ssafy.com/s07-final/S07P31A402
Credentials
kms940125/****** (402 프로젝트)
+ Add
고급...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/frontend




```
cd ${WORKSPACE}/frontend/hoodies
chmod 755 scripts
chmod 755 scripts/dev.sh
./scripts/dev.sh
```

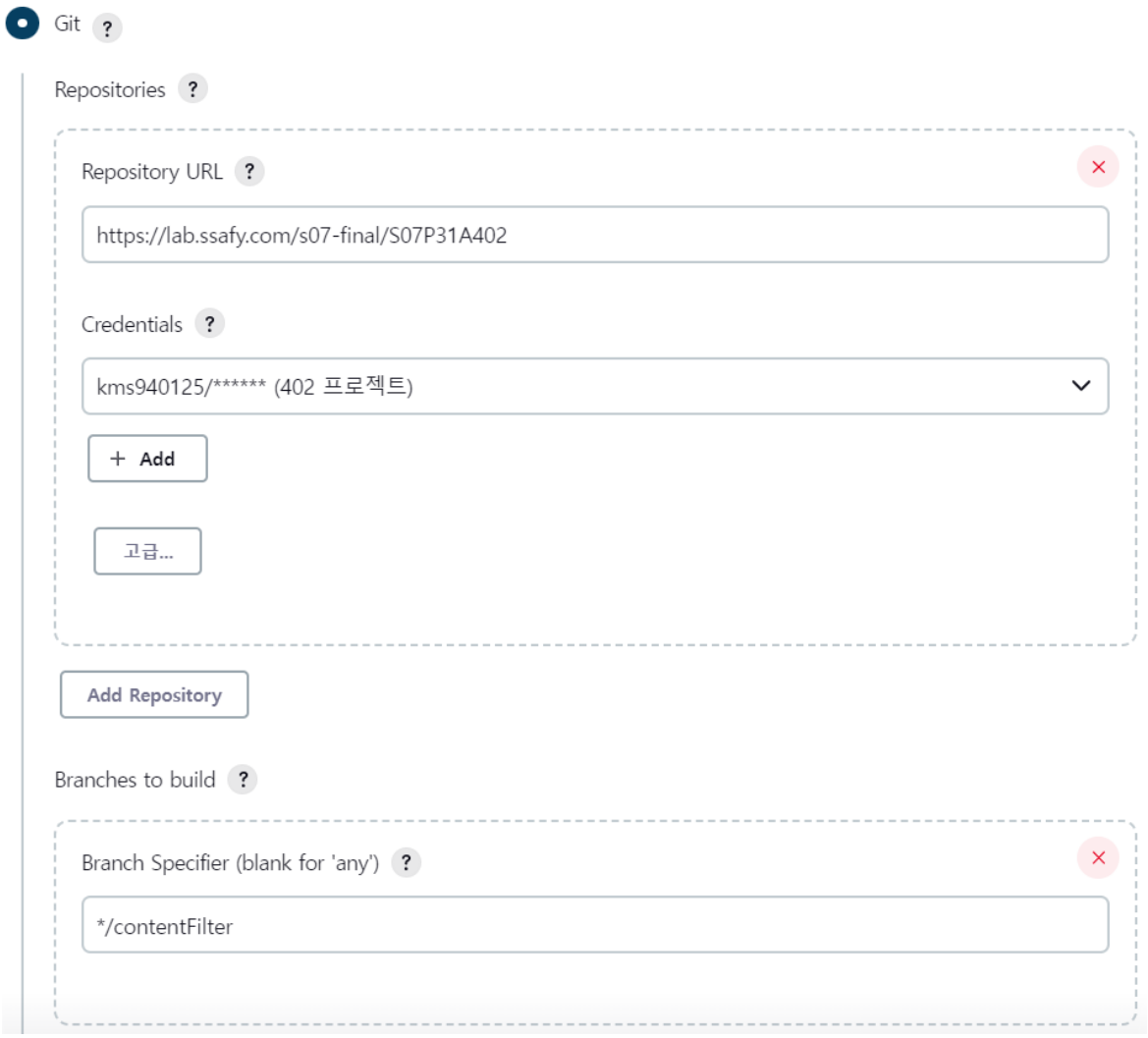
- AI 프로젝트(text filtering server) 생성
  - 이 프로젝트에서는 branches to build를 frontend로 설정
  - 자동 배포를 위해 jenkins를 사용
  - Execute Shell을 다음과 같이 설정

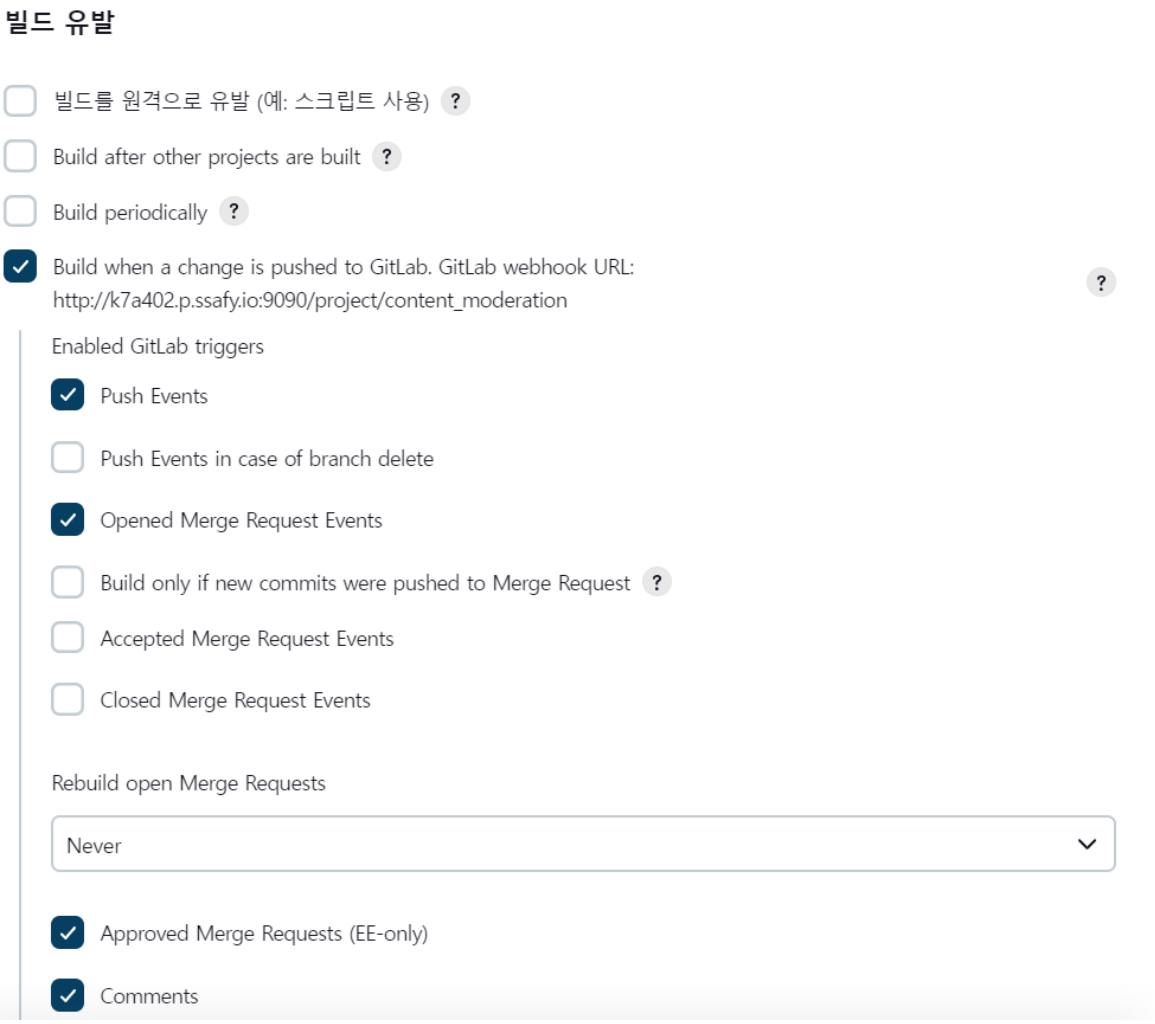

Git
Repositories
Repository URL
https://lab.ssafy.com/s07-final/S07P31A402
Credentials
kms940125/****** (402 프로젝트)
+ Add
고급...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/AI




```
cd ${WORKSPACE}/AI/HateSpeechDetector

docker ps -q --filter name=hoodies_ai_container | grep -q . && docker stop hoodies_ai_container && docker rm hoodies_ai_container

docker ps -a -q --filter name=hoodies_ai_container | grep -q . && docker rm hoodies_ai_container

docker build -t hoodies_ai_image .
docker run -d -p 8081:8081 --name hoodies_ai_container hoodies_ai_image
```

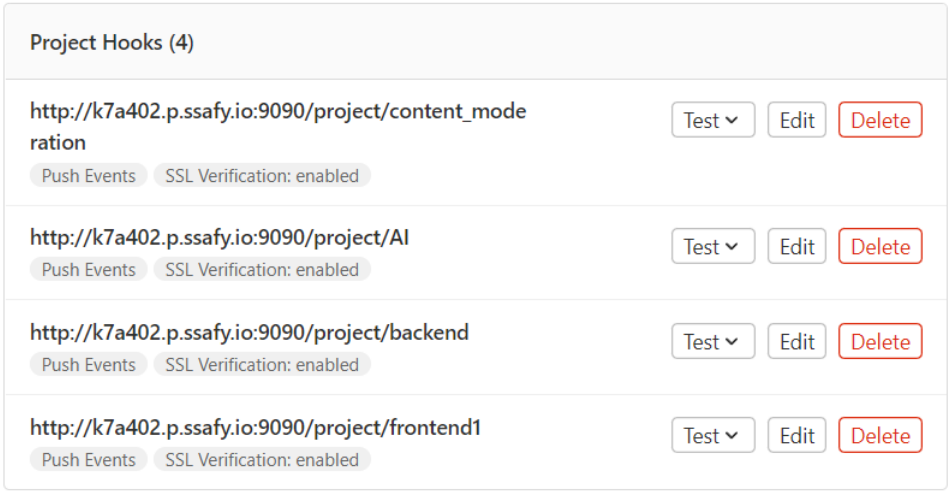
- content_moderation 프로젝트(image filtering server) 생성
  - 이 프로젝트에서는 branches to build를 contentFilter로 설정
  - 자동 배포를 위해 jenkins를 사용
  - Execute Shell을 다음과 같이 설정


Git
Repositories
Repository URL
https://lab.ssafy.com/s07-final/S07P31A402
Credentials
kms940125/****** (402 프로젝트)
+ Add
고급...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/contentFilter




```
cd ${WORKSPACE}/contentFilter
cp /var/jenkins_home/winter-alliance-368105-8cf0b5f21e09.json .

docker ps -q --filter name=hoodies_filter_container | grep -q . && docker stop hoodies_filter_container && docker rm hoodies_filte

docker ps -a -q --filter name=hoodies_filter_container | grep -q . && docker rm hoodies_filter_container

docker build -t hoodies_filter_image .
docker run -d -p 8083:8083 --name hoodies_filter_container hoodies_filter_image
```

- webhook 설정



3 - 1 frontend 수동 배포

가) 클론받은 프로젝트 폴더에서 프론트엔드 폴더에 있는 Hoodies 폴더로 이동합니다.

나) 도커 이미지 생성에 필요한 Dockerfile은 다음과 같습니다.

```
# DockerFile
FROM node:16

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

ENV PATH /usr/src/app/node_modules/.bin:$PATH

COPY package*.json ./

RUN npm install --force

COPY ./ ./

EXPOSE 3000

CMD ["npm", "run", "start"]
```

다) 도커 이미지를 생성한 후, 도커 컨테이너를 통해 프론트엔드를 배포합니다.

```
# 저장소 클론
git clone https://lab.ssafy.com/s07-final/S07P31A402.git

# 프론트엔드 폴더로 이동
cd frontend/hoodies

# 도커 컨테이너에 있는 기존 도커 이미지 stop
docker ps -q --filter name=hoodies_react_container | grep -q . && docker stop hoodies_react_container && docker rm hoodies_react_c

# 도커 컨데이너에 있는 기존 도커 이미지 삭제
docker ps -a -q --filter name=hoodies_react_container | grep -q . && docker rm hoodies_react_container

# 도커 이미지 생성
docker build -t hoodies_react_image .

# 도커 이미지 실행
docker run -d -p 3000:3000 --name hoodies_react_container hoodies_react_image
```

3 - 2 Backend 수동 배포

Dockerfile 내용 확인

가) 클론받은 프로젝트 폴더에서 Hoodies 폴더로 이동합니다.

나) 도커 이미지를 생성하기 위한 도커파일은 다음과 같습니다.

```
FROM openjdk:8-jdk

WORKDIR .

COPY build/libs/Hoodies-0.0.1-SNAPSHOT.jar application.jar

EXPOSE 8080

CMD ["java", "-jar", "application.jar"]
```

다)도커 이미지를 생성한 후, 도커 컨테이너를 통해 백엔드를 배포합니다,

```
git clone https://lab.ssafy.com/s07-final/S07P31A402.git

cd hoodies

cp -r /var/jenkins_home/resources src/main/

docker ps -q --filter name=hoodies_spring_container | grep -q . && docker stop hoodies_spring_container && docker rm hoodies_spring

docker ps -a -q --filter name=hoodies_spring_container | grep -q . && docker rm hoodies_spring_container

docker build -t hoodies_spring_image .
docker run -d -p 8080:8080 --name hoodies_spring_container hoodies_spring_image
```

3 - 3 Text Filtering Server 수동 배포

가) 클론 받은 후, ai 디렉터리 안에 있는 HateSpeechDetector 폴더로 이동

나) 도커 컨테이너를 통한 실행

```
docker ps -q --filter name=hoodies_ai_container | grep -q . && docker stop hoodies_ai_container && docker rm hoodies_ai_container

docker ps -a -q --filter name=hoodies_ai_container | grep -q . && docker rm hoodies_ai_container

docker build -t hoodies_ai_image .
docker run -d -p 8081:8081 --name hoodies_ai_container hoodies_ai_image
```

3 - 4 Image Filtering Server 수동 배포

가)클론 받은 후, contentFilter 폴더로 이동

나)도커 컨테이너를 통한 실행

```
cp /var/jenkins_home/winter-alliance-368105-8cf0b5f21e09.json .
# google vision API key file을 복사

docker ps -q --filter name=hoodies_filter_container | grep -q . && docker stop hoodies_filter_container && docker rm hoodies_filter

docker ps -a -q --filter name=hoodies_filter_container | grep -q . && docker rm hoodies_filter_container

docker build -t hoodies_filter_image .
docker run -d -p 8083:8083 --name hoodies_filter_container hoodies_filter_image
백그라운드 실행
```

---

3 - 5 SSL 인증서 적용

- SSL 인증서를 적용하기 위해 다음과 같은 명령어를 입력합니다.

```
sudo apt-get install certbot python3-certbot-nginx

sudo service stop nginx

sudo certbot certonly --standalone -d k7a402.p.ssafy.io
```

- 이 경우 /etc/letsencrypt/live/k7a402.p.ssafy.iol에 ssl 인증서가 설치됩니다.
- /etc/nginx/sites-available로 이동, 아래와 같은 파일을 생성합니다.

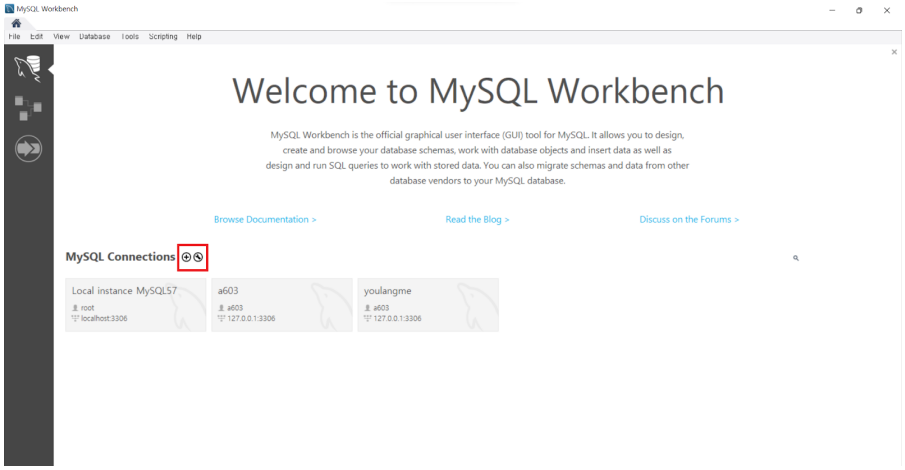
```
server {
         location /{
            proxy_pass http://localhost:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Origin "";

            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
          }

          location /api/ {
            proxy_pass http://localhost:8080/;
         }

         location /ai{
            proxy_pass http://localhost:8081/;
         }
         location /cm{
            proxy_pass http://localhost:8083/;
         }




         listen 443 ssl;
         server_name k7a402.p.ssafy.io;
                ssl_certificate /etc/letsencrypt/live/k7a402.p.ssafy.io/fullchain.pem; # managed by Certbot
                ssl_certificate_key /etc/letsencrypt/live/k7a402.p.ssafy.io/privkey.pem; # managed by Certbot
}
server {
    if ($host = k7a402.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
        server_name k7a402.p.ssafy.io;
    return 404; # managed by Certbot
 }
```

- 80 port로 접근할 경우 ssl인증서가 적용된 443 port 로 리다이렉트됩니다.
- 또한 443 port의 /api/로 접근 시에는 localhost:8080로 분기 처리됩니다.
- 한편 443 port의 /ai로 접근 시에는 localhost:8081/ai로 분기 처리됩니다.
- 마지막으로 443 port의 /cm으로 접근 시에는 localhost:8081/cm으로 분기 처리됩니다.
- 이후 다음 명령어를 입력하면 ssl 인증서가 적용됩니다

```
sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-enabled/[파일명]
# 필자의 경우 mafya.conf

sudo nginx -t

sudo service restart nginx
```

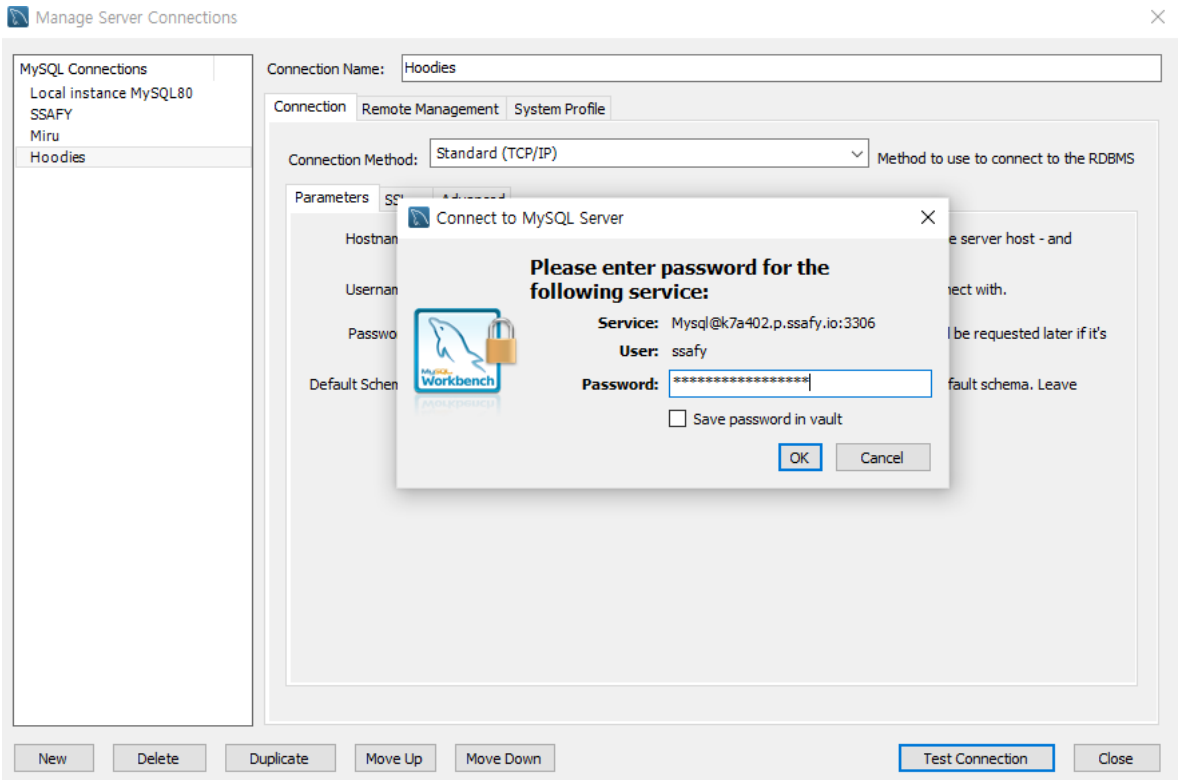
4. DB 계정

가. MySql workbench 추가하기

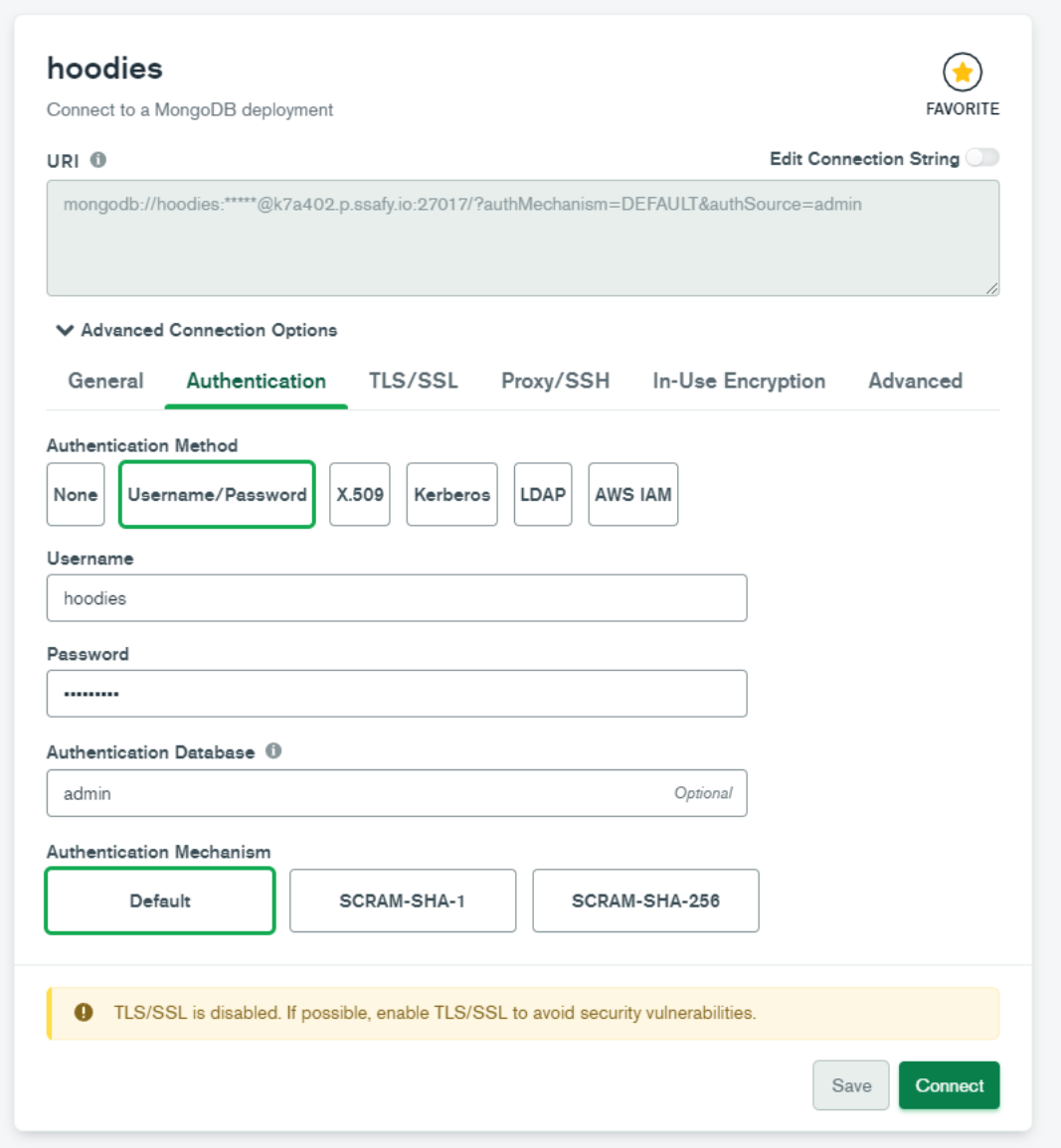


MySql Workbench를 열어서 새로운 내용을 추가하기 위해 “+” 버튼을 눌러줍니다.

나. EC2 계정 정보 설정



다. MongoDB 사용하기



MongoDB의 username은 hoodies, password는 a402!1201를 사용하였습니다.

MySQL의 username은 ssafy, password는 a402!1201를 사용하였습니다.

기존 root 계정이 아닌 별도의 계정을 생성하여 프로젝트를 진행하였습니다.

5. 프로퍼티 정의

가) nginx 세팅

- Docker 사용 시에는 /etc/nginx/sites-available로 이동한 후 아래와 같은 파일을 생성합니다.

```
server {
        location /{
          proxy_pass http://localhost:3000;
          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection "upgrade";
          proxy_set_header Origin "";

          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header Host $http_host;
        }

        location /api/ {
```

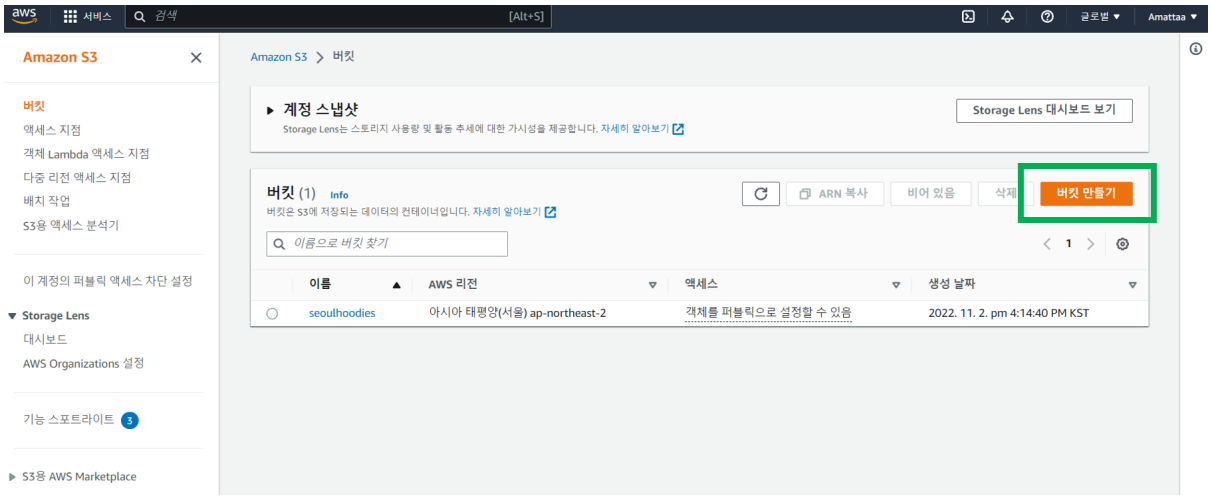
```
            proxy_pass http://localhost:8080/;
        }

        location /ai{
            proxy_pass http://localhost:8081/;
        }
        location /cm{
            proxy_pass http://localhost:8083/;
        }



        listen 443 ssl;
        server_name k7a402.p.ssafy.io;
              ssl_certificate /etc/letsencrypt/live/k7a402.p.ssafy.io/fullchain.pem; # managed by Certbot
              ssl_certificate_key /etc/letsencrypt/live/k7a402.p.ssafy.io/privkey.pem; # managed by Certbot
}
server {
    if ($host = k7a402.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
        server_name k7a402.p.ssafy.io;
    return 404; # managed by Certbot
 }
```

- 다음 명령어를 입력합니다.

```
sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-enabled/[파일명]
# 필자의 경우 mafya.conf

sudo nginx -t

sudo service restart nginx
```

6. 외부 서비스

가. S3 설정



- S3 버킷을 생성 후, 권한 탭에서 CORS 설정하고 ACL 비활성화 설정
- Spring /src/main/resource/application.yml에 s3 설정 추가

나. 스마일게이트의 Korea unsmaile dataset 적용

Korean UnSmile Dataset

UnSmile

Smilegate AI

Smilegate AI에서 공개하는 한국어 혐오표현 "☹ UnSmile" 데이터셋입니다.

1. 데이터셋 overview

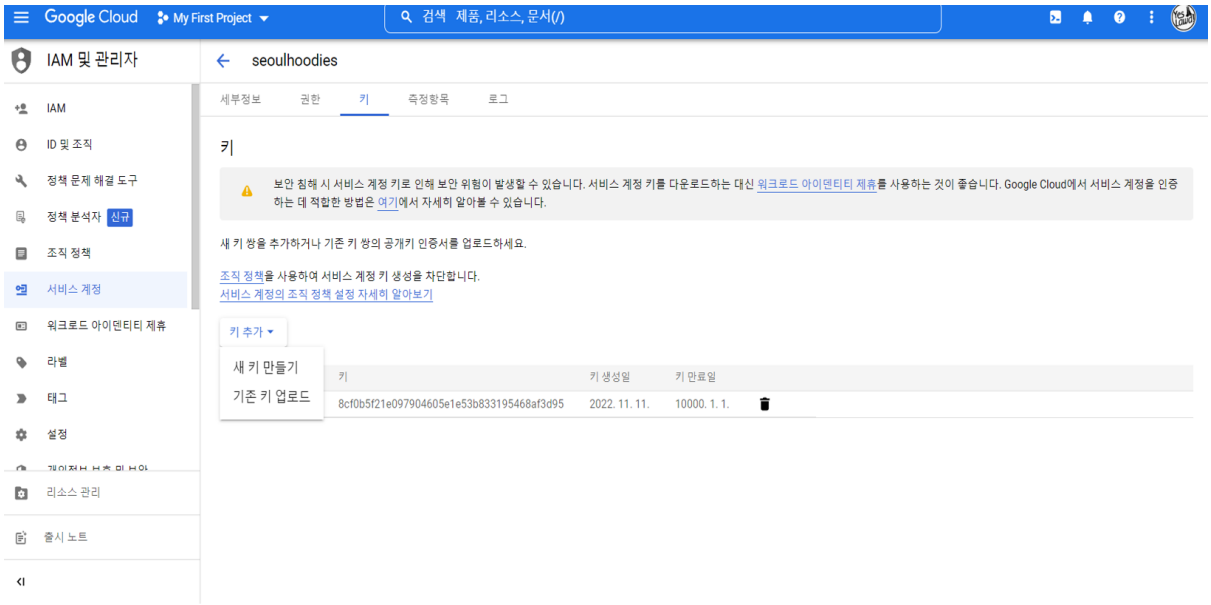
본 데이터셋에서의 혐오 표현은 "특정 사회적 (소수자) 집단에 대한 적대적 발언, 조롱, 희화화, 편견을 재생산하는 표현"으로 정의합니다.

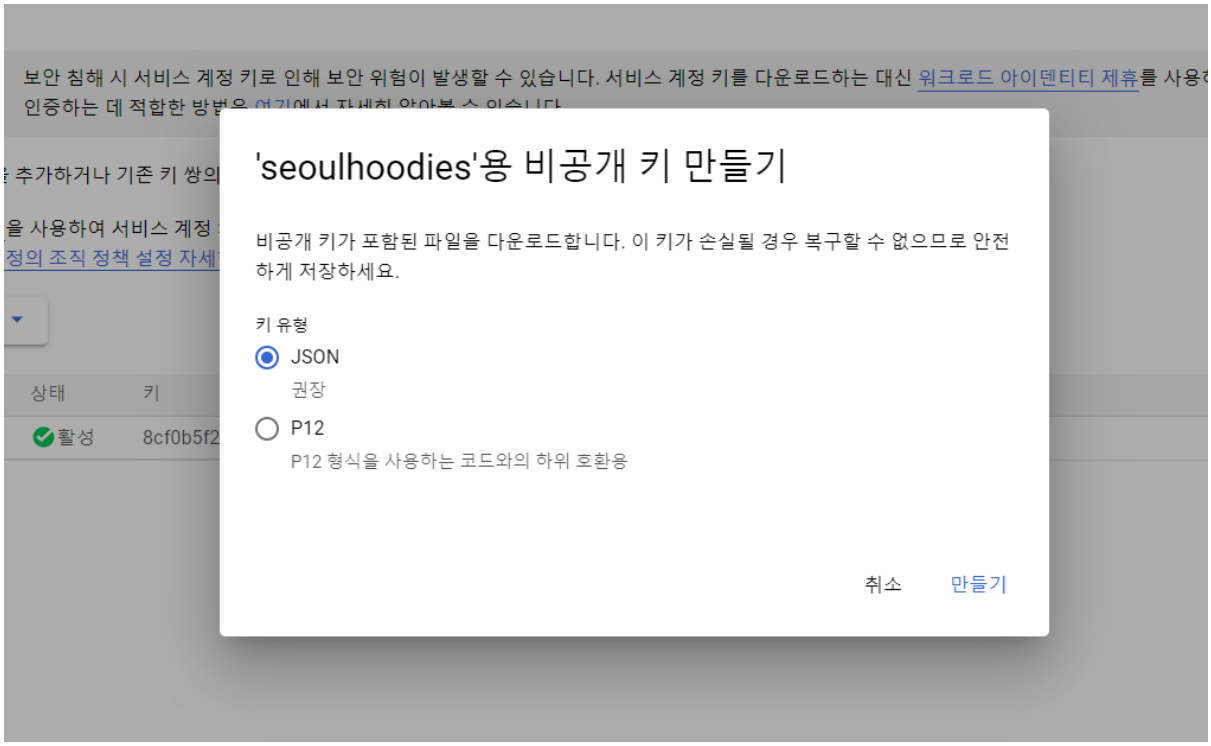
- **혐오의 대상이 속한 집단**을 명확히 지칭하는 비하.차별발언
- 대상에 대한 고정관념 (e.g. 동양인은 수학 잘하지 않아?, 역시 흑형이라 운동을 잘하네)
- 대상의 특성이나 성향을 특정한 통념에 고착시키는 발언 (e.g. 여자는 집에서 애나 봐야지!, 임신은 축복이지!, 게이는 잘생겨야지)
- 화자 스스로를 자조적으로 표현하는 경우는 혐오 발언이 아님 (e.g. 아 내가 급식충이다!)

단일 데이터는 [혐오표현, 악플/욕설, clean]으로 분류될 수 있으며, 혐오 표현은 다중 레이블(multi-label)로 전문가 집단을 통해 레이블링되었습니다.

- https://github.com/smilegate-ai/korean_unsmile_dataset를 참조
- 위 주소의 쥬피터 노트북으로 이동, requirement를 파악한 후 ec2 설정(GPU없음)에 맞춰 도입

다. Google Vision API(safe search)

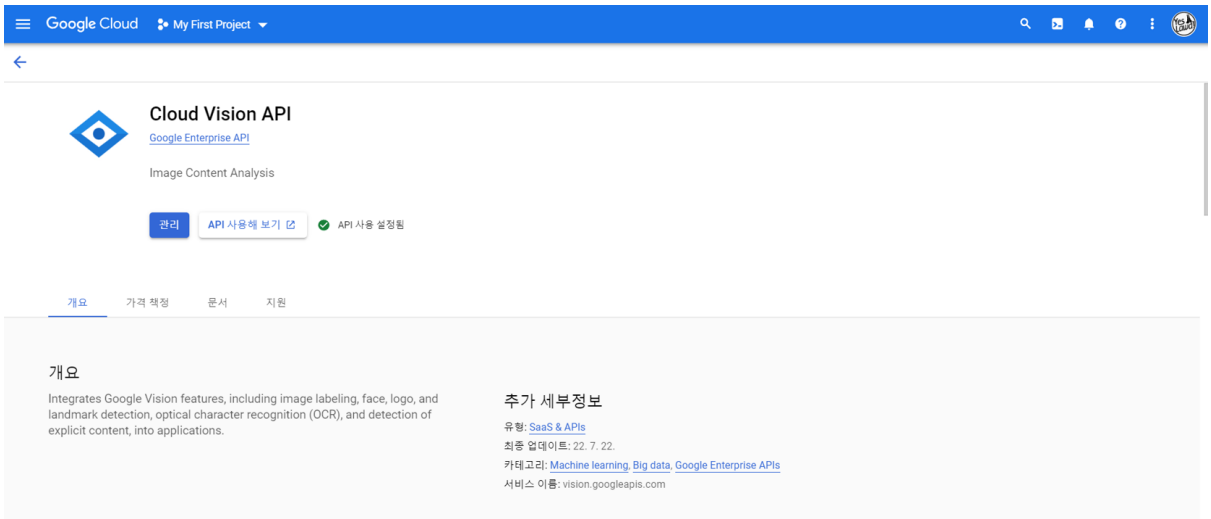




- 키 발급 후 서버 경로인 /home/ubuntu/jenkins_home에 저장
- 빌드 시 위의 경로에 있는 key 파일을 복사 후 사용

```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'winter-alliance-368105-8cf0b5f21e09.json'
```

- 이미지 필터링 플라스크 서버에 다음 코드를 추가 후, key 문제 해결



- Google vision api를 검색한 후, 이를 적용함.

라. MatterMost API 적용

- mm 메시지를 보낼 사용자 계정에 로그인
- mm 메시지를 받을 사용자의 아이디 입력
- 보낼 사용자와 받는 사용자와의 다이렉트 채널 주소 획득

- 해당 채널을 통해 메시지 전송
- mm 메시지를 보낼 사용자 계정 정보는 Spring /src/main/resource/application.yml에 추가