포팅 메뉴얼

▼ 목차

📌 개발 환경

Android

- IDE
  - Android Studio `2024.1.1`
- Android
  - AndroidX

        - android-gradlePlugin: `com.android.tools.build:gradle:8.5.0`
        - androidx-core-ktx: `androidx.core:core-ktx:1.13.1`
        - androidx-appcompat: `androidx.appcompat:appcompat:1.7.0`
        - material: `com.google.android.material:material:1.12.0`
        - androidx-activity: `androidx.activity:activity:1.9.1`
        - androidx-constraintlayout: `androidx.constraintlayout:constraintlayout:2.1.4`
        - lifecycle-runtime-ktx: `androidx.lifecycle:lifecycle-runtime-ktx:2.8.3`
        - lifecycle-extensions: `androidx.lifecycle:lifecycle-extensions:2.2.0`
    - 네이버 지도
        - map-sdk: `com.naver.maps:map-sdk:3.19.1`
    - 정적 코드 분석
        - detekt-gradle: `io.gitlab.arturbosch.detekt:detekt-gradle-plugin:1.23.5`
        - detekt-formatting: `io.gitlab.arturbosch.detekt:detekt-formatting:1.23.5`
    - 결제
        - bootpay: `io.github.bootpay:android:4.4.3`
    - Kotlin, Coroutine
        - kotlin-gradlePlugin: `org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.22`
        - coroutines-core: `org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.3`
        - coroutines-android: `org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3`
        - kotlinx-coroutines-test: `org.jetbrains.kotlinx:kotlinx-coroutines-test:1.7.1`
    - Android Jetpack
        - navigation-fragment-ktx: `androidx.navigation:navigation-fragment-ktx:2.7.7`
        - navigation-ui-ktx: `androidx.navigation:navigation-ui-ktx:2.7.7`
        - androidx-datastore-preferences-core: `androidx.datastore:datastore-preferences-core:1.1.1`
        - datastore-preferences: `androidx.datastore:datastore-preferences:1.1.1`

        - androidx-paging-runtime-ktx: `androidx.paging:paging-runtime-ktx:3.3.2`
    - Logging
        - timber: `com.jakewharton.timber:timber:5.0.1`
    - Sign
        - androidx-credentials: `androidx.credentials:credentials:1.2.2`
        - androidx-credentials-play-services-auth: `androidx.credentials:credentials-play-services-auth:1.2.2`
        - googleid: `com.google.android.libraries.identity.googleid:googleid:1.1.1`
        - naver-oauth: `com.navercorp.nid:oauth:5.10.0`
    - Serialization
        - kotlinx-serialization-core: `org.jetbrains.kotlinx:kotlinx-serialization-core:1.6.2`
        - kotlinx-serialization-json: `org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.2`
        - gson: `com.google.code.gson:gson:2.10.1`
    - Testing
        - robolectric: `org.robolectric:robolectric:4.10.3`
        - mockk: `io.mockk:mockk:1.13.3`
        - junit: `junit:junit:4.13.2`
        - androidx-junit: `androidx.test.ext:junit:1.2.1`
        - androidx-espresso-core: `androidx.test.espresso:espresso-core:3.6.1`
    - Hilt
        - hilt-android-compiler: `com.google.dagger:hilt-android-compiler:2.51.1`
        - hilt-android-testing: `com.google.dagger:hilt-android-testing:2.51.1`
        - hilt-android: `com.google.dagger:hilt-android:2.51.1`
        - hilt-compiler: `com.google.dagger:hilt-compiler:2.51.1`
        - androidx-hilt-navigation-fragment: `androidx.hilt:hilt-navigation-fragment:1.2.0`
        - androidx-hilt-compiler: `androidx.hilt:hilt-compiler:1.2.0`

- Network
  - okhttp: `com.squareup.okhttp3:okhttp:4.12.0`
  - okhttp-bom: `com.squareup.okhttp3:okhttp-bom:4.12.0`
  - okhttp-loggingInterceptor: `com.squareup.okhttp3:logging-interceptor:4.12.0`
  - retrofit: `com.squareup.retrofit2:retrofit:2.11.0`
  - retrofit-converter-gson: `com.squareup.retrofit2:converter-gson:2.11.0`
  - retrofit-converter-sclars: `com.squareup.retrofit2:converter-scalars:2.11.0`
  - retrofit-converter-kotlinxSerialization: `com.squareup.retrofit2:converter-kotlinx-serialization:2.11.0`
  - hivemq-mqtt-client: `com.hivemq:hivemq-mqtt-client:1.3.3`
- Media
  - glide: `com.github.bumptech.glide:glide:4.15.0`
  - lottie: `com.airbnb.android:lottie:6.5.0`
  - circleimageview: `de.hdodenhof:circleimageview:3.1.0`
  - material-calendarview: `com.github.prolificinteractive:material-calendarview:2.0.0`
  - threetenabp: `com.jakewharton.threetenabp:threetenabp:1.2.1`
- PlayService
  - play-services-basement: `com.google.android.gms:play-services-basement:18.4.0`
  - play-services-location: `com.google.android.gms:play-services-location:21.3.0`
- Permission
  - ted-permission-normal: `io.github.ParkSangGwon:tedpermission-normal:3.3.0`
  - ted-permission-coroutine: `io.github.ParkSangGwon:tedpermission-coroutine:3.3.0`
- QR
  - zxing-android-embedded: `com.journeyapps:zxing-android-embedded:4.3.0`
- Firebase

        - firebase-bom: `com.google.firebase:firebase-bom:33.3.0`
        - firebase-messaging-ktx: `com.google.firebase:firebase-messaging-ktx:24.0.1`
- CI/CD
    - Gitlab Pipeline Android CI/CD

BackEnd

- IDE
    - IntelliJ `2024.2.2`
- Java
    - Java OpenJDK `17.0.12`
- Spring
    - Spring Boot `3.3.3`
    - Spring Security `6.3.3`
    - Spring Batch `5.1.2`
    - Spring Data JPA `3.3.3`
    - JWT `0.11.5`
    - WebSocket `6.1.13`
    - MQTT `5.5.0`
    - Firebase-Admin `9.3.0`
    - Lombok `1.18.34`
- etc
    - Gradle `8.8.0`
    - Swagger `3.0.0`

ROS

- Autonomous Driving
    - Ubuntu `22.04`
    - ROS `noetic`

    - Docker
- Simulator
    - MORAI-Sim `22.R2.1`

AI

- Tensorflow `2.17.0`
- SciKit-Learn `1.5.2`
- SciPy `1.58.2`
- Docker
- Pyaudio - SpeechRecognizer `0.2.11`
- Pydub `0.25.1`
- Langchain `0.3.3`
- Langchain-OpenAI `0.2.2`
- Librosa `0.10.2.post1`
- boto3 `1.35.36`
- Flask `3.0.3`
- Firebase-Admin `6.5.0`

Database

- MySQL `8.0.38`
- Redis `7.4.0`
- AWS S3 Bucket Cloud
- Firebase

Infra

- AWS EC2 `20.04.6`
- Docker `27.1.1`
- Docker Compose `2.20.2`
- Nginx `1.18.0`
- Jenkins `2.475`

- GitLab Runner `16.0.2`

Collaboration

- Jira
- GitLab
- Notion
- Matter Most

📌 환경 변수 설정

Android

- CI환경 Variable
  - 📄 **google-services.json** → BASE64 인코딩 해둠

```
ewogICJwcm9qZWN0X2luZm8iOiB7CiAgICAicHJvamVjdF9udW1iZXIi
OiAiMjQ0MTA3MTMwODQxIiwKICAgICJwcm9qZWN0X2lkIjogImRydGFh
LWM3ODNiIiwKICAgICJzdG9yYWdlX2J1Y2tldCI6ICJkcnRhYS1jNzgz
Yi5hcHBzcG90LmNvbSIKICB9LAogICJjbGllbnQiOiBbCiAgICB7CiAg
ICAgICJjbGllbnRfaW5mbyI6IHsKICAgICAgICAibW9iaWxlc2RrX2Fw
cF9pZCI6ICIxOjI0NDEwNzEzMDg0MTphbmRyb2lkOjQzODExOGYwMDcy
NWJkZTg1Y2NiYzgiLAogICAgICAgICJhbmRyb2lkX2NsaWVudF9pbmZv
IjogewogICAgICAgICAgInBhY2thZ2VfbmFtZSI6ICJjb20uZHJ0YWEu
YW5kcm9pZCIKICAgICAgICB9CiAgICAgIH0sCiAgICAgICJvYXV0aF9j
bGllbnQiOiBbXSwKICAgICAgImFwaV9rZXkiOiBbCiAgICAgICAgewog
ICAgICAgICAgImN1cnJlbnRfa2V5IjogIkFJemFTeUMtOURHNEE3UGpL
amNnRWRqTjhLVlV2ODEzRkZHOVVyUSIKICAgICAgICB9CiAgICAgIF0s
CiAgICAgICJzZXJ2aWNlcyI6IHsKICAgICAgICAiYXBwaW52aXRlX3Nl
cnZpY2UiOiB7CiAgICAgICAgICAib3RoZXJfcGxhdGZvcm1fb2F1dGhf
Y2xpZW50IjogW10KICAgICAgICB9CiAgICAgIH0KICAgIH0KICBdLAog
ICJjb25maWd1cmF0aW9uX3ZlcnNpb24iOiAiMSIKfQ==
```

  - 📄 **local.properties**

```
NAVER_MAP_CLIENT_ID="jidsvf0ejh"
NAVER_MAP_CLIENT_ID_MANIFEST=jidsvf0ejh
NAVER_MAP_CLIENT_SECRET="BuPMiR6SS0Bx2SAsJw5TCK04nJcl6E8
RagC2tem9"
NAVER_CLIENT_ID="rfIAx_6VLWEJ9KSKCUyN"
NAVER_CLIENT_SECRET="Or97zpCtMF"
GOOGLE_LOGIN_CLIENT_ID="1045217773828-8t9u2vdqdg5djgb34o
2cdc653ht509v9.apps.googleusercontent.com"
TOUR_API_KEY="mjiIcWwH4AM3YLzONojZNLkMkwOqKA1B42EzgkSIw8
1H5TU06Gtbd0wA+ziWZTQzyqF9cFYAIRkNVB/12TwE7A=="
BASE_URL="https://j11d211.p.ssafy.io/"
MQTT_URL="j11d211.p.ssafy.io"
BOOTPAY_APP_ID="66e2dabea3175898bd6e4b23"
APP_KEY="kD0aI0A7ZU7KtrWxM2TdC4XXifP7wto9a38E6u4G"
```

BackEnd

- 📄 **application.yml**

```
spring:
  main:
    web-application-type: servlet
    allow-circular-references: true
    allow-bean-definition-overriding: true

  profiles:
    active: prod

  server:
    port: 8080  # 서버 포트 설정
    forward-headers-strategy: framework

  logging:
    level:
      root: DEBUG    # 전체 애플리케이션의 기본 로그 레벨 설정
      com.d211.drtaa: DEBUG   # 특정 패키지에 대한 로그 레벨 설정
      org.springframework.web: DEBUG   # 스프링 웹 관련 로그 레
      org.apache.coyote. http11: trace
```

```
servlet:
  multipart:
    enabled: true
    max-file-size: 10MB
    max-request-size: 10MB

# Batch
batch:
  jdbc:
    initialize-schema: always
  job:
    enabled: false

# JPA
jpa:
  hibernate:
    ddl-auto: none  # 스키마 자동 생성 전략 (update, create,
  show-sql: true  # SQL 쿼리 로깅 여부
  properties:
    hibernate:
      format_sql: true  # SQL 쿼리 포맷팅 여부

# MySQL
datasource:
  url: jdbc:mysql://j11d211.p.ssafy.io:3307/Drtaa?server
  username: {MySQL username}
  password: {MySQL password}
  hikari:
    maximum-pool-size: 10  # 커넥션 풀의 최대 크기 (필요에 따라

# Redis
data:
  redis:
    # host: localhost
    host: j11d211.p.ssafy.io
    port: 6380
    password: {Redis password}
```

```
  cache:
    type: redis

  # MQTT
  mqtt:
    broker-url: tcp://{MQTT broker-url}:1883
    client-id: {MQTT client-id}
    topic: pub/topic

# JWT
jwt:
  secret-key: {JWT secret-key}
  access-token:
    expiretime: 3600000 # 1시간
  refresh-token:
    expiretime: 2592000000 # 30일

# CORS
cors:
  allowed-origins: 'https://j11d211.p.ssafy.io'
  allowed-methods: GET,POST,PUT,DELETE,OPTIONS
  allowed-headers: '*'
  max-age: 3600

# S3
cloud:
  aws:
    s3:
      bucket: {S3 bucket}
    credentials:
      access-key: {S3 access-key}
      secret-key: {S3 secret-key}
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
```

```
# Firebase
firebase:
  path: drtaa-firebase-key.json

# WebSocket
websocket:
  server:
    url: ws://j11d211.p.ssafy.io:8765

# Naver(Blog Search)
naver:
  client-id: {Naver client-id}
  client-secret: {Naver client-secret}

# OpenWeather
weather:
  api-key: {OpenWeather api-key}
```

- 📄 **Dockerfile**

```
FROM openjdk:17-jdk-alpine

# 작업 디렉토리 설정, /app 디렉토리로 이동
WORKDIR /app

# 현재 디렉토리에 있는 모든 파일을 컨테이너의 /app 디렉토리로 복사
COPY . .

# 파일 목록 출력
RUN ls -al

# 현재 디렉토리 경로 출력
RUN pwd

# Gradle 빌드를 위한 준비 단계
# .gradle 디렉토리 삭제 (캐시된 파일을 제거하여 클린 빌드 보장)
```

```
RUN rm -rf .gradle

# gradlew에 실행 권한을 부여한 후 Gradle 빌드 실행 (clean 빌드)
RUN chmod +x ./gradlew && ./gradlew clean build

# 빌드 결과 확인 (JAR 파일이 있는지 확인)
RUN ls -al build/libs

# 생성된 JAR 파일을 app.jar로 이름 변경하여 컨테이너 내에 복사
RUN cp build/libs/Drtaa-0.0.1-SNAPSHOT.jar app.jar

# 컨테이너의 시간대를 서울(Asia/Seoul)로 설정
ENV TZ=Asia/Seoul

# 컨테이너 시작 시 실행할 명령어 설정 (JAR 파일 실행)
ENTRYPOINT ["java", "-jar", "./app.jar"]
```

ROS

- 📄 **Dockerfile.dev.morai**

```
# Step 1: ROS Noetic을 포함한 기본 이미지 설정
FROM osrf/ros:noetic-desktop-full-focal

# Step 2: 필수 패키지 및 빌드 도구 설치
RUN apt-get update && apt-get install -y \
    # ROS 패키지 (GUI 도구 제외)
    ros-noetic-image-transport \
    ros-noetic-compressed-image-transport \
    ros-noetic-rosbridge-server \
    ros-noetic-rosserial \
    \
    # 개발 도구 및 빌드 필수 패키지
    build-essential \
    cmake \
    python3-catkin-tools \
    git \
    vim \
```

```
    nano \
    \
    # Python 및 기타 유틸리티
    python3-opencv \
    python3-pip \
    python3-venv \
    \
    # 디버깅 및 성능 분석 도구
    gdb \
    valgrind \
    htop \
    \
    # 코드 품질 및 형상 관리 도구
    clang-format \
    cppcheck \
    git-lfs \
    \
    # 트리 구조 보기 도구
    tree \
    \
    # libpcap-dev 설치
    libpcap-dev \
    \
    # 클린업
    && rm -rf /var/lib/apt/lists/*

# 라이브러리 설치
RUN pip3 install --no-cache-dir pyproj && \
    pip3 install scikit-learn && \
    pip install "numpy>=1.19.5,<1.27.0"

# Step 3: catkin workspace 디렉토리 생성
RUN mkdir -p /root/catkin_ws/src

# Step 4: 작업 디렉토리 설정
WORKDIR /root/catkin_ws

# Step 5: 소스 파일 복사
```

```
COPY ./catkin_ws/src /root/catkin_ws/src

# Step 6: 기본 환경 설정
RUN echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc 
    echo "source /root/catkin_ws/devel/setup.bash" >> ~/.b
    echo 'alias cm="catkin_make"' >> ~/.bashrc && \
    echo 'alias scds="source /root/catkin_ws/devel/setup.b
    echo 'alias rosbridge_server="roslaunch rosbridge_serv

# Step 7: catkin workspace 빌드
WORKDIR /root/catkin_ws
RUN /bin/bash -c "source /opt/ros/noetic/setup.bash && cat

# 기본 개발 환경 이미지를 위한 CMD 설정 (roslaunch 실행)
# CMD ["roslaunch", "rosbridge_server", "rosbridge_websock
CMD ["bash"]
```

📌 배포 환경 설정

0. 초기 세팅

1. EC2 초기 설정

    a. 접속

    ```
    # sudo ssh -i [pem키 위치] [접속 계정]@[접속할 도메인]
    sudo ssh -i J11D211.pem ubuntu@j11d211.p.ssafy.io
    ```

    b. 업데이트

    ```
    # 설치 가능한 패키지의 최신 목록을 업데이트
    sudo apt update
    # 시스템에 설치된 모든 패키지를 최신 버전으로 업그레이드
    sudo apt upgrade -y
    # 필수적인 개발 도구들을 설치
    sudo apt install -y build-essential
    ```

```
# 패키지 관리자 업데이트
sudo apt-get update
sudo apt-get upgrade
```

c. 시간 설정

```
# 한국 시간 설정
sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localti
# 시간 확인
date
```

d. ufw 포트 설정

```
# ufw 상태 확인
sudo ufw status
# 사용할 포트 허용하기
sudo ufw allow 포트번호
# 등록한 포트 조회하기
sudo ufw show added
# ufw 활성화 하기
sudo ufw enable
# ufw 상태 및 등록된 rule 확인하기
sudo ufw status numbered
# ufw 구동된 상태에서 포트 추가하기
sudo ufw allow 포트번호
# 포트 정상 등록 확인
sudo ufw status numbered
# 등록한 포트 삭제
sudo ufw status numbered
```

2. Docker 설치

a. 설치

```
sudo apt-get update
sudo apt-get install \
   ca-certificates \
   curl \
```

```
    gnupg \
    lsb-release
```

b. Docker의 GPG키 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
```

c. 저장소 설정

```
echo \
 "deb [arch=$(dpkg --print-architecture) signed-by=/etc
 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources
```

d. Docker Engine 설치

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd
```

e. 확인

```
sudo docker run hello-world
```

3. Docker compose 설치

1. Docker

- BackEnd
  1. MySQL Docker image pull

```
sudo docker pull mysql:8.0.38
```

  2. Redis Docker image pull

```
sudo docker volume create redis
sudo docker pull redis
```

```
# /var/lib/volumes/redis/redis.conf 파일 생성
cd /var/lib/docker/volumes/redis/
sudo vi redis.conf
```

- 📄 **redis.conf**

```
bind 0.0.0.0
port 6379
appendonly yes
appendfilename "appendonly.aof"
requirepass ssafygumid211drtaateammember
```

```
# Docker image 확인
docker images
```

3. docker-compose.yml 파일 생성

```
touch docker-compose.yml
vi docker-compose.yml
```

- 📄 **docker-compose.yml**

```
services:
  jenkins:
    image: jenkins/jenkins:jdk17
    container_name: jenkins-container
    user: root
    ports:
      - "8080:8080"
    volumes:
      - /home/ubuntu/jenkins-backup:/var/jenkins_hom
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/bin/docker
    environment:
      TZ: "Asia/Seoul"
  mysql:
    image: mysql:8.0.38
```

```
    container_name: mysql-container
    ports:
      - "3306:3306"
    volumes:
      - /mysql-volume:/var/lib/mysql
    environment:
      MYSQL_DATABASE: Drtaa
      MYSQL_ROOT_PASSWORD: ssafygumid211drtaaleejung
      MYSQL_USER: d211
      MYSQL_PASSWORD: ssafygumid211drtaateammember
      TZ: "Asia/Seoul"
  redis:
    image: redis:latest
    container_name: redis-container
    ports:
      - "6379:6379"
    volumes:
      - /var/lib/docker/volumes/redis/_data:/data
      - /var/lib/docker/volumes/redis/redis.conf:/us
    command: ["redis-server", " /usr/local/etc/redis
```

4. docker-compose 실행

```
sudo docker-compose up -d
# 확인
docker ps -a
```

- ROS

2. Nginx

- EC2에 Nginx 설치하고 Spring Boot 적용

```
sudo apt-get update
```

1. Nginx 설치

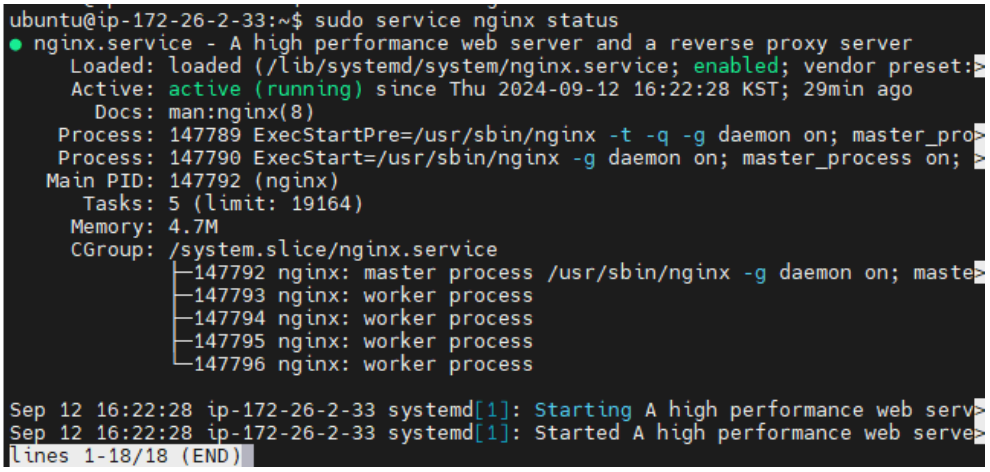
```
sudo apt install nginx
```

2. Nginx 실행

```
sudo service nginx start
```

4. Nginx가 잘 실행되는지 확인!

```
sudo service nginx status
```



Nginx 설정 파일

`proxy_params` 파일은 클라이언트와 Nginx 프록시 간 설정파일

/etc/nginx/proxy_params

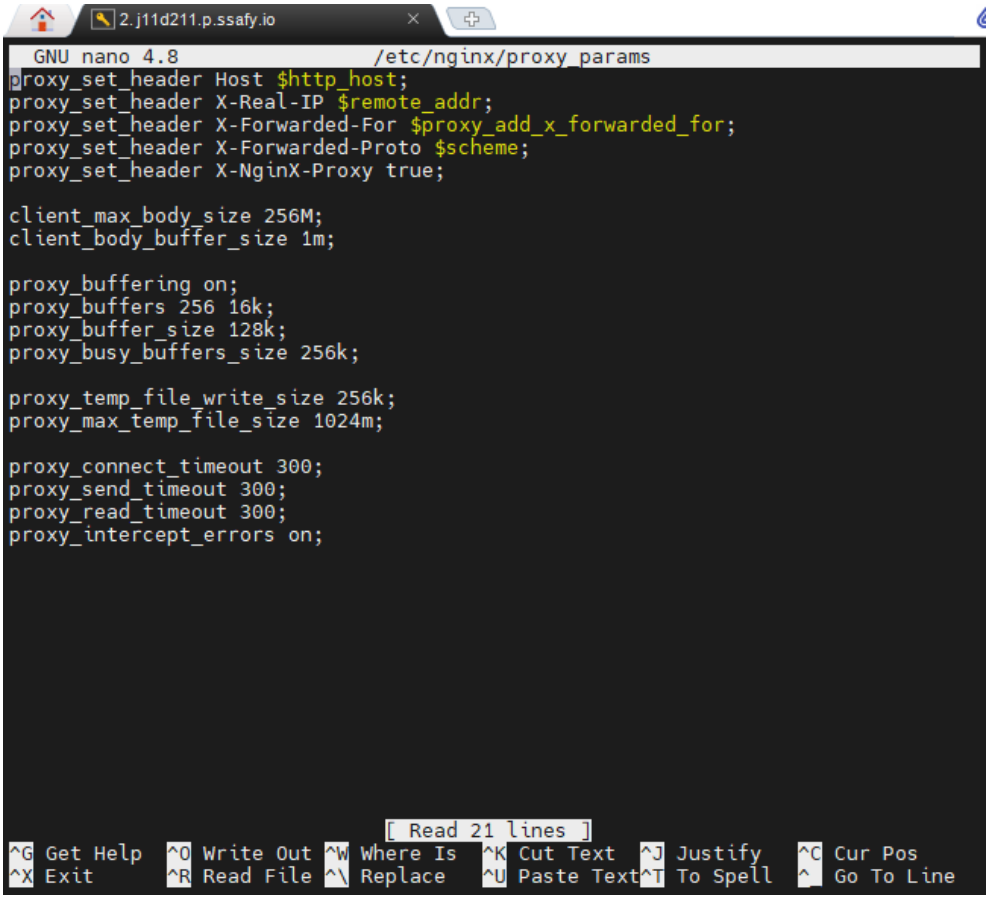
```
//원래의 호스트 헤더 값을 전달합니다.
proxy_set_header Host $http_host;

//실제 클라이언트의 IP 주소를 전달합니다.
proxy_set_header X-Real-IP $remote_addr;

//클라이언트의 IP와 프록시 서버의 IP를 전달합니다.
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_fo

//클라이언트와의 연결 프로토콜(예: http 또는 https)을 전달합니다.
proxy_set_header X-Forwarded-Proto $scheme;
```

1. proxy_params 파일 수정

```
sudo nano /etc/nginx/proxy_params
```

2. 코드 적용

```


proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_fo
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-NginX-Proxy true;

client_max_body_size 256M;
client_body_buffer_size 1m;

proxy_buffering on;
proxy_buffers 256 16k;
proxy_buffer_size 128k;
proxy_busy_buffers_size 256k;

proxy_temp_file_write_size 256k;
proxy_max_temp_file_size 1024m;

proxy_connect_timeout 300;
proxy_send_timeout 300;
proxy_read_timeout 300;
proxy_intercept_errors on;
```



sites-available

사이트의 server block이나 virtual host설정을 하는 디렉토리.

각 호스트는 도메인, 서브도메인 또는 IP 주소 별로 웹 서버의 동작을 정의합니다.

1. 명령어를 입력해서 server block 생성

```
sudo nano /etc/nginx/sites-available/{domain}
```

2. 코드 입력

```
server { # server 블록
    listen 80;

    server_name {domain} www.{domain};

    access_log /var/log/nginx/proxy/access.log;
    error_log /var/log/nginx/proxy/error.log;

    location / { # location 블록
```

```
        include /etc/nginx/proxy_params;
        proxy_pass http://{퍼블릭IP주소}:8088;    # reverse
    }
}
```

파일 활성화

1. 서버블록을 생성했으니 활성화

```
sudo ln -s /etc/nginx/sites-available/{domain} /etc/nginx/
```

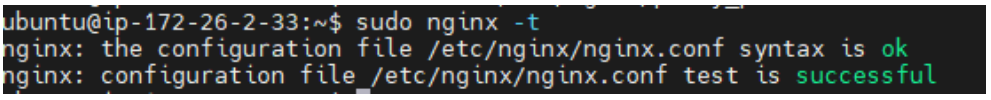
2. 기본 구성 파일 삭제

default 파일 삭제

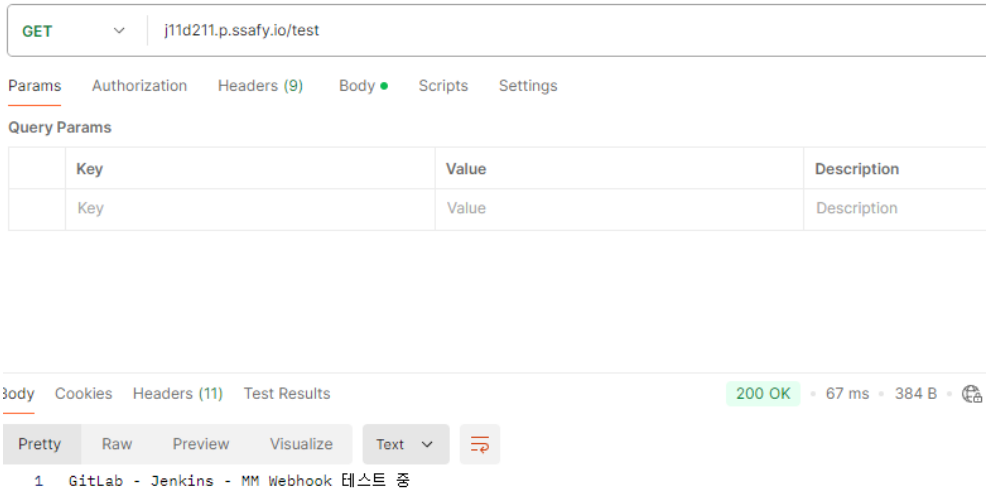
```
sudo rm  /etc/nginx/sites-available/default
sudo rm  /etc/nginx/sites-enabled/default
```

3. 연결 테스트 해보고 문제가 없으면 재시작

```
sudo nginx -t
```




```
sudo service nginx reload
```



- SSL 인증서를 통해 Https 적용

1. 스냅 패키지 설치

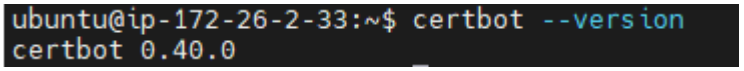
```
sudo snap install core
sudo snap refresh core
```

2. certbot 패키지 설치

```
sudo snap install --classic certbot
sudo apt install python3-certbot-nginx
```

3. 명령어 편의를 위해 심볼릭 링크 생성

```
sudo ln -s /snap/bin/cerbot/ /usr/bin/cerbot
//버전 확인
certbot --version
```



4. 인증서 발급

```
sudo certbot --nginx -d [example.com]
```

5. sites-available 수정

```
sudo nano /etc/nginx/sites-available/{domain}
```

```
server {
    listen 80;
    server_name j11d211.p.ssafy.io;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name j11d211.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/p.ssafy.io/fullc
    ssl_certificate_key /etc/letsencrypt/live/p.ssafy.io/p

    access_log /var/log/nginx/proxy/access.log;
    error_log /var/log/nginx/proxy/error.log;

    location / {
        include /etc/nginx/proxy_params;
        proxy_pass http://localhost:8080;
    }
}
```

첫 번째 서버 블록

- 80포트(HTTP)로 들어오는 모든 트래픽을 HTTPS로 리다이렉트

두 번째 서버 블록

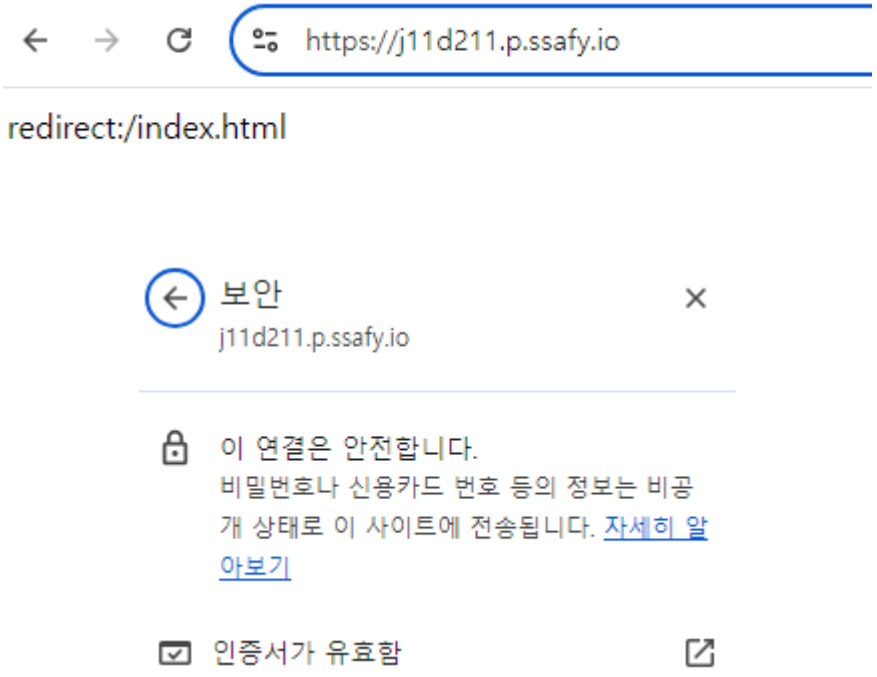
- 443포트(HTTPS)에서 SSL을 사용하여 리스닝합니다.
- SSL 인증서와 키 파일 경로 지정

6. Nginx 설정 테스트

```
sudo nginx -t
```

7. 오류가 없다면 Nginx 재시작

```
sudo systemctl restart nginx
```



3. Jenkins

1. 마운트 할 볼륨 디렉토리 생성

```
cd /home/ubuntu && mkdir jenkins-data
```

2. jenkins container 생성 및 구동

```
sudo docker run -d -p 8080:8080 -v /home/ubuntu/jenkins-da
```

3. 구동 상태 확인

```
docker ps -a
docker logs jenkins-container
```

4. Jenkins 환경 설정 변경
    a. update center에 필요한 CA 파일 다운로드

```
sudo mkdir update-center-rootCAs
sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-up
```

b. jenkins의 default설정에서 특정 미러 사이트로 대체

```
sudo sed -i 's#https://updates.jenkins.io/update-center
```

c. jenkins서비스 재구동

```
sudo docker restart jenkins-container
```

5. 보안 설정 확인

```
# true인지 확인
vi /home/ubuntu/jenkins-data/config.xml
```

6. Jenkins 접속

```
http://j11d211.p.ssafy.io:8080
```

- 초기 비밀번호 확인

```
# jenkins 컨테이너에 접속
sudo docker exec -it jenkins bash
# 초기 관리자 키 확인
cat /var/jenkins_home/secrets/initialAdminPassword
```

7. 기본 플러그인 설치
8. Jenkins 설정

- 📄 **PipeLine Script**

```
pipeline {
    agent any

    stages {
        stage('Git Clone') {
```

```
            steps {
                git branch: 'develop',
                credentialsId: {GitLab의 ID},
                url: "https://lab.ssafy.com/s11-mobilit
            }
            post {
                failure {
                  echo 'Git Repository clone failure !'
                }
                success {
                  echo 'Git Repository clone success !'
                }
            }
        }

        stage('Download application.yml') {
            steps {
                withCredentials([file(credentialsId: 'a
                    script {
                        // application.yml 파일이 있는지 확
                        def fileExists = sh(script: 'te

                        if (fileExists == 'true') {
                            sh 'rm /var/jenkins_home/wo
                        }

                        sh 'cp $ymlFile /var/jenkins_ho
                    }
                }
            }
        }

        stage('Download drtaa-firebase-key.json') {
            steps {
                withCredentials([file(credentialsId: 'f
                    script {
                         // drtaa-firebase-key.json 파일
                        def fileExists = sh(script: 'te
```

```
                    if (fileExists == 'true') {
                        sh 'rm /var/jenkins_home/wo
                    }
                    sh 'cp $jsonFile /var/jenkins_h
                }
            }
        }

        stage('Docker_Build Backend') {
            steps {
                dir('/var/jenkins_home/workspace/D211/D
                    script {
                        sh 'ls -al'
                        sh 'whoami'
                        sh 'groups jenkins'
                        sh 'docker ps -a'

                        // 기존 컨테이너가 있는지 확인
                        def containerExists = sh(script

                        if (containerExists == 'my-cont
                            sh 'docker rm -f my-contain
                        }

                        sh 'docker build -t tem .'
                        sh 'docker run -d --name my-con
                    }
                }
            }
            post {
                success {
                    script {
                        def Author_ID = sh(script: "git
                        def Author_Name = sh(script: "g
                        mattermostSend(color: 'good',
                            message: "빌드 성공: ${env.JO
```

```
                            endpoint: {MM의 채널 URL}
                        )
                    }
                }
                failure {
                    script {
                        def Author_ID = sh(script: "git
                        def Author_Name = sh(script: "g
                        mattermostSend(color: 'danger',
                            message: "빌드 실패: ${env.JO
                            endpoint: {MM의 채널 URL}
                        )
                    }
                }
            }
        }
    }
}
```

1. **Git Clone**

   GitLab의 develop branch를 clone합니다.

2. **Download application.yml**
   Jenkins의 credentials에서
   `application_yml` 자격 정보를 가져옵니다.

3. **Download drtaa-firebase-key.json**

   Jenkins의 credentials에서 `firebase_key` 자격 정보를 가져옵니다.

4. **Docker_Build Backend**

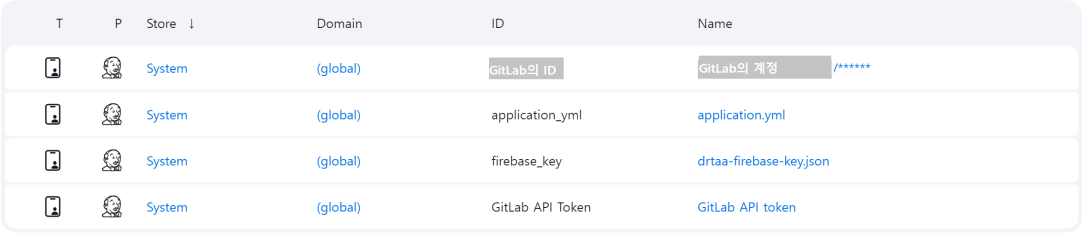
   Jenkins가 작업 중인 백엔드 디렉터리로 이동한 후 환경을 확인합니다.

   기존 실행 중인 컨테이너가 있는지 확인하고 새로운 이미지를 빌드해 `my-container`라는 이름의 컨테이너를 백그라운드에서 실행합니다.

5. **Notification to MM**

   빌드 완료 후 Mattermost로 빌드 성공/실패 알림을 보냅니다.

- Credential



T	P	Store ↓	Domain	ID	Name
		System	(global)	GitLab의 ID	GitLab의 계정 /******
		System	(global)	application_yml	application.yml
		System	(global)	firebase_key	drtaa-firebase-key.json
		System	(global)	GitLab API Token	GitLab API token

- Plugin 추가 설치
  - Pipeline Graph View Plugin
  - GitLab Plugin
  - Mattermost Notification Plugin

4. GitLab Runner

- Android
  1. docker-compose.yml 생성
     a. 📄 **docker-compose.yml**

```
version: "3"

services:
  gitlab-runner:
    container_name: gitlab-runner
    image: 'gitlab/gitlab-runner:latest'
    restart: always
    volumes:
    - './config:/etc/gitlab-runner'
    - '/var/run/docker.sock:/var/run/docker.sock'
```

     b. gitlab runner image pull
     c. docker volume mount
  2. docker-compose up -d (데몬으로 실행해야 runner 받아서 돌아감..)
  3. docker compose exec gitlab-runner bash
     a. gitlab-runner regist → url, token

4. .gitlab_ci.yml 작성 후 pipeline 등록 - 환경변수들은 따로 gitlab variable에 등록해둬야한다.
    a. 📄 **.gitlab_ci.yml**

```
image: eclipse-temurin:17-jdk-jammy

variables:
  ANDROID_COMPILE_SDK: "34"
  ANDROID_BUILD_TOOLS: "34.0.0"
  ANDROID_SDK_TOOLS: "9477386"

before_script:
  - cd Drtaa-Android
  - apt-get --quiet update --yes
  - apt-get --quiet install --yes wget unzip
  - export ANDROID_HOME="${PWD}/android-sdk-root"
  - install -d $ANDROID_HOME
  - wget --no-verbose --output-document=$ANDROID_HOM
  - unzip -q -d "$ANDROID_HOME/cmdline-tools" "$ANDR
  - mv -T "$ANDROID_HOME/cmdline-tools/cmdline-tools
  - export PATH=$PATH:$ANDROID_HOME/cmdline-tools/la
  - sdkmanager --version
  - yes | sdkmanager --licenses > /dev/null || true
  - sdkmanager "platforms;android-${ANDROID_COMPILE_
  - sdkmanager "platform-tools"
  - sdkmanager "build-tools;${ANDROID_BUILD_TOOLS}"
  - chmod +x ./gradlew
  - echo "$GOOGLE_SERVICES" | base64 -d > app/google
  - echo "sdk.dir=$ANDROID_SDK_ROOT" > local.propert
  - echo "NAVER_MAP_CLIENT_ID=\"$NAVER_MAP_CLIENT_ID
  - echo "NAVER_MAP_CLIENT_ID_MANIFEST=$NAVER_MAP_CL
  - echo "NAVER_MAP_CLIENT_SECRET=\"$NAVER_MAP_CLIEN
  - echo "NAVER_CLIENT_ID=\"$NAVER_CLIENT_ID\"" >> l
  - echo "NAVER_CLIENT_SECRET=\"$NAVER_CLIENT_SECRET
  - echo "GOOGLE_LOGIN_CLIENT_ID=\"$GOOGLE_LOGIN_CLI
  - echo "TOUR_API_KEY=\"$TOUR_API_KEY\"" >> local.p
  - echo "BASE_URL=\"$BASE_URL\"" >> local.propertie
  - echo "MQTT_URL=\"$MQTT_URL\"" >> local.propertie
```

```
    - echo "BOOTPAY_APP_ID=\"$BOOTPAY_APP_ID\"" >> loc
    - echo "APP_KEY=\"$APP_KEY\"" >> local.properties

detektAnalysis:
  stage: test
  script:
    - ./gradlew detekt
  artifacts:
    paths:
      - Drtaa-Android/build/reports/detekt/
    expire_in: 8h
  allow_failure: false
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_even
      changes:
        - Drtaa-Android/**/*

assembleDebug:
  interruptible: true
  stage: build
  script:
    - ./gradlew assembleDebug
  artifacts:
    paths:
      - Drtaa-Android/app/build/outputs/apk/debug/ap
      - Drtaa-Android/app/build/outputs/logs/
    expire_in: 8h
  rules:
    - if: $CI_COMMIT_BRANCH == "develop" && $CI_PIPE
      changes:
        - Drtaa-Android/**/*
```

5. unstaged, staged tag 결정해서 각 stage build 실행

📌 DB 덤프

Drtaa.sql

📌 프로젝트에 사용된 외부 서비스

외부 저장소

- S3

  모바일 애플리케이션 관련 사진 데이터 저장을 위해 사용
- Firebase

  FCM 알림을 위해 사용

외부 API

- 결제
  - BootPay

    https://www.bootpay.co.kr/
- 소셜 로그인
  - Naver

    https://developers.naver.com/docs/login/overview/overview.md
  - Google

    https://cloud.google.com/identity-platform/docs/web/google?hl=ko
- 관광 API

  https://api.visitkorea.or.kr/#/
- Naver 지도 API

  https://www.ncloud.com/product/applicationService/maps
- Naver 블로그 검색 API

  https://developers.naver.com/docs/serviceapi/search/blog/blog.md#블로그

- Naver 이미지 검색 API

  https://developers.naver.com/docs/serviceapi/search/image/image.md#이미지

- OpenWeather API

  https://openweathermap.org/api