포팅메뉴얼

기술스택

▼ 개발환경

OS

- window 10
- Ubuntu 20.04 LTS

IDE

- Android Studio Dolphin | 2021.3.1

Datebase

- Firebase

Docker

- Docker 20.10.18

EC2 기본

▼ Docker 설치

```
# 오래된 버전 삭제하기
sudo apt-get remove docker docker-engine docker.io containerd runc

# repository 설정하기
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Docker의 Official GPG Key 를 등록
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

#stable repository 를 등록
echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Docker Engine 설치하기
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io

# 설치확인
docker --version
```

▼ Docker Image pull

- node:16-alpine

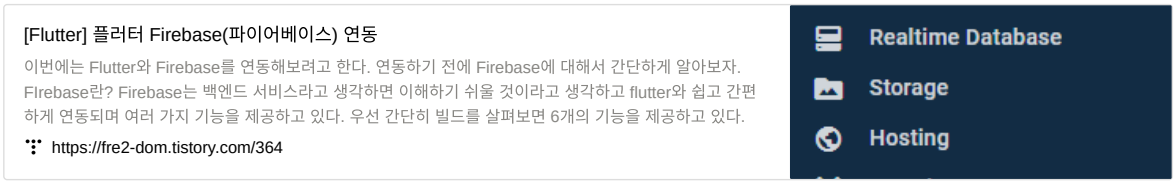
```
docker pull node:16-alpine
```

- openjdk:8-jre-alpine

```
docker pull openjdk:8-jre-alpine
```

DB

▼ Firebase 설정



1단계 : 필요한 명령줄 도구 설치

1. 아직 설치하지 않았다면 Firebase CLI를 설치합니다 .
2. 다음 명령을 실행하여 Google 계정을 사용하여 Firebase에 로그인합니다.

   ```
   firebase login
   ```

3. 디렉토리에서 다음 명령을 실행하여 FlutterFire CLI를 설치합니다.

   ```
   dart pub global activate flutterfire_cli
   ```

2단계 : Firebase를 사용하도록 앱 구성

FlutterFire CLI를 사용하여 Firebase에 연결하도록 Flutter 앱을 구성합니다.

Flutter 프로젝트 디렉터리에서 다음 명령을 실행하여 앱 구성 워크플로를 시작합니다.

```
flutterfire configure
```

3단계 : 앱에서 Firebase 초기화

1. Flutter 프로젝트 디렉터리에서 다음 명령을 실행하여 핵심 플러그인을 설치합니다.

   ```
   flutter pub add firebase_core
   ```

2. Flutter 프로젝트 디렉터리에서 다음 명령을 실행, Flutter 앱의 Firebase 구성 최신 상태 확인.

   ```
   flutterfire configure
   ```

3. `lib/main.dart` 파일에서 Firebase 핵심 플러그인과 이전에 생성한 구성 파일을 가져옵니다.

   ```
   import 'package:firebase_core/firebase_core.dart';
   import 'firebase_options.dart';
   ```

4. 또한 `lib/main.dart` 파일에서 구성 파일에서 내보낸 `DefaultFirebaseOptions` 객체를 사용하여 Firebase를 초기화 합니다.

```
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

5. Flutter 애플리케이션을 리빌드합니다.

```
flutter run
```

4단계 : Firebase 플러그인 추가

각 Firebase 제품(예: Cloud Firestore, 인증, 분석 등)에 하나씩 다양한 Firebase Flutter 플러그인 을 통해 Flutter 앱에서 Firebase에 액세스합니다.

Firebase Flutter 플러그인을 추가하는 방법은 다음과 같습니다.

1. Flutter 프로젝트 디렉터리에서 다음 명령을 실행합니다.

```
flutter pub addPLUGIN_NAME
```

2. Flutter 프로젝트 디렉터리에서 다음 명령을 실행합니다.

```
flutterfire configure
```

3. 완료되면 Flutter 프로젝트를 다시 빌드하십시오.

```
flutter run
```

구글 설정 파일(Google-Service.json) 다운로드

1. Firebase에 로그인하고 프로젝트를 엽니다.
2. 아이콘을 클릭하고 **프로젝트 설정**을 선택합니다.
3. **내 앱** 카드에서 구성 파일이 필요한 앱의 패키지 이름을 목록에서 선택합니다.
4. **google-services.json**을 클릭합니다.
5. File은 app 수준에 복사해 위치시킵니다.

Window에서 SHA-1 확인하기

```
keytool -list -v -keystore "%USERPROFILE%₩.android₩debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

Spring Boot 배포

▼ Docker를 활용한 배포

- Dockerfile을 project root에 작성

Dockerfile

```
FROM adoptopenjdk:8-jdk-hotspot AS builder
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
```

```
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

FROM adoptopenjdk:8-jdk-hotspot
COPY --from=builder build/libs/*.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]
```

- Build Step: Execute shell

```
cd MessageAPI
chmod +x gradlew
docker build -t message-api:0.0.1 .
docker run -d -p 8080:8080 --name message-api message-api:0.0.1
```

- chmod +x gadlew 빌드할 권한부여
- docker build -t message-api:0.0.1 message-api라는 이름으로 도커이미지를 빌드 명령.
- docker run -d -p 8080:8080 --name message-api message-api:0.0.1 message-api:0.0.1를 8080포트에서, 컨테이너 이름은 message-api으로 해서 작동 명령

Nginx

▼ docker container 설치

[SSL] Docker Nginx, Certbot & Let's Encrypt를 사용해 SSL 인증서 발급

EC2 ubuntu 18.04 Docker CE, CE-CLI 20.10.6 Docker Compose 1.29.1 docker-compose.yml init-letsencrypt.sh data/nginx/conf.d/app.conf data/nginx/conf.d/app.conf server { listen 80; listen [::]:80; server_name {domain}; // 등록한 도메인 추가 location /.well-known/acme-challenge/ { allow all; root

https://velog.io/@fordevelop/Docker-Nginx-Certbot-Lets-Encrypt%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%B4-SSL-%EC%9D%B8%EC%A6%9D%EC%84%9C-%EB%B0%9C%EA%B8%89


```
docker run --name nginx -p 80:80 -dt nginx
```

- 우선 nignx 설정파일 경로를 찾아서 접근(docker container 안에서 접근 등)

```
sudo su
# nginx.conf 파일 위치 찾기
find / -name nginx.conf
결과로 나온 위치 확인 후 해당 경로로 이동하는데 해당 경로는 nginx.conf 파일이있는 경로이고,
우리가 수정할 default.conf은 위에 나온경로/conf.d 아래있음

#directory이동
cd 경로/conf.d

# default.conf 수정
nano default.conf
```

default.conf 파일이 이미 있는 경우 복사 가능하다.

```
docker cp default.conf nginx:/etc/nginx/conf.d/default.conf
docker exec nginx nginx -s reload
```

▼ default.conf

- 기본 설정에 client 부분만 수정, 추가 (bold체 부분)

```
upstream client{
  server homelaone.kr:80;
}
```

```
upstream message-api{
  server homelaone.kr:8080;
}

server {
    listen       80;
    server_name  homelaone.kr;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
    location /.well-known/acme-challenge/ {
            allow all;
            root /var/www/certbot;
    }
    location /privacy {
        root   /usr/share/nginx/html;
        index  watchoutP.html watchoutP.htm;
    }
    location /terms {
        root   /usr/share/nginx/html;
        index  watchoutT.html watchoutT.htm;
    }
    location /terms/watch-out {
        proxy_pass http://client/terms;
    }
    location /privacy/watch-out {
        proxy_pass http://client/privacy;
    }
    location /api {
        proxy_pass http://message-api/api;
    }
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

}

server {
    listen 443 ssl;
    server_name homelaone.kr;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/homelaone.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/homelaone.kr/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://homelaone.kr:80;
        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;
    }
}
```

```
# conf 파일 수정 후 수정사항을 반영
$ docker container exec <container> nginx -s reload
<>에는 현재 가동중인 nginx 컨테이너명 또는 ID를 작성
확인이 필요하다면 docker ps 로 확인
```

방화벽

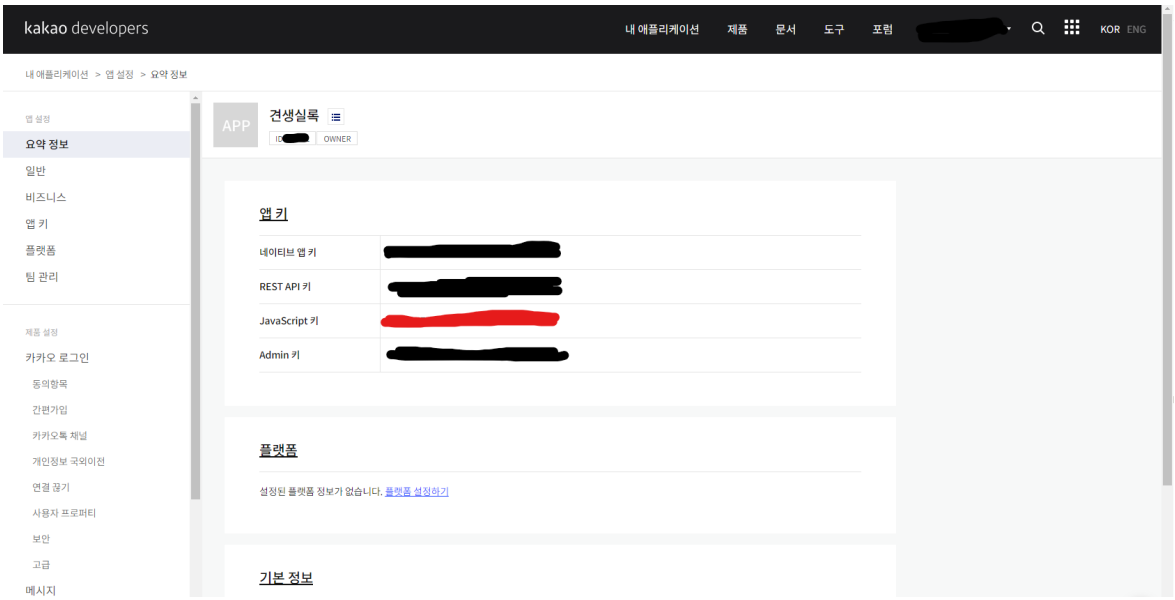
▼ 사용 포트

```
To                         Action      From
--                         ------      ----
22/tcp                     ALLOW IN    Anywhere // ssh
80                         ALLOW IN    Anywhere
443                        ALLOW IN    Anywhere
```

Flutter 카카오 맵 연동

▼ 카카오 맵 연동

1. 카카오 디벨로퍼 가입 후 앱 생성
2. 앱 키 중 JavaScript 키, Rest API 키 복사



3. 아래 내용으로 .env 파일 생성(위치: `<app dir>/.env`)

```
kakaoMapAPIKey=<발급받은 kakaoMapApi(JavaScript 키)>
kakaoRestAPIKey=<카카오 지도의 Rest API 키>
```

gabia 메시징 서비스

▼ 서비스 이용 방법

- 가비아 메시징 서비스에서 신청서 작성 및 비용 지불을 통해 서비스를 신청합니다.
- 서비스 신청이 승인되면 수령한 API KEY를 활용해 다음과 같은 양식의 message.yml을 작성해 src/main/resources에 저장합니다.

```
open-api:
  sms-id: 가비아에 등록한 계정 ID
  api-key: 가비아에서 수령한 API KEY
```

- 이제 스프링 부트 프로젝트를 실행하면 LMS 및 MMS 메세지를 적절한 API 호출을 통해 전송할 수 있습니다.

안전 지도 데이터

▼ 데이터 출처

CCTV 데이터

- 서울 열린데이터 광장 - 서울시 (안심이) CCTV 설치 현황
- http://data.seoul.go.kr/dataList/OA-20923/S/1/datasetView.do

안심택배함 데이터

- 서울 열린데이터 광장 - 서울시 여성 안심 택배함(안심이) 설치 장소
- http://data.seoul.go.kr/dataList/OA-20922/S/1/datasetView.do

비상벨 데이터

- ~~공공데이터포털 - 전국안전비상벨위치표준데이터~~
- 행정안전부 - 안전비상벨위치정보
- https://www.localdata.go.kr/lif/lifeCtacDataView.do?opnEtcSvcId=12_04_09_E

▼ 데이터 사용 방법

서울 열린데이터 광장

- 로그인 후 일반 인증키 신청을 통해 인증키를 받습니다.
- 위 데이터 링크에 있는 OPEN API 호출 URL을 통해 원하는 데이터를 호출합니다.

행정안전부 - 안전비상벨위치정보

- 위 링크에서 EXCEL 버튼을 클릭해 데이터를 다운로드받습니다.
- 다운로드 받은 .xlsx 파일에서 위도 경도를 제외한 나머지 데이터를 제거하고 위도 Column 명을 WGSXPT로, 경도 Column 을 WGSYPT로 변경한 뒤 .csv 확장자로 변경해 저장합니다.
- .csv 파일을 .json 파일로 바꿔주는 사이트에서 위 .csv 파일을 .json 파일로 변경한 후 /assets/json 폴더에 넣습니다.

Flutter 구글 플레이스토어 수동배포

▼ Flutter Build(.aab app bundle 파일 생성)

1. (이미 있다면 생략 가능) 키스토어 파일(.jks)만들기

맥/리눅스:

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

윈도우:

```
keytool -genkey -v -keystore c:/Users/<유저명>/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias <참조할 이름>
```

2. `<app dir>/android/key.properties` 파일을 만들어 참조하기

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=<생성하고 싶은 키 이름>
storeFile=<key store 파일 위치, 예) /Users/<user name>/key.jks>
```

▼ Gradle 수정하기 (`<app dir>/android/app/build.gradle` 파일을 수정)

```
// 이 위치에 코드를 삽입합니다.
   android {
```

`key.properties` 파일로부터 keystore 정보를 가져올 수 있도록 변경하세요.

```
   def keystoreProperties = new Properties()
   def keystorePropertiesFile = rootProject.file('key.properties')
   if (keystorePropertiesFile.exists()) {
       keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
   }
```

```
// 이 위치에 코드를 삽입합니다.
   android {
```

다음 코드를:

```
   buildTypes {
       release {
           // TODO: release 빌드 버전을 위한 서명 구성을 추가하세요.
           // 현재는 `flutter run --release`가 디버그용 키로 서명되어 동작합니다.
           signingConfig signingConfigs.debug
       }
   }
```

다음과 같이 변경하세요:

```
   signingConfigs {
       release {
           keyAlias keystoreProperties['keyAlias']
           keyPassword keystoreProperties['keyPassword']
           storeFile file(keystoreProperties['storeFile'])
           storePassword keystoreProperties['storePassword']
       }
   }
   buildTypes {
       release {
           signingConfig signingConfigs.release
       }
   }
```

이제 앱의 release 빌드에서 자동으로 서명이 될 것 입니다.

▼ Proguard 사용(난독화 및 축소)

Step 1 - Proguard 구성하기

Proguard Rule을 구성하기 위하여 `/android/app/proguard-rules.pro` 파일을 생성하고 다음과 같은 규칙을 추가합니다.

```
## Flutter wrapper
-keep class io.flutter.app.** { *; }
-keep class io.flutter.plugin.**  { *; }
-keep class io.flutter.util.**  { *; }
-keep class io.flutter.view.**  { *; }
-keep class io.flutter.**  { *; }
-keep class io.flutter.plugins.**  { *; }
-dontwarn io.flutter.embedding.**
```

이 설정은 Flutter 엔진 라이브러리를 보호합니다. 다른 라이브러리(예. Firebase)를 보호하기 위해서는 추가적으로 규칙을 작성해야합니다.

Step 2 - 난독화와 축소 사용하기

`/android/app/build.gradle` 파일을 열고 `buildTypes`가 정의된 부분을 찾으세요. `release` 설정 부분 안에 `minifiyEnabled`와 `useProguard` 플래그가 true로 설정되어 있습니다. 여기에 step 1 에서 만든 Proguard 규칙 파일을 추가해야 합니다.

참고: 난독화와 축소 과정으로 인해 Android 앱의 컴파일 시간이 크게 늘어날 수 있습니다.

```
android {

    ...

    buildTypes {

        release {
```

```
            signingConfig signingConfigs.release

            minifyEnabled true
            useProguard true

            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

        }
    }
}
```

▼ 앱 매니페스트 검토하기

올바른 앱 설정을 위하여 `<app dir>/android/app/src/main`에 있는 기본 앱 매니페스트 파일인 `AndroidManifest.xml`을 검토하고 올바른 값들을 포함하는지 확인하세요. 특히:

- `application`: 앱의 최종 이름을 반영하기 위해 `application`에 있는 `android:label`을 수정하세요.

▼ 빌드 구성 검토하기

올바른 빌드 구성을 위하여 `<app dir>/android/app`에 있는 기본 Gradle build file 파일인 `build.gradle`을 검토하고 올바른 값들을 포함하는지 확인하세요. 특히:

- `defaultConfig`:
  - `applicationId`: 고유한 최종 (Application Id)appid를 지정하세요.
  - `versionCode` & `versionName`: 내부 앱 버전 번호를 지정하고, 문자열 형태로 명시하세요. pubspec.yaml 파일에 `version` 속성을 설정하여 내부 앱 버전 번호를 문자열 형태로 지정할 수 있습니다. 버전 정보 지침에 대해서는 버전 문서를 참조하세요.
  - `minSdkVersion` & `targetSdkVersion`: 최소 API 레벨과 개발 대상 버전으로 지정한 지정 API 레벨을 명시하세요. 자세한 내용은 버전 문서의 API 레벨 영역을 참조하세요.

▼ release 앱 번들 빌드하기

여기서는 어떻게 release 앱 번들을 빌드하는지 소개합니다. 위 단락의 서명하기를 완료했다면, release 번들이 서명될 것입니다.

터미널:

```
1. cd <app dir>를 입력하세요(당신의 앱 디렉토리로 <app dir> 를 변경하세요.)
2. flutter build appbundle을 실행하세요.(flutter build 실행은 기본적으로 release 빌드 입니다.)
```

`<app dir>/build/app/outputs/bundle/release/app-release.aab`

경로로 출력됩니다.

※ Flutter 빌드가 여러 번 되는 경우 빌드 정보를 변경하려면 `pubspec.yaml` 파일 내부의

`version: 1.0.2+3` (빌드이름+빌드번호)와 같은 형식의 빌드 정보를 수정한 뒤 빌드해주세요.

▼ git에 공유되지 않은 설정파일

```
kakaoMapAPIKey=<카카오 지도의 웹 API 키>
kakaoRestAPIKey=<카카오 지도의 Rest API 키>
cctvAPIKey=<>
openAPIKey=<>
inviteRandomKey=<초대 코드 암/복호화 작업에 사용되는 랜덤하게 생성된 키(직접 생성)>
```