



통합 배포 가이드

[버전 정보](#)

[외부 서비스 사전 세팅](#)

[S3\(이미지 저장용\)](#)

[소셜 로그인](#)

[서버 배포 과정](#)

[폴더 구조 세팅](#)

[DB Init](#)

[Docker 설치](#)

[Docker-compose](#)

[Nginx + SSL](#)

[Jenkins](#)

[안드로이드 빌드](#)

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

AWS 리전

아시아 태평양(서울) ap-northeast-2 ▼

기존 버킷에서 설정 복사 - 선택 사항

다음 구성의 버킷 설정만 복사됩니다.

버킷 선택

객체 소유권 [Info](#)

다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

☒ ACL 비활성화됨(권장)

이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.

☐ ACL 활성화됨

이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.

객체 소유권

버킷 소유자 적용

이 버킷의 퍼블릭 액세스 차단 설정

퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책, 액세스 지점 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 이 버킷 및 해당 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 모든 퍼블릭 액세스 차단을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지점에만 적용됩니다. AWS에서는 모든 퍼블릭 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 이 버킷 또는 내부 객체에 대한 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

☐ 모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

2. IAM 사용자 추가

- (1) 액세스 키 - 프로그래밍 방식 선택

사용자 세부 정보 설정

동일한 액세스 유형 및 권한을 사용하여 한 번에 여러 사용자를 추가할 수 있습니다. 자세히 알아보기

사용자 이름*

[다른 사용자 추가](#)

AWS 액세스 유형 선택

이러한 사용자가 주로 AWS에 액세스하는 방법을 선택합니다. 프로그래밍 방식의 액세스만 선택하면 사용자가 위임된 역할을 사용하여 콘솔에 액세스하는 것을 방지할 수 없습니다. 액세스 키와 자동 생성된 암호가 마지막 단계에서 제공됩니다. 자세히 알아보기

- AWS 자격 증명 유형 선택***
- ☒ **액세스 키 - 프로그래밍 방식 액세스**
AWS API, CLI, SDK 및 기타 개발 도구에 대해 액세스 키 ID 및 비밀 액세스 키 둘(둘)을 생성합니다.
 - ☐ **암호 - AWS 관리 콘솔 액세스**
사용자가 AWS Management Console에 로그인할 수 있도록 허용하는 비밀번호를 생성합니다.

(2) 기존 정책 직접 연결 - AmazonS3FullAccess 선택

▼ 권한 설정

그룹에 사용자 추가

기존 사용자에서 권한 복사

기존 정책 직접 연결

정책 생성

정책 필터 1 결과 표시

정책 이름	유형	사용 용도
<input type="checkbox"/> AmazonS3FullAccess	AWS 관리형	Permissions policy (1)

▼ 권한 경계 설정

(3) 액세스 키, 시크릿 키 발급 (추후 환경설정에서 사용)

사용자 추가

1 2 3 4 5

✓ **성공**
아래에 표시된 사용자를 생성했습니다. 사용자 보안 자격 증명을 보고 다운로드할 수 있습니다. AWS Management Console 로그인을 위한 사용자 지침을 이메일로 보낼 수도 있습니다. 지금이 이 자격 증명을 다운로드할 수 있는 마지막 기회입니다. 하지만 언제든지 새 자격 증명을 생성할 수 있습니다.

AWS Management Console 액세스 권한이 있는 사용자가 <https://571127262829.signin.aws.amazon.com/console>에 로그인할 수 있습니다.

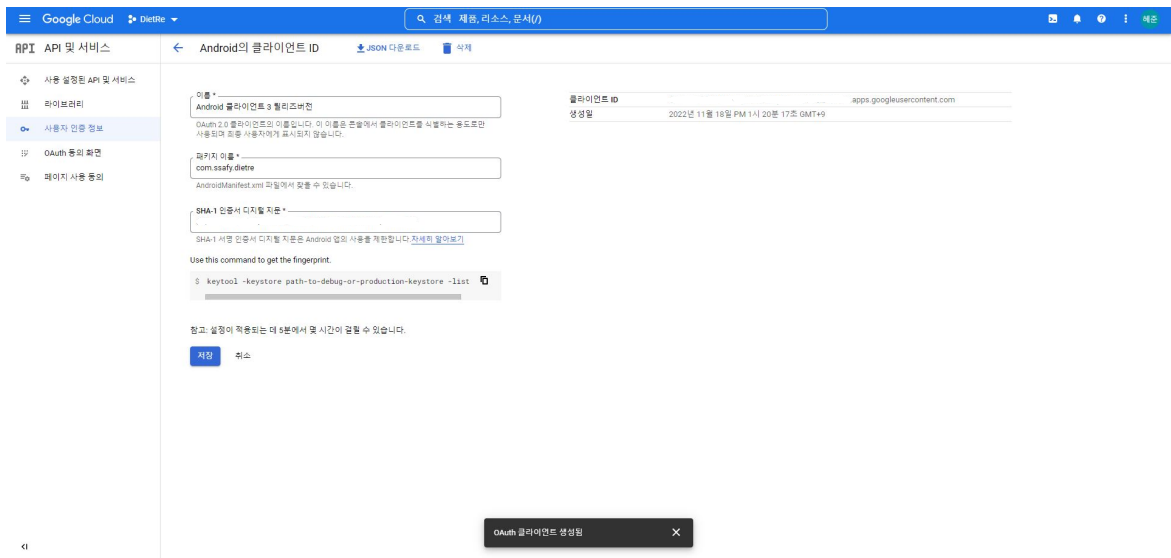
[.csv 다운로드](#)

	사용자	액세스 키 ID	비밀 액세스 키
▶	✓ user		***** 표시

소셜 로그인

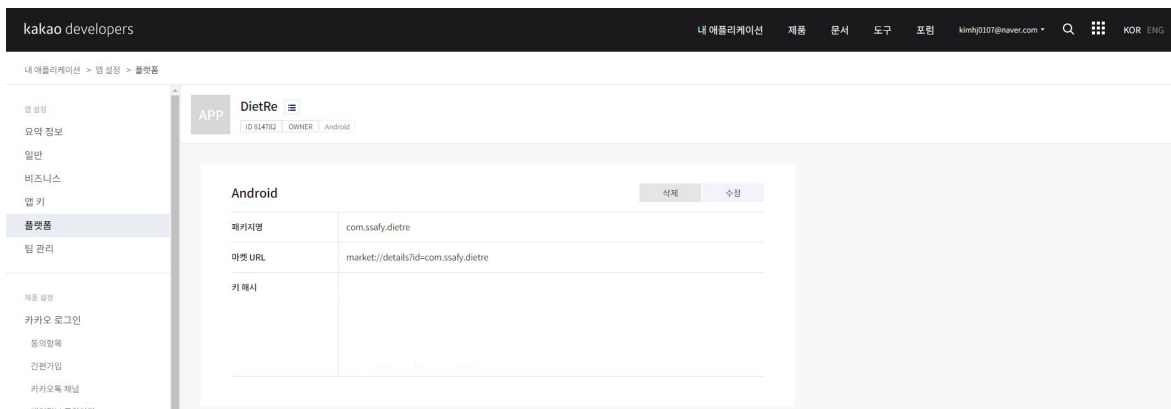
구글

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



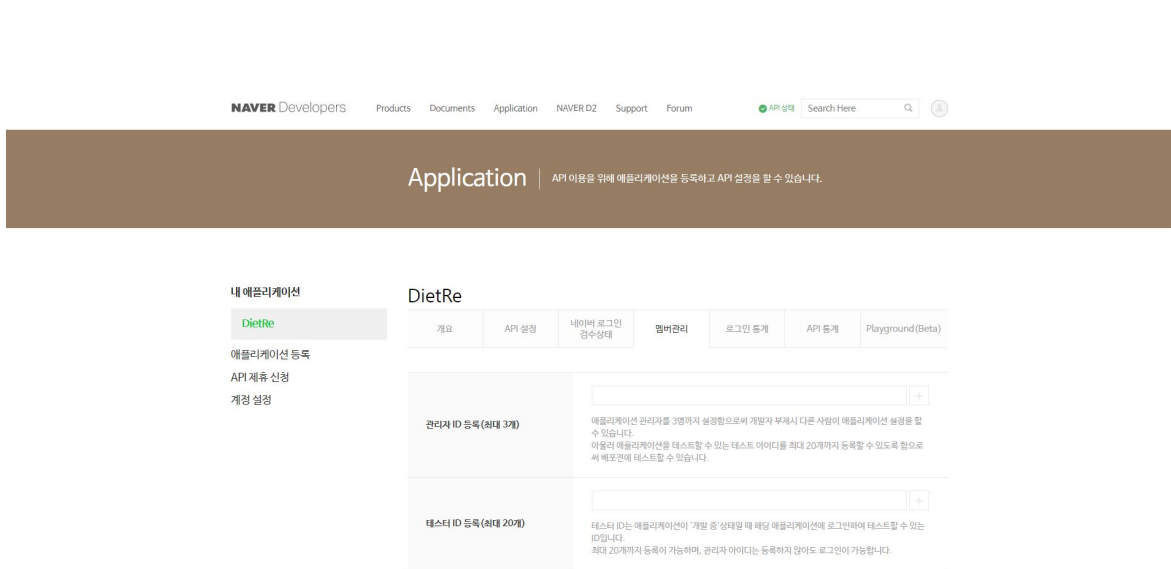
카카오

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 dietre
```

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&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.properties
```

- 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.properties
```

- 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/bin/docker-compose
```

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 캡처입니다.

```

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-compose

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

▼ 캡처

The screenshot shows the Jenkins configuration interface. On the left is a sidebar with navigation links: '사람' (People), '빌드 기록' (Build History), 'Jenkins 관리' (Jenkins Management), and 'My Views'. Below these are expandable sections for '빌드 대기 목록' (Build Queue) and '빌드 실행 상태' (Build Execution Status). The main content area has a yellow warning banner at the top about distributed builds. Below this, the 'System Configuration' section includes '시스템 설정' (System Settings), 'Global Tool Configuration', and '플러그인 관리' (Plugin Management). The 'Security' section includes 'Configure Global Security', 'Manage Credentials', and 'Configure Credential Providers'.

- 우선 Manage Credentials로 간후 adding some credentials 클릭

Global credentials (unrestricted)

[+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
This credential domain is empty. How about adding some credentials?			

아이콘: [S](#) [M](#) [L](#)

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

Kind

Username with password

Scope ?

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

Username ?

☐ Treat username as secret ?

Password ?

ID ?

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

Configuration

소스 코드 관리

General

소스 코드 관리

빌드 유발

빌드 환경

Build Steps

빌드 후 조치

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://lab.ssfy.com/s07-final/S07P31A105.git

Credentials ?

djg4053@gmail.com/*****

+ Add

고급...

Add Repository

- Branch Specifier에는 내가 실제로 빌드 과정에 이용하고 싶은 브랜치를 입력

Credentials ?

djg4053@gmail.com/*****

+ Add

고급...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/test

Add Branch

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

Configuration

General

소스 코드 관리

빌드 유발

빌드 환경

Build Steps

빌드 후 조치

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://k7a105.p.ssafy.io:8090/project/dietre ?
 - 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

- 밑에 고급 버튼을 클릭

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

Never ▼

- ☒ Approved Merge Requests (EE-only)
- ☒ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급...

- 밑으로 내려서 **secret tokens**라고 된 칸을 찾은 후 생성된 키를 복사

Pending build name for pipeline ?

- ☐ Cancel pending merge request builds on update

Allowed branches

- ☒ Allow all branches to trigger this job ?
- ☐ Filter branches by name ?
- ☐ Filter branches by regex ?
- ☐ Filter merge request by label

Secret token ?

48f6bafc6c84bff8d31f6bfd4b707e26

Generate

Clear

- 테스트를 위해 밑으로 내려서 **Build Steps**에 **Execute Shell** → **ls**를 입력한다.

Build Steps

Execute shell ?

Command

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

ls

고급...

Add build step ▾

- 이제 깃랩 레포로 가서 설정에 있는 Settings - webhooks 클릭

S S07P31A105

Project information

Repository

Issues 0

Merge requests 0

CI/CD

Security & Compliance

Deployments

Packages & Registries

Infrastructure

Monitor

Analytics

Wiki

Snippets

Settings

General

Integrations

Webhooks

Access Tokens

s07-final > S07P31A105 > Webhook Settings

Search page

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://k7a105.p.ssafy.io:8090/project/dietre

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

Secret token

48f6baf6c84bf8d31f6bfd4b707e26

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

Trigger

☒ Push events

test

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.

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

SSL verification

- ☒ Enable SSL verification

Add webhook

Project Hooks (1)

http://k7a105.p.ssafy.io:8090/project/dietre

Push Events SSL Verification: enabled

Test ▾

Edit

Delete

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}

▼ 캡처

Global properties

☐ Disable deferred wipeout on this node ?

☒ Environment variables

키-값 목록 ?

이름	IMAGE_PREFIX
값	https://dietre-images.s3.ap-northeast-2.amazonaws.com/

10. 다음과 같이 빌드합니다.

- 1단계

▼ 캡처 (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
```

Execute shell ?

Command

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

```
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
```

고급...

- 2단계

▼ 캡처

Invoke Gradle script ?

☐ Invoke Gradle ?

☒ Use Gradle Wrapper ?

☐ Make gradlew executable

Wrapper location ?

Tasks ?

▼

Switches ?

▼

이 후 '고급' 을 클릭 후 다음과 같이 설정해줍니다.

☐ Pass all job parameters as Project properties

Root Build script ?

Build File ?

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

☐ Force GRADLE_USER_HOME to use workspace ?

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

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

▼ 캡처

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

Execute shell ?

Command

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

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

New Key Store

Key store path: ers\leefo\Desktop\files\research\mobile\SignKey\IBKey.jks

Password: Confirm:

Key

Alias: IBKey

Password: Confirm:

Validity (years): 25

Certificate

First and Last Name: Forest Lee

Organizational Unit: Forest

Organization: Forest

City or Locality: Seoul

State or Province: Seoul

Country Code (XX): KR

OK Cancel

4. 다음과 같은 옵션으로 생성

