# 통합 배포 가이드



## 버전 정보

- 호스트 머신(EC2) 설정
  - Docker : `20.10.17`
  - Docker-compose : `1.26.0`
- 웹 서버 설정
  - Nginx : `1.23.1-alpine`
    - 인증서 통합 관리 및 리버스 프록시 역할로 사용
- Back-end(Java) 설정
  - Java
    - `OpenJDK 11`
  - WAS, Web server
    - Spring boot 2.7.5의 내장 WAS spring-boot-starter-web-2.7.5
    - Spring boot 2.7.5의 내장 Web server spring-boot-starter-tomcat-2.7.5
  - IDE
    - `IntelliJ 2022.1.2`
- Back-end (Python) 설정
  - 서버 프레임워크
    - FastAPI 0.85
  - 서버
    - `uvicorn 0.18.3`
- 안드로이드 설정
  - IDE
    - Android Studio 2021.2.1.16 Chipmunk
  - 언어
    - Java 11
  - Minimum SDK

- API 21(Android 5.0, Lollipop)

# 외부 서비스 사전 세팅

## S3(이미지 저장용)

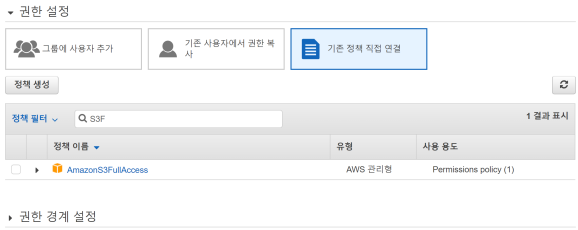
1. **버킷 생성**

- https://s3.console.aws.amazon.com/
- public access 허용 옵션으로 생성
- 아시아, 태평양(서울) ap-northeast-2 지역으로 생성


2. **IAM 사용자 추가**

(1) 엑세스 키 - 프로그래밍 방식 선택


(2) 기존 정책 직접 연결 - AmazonS3FullAccess 선택

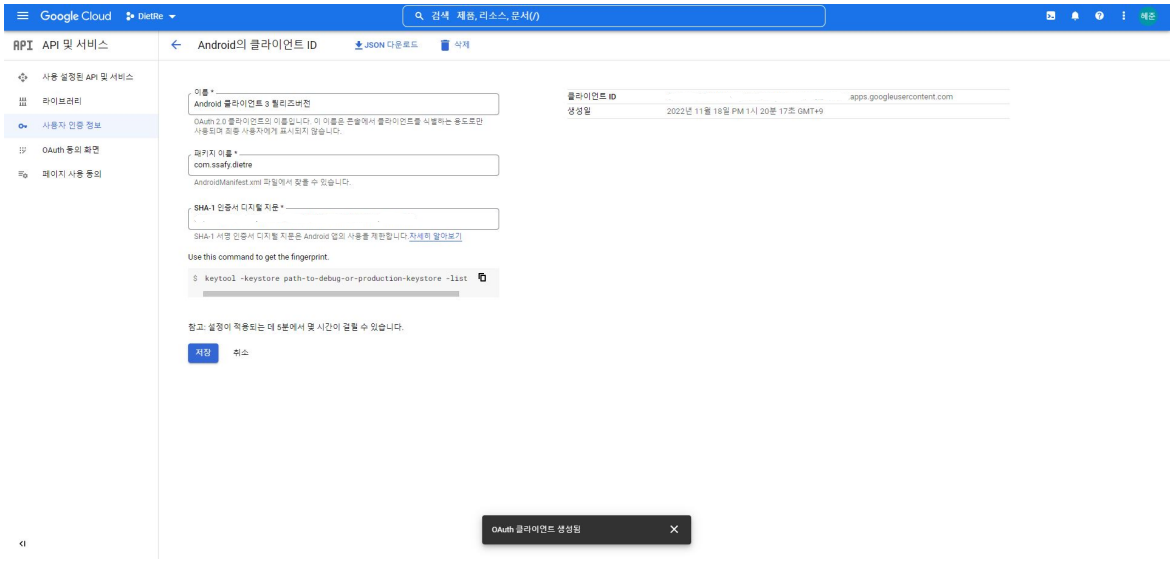



(3) 엑세스 키, 시크릿 키 발급 (추후 환경설정에서 사용)


## 소셜 로그인

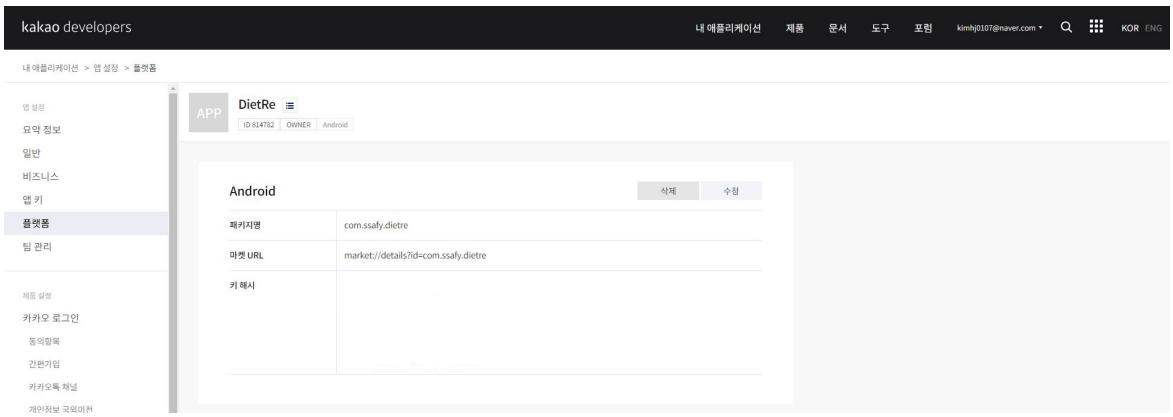
### 구글

1. 앱의 SHA-1 디지털 지문을 발급 받는다.
2. 디지털 지문을 구글 oauth 홈페이지(https://console.cloud.google.com/apis/credentials)에 등록한다.



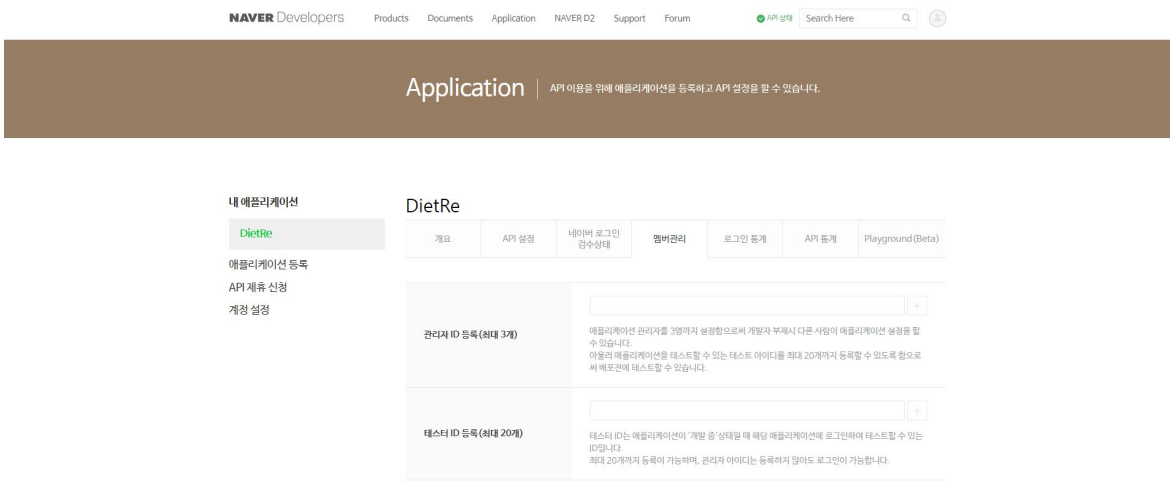

**카카오**

1. 앱의 SHA-1 디지털 지문을 발급 받고 Hash한 키를 얻는다.
2. 그 해시키를 카카오 개발자 홈페이지(https://developers.kakao.com/console/app)에 등록한다.




**네이버**

1. 네이버 개발자 홈페이지(https://developers.naver.com/apps/#/myapps/)에 네이버 ID를 등록한다.




## 서버 배포 과정

- 이하 과정은 모두 EC2 인스턴스에서 이루어집니다.
- 아래 내용 중 몇 가지의 변수화된 이름들이 있습니다. 이 내용은 다음과 같이 입력합니다.
- ${server_name} - 실행시키는 서버의 도메인 이름
- ${url} - https://${server_name}
- ${port} 방화벽이 걸려있지 않은 임의의 포트 (서로 중복되지만 않으면 어떤 포트이든 상관없음)
- ${userName} - db 사용자명
- ${pw} db 암호

## 폴더 구조 세팅

1. Dietre 서비스 배포를 위한 `dietre` 디렉토리 생성

```
cd /home/ubuntu
mkdir dietre
cd dietr
```

2. 다음과 같이 폴더를 세팅합니다.

```
mkdir db
mkdir deploy
mkdir secrets
```

3. docker-compose.yml 파일을 생성합니다.

```
vim ./deploy/docker-compose.yml
```

▼ docker-compose.yml 파일 내용

```
# docker-compose version
version: "3.8"

#container list
services:
  db:
    image: mysql:5.7
    container_name: mysql
    restart: always
    volumes:
      # Mount cotainer drive to real drive
      - /home/ubuntu/dietre/db:/docker-entrypoint-initdb.d
    networks:
      # network inside of a container
      - app-network
    ports:
      - "32000:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      TZ: "Asia/Seoul"
    privileged: true
  spring_backend:
    image: backend:1
    build:
      context: ../backend
    container_name: spring_backend
    restart: always
    ports:
      - "8080:8080"
    environment:
      TZ: "Asia/Seoul"
      # Spring application.properties DB
      SPRING_DATASOURCE_URL: "jdbc:mysql://db:3306/dietre?userUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&z
      server.address: 0.0.0.0
    networks:
      - app-network
    depends_on:
      - db
  dietre_media:
    build:
      context: ../dietre-media
    container_name: dietre_media
    restart: always
    ports:
      - "9000:9000"
    environment:
      TZ: "Asia/Seoul"
      server.address: 0.0.0.0
    networks:
      - app-network
  python_backend:
    build:
      context: ../python
      dockerfile: Dockerfile
    container_name: python_backend
    restart: always
    ports:
      - 9090:9090
    environment:
      TZ: "Asia/Seoul"
    networks:
      - app-network
    depends_on:
      - db
  certbot:
    depends_on:
      - nginx
    image: certbot/certbot
    container_name: certbot
    volumes:
      - /home/ubuntu/dietre/certbot-etc:/etc/letsencrypt
      - /home/ubuntu/dietre/dist:/var/www/html
    command: renew
  nginx:
    image: nginx:alpine
    container_name: nginx
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/dietre/dist:/var/www/html
      - /home/ubuntu/dietre/nginx-conf:/etc/nginx/conf.d
```

```
      - /home/ubuntu/dietre/certbot-etc:/etc/letsencrypt
  jenkins:
    restart: always
    image: jenkins/jenkins:lts
    user: root
    container_name: jenkins
    ports:
      - "8090:8080"
      - "50000:50000"
    volumes:
      - /jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/ubuntu/dietre/secrets:/var/jenkins_home/secrets
      - /home/ubuntu/dietre/images:/var/jenkins_home/images
volumes:
  certbot-etc:
  dist:
  nginx-conf:
  secrets:
  images:
networks:
  app-network:
    driver: bridge
```

4. secret 파일 입력

```
vim ./secrets/application-secrets.properites
```

- 파일 내용은 다음과 같다. (client_id와 client_secrets는 위에서 발급받은 값을 직접 입력한다.)

```
server.port=8080
server.address=localhost
server.servlet.contextPath=/api
#database
spring.jpa.hibernate.naming.implicit-strategy=org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
spring.data.web.pageable.one-indexed-parameters=true
spring.datasource.url=jdbc:mysql://localhost:3306/dietre?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDate
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.hikari.username=${userName}
spring.datasource.hikari.password=${pw}
spring.datasource.initialization-mode=always
spring.datasource.sql-script-encoding=UTF-8
# jwt
jwt.secret=${secret}
jwt.expiration=1800000
# google
spring.security.oauth2.client.registration.google.redirect-uri=https://${server_name}/api/login/oauth2/code/google
spring.security.oauth2.client.registration.google.url=https://accounts.google.com/o/oauth2/v2/auth
spring.security.oauth2.client.registration.google.callback-url=https://${server_name}/api/test/permit-all
spring.security.oauth2.client.registration.google.client-id=${client_id}.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=${client_secrets}
spring.security.oauth2.client.registration.google.scope=profile,email
# kakao
spring.security.oauth2.client.registration.kakao.client-id=${client_id}
spring.security.oauth2.client.registration.kakao.client-secret=${client_secrets}
spring.security.oauth2.client.registration.kakao.scope=profile_nickname, account_email
spring.security.oauth2.client.registration.kakao.client-name=Kakao
spring.security.oauth2.client.registration.kakao.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.kakao.redirect-uri=https://${server_name}/api/login/oauth2/code/kakao
spring.security.oauth2.client.registration.kakao.client-authentication-method=POST
spring.security.oauth2.client.provider.kakao.authorization-uri=https://kauth.kakao.com/oauth/authorize
spring.security.oauth2.client.provider.kakao.token-uri=https://kauth.kakao.com/oauth/token
spring.security.oauth2.client.provider.kakao.user-info-uri=https://kapi.kakao.com/v2/user/me
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
# naver
spring.security.oauth2.client.registration.naver.client-id=${client_id}
spring.security.oauth2.client.registration.naver.client-secret=${client_secrets}
spring.security.oauth2.client.registration.naver.scope=name,email
spring.security.oauth2.client.registration.naver.client-name=Naver
spring.security.oauth2.client.registration.naver.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.naver.redirect-uri=https://${server_name}/api/login/oauth2/code/naver
#spring.security.oauth2.client.registration.naver.client-authentication-method=POST
spring.security.oauth2.client.provider.naver.authorization-uri:https://nid.naver.com/oauth2.0/authorize
spring.security.oauth2.client.provider.naver.token-uri:https://nid.naver.com/oauth2.0/token
spring.security.oauth2.client.provider.naver.user-info-uri:https://openapi.naver.com/v1/nid/me
spring.security.oauth2.client.provider.naver.user-name-attribute:response
```

- secret 파일을 하나 더 생성해준다.

```
vim ./secrets/application-image-secrets.properites
```

- 내용은 다음과 같다. access_key와 secret_key는 s3 서비스에서 발급받은 값을 입력한다.

```
custom.path.upload-images=/var/
server.port=9000
server.address=localhost
server.servlet.contextPath=/image
cloud.aws.credentials.access-key=${access_key}
cloud.aws.credentials.secret-key=${secret_key}
cloud.aws.region.static=ap-northeast-2
cloud.aws.s3.bucket=dietre-images
cloud.aws.stack.auto=false
spring.servlet.multipart.maxFileSize=1000MB
spring.servlet.multipart.maxRequestSize=1000MB
```

## DB Init

1. `init.sql` 작성

```
vim ./db/init.sql
```

- `db` : 스키마와 유저를 생성하기 위한 sql이 들어가는 디렉토리
- 만약 vim이 설치되어 있지 않을 경우 vim 설치 후 진행

```
sudo apt-get update
sudo apt-get install vim
```

- `init.sql` 내용

```
create database IF NOT EXISTS `dietre` collate utf8mb4_general_ci;
create user '${userName}'@'%' identified by '${pw}';
grant all privileges on *.* to ${userName}@'%';
flush privileges;
```

## Docker 설치

1. 유틸 설치

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

2. 키 생성

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

3. Repository 추가 후 Update

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

4. Docker 설치

```
sudo apt install docker-ce
```

5. Docker 설치 확인

```
sudo systemctl status docker
```

6. Docker 권한 설정

```
sudo usermod -aG docker ubuntu
```

## Docker-compose

1. Docker-compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bir
```

2. 심볼릭 링크 생성, 실행 권한 부여

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
```

3. 설치 확인

```
docker-compose -v
```

## Nginx + SSL

1. Nginx 배포를 위한 디렉토리 생성

```
mkdir dist
mkdir certbot-etc
mkdir nginx-conf
```

- `dist` : front-end 빌드한 결과를 넣는 디렉토리
- `certbot-etc` : 인증서 디렉토리
- `nginx-conf` : Nginx의 config 파일 저장용 디렉토리

2. certbot 컨테이너로 인증서 생성을 위한 `conf` 파일 작성

```
vim ./nginx-conf/nginx.conf
```

- `nginx.conf` 파일 내용

```
server {
  listen 80;
  listen [::]:80;

  server_name ${server_name};
  index index.html index.htm;
  root /var/www/html;

  location ~ /.well-known/acme-challenge {
    allow all;
    root /var/www/html;
  }

  location / {
    try_files $uri $uri/ /index.html;
  }

  location ~ /\.ht {
    deny all;
  }

  location = /favicon.ico {
    log_not_found off; access_log off;
  }

  location = /robots.txt {
    log_not_found off; access_log off; allow all;
  }

  location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
    expires max;
    log_not_found off;
  }
}
```

3. Docker-compose로 인증서 생성하기 위한 `docker-compose-init-certbot.yml`

```
vim ./deploy/docker-compose-init-certbot.yml
```

- docker-compose-init-certbot.yml (이메일 자리에 이메일 입력)

```
version: '3'

services:
    certbot:
        depends_on:
            - nginx
        image: certbot/certbot
        volumes:
            - /home/ubuntu/dietre/certbot-etc:/etc/letsencrypt
            - /home/ubuntu/dietre/dist/:/var/www/html
        command: certonly --webroot --webroot-path=/var/www/html --email ${이메일} --agree-tos --no-eff-email --staging -d ${url}
    nginx:
        image: nginx:alpine
        restart: unless-stopped
        ports:
            - "80:80"
            - "443:443"
        volumes:
            - /home/ubuntu/dietre/dist:/var/www/html
            - /home/ubuntu/dietre/nginx-conf:/etc/nginx/conf.d
            - /home/ubuntu/dietre/certbot-etc:/etc/letsencrypt
        networks:
            - app-network

volumes:
    certbot-etc:
    dist:
    nginx-conf:

networks:
    app-network:
        driver: bridge
```

4. `docker-compose up` 으로 컨테이너 생성

```
docker-compose -f ./docker-compose-init-certbot.yml up
```

- 정상 작동시 certbot 컨테이너는 Exit 0으로 정상 종료되며 `./certbot-etc` 에서 확인 가능

5. 전 단계는 인증서를 임시로 발급하는 staging 단계에 해당되기 때문에 다음과 같이 docker-compose-init-certbot.yml 파일에서 한줄을 변경해서 정식 인증을 받는다.

```
command: certonly --webroot --webroot-path=/var/www/html --email ${이메일} --agree-tos --no-eff-email --force-renewal -d ${주소}
```

6. Container 재생성

```
cd deploy
docker-compose up -f docker-compose-init-certbot.yml --force-recreate --no-deps certbot
cd ..
```

7. SSL 설정 다운로드

```
curl -sSLo nginx-conf/options-ssl-nginx.conf https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot_nginx/
```

8. Nginx 설정 수정

```
vim ./nginx-conf/default.conf
```

- `nginx.conf` 내용

```
server {
  listen 80;
  listen [::]:80;

  server_name ${server_name};
  index index.html index.htm;
  root /var/www/html;

  location ~ /.well-known/acme-challenge {
    allow all;
    root /var/www/html;
  }

  location / {
    rewrite ^ https://$host$request_uri? permanent;
    try_files $uri $uri/ /index.html;
  }
}

server {
  listen 443 ssl http2;
  listen [::]:443 ssl http2;
  server_name ${server_name};

  index index.html index.htm;
  root /var/www/html;

  server_tokens off;
  client_max_body_size 100M;

  ssl_certificate /etc/letsencrypt/live/k7a105.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/k7a105.p.ssafy.io/privkey.pem;
  ssl_trusted_certificate /etc/letsencrypt/live/k7a105.p.ssafy.io/chain.pem;
  include /etc/nginx/conf.d/options-ssl-nginx.conf;

  add_header X-Frame-Options "SAMEORIGIN" always;
  add_header X-XSS-Protection "1; mode=block" always;
  add_header X-Content-Type-Options "nosniff" always;
  add_header Referrer-Policy "no-referrer-when-downgrade" always;
  add_header Content-Security-Policy "default-src * data: 'unsafe-eval' 'unsafe-inline'" always;
  # add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload" always;
  # enable strict transport security only if you understand the implications
```

```
  location / {
    try_files $uri $uri/ /index.html;
  }

  location /api {
    proxy_pass http://${server_name}:${port};
    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;
    proxy_set_header X-Forwarded-Proto $scheme;
  }

  location /image {
    proxy_pass http://${server_name}:${port};
    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;
    proxy_set_header X-Forwarded-Proto $scheme;
    client_max_body_size 0;
  }

  location /internal {
    proxy_pass http://${server_name}:${port};
    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;
    proxy_set_header X-Forwarded-Proto $scheme;
  }

}
```

9. crontab에 인증서 자동 갱신을 위한 스크립트 등록

```
vim ./deploy/ssl_renew.sh
```

- `ssl_renew.sh`

```
#!/bin/bash
COMPOSE="/usr/local/bin/docker-compose --no-ansi"
DOCKER="/usr/bin/docker"

cd /home/ubuntu/dietre/deploy
$COMPOSE run certbot renew && $COMPOSE kill -s SIGHUP nginx
$DOCKER system prune -af
```

```
chmod +x ./deploy/ssl_renew.sh
sudo crontab -e
0 12 * * * /home/ubuntu/dietre/dietre/ssl_renew.sh >> /var/log/cron.log 2>&1
```

9. Nginx, certbot 컨테이너 재생성 및 전체 컨테이너 생성

```
docker-compose up
docker-compose up --force-recreate --no-deps nginx
docker-compose up --force-recreate --no-deps certbot
```

## Jenkins

1. 최초 설치 및 실행
   - 위에 언급된 docker-compose.yml 파일에서 jenkins부분의 캡쳐입니다.

```
jenkins:
    restart: always
    image: jenkins/jenkins:lts
    user: root
    container_name: jenkins
    ports:
```

```
      - "${port}:8080"
      - "50000:50000"
    volumes:
      - /jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/ubuntu/dietre/secrets:/var/jenkins_home/secrets
```

- 다음 명령어로 실행합니다.

```
cd deploy
docker-compose up -d jenkins
cd ..
```

2. jenkins 컨테이너 내부에 docker, docker-compose 설치

```
docker exec -it --user root <container id> bash

curl https://get.docker.com/ > dockerinstall && chmod 777 dockerinstall && ./dockerinstall

curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/bin/docker-com

chmod +x /usr/bin/docker-compose

exit
```

3. 호스트 머신에서 docker.sock 파일의 접근권한 변경

```
sudo chmod 666 /var/run/docker.sock
```

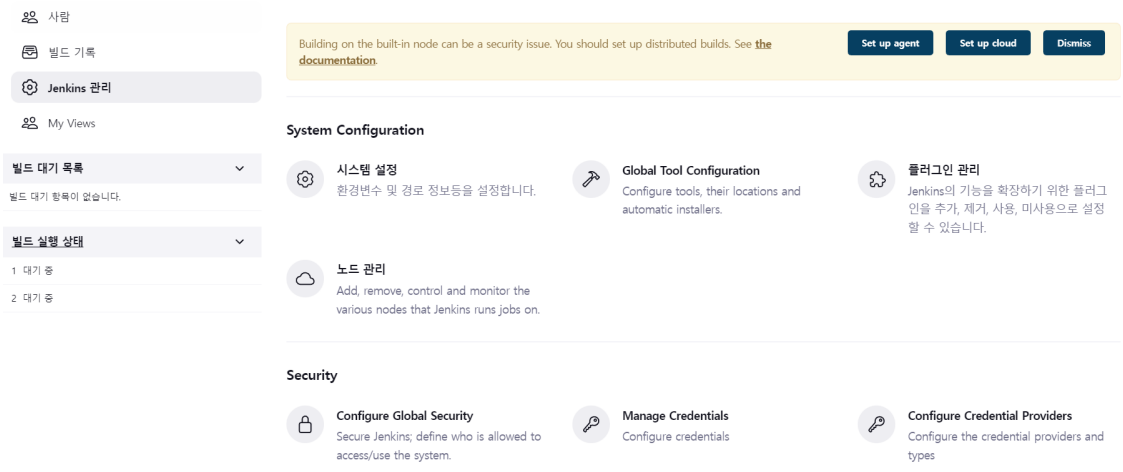
4. 도메인:${port} 주소로 jenkins에 접속

- 최초 비밀번호는 호스트 머신에서 다음 명령을 통해서 획득 가능

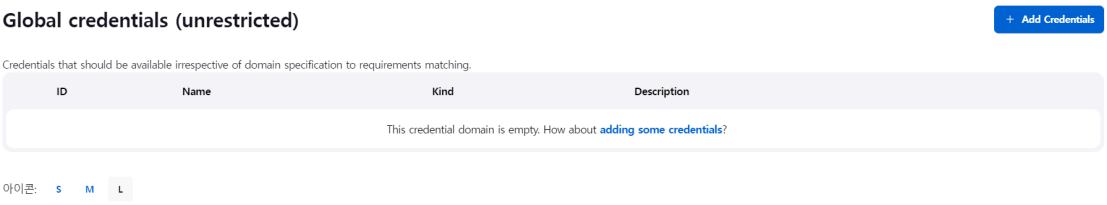
```
docker exec -it <container id> bash
cat var/jenkins_home/secrets/initialAdminPassword
```

5. 기본 플러그인으로 설치 후 안내에 따라서 초기 설정
6. gitlab 설치 (플러그인 관리에서 설치)
7. 자신의 정보에 따라서 gitlab 연동 설정

▼ 캡쳐

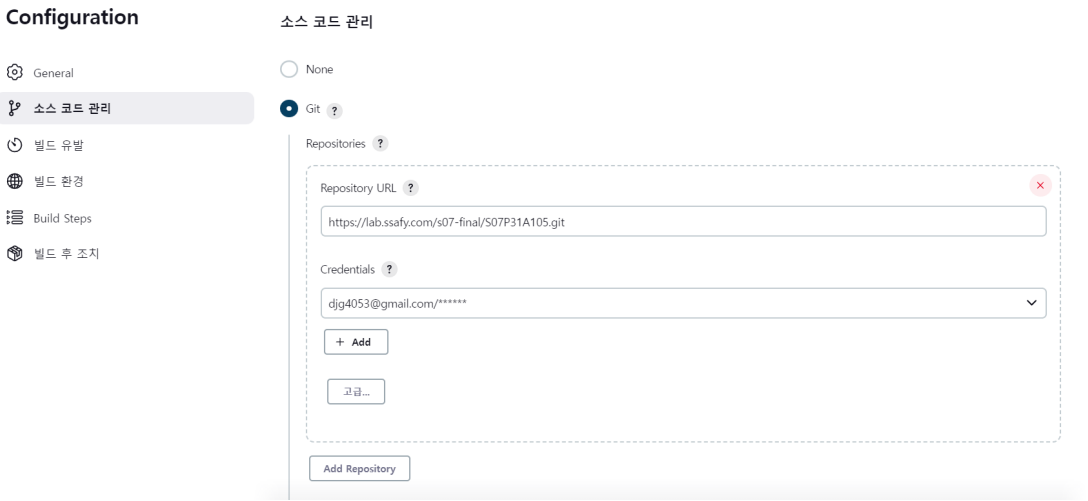



- 우선 Manage Credentials로 간후 adding some credentials 클릭



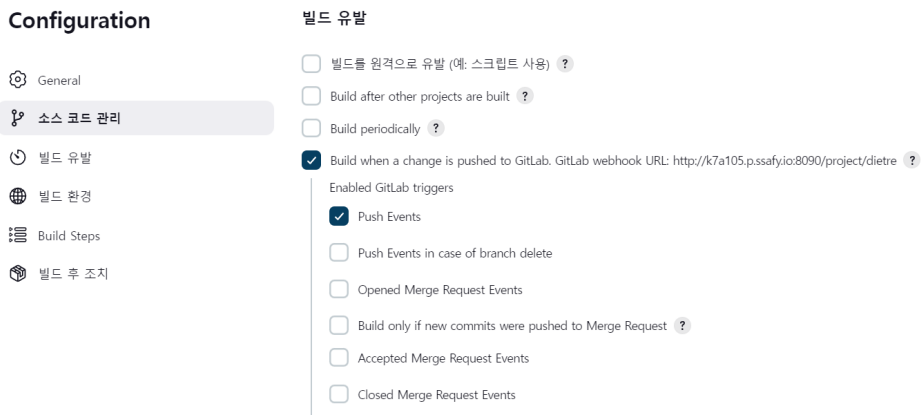

- username and password로 선택 후 Username에 깃랩 이메일 패스워드에 패스워드 설정


- 이제 내 프로젝트로 들어간다음 소스 코드 관리로 들어가서 다음과 같이 입력

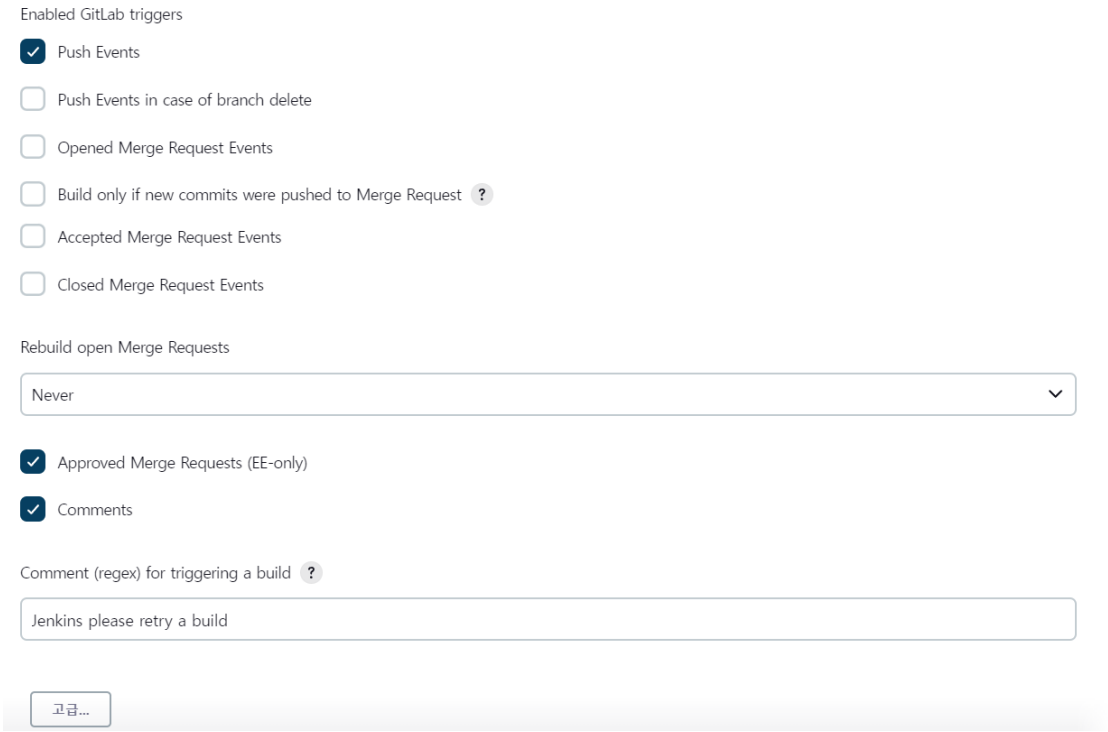



- Branch Specifier에는 내가 실제로 빌드 과정에 이용하고 싶은 브랜치를 입력

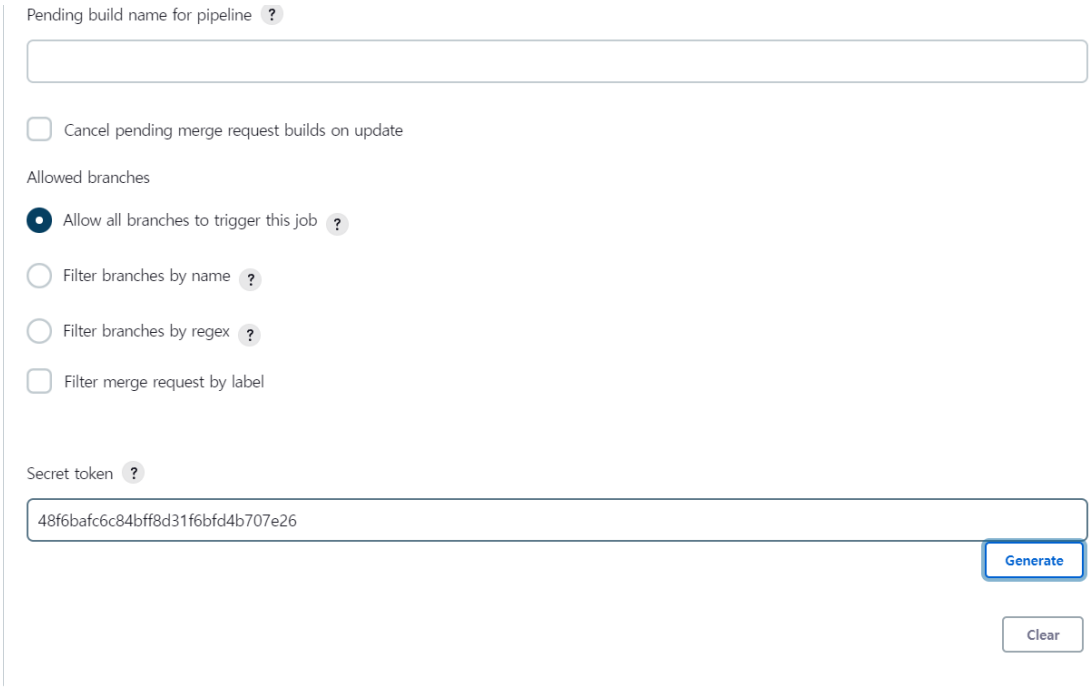

- 이제 살짝 화면을 내려 gitlab hook을 연결, Build when ~ 클릭 후 Push Events클릭
- 옆에 써있는 GitLab webhook URL은 후에 쓰기 때문에 위치를 기억해두기



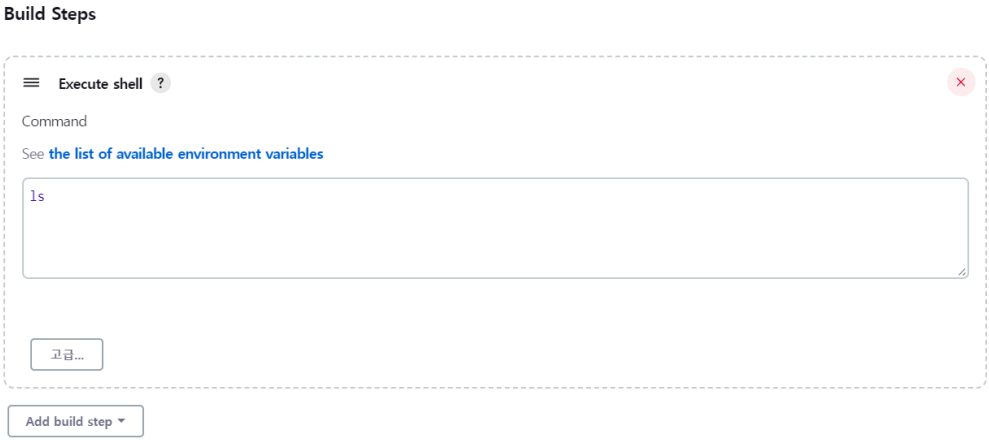

- 밑에 고급 버튼을 클릭



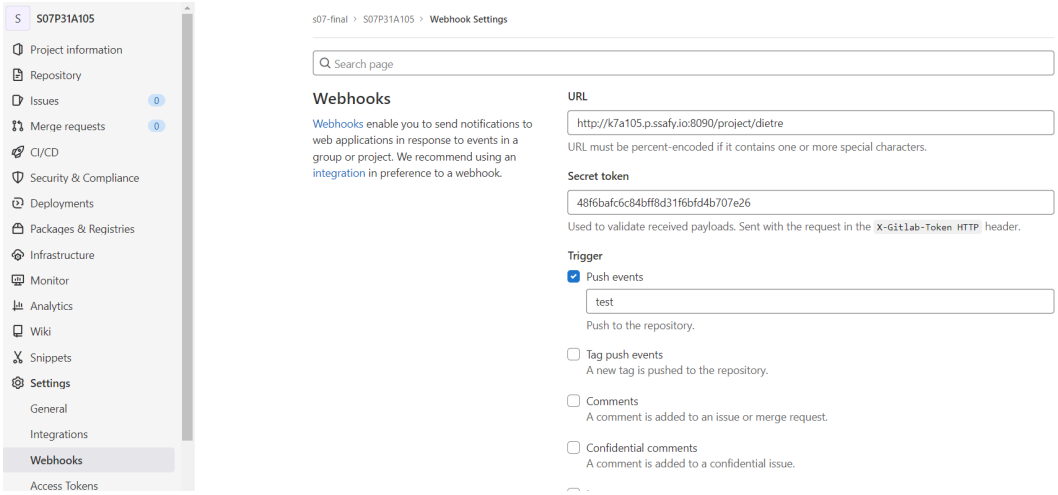

- 밑으로 내려서 secret tokens라고 된 칸을 찾은 후 생성된 키를 복사



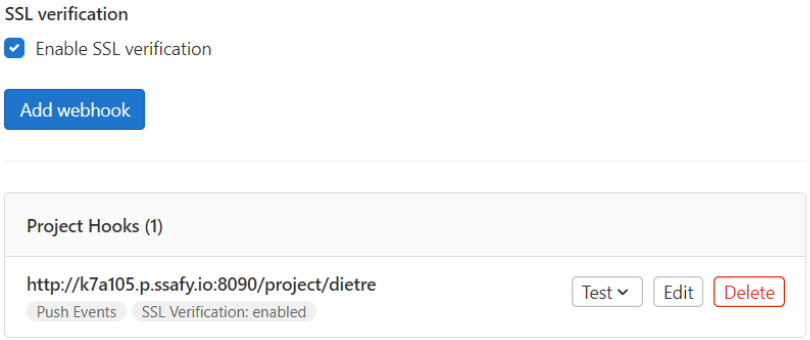

- 테스트를 위해 밑으로 내려서 Build Steps에 Execute Shell → ls를 입력한다.




- 이제 깃랩 레포로 가서 설정에 있는 Settings - webhooks 클릭

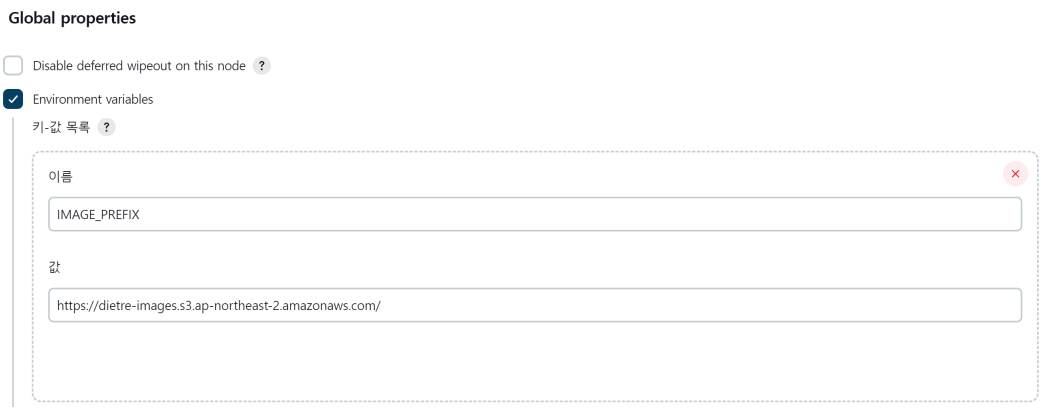



- 요렇게 위에서 얻은 webhook URL과 토큰 값을 복사한 후 Push Events에 브랜치 명을 입력하고(어떤 브랜치에 푸쉬되면 훅이 발동될지 선택)
- 밑에 내려서 Add Webhook 클릭 후 테스트



8. Global Tool Configuration 설정에서 gradle 6.7 버전 설치
10. 환경변수 관련 설정
    - Configure System에 가서 다음과 같이 이용할 환경변수들을 세팅해줍니다.
    - 이 프로젝트에 사용되는 환경변수는 다음과 같습니다.
        - IMAGE_PREFIX - S3 저장소 주소
        - IMAGE_UPLOAD_URL - ${server_name}
        - MYSQL_ROOT_PASSWORD - ${db_password}

    ▼ 캡쳐



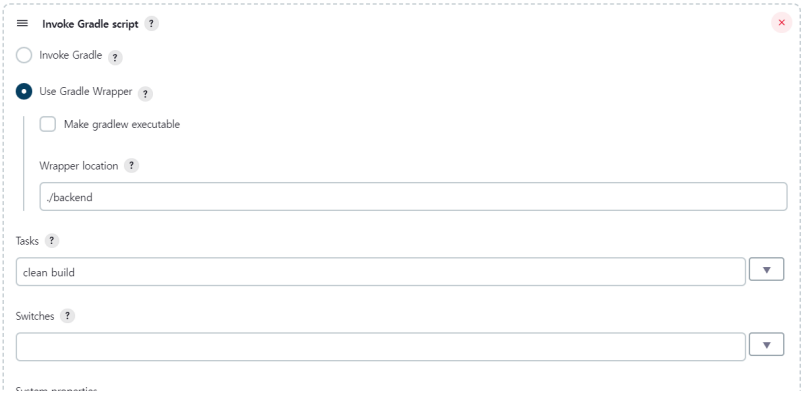

10. 다음과 같이 빌드합니다.
    - 1단계

    ▼ 캡쳐 (전문 별도 기재)

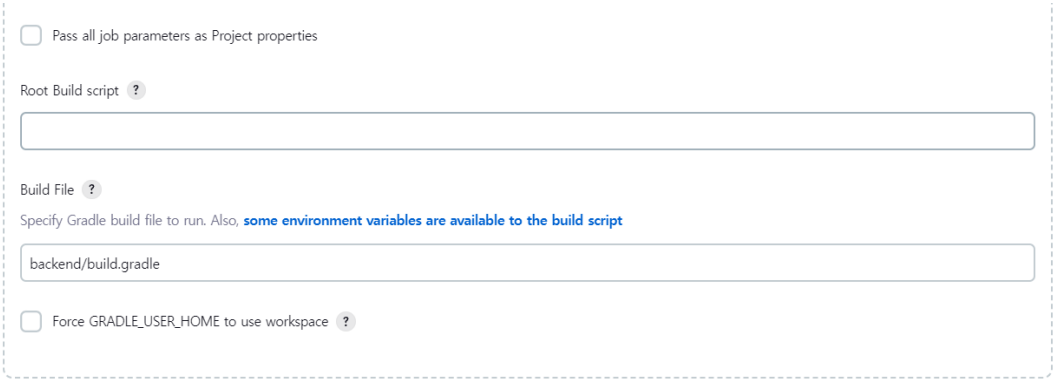
```
cp ../../secrets/application-secret.properties backend/src/main/resources/application-secret.properties
cp ../../secrets/application-image-secret.properties dietre-media/src/main/resources/application-image-secret.properties
cd backend
chmod 755 gradlew
cd ../dietre-media
chmod 755 gradlew
```


    - 2단계

    ▼ 캡쳐




이 후 '고급' 을 클릭 후 다음과 같이 설정해줍니다.




이 후  gradle script 탭을 하나 더 생성해 dietre-media에 대해 똑같은 작업을 수행합니다.

- 3단계 - Execute shell 탭 하나 추가 후 하기 내용 작성

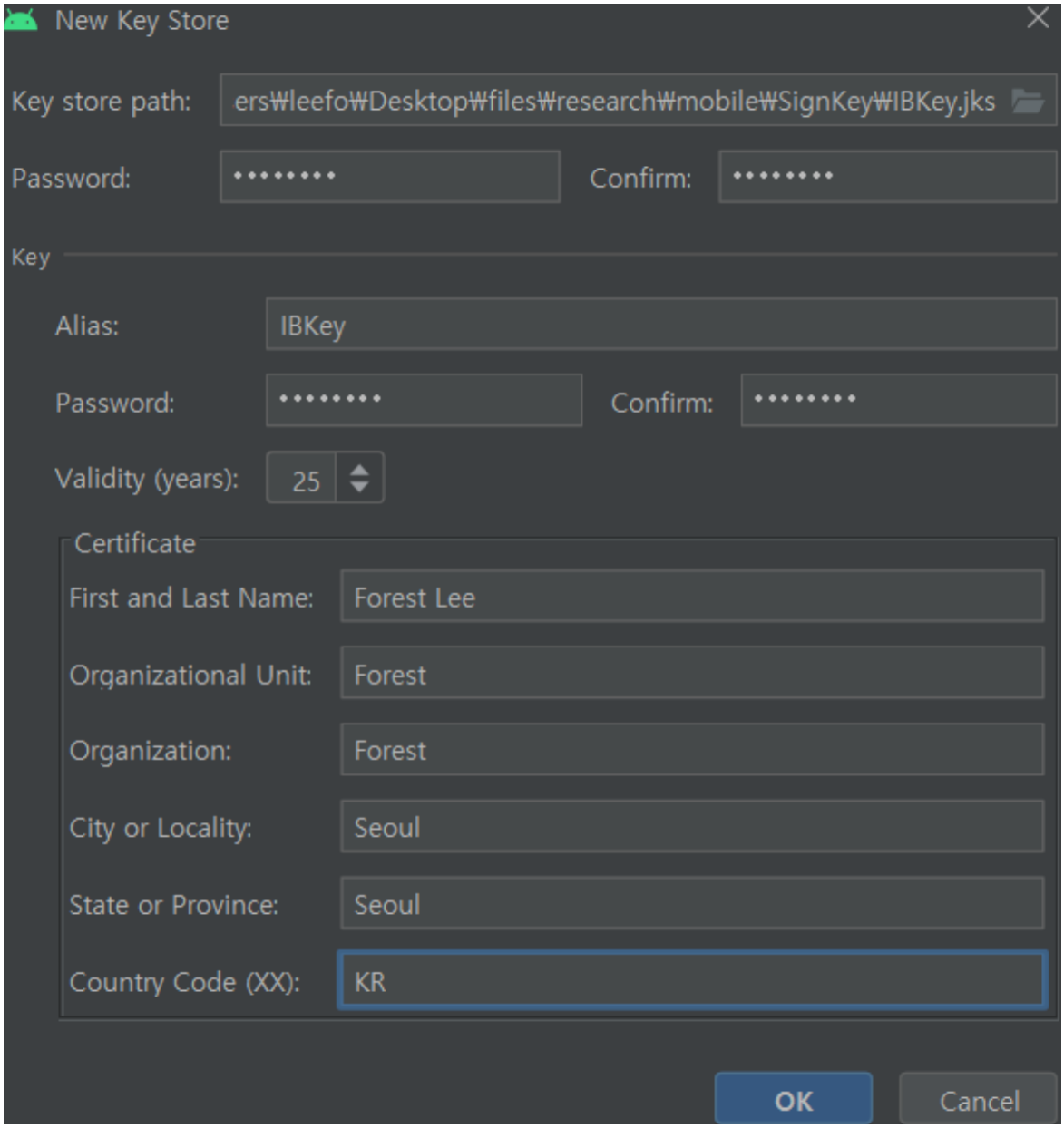
  ▼ 캡쳐

```
cd deploy
docker-compose up --build -d python_backend spring_backend dietre_media certbot
docker-compose up -d --force-recreate --no-deps nginx
```

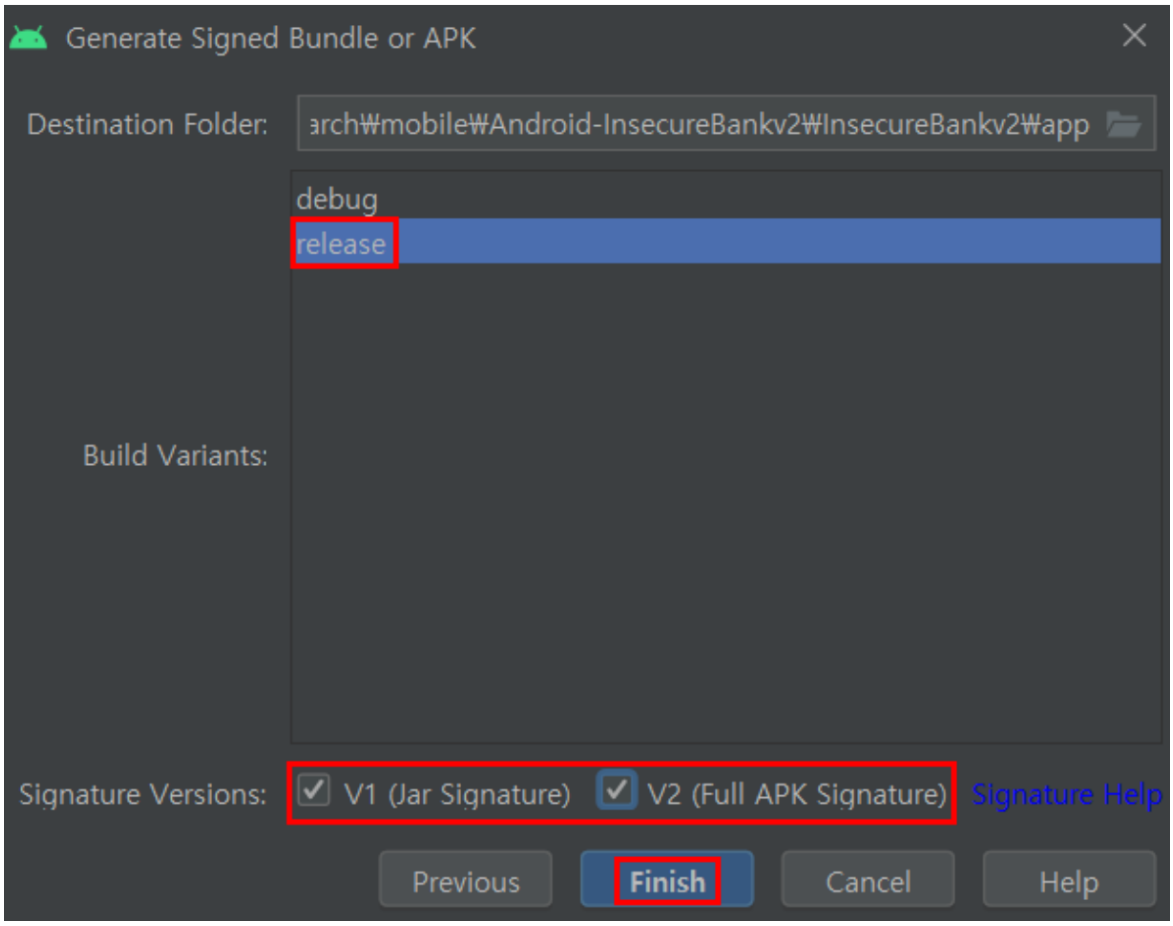

## 안드로이드 빌드

- 아래 내용은 https://liveyourit.tistory.com/158 내용을 참조하였습니다.
- 완성된 apk 파일을 함께 첨부하였습니다.

1. 안드로이드 스튜디오에서 Build - Generate Signed Bundle - APK 클릭
2. Create new 클릭 후 파일 저장할 경로 지정
3. 다음과 예시와 같이 키 파일 생성




4. 다음과 같은 옵션으로 생성


Generate Signed Bundle or APK
Destination Folder:
arch₩mobile₩Android-InsecureBankv2₩InsecureBankv2₩app
debug
release
Build Variants:
Signature Versions:
V1 (Jar Signature)
V2 (Full APK Signature)
Signature Help
Previous
Finish
Cancel
Help