Starry Night

PORTING MANUAL

목 차

I. 빌드 및 배포

II. 외부 서비스

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-Oubuntu120.04

Docker : 20.10.21

Node.js : 18.12.1

MySQL : 8.0.31-1.el8

Nginx : 1.23.2

Jenkins : 2.361.3

Unity : 2021.3.11f1

2. 설정 파일 목록

React

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

SpringBoot

- **application.properties** : /jenkins/workspace/backend/backend/src/main/resources

- **application-oauth.properties** : /jenkins/workspace/backend/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

- **Dockerfile**

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

Springboot

- application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://mysql:3306/dreammungz?useSSL=fals
e&useUnicode=true&serverTimezone=Asia/Seoul
spring.datasource.username={MySQL 사용자 이름}
spring.datasource.password={MySQL 패스워드}

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true

server.port=8081
```

- 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
#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_nickname, 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
```

- 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 전 cleanQuery'
./gradlew cleanQuerydslSourceDir

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 8081:8081 --name springbootapp --network
ubuntu_default -d springbootapp
```

Docker

- docker-compose.yml

```
version: "3"
services:
  mysql:
    image: mysql

    container_name: mysql
    environment:
      MYSQL_DATABASE: {scheme명}
      MYSQL_ROOT_PASSWORD: {root 계정 패스워드}
    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 default_server;
        listen [::]:80 default_server;
        server_name starry-night.kr www.starry-night.kr;
```

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

server {
        listen 443 ssl;
        listen [::]:443 ssl;
        server_name starry-night.kr;
        access_log off;
        ssl_certificate /etc/letsencrypt/live/starry-night.kr/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/starry-
night.kr/privkey.pem;

        location /api/ {
                proxy_pass http://starry-night.kr:8081/;
                proxy_set_header Host $host;
                proxy_redirect off;
        }

        location / {
                proxy_pass http://starry-night.kr:3000;
                proxy_set_header Host $host;
                proxy_redirect off;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
        }
}
```

4. **Docker 설치**

- Docker Install

```
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 Install

```
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

해당 경로에 Key 생성 여부 확인
ssl_certificate /etc/letsencrypt/live/{도메인}/fullchain.pem
ssl_certificate_key /etc/letsencrypt/live/{도메인}/privkey.pem

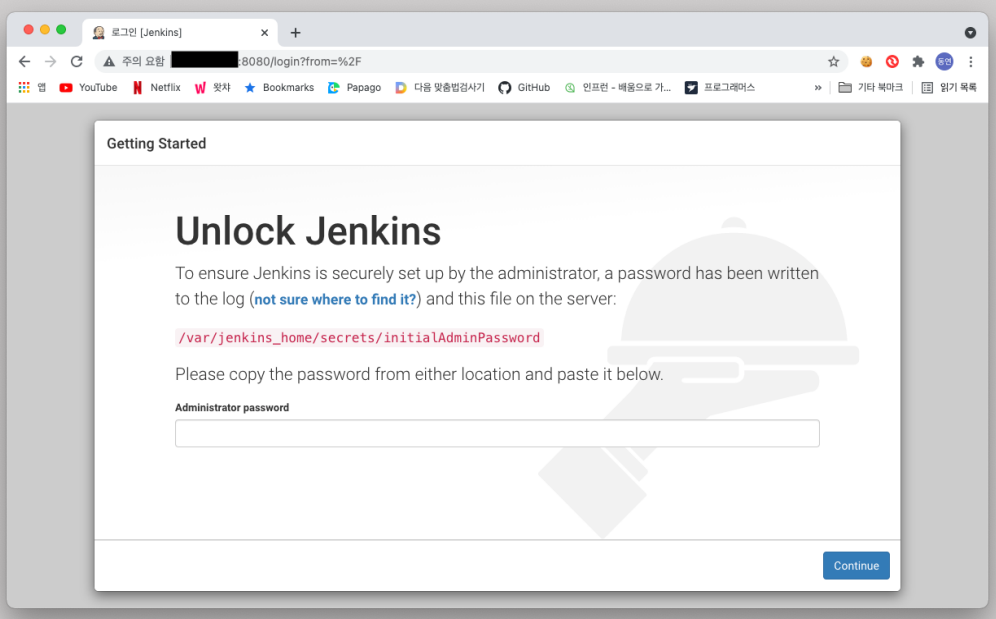
6. Infra 배포

- docker-compose 실행

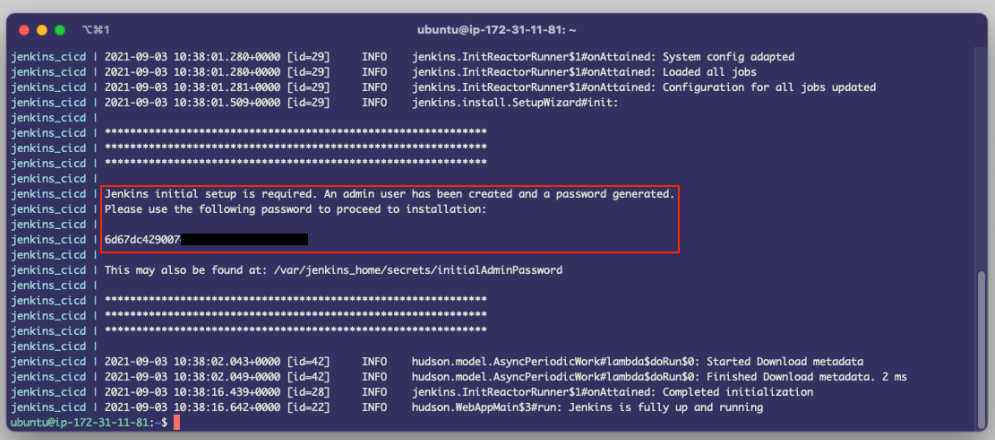
cd /home/ubuntu/mysql,jenkins,nginx (docker-compose.yml 경로에서)

sudo docker-compose up -–build –d

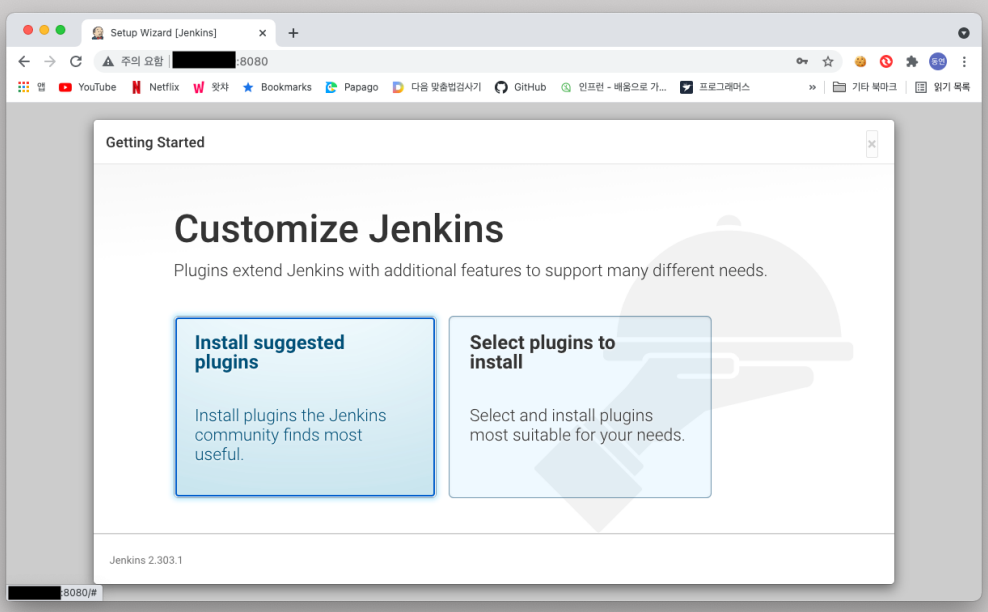
- Jenkins 플러그인 및 docker-in-docker 설치



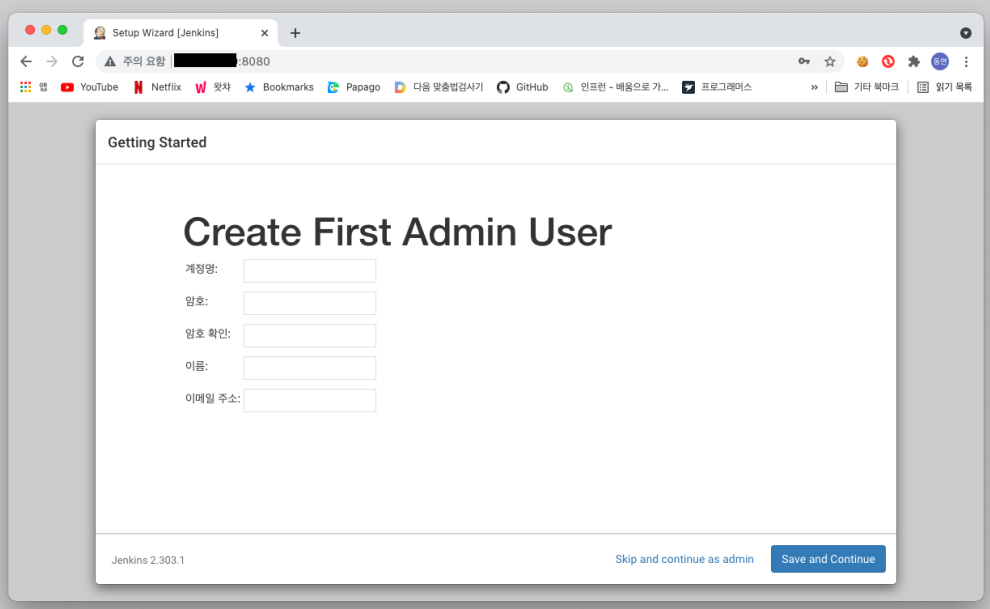
1. http://서비스도메인:9090 으로 접속하여 jenkins 페이지 진입



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

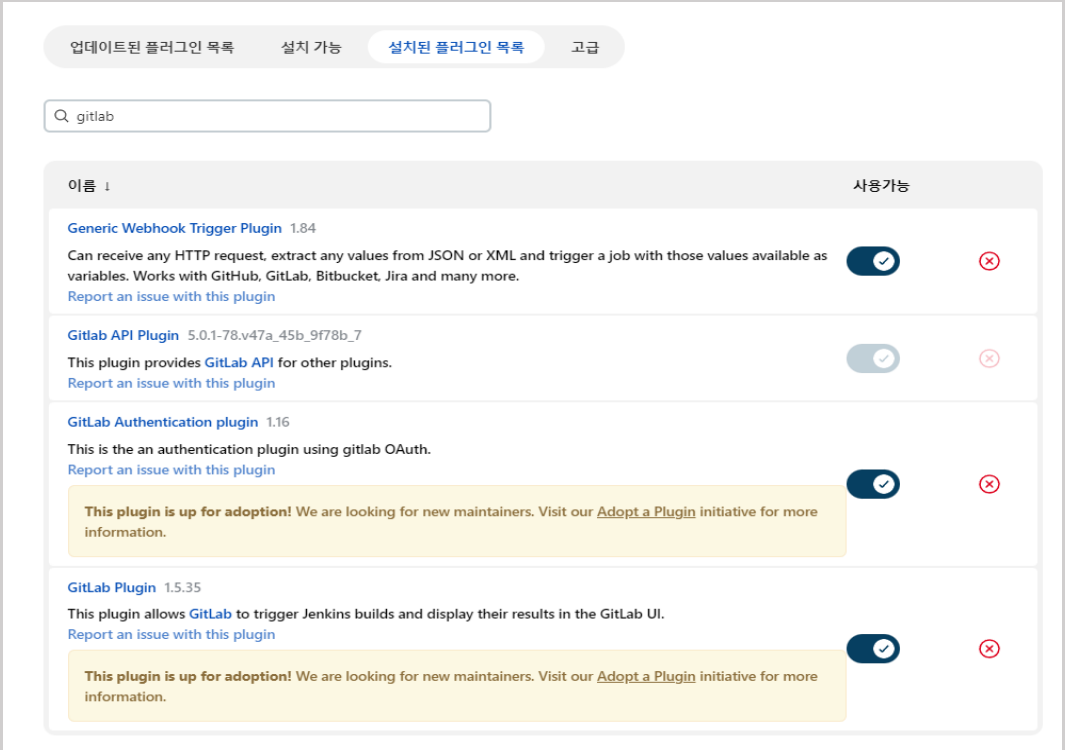


3. Install suggested plugins을 선택하여 플러그인들을 설치



4. 생성할 관리자 계정 정보를 입력하고 Save and Continue

5. Jenkins 접속 URL확인 후 Save and Finish



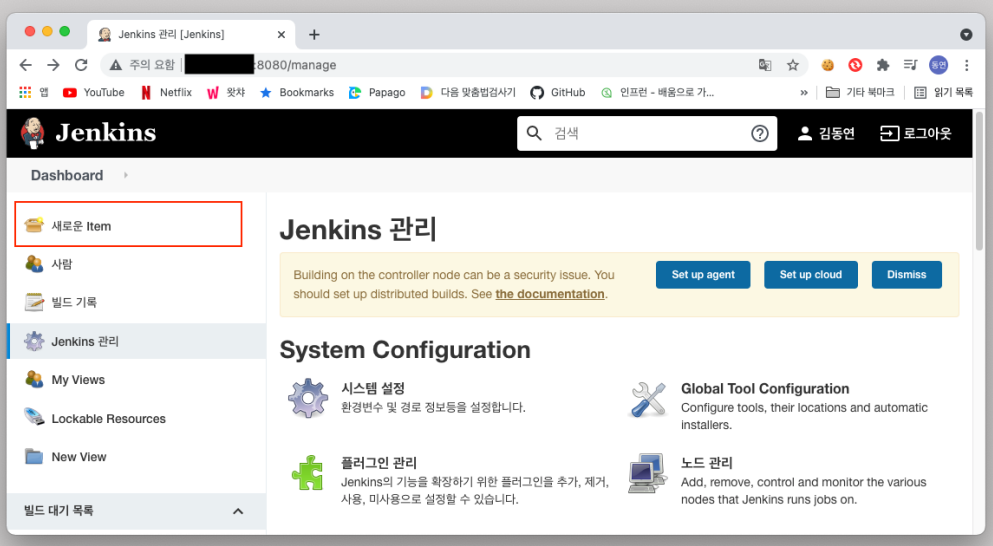
6. 메인 화면에서 DashBoard -> Manager JenKins -> Plugin Manager에서 gitlab, docker에 대해서 플러그인을 설치

7. 서버 콘솔로 돌아가 Jenkins 내부에 docker를 설치

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

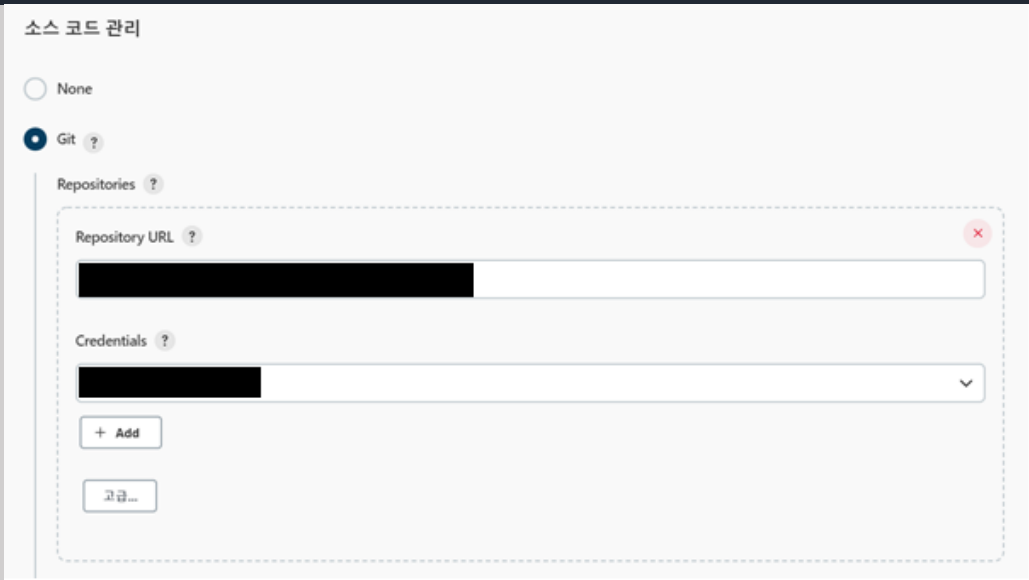
7. Backend CI/CD

- Jenkins Job 설정

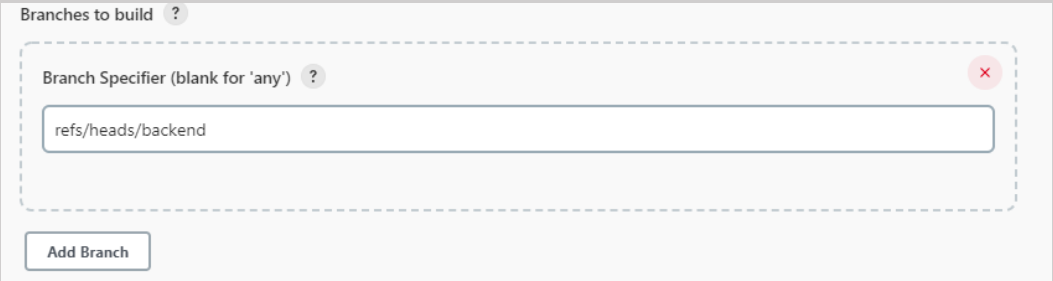


1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭

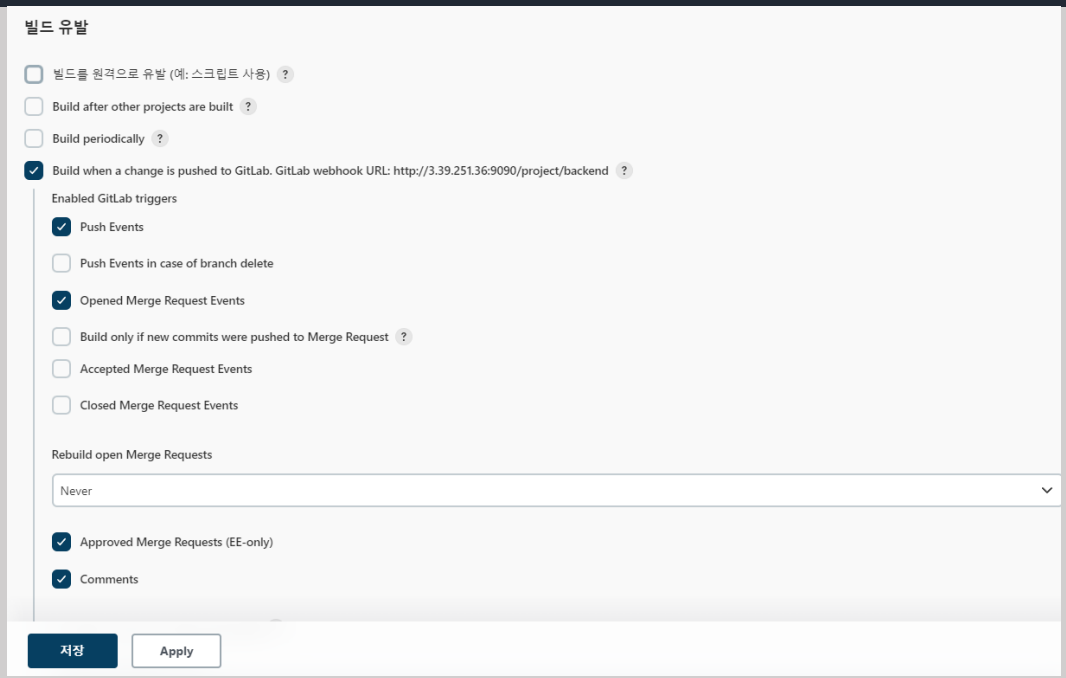
2. FreeStyle project를 선택하고 item name은 backend로 설정 후 OK



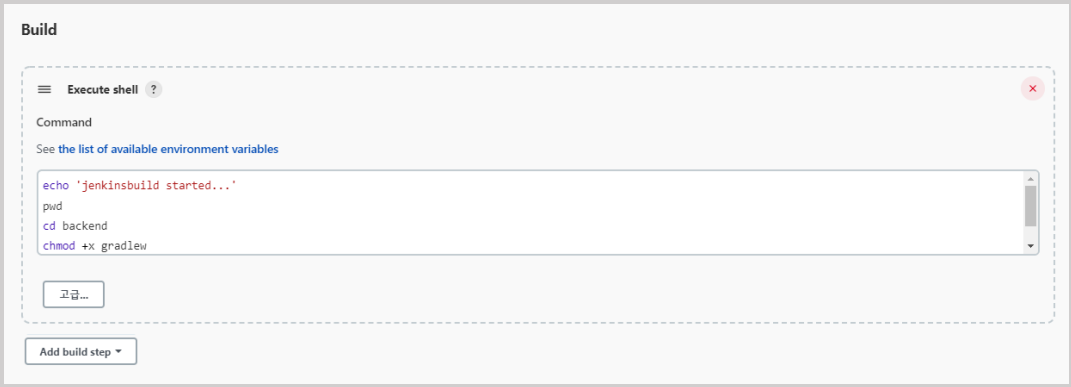
3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력
Credentials의 Add를 클릭,
Domain -> Global credentials
Kind -> Username with password
Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디
Password -> Username에 작성한 Gitlab 계정 비밀번호
순서대로 입력하고 드롭박스에서 생성된 Credential을 선택



4. backend 브랜치의 내용만을 받아와서 빌드하기 위한 설정



5. 빌드 유발을 다음과 같이 설정
하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록



6. Build 탭에 Excute shell을 선택하고
cd backend
chmod +x gradlew
chmod +x deploy.sh
./deploy.sh

- Jenkins에 sudo 권한 부여

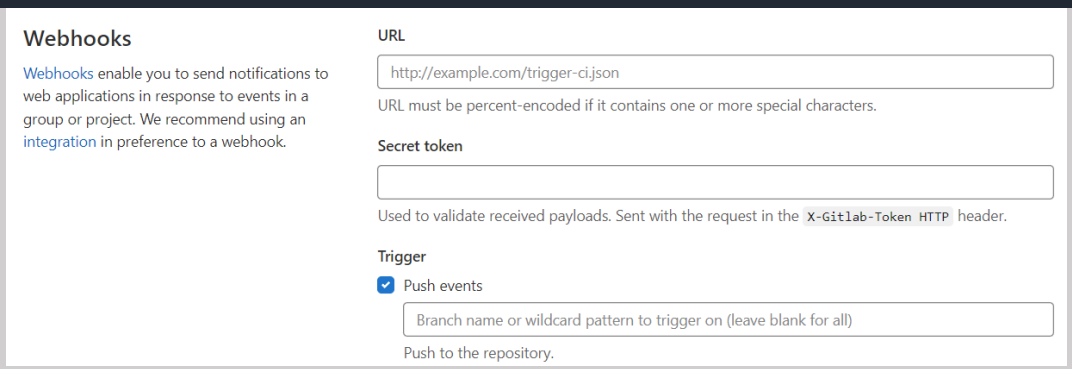
sudo vim /etc/sudoers

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

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

'

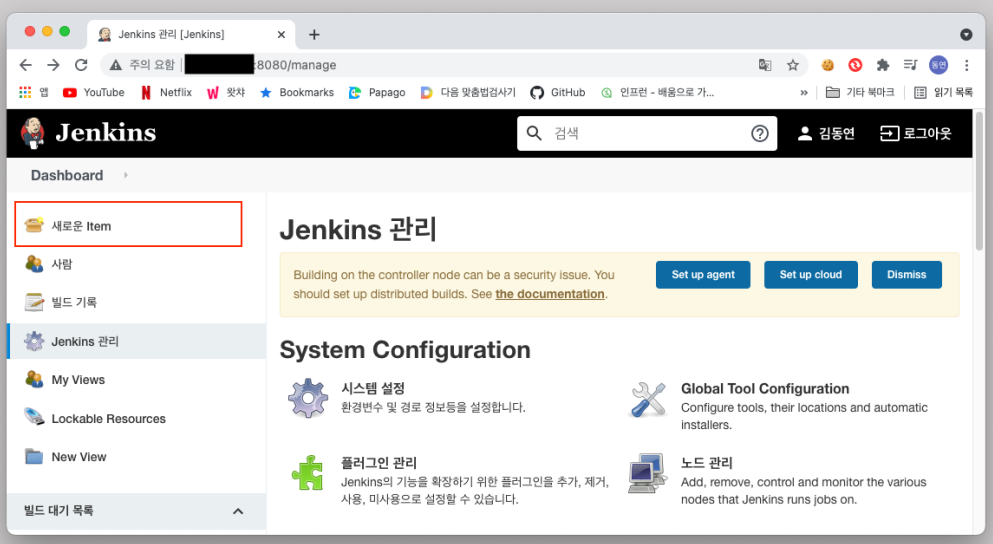
- Gitlab Webhook 설정



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

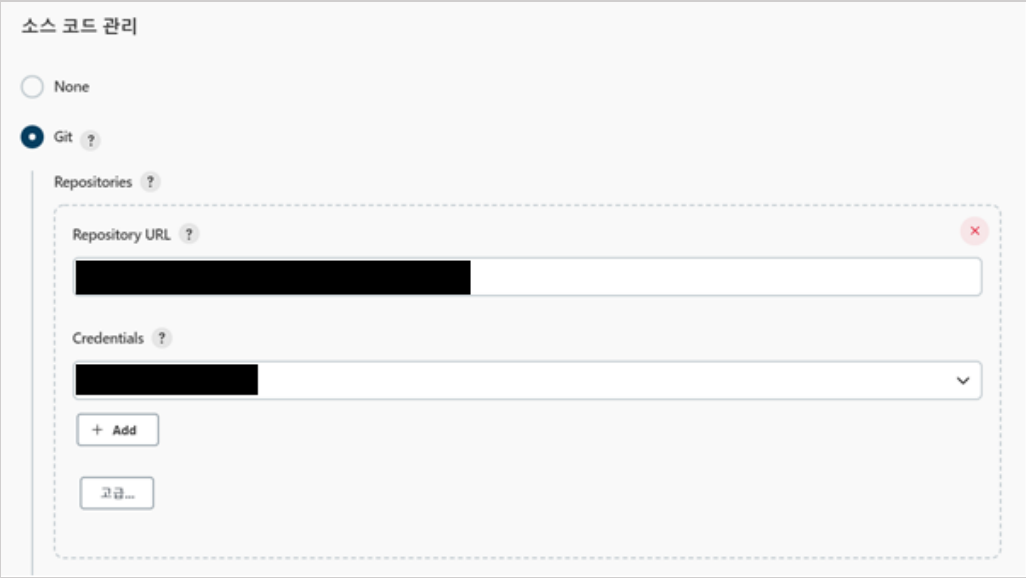
8. Frontend CI/CD

- Jenkins Job 설정



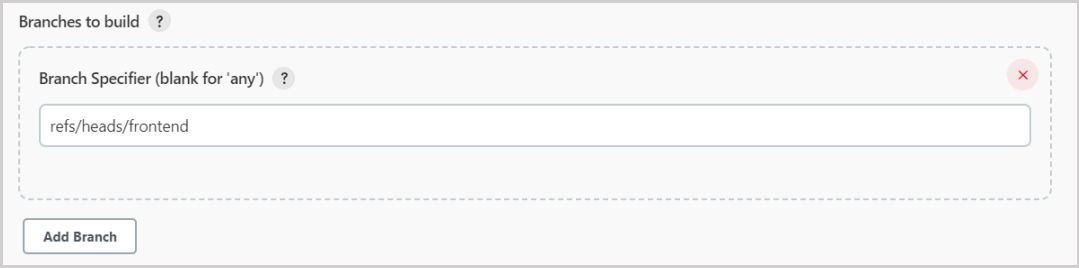
1. Jenkins 메인 화면 -> Dashboard -> 새로운 Item 클릭

2. FreeStyle project를 선택하고 item name은 backend로 설정 후 OK

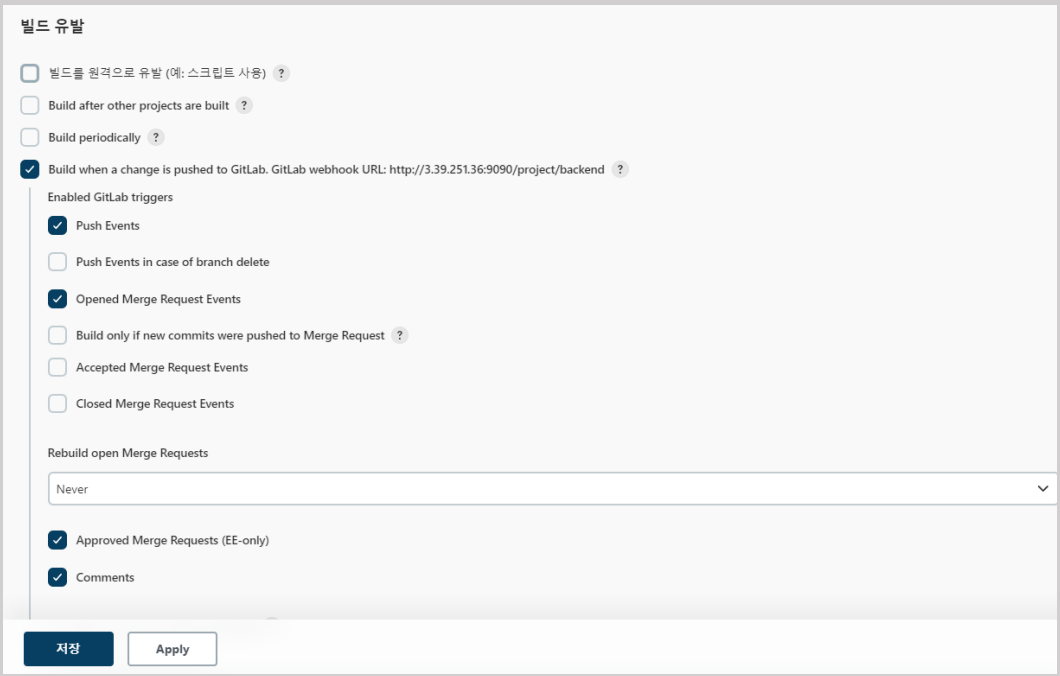


3. Repositories URL에는 프로젝트 레포지토리의 HTTPS Clone 주소 입력
Credentials의 Add를 클릭,
Domain -> Global credentials
Kind -> Username with password
Username -> 레포지토리 접근 권한이 있는 Gitlab 계정 아이디
Password -> Username에 작성한 Gitlab 계정 비밀번호

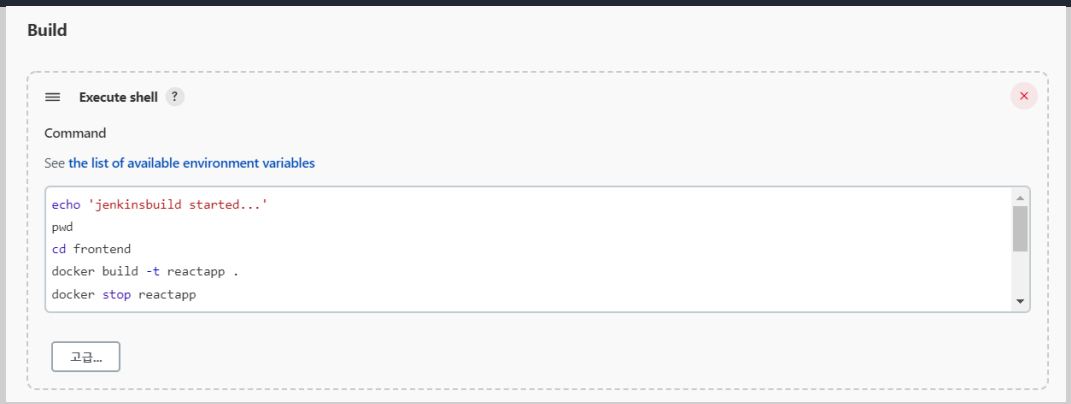
순서대로 입력하고 드롭박스에서 생성된 Credential을 선택



4. Frontend 브랜치의 내용만을 받아와서 빌드하기 위한 설정

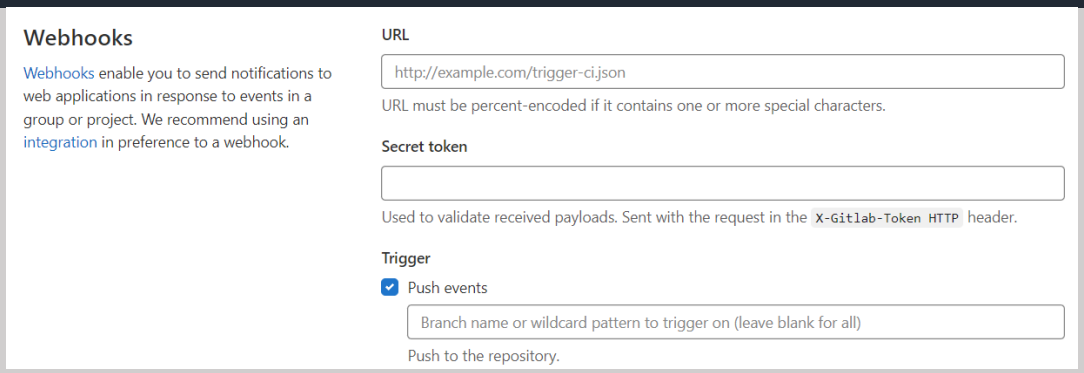


5. 빌드 유발을 다음과 같이 설정
하단의 고급 기능을 눌러 Secret Token을 Generate하여 기록



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

- Gitlab Webhook 설정



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. Unity 빌드

- Unity 프로젝트 빌드

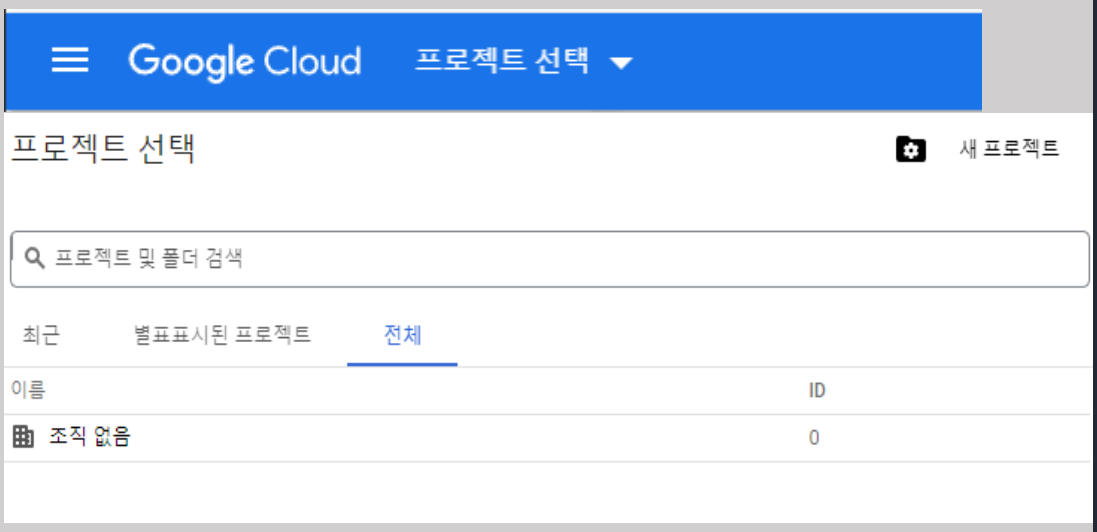
1. Unity 프로젝트를 실행하고 Ctrl + Shift + B
2. 플랫폼을 PC, Mac & Linux Standalone으로 설정
3. Build 클릭

II. 외부 서비스

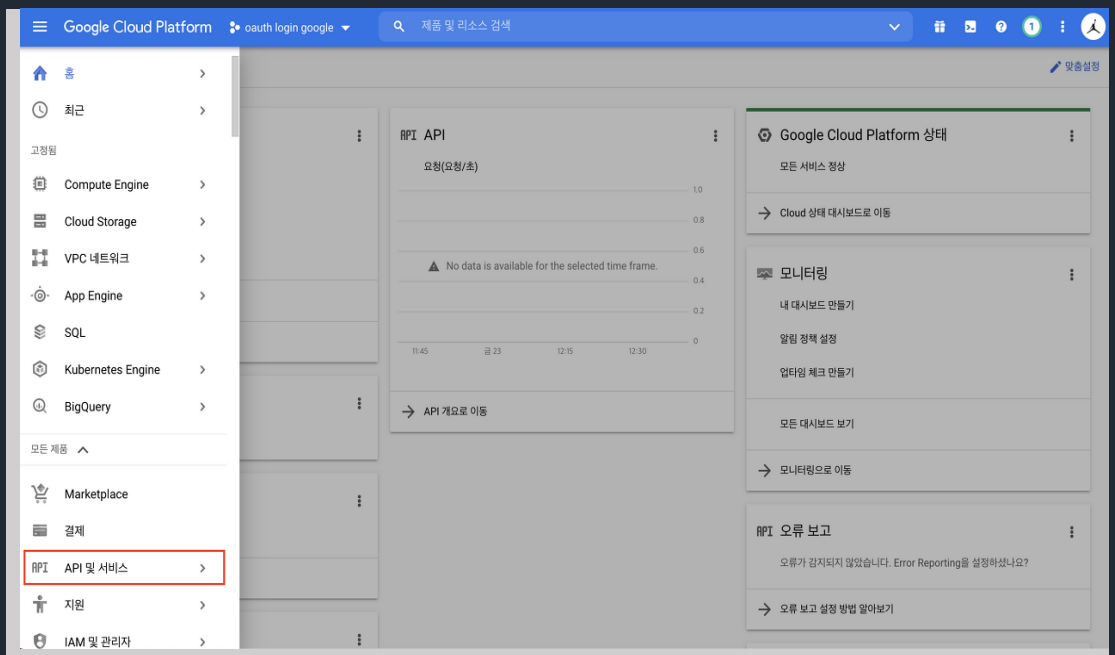
1. 소셜 로그인

- 구글(Google)

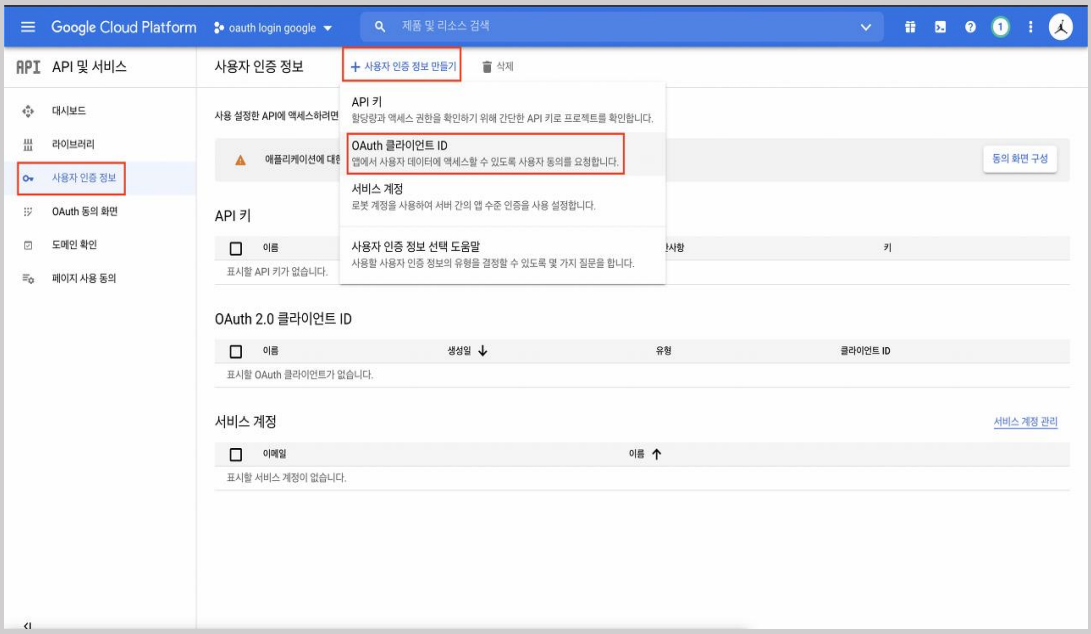
1. https://console.cloud.google.com/home/dashboard 로 접속한다.



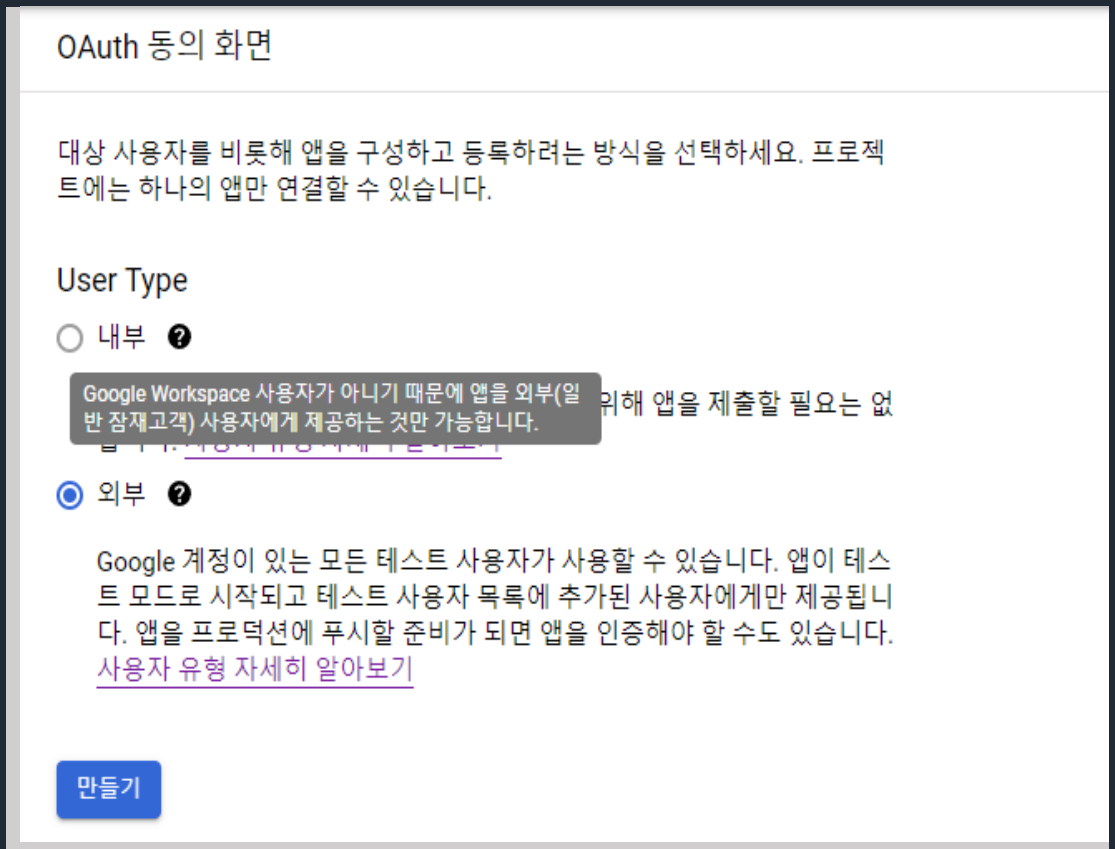
2. 프로젝트 선택을 클릭하고 모달 우측 상단의 새 프로젝트 클릭



3. API 및 서비스 선택

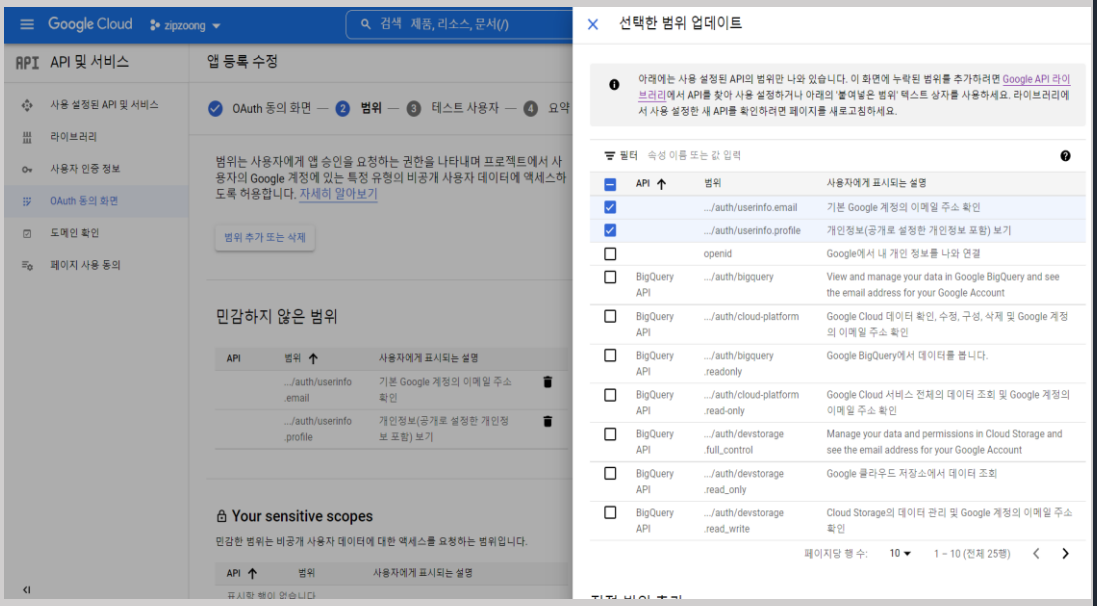


4. 사용자 인증정보 > 사용자 인증 정보 만들기 > Oauth 클라이언트 ID

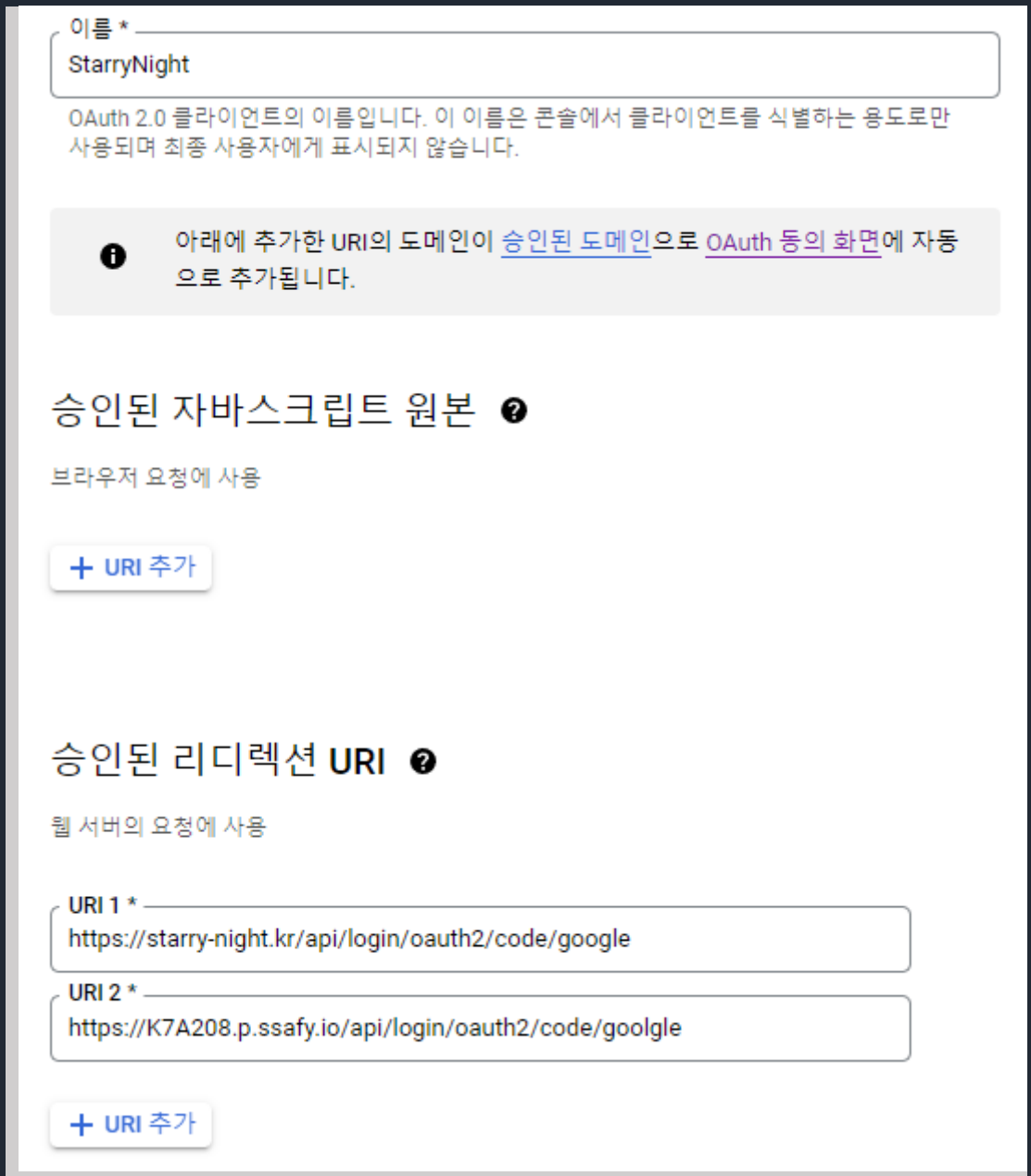


5. 동의 화면 구성을 클릭하고 해당 화면에서 외부로 선택하여 외부 사용자들이 사용할 수 있도록 설정

6. 앱 정보를 입력하고, 개발자 연락처 정보에 이메일 작성



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

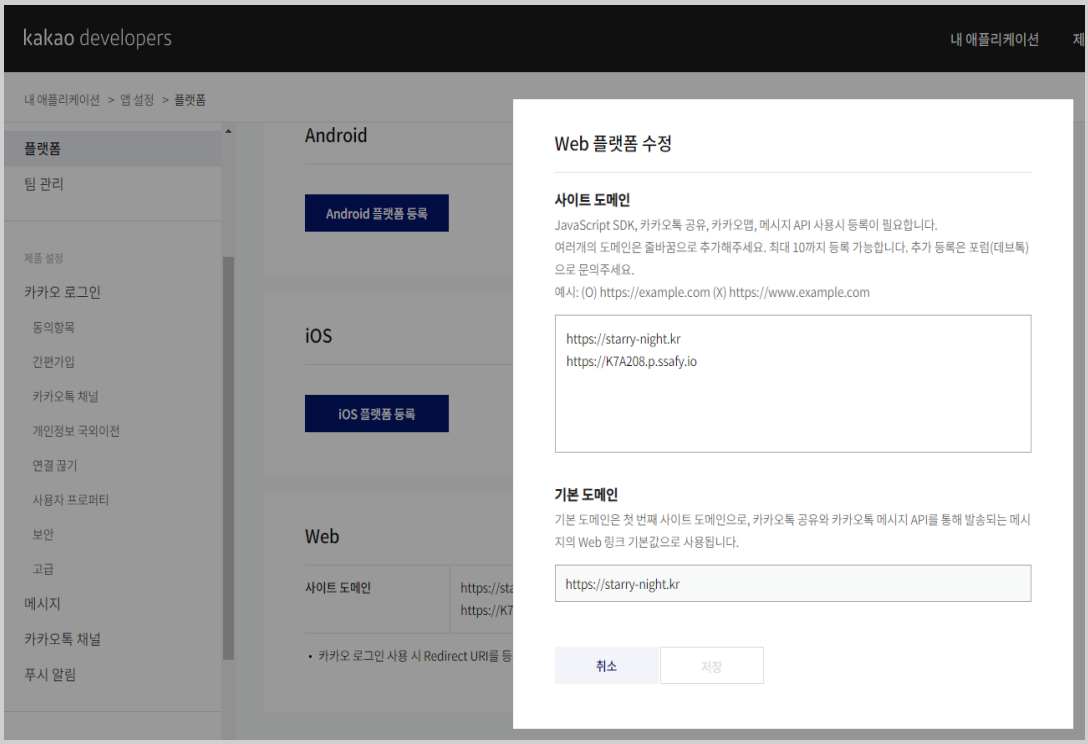
- 카카오(Kakao)

1. https://developers.kakao.com/ 로 접속

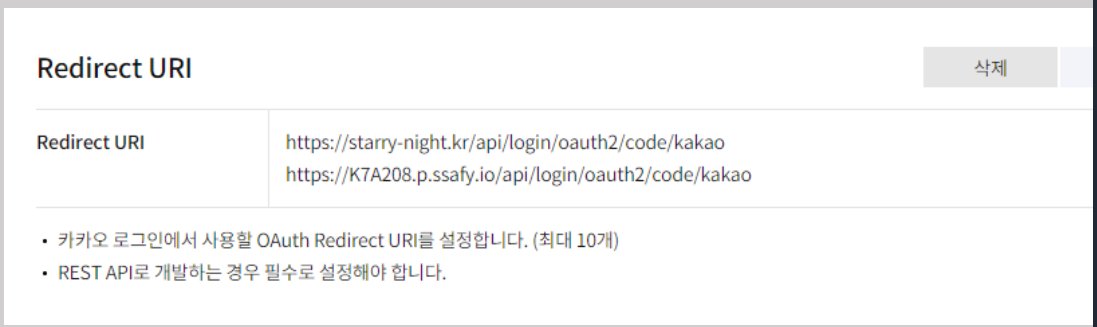
2. 내 애플리케이션 > 애플리케이션 추가하기 > 앱 이름 입력 > 사업자명 입력 > 저장

3. 좌측 Nav 바에서 앱 설정 > 요약 정보 > 앱 키 > REST API키

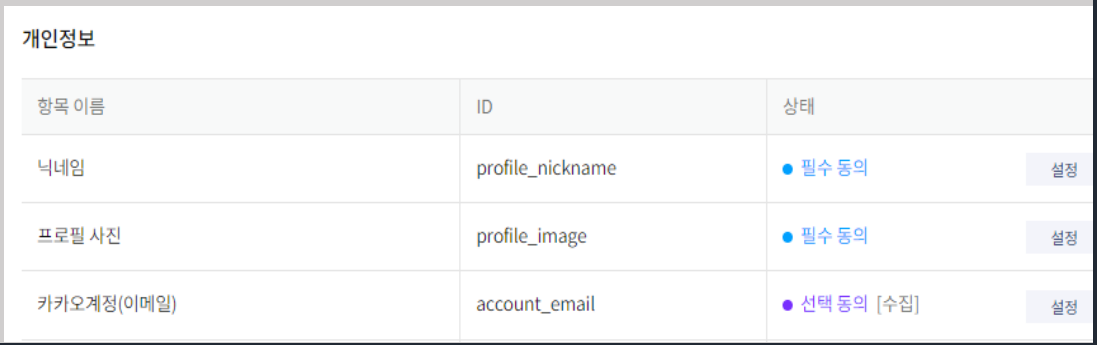
제품 설정 > 카카오 로그인 > 보안 > Client Secret의 코드 발급



4. 좌측 Nav 바에서 앱 설정 > 플랫폼 > Web > Web 플랫폼 등록 필요 정보를 기입



5. 제품 설정 > 카카오 로그인 클릭 Redirect URI를 입력

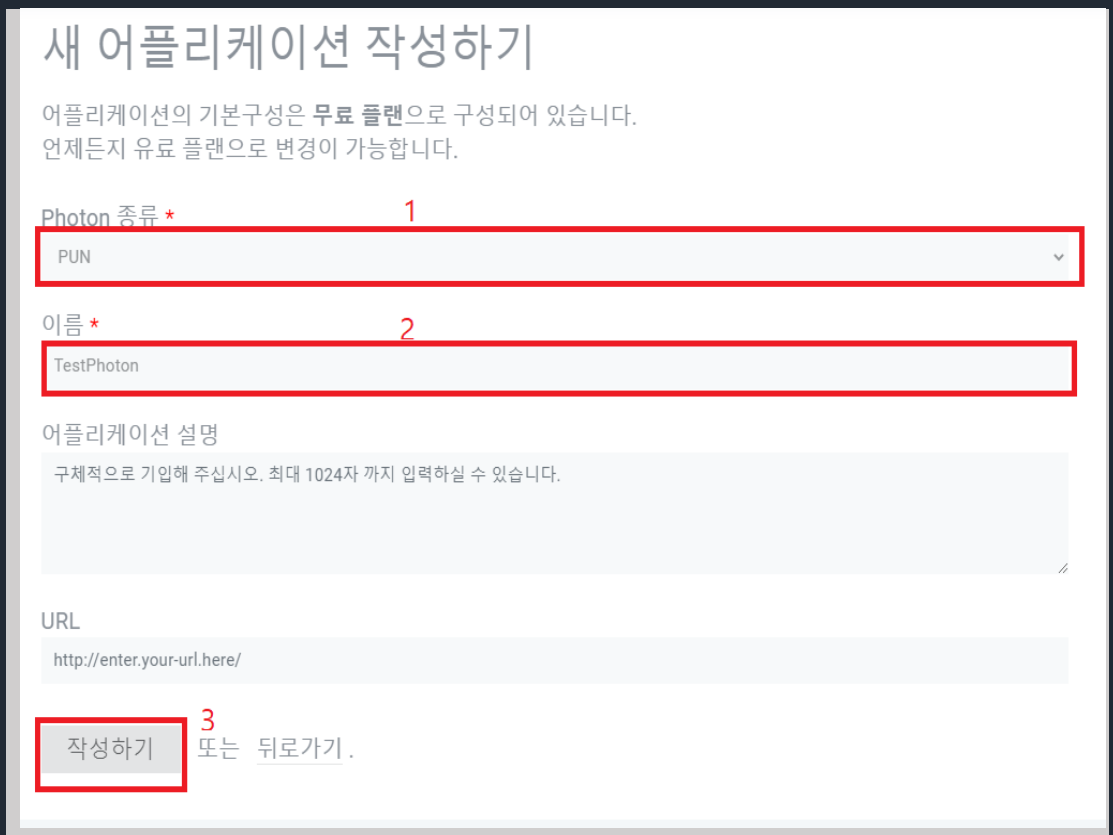


6. 제품 설정 > 카카오 로그인 > 동의항목에서 위처럼 설정

2. Photon Network 연동

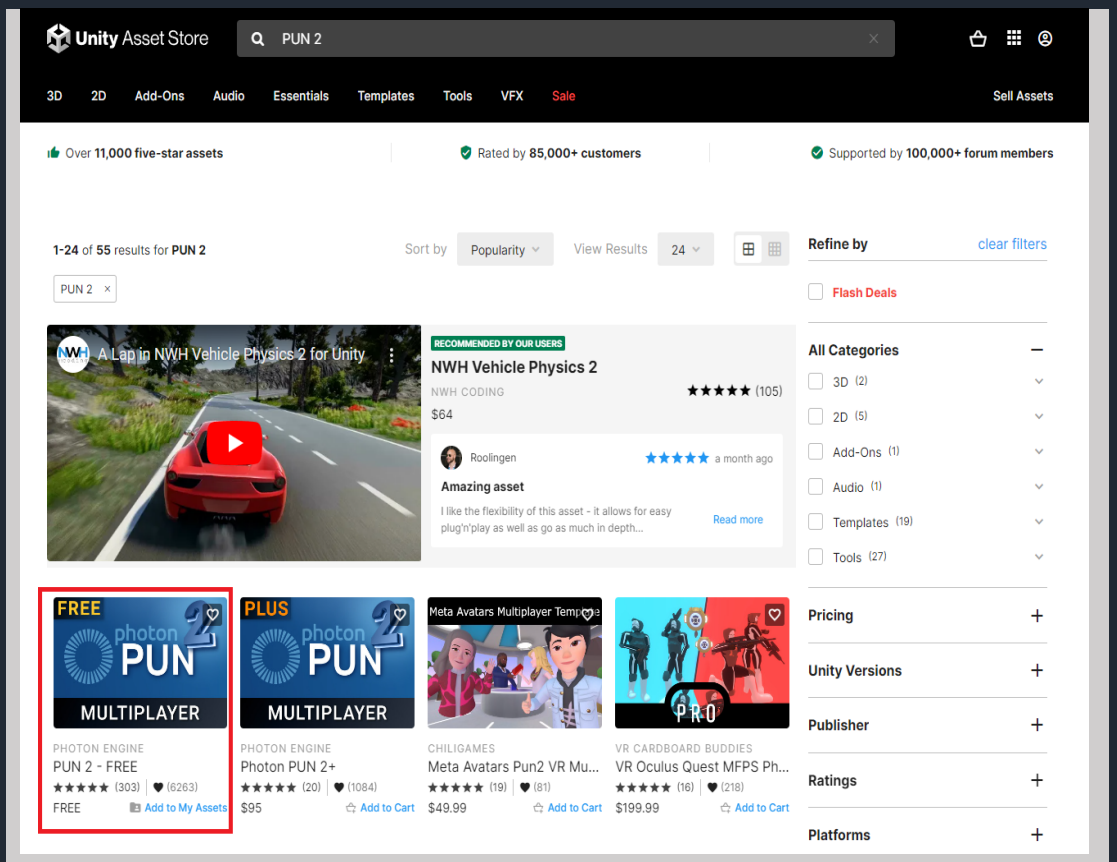
1. https://www.photonengine.com/ko-KR/ 에 접속하여 회원가입

2. 새 어플리케이션 만들기를 클릭

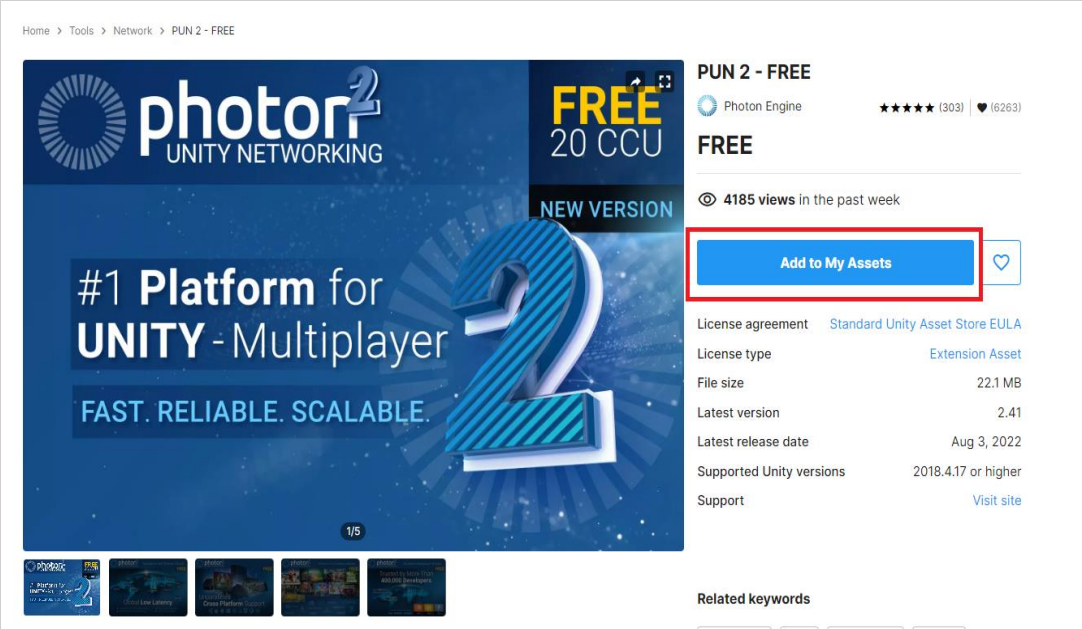


3. Photon 종류를 선택하고 어플레케이션 명을 입력 후 작성하기

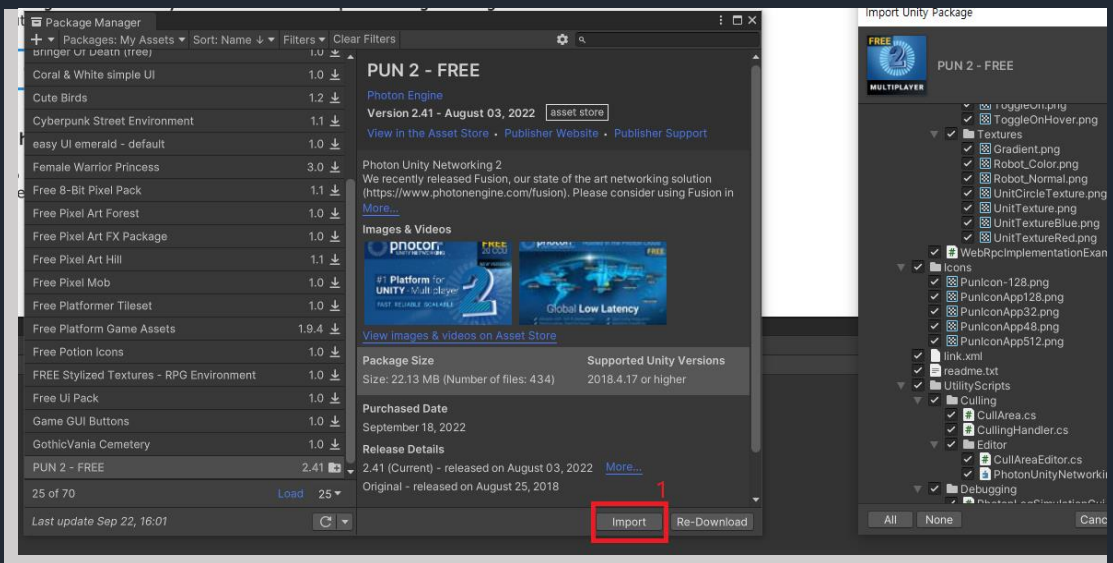
4. 어플리케이션 ID를 복사



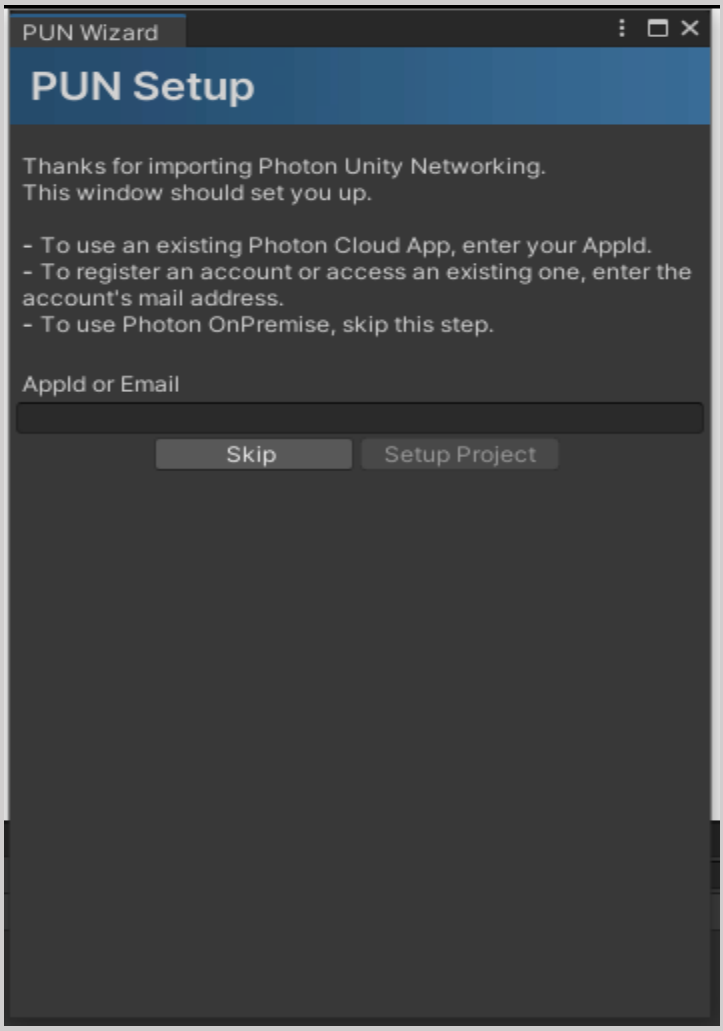
5. https://assetstore.unity.com/에서 PUN 2 검색



6. 나의 에셋에 추가



7. 프로젝트 화면에서 해당 에셋을 import



8. AppId or Email 칸에 아까 복사해둔 어플리케이션 ID 입력