포팅 메뉴얼

순서

1. 포트 설정

서비스	포트	설명
SSH	22	원격 접속
Nginx	80, 443	웹 서버 HTTP, HTTPS
MySQL	3306	데이터베이스 접근
프론트엔드	3000	React 개발 서버
백엔드	8080:8081	Spring Boot 서버 (Blue-Green 배포)
Jenkins	8080	CI/CD 관리
Redis	6379	캐시 서버
OpenVidu	8880, 4443, 3478, 5443, 5349	WebRTC 미디어 서버

2. 네트워크 설정

오픈비두를 제외한 나머지 컨테이너들의 모든 네트워크는 동일해야함.

```
docker network create app-network
```

3. Mysql 설정

포트 3306 으로 설정

한국 서버로 설정!

MySQL 설치

```
docker run -e MYSQL_ROOT_PASSWORD=root \
  -p 3306:3306 \
  --name mysql \
  --network app-network \ # 모든 서비스가 같은 net에 있어야 함.
  -v /home/ubuntu/docker-mysql/mindon_data:/var/lib/mysql \ # 필수 남아있어야 함
  -d mysql:8.0.28
```

아래걸로 실행

```
docker run -e MYSQL_ROOT_PASSWORD=root \
  -e TZ=Asia/Seoul \
  -p 3306:3306 \
  --name mysql \
  --network app-network \
  -v /home/ubuntu/docker-mysql/mindon_data:/var/lib/mysql \  # 볼륨 설정
  -v /etc/localtime:/etc/localtime:ro \
  -v /etc/timezone:/etc/timezone:ro \
  --restart=always \
  -d mysql:8.0.28
```

MySQL 데이터베이스 설정

```
docker exec -it mysql mysql -uroot -proot
CREATE DATABASE on_board;
SHOW DATABASES;
docker restart mysql
```

data 넣기

1. workbench 환경에 data 설정
2. 직접 삽입하는 방법

```
scp -i "C:\경로\to\your-key.pem" init.sql ubuntu@YOUR_EC2_IP:/home/ubuntu/init.sql

scp -i "C:\Users\SSAFY\.ssh\I12B103T.pem" init.sql ubuntu@i12b103.p.ssafy.io:/home/ubuntu/init.sql
```

- ec2 환경에서 sql 실행

```
docker exec -i mysql mysql -uroot -proot < /home/ubuntu/init.sql
```

- 확인하기

```
docker exec -it mysql mysql -uroot -proot
SHOW DATABASES;
USE on_board;
SHOW TABLES;
```

4. Redis 설정

```
docker pull redis
docker run -d --name redis --network app-network redis
```

5. 오픈비두 설정

openvidu 설치 시 주의사항

https://velog.io/@ohjinseo/ReactJS-EC2-OpenVidu-%ED%99%98%EA%B2%BD%EC%97%90%EC%84%9C-%ED%99%94%EC%83%81%ED%9A%8C%EC%9D%98-%EA%B5%AC%ED%98%84%ED%95%B4%EB%B3%B4%EA%B8%B0

OpenVidu 설치전 주의사항!

다음으로 OpenVidu 설치 전 반드시 아래 포트들이 열려 있어야 정상적으로 서버가 작동한다.
EC2 보안 그룹 - 인바운드 규칙에서 아래 포트들에 대해서는 모든 IP를 허용해준다.

- **22 TCP**: to connect using SSH to admin OpenVidu.
- **80 TCP**: if you select Let's Encrypt to generate an SSL certificate this port is used by the generation process.
- **443 TCP**: OpenVidu server and application are published by default in standard https port.
- **3478 TCP+UDP**: used by STUN/TURN server to resolve clients IPs.
- **40000 - 57000 TCP+UDP**: used by Kurento Media Server to establish media connections.
- **57001 - 65535 TCP+UDP**: used by TURN server to establish relayed media connections.

그리고 또 하나 주의할 점은 아래 밑줄 그인 포트들에 대해서는 OpenVidu를 구성하는 컨테이너들이 할당받아야 한다. 즉, 다른 서비스들이 해당 포트 중 1개라도 할당받고 있다면 OpenVidu 서비스가 정상적으로 실행되지 않는다.

특히, 많이 사용되는 Nginx(80번), Redis(6379번) 등을 이미 설치하였다면 PID를 찾아 미리 kill -9 시켜주어야 한다.

- **Free ports inside the server**: OpenVidu platform services will need the following ports to be available in the machine: 80, 443, 3478, 5442, 5443, 6379 and 8888. If some of these ports is used by any process, OpenVidu platform won't work correctly. It is a typical error to have an NGINX process in the system before installing OpenVidu. Please uninstall it.

nginx와 오픈비두가 같은 80 포트를 사용하기 때문에 nginx보다 openvidu 설치를 먼저 진행하는 것이 좋다.

8880, 4443 포트 사용

혹 5443 사용 시 포트가 겹친다는 오류가 발생한다면 4443으로 변경 시 적용 가능

오픈비두 설치 및 실행

```
sudo mkdir -p /opt/openvidu
cd /opt/openvidu
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | sudo bash
```

오픈비두 /opt/openvidu/.env 파일 설정

```
sudo vim .env
```

```
DOMAIN_OR_PUBLIC_IP=i12b103.p.ssafy.io
OPENVIDU_SECRET={사용자 설정}# 원하는 비밀키로 변경
CERTIFICATE_TYPE=letsencrypt
LETSENCRYPT_EMAIL=your@email.com  # 이메일 주소 입력
HTTP_PORT=8880
HTTPS_PORT=4443
LETSENCRYPT_CERT_PATH=/etc/letsencrypt/live/your-domain.com/fullchain.pem
LETSENCRYPT_CERT_KEY_PATH=/etc/letsencrypt/live/your-domain.com/privkey.pem

recording 관련 부분 true 로 변경
```

오픈비두 /opt/openvidu/docker-compose.yml 파일 설정

초반에 실행 시 /opt/openvidu/.env 파일을 읽지 못하기 때문에 수동으로 설정

```
services:

  openvidu-server:
    image: openvidu/openvidu-server:2.31.0
    restart: on-failure
    network_mode: host
    entrypoint: ['/usr/local/bin/entrypoint.sh']
    volumes:
      - ./coturn:/run/secrets/coturn
      - /var/run/docker.sock:/var/run/docker.sock
      - ${OPENVIDU_RECORDING_PATH}:${OPENVIDU_RECORDING_PATH}
      - ${OPENVIDU_RECORDING_CUSTOM_LAYOUT}:${OPENVIDU_RECORDING_CUSTOM_LAYOUT}
      - ${OPENVIDU_CDR_PATH}:${OPENVIDU_CDR_PATH}
      - /opt/openvidu/.env:/opt/openvidu/.env  # 이부분 추가 제대로 값을 읽어오지 못함
    env_file:
      - .env
    environment:
      - SERVER_SSL_ENABLED=false
      - SERVER_PORT=5443
      - KMS_URIS=["ws://localhost:8888/kurento"]
      - COTURN_IP=${COTURN_IP:-auto-ipv4}
      - COTURN_PORT=${COTURN_PORT:-3478}
    logging:
      options:
        max-size: "${DOCKER_LOGS_MAX_SIZE:-100M}"

  kms:
    image: ${KMS_IMAGE:-kurento/kurento-media-server:7.1.1}
    restart: always
    network_mode: host
    ulimits:
      core: -1
    volumes:
      - /opt/openvidu/kms-crashes:/opt/openvidu/kms-crashes
      - ${OPENVIDU_RECORDING_PATH}:${OPENVIDU_RECORDING_PATH}
      - /opt/openvidu/kurento-logs:/opt/openvidu/kurento-logs
    environment:
      - KMS_MIN_PORT=40000
      - KMS_MAX_PORT=57000
      - GST_DEBUG=${KMS_DOCKER_ENV_GST_DEBUG:-}
      - KURENTO_LOG_FILE_SIZE=${KMS_DOCKER_ENV_KURENTO_LOG_FILE_SIZE:-100}
      - KURENTO_LOGS_PATH=/opt/openvidu/kurento-logs
    logging:
      options:
        max-size: "${DOCKER_LOGS_MAX_SIZE:-100M}"

  coturn:
    image: openvidu/openvidu-coturn:2.31.0
    restart: on-failure
    ports:
      - "${COTURN_PORT:-3478}:${COTURN_PORT:-3478}/tcp"
      - "${COTURN_PORT:-3478}:${COTURN_PORT:-3478}/udp"
    env_file:
```

```
      - .env
    volumes:
      - ./coturn:/run/secrets/coturn
    command:
      - --log-file=stdout
      - --listening-port=${COTURN_PORT:-3478}
      - --fingerprint
      - --min-port=${COTURN_MIN_PORT:-57001}
      - --max-port=${COTURN_MAX_PORT:-65535}
      - --realm=openvidu
      - --verbose
      - --use-auth-secret
      - --static-auth-secret=$${COTURN_SHARED_SECRET_KEY}
    logging:
      options:
        max-size: "${DOCKER_LOGS_MAX_SIZE:-100M}"

  nginx:
    image: openvidu/openvidu-proxy:2.31.0
    restart: always
    network_mode: host
    volumes:
      - ./certificates:/etc/letsencrypt
      - ./owncert:/owncert
      - ./custom-nginx-vhosts:/etc/nginx/vhost.d/
      - ./custom-nginx-locations:/custom-nginx-locations
      - ${OPENVIDU_RECORDING_CUSTOM_LAYOUT}:/opt/openvidu/custom-layout
    environment:
      - DOMAIN_OR_PUBLIC_IP=${DOMAIN_OR_PUBLIC_IP}
      - CERTIFICATE_TYPE=${CERTIFICATE_TYPE}
      - LETSENCRYPT_EMAIL=${LETSENCRYPT_EMAIL}
      - PROXY_HTTP_PORT=${HTTP_PORT:-}
      - PROXY_HTTPS_PORT=${HTTPS_PORT:-}
      - PROXY_HTTPS_PROTOCOLS=${HTTPS_PROTOCOLS:-}
      - PROXY_HTTPS_CIPHERS=${HTTPS_CIPHERS:-}
      - PROXY_HTTPS_HSTS=${HTTPS_HSTS:-}
      - ALLOWED_ACCESS_TO_DASHBOARD=${ALLOWED_ACCESS_TO_DASHBOARD:-}
      - ALLOWED_ACCESS_TO_RESTAPI=${ALLOWED_ACCESS_TO_RESTAPI:-}
      - PROXY_MODE=CE
      - WITH_APP=true
      - SUPPORT_DEPRECATED_API=${SUPPORT_DEPRECATED_API:-false}
      - REDIRECT_WWW=${REDIRECT_WWW:-false}
      - WORKER_CONNECTIONS=${WORKER_CONNECTIONS:-10240}
      - PUBLIC_IP=${PROXY_PUBLIC_IP:-auto-ipv4}
    logging:
      options:
        max-size: "${DOCKER_LOGS_MAX_SIZE:-100M}"
```

openvidu 시작

```
cd /opt/openvidu
./openvidu start
```

⇒ https://i12b103.p.ssafy.io:4443 하면 접속 가능

이 때 접속하는 id : admin, pw : 설정한 secret_key

6. 젠킨스 설정

EC2 환경에 jenkins 설치

```
docker pull jenkins/jenkins:lts
```

```
docker run -d \
  --name jenkins \
  --group-add $(stat -c '%g' /var/run/docker.sock) \
  --network app-network \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v jenkins_home:/var/jenkins_home \
  -p 8080:8080 \
  jenkins/jenkins:lts
```

⇒ http://i12b103.p.ssafy.io:8080/ 접근 가능

```
docker logs jenkins [젠킨스 서버 아이디]
중간 logs로 pw 알려줌 그걸로 접속
```

젠킨스 안에 docker 설정

- 일단 우분투 환경에 apt 저장소 업데이트

```
sudo apt-get update
sudo apt-get upgrade -y
```

```
docker exec -it -u root jenkins bash
apt-get update && apt-get install -y docker.io

exit
docker restart jenkins
docker exec -it jenkins docker --version
```

▼ excete shell

```
#!/bin/bash

# 환경 변수 설정
BACKEND_CONTAINER_NAME="backend"
BACKEND_IMAGE="backend:latest"
FRONTEND_IMAGE="frontend-build:latest"
FRONTEND_BUILD_DIR="/home/ubuntu/frontend_build"
NGINX_CONTAINER_NAME="nginx_server"

export SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL}
export SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME}
export SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD}
export SPRING_REDIS_HOST=${SPRING_REDIS_HOST}
export SPRING_REDIS_PORT=${SPRING_REDIS_PORT}
```

```
export OPENVIDU_URL=${OPENVIDU_URL}
export OPENVIDU_SECRET=${OPENVIDU_SECRET}
export JWT_ISSUER=${OPENVIDU_SECRET}
export JWT_SECRET_KEY=${JWT_SECRET_KEY}
export SESSION_INACTIVE_TIMEOUT=${SESSION_INACTIVE_TIMEOUT}
export LIQUIBASE_CHANGE_LOG=${LIQUIBASE_CHANGE_LOG}
export  OPENAI_API_KEY=${OPENAI_API_KEY}
export  RECORDING_DIRECTORY=${RECORDING_DIRECTORY}
export  FFMPEG_PATH=${FFMPEG_PATH}
export  GOOGLE_APPLICATION_CREDENTIALS=${GOOGLE_APPLICATION_CREDENTIALS}


# 개발/배포 환경 분리
if [ "$ENV" = "development" ]; then
   echo "VITE_APP_API_URL=http://localhost:8080" > FrontEnd/.env
else
   echo "VITE_APP_API_URL=https://i12b103.p.ssafy.io" > FrontEnd/.env
fi

# echo "NODE_ENV=production" > FrontEnd/.env

#  🔹 Jenkins Workspace로 이동
cd /var/jenkins_home/workspace/Jenkins_CI_CD || { echo "❌ Jenkins workspace 없음"; exit 1; }

#  🔹 기존 Nginx 컨테이너 삭제
echo "🛑 기존 Nginx 컨테이너 삭제..."
docker rm -f $NGINX_CONTAINER_NAME || true

#  🔹 프론트엔드 Docker 빌드
echo "🚀 [1] 프론트엔드 Docker 빌드 시작..."
docker build --no-cache -t frontend-build -f FrontEnd/Dockerfile FrontEnd  || { echo "❌ 프론트엔드 Docker 빌

#  🔹 Nginx 컨테이너 실행 (프론트엔드 제공)
echo "🚀 [2] Nginx 컨테이너 실행..."
docker run -d --name $NGINX_CONTAINER_NAME \
  --network app-network \
  -p 80:80 \
  -p 443:443 \
  -v /home/ubuntu/nginx/nginx.conf:/etc/nginx/conf.d/default.conf \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  $FRONTEND_IMAGE || { echo "❌ Nginx 컨테이너 실행 실패"; exit 1; }


echo "✅ Nginx 컨테이너가 정상 실행되었습니다!"

#  🔹 기존 백엔드 컨테이너 삭제
echo "🛑 [4] 기존 백엔드 컨테이너 삭제..."
docker rm -f $BACKEND_CONTAINER_NAME || true
docker image prune -f

#  🔹 백엔드 Docker 빌드
echo "🚀 [5] 백엔드 Docker 이미지 빌드..."
docker build -t $BACKEND_IMAGE -f BackEnd/mindon/Dockerfile BackEnd/mindon || { echo "❌ 백엔드 Docke

#  🔹 새로운 백엔드 컨테이너 실행
```

```
echo "🟢 [6] 새로운 백엔드 컨테이너 실행..."
docker run -d --name $BACKEND_CONTAINER_NAME \
  --network app-network \
  -p 8081:8080 \
  -e TZ=Asia/Seoul \
  -e "SPRING_PROFILES_ACTIVE=prod" \
  -e "SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL}" \
  -e "SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME}" \
  -e "SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD}" \
  -e "SPRING_REDIS_HOST=${SPRING_REDIS_HOST}" \
  -e "SPRING_REDIS_PORT=${SPRING_REDIS_PORT}" \
  -e "OPENVIDU_URL=${OPENVIDU_URL}" \
  -e "OPENVIDU_SECRET=${OPENVIDU_SECRET}" \
  -e "JWT_ISSUER=${JWT_ISSUER}" \
  -e "JWT_SECRET_KEY=${JWT_SECRET_KEY}" \
  -e "SESSION_INACTIVE_TIMEOUT=${SESSION_INACTIVE_TIMEOUT}" \
  -e "LIQUIBASE_CHANGE_LOG=${LIQUIBASE_CHANGE_LOG}" \
  -e "OPENAI_API_KEY=${OPENAI_API_KEY}" \
  -e "RECORDING_DIRECTORY=${RECORDING_DIRECTORY}" \
  -e "FFMPEG_PATH=${FFMPEG_PATH}" \
  -e "GOOGLE_APPLICATION_CREDENTIALS=${GOOGLE_APPLICATION_CREDENTIALS}" \
  -v /home/ubuntu/stt.json:/opt/stt.json \
  $BACKEND_IMAGE || { echo "❌ 백엔드 컨테이너 실행 실패"; exit 1; }

echo "✅ 백엔드 컨테이너가 정상 실행되었습니다!"
```

젠킨스 권한 변경

- /var/lib/jenkins 권한 변경

```
sudo mkdir -p /var/lib/jenkins/frontend
sudo chown -R jenkins:jenkins /var/lib/jenkins/frontend
sudo chmod -R 775 /var/lib/jenkins/frontend
```

  - `chown` → `jenkins` 유저가 접근할 수 있도록 소유권 변경.
  - `chmod` → 읽기/쓰기 권한 추가.
- 서비스 재시작

```
ps aux | grep jenkins
kill -9 <PID>
nohup java -jar /usr/share/jenkins/jenkins.war --httpPort=8080 > /var/log/jenkins.log 2>&1 &
```

젠킨스 컨테이너 환경에 node.js 설치

- 컨테이너 내부 접속

```
docker exec -it jenkons bash
```

- 설치

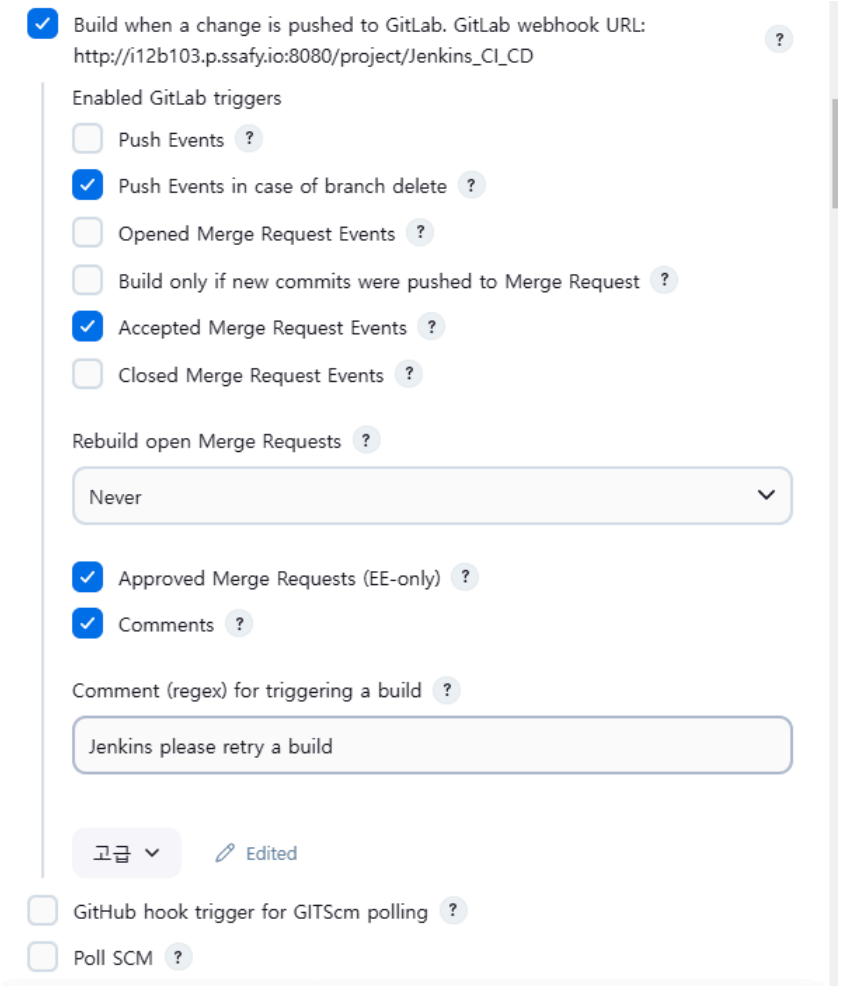
```
curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh | bash
touch ~/.bashrc
echo 'export NVM_DIR="$HOME/.nvm"' >> ~/.bashrc
```

```
echo '[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"' >> ~/.bashrc
echo '[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"' >> ~/.bashrc

source ~/.bashrc
nvm install 20.10.0
nvm use 20.10.0
nvm alias default 20.10.0
pm install -g yarn
```

젠킨스 웹 훅 설정

dev만 바라보도록 변경



7. 백엔드 Docker file 설정

Spring boot 포트 설정

8080 젠킨스 포트와 겹치지 않도록 8081로 설정

Docker file

ffmpeg 설정, openvidu 녹음 경로 저

```
# 1. Base 이미지 선택 (빌드용)
FROM openjdk:17-jdk-slim as builder

# ffmpeg 설치
RUN apt-get update && apt-get install -y ffmpeg

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. Gradle 빌드 환경 설정 (캐싱 최적화)
COPY gradlew .
COPY gradle gradle
COPY build.gradle settings.gradle ./
RUN chmod +x gradlew
RUN ./gradlew dependencies --no-daemon

# 4. 프로젝트 코드 복사 및 빌드 실행
COPY src src
RUN ./gradlew build --no-daemon

# 5. 실행용 JDK 이미지 설정
FROM openjdk:17-jdk-slim

# 타임존 설정 추가
ENV TZ=Asia/Seoul
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

# ffmpeg 설치 (실행 환경에도 필요할 경우 추가)
RUN apt-get update && apt-get install -y ffmpeg

# 6. 실행 디렉토리 설정
WORKDIR /app

# 7. 빌드된 JAR 파일 복사
COPY --from=builder /app/build/libs/*.jar app.jar

# 8. Spring Boot는 기본 8080 포트에서 실행됨
EXPOSE 8080

# 9. 컨테이너 실행 명령어
ENTRYPOINT ["java", "-jar", "app.jar"]
```

8. 프론트엔드 Docker file 설정

nginx.conf 파일 설정

1. 기존 /etc/nginx/nginx.conf 파일 백업

   ```
   sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak
   ```

2. /etc/nginx/nginx.conf 파일 설정

   ```
   sudo vim /etc/nginx/nginx.conf
   ```

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

http
    sendfile on;
    tcp_nopush on;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

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

    gzip on;
    gzip_disable "msie6";

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;  # ✅ sites-enabled에 있는 설정을 포함하는 부분 (중요)
}
```

3. /etc/nginx/sites-available/ 에 파일 생성
mindon이라고 생성

```
sudo nano /etc/nginx/sites-available/mindon
```

```
# HTTP 서버: 모든 요청을 HTTPS로 리다이렉트
server {
    listen 80;
    server_name i12b103.p.ssafy.io;

    # 모든 HTTP 요청을 HTTPS로 리다이렉트
    return 301 https://$host$request_uri;
}

# HTTPS 서버
server {
    listen 443 ssl;
    server_name i12b103.p.ssafy.io;

    # SSL 인증서 경로 (Let's Encrypt 인증서)
    ssl_certificate /etc/letsencrypt/live/i12b103.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i12b103.p.ssafy.io/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    # 보안: 숨김 파일(.env, .git 등) 접근 차단
```

```
    location ~ /\.(?!well-known).* {
        deny all;
        access_log off;
        log_not_found off;
    }

    # 정적 파일 및 SPA 라우팅
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    # API 프록시 설정: backend 컨테이너로 요청 전달
    location /api {
        proxy_pass http://backend: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;
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

- react 빌드 파일을 직접 서빙하도록 설정
  root /var/www/html; # React 빌드 파일을 위치할 곳
  index index.html;
- Spring boot api 8081 로 프록시 설정
- HTTPS 강제 리다이렉트
- SSL 인증서 반영
  - Let’s Encryp 인증서 적용

4. /etc/nginx/sites-enabled 심볼릭 링크 생성

```
sudo ln -s /etc/nginx/sites-available/mindon /etc/nginx/sites-enabled/
```

만약 sites-enabled에 다른 설정 파일이 있다면 삭제

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

프론트 엔드 Docker file 설정

```
# Node.js 20 버전 기반으로 빌드
FROM node:20.10.0 AS build
WORKDIR /app

# Corepack 활성화 및 Yarn Berry 설정
RUN corepack enable
```

```
##RUN yarn set version berry

# Yarn 캐시 폴더를 Linux 환경에 맞게 재설정
##RUN yarn config set cacheFolder /app/.yarn/cache

# package.json 및 yarn.lock 복사 (프로젝트 의존성 파일)
COPY package.json yarn.lock ./

# 의존성 설치 및 타입스크립트 설치
RUN yarn install
##RUN yarn add -D typescript
##RUN yarn install --check-cache

# 나머지 소스 파일 복사
COPY . .

# .pnp.cjs 파일이 생성되었는지 확인 (PnP 모드 정상 작동 여부 점검)
##RUN test -f /app/.pnp.cjs || { echo "❌ .pnp.cjs 파일이 없습니다! 빌드를 중단합니다."; exit 1; }

# 빌드 실행 (yarn build 스크립트가 package.json에 등록되어 있어야 함)
RUN yarn build

# 빌드 결과물 확인
RUN ls -al /app/ && ls -al /app/dist/ || echo "⚠️ 빌드된 dist 폴더가 없습니다!"

# Nginx 컨테이너를 위한 멀티 스테이지 빌드: 빌드 결과물만 복사
FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html

EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"]
```

pnp 설정 파일 읽기

- 로컬에 있는 .pnp.cjs, .pnp.loader.mjs 파일 지워야 리눅스에서 읽을 수 있음
- 또는 프론트에서 그 부분을 /app 으로 수정을 해야함.
- vit.config.ts 파일 수정

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import path from "path";  // path 모듈 import

export default defineConfig({
  server: {
    port: 3000,
  },
  plugins: [react()],
  resolve: {
    alias: {
      "@": path.resolve(__dirname, "src"),
      "@components": path.resolve(__dirname, "src/components"),
      "@pages": path.resolve(__dirname, "src/pages"),
      "@assets": path.resolve(__dirname, "src/assets"),
      "@apis": path.resolve(__dirname, "src/apis"),
```

```
      "@stores": path.resolve(__dirname, "src/stores"),
      "@hooks": path.resolve(__dirname, "src/hooks"),
      "@utils": path.resolve(__dirname, "src/utils"),
    },
  },
});
```

9. 젠킨스 셸 설정

```
#!/bin/bash

# 환경 변수 설정
BACKEND_CONTAINER_NAME="backend"
BACKEND_IMAGE="backend:latest"
FRONTEND_IMAGE="frontend-build:latest"
FRONTEND_BUILD_DIR="/home/ubuntu/frontend_build"
NGINX_CONTAINER_NAME="nginx_server"

export SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL}
export SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME}
export SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD}
export SPRING_REDIS_HOST=${SPRING_REDIS_HOST}
export SPRING_REDIS_PORT=${SPRING_REDIS_PORT}
export OPENVIDU_URL=${OPENVIDU_URL}
export OPENVIDU_SECRET=${OPENVIDU_SECRET}
export JWT_ISSUER=${OPENVIDU_SECRET}
export JWT_SECRET_KEY=${JWT_SECRET_KEY}
export SESSION_INACTIVE_TIMEOUT=${SESSION_INACTIVE_TIMEOUT}
export LIQUIBASE_CHANGE_LOG=${LIQUIBASE_CHANGE_LOG}
export  OPENAI_API_KEY=${OPENAI_API_KEY}
export  RECORDING_DIRECTORY=${RECORDING_DIRECTORY}
export  FFMPEG_PATH=${FFMPEG_PATH}
export  GOOGLE_APPLICATION_CREDENTIALS=${GOOGLE_APPLICATION_CREDENTIALS}


# 개발/배포 환경 분리
if [ "$ENV" = "development" ]; then
   echo "VITE_APP_API_URL=http://localhost:8080" > FrontEnd/.env
else
   echo "VITE_APP_API_URL=https://i12b103.p.ssafy.io" > FrontEnd/.env
fi

# echo "NODE_ENV=production" > FrontEnd/.env

#  🔹 Jenkins Workspace로 이동
cd /var/jenkins_home/workspace/Jenkins_CI_CD || { echo "❌ Jenkins workspace 없음"; exit 1; }

#  🔹 기존 Nginx 컨테이너 삭제
echo "🛑 기존 Nginx 컨테이너 삭제..."
docker rm -f $NGINX_CONTAINER_NAME || true

#  🔹 프론트엔드 Docker 빌드
echo "🚀 [1] 프론트엔드 Docker 빌드 시작..."
docker build --no-cache -t frontend-build -f FrontEnd/Dockerfile FrontEnd  || { echo "❌ 프론트엔드 Docker 빌드 
```

```
# 🔹 Nginx 컨테이너 실행 (프론트엔드 제공)
echo "🚀 [2] Nginx 컨테이너 실행..."
docker run -d --name $NGINX_CONTAINER_NAME \
  --network app-network \
  -p 80:80 \
  -p 443:443 \
  -v /home/ubuntu/nginx/nginx.conf:/etc/nginx/conf.d/default.conf \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  $FRONTEND_IMAGE || { echo "❌ Nginx 컨테이너 실행 실패"; exit 1; }


echo "✅ Nginx 컨테이너가 정상 실행되었습니다!"

# 🔹 기존 백엔드 컨테이너 삭제
echo "🛑 [4] 기존 백엔드 컨테이너 삭제..."
docker rm -f $BACKEND_CONTAINER_NAME || true
docker image prune -f

# 🔹 백엔드 Docker 빌드
echo "🚀 [5] 백엔드 Docker 이미지 빌드..."
docker build -t $BACKEND_IMAGE -f BackEnd/mindon/Dockerfile BackEnd/mindon || { echo "❌ 백엔드 Docker 빌

# 🔹 새로운 백엔드 컨테이너 실행
echo "🟢 [6] 새로운 백엔드 컨테이너 실행..."
docker run -d --name $BACKEND_CONTAINER_NAME \
  --network app-network \
  -p 8081:8080 \
  -e TZ=Asia/Seoul \
  -e "SPRING_PROFILES_ACTIVE=prod" \
  -e "SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL}" \
  -e "SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME}" \
  -e "SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD}" \
  -e "SPRING_REDIS_HOST=${SPRING_REDIS_HOST}" \
  -e "SPRING_REDIS_PORT=${SPRING_REDIS_PORT}" \
  -e "OPENVIDU_URL=${OPENVIDU_URL}" \
  -e "OPENVIDU_SECRET=${OPENVIDU_SECRET}" \
  -e "JWT_ISSUER=${JWT_ISSUER}" \
  -e "JWT_SECRET_KEY=${JWT_SECRET_KEY}" \
  -e "SESSION_INACTIVE_TIMEOUT=${SESSION_INACTIVE_TIMEOUT}" \
  -e "LIQUIBASE_CHANGE_LOG=${LIQUIBASE_CHANGE_LOG}" \
  -e "OPENAI_API_KEY=${OPENAI_API_KEY}" \
  -e "RECORDING_DIRECTORY=${RECORDING_DIRECTORY}" \
  -e "FFMPEG_PATH=${FFMPEG_PATH}" \
  -e "GOOGLE_APPLICATION_CREDENTIALS=${GOOGLE_APPLICATION_CREDENTIALS}" \
  -v /home/ubuntu/stt.json:/opt/stt.json \
  $BACKEND_IMAGE || { echo "❌ 백엔드 컨테이너 실행 실패"; exit 1; }

echo "✅ 백엔드 컨테이너가 정상 실행되었습니다!"
```

10. 배포 / 개발 환경 분리

공통

- application.yml 파일

```
spring:
  jackson:
    date-format: yyyy-MM-dd'T'HH:mm:ss'Z'
    time-zone: Asia/Seoul
  config:
    import: optional:file:.env[.properties]
  profiles:
    active: ${SPRING_PROFILES_ACTIVE:dev}
  ssl:
    enabled: false

  jpa:
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.MySQL8Dialect
        jdbc: time_zone=Asia/Seoul
    defer-datasource-initialization: true

  datasource:
    url: ${SPRING_DATASOURCE_URL}
    username: ${SPRING_DATASOURCE_USERNAME}
    password: ${SPRING_DATASOURCE_PASSWORD}

  data:
    redis:
      host: ${SPRING_REDIS_HOST}
      port: ${SPRING_REDIS_PORT}
  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB # 최대 업로드 가능한 파일 크기
      max-request-size: 10MB # 최대 요청 크기

jwt:
  issuer: ${JWT_ISSUER}
  secret_key: ${JWT_SECRET_KEY}

liquibase:
  change-log: ${LIQUIBASE_CHANGE_LOG}

session:
  inactive-timeout: ${SESSION_INACTIVE_TIMEOUT}

openai:
  api-key: ${OPENAI_API_KEY}

openvidu:
  url: ${OPENVIDU_URL}
  secret: ${OPENVIDU_SECRET}

google:
  credentials:
```

```
    file: ${GOOGLE_APPLICATION_CREDENTIALS}

file:
  upload-dir: /tmp # 파일 업로드 디렉토리 설정

recording:
  directory: ${RECORDING_DIRECTORY}

ffmpeg:
  path: ${FFMPEG_PATH}
```

개발 환경 변수 설정

프론트 엔드

- .env.development 파일

```
VITE_APP_API_URL=https://i12b103.p.ssafy.io"
```

백엔드

- .env.development 파일

```
SPRING_DATASOURCE_URL=jdbc:mysql://localhost:3306/on_board?useSSL=false&serverTimezone=Asia/Se
SPRING_DATASOURCE_USERNAME={db 이름}
SPRING_DATASOURCE_PASSWORD= {db 비밀번호}
SPRING_REDIS_HOST=localhost
SPRING_REDIS_PORT= {포트번호}

OPENVIDU_URL=http://i12b103.p.ssafy.io:5443
OPENVIDU_SECRET={비밀키}

JWT_ISSUER=jsjs3489@gmail.com
JWT_SECRET_KEY={비밀키}

SESSION_INACTIVE_TIMEOUT=31536000
LIQUIBASE_CHANGE_LOG=classpath:/db/changelog/changelog-master.yaml

OPENAI_API_KEY={ai 비밀키}

RECORDING_DIRECTORY=C:/dev/recordings/

FFMPEG_PATH=C:/Users/SSAFY/FFMPEG/ffmpeg-2025-02-13-git-19a2d26177-full_build/bin/ffmpeg.exe

GOOGLE_APPLICATION_CREDENTIALS={경로}
```

- application-dev.yml

```
spring:
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update

  datasource:
```

```
    url: ${SPRING_DATASOURCE_URL}
    username: ${SPRING_DATASOURCE_USERNAME}
    password: ${SPRING_DATASOURCE_PASSWORD}

  data:
    redis:
      host: ${SPRING_REDIS_HOST}
      port: ${SPRING_REDIS_PORT}

openvidu:
  url: ${OPENVIDU_URL}
  secret: ${OPENVIDU_SECRET}
```

배포 환경 변수 설정

젠킨스 Credentials 에 환경 변수 설정 후 jenkins shell 에서 가져오기

- stt 경로의 경우 ec2 환경 내 stt.json 파일 설정 후 경로를 볼륨 마운트