PLANEAT 포팅메뉴얼

I. 빌드 및 배포

1. 개발 환경

Server : AWS EC2 Ubuntu 20.04 LTS
Visual Studio Code : 1.70.1
IntelliJ IDEA : 2022.2(Ultimate Edition) 17.0.3+7-b469.32 amd64
JDK : openjdk-8
Docker : 20.10.18
Node.js : 18.9.1
MySQL : 8.0.30-1.el8
Nginx : 1.23.1
Jenkins : 2.361.1

2. 설정 파일 목록

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/src/main/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 요청 URL : ex) 서비스 도메인/api}

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_JAVASCRIPT_KEY={카카오 자바스크립트 KEY}
REACT_APP_KAKAO_REST_API_KEY={카카오 REST API KEY}
REACT_APP_KAKAO_CLIENT_ID={카카오 클라이언트 ID}
REACT_APP_KAKAO_CLIENT_SECRET={카카오 클라이언트 SECRET}
REACT_APP_KAKAO_REDIRECT_URI={카카오 리다이렉트 URI}
```

- Dockerfile

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

Spring

- application.properties

```
server.port=9000

spring.datasource.url=jdbc:mysql://{DB컨테이너이름}:3306/{SCHEME이름}?serverTimezone=Asia/Seoul&useUniCode=yes&characterEncoding=UTF-8&
zeroDateTimeBehavior=convertToNull
spring.datasource.username={사용자계정ID}
spring.datasource.password={사용자계정PASSWORD}

spring.jpa.show-sql=true

spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=none

spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.profiles.include=oauth

logging.level.root=info

spring.jackson.serialization.fail-on-empty-beans=false

# context path
server.servlet.context-path=/api

# AWS S3 IAM user
cloud.aws.credentials.accessKey={AWS ACCESS KEY}
cloud.aws.credentials.secretKey={AWS SECRET KEY}
# AWS S3 bucket
```

```
cloud.aws.s3.bucket={버킷 이름}
cloud.aws.region.static=ap-northeast-2
cloud.aws.stack.auto=false

# swagger path
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

- application-oauth.properties

```
# GOOGLE LOGIN
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


# KAKAO LOGIN
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_nickname, account_email
spring.security.oauth2.client.registration.kakao.client-name=Kakao

# KAKAO PROVIDER
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


# NAVER LOGIN
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

# NAVER PROVIDER
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
```

- Dockerfile

```
FROM openjdk:8-jdk
ARGJAR_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 9000:9000 --name springbootapp --network ubuntu_default -d springbootapp
```

Docker

- docker-compose.yml

```
version: "3"
services:
  mysql:
    image: mysql
    container_name: mysql
    environment:
      MYSQL_DATABASE: {SCHEME 이름}
      MYSQL_ROOT_PASSWORD: {ROOT 계정 PASSWORD}
    volumes:
      - /mysql:/var/lib/mysql
    ports:
      - 3306:3306
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt
      - ./nginx/conf.d:/etc/nginx/conf.d

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    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 {서비스 도메인} www.{서비스 도메인};
    return 301 https://$server_name$request_uri;
}
server {
    listen 443 ssl;
    server_name {서비스 도메인};
    access_log off;

    ssl_certificate
    /etc/letsencrypt/live/{서비스 도메인}/fullchain.pem;
    ssl_certificate_key
    /etc/letsencrypt/live/{서비스 도메인}/privkey.pem;

    location / {
        proxy_pass http://{서비스 도메인}: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://{서비스 도메인}:9000/api/;
        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 설치

```
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) signedby=/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 dockercompose-plugin
```

- Docker Compose 설치

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/dockercompose-$(uname -s)-$(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 서비스 도메인}
```

이메일 작성 후 Agree

뉴스레터 수신 여부 No

```
ssl_certificate /etc/letsencrypt/live/{서비스 도메인}/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/{서비스 도메인}/privkey.pem;
```

6. DB 및 Infra 배포

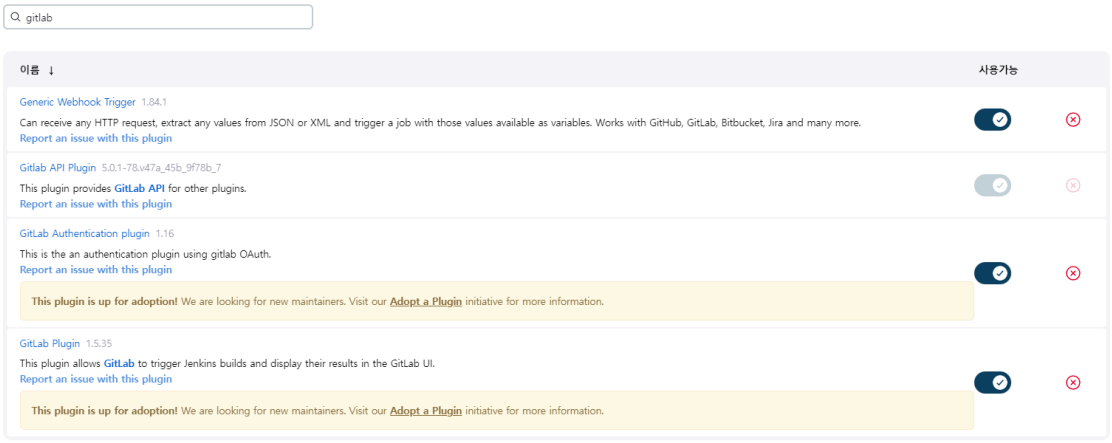
1. docker-compose 작성 (docker-compose.yml)
   위치 : /home/ubuntu
2. docker-compose 실행

   /home/ubuntu (docker-compose.yml 있는 경로에서)

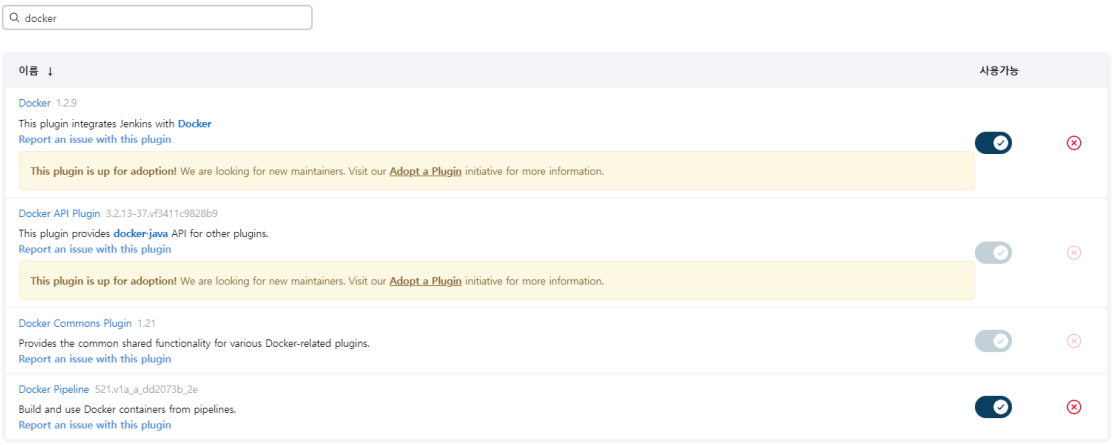
   ```
   sudo docker-compose up -build -d
   ```

3. Nginx 설정 (app.conf)
   위치 : /home/ubuntu/nginx/conf.d
4. Jenkins 설정
   i. http://{서비스 도메인}:9090 으로 접속하여 jenkins 페이지 진입
   ii. 서버 콘솔에서 sudo docker logs {Jenkins 컨테이너 이름}으로
   Administrator password를 확인하고 입력
   iii. Install suggested plugins를 선택하여 권장 플러그인 설치
   iv. 생성할 관리자 계정 정보를 입력하고 Save and Continue
   v. Jenkins 접속 URL 확인 후 Save and Finish

vi. 메인 화면에서 DashBoard > Manager Jenkins > Plugin Manager
gitlab 검색 후 스크린샷과 같이 설치



docker 검색 후 스크린샷과 같이 설치



vii. 서버 콘솔에서 Jenkins 내부에 docker를 설치

```
$ sudo docker exec -it {jenkins 컨테이너명} /bin/bash
$ apt-get update -y
$ apt-get install -y
$ apt-get install docker.io -y
$ docker –v
```

7. CI/CD

BackEnd - 설정파일

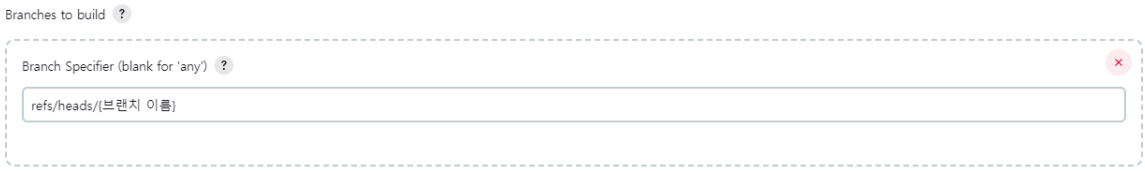
- application.properties 작성 (application.properties)
- application-oauth.properties 작성 (application-oauth.properties)
- Dockerfile 작성 (Dockerfile)

BackEnd - Jenkins 설정

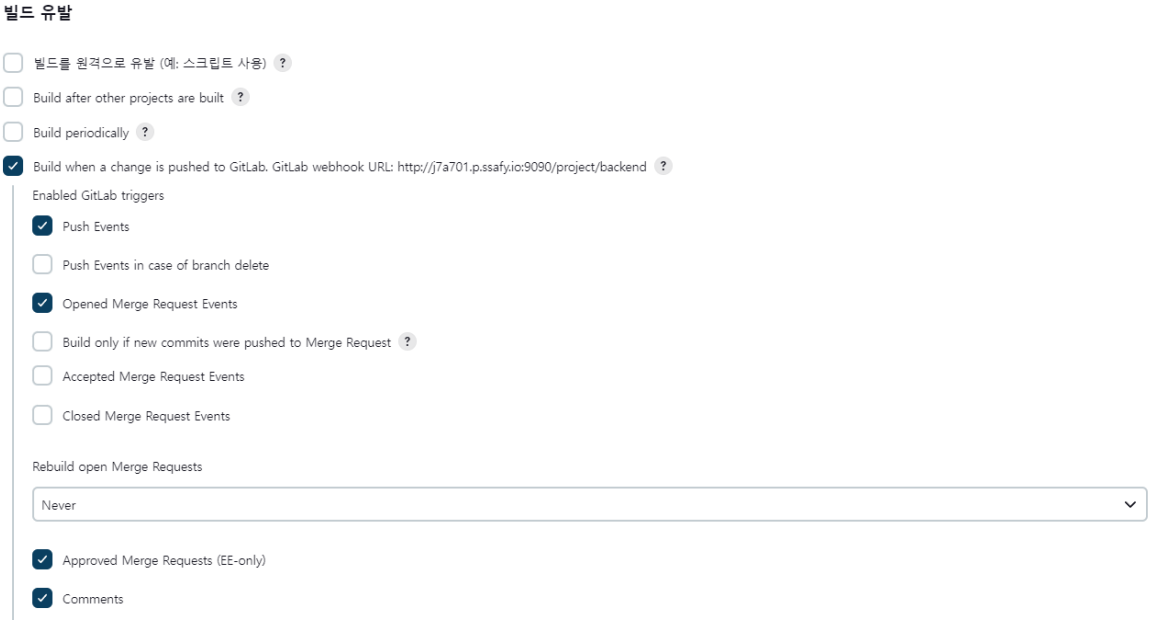
1. Jenkins 메인 화면 > Dashboard > 새로운 Item
2. FreeStyle project 선택하고 item name은 backend로 설정 후 OK
3. Repositories URL에는 프로젝트 레포지토리의 HTTPS 주소 입력
    - Credentials > Add 클릭

- Domain > Global credentials
- Kind > Username with password
- Username > 레포지토리 접근 권한이 있는 GitLab 계정 아이디
- Password > 레포지토리 접근 권한이 있는 GitLab 계정 비밀번호

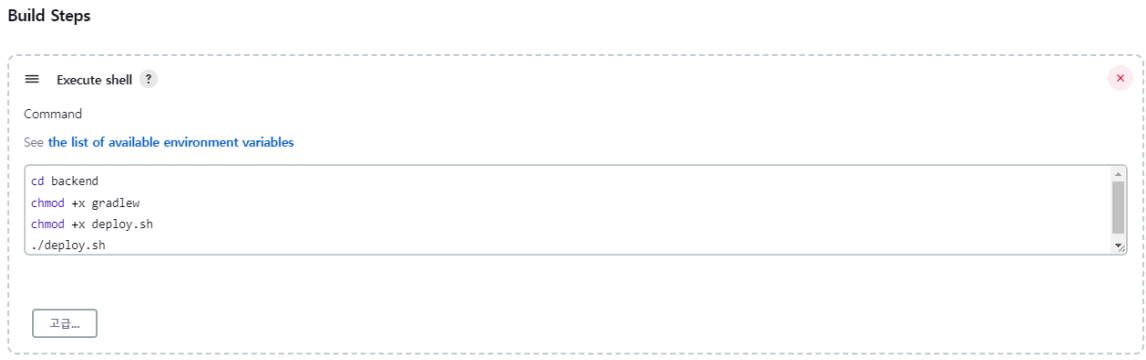
4. 생성된 Credential을 선택
5. 스크린샷과 같이 설정



{브랜치 이름} 브랜치의 내용만을 받아와서 빌드



Push, Merge Request가 발생할 때마다 빌드 유발



backend 폴더로 이동하여 gradlew 와 deploy.sh 파일에 실행할 수 있는 권한 부여

```
cd backend
chmod +x gradlew
chmod +x deploy.sh
./deploy.sh
```

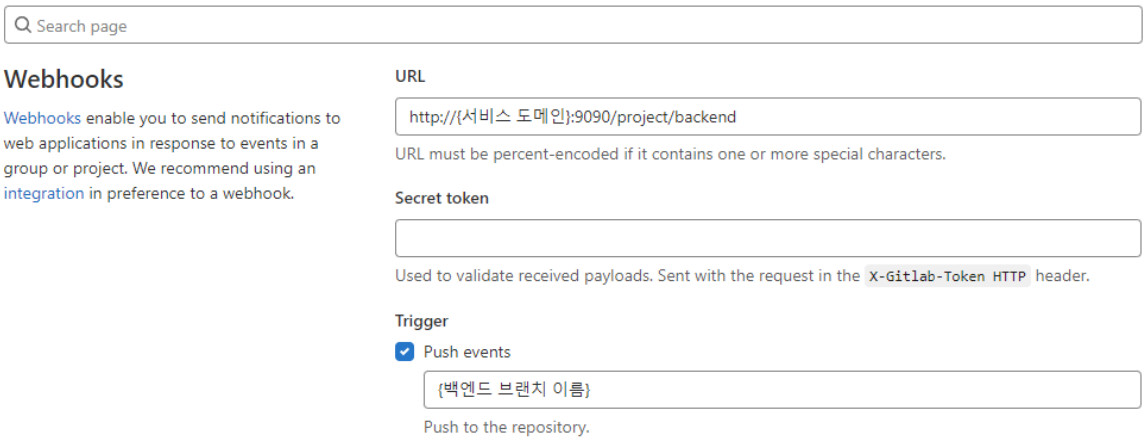
6. Jenkins에 sudo 권한 부여 **(중요한 파일이기 때문에 오타 없는지 한번 더 확인)**

```
$ 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

7. GitLab Webhook 작성 (Settings > Webhook)

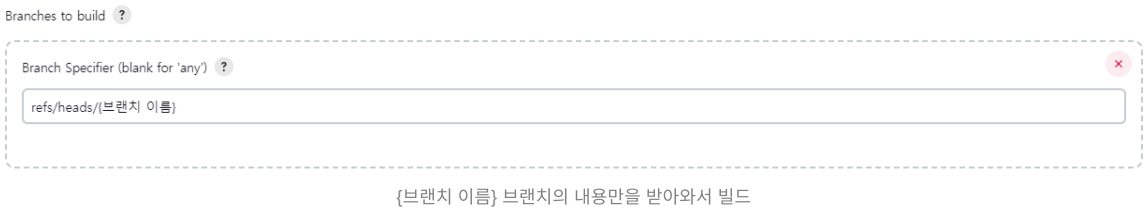


FrontEnd - 설정파일

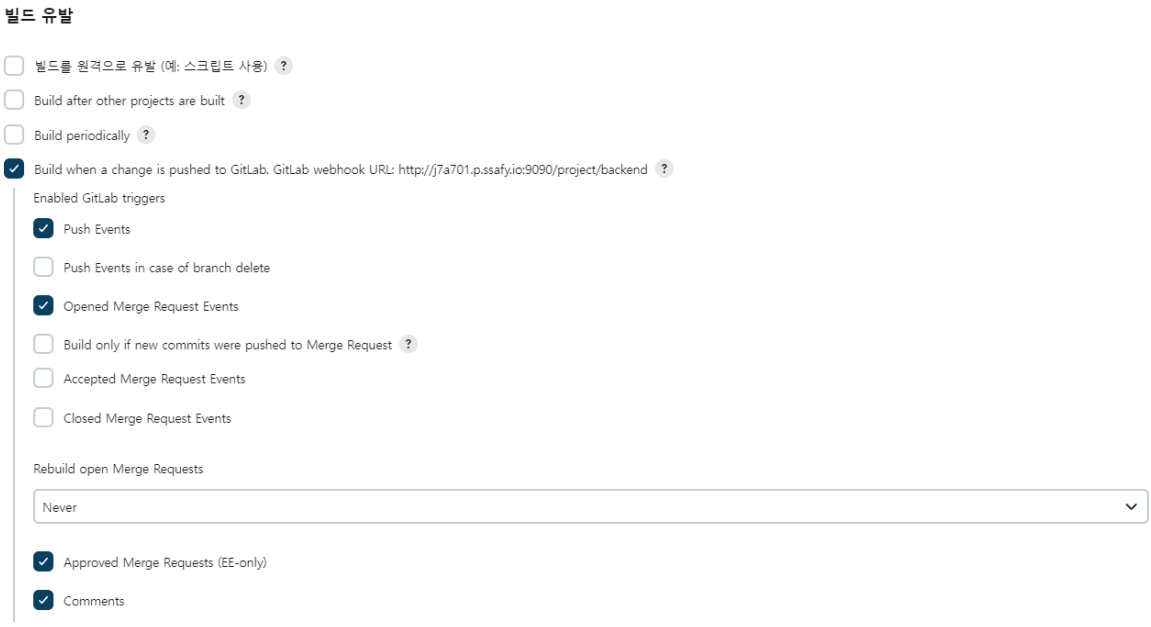
- .env 작성 (.env)
- Dockerfile 작성 (Dockerfile)

FrontEnd - Jenkins 설정

1. Jenkins 메인 화면 > Dashboard > 새로운 Item
2. FreeStyle project 선택하고 item name은 frontend로 설정 후 OK
3. Repositories URL에는 프로젝트 레포지토리의 HTTPS 주소 입력
4. 이전 과정에서 생성했던 Credential을 선택
5. 스크린샷과 같이 설정

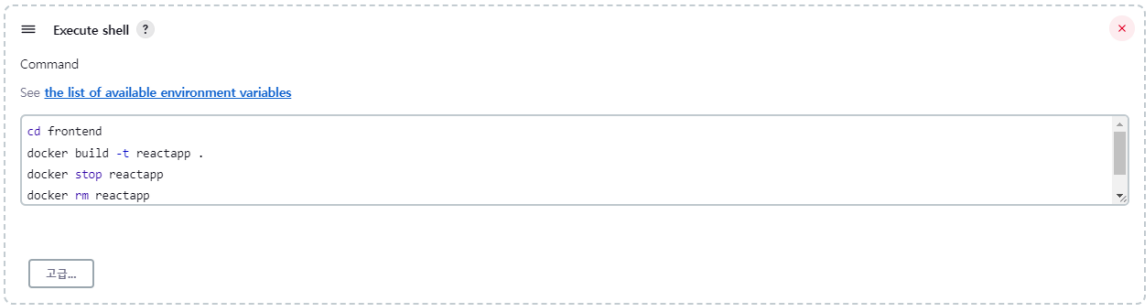


{브랜치 이름} 브랜치의 내용만을 받아와서 빌드



Push, Merge Request가 발생할 때마다 빌드 유발

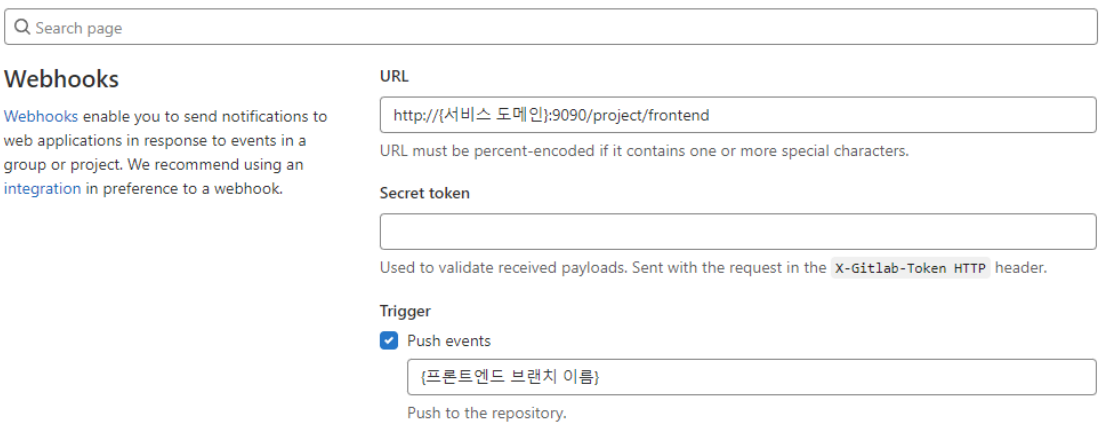
Build Steps



frontend 폴더로 이동하여 도커 컨테이너 생성, 기존 컨테이너 중지 후 삭제, 새로운 컨테이너 run

```
cd frontend
docker build -t reactapp .
docker stop reactapp
docker rm reactapp
docker run -p 3000:3000 --name reactapp --network ubuntu_default -d reactapp
```

7. GitLab Webhook 작성 (Settings > Webhook)



II. 외부 서비스

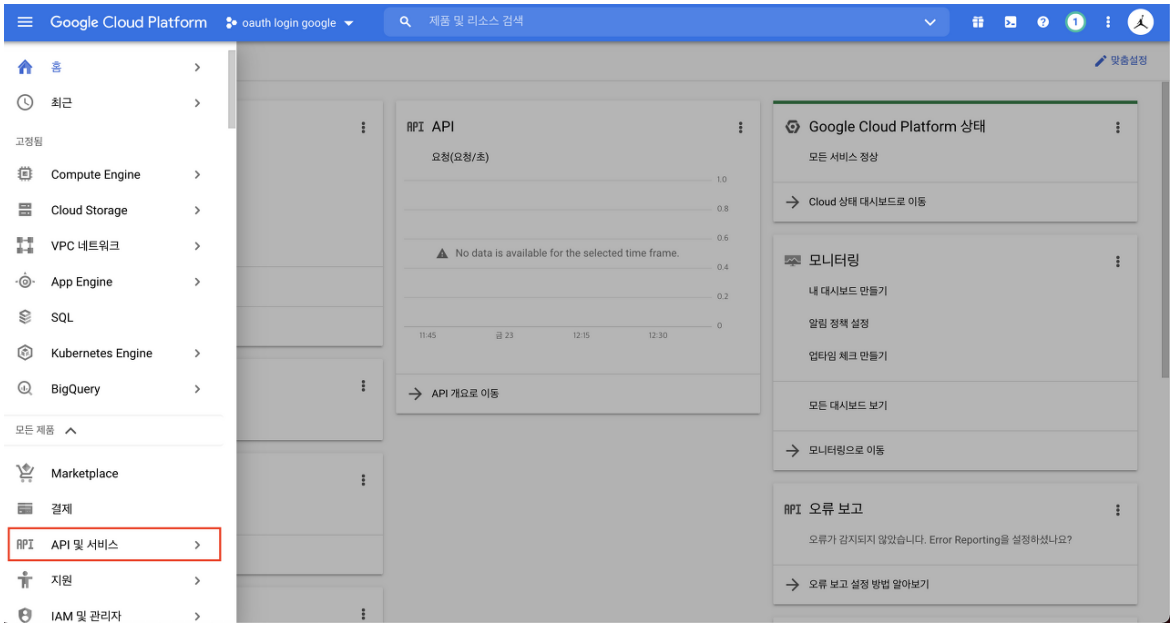
1. 소셜 로그인

Google(구글)

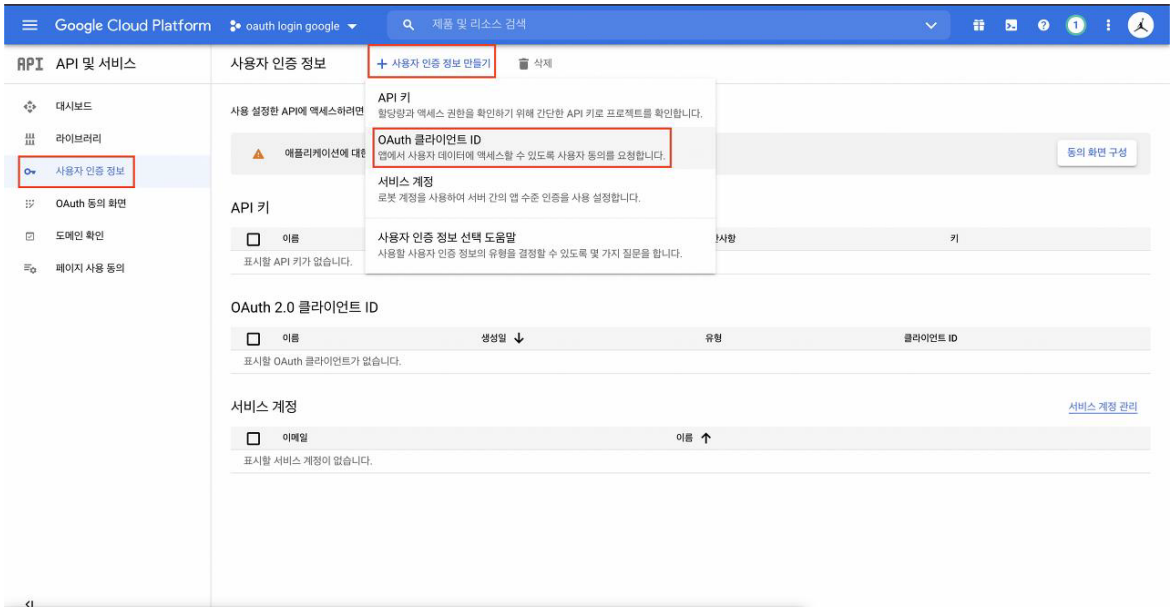
1. https://console.cloud.google.com/home/dashboard로 접속한다.

2. 프로젝트 선택을 클릭하고 모달 우측 상단의 새 프로젝트 클릭

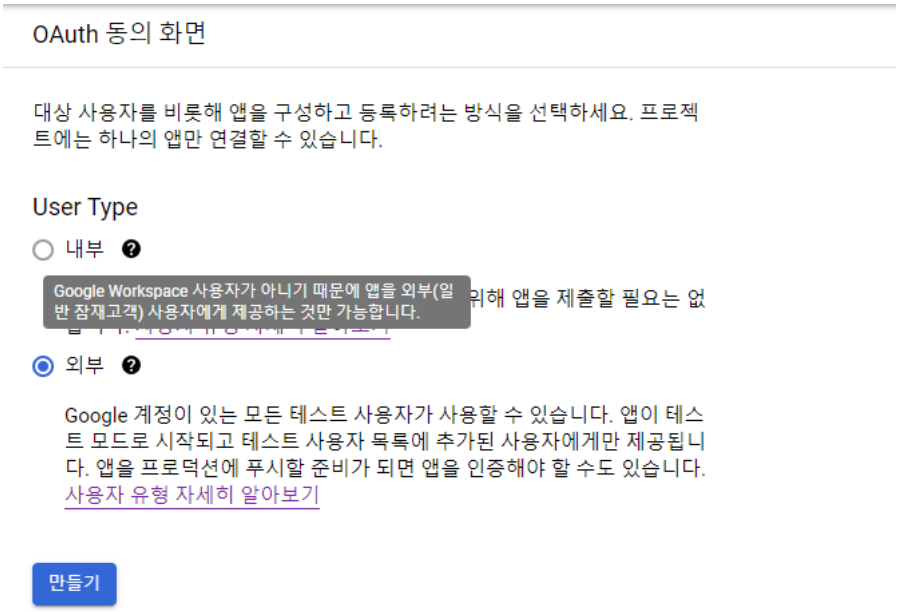
3. API 및 서비스



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

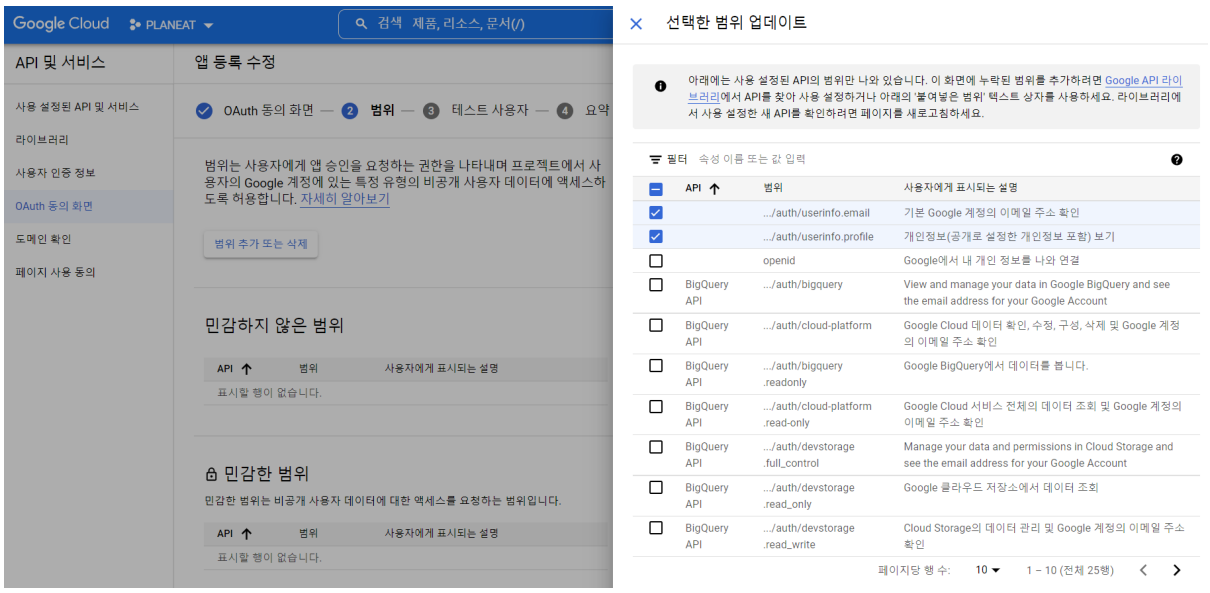


5. 동의 화면 구성을 클릭하고 해당 화면에서 외부로 선택하여 외부 사용자들이 사용할 수 있도록 설정한다.



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

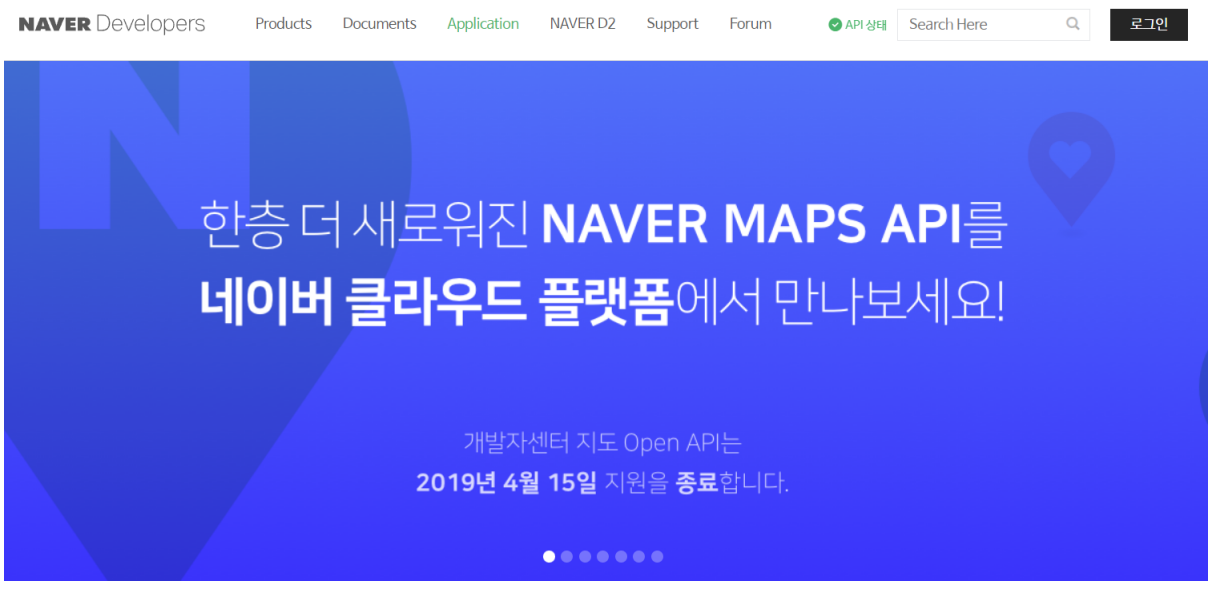
7. 범위 추가 또는 삭제 > email, profile 선택 후 저장
   민감하지 않은 범위에 추가된 것 확인



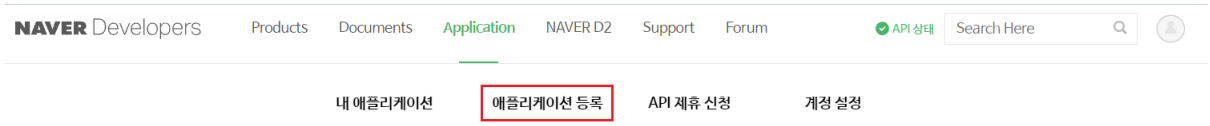
8. 사용자 인증정보 > 사용자 인증정보 만들기 > Oauth 클라이언트 ID
   승인된 리디렉션 URI 작성

Naver(네이버)

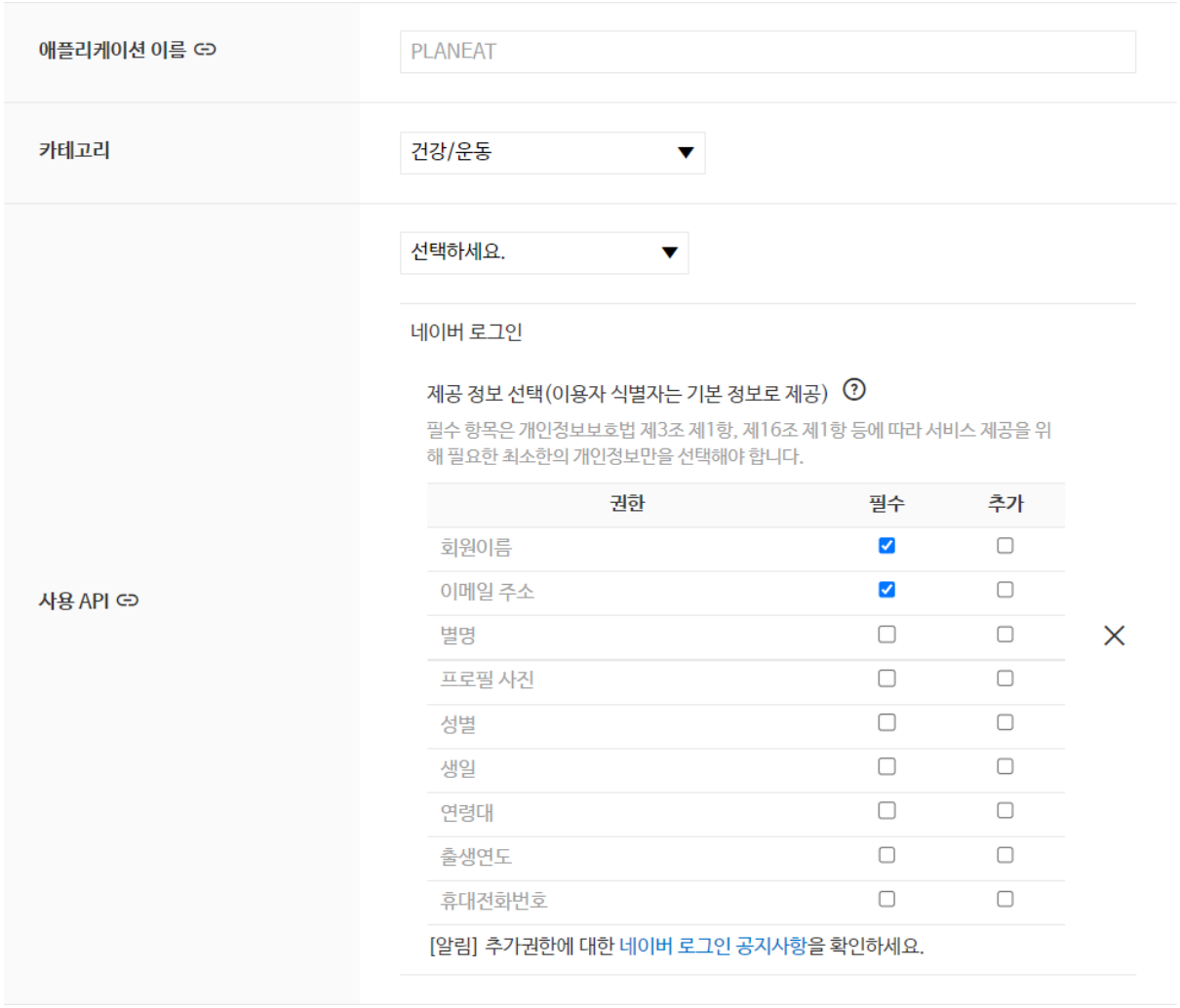
1. https://developers.naver.com/main/로 접속한다.

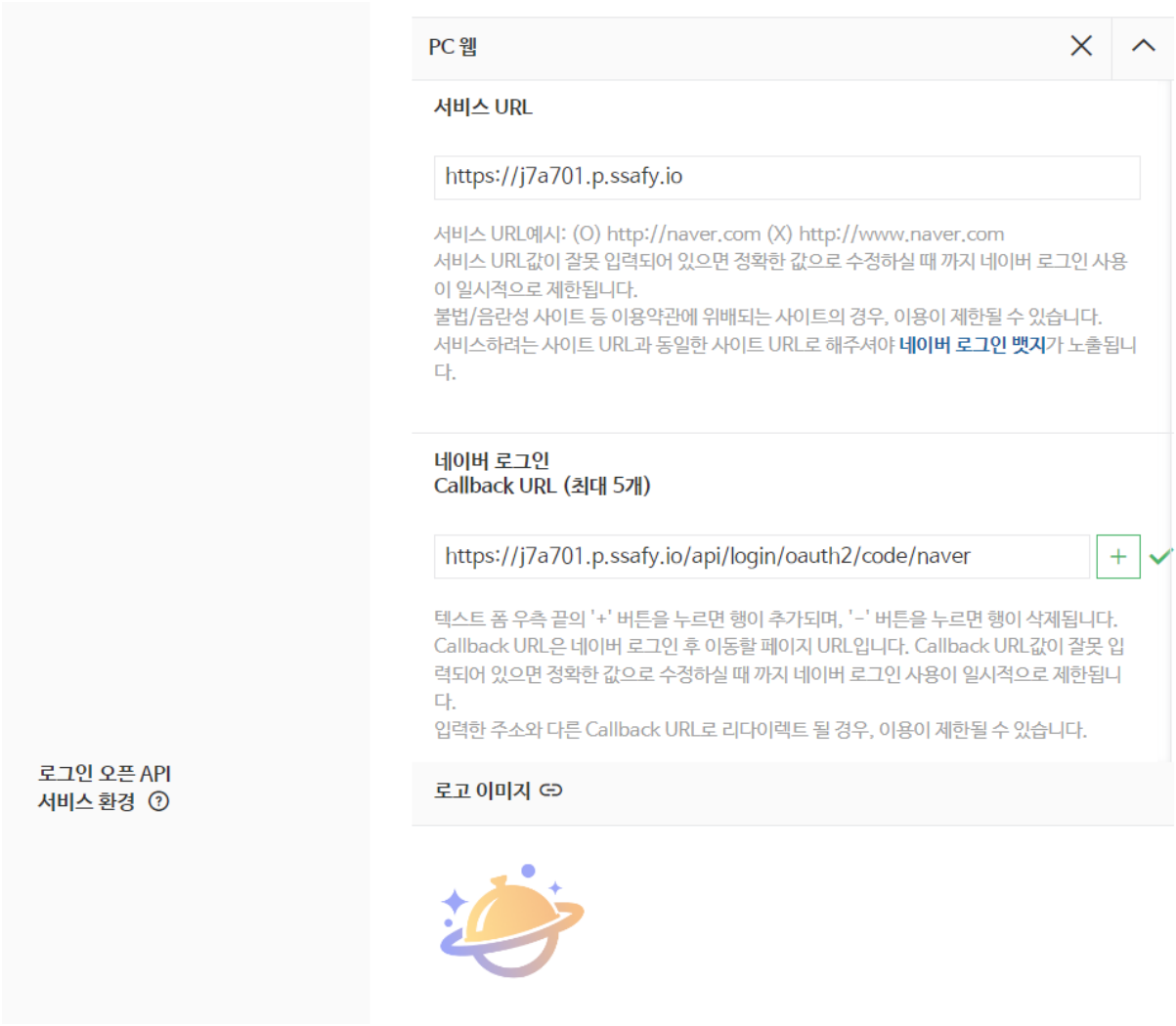


2. 상단의 Application > 애플리케이션 등록 클릭


NAVER Developers
Products
Documents
Application
NAVER D2
Support
Forum
API 상태
Search Here
내 애플리케이션
애플리케이션 등록
API 제휴 신청
계정 설정

3. 필요 정보를 기입한다.


애플리케이션 이름
PLANEAT
카테고리
건강/운동
사용 API
선택하세요.
네이버 로그인
제공 정보 선택(이용자 식별자는 기본 정보로 제공)
필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보만을 선택해야 합니다.
권한
필수
추가
회원이름
이메일 주소
별명
프로필 사진
성별
생일
연령대
출생연도
휴대전화번호
[알림] 추가권한에 대한 네이버 로그인 공지사항을 확인하세요.

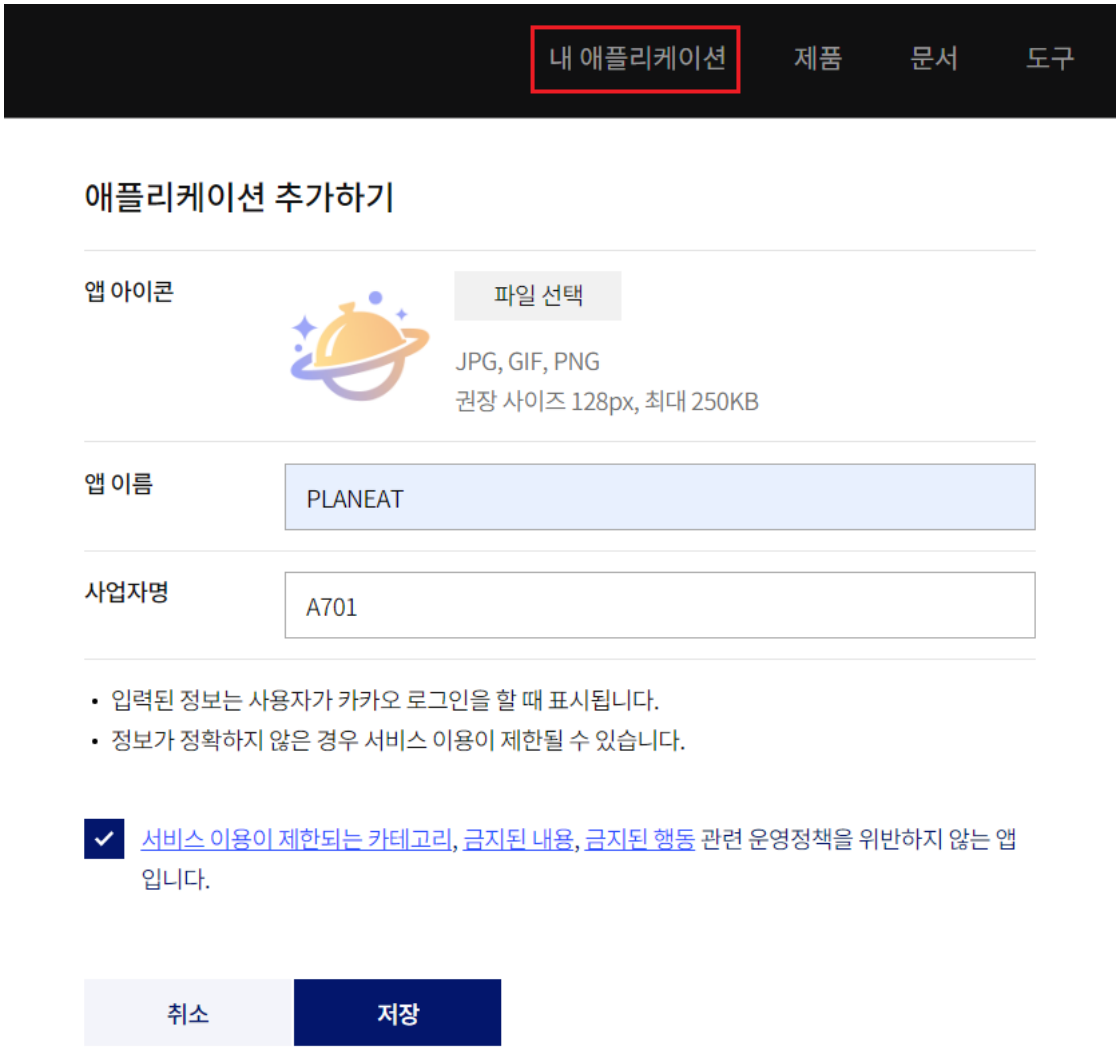


4. 멤버 관리에서 테스터 ID를 등록한다.

Kakao(카카오)

1. https://developers.kakao.com/로 접속한다.

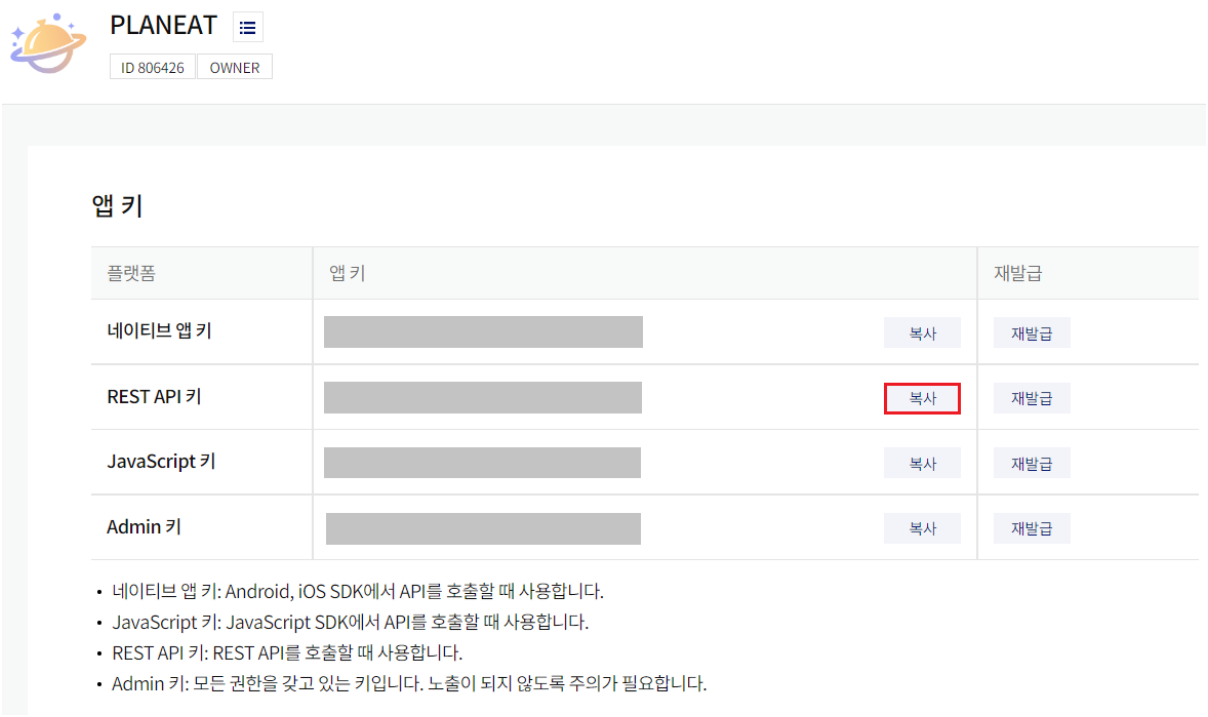
2. 내 애플리케이션 > 애플리케이션 추가하기 > 앱 이름 입력 > 사업자명 입력 > 저장



3. 좌측 Nav 바에서

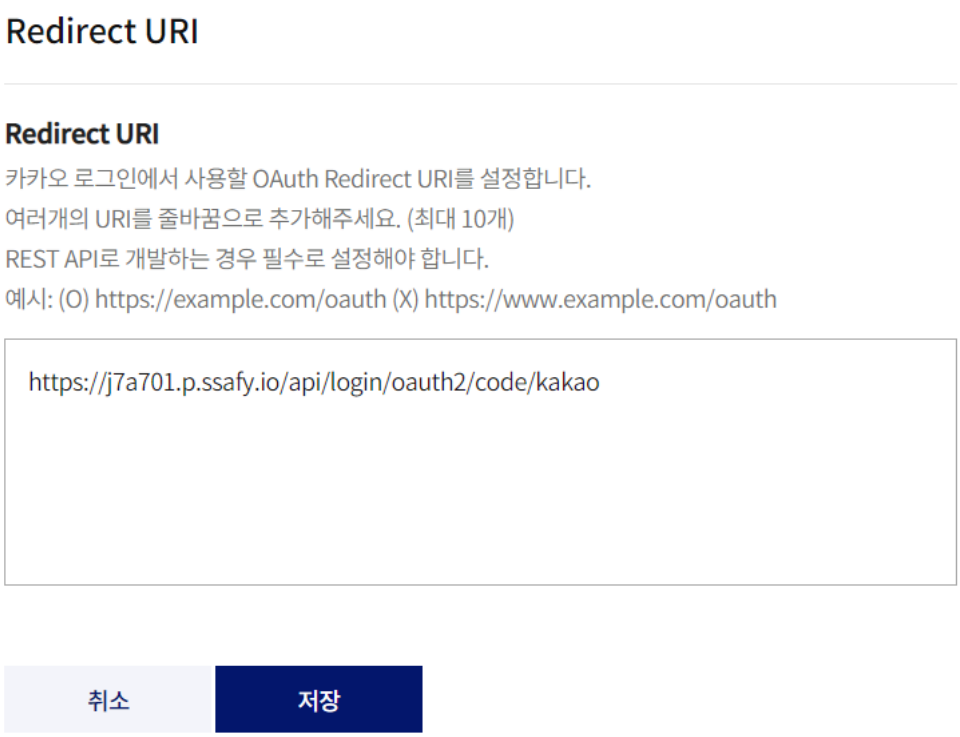
   앱 설정 > 요약 정보 > 앱 키 > REST API 키와

   제품 설정 > 카카오 로그인 > 보안 > Client Secret 의 코드를 기록해둔다.

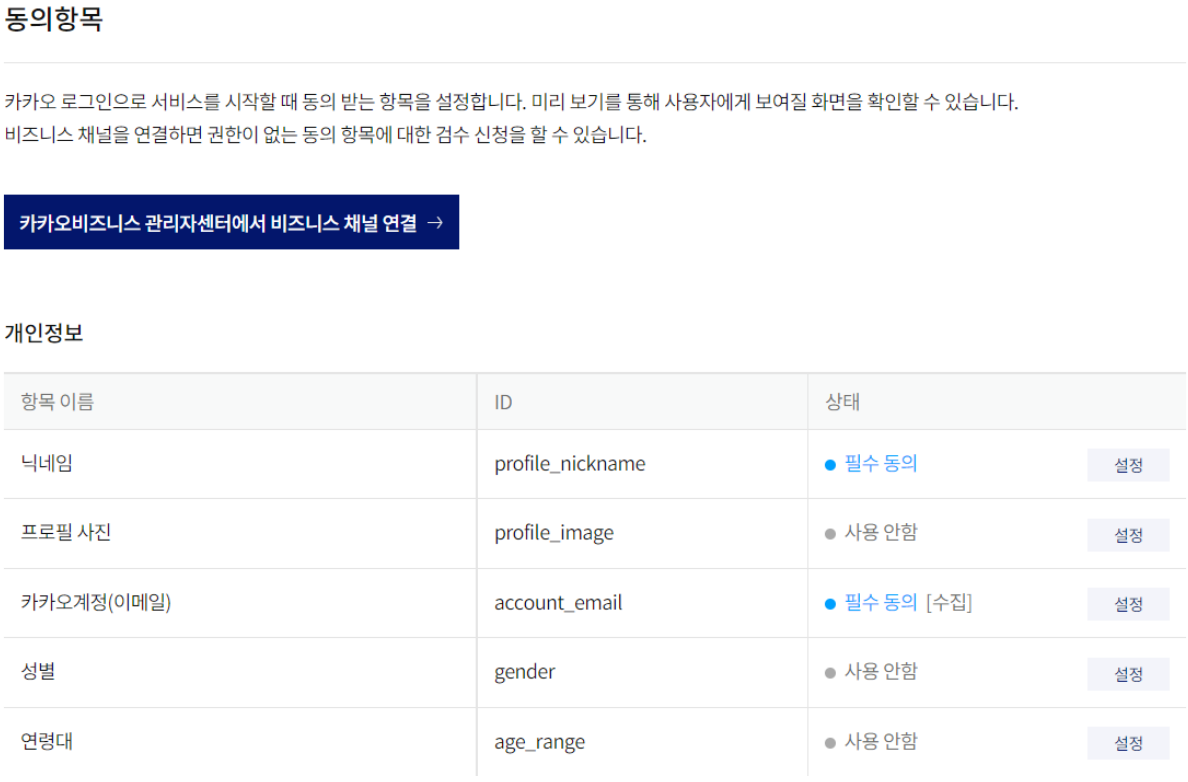


4. 좌측 Nav 바의
   앱 설정 > 플랫폼 > Web > Web 플랫폼 등록에서 필요 정보를 기입한다.

5. 제품 설정 > 카카오 로그인 클릭 후 Redirect URI 를 입력한다.



6. 제품 설정 > 카카오 로그인 > 동의항목에서 아래와 같이 설정한다.



- application-oauth.properties 작성 (application-oauth.properties)

2. Hadoop

1. 데이터 분산 처리

- 데이터를 하둡 분산 파일 시스템에 저장

```
hdfs dfs -put review(파일명) review(파일명)
```

2. 하둡 mapreduce 실행

- wordcount 로직

```
package ssafy;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.util.GenericOptionsParser;

public class Wordcount {
	/*
	Object, Text : input key-value pair type (always same (to get a line of input file))
	Text, IntWritable : output key-value pair type
	*/
	public static class TokenizerMapper
			extends Mapper<Object,Text,Text,IntWritable> {

		// variable declairations
		private final static IntWritable one = new IntWritable(1);
		private Text word = new Text();
		private String filename;
		protected void setup(Context context) throws IOException, InterruptedException{
			filename = ((FileSplit)context.getInputSplit()).getPath().getName();
		}


		// map function (Context -> fixed parameter)
		public void map(Object key, Text value, Context context)
				throws IOException, InterruptedException {

			// value.toString() : get a line
			StringTokenizer itr = new StringTokenizer(value.toString());
			while ( itr.hasMoreTokens() ) {
				String token = filename + "     " + itr.nextToken();
				word.set(token);

				// emit a key-value pair
				context.write(word,one);
			}
		}
	}

	/*
	Text, IntWritable : input key type and the value type of input value list
	Text, IntWritable : output key-value pair type
	*/
	public static class IntSumReducer
			extends Reducer<Text,IntWritable,Text,IntWritable> {

		// variables
		private IntWritable result = new IntWritable();

		// key : a disticnt word
		// values :  Iterable type (data list)
		public void reduce(Text key, Iterable<IntWritable> values, Context context)
				throws IOException, InterruptedException {

			int sum = 0;
			for ( IntWritable val : values ) {
				sum += val.get();
			}
			result.set(sum);
```

```
                        context.write(key,result);
                }
        }


        /* Main function */
        public static void main(String[] args) throws Exception {
                Configuration conf = new Configuration();
                String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
                if ( otherArgs.length != 2 ) {
                        System.err.println("Usage: <in> <out>");
                        System.exit(2);
                }
                Job job = new Job(conf,"word count");
                job.setJarByClass(Wordcount.class);

                // let hadoop know my map and reduce classes
                job.setMapperClass(TokenizerMapper.class);
                job.setReducerClass(IntSumReducer.class);

                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

                // set number of reduces
                job.setNumReduceTasks(2);

                // set input and output directories
                FileInputFormat.addInputPath(job,new Path(otherArgs[0]));
                System.exit(job.waitForCompletion(true) ? 0 : 1 );
        }
}
```

- Driver.java

```
package ssafy;

import org.apache.hadoop.util.ProgramDriver;

public class Driver {
        public static void main(String[] args) {
                int exitCode = -1;
                ProgramDriver pgd = new ProgramDriver();
                try {

                        pgd.addClass("wordcount", Wordcount.class, "A map/reduce program that performs word counting.");
                        pgd.addClass("wordcount1char", Wordcount1char.class, "A map/reduce program that counts the 1st character of wo
                        pgd.addClass("wordcountsort", Wordcountsort.class, "A map/reduce program that output frequency of the words in
                        pgd.addClass("inverted", InvertedIndex.class, "A map/reduce program that generates the inverted index using wo
                        pgd.addClass("matadd", MatrixAdd.class, "A map/reduce program that computes the addition of two matrics.");
                        pgd.driver(args);
                        exitCode = 0;
                }
                catch(Throwable e) {
                        e.printStackTrace();
                }

                System.exit(exitCode);
        }
}
```

- 수집한 리뷰 데이터 wordcount 실행

```
hadoop jar ssafy.jar(jar파일명) wordcount review(파일명) wordcount_review(결과 파일명)
```

3. 결과 파일 가져오기

```
hdfs dfs -get wordcount_review wordcount_review
```