



# 포팅 문서

## 목차

### 1. 개발 환경

### 2. 배포서버 환경 구성(CI/CD)

### 3. 외부 서비스 정보

### 4. 프로젝트에 활용되는 프로퍼티 파일



# 목차

1. 개발환경
2. 배포서버 환경 구축(CI/CD)
3. 외부 서비스 정보
4. 프로젝트에 활용되는 프로퍼티 파일



# 1. 개발 환경

## 형상관리

- Gitlab

## 이슈 관리

- Jira

## OS

- Window 10

## Communication

- Notion
- Discord
- Mattermost

## CI/CD

- Jenkins 2.387.1
- Docker 23.0.1

## UI/UX

- Figma

## 데이터 분석

- R 4.2.3
  - Library
    - Lubridate
    - Dplyr
    - Zoo
    - Tidy
    - Caret
    - RandomFor
    - Tensorflow
    - Keras
    - Torch
- AI
  - Python 3.8
  - Tensorflow
  - Keras
  - YOLOv5
  - Flask

## IDE

- IntelliJ 2022.3.1
- Android Studio 2022.1.1.21

## DataBase

- MariaDB 10.11.2
- Redis 7.0.10

## Server

- AWS EC2 Ubuntu 20.04.6 LTS
- Tomcat 9.0.71
- Nginx 1.15.12

## 기타 편의 툴

- PostMan

## Front-end 기술 스택

- Android Gradle Plugin 7.3.1
- Gradle 7.4
- Kotlin 1.8.10
- Jetpack Compose 2023.01
- Dagger-Hilt 2.45
- Google Map 18.1.0
- Firebase 31.2.3
- OpenWeather
- Exifinterface:1.3.6

## Back-end 기술 스택

- Java Open-JDK 11
- Gradle 7.6.1
- Spring boot 2.7.7
- Spring Data JPA
- Spring batch
- Spring Security
- Junit5
- Firebase
- OAuth2.0
- Rserve



## 2. 배포서버 환경 구성(CI/CD)

1. 서버 접속하기
  2. Docker 및 docker-compose 설치
  3. 젠킨스 설치 및 실행
  4. 젠킨스 세부 설정
  5. Gitlab WebHook 설정하기
  6. Nginx 설정하기(SSL인증)
  7. R Server 설정하기
  8. MariaDB 설정하기
  9. Redis 설정하기
  10. 최종 docker-compose.yml
  11. 프로젝트 내부에서 api-server, batch-server 설정하기
- [참고] 네트워크 확인 및 생성 추가

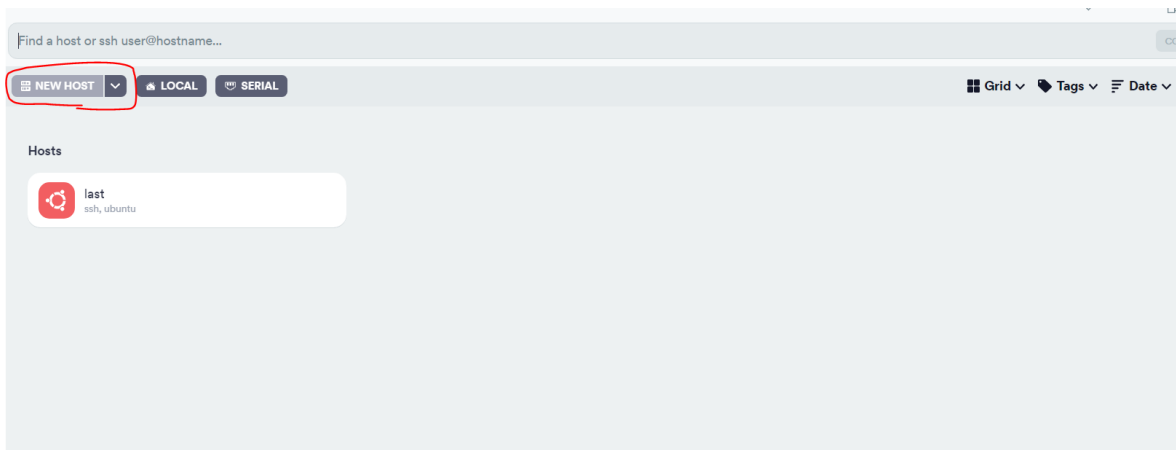
### ▼ 1. 서버 접속하기

Termius는 SSH, SFTP 같은 다양한 프로토콜을 지원하는 모바일 및 데스크톱 SSH 클라이언트입니다. Termius를 설치합니다.

#### Termius - SSH platform for Mobile and Desktop

Termius helps to organize the work of multiple DevOps and engineering teams. It reduces the admin work for managing users. Enterprise compliance. SOC2 II report.

<https://termius.com/>



terminus를 설치하면 위와 같은 화면이 생성되는데, 상단에 있는 new host를 클릭합니다.

**New Host**

Label: cicd-project

Address: 127.0.0.1

Parent group: Groups

Add a Tag

Backspace as Ctrl+H: ☐

Share this host

SSH: ☒

Port: 22

Username: ubuntu

Password:

Set a Key Set an Identity

SSH Agent Forwarding: ☐

Startup Snippet Snippets

Host Chaining Edit Chain

**New Key** SAVE

Label

Private key \*

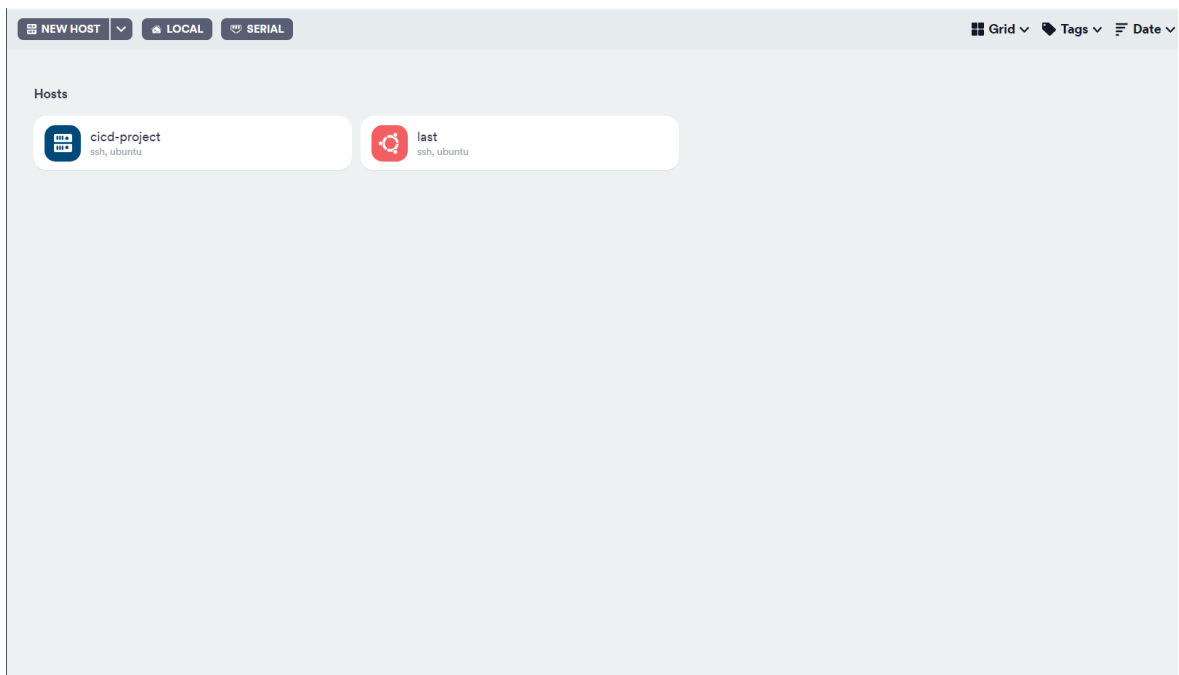
Public key

Certificate

Drag and drop a private key file to import

Import from key file

그러면 위와 같은 탭이 우측에 생성됩니다. Address에 서버의 ip주소를 적어줍니다. Port의 경우 기본적으로 SSH는 22번 포트를 사용합니다. 그리고 AWS에서 ubuntu 인스턴스를 생성하면 Username은 ubuntu로 설정되어 있습니다. password는 pem키를 사용할 것인데, password 아래에 있는 **Set a Key**를 클릭하고 **New Key**를 선택하면 오른쪽에 있는 화면이 나옵니다. 여기서 pem파일을 제일 하단의 영역에 드래그한 후 저장해줍니다.



이렇게 새로운 서버가 생성된 것을 확인할 수 있고, 더블 클릭을 통해서 원격 서버에 쉽게 접근할 수 있습니다.

## ▼ 2. Docker 및 docker-compose 설치

### 1. 패키지 업데이트 진행

```
sudo apt update & apt upgrade
```

### 2. Docker 설치에 필요한 필수 패키지 설치

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

### 3. Docker의 GPG key 인증

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

### 4. Docker Repository 등록

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

### 5. Docker 설치

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

### 6. docker-compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose # chmod를 통해서 권한 설정
docker-compose -v # docker-compose가 제대로 설치되는지 확인하기
```

### 7. Docker 실행

```
sudo systemctl enable docker # 서버가 실행될 때마다 docker가 자동으로 실행되도록 설정
```

```
sudo service docker start # docker를 실행하는 명령어
```

## ▼ 3. 젠킨스 설치 및 실행

### 1. jenkins Dockerfile 설정

```
FROM jenkins/jenkins:lts
USER root

RUN apt-get update \
  && apt-get -y install lsb-release \
  && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg \
  && echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $" \
  && apt-get update \
  && apt-get -y install docker-ce docker-ce-cli containerd.io
RUN usermod -u 1000 jenkins && \
  groupmod -g 998 docker && \
  usermod -aG docker jenkins
```

단순하게 jenkins이미지만 가지고 jenkins 컨테이너를 생성해도 되지만, 위처럼 설정하는 이유는 jenkins내에서도 docker를 사용하기 위함입니다. (docker.sock을 활용하여 통신)

```
RUN usermod -u {사용자 id} jenkins && groupmod -g {도커 그룹 아이디} docker && usermod -aG docker jenkins
```

사용자 id는 `id -u` 명령어를 통해서 확인이 가능하며, 도커 그룹 아이디의 경우 `cat /etc/group | grep docker` 를 통해서 확인이 가능합니다.

본 프로젝트의 경우 Rserver의 경우에도 Dockerfile을 작성하기때문에, 파일이름을 `Dockerfile_jenkins` 로 설정해줍니다.

## 2. 네트워크 생성

```
docker network create --gateway [ip주소] --subnet [ip주소] [네트워크명]

ex.
docker network create --gateway 172.19.0.1 --subnet 172.19.0.0/24 D106-network
```

하나의 docker-compose.yml안에 설정된 컨테이너들은 자동으로 동일한 네트워크를 사용하지만, 다른 docker-compose.yml에 설정된 컨테이너들은 다른 네트워크를 사용하기 때문에 다른 docker-compose.yml에 설정된 컨테이너들과 통신하기 위해서 미리 네트워크를 생성해둡니다.

## 3. 볼륨 생성

```
docker volume create jenkins
```

## 4. docker-compose.yml작성

```
version: '3.2'
services:
  jenkins:
    container_name: jenkins
    build:
      context: . # Dockerfile의 위치는 현재 경로
      dockerfile: Dockerfile_jenkins # Dockerfile의 파일명
    volumes:
      - jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /usr/bin/docker:/usr/bin/docker
    ports:
      - "8080:8080"
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.2 # 원래는 자동으로 할당되지만, 다른 컨테이너와 통신을 위해서 직접 설정
networks:
  D106-network:
    external: true # 외부 네트워크 설정
volumes:
  jenkins:
```

## 5. 컨테이너 생성

```
docker-compose up -d
```

-d 옵션은 background에서 실행한다는 설정입니다. Dockerfile변경 이후에 새롭게 빌드하고 싶다면 `--build` 옵션을 추가해서 실행합니다.

# ▼ 4. 젠킨스 세부 설정

## 1. http://{server host}:8080으로 접속하기

젠킨스는 기본적으로 8080 port를 사용하기 때문에 `서버ip:8080` 으로 접속합니다.

## 2. 젠킨스 비밀번호 입력하기



## Getting Started

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

젠킨스에 처음 접속하면, 위와 같은 페이지가 나오면서 초기 비밀번호를 입력하라고 하는데, 해당 비밀번호를 설정하는 방법은 2가지가 있다.

첫번째 방법:

```
docker logs jenkins
```

```
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
*****
*****
*****
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
*****
*****
```

젠킨스 컨테이너의 로그를 보면 관리자 계정의 비밀번호를 확인할 수 있다.

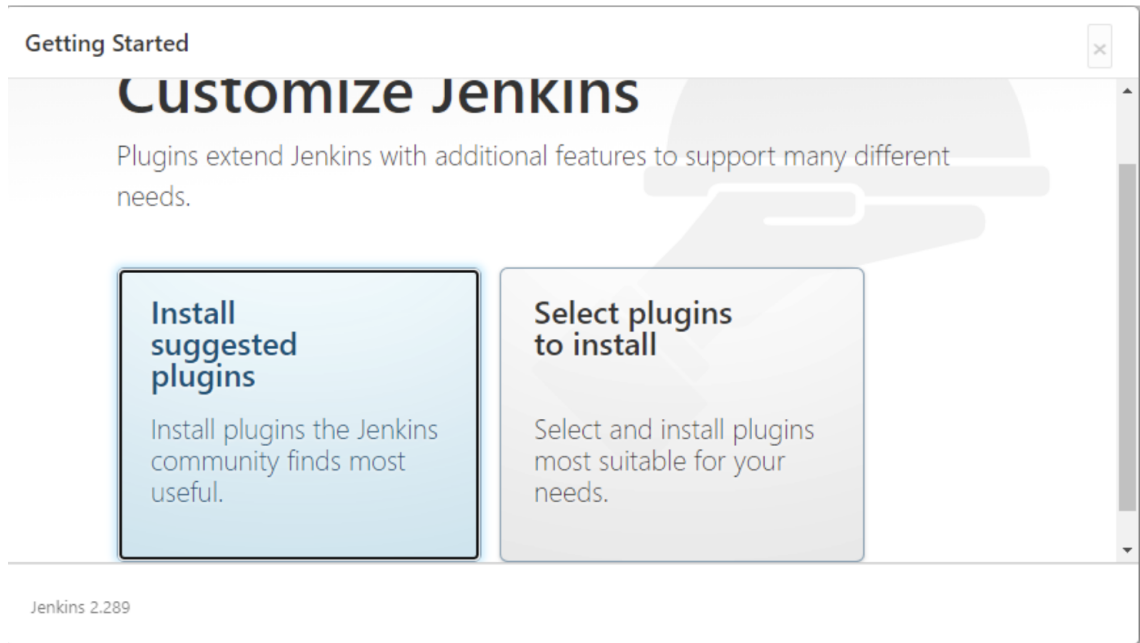
두번째 방법:

```
docker exec -it jenkins /bin/bash
```

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

먼저 jenkins 컨테이너에 접속하고, 아래 경로에서 비밀번호를 확인합니다.

3. 비밀번호를 입력하면, 아래와 같은 화면이 나오는데, Install suggested plugins를 선택합니다.



4. 설치 후 계정 정보를 입력하는 부분이 나오는데, 본인이 설정하고 싶은 대로 설정합니다. (이후 젠킨스 접속할 때마다 입력하는 정보)

5. 플러그인 추가 설치

젠킨스 관리 → plugin Manage로 접근합니다.

우리가 설치해야 할 플러그인에는 Gitlab, Gradle이 있습니다.

초기 설정에서 install suggested plugins를 선택했다면, Gradle의 경우 이미 설치되었을 것이기 때문에 Gitlab plugin을 설치 하고 jenkins를 재시작해줍니다.

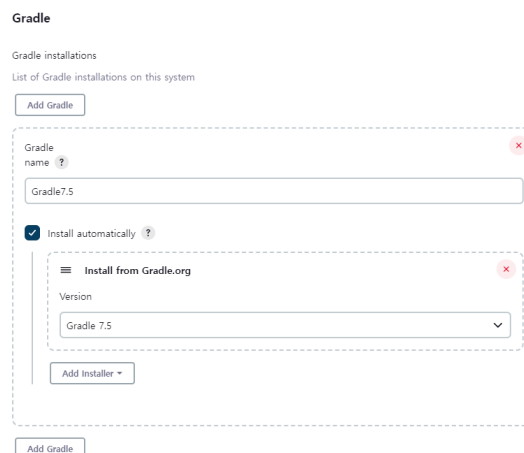
만약 jenkins페이지에서 재시작이 정상적으로 동작하지 않는다면, 아래 명령어를 통해서 jenkins컨테이너 재시작해줍니다.

```
docker restart {container_id 또는 container name}
```

6. gradle 버전 설정

Jenkins관리 → Global Tool Configuration으로 접근합니다.

해당 페이지에서 Gradle을 아래와 같이 설정해줍니다. (프로젝트 gradle버전과 통일)



7. Time Zone 설정

Jenkins 관리 → Manage Users → admin 수정 → User Defind Time Zone

Asia/Seoul로 변경하기

## 8. jenkins 내부에 docker-compose 설치하기

jenkins에서 docker-compose사용하기 위하여 docker-compose를 설치합니다.

```
docker exec -it {jenkins container 이름} /bin/bash # jenkins에 접속
```

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose -v
```

## ▼ 5. Gitlab WebHook 설정하기

### 1. Credential등록

gitlab webhook을 등록할 때 필요한 credential을 등록합니다.

credential을 생성하기 위해서는 먼저 gitlab의 accessToken이 필요합니다. gitlab프로젝트 → setting → accessToken에서 아래처럼 access token을 생성합니다.

#### Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API.

You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

#### Add a project access token

Enter the name of your application, and we'll return a unique project access token.

##### Token name

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

##### Expiration date

##### Select a role

##### Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☐ api  
Grants complete read and write access to the scoped project API, including the Package Registry.
- ☐ read\_api  
Grants read access to the scoped project API, including the Package Registry.
- ☐ read\_repository  
Grants read access (pull) to the repository.
- ☐ write\_repository  
Grants read and write access (pull and push) to the repository.

Create project access token

credential을 등록하기 위해서 jenkins관리 → Manage Credential에 들어갑니다. 그리고 System → Global credentials(unrestricted)로 접근하고 상단에 있는 **Add Credentials** 버튼을 클릭합니다.

Dashboard > Jenkins 관리 > Credentials > System > Global credentials (unrestricted) >

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

GitLab username

☐ Treat username as secret ?

Password ?

Access token

ID ?

Gitlab ID

Description ?

Create

username은 gitlab username을 적고, password는 발급 받은 access token을 입력합니다. ID는 credential을 구분하기 위한 값으로 gitlab ID를 입력해줍니다.

## 2. 프로젝트 생성


### a. 새로운 Item 생성


item이름을 선택하고 Freestyle project를 선택합니다.

Enter an item name

campinity-develop2

» Required field

 **Freestyle project**  
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

 **Pipeline**

b. 소스코드 관리 목록에서 GIT을 선택하고 jenkins에 연결하고 싶은 repository의 경로와 위 과정에서 생성해둔 Credential을 설정해줍니다. 그리고 빌드를 유발시킬 브랜치를 설정합니다.

Git ?

Repositories ?

Repository URL ? ×

[Redacted URL] Git repo 경로

Credentials ?

[Redacted Credentials] 생성한 Credentials 설정 ▼

Add ▼

고급 ▼

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ? ×

\*/develop\_be 빌드를 유발시킬 브랜치 입력

- c. 빌드 유발 설정(이번 프로젝트에서는 mr이 발생한 경우에만 jenkins로 빌드를 할 예정이기 때문에, 빌드 유발을 아래와 같이 설정합니다. (빌드 유발 제일 아래에 보면 고급이라는 버튼이 있는데, 이 부분은 나중에 gitlab에서 webhook을 설정할 때 사용합니다.))

### 빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://j8d106.p.ssafy.io:8080/project/nassafy> ?

Enabled GitLab triggers

☐ Push Events

☐ Push Events in case of branch delete

☐ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request ?

☒ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

☐ Approved Merge Requests (EE-only)

☐ Comments

Comment (regex) for triggering a build ?

고급 ▾

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

#### d. 빌드 스텝

먼저 gradle을 통해서 spring boot 프로젝트를 빌드합니다. 그리고 shell을 통해서 docker-compose를 통해서 컨테이너를 생성합니다.

### Build Steps

Invoke Gradle script ?

Invoke Gradle ?

Gradle Version

Gradle7.6.1

☐ Use Gradle Wrapper ?

Tasks ?

build -p /var/jenkins\_home/workspace/nassafy/server/aro

Execute shell ?

Command

See [the list of available environment variables](#)

cd server  
cd aro  
docker-compose up --build -d

Add build step ▾

### 3. WebHook 설정하기

- a. gitlab repository → setting → webhook으로 들어가면 아래와 같은 페이지가 나옵니다.

## Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

`http://example.com/trigger-ci.json`

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

### Trigger

- ☒
- Push events

Branch name or wildcard pattern to trigger on (leave blank for all)

Push to the repository.

- ☐
- Tag push events

A new tag is pushed to the repository.

- Comments

A comment is added to an issue or merge request.

- ☐
- Confidential comments

A comment is added to a confidential issue.


- ☐
- Issues events

An issue is created, updated, closed, or reopened.

b. webhook url정보 적기

webhook의 url정보는 jenkins 빌드 유발에 나와 있는 webhook url을 입력해줍니다.

### 빌드 유발

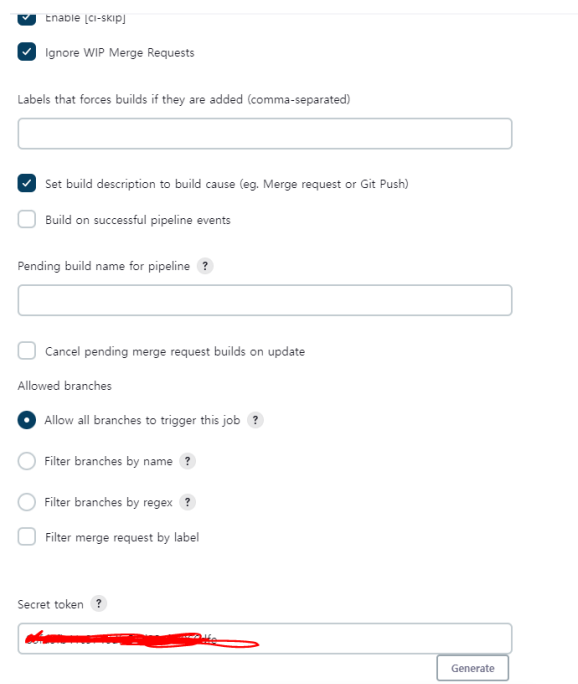
- ☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL:  


#### Enabled GitLab triggers

- ☐ Push Events
- ☐ Push Events in case of branch delete
- ☐ Opened Merge Request Events
- ☐ Build only if new commits were pushed to Merge Request ?
- ☒ Accepted Merge Request Events
- ☐ Closed Merge Request Events

c. webhook secret token

secret key는 빌드 유발의 고급 탭에서 생성할 수 있습니다. 고급 버튼을 클릭하고, 다른 설정은 그대로 둔 후 generate버튼을 클릭하면 secret token이 생성됩니다.



The image shows the 'Advanced' tab of the Jenkins GitLab webhook configuration. It includes several checkboxes: 'enable [ci-skip]' (checked), 'Ignore WIP Merge Requests' (checked), 'Set build description to build cause (eg. Merge request or Git Push)' (checked), and 'Build on successful pipeline events' (unchecked). There is a text input field for 'Labels that forces builds if they are added (comma-separated)'. Below that is 'Pending build name for pipeline' (unchecked) with a text input field. Then 'Cancel pending merge request builds on update' (unchecked). Under 'Allowed branches', 'Allow all branches to trigger this job' is selected. Other options like 'Filter branches by name', 'Filter branches by regex', and 'Filter merge request by label' are unselected. At the bottom, there is a 'Secret token' field with a redacted value and a 'Generate' button.

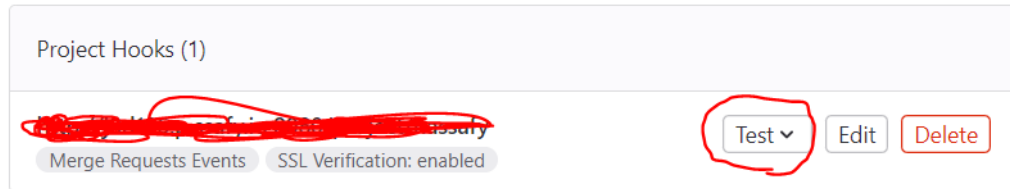


#### d. webhook trigger

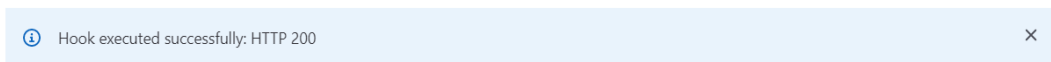
trigger의 경우 이번 프로젝트에서는 merge request가 발생했을 때, 다시 빌드하기 때문에 merge request로 설정하고, 각자 상황에 맞게 선택합니다.

#### e. webhook test

이렇게 webhook을 설정하면 아래처럼 webhook이 생성되고, test탭이 생성됩니다.



test에서 merge request를 선택한 후 아래처럼 200 응답이 오면 정상적으로 webhook이 설정된 것입니다.



## ▼ 6. Nginx 설정하기(SSL인증)

### 1. docker-compose.yml 작성하기

https를 적용하기 위해서는 nginx와 certbot이 필요합니다. certbot의 경우 letsencrypt에서 무료 SSL을 자동으로 발급받기 위한 클라이언트입니다. docker-compose.yml을 아래처럼 작성합니다.

certbot으로 발급 받는 SSL의 경우, 90일마다 만료되기 때문에 자동으로 재발급 받도록 해주어야 합니다. 따라서 entypoint를 아래처럼 작성하여 자동으로 재발급받도록 합니다.

```
version: '3.2'
services:
  nginx:
    container_name: nginx
    image: nginx:1.15-alpine
    volumes:
      - ./data/nginx:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    networks:
      D106-network:
        ipv4_address: 172.19.0.5
    ports:
      - "80:80"
      - "443:443"
    command: "/bin/sh -c 'while ;; do sleep 6h & wait $$(!); nginx -s reload; done & nginx -g \"daemon off;\""
  certbot:
    container_name: certbot
    image: certbot/certbot
    restart: unless-stopped
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    networks:
      D106-network:
        ipv4_address: 172.19.0.6
    entypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h & wait $$(!); done;'"
networks:
  D106-network:
    external: true
```

### 2. data/nginx 경로에 app.conf 파일 작성하기

아래처럼 app.conf를 작성하여 nginx를 설정합니다.

아래 설정은 80번 포트로 요청이 들어오면 443번 포트로 리다이렉트를 시키며, 443으로 들어온 경우 ssl인증을 처리하고 host\_name이하 경로가 /api 인 경우에는 http://{host\_name}:8000/api 로 리다이렉트 시키고, /flask 인 경우에는 http://{host\_name}:9000/flask 로 리다이렉트 시킵니다.

```
server {
    listen 80;
    server_name {host_name};
    server_tokens off;
    client_max_body_size 1G; # media type 파일 용량 설정

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name {host_name};
    server_tokens off;
    client_max_body_size 1G;

    ssl_certificate /etc/letsencrypt/live/{host_name}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/{host_name}/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location /api {
        proxy_pass http://{host_name}:8000/api;
        proxy_set_header    Host            $http_host;
        proxy_set_header    X-Real-IP        $remote_addr;
        proxy_set_header    X-Forwarded-For  $proxy_add_x_forwarded_for;
    }

    location /flask {
        proxy_pass http://{host_name}:9000/flask;
        proxy_set_header    Host            $http_host;
        proxy_set_header    X-Real-IP        $remote_addr;
        proxy_set_header    X-Forwarded-For  $proxy_add_x_forwarded_for;
    }
}
```

### 3. `init-letsencrypt.sh` 파일 작성 및 실행하기

nginx를 실행하기 위해서는 ssl인증서가 필요하지만, ssl인증서를 받기 위해서는 nginx를 실행해야 합니다. 따라서 처음 실행할 때, dummy certificate를 받아서 실행해야 합니다. `init-letsencrypt.sh` 는 dummy certificate를 받아오기 위한 설정이 담긴 파일입니다. 아래 경로에서 해당 코드를 가져옵니다.

```
curl -L https://raw.githubusercontent.com/wmnd/nginx-certbot/master/init-letsencrypt.sh > init-letsencrypt.sh
```

해당 파일을 다운받고, `init-letsencrypt.sh` 파일을 열어서 domain명과 email을 본인에게 맞게 수정해줍니다.

```
chmod +x init-letsencrypt.sh # 권한 설정
```

```
sudo ./init-letsencrypt.sh # 실행
```

그 후 해당 파일의 권한을 설정해주고, 실행합니다. 그러면 `/data/certbot` 에 관련 파일들이 생성된 것을 확인할 수 있습니다.

### 4. nginx실행하기

```
docker-compose up -d
```


### 5. test

이후 `http://{host_name}` 로 접속해도 `https://{host_name}` 로 잘 접속이 되는지 확인합니다.

### 6. 참고자료

### Nginx and Let's Encrypt with Docker in Less Than 5 Minutes


Getting Nginx to run with Let's Encrypt in a docker-compose environment is more tricky than you'd think ...

 <https://pentacent.medium.com/nginx-and-lets-encrypt-with-docker-in-less-than-5-minutes-b4b8a60d3a71>



### GitHub - wmnnd/nginx-certbot: Boilerplate configuration for nginx and certbot with docker-compose

Boilerplate configuration for nginx and certbot with docker-compose - GitHub - wmnnd/nginx-certbot: Boilerplate configuration for nginx and certbot with docker-compose

 <https://github.com/wmnnd/nginx-certbot>

### wmnnd/nginx-certbot

Boilerplate configuration for nginx and certbot with docker-compose

 8 Contributors  36 Issues  3k Stars  1k Forks

## ▼ 7. R Server 설정하기

### 1. Rstudio Dockerfile 생성하기

```
FROM rocker/rstudio:latest

# TensorFlow 및 Keras 설치
RUN apt-get update && apt-get install -y \
    python3-pip \
    libpython3-dev \
    python3-venv \
    && python3 -m venv /opt/venv \
    && . /opt/venv/bin/activate \
    && pip3 install --upgrade pip \
    && pip3 install tensorflow \
    && pip3 install keras

# R 패키지 설치
RUN R -e "install.packages(c('Rserve', 'tensorflow', 'keras'), repos='http://cran.us.r-project.org/')"

# Rserve 실행
CMD ["R", "-e", "Rserve::run.Rserve(remote=TRUE)", "--no-save", "--slave"]
```

본 프로젝트에서는 Spring batch에서 R server에 접속해서 R script를 실행시켜야 했기 때문에 Rserve를 설치해주고, tensorflow, keras를 사용하는 R script를 실행시킬 것이기 때문에 파이썬 가상환경을 만들어서 tensorflow와 keras를 설치해줍니다. 그리고 CMD를 설정해서 컨테이너 실행시 자동으로 Rserve를 통해서 커넥션을 열어두도록 합니다. Rserve서버와 R 클라이언트가 서로 다른 ip에서 실행되는 경우에는 remote=TRUE 옵션을 주어야 접속이 가능합니다.

### 2. docker-compose.yml 설정하기

```
version: '3.2'
services:
  rstudio:
    container_name: rstudio
    build:
      context: .
      dockerfile: Dockerfile_rstudio
    volumes:
      - ./rstudio:/home/rstudio
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.4
    ports:
      - "8787:8787"
      - "6311:6311"
networks:
  D106-network:
    external: true
```

### 3. 컨테이너에 접속해서 6311포트가 열려 있는지 확인하기

Rserve는 기본적으로 6311포트를 사용합니다. 아래 과정을 통해서 6311포트가 열려 있는지 확인합니다.

```
sudo docker exec -it rstudio /bin/bash
```

```
apt update
apt upgrade
apt install net-tools
sudo netstat -plnut
```

#### 4. R 콘솔에서 파이썬 가상환경 활성화 및 필요한 패키지 설치 및 로드

```
sudo docker exec -it rstudio /bin/bash # R 컨테이너 접속
```

```
R # R 컨테이너 접속 후 해당 명령어를 통해서 R 콘솔 활성화
```

```
install.packages("reticulate") # 가상환경 설정을 위한 패키지 설치 및 로드
library(reticulate)
```

```
use_virtualenv("/opt/venv") # Dockerfile에서 설정해준 파이썬 가상환경 활성화
```

```
install.packages("tensorflow")
install.packages("keras")

library(tensorflow)
library(keras)
```

#### 5. Working Directory에 R Script 생성

기본적으로 WD는 `/home/rstudio` 입니다. 해당 폴더에 우리가 실행시키고자 하는 파일을 생성해둡니다.

#### 6. R client(Spring)에서 접속 및 스크립트 실행하기

##### a. 의존성 추가

```
implementation group: 'org.rosuda.REngine', name: 'Rserve', version: '1.8.1'
```

##### b. connection 연결 및 스크립트 파일 실행

```
RConnection conn = null;
double[] x;
try {
    conn = new RConnection("rstudio", 6311); // 로컬에서는 host를 서버 ip로, 서버 환경에서는 rstudio컨테이너명으로 변경
    REXP exp = conn.eval("source('/home/rstudio/getAceLocData2.R')");
    RList rList = conn.eval("getAceLocData()").asList();

    x = rList.at("GES_X").asDoubles(); // R 실행결과를 컬럼단위로 접근 가능

} catch (RserveException | REXPMismatchException e) {
    throw new RuntimeException(e);
} finally {
    if (conn != null) {
        conn.close();
    }
}
```

## ▼ 8. MariaDB 설정하기

### 1. docker-compose.yml 설정하기

```

version: '3.2'
services:
  database:
    image: mariadb
    container_name: database
    volumes:
      - mariadb:/var/lib/mysql
    restart: always
    environment:
      TZ: Asia/Seoul
      MYSQL_ROOT_PASSWORD: root 비밀번호
      MYSQL_DATABASE: database 이름
    command: ['--character-set-server=utf8mb4',
              '--collation-server=utf8mb4_unicode_ci']
    ports:
      - "3306:3306"
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.3
networks:
  D106-network:
    external: true
volumes:
  mariadb:

```

## 2. 권한 설정

```
docker exec -it database /bin/bash
```

```
mysql -u root -p # 이렇게 입력하면 비밀번호 입력창이 나오는데, docker-compose.yml에 설정한 root비밀번호를 입력합니다.
```

```
use mysql
```

```
SELECT host, user, password FROM user;
```

위 명령어를 입력하면 아래 사진처럼 현재 db에 접근할 수 있는 ip들을 확인할 수 있습니다. 아래 사진을 보면 host가 %로 설정되어 있는 것을 확인할 수 있는데, 이것은 모든 ip에서 root로 접속을 허용한다는 의미입니다. 이렇게 해주면 나중에 해킹을 당할 수 있는 위험이 크기 때문에 해당 host를 삭제합니다. (주의!! localhost는 없애면 안됩니다. → 없애면 서버에서 db에 접근이 불가능해집니다.)

```

MariaDB [mysql]> select user, host, password FROM user;
+-----+-----+-----+
| User      | Host      | Password                                     |
+-----+-----+-----+
| mariadb.sys | localhost |                                             |
| root       | localhost | *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B |
| root       | %         | *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B |
+-----+-----+-----+
3 rows in set (0.002 sec)

```

```
drop user root@'%'; # root에 모든 곳에서 접근 가능하도록 하는 설정을 삭제
```

아래 명령어를 통해서 각각 사용하는 ip주소에 대해서 접속 권한을 부여합니다. (주의! 보통 사용하는 wifi가 달라지면 ip주소도 달라지기 때문에 해당 부분을 고려해서 권한을 부여합니다. 또한 각 컨테이너들 역시 해당 권한을 따로 다 설정해주어야 합니다. )

```
grant all privileges on *.* to root@'ip주소';
```

## ▼ 9. Redis 설정하기

### 1. docker-compose.yml 설정하기

```
version: '3.2'
services:
  redis:
    hostname: redis
    container_name: redis
    image: redis:alpine
    command: redis-server /usr/local/etc/redis/redis.conf --requirepass 비밀번호 --port 6379
    environment:
      TZ: "Asia/Seoul"
      REDIS_PASSWORD: 비밀번호
    ports:
      - 6379:6379
    volumes:
      - ./redis/redis.conf:/usr/local/etc/redis/redis.conf
      - redis:/data
    networks:
      D106-network:
        ipv4_address: 172.19.0.8
networks:
  D106-network:
    external: true
volumes:
  redis:
```

### 2. redis.conf 설정(서버에 redis폴더를 만들고 해당 폴더에 redis.conf파일 생성)

```
bind 0.0.0.0 # 모든 ip에서 접근 가능하도록 설정

protected-mode yes

requirepass {비밀번호} # 비밀번호 설정

port 6379

dir /data

save 900 1

save 300 10

save 60 10000
```

## ▼ 10. 최종 docker-compose.yml

```
version: '3.2'
services:
  jenkins:
    container_name: jenkins
    build:
      context: .
      dockerfile: Dockerfile_jenkins
    volumes:
      - jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /usr/bin/docker:/usr/bin/docker
    ports:
      - "8080:8080"
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.2
  database:
```

```

image: mariadb
container_name: database
volumes:
  - mariadb:/var/lib/mysql
restart: always
environment:
  TZ: Asia/Seoul
  MYSQL_ROOT_PASSWORD: ssafy1234!
  MYSQL_DATABASE: nassafy
command: ['--character-set-server=utf8mb4',
  '--collation-server=utf8mb4_unicode_ci']

ports:
  - "3306:3306"
logging:
  driver: "json-file"
  options:
    max-file: "5"
    max-size: "100000000"
networks:
  D106-network:
    ipv4_address: 172.19.0.3
rstudio:
  container_name: rstudio
  build:
    context: .
    dockerfile: Dockerfile_rstudio
  volumes:
    - ./rstudio:/home/rstudio
  logging:
    driver: "json-file"
    options:
      max-file: "5"
      max-size: "100000000"
  networks:
    D106-network:
      ipv4_address: 172.19.0.4
  ports:
    - "8787:8787"
    - "6311:6311"
nginx:
  container_name: nginx
  image: nginx:1.15-alpine
  volumes:
    - ./data/nginx:/etc/nginx/conf.d
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  networks:
    D106-network:
      ipv4_address: 172.19.0.5
  ports:
    - "80:80"
    - "443:443"
  command: "/bin/sh -c 'while ;; do sleep 6h & wait $$(!); nginx -s reload; done & nginx -g \"daemon off;\""
certbot:
  container_name: certbot
  image: certbot/certbot
  restart: unless-stopped
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  networks:
    D106-network:
      ipv4_address: 172.19.0.6
  entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h & wait $$(!); done;'"
redis:
  hostname: redis
  container_name: redis
  image: redis:alpine
  command: redis-server /usr/local/etc/redis/redis.conf --requirepass ssafy1234! --port 6379
  environment:
    TZ: "Asia/Seoul"
    REDIS_PASSWORD: ssafy1234!
  ports:
    - 6379:6379
  volumes:
    - ./redis/redis.conf:/usr/local/etc/redis/redis.conf
    - redis:/data
  networks:
    D106-network:
      ipv4_address: 172.19.0.8
networks:
  D106-network:
    external: true
volumes:
  jenkins:
  mariadb:
  redis:

```

위의 모든 컨테이너 설정들을 모으면 위와 같이 하나의 docker-compose.yml파일을 작성할 수 있습니다.

## ▼ 11. 프로젝트 내부에서 api-server, batch-server 설정하기

본 프로젝트는 멀티 모듈 환경에서 구성하였습니다.

### 1. 각 모듈의 최상위 폴더에 Dockerfile설정하기

#### a. api-server

```
FROM openjdk:11 as build

ARG JAR_FILE=./build/libs/*.jar

ADD ${JAR_FILE} app.jar

ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "/app.jar"]
```

#### b. batch-server

```
FROM openjdk:11 as build

ARG JAR_FILE=./build/libs/*.jar

ADD ${JAR_FILE} batch.jar

ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "/batch.jar"]
```

### 2. 프로젝트 최상위 폴더에 docker-compose.yml설정하기

```
version: '3.2'
services:
  web-server:
    container_name: api-server
    build: ./api-module
    ports:
      - "8000:8000"
    environment:
      SPRING_DATABASE_URL: jdbc:mariadb://database:3306/{database name}
      SPRING_DATABASE_USERNAME: root
      SPRING_DATABASE_PASSWORD: database 비밀번호
      TZ: Asia/Seoul
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.7
  batch-server:
    container_name: batch-server
    build: ./batch-module
    ports:
      - "8001:8001"
    environment:
      SPRING_DATABASE_URL: jdbc:mariadb://database:3306/{database name}
      SPRING_DATABASE_USERNAME: root
      SPRING_DATABASE_PASSWORD: database 비밀번호
      TZ: Asia/Seoul
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "100000000"
    networks:
      D106-network:
        ipv4_address: 172.19.0.9
networks:
  D106-network:
    external: true
```



## ▼ [참고] 네트워크 확인 및 생성 추가

1. docker network 목록 확인하기

```
docker network ls
```

2. 컨테이너 network 확인하기(제일 아래 쪽에서 확인 가능)

```
docker container inspect [컨테이너명]
```

3. network 생성하기

```
docker network create [네트워크명]
```

4. network gateway 설정 및 subnet 설정하기

```
docker network create --gateway [ip주소] --subnet [ip주소] [네트워크명]
```

5. run 명령어에서 network 설정하기(`-network` 옵션 사용)

```
docker run --network [네트워크명] ~~~~~
```

6. 이미 실행중인 컨테이너에 네트워크 추가하기

```
docker network connect [네트워크명] [컨테이너명]
```

- 컨테이너들은 같은 네트워크에 소속되어 있어야 컨테이너명을 통해서 서로 통신할 수 있습니다. ip 주소의 경우 유동적으로 변하기 때문에 ip주소를 사용하기에는 한계가 있습니다.



## 3. 외부 서비스 정보

### 1. 소셜인증 로그인

#### 1. NAVER

[애플리케이션 등록](#)

[회원 정보 요청하기](#)

#### 2. GITHUB

[애플리케이션 등록](#)

### 2. FCM 설정

[Firebase 앱 등록 및 프로젝트 설정](#)

[프로젝트 생성](#)

[앱 추가하기](#)

## 1. 소셜인증 로그인

### 1. NAVER

네이버 아이디 로그인

<https://developers.naver.com/main/>

### 애플리케이션 등록

#### 1. 애플리케이션 등록

<https://developers.naver.com/apps/#/register>

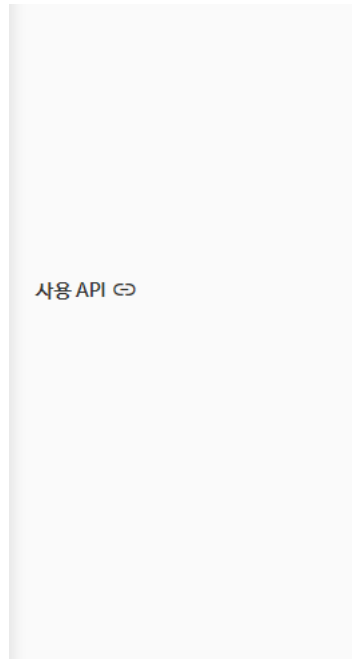
## 애플리케이션 등록 (API 이용신청)

애플리케이션의 기본 정보를 등록하면, 좌측 **내 애플리케이션** 메뉴의 서브 메뉴에 등록하신 애플리케이션 이름으로 서브 메뉴가 만들어집니다.

애플리케이션 이름 ⇄	<div>애플리케이션 이름 ⓘ</div> <ul style="list-style-type: none"><li>• 네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요.</li><li>• 40자 이내의 영문, 한글, 숫자, 공백문자, 쉼표(,), "/" , "-" , "_" , 만 입력 가능합니다.</li></ul>
사용 API ⇄	<div>선택하세요. ▼ ⓘ</div> <p>사용할 API를 추가해 주세요.</p>

#### 2. 사용 API - 네이버 로그인

#### 3. 제공할 데이터 선택



#### 네이버 로그인

제공 정보 선택(이용자 식별자는 기본 정보로 제공) ②

필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보를 선택해야 합니다.

권한	필수	추가
회원이름	<input type="checkbox"/>	<input type="checkbox"/>
연락처 이메일 주소	<input type="checkbox"/>	<input type="checkbox"/>
별명	<input type="checkbox"/>	<input type="checkbox"/>
프로필 사진	<input type="checkbox"/>	<input type="checkbox"/>
성별	<input type="checkbox"/>	<input type="checkbox"/>
생일	<input type="checkbox"/>	<input type="checkbox"/>
연령대	<input type="checkbox"/>	<input type="checkbox"/>
출생연도	<input type="checkbox"/>	<input type="checkbox"/>
휴대전화번호	<input type="checkbox"/>	<input type="checkbox"/>

×

#### 4. 로그인 오픈 API 서비스 환경

- 안드로이드



#### 안드로이드

##### 다운로드 URL

구글 플레이, 네이버 앱스토어 등 다운로드 할 수 있는 URL

불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.

##### 안드로이드 앱 패키지 이름

안드로이드 앱의 패키지 이름을 입력하세요 : 예) com.nhn.android.sam

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다. 2015년 8월 7일 이전에 등록된 앱은 앱 패키지 이름 대신 안드로이드 Intent가 등록되어 있어도 동작됩니다.

반드시 검수에 통과되어야 네이버 로그인 사용이 가능합니다. [사전 검수 관련 공지사항](#)을 확인하세요.

- [애플리케이션 이름] 설정을 확인해 주세요.
- [로그인 오픈 API > 안드로이드 > 다운로드 URL] 설정을 확인해 주세요.
- [로그인 오픈 API > 안드로이드 > 안드로이드 앱 패키지 이름] 설정을 확인해 주세요.

- PC 웹

로그인 오픈 API  
서비스 환경 ②

PC 웹

서비스 URL

서비스 URL예시: (O) http://naver.com (X) http://www.naver.com

서비스 URL예시: (O) http://naver.com (X) http://www.naver.com  
 서비스 URL값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.  
 불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.  
 서비스하려는 사이트 URL과 동일한 사이트 URL로 해주셔야 **네이버 로그인 배지**가 노출됩니다.

네이버 로그인  
Callback URL (최대 5개)

Callback URL예시: http://YOUR\_DOMAIN/Api/Member/Oauth2C

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.  
 Callback URL은 네이버 로그인 후 이동할 페이지 URL입니다. Callback URL값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.  
 입력한 주소와 다른 Callback URL로 리다이렉트 될 경우, 이용이 제한될 수 있습니다.

반드시 검수에 통과되어야 네이버 로그인 사용이 가능합니다. [사전 검수 관련 공지사항](#)을 확인하세요.

- [\[애플리케이션 이름\]](#) 설정을 확인해 주세요.
- [\[로그인 오픈 API\] > PC 웹 > 서비스 URL](#) 설정을 확인해 주세요.
- [\[로그인 오픈 API\] > PC 웹 > Callback URL](#) 설정을 확인해 주세요.

## 5. Client ID, Client Secret 확인

### 애플리케이션 정보

Client ID	O5DmDZZ2LsLa_V6xlhR1
Client Secret	<div style="border: 1px solid #ccc; padding: 5px; text-align: center;">             .....           </div> <div style="text-align: center; margin-top: 5px;"> <span style="border: 1px solid #008000; padding: 2px 5px; color: #008000;">보기</span> </div>

### 회원 정보 요청하기

#### 1. Code 요청하기

```
[Request URL]
https://nid.naver.com/oauth2.0/authorize?client_id={Client_ID}&response_type=code&redirect_uri={redirect URL}

[Response URL]
{redirect URL}?code={code}&state=
```

#### 2. Token 요청하기

```
[Request URL]
https://nid.naver.com/oauth2.0/token?grant_type=authorization_code&client_id={ClientID}&client_secret={ClientSecret}&code={code}&state={state}

[Response Body]
{
  "access_token": ,
  "refresh_token": ,
  "token_type": "bearer",
  "expires_in": "3600"
}
```

### 3. 유저 정보 요청하기

```
[Request URL]
https://openapi.naver.com/v1/nid/me

[Request Header]
{
  "Authorization" : {token_type} {access_token}
}

[Response Body]
{
  "resultcode": "00",
  "message": "success",
  "response": {
    "id": ,
    "nickname": ,
    "email":
  }
}
```

애플리케이션 등록시 선택한 데이터를 response에 담아서 전달

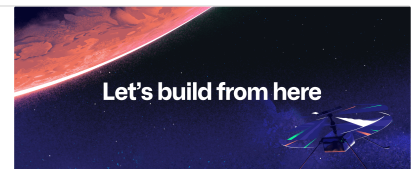
## 2. GITHUB

<https://github.com/>

GitHub: Let's build from here

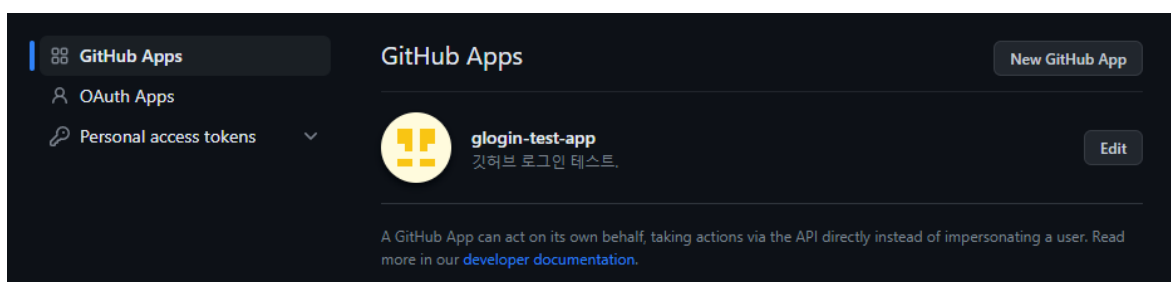
GitHub is where over 100 million developers shape the future of software, together. Contribute to the open source community, manage your Git repositories, review code like a pro, track bugs and fea...

<https://github.com/>

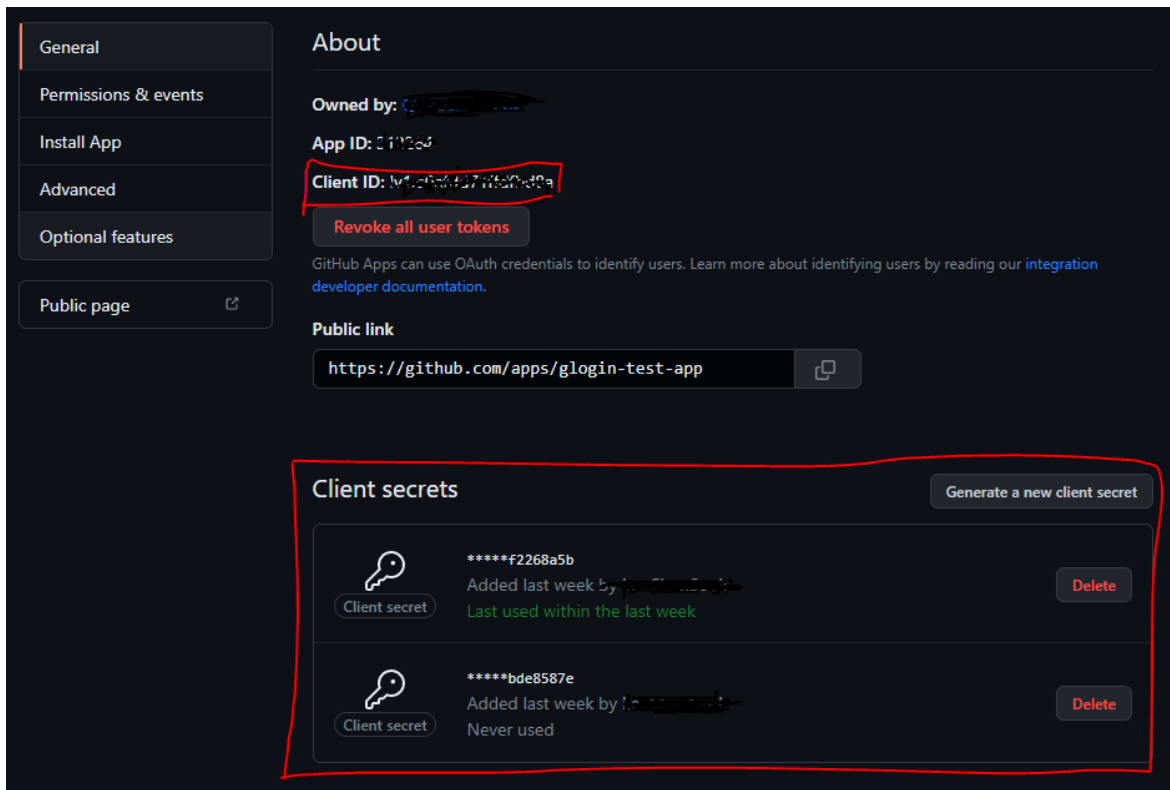


## 애플리케이션 등록

- Setting - Developer Setting - New GitHub App



- Client ID, Client Secret 확인



### 1. code 발급

```
[Request URL]
https://github.com/login/oauth/authorize?client_id={Client ID}

[Response URL]
{redirect URL}?code={code}
```

### 2. 토큰 발급

```
[Request URL]
https://github.com/login/oauth/access_token?grant_type=authorization_code&client_id={client ID}&client_secret={client Secret}&code={code}

[Response Body]
{
  "access_token": "",
  "expires_in": 28800,
  "refresh_token": "",
  "refresh_token_expires_in": 15811200,
  "token_type": "bearer",
  "scope": ""
}
```

### 3. 유저 정보 조회

```
[Request URL]
https://api.github.com/user

[Request Header]
{
  "Authorization" : {token_type} {access_token}
}

[Response Body]
{
```

```

"login":,
"id":,
"node_id":,
"avatar_url":,
"gravatar_id":,
"url":,
"html_url":,
"followers_url":,
"following_url":,
"gists_url":,
"starred_url":,
"subscriptions_url":,
"organizations_url":,
"repos_url":,
"events_url":,
"received_events_url":,
"type":,
"site_admin":,
"name":,
"company":,
"blog":,
"location":,
"email":,
"hireable":,
"bio":,
"twitter_username":,
"public_repos":,
"public_gists":,
"followers":,
"following":,
"created_at":,
"updated_at":
}

```

애플리케이션 등록시 선택한 데이터를 전달

## 2. FCM 설정

### Firestore 앱 등록 및 프로젝트 설정

먼저 구글 계정에 로그인하고 firebase 홈페이지에서 오른쪽 상단의 [콘솔로 이동] 을 클릭해서 콘솔에 접속

<https://console.firebase.google.com/u/0/?hl=ko>

### 프로젝트 생성

× 프로젝트 만들기(1/3단계)

## 프로젝트 이름을 지정하여 시작하기 ②

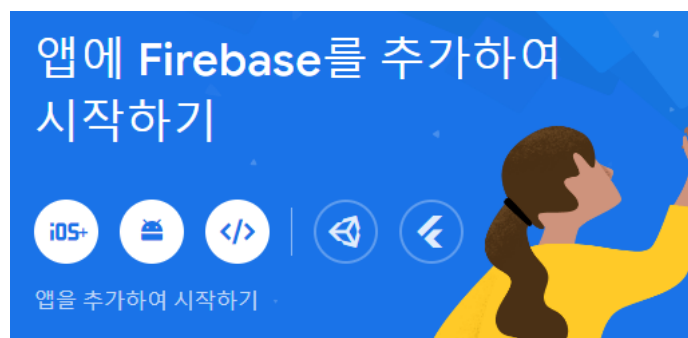
프로젝트 이름 입력

my-awesome-project-id

계속

### 앱 추가하기

1. 앱 선택하기 - (ios, android, web)



2. Android App package 이름 등록

× Android 앱에 Firebase 추가

1 앱 등록

Android 패키지 이름 ②

com.company.appname

3. 구성 파일 (google-service.json) 파일 다운로드 및 추가



## 2 구성 파일 다운로드 후 추가

Android 스튜디오에 대한 안내(아래 참조) | [Unity](#) [C++](#)

### ↓ google-services.json 다운로드

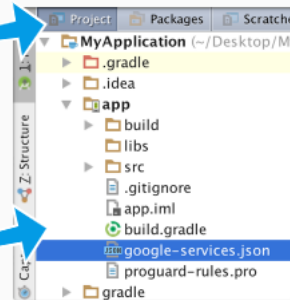
Android 스튜디오에서 프로젝트 뷰로 전환하여 프로젝트 루트 디렉터리를 표시합니다.

다운로드한 google-services.json 파일을 모듈(앱 수준) 루트 디렉터리로 이동합니다.



google-services.json

다음



## 4. Firebase SDK 추가



## 4. 프로젝트에 활용되는 프로퍼티 파일

### 1. api-module

- application.yml

```
spring:
  profiles:
    active: prod # 개발 환경에서는 local, 서버 환경에서는 prod
```

- application-local.yml

```
server:
  port: 8000

logging:
  level:
    com.amazonaws.util.EC2MetadataUtils: error
    root: warn
    com.nassafy.api: DEBUG
    org.hibernate.type.descriptor.sql: DEBUG
    org.hibernate.SQL: DEBUG

spring:
  config:
    activate:
      on-profile: local
  s3:
    bucket: {bucket_name}
    access-key: {bucket_access-key}
    secret-key: {bucket_secret-key}

datasource:
  driver-class-name: org.mariadb.jdbc.Driver
  url: jdbc:mariadb://localhost:3306/{database_name}
  username: {username}
  password: {password}

jpa:
  hibernate:
    ddl-auto: update
    show-sql: true
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MariaDBDialect
      format_sql: true
  defer-datasource-initialization: true

servlet:
  multipart:
    max-file-size: 100MB
    max-request-size: 100MB

mail:
  host: smtp.gmail.com
  port: 587
  username: {앱 비밀번호 발급받은 google 계정}
  password: {발급받은 앱 비밀번호}
```

```

properties:
  mail:
    smtp:
      starttls:
        enable: true
      auth: true
  redis:
    host: {host_name}
    port: 6379
    password: {redis password}
  jwt:
    header: Authorization
    secret: {secret key}
    token-validity-in-seconds: 86400

cloud:
  aws:
    region:
      static: ap-northeast-2
    stack:
      auto: false

sns:
  naver:
    url: https://openapi.naver.com/v1/nid/me
  github:
    url: https://api.github.com/user

```

- application-prod.yml

```

server:
  port: 8000

logging:
  level:
    com.amazonaws.util.EC2MetadataUtils: error
    root: warn
    com.nassafy.api: DEBUG
  org:
    springframework:
      web:
        filter: DEBUG

spring:
  config:
    activate:
      on-profile: prod
  s3:
    bucket: {bucket_name}
    access-key: {bucket_access-key}
    secret-key: {bucket_secret-key}

  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://{database container name}:3306/{database name}
    username: {username}
    password: {password}

  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MariaDBDialect
        format_sql: true
    defer-datasource-initialization: true

```

```

servlet:
  multipart:
    max-file-size: 100MB
    max-request-size: 100MB

mail:
  host: smtp.gmail.com
  port: 587
  username: {앱 비밀번호 발급받은 google 계정}
  password: {발급받은 앱 비밀번호}
  properties:
    mail:
      smtp:
        starttls:
          enable: true
        auth: true

redis:
  host: {redis container name}
  port: 6379
  password: {redis password}

jwt:
  header: Authorization
  secret: {secret key}
  token-validity-in-seconds: 86400

cloud:
  aws:
    region:
      static: ap-northeast-2
    stack:
      auto: false

sns:
  naver:
    url: https://openapi.naver.com/v1/nid/me
  github:
    url: https://api.github.com/user

```

## 2. batch-module

- application.yml

```

spring:
  profiles:
    active: prod # 개발 환경에서는 local, 서버에서는 prod

prob:
  pivot : 10

```

- application-prod.yml

```

server:
  port: 8001

logging:
  level:
    root: warn
    com: DEBUG
    com.nassafy.batch: DEBUG
    org.hibernate.type.descriptor.sql: DEBUG

```

```

    org:
      springframework:
        web:
          filter: DEBUG

spring:
  config:
    activate:
      on-profile: prod
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://{database container name}:3306/{database}
    username: {username}
    password: {password}
    hikari:
      maximum-pool-size: 20
  jpa:
    hibernate:
      ddl-auto: update
      format_sql: true
      show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MariaDBDialect
      defer-datasource-initialization: true
  redis:
    host: {redis container name}
    port: 6379
    password: {redis password}
  batch:
    job:
      enabled: false
    jdbc:
      isolation-level-for-create: default
  jwt:
    header: Authorization
    secret: {secret key}
    token-validity-in-seconds: 86400

openweathermap:
  api-key: {open weather api key}

fcm :
  key :
    scope : https://www.googleapis.com/auth/cloud-platform
    path : {fcm project json file name}
  api_url : https://fcm.googleapis.com/v1/projects/{project id}/messages:send
  projectID: {project id}

r:
  host: {rstudio container name}

```

- application-local.yml

```

server:
  port: 8001

logging:
  level:
    root: warn
    com: DEBUG
    com.nassafy.batch: DEBUG
    org.hibernate.type.descriptor.sql: DEBUG

openweathermap:
  api-key: {open weather key}
spring:

```

```

config:
  activate:
    on-profile: local
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://localhost:3306/{database name}
    username: {username}
    password: {password}
    hikari:
      maximum-pool-size: 20
  jpa:
    hibernate:
      ddl-auto: update
      format_sql: true
      show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MariaDBDialect
      defer-datasource-initialization: true
  redis:
    host: {host name}
    port: 6379
    password: {password}
  batch:
    job:
      enabled: false
    jdbc:
      isolation-level-for-create: default
  jwt:
    header: Authorization
    secret: {secret key}
    token-validity-in-seconds: 86400

  fcm :
    key :
      scope : https://www.googleapis.com/auth/cloud-platform
      path : {fcm project json file name}
      api_url : https://fcm.googleapis.com/v1/projects/{project id}/messages:send
      projectID: {project id}

  r:
    host: {server host name}

```

### 3. Android

- build.gradle (:project)

```

buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.3.15'
    }
}

plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.8.10' apply false

    //safeArgs
    id 'androidx.navigation.safeargs' version '2.4.2' apply false

    //dagger-hilt
    id 'com.google.dagger.hilt.android' version '2.45' apply false

    //google map
    id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin' version '2.0.1' apply false

```

```

}

task clean(type: Delete) {
    delete rootProject.buildDir
}

```

- build.gradle (:app)

```

plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-parcelize'
    id 'org.jetbrains.kotlin.kapt'
    id 'androidx.navigation.safeargs.kotlin'
    id 'com.google.dagger.hilt.android'
    id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin'
}

apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'

Properties properties = new Properties()
properties.load(project.rootProject.file('local.properties').newDataInputStream())

android {
    namespace 'com.nassafy.aro'
    compileSdk 33

    defaultConfig {
        applicationId "com.nassafy.aro"
        minSdk 26
        targetSdk 33
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        buildConfigField "String", "WEATHER_API_KEY", properties['WEATHER_API_KEY']
        buildConfigField "String", "GITHUB_CLIENT_ID", properties['GITHUB_CLIENT_ID']
        buildConfigField "String", "GITHUB_CLIENT_SECRET", properties['GITHUB_CLIENT_SECRET']
        buildConfigField "String", "NAVER_CLIENT_ID", properties['NAVER_CLIENT_ID']
        buildConfigField "String", "NAVER_CLIENT_SECRET", properties['NAVER_CLIENT_SECRET']

        //add to blur image
        renderscriptTargetApi 18
        renderscriptSupportModeEnabled true
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    // Allow references to generated code
    kapt {
        correctErrorTypes true
    }
    viewBinding {
        enabled = true
    }
    buildFeatures {
        viewBinding true
    }
}

```

```

        compose true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.4.3"
    }
}

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

    //viewmodel dependency 추가
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'

    //liveData dependency 추가
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.4.1'
    implementation "androidx.activity:activity-ktx:1.1.0"

    //framework ktx dependency 추가
    implementation "androidx.fragment:fragment-ktx:1.5.4"
    implementation 'androidx.activity:activity-ktx:1.2.2'

    // Jetpack Navigation Kotlin
    def nav_version = "2.4.2"
    apply plugin: "androidx.navigation.safeargs.kotlin"
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"

    // google maps, util
    implementation 'com.google.android.gms:play-services-maps:18.1.0'
    implementation 'com.google.maps.android:android-maps-utils:3.4.0'
    implementation 'com.google.android.gms:play-services-location:21.0.1'
    implementation 'com.google.maps.android:maps-ktx:3.4.0'
    implementation 'com.google.maps.android:maps-utils-ktx:3.4.0'

    // 레트로핏
    // https://github.com/square/retrofit
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'

    // https://github.com/square/retrofit/tree/master/retrofit-converter/gson
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'com.google.code.gson:gson:2.9.0'
    implementation 'com.squareup.retrofit2:adapter-rxjava:2.1.0'

    // okhttp3
    implementation 'com.squareup.okhttp3:okhttp:3.11.0'
    implementation 'com.squareup.okhttp3:logging-interceptor:3.11.0'

    // 코루틴
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.0'
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-play-services:1.3.1"

    // permission
    implementation 'io.github.ParkSangGwon:tedpermission-normal:3.3.0'

    // picasso
    implementation 'com.squareup.picasso:picasso:2.8'
    implementation 'jp.wasabeef:picasso-transformations:2.4.0'

    // dagger-hilt
    implementation "com.google.dagger:hilt-android:2.45"
    kapt "com.google.dagger:hilt-compiler:2.45"
    implementation 'androidx.hilt:hilt-navigation-fragment:1.0.0'

```



```

// viewPager2
implementation "androidx.viewpager2:viewpager2:1.0.0"

// UI Tests
androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
debugImplementation 'androidx.compose.ui:ui-test-manifest'

//Compose 설정
def composeBom = platform('androidx.compose:compose-bom:2023.01.00')
implementation composeBom
androidTestImplementation composeBom
implementation 'androidx.compose.material3:material3'

// Android Studio Preview support
implementation 'androidx.compose.ui:ui-tooling-preview'
debugImplementation 'androidx.compose.ui:ui-tooling'

implementation 'androidx.compose.material:material-icons-core'
implementation 'androidx.compose.material:material-icons-extended'
implementation 'androidx.compose.material3:material3-window-size-class'
implementation 'androidx.activity:activity-compose:1.6.1'
implementation 'androidx.lifecycle:lifecycle-viewmodel-compose:2.5.1'
implementation 'androidx.compose.runtime:runtime-livedata'

//compose viewPager
implementation "com.google.accompanist:accompanist-pager:0.20.1"
implementation "com.google.accompanist:accompanist-pager-indicators:0.20.1"

//naver
implementation 'com.navercorp.nid:oauth-jdk8:5.4.0' // jdk 8

// carouselrecyclerview
implementation 'com.github.sparrow007:carouselrecyclerview:1.2.6'

// MPAndroidChart
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'

// Compose Coil
implementation "io.coil-kt:coil-compose:1.3.2"
implementation "io.coil-kt:coil-svg:1.3.2"

// imagePicker
implementation 'com.github.dhaval2404:imagepicker:2.1'

// FCM
implementation platform('com.google.firebase:firebase-bom:31.2.3')
implementation 'com.google.firebase:firebase-analytics-ktx'
implementation 'com.google.firebase:firebase-messaging-ktx' //FCM-push
implementation 'com.google.firebase:firebase-dynamic-module-support:16.0.0-beta03'

// scalablelayout
implementation 'com.ssomai:android.scalablelayout:2.1.6'

// ExifInterface
implementation 'androidx.exifinterface:exifinterface:1.3.6'

implementation 'com.drewnoakes:metadata-extractor:2.18.0'
}

```