Run with me 배포 매뉴얼

본 문서는 안드로이드로 개발된 Run with me 를 사용하기 위한 가이드를 안드로이드와 백엔드 스프링 서버를 배포하는 과정에 대해 서술하고 있습니다.

# 안드로이드 배포

## AOS 개발 환경

**안드로이드**

- Android Studio Bumblebee (2021.1.1 Patch 2)

- targetSDK 31

- minSDK 23

- Kotlin

**Wear OS**

- Android Studio Bumblebee (2021.1.1 Patch 2)

- targetSDK 32

- minSDK 30

- Kotlin

# AOS 기술 스택

안드로이드

- Retrofit 2.9.0 (통신 라이브러리)

- OkHttp

- Ted Permission 3.3.0 (안드로이드 권한 라이브러리)

- Dagger-Hilt (의존성 주입 라이브러리)

- JetPack Paging3 (페이징 라이브러리)

- Coroutines Flow (비동기 데이터 처리 라이브러리)

- Glide 4.12.0 (이미지 로드 라이브러리)

- ViewModel-ktx 2.3.1

- Fragment-ktx 1.3.6

- Navigation 2.3.5 (화면 전환, 스택 관리 라이브러리)

- Lottie (안드로이드 애니메이션 라이브러리)

- Google Map 18.0.2 (지도 라이브러리)

- OAuth (로그인 보안 라이브러리)

- MPAndroidChart 3.1.0 (통계 라이브러리)

- FCM (파이어베이스 푸시 알림 라이브러리)

- CalendarView (캘린더 라이브러리)

- Room (내부 데이터 베이스 라이브러리)

- Wearable Service 17.1.0 (폰, 워치 데이터 교환 라이브러리)

**Wear OS**

- Retrofit 2.9.0 (통신 라이브러리)

- OkHttp - Ted Permission 3.3.0 (안드로이드 권한 라이브러리)

- Dagger-Hilt (의존성 주입 라이브러리)

- Coroutines Flow (비동기 데이터 처리 라이브러리)

- Glide 4.12.0 (이미지 로드 라이브러리)

- ViewModel-ktx 2.3.1

- Fragment-ktx 1.3.6

- Google Map 18.0.2 (지도 라이브러리)

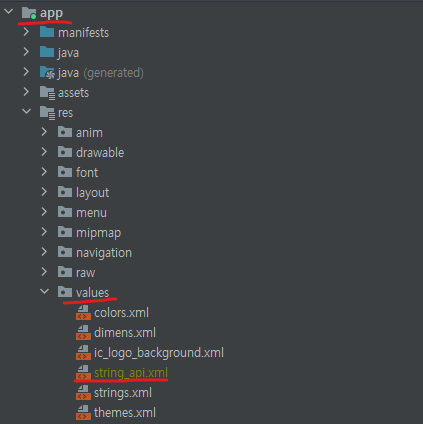
- OAuth (로그인 보안 라이브러리)

- Wearable Service 17.1.0 (폰, 워치 데이터 교환 라이브러리)

# AOS 포팅 가이드

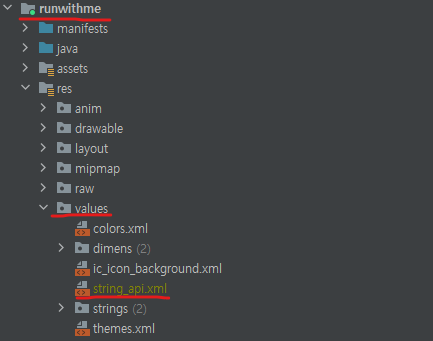
<https://drive.google.com/file/d/1b6l_UzxUQC9TSaValOTI0EhWqmYe9O7I/view?usp=sharing>

## API키



App -> res -> values

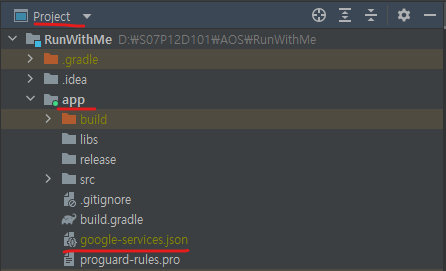
string\_api.xml 추가



runwithme -> res -> values

string\_api.xml 추가

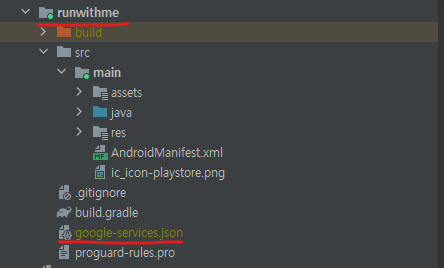
## google-service.json



Project로 변경 후

app

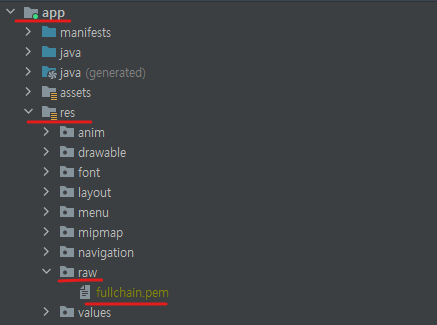
google-services.json 추가



Runwithme

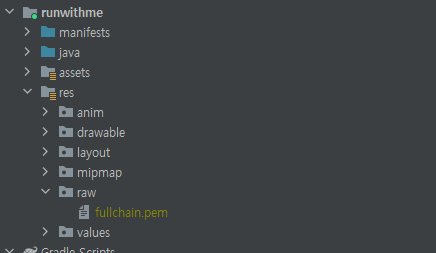
google-services.json 추가

## fullchain.pem



app -> res -> raw

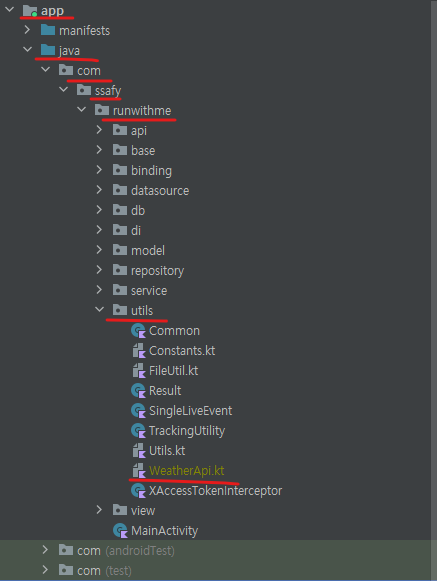
fullchain.pem 추가



runwithme -> res -> raw

fullchain.pem 추가

## WeatherApi



app -> java -> com -> ssafy - > runwithme -> utils

WeatherApi.kt 추가

# 백엔드 배포 과정

## 개발환경 및 배포 환경

개발 환경

-Spring boot version : 2.7.1

-빌드 도구 : spring-boot-maven-plugin, 스프링 부트 버전에 해당하는 내장 메이븐을 사용한다.

-IDE : sts-3.9.14.RELEASE

배포 환경

배포 서버 : Ubuntu 20.04.4 LTS

Docker : 20.10.17

Maven : Apache Maven 3.6.3

Nginx : nginx/1.18.0 (ubuntu)

Certbot : certbot 0.40.0

## API 서버활성을 위한 배포과정

1. Git lab 의 <https://lab.ssafy.com/s07-webmobile4-sub2/S07P12D101> 위치로 이동하여 Clone한다. <Deploy> branch로 checkout 한다.
2. 아래의 명령어를 이용하여 jar파일을 생성한다.

|  |
| --- |
| cd ./BE/gumid101  mvn package -Dmaventest.skip=true chmod 777 ./target/\*.jar |

1. 루트 작업위치에 Dockerfile을 생성하여 아래와 같이 작성한다.

|  |
| --- |
| FROM openjdk:8-jdk  ENTRYPOINT java -jar -Dspring.profiles.active=deploy /deploy/\*.jar  EXPOSE 8080 |

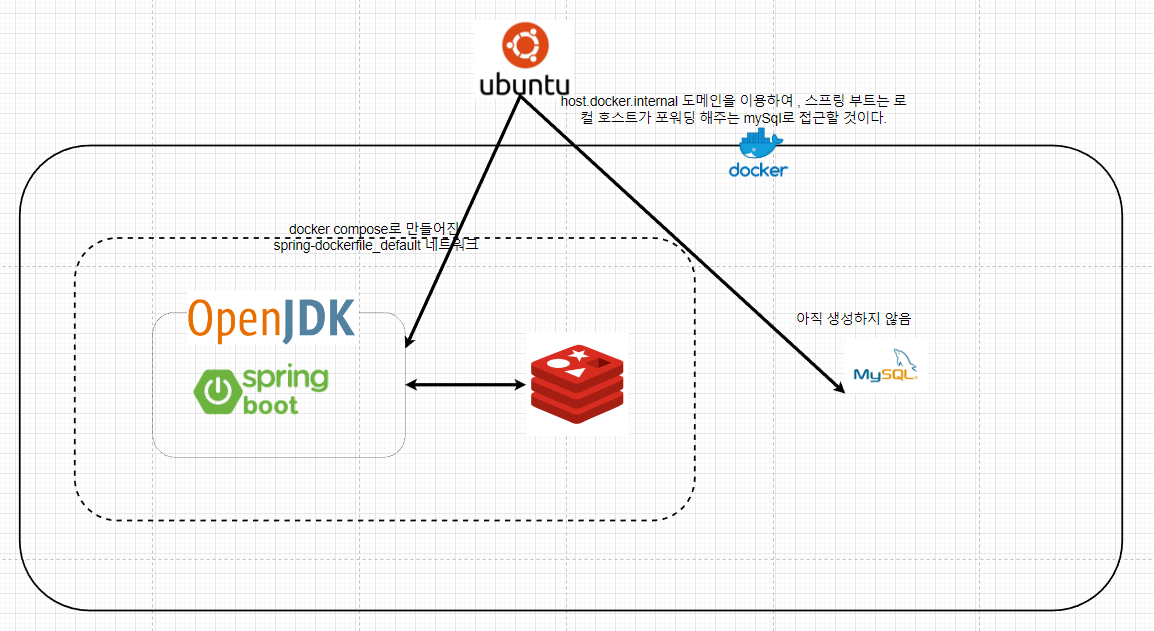
1. docker-compose.yml를 이용하기 위하여 임의로 작업위치를 선정한다.   
   Ex) /home/nonbiri/spring-docker, 그리고 그 하위에 /home/nonbiri/spring-docker/spring-compose 디렉터리를 생성 후 3번에서 작업한 dockerfile을 위치시킨다. 그후 /home/nonbiri/spring-docker 디렉터리에 docker-componse.yml을 생성 후 아래와 같이 작성한다.

|  |
| --- |
| version: "3"  services:  spring:  user: root  container\_name: spring-compose  build:  context: spring-compose  dockerfile: Dockerfile  ports:  - 8080:8080  volumes:  - /home/nonbiri/compose/jenkins/workspace/runwithme/BE/gumid101/target:/deploy  extra\_hosts:  - "host.docker.internal:host-gateway"  environment:  - TZ=Asia/Seoul  redis:  image: redis  ports:  - 16379:6379 |

노란색으로 줄 쳐진 부분은, 2번 과정에서 생성된 jar파일이 위치하는 디렉터리여야 한다.

1. docker-compose.yml 파일이 위치하는 곳으로 이동, 아래 와 같은 명령어를 입력한다.

|  |
| --- |
| docker-compose up --force-recreate --build -d  docker image prune -f |

1. 위 과정을 마치면 스프링 부트 서버는 동작을 해야할텐데 하지 않을 것이다. application.yml 에 data source로 적혀진 DB가 아직 존재하지 않기 때문이다.  
   
2. MySql 서버를 만들기 위하여 임의의 위치를 정하여 이동한다. 그 후 docker-compose.yml을 만든 후 아래와 같이 작성한다.

|  |
| --- |
| version: '3' # docker compose 버전  services:  local-db:  image: library/mysql:8.0.28  container\_name: run\_with\_me\_mysql # 컨테이너 이름  restart: always  ports:  - 13306:3306 # 로컬의 13306 포트를 컨테이너의 3306포트로 연결  environment:  MYSQL\_USER: d101rwm  MYSQL\_PASSWORD: jgalgostudy  MYSQL\_ROOT\_PASSWORD: jgalgostudy  TZ: Asia/Seoul  volumes:  - ./db/mysql/data:/var/lib/mysql  - ./db/mysql/init:/docker-entrypoint-initdb.d |

1. 아래의 명령어를 통해 spring boot에서 접속하는 계정에 권한을 준다.

|  |
| --- |
| docker exec -it mysql-container bash  mysql -u root -p => 패스 워드를 입력 : 위 docker-compose.yml에 작성한 루트 비번사용  Grant All PRIVILEGES ON \*.\* TO ‘d101rwm’@’%’ create database commonssafy default character set utf8mb4 collate utf8mb4\_bin |

1. 5번의 명령어를 다시 입력한다. 그러면 이제 정상적으로 작동한다. Private git lab을 사용함으로, 관계되는 모든 리소스에 대한 설정값들이 이미 적혀있다.
2. 이제 nginx Web Server를 설치하자. Nginx 는 도커를 사용하지 않고 ubuntu 서버에 직접 설치하는 방식을 사용한다.
3. 아래의 명령어를 사용하여 nginx를 설치하자.

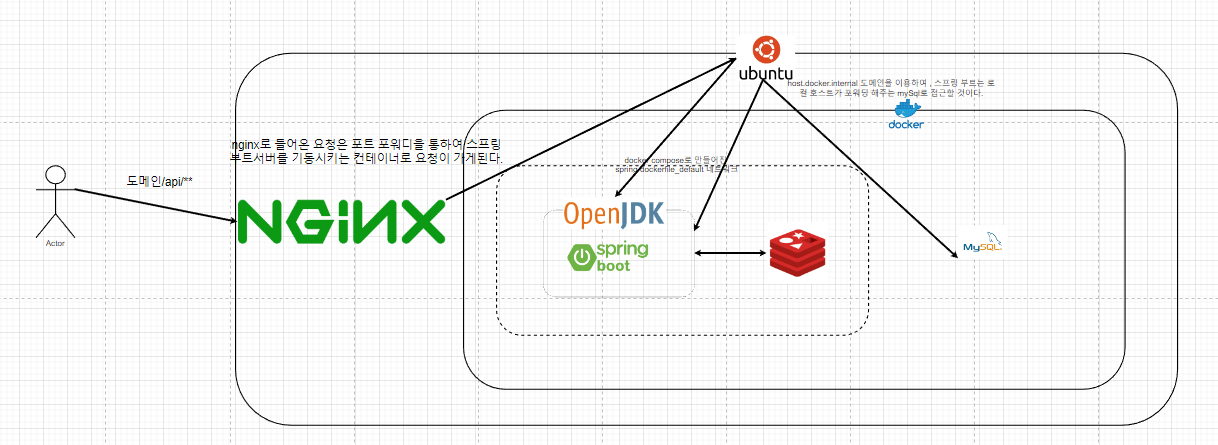
|  |
| --- |
| sudo apt-get update  sudo apt-get install nginx #이렇게 할 경우 nginx 공식에서 제공하는 최신 버전은 아니다.  nginx -v  systemctl status nginx 를 통해 잘 동작하는 지 확인 |

1. 설치가 완료 되었다면 스프링 부트 역프록시 설정을 하기위해 아래의 명령어를 입력한다.

|  |
| --- |
| cd /etc/nginx/sites-enabled  vi default |

1. 사용할Server 블록에 아래와 같이 삽입한다.

|  |
| --- |
| location /api {  proxy\_pass http://localhost:8080/api;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header Host $http\_host;  } |

1. 설정 파일을 고쳤음으로 nginx를 재시작한다. 그러면 이제 아래의 상황과 같게된다.  
   => systemctl restart nginx  
   
2. 이제 http://도메인/api/\*\* 로 요청을 보내면 잘 동작할 것이다. 하지만 우리는 https 를 적용할 것이다. 따라서 우선 nginX를 종료시킨다.  
   => systemctl stop nginx  
   => sudo netstat -ntlp | grep :80 # 80 포트는 비워져있어야한다.
3. 무료로 SSL 적용을 위해 certbot을 다운 받아 standalone방식으로 인증서를 발급받고 적용한다.

|  |
| --- |
| sudo apt update  sudo apt-get install letsencrypt -y  certbot certonly –standalone -d <도메인(사이트명,ec2 도메인)> |

1. 위 과정을 거치면 /etc/letsencrypt/live/ 밑에 도메인 명을 이름으로하는 디렉터리가 생성되고 그 하위에는 여러 pem파일들이 생긴다.
2. 이제 nginx 설정을하는 ~/sites-enabled/default 파일을 수정한다.  
   13번 과정에서 설정했던 server블록에 아래와 같이 적어준다. 노란색 영역이 nginx에 ssl인증서를 적용하기 위한 핵심 부분이다. 회색 부분은 뒤에 설명할 react 배포를 위한 설정이다. 추가로 갈색 부분은 http 프로토콜을 사용하여 80포트로 접근할 경우 https를 사용하는 443포트로 redircect하기 위한 설정이다.

|  |
| --- |
| server {  listen 443 ssl default\_server;  listen [::]:443 ssl default\_server;  client\_max\_body\_size 20M;  ssl\_certificate /etc/letsencrypt/live/i7d101.p.ssafy.io/fullchain.pem;  ssl\_certificate\_key /etc/letsencrypt/live/i7d101.p.ssafy.io/privkey.pem;  server\_name i7d101.p.ssafy.io;  location /api {  # First attempt to serve request as file, then  # as directory, then fall back to displaying a 404.  proxy\_pass http://localhost:8080/api;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header Host $http\_host;  }  location /manager {  proxy\_pass http://localhost:3000;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header Host $http\_host;  }  }  server {  return 301 https://$host$request\_uri;  # managed by Certbot  listen 80;  server\_name regacy\_port;  return 404; # managed by Certbot  } |

1. 설정이 끝났으니 nginx를 재시작한다. 이제 api서버 설정은 끝났다.   
   => systemctl restart nginx

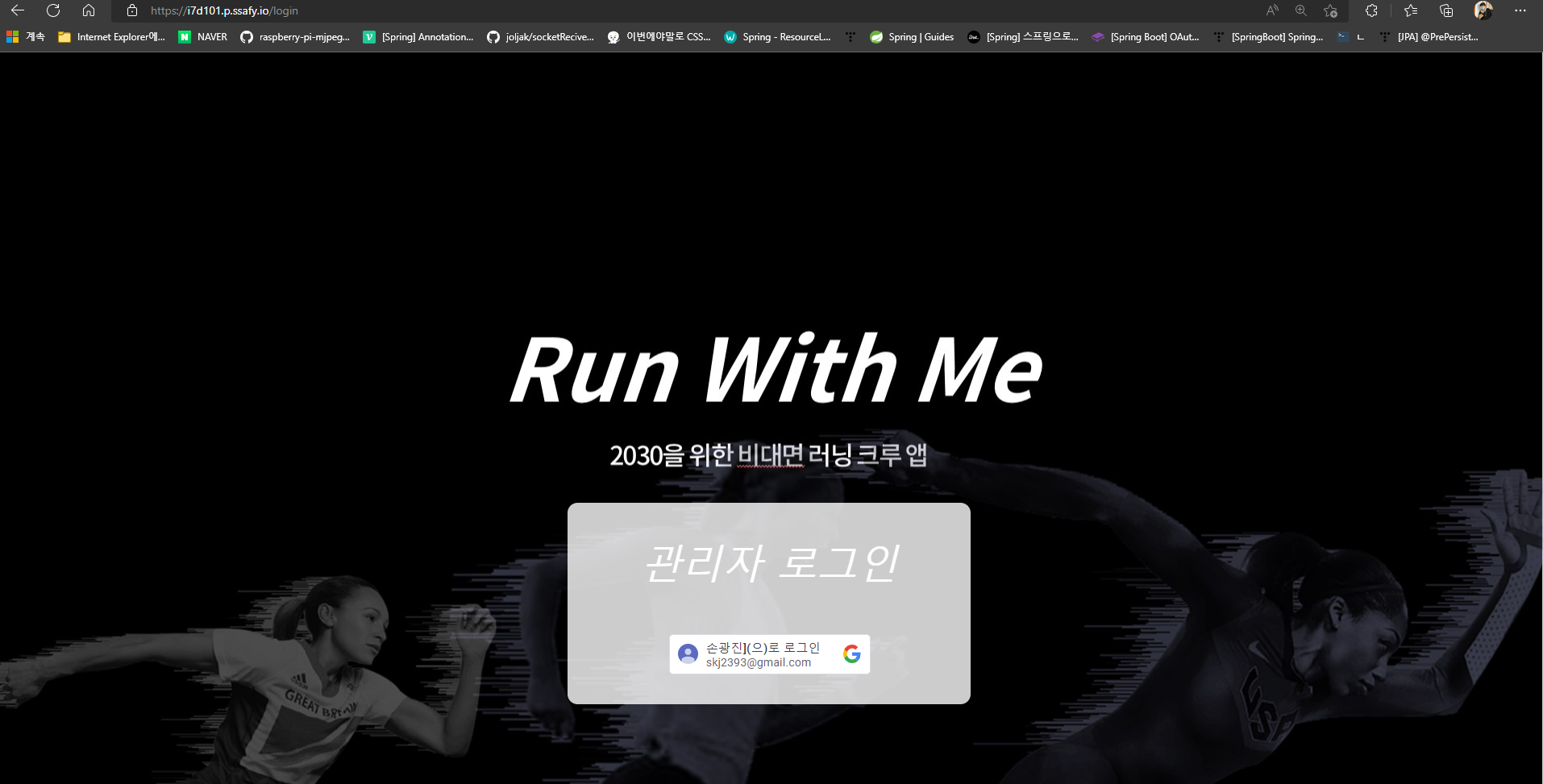
텍스트이(가) 표시된 사진

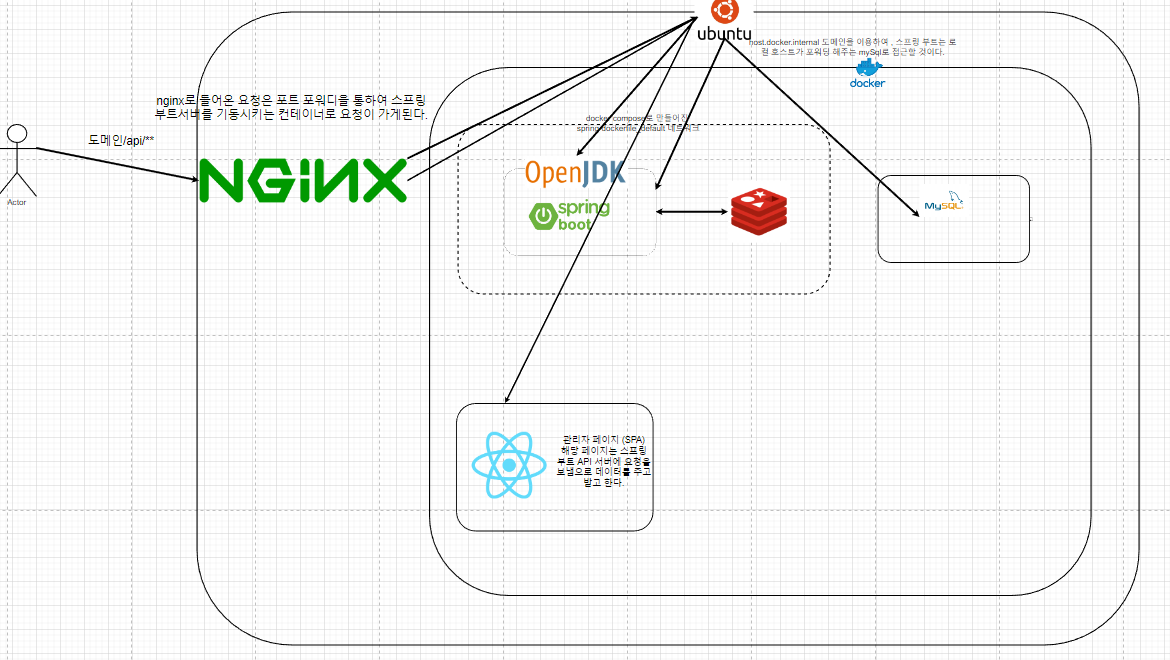
자동 생성된 설명

# 관리자 페이지 배포

1. Git lab 의 <https://lab.ssafy.com/s07-webmobile4-sub2/S07P12D101> 위치로 이동하여 Clone한다. <FEDeploy> branch로 checkout 한다.
   1. 특이사항 : FDDeploy 브랜치와 Deploy 브랜치는 동일한 브랜치로 간주할 수 있습니다. 하지만 저희조에서는 Jenkins를 이용한 재배포를 복습하기 위하여 따로 브랜치를 생성하였습니다. Deploy 브랜치로 해도 동일하겠으나 FEDeploy를 사용하는 것을 권장합니다.
2. 2번 API 서버활성을 위한 배포과정과는 다르게 react 작업 공간 root에 있는 Dockerfile을 사용하면 됩니다.

|  |
| --- |
| FROM node:16.14.0  WORKDIR /usr/src/app  COPY package.json ./  COPY ./ ./  RUN npm install  RUN npm run build  RUN yarn global add serve  COPY ./ ./  CMD [ "npm", "run", "start" ] |

1. 아래의 명령을 이용하여 컨테이너화 시킵니다.  
   => docker build --tag feimg:fetag .
2. <도메인>/manager 로 접속, 소셜로그인  
   
3. 신고글을 처리할 수 있는 페이지가 주어진다.  
   .env 파일또한 포함되어있음으로 따로 설정할 필요가 없다.  
   텍스트이(가) 표시된 사진

   자동 생성된 설명
4. 위 과정을 다 거치고 나면 서버의 구조는 밑에 그림과 같다.  
   

# 외부 서비스에 대한 설명

## Google Oauth, Naver Oauth , Kakao Oauth

application.yml 파일에는 해당 내용이 있어야한다.

spring:

profiles:

include:

- oauth

application-oauth.yml 에는 해당 내용이 있어야한다.

spring:

security:

oauth2:

client:

registration:

google:

client-id: <구글 GCP Oauth설정 후 받은 클라이언트 ID>

client-secret: <시크릿 키>

scope: profile, email

naver:

client-id: <네이버 Oauth 이용 신청 후 받은 ID>

client-secret: <시크릿 키>

redirect-uri: <사용안한다.>

authorization-grant-type: authorization\_code

scope: name, email, profile\_image

client-name: Naver

kakao:

client-id: <카카오 Oauth 서비스 신청 후 발급받은 아이디>

client-secret: <시크릿 키>

redirect-uri: <사용안한다.>

authorization-grant-type: authorization\_code

client-authentication-method: POST

client-name: Kakao

scope:

- profile\_nickname

- profile\_image

- account\_email

provider:

naver:

authorization\_uri: https://nid.naver.com/oauth2.0/authorize

token\_uri: https://nid.naver.com/oauth2.0/token

user-info-uri: https://openapi.naver.com/v1/nid/me

user\_name\_attribute: response

kakao:

authorization-uri: https://kauth.kakao.com/oauth/authorize

token-uri: https://kauth.kakao.com/oauth/token

user-info-uri: https://kapi.kakao.com/v2/user/me

user-name-attribute: id

사실 위에 명시된 모든 값들이 이용되기 위해서는 클라이언트에서 권한서버로 로그인 성공 후 프로바이더가 발급해주는 id\_token 혹은 authorized\_token(프로바이더 마다 이름이 다르다.) 을 통해 리소스 서버(스프링 부트)에서 프로바이더로 accessToken을 얻고 유저 프로필에 대한 정보를 받는 과정이여야 한다. 하지만 우리는 구글의 경우 id\_token을 통해 구글 라이브러리를 사용하여 하지만, 네이버와 카카오의 경우 Android가 client 임으로 accessToken을 얻는 과정까지 수행 후 리소스 서버(스프링 부트) 클라이언트가 준 accessToken을 통해 유저 프로필을 얻고 있다. 따라서 위 속성들은 부분적으로 불필요한 경우가 있으나 확장성과 안정성을 위해 모두 입력해주기를 바란다.

## Firebase

Firebase홈페이지로 이동 후, 스프링 부트에 사용될 json(해당 파일은 클라우드 메시지 탭-> 웹 구성 -> 웹 푸시 인증서에서 발급받는다) 파일을 받은 후 src/main/resources/firebase 밑으로 이동, 붙여넣기 하면 된다.

## 4.3. Amazon S3

[[SpringBoot] SpringBoot를 이용한 AWS S3에 여러 파일 업로드 및 삭제 구현하기 — ROOPRETELCHAM (tistory.com)](https://earth-95.tistory.com/117)

# 젠킨스를 통한 자동배포 시 필요한 bash script

## 스프링 부트 배포

cd ./BE/gumid101 # 스프링 부트 작업 위치까지 이동

pwd

mvn clean # maven 클린

mvn package -Dmaven.test.skip=true # 테스팅 과정없이 패키지화 => jar 파일 생성

pwd

chmod 777 ./target/\*.jar # 생성된 jar파일의 권한을 모두에게 허용

echo "connect to localhost(docker container parents)"

ssh -t -t nonbiri@host.docker.internal <<EOF #jenkins 컨테이너에서 로컬 서버로 접속

cd /home/nonbiri/compose/spring-dockerfile/spring-compose # 컴포즈 파일 실행

docker-compose up --force-recreate --build -d # 기존에 있던 image 컨테이너 없애고 빌드,런

docker image prune -f # 사용안하는 이미지 제거

echo "we need to do builded jar file chmod"

cd /home/nonbiri/compose/jenkins/workspace/runwithme/BE/gumid101/target/

ls -lrt

echo "complete and exit"

exit

EOF

해당 블로그가 매우 잘 정리되어 있다. 젠킨스를 가동하는 컨테이너의 경우 maven이 설치되어 있지 않음으로 따로 설치가 필요하다. 다른 방법으로는 위의 스크립트를 조금 수정하여 호스트에 maven을 설치하여 jar파일을 만드는 방법도 나쁘지 않다.

[[Jenkins] Docker를 활용한 Spring boot 프로젝트 CI & CD (tistory.com)](https://skyblue300a.tistory.com/14)

## 리액트 배포

ssh -t -t nonbiri@host.docker.internal <<EOF # 로컬 호스트 접속

cd /home/nonbiri/compose/jenkins/workspace/runwithmeFE/reactFE/run-with-me/

docker stop front #기존 컨테이너 중지

docker rm front #기존 컨테이너 삭제

docker rmi feimg:fetag #기존이미지 제거

docker build --tag feimg:fetag . #도커파일을 이용한 이미지 생성

docker run -d --name front -p 3000:3000 feimg:fetag #생성된 이미지를 컨테이너 가동에 사용

exit

EOF