포팅메뉴얼

서버 설정

1. Nginx 설치

```
sudo apt-get install nginx
sudo nginx -v
sudo systemctl stop nginx
```

2. 서버 Nginx 설정

```
server {
        location /{
                proxy_pass http://127.0.0.1:3000;
        }

        location /api {
                proxy_pass http://127.0.0.1:8080;
                proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $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;

    proxy_buffer_size          128k;
          proxy_buffers              4 256k;
          proxy_busy_buffers_size    256k;
        }

        location ~ ^/(swagger|webjars|configuration|swagger-resources|v2|csrf) {
               proxy_pass http://127.0.0.1:8080;
               proxy_set_header Host $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;
                }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/{YOUR_DOMAIN}/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/{YOUR_DOMAIN}/privkey.pem; # managed by Certbot
    # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = {YOUR_DOMAIN}) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
        server_name {YOUR_DOMAIN};
    return 404; # managed by Certbot
}
```

2. ssl 인증

```
sudo apt-get install letsencrypt
sudo letsencrypt certonly --standalone -d [도메인]
```

3. Nginx에 ssl 설정

```
cd /etc/letsencrypt/live/[도메인]
cd /etc/nginx/sites-available
sudo vim [파일명].conf
sudo ln -s /etc/nginx/sites-available/[파일명] /etc/nginx/sites-enabled/[파일명]
sudo nginx -t
```

```
sudo systemctl restart nginx
sudo systemctl status nginx
```

Openvidu

1. 설치

```
sudo su
cd /opt
curl <https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh> | bash
cd openvidu
./openvidu start  //시작
```

2. 설정 파일

/opt/openvidu/ 하위에 .env 파일 생성

```
# OpenVidu configuration
# ----------------------
# 도메인 또는 퍼블릭IP 주소
DOMAIN_OR_PUBLIC_IP={YOUR_DOMAIN}

# 오픈비두 서버와 통신을 위한 시크릿
OPENVIDU_SECRET={YOUR_SECRET}

# Certificate type
CERTIFICATE_TYPE=letsencrypt

# 인증서 타입이 letsencrypt일 경우 이메일 설정
LETSENCRYPT_EMAIL={YOUR_EMAIL}

# HTTP port
HTTP_PORT=8442

# HTTPS port(해당 포트를 통해 오픈비두 서버와 연결)
HTTPS_PORT=8443
```

프론트엔드 설정

1. 프로젝트 루트에 도커 파일 작성 (파일이름 : Dockerfile)

```
FROM nginx:stable-alpine

WORKDIR /app

RUN mkdir ./build

ADD ./build ./build

RUN rm /etc/nginx/conf.d/default.conf

COPY ./nginx.conf /etc/nginx/conf.d

EXPOSE 3000

CMD ["nginx", "-g", "daemon off;"]
```

2. 프로젝트 루트에 엔진엑스 설정 파일 작성 (파일이름 : nginx.conf)

```
server {
    listen 3000;
    location / {
        root /app/build;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

3. .env 파일 작성 (내용은 서버의 도메인에 따라 달라짐)

```
REACT_APP_IP_ADDRESS= 여기에 url 입력 (ex: https://test.com:8080)
```

4. 명령어를 통해 빌드 후 배포
   a. 프로젝트 빌드

      npm run build

   b. 도커 이미지 생성

      docker build -t {태그 이름} .

   c. 도커 이미지 도커 허브로 푸쉬(서버에서 도커 이미지를 생성한 경우 생략 가능)

      docker push {도커 허브 계정/도커 허브 레포지토리 : 태그}

   d. 서버에서 도커 이미지 받기(c 단계에서 입력한 계정의 레포지토리에서 받아옴)

      docker pull {도커 허브 계정/도커 허브 레포지토리 : 태그}

   e. 서버에서 도커 이미지 실행 (3000포트로 매핑)

      docker run —rm -p 3000:3000 실행할 이미지 이름:태그

백엔드 설정

1. src/main/resource 하위에 서버 설정 파일 작성 (파일 이름 : application.yml)

   DB 계정이나 서버 URL은 환경에 따라 변경 필요

   도커에서 prod 버전으로 빌드하므로 서버 배포 시 prod 버전 필수

```
# default
server:
  servlet:
    context-path: /api/v1
  port: 8080
  ssl:
    enabled: false

spring:

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/arc?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8&allowPublicKeyRetrieval=tr
    username: {YOUR_ACCOUNT}
    password: {YOUR_PASSWORD}
    initialization-mode: always

  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    open-in-view: true
    hibernate:
      ddl-auto: create
      naming:
        physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
      use-new-id-generator-mappings: false
    defer-datasource-initialization: true
    properties:
      hibernate:
        format_sql: true
        show-sql: true

  sql:
    init:
      mode: always
      data-locations: classpath:data.sql
```

```
logging:
  level:
    org.hibernate.SQL: debug

OPENVIDU_URL: http://localhost:4443/
OPENVIDU_SECRET: {YOUR_OPENVIDU_SECRET}

allowed-origins: "http://localhost:3000"

--- # prod

server:
  servlet:
    context-path: /api/v1
  port: 8080
  ssl:
    enabled: false

spring:
  config:
    activate:
      on-profile: prod

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://{YOUR_SERVER_DOMAIN}:3306/arc?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: {YOUR_ACCOUNT}
    password: {YOUR_PASSWORD}
    data: classpath:data.sql
    initialization-mode: always

  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    open-in-view: true
    hibernate:
      ddl-auto: create
      naming:
        physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
      use-new-id-generator-mappings: false
    defer-datasource-initialization: true
    properties:
      hibernate:
        format_sql: false
        show-sql: false
      dialect: org.hibernate.dialect.MySQL5InnoDBDialect

  sql:
    init:
      mode: always

logging:
  level:
    org.hibernate.SQL: debug

OPENVIDU_URL: https://{YOUR_SERVER_DOMAIN}:8443/
OPENVIDU_SECRET: YOUR_OPENVIDE_SECRET
```

2. DB 초기 데이터 입력 (파일 이름 : data.sql)

   src/main/resources 하위 data.sql에 아래 내용 작성

```
INSERT INTO arc.gamemap (id, title, start_location_x, start_location_y)
VALUES
  (1, "basic map", 0.0, 0.0)
;

INSERT INTO arc.mission (id, title)
VALUES
  (1, "공터 정화조 청소"),
  (2, "창고 비품 보충"),
  (3, "연구실 시약 온도 조절"),
  (4, "식당 음식 간보기"),
  (5, "의무실에서 약품 꺼내기"),
  (6, "라운지 벨튀범 쫓기"),
  (7, "휴게실에서 메세지 응답"),
  (8, "연구실에서 뇌파 분석"),
  (9, "라운지에서 망보기"),
  (10, "연결통로 배전반 복구")
;

INSERT INTO arc.gamemap_mission (gamemap_id, mission_id, x, y)
VALUES
  (1, 1, 17.5, -1.5),
```

```
  (1, 2, 14, 16.5),
  (1, 3, -18.5, -3.5),
  (1, 4, -17, -9.5),
  (1, 5, -19, 17.5),
  (1, 6, 4.5, -12.5),
  (1, 7, 16.5, -13.5),
  (1, 8, -13.5, 6),
  (1, 9, 1, -20)
;
```

2. 프로젝트 루트에 도커 파일 작성 (파일이름 : Dockerfile)

```
FROM openjdk:11-jdk-slim

ARG JAR_FILE=build/libs/*.jar

COPY ${JAR_FILE} app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod", "/app.jar"]
```

3. 명령어를 통해 빌드 후 배포
   a. 프로젝트 빌드(윈도우 기준)

      ./gradlew clean build

   b. 도커 이미지 생성

      docker build -t {태그 이름} .

   c. 도커 이미지 도커 허브로 푸쉬(서버에서 도커 이미지를 생성한 경우 생략 가능)

      docker push {도커 허브 계정/도커 허브 레포지토리 : 태그}

   d. 서버에서 도커 이미지 받기(c 단계에서 입력한 계정의 레포지토리에서 받아옴)

      docker pull {도커 허브 계정/도커 허브 레포지토리 : 태그}

   e. 서버에서 도커 이미지 실행 (8080포트로 매핑)

      docker run —rm -p 8080:8080 실행할 이미지 이름:태그