



# 포팅 메뉴얼

## ▼ 1 프로젝트 기술 스택

### Frontend

- react 18.2.0,
- redux 5.0.1
- redux-persist 6.0.0
- flowbite-react 0.7.2
- tailwindcss 3.4.1
- socket.io 4.7.4

### Web RTC

- openvidu-browser 2.29.1

### 협업 툴

- Gitlab
- Jira
- Notion
- Mattermost

## ▼ 2 백엔드 빌드

### ▼ build.gradle

```
// bulid.gradle

plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.1'
    id 'io.spring.dependency-management' version '1.1.4'
}
```

### Backend

- spring boot 3.2.1
- spring-boot-starter-data-jpa
- spring security 6.2.1
- lombok 1.18.30
- azul-17
- mysql 8.2
- redis 7.0.4

### CI/CD

- docker 25.0.1
- docker-compose 2.21.0
- jenkins LTS 2.401.1

### UI/UX

- figma

```

group = 'com.ssafy.fiveguys'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
    dependencies {
        implementation group: 'com.auth0', name: 'java-jwt',
version: '4.4.0'
        implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
        implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
        implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'
        implementation 'org.springframework.boot:spring-boot-
starter-data-jpa'
        implementation 'org.springframework.boot:spring-boot-
starter-data-redis'
        implementation 'org.springframework.boot:spring-boot-
starter-security'
        implementation 'org.springframework.boot:spring-boot-
starter-oauth2-client'
        implementation 'org.springframework.boot:spring-boot-
starter-web'
        implementation 'org.springframework.boot:spring-boot-
starter-aop'
        implementation group: 'org.springframework.boot', nam
e: 'spring-boot-starter-mail'
        implementation 'org.springdoc:springdoc-openapi-start
er-webmvc-ui:2.1.0'
        implementation 'org.springframework.boot:spring-boot-
starter-thymeleaf'
        implementation 'org.springframework.boot:spring-boot-
starter-validation'
        implementation 'com.querydsl:querydsl-jpa:5.0.0:jakar
ta'
    }
}

```

```

        implementation 'com.github.gavlyukovskiy:p6spy-spring
-boot-starter:1.5.6'
        // 스프링 부트 스타터
        implementation 'org.springframework.boot:spring-boot-
starter-data-neo4j'
        // Neo4j 드라이버 의존성
        implementation 'org.neo4j.driver:neo4j-java-driver'
        compileOnly 'org.projectlombok:lombok'
        developmentOnly 'org.springframework.boot:spring-boot
-devtools'
        runtimeOnly 'com.mysql:mysql-connector-j'
        annotationProcessor 'org.projectlombok:lombok'
        annotationProcessor "com.querydsl:querydsl-apt:${depe
ndencyManagement.importedProperties['querydsl.version']}:jaka
rta"
        annotationProcessor "jakarta.annotation:jakarta.annot
ation-api"
        annotationProcessor "jakarta.persistence:jakarta.pers
istence-api"
        testImplementation 'org.springframework.boot:spring-b
oot-starter-test'
        testImplementation 'org.springframework.security:spr
ing-security-test'
    }
}

tasks.named('test') {
    useJUnitPlatform()
}

clean {
    delete file('src/main/generated')
}

```

**⚠ src > main > resources에 application-oauth.properties, application-smtp.properties 추가한후 application.properties에**  
**spring.profiles.include=oauth, smtp 추가**

▼ application-oauth.properties

```

#google
spring.security.oauth2.client.registration.google.client-id=
구글에서 발급 받은 클라이언트 아이디
spring.security.oauth2.client.registration.google.client-secret=
구글에서 발급 받은 클라이언트 시크릿 키
spring.security.oauth2.client.registration.google.scope=profile,email
spring.security.oauth2.client.registration.google.redirect-uri=
도메인 주소/login/oauth2/code/google

# naver
spring.security.oauth2.client.registration.naver.client-id=
네이버에서 발급 받은 클라이언트 아이디
spring.security.oauth2.client.registration.naver.client-secret=
네이버에서 발급 받은 클라이언트 시크릿 키
spring.security.oauth2.client.registration.naver.scope=name,
email
spring.security.oauth2.client.registration.naver.authorization-grant-type=
authorization_code
spring.security.oauth2.client.registration.naver.redirect-uri=
도메인 주소/login/oauth2/code/naver
spring.security.oauth2.client.registration.naver.client-name=Naver
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

#kakao
spring.security.oauth2.client.registration.kakao.client-id=
카카오에서 발급 받은 클라이언트 아이디
spring.security.oauth2.client.registration.kakao.client-secret=
카카오에서 발급 받은 클라이언트 시크릿 키
spring.security.oauth2.client.registration.kakao.scope=profile_nickname,
account_email
spring.security.oauth2.client.registration.kakao.client-authentication-
method=client_secret_post
spring.security.oauth2.client.registration.kakao.redirect-uri=

```

```

i=도메인 주소/login/oauth2/code/kakao
spring.security.oauth2.client.registration.kakao.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.kakao.client-name=kakao
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

```

#### ▼ application-smtp.properties

```

## Google SMTP
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=구글 이메일
spring.mail.password=발급받은 비밀번호
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.auth=true

```

#### ▼ application-properties

```

spring.output.ansi.enabled=always

# MySQL Database
spring.datasource.url=jdbc:mysql://localhost:3306/secretzoo?characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=fiveguys
spring.datasource.password=zoo1qsecret2wguys3efive4r!
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# redis for server
# redis for spring security
spring.data.redis.host=redis1
spring.data.redis.port=6379
# redis for ranking
spring.data.redis.host2=redis2
spring.data.redis.port2=6379

```

```
##redis for local test
#spring.data.redis.host=localhost
#spring.data.redis.port=6379
#spring.data.redis.host2=localhost
#spring.data.redis.port2=6380

# UTF-8
server.servlet.encoding.charset=UTF-8
server.servlet.encoding.force=true
spring.jackson.time-zone=Asia/Seoul

# JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=mysql
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
#spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.properties.hibernate.format_sql=true

logging.level.root=info
logging.level.com.ssafy.fiveguys.game=debug

#oauth
spring.profiles.include=oauth, smtp

spring.devtools.livereload.enabled=true
spring.freemarker.cache=false

#rank 스케줄링 관련 변수
rank.update.interval.minutes=1
rank.max.count=10
rank.update.interval=10000

#static 경로
#spring.resources.static-locations=classpath:/static/
```

```
#서버 에러 메시지 설정
server.error.whitelabel.enabled=false
server.error.include-exception=true
server.error.include-message=never
server.error.include-stacktrace=never
server.error.include-binding-errors=never

#타임 리프 설정
#spring.thymeleaf.cache=false

#swagger 설정
springdoc.swagger-ui.path=/swagger-ui
springdoc.api-docs.path=/api-docs
springdoc.swagger-ui.disable-swagger-default-url=true
```

로컬 테스트 환경과 서버 배포 환경에서 변경해야 하는 부분은 application-properties에 redis 설정 부분에 있습니다.

### 로컬 테스트 환경

로컬에서는 redis를 localhost의 port번호로 구분해서 접속가능하다. 이때 redis는 도커 컨테이너로 6379, 6380 포트를 개방해서 각각 실행 시킨 상태로 준비하면 된다.

#### application-properties

```
##redis for local test
spring.data.redis.host=localhost
spring.data.redis.port=6379
spring.data.redis.host2=localhost
spring.data.redis.port2=6380
```

### 서버 배포 환경

서버 배포시에는 도커 컨테이너들이 동일한 네트워크에 묶여 있기 때문에, 컨테이너 이름으로 접근이 가능하다. 따라서, 컨테이너 이름으로 host를 식별하고 port는 동일한 6379 포트를 사용한다.

#### application-properties

```
# redis for server
# redis for spring security
spring.data.redis.host=redis1
```

```
spring.data.redis.port=6379
# redis for ranking
spring.data.redis.host2=redis2
spring.data.redis.port2=6379
```

## ▼ 3 프론트 엔드 빌드

# React Client

## 패키지 버전

```
"@emotion/react": "^11.11.3",
"@emotion/styled": "^11.11.0",
"@reduxjs/toolkit": "^2.1.0",
"@testing-library/jest-dom": "^5.17.0",
"@testing-library/react": "^13.4.0",
"@testing-library/user-event": "^13.5.0",
"axios": "^1.6.5",
"flowbite-react": "^0.7.2",
"framer-motion": "^11.0.3",
"install": "^0.13.0",
"npm": "^10.3.0",
"openvidu-browser": "^2.29.1",
"react": "^18.2.0",
"react-confetti": "^6.1.0",
"react-dom": "^18.2.0",
"react-lottie": "^1.2.4",
"react-redux": "^9.1.0",
"react-router-dom": "^6.21.3",
"react-scripts": "^5.0.1",
"react-transition-group": "^4.4.5",
"redux": "^5.0.1",
"redux-persist": "^6.0.0",
"redux-toolkit": "^1.1.2",
"socket.io-client": "^4.7.4",
"sweetalert2": "^11.10.5",
"tailwindcss": "^3.4.1",
"uuid": "^9.0.1",
"web-vitals": "^2.1.4"
```



## 패키지 설치

```
frontend/client > npm i
```

## 클라이언트 실행

```
frontend/client > npm start
```

# Node Server

## 패키지 버전

```
"axios": "^1.6.7",  
"cors": "^2.8.5",  
"express": "^4.18.2",  
"openvidu-browser": "^2.29.1",  
"socket.io": "^4.7.4"
```

## 패키지 설치

```
frontend/server > npm i
```

## 소켓 서버 실행

```
frontend/server > node index.js
```

## 로컬 테스트 환경

로컬에서는 노드 서버를 `localhost :3001` 로 접속한다.

- frontend/client/src/App.js

App.js : 25번째 줄

```
const socket = io("http://localhost:3001"); //로컬 서버
```

- frontend/server/index.js

index.js : 45번째 줄

```
const serverURL = "http://localhost:3000";
```

## 서버 배포 환경

서버 배포 환경에서는 서버 도메인으로 접속한다.

- frontend/client/src/App.js

App.js : 25번째 줄

```
const socket = io('https://secretzoo.site'); //서버 도메인
```

- frontend/server/index.js

index.js : 45번째 줄

```
const serverURL = 'https://secretzoo.site'
```

## ▼ 4 EC2 서버 환경 설정

### (1) 우분투 서버 한국 표준시로 변경 (UTC+9)

```
sudo timedatectl set-timezone Asia/Seoul
```

### (2) 카카오 미러 서버 활용

- 기본 서버가 \*.ubuntu.com 이라는 해외 서버이기 때문에, 패키지 갱신 속도가 비교적 빠른 국내 미러 서버를 활용하는 것이 효율적임. 가장 많이 이용하는 성능 좋은 미러 서버는 카카오 서버

```
sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/g' /etc/apt/sources.list
```

- 미러 서버 업데이트 후

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

### (3) SWAP 영역 할당

- 스왑 영역 할당 (ex : 4GB)

```
sudo fallocate -l 4G /swapfile
```

- swapfile 권한 수정

```
sudo chmod 600 /swapfile
```

- swapfile 생성

```
sudo mkswap /swapfile
```

- swapfile 활성화

```
sudo swapon /swapfile
```

- 시스템이 재부팅해도 swap 유지 설정

```
sudo echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

- swap 영역이 할당 확인

```
free -h
```

## ▼ 5 EC2 도커 관련 가이드

### (1) Docker 설치

- Docker 기본 설치 및 설정 과정

```
# docker설치
sudo apt-get -y install docker-ce docker-ce-cli containerd.io

# ubuntu 유저에게 권한 부여 -> sudo 없이 docker 명령어 사용 가능
sudo usermod -aG docker ubuntu

# docker 서비스 재시작
sudo service docker restart

# 설치 버전 확인
docker -v
```

- Docker-compose 설치

```
# docker-compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/v2.21.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# 권한 변경
sudo chmod +x /usr/local/bin/docker-compose

# 설치 버전 확인
docker-compose -v
```

- Docker 기본 명령어

```
# 동작 중인 컨테이너 확인
docker ps

# Docker 컨테이너 모두 조회 (실행, 종료, 재시작 등 모든 상태 조회)
docker ps -a

# 실행되고 있는 컨테이너 로그 조회
docker logs {컨테이너 ID or 컨테이너 이름}

# 로그 실시간으로 보는 방법
docker logs -f {컨테이너 ID or 컨테이너 이름}

# 컨테이너 삭제
docker rm {컨테이너 ID}

# 컨테이너 모두 삭제
docker rm `docker ps -a -q`

# 이미지 확인
docker image ls

# 이미지 삭제
docker rmi {이미지 ID}
```

## ▼ 6 Jenkins / Nginx

## (1) Jenkins / Nginx 설치 및 실행 (Docker)

- jenkins - Dockerfile
- nginx

docker-compose.yml 파일 위치에서 docker-compose up -d 명령어 입력하면 jenkins 컨테이너와 nginx 컨테이너가 실행

### docker-compose.yml

```
version: '3'
services:
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile
    # image: jenkins/jenkins:lts
    container_name: jenkins
    # ports: # nginx를 리버스 프록시 / 이 상황은 WAS를 노출 안시킴 (nginx가 jenkins로 들어간다.)
    #   - 8080:8080
    volumes:
      - /home/ubuntu/jenkins:/var/jenkins_home #host의 jenkins_home을 가져와서 ubuntu의 jenkins로 가져와서 추가
      - /home/ubuntu/.ssh:/var/jenkins_home/.ssh #젠킨스의 ssh의 명령어를 걸 때 호스트의 .ssh 인증서를 공용해서 씬
      - /var/run/docker.sock:/var/run/docker.sock #host의 docker engine 사용을 위해 추가
    networks:
      - docker_nat

  nginx:
    image: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
      - /home/ubuntu/secretzoo:/etc/nginx/secretzoo/
      - /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d # conf.d 를 만듦 (nginx를 통해서 jenkins)
      - /home/ubuntu/nginx/cert:/etc/cert # 인증서 파일을 공유시키기 위해서
      - /etc/letsencrypt:/etc/cert2
```

```

restart: always # 꺼져도 다시 실행
depends_on:
  - jenkins # jenkins가 실행되고 나서 nginx를 실행하겠다는 의미
networks:
  - docker_nat # 네트워크는 docker_nat (가상네트워크 그룹을 만들어
서 nginx랑 jenkins가 docker_nat 네트워크에서 사용한다.)

networks:
  docker_nat:
    external: true

```

## Dockerfile

젠킨스 내부에 도커와 도커 컴포즈를 설치해서 파이프라인에서 docker 관련 명령어를 사용할 수 있도록 환경 설정

```

FROM jenkins/jenkins:lts
USER root

RUN apt-get update && \
  apt-get -y install apt-transport-https \
  ca-certificates \
  curl \
  gnupg2 \
  software-properties-common && \
  curl -fsSL https://download.docker.com/linux/$(. /etc/os-re
lease; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
  add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /
etc/os-release; echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
  apt-get update && \
  apt-get -y install docker-ce

RUN groupadd -f docker
RUN usermod -aG docker jenkins

# 도커 컴포즈 설치
RUN curl -L "https://github.com/docker/compose/releases/latest/
download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/

```

```
bin/docker-compose && \
  chmod +x /usr/local/bin/docker-compose
```

## (2) Letsencrypt SSL 인증서 발급

```
## letsencrypt 설치
sudo apt update
sudo apt-get install letsencrypt -y

## 설치 확인
sudo certbot --help

## 도메인 입력후 SSL 인증서 발급
sudo certbot certonly --manual --preferred-challenges dns -d
"*.secretzoo.site" -d "secretzoo.site"

## 화면에 나오는 난수를 DNS의 TXT 레코드에 등록 후 ENTER
```

### DNS의 TXT 레코드 예시

DNS 설정 <span>레코드 수정</span>					
타입	호스트	값/위치	TTL	우선 순위	서비스
A	@	13.125.230.27	600		DNS 설정
A	spring	13.125.230.27	1800		DNS 설정
A	openvidu	13.125.230.27	1800		DNS 설정
TXT	_acme-challenge	"hXINJ8rhW7ms70D0YsglYdZnLkrqWegUOslof532Fms"	600		DNS 설정
TXT	_acme-challenge	"C2JDApaaCOboH7IiIZG4MJ8pbil7UfcEnv5vcHS3dE"	600		DNS 설정
TXT	_acme-challenge.openvidu.secretzoo.site	"kNPvkiEnn1F_BuMW2i85D4VvFefReeO7JazoPLYJS-U"	600		DNS 설정

```
## 인증서 발급 확인
sudo ls /etc/letsencrypt/live
```

## ▼ 7 Spring / React

### (1) 백엔드 서버 빌드 및 실행 (docker)

- spring - Dockerfile로 자체 이미지 생성후 사용
- mysql - mysql8.2 공식 이미지 사용
- redis - redis:latest 공식 이미지 사용

spring 백엔드 서버는 mysql, redis 데이터베이스 컨테이너를 실행 시킨 후에 실행 시키도록 의존성 관계를 설정해서 컨테이너를 빌드한다.

백엔드 프로젝트의 최상위 디렉토리에 docker-compose.yml 파일을 위치시킨 후에, 해당 디렉토리의 위치에서 `docker-compose up -d` 명령어로 백엔드 서버를 실행

### docker-compose.yml

```
version: '3'

services:
  mysql:
    image: mysql:8.2
    command: mysqld --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci --default-authentication-plugin=mysql_native_password
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: zoo1qsecret2wguys3efive4r!
      MYSQL_DATABASE: secretzoo
      MYSQL_ROOT_HOST: '%'
      MYSQL_USER: fiveguys
      MYSQL_PASSWORD: zoo1qsecret2wguys3efive4r!
      TZ: 'Asia/Seoul'
    ports:
      - "3306:3306"
    volumes:
      - "/home/ubuntu/db/mysql_vol:/var/lib/mysql"

  networks:
    - docker_nat

  redis1:
    image: redis:latest
    container_name: redis1
    hostname: redis1
    volumes:
      - /home/ubuntu/db/redis1_vol:/data
```



```

        - /home/ubuntu/db/redis1_vol/redis1.conf:/etc/redis/redis.conf
    # ports: # 외부 포트 닫음
    #   - 6379:6379
    extra_hosts:
        - host.docker.internal:host-gateway
    networks:
        - docker_nat

redis2:
    image: redis:latest
    container_name: redis2
    hostname: redis2
    volumes:
        - /home/ubuntu/db/redis2_vol:/data
        - /home/ubuntu/db/redis2_vol/redis2.conf:/etc/redis/redis.conf
    # ports:
    #   - 6380:6379
    extra_hosts:
        - host.docker.internal:host-gateway
    networks:
        - docker_nat

spring-app:
    container_name: spring-app
    build:
        context: .
        dockerfile: Dockerfile
    # ports:
    #   - 8080:8080
    networks:
        - docker_nat
    environment:
        SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/secretzoo?
        autoReconnect=true&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC&useLegacyDatetimeCode=false
        SPRING_DATASOURCE_USERNAME: "root"
        SPRING_DATASOURCE_PASSWORD: "zoo1qsecret2wguys3efive4r!"
        SPRING_REDIS_HOST: redis1
        SPRING_REDIS_PORT: 6379
        SPRING_REDIS_HOST2: redis2

```

```

    SPRING_REDIS_PORT2: 6379
  depends_on:
    - mysql
    - redis1
    - redis2
  networks:
    docker_nat:
      external: true

```

## Dockerfile

spring 서버는 Dockerfile로 컨테이너 이미지를 생성

.jar file 위치 → build/libs/exmple.jar

```

FROM openjdk:17-alpine

#build된 jar 파일
ARG JAR_FILE_PATH=build/libs

# 호스트의 JAR 파일을 컨테이너로 복사
COPY ${JAR_FILE_PATH}/game-0.0.1-SNAPSHOT.jar myboot.jar

# 실행시 사용할 환경 변수 설정 (예: 프로파일 설정)
# ENV SPRING_PROFILES_ACTIVE=dev,oauth

ENTRYPOINT ["java", "-jar", "./myboot.jar"]
# 환경 변수 사용하는 경우
# ENTRYPOINT ["java", "-Dspring.profiles.active=${SPRING_PROFILES_ACTIVE}", "-jar", "./myboot.jar"]

```

## (2) 프론트엔드 서버 빌드 및 실행(Docker)

- react- Dockerfile로 리액트 자체 이미지 빌드
- node - Dockerfile로 노스 자체 이미지 빌드

프론트 프로젝트의 최상위 디렉토리에 docker-compose.yml 파일을 위치시킨 후에, 해당 디렉토리의 위치에서 `docker-compose up -d` 명령어로 프론트 서버를 실행

### docker-compose.yml

```

version: '3'

services:
  react-app:
    build:
      context: ./client
      dockerfile: Dockerfile
    container_name: react-app
    expose:
      - 3000
    networks:
      - docker_nat

  node-server:
    build:
      context: ./server
      dockerfile: Dockerfile # Node.js 서버를 위한 Dockerfile
    container_name: node-server
    expose:
      - 3001
    restart: always
    networks:
      - docker_nat
networks:
  docker_nat:
    external: true

```

## Dockerfile - react

```

# 기반 이미지 설정
FROM node:20

# 앱 디렉터리 생성
WORKDIR /usr/src/app/client

# 앱 종속성 설치
COPY package.json ./

RUN npm install

# 앱 소스 추가

```

```

COPY . .

# 실행 파일에 대한 실행 권한을 부여합니다.
RUN chmod +x node_modules/.bin/*

# 앱 빌드
RUN npm run build

# 앱 실행
CMD ["npm", "start"]

```

### Dockerfile - node

```

# 기반 이미지 설정
FROM node:20

# 앱 디렉터리 생성
WORKDIR /usr/src/app/server

# 앱 종속성 설치
COPY package.json ./

RUN npm install

# 앱 소스 추가
COPY . .

# 실행 파일에 대한 실행 권한을 부여합니다.
RUN chmod +x node_modules/.bin/*

# 앱 실행
CMD ["node", "index.js"]

```

## ▼ 8 Openvidu

### openvidu 시그널링 spring 서버 빌드

secretzoo/openvidu/docker-compose.yml

해당 파일 위치에서 **docker-compose up -d**

**docker-compose.yml**

```
version: '3'

services:
  openvidu-app:
    container_name: openvidu-app
    build:
      context: .
      dockerfile: Dockerfile
    networks:
      - docker_nat
    environment:
      - SERVER_PORT=5442
      - OPENVIDU_URL=https://openvidu.secretzoo.site:8443
      - OPENVIDU_SECRET=MY_SECRET
    networks:
      docker_nat:
        external: true
```

openvidu on premise 설정

/opt/openvidu/docker-compose.yml

해당 파일 위치에서 **docker-compose up -d**

**docker-compose.yml**

```
version: '3.1'

services:
  openvidu-server:
    image: openvidu/openvidu-server:2.29.0
    restart: on-failure
    network_mode: host
    #networks: docker_nat
    entrypoint: ['/usr/local/bin/entrypoint.sh']
    volumes:
      - ./coturn:/run/secrets/coturn
      - /var/run/docker.sock:/var/run/docker.sock
      - ${OPENVIDU_RECORDING_PATH}:${OPENVIDU_RECORDING_P
```

```

ATH}
    - ${OPENVIDU_RECORDING_CUSTOM_LAYOUT}:${OPENVIDU_RECORDING_CUSTOM_LAYOUT}
    - ${OPENVIDU_CDR_PATH}:${OPENVIDU_CDR_PATH}
env_file:
    - .env
environment:
    - SERVER_SSL_ENABLED=false
    - SERVER_PORT=5443
    - KMS_URI=[ "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:7.0.1}
    restart: always
    network_mode: host
    #networks: docker_nat
    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:2.29.0

```

```

restart: on-failure
#network_mode: host
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:2.29.0
  restart: always
  network_mode: host
  #networks: docker_nat
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt
      #- ./certificates:/etc/letsencrypt
    - ./owncert:/owncert
    - ./custom-nginx-vhosts:/etc/nginx/vhost.d/
    - ./custom-nginx-locations:/custom-nginx-locations
    - ${OPENVIDU_RECORDING_CUSTOM_LAYOUT}:/opt/openvid
u/custom-layout
  environment:
    - DOMAIN_OR_PUBLIC_IP=${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=${ALLOWED_ACCESS_TO_D
ASHBOARD:-}
- ALLOWED_ACCESS_TO_RESTAPI=${ALLOWED_ACCESS_TO_RES
TAPI:-}
- 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 도메인, 비밀번호, PORT 설정

/opt/openvidu/ .env

.env

```

DOMAIN_OR_PUBLIC_IP=openvidu.secretzoo.site

# OpenVidu SECRET used for apps to connect to OpenVidu server a
nd users to access to OpenVidu Dashboard
OPENVIDU_SECRET=MY_SECRET

# Certificate type:
# - selfsigned: Self signed certificate. Not recommended for p
roduction use.
#
#           Users will see an ERROR when connected to web
page.
# - owncert: Valid certificate purchased in a Internet serv
ices company.
#
#           Please put the certificates files inside folde
r ./owncert
#
#           with names certificate.key and certificate.cer
t
# - letsencrypt: Generate a new certificate using letsencrypt.
Please set the
#
#           required contact email for Let's Encrypt in LE

```



```

TSENCRYPT_EMAIL
#           variable.
CERTIFICATE_TYPE=letsencrypt

# If CERTIFICATE_TYPE=letsencrypt, you need to configure a valid email for notifications
LETSENCRYPT_EMAIL=hyunsu9269@gmail.com
# Proxy configuration
# If you want to change the ports on which openvidu listens, uncomment the following lines

# Allows any request to http://DOMAIN_OR_PUBLIC_IP:HTTP_PORT/ to be automatically
# redirected to https://DOMAIN_OR_PUBLIC_IP:HTTPS_PORT/.
# WARNING: the default port 80 cannot be changed during the first boot
# if you have chosen to deploy with the option CERTIFICATE_TYPE=letsencrypt
HTTP_PORT=8081

# Changes the port of all services exposed by OpenVidu.
# SDKs, REST clients and browsers will have to connect to this port
HTTPS_PORT=8443

```

## ▼ 9 Nginx 서버 최종 conf 설정

### Nginx 서버 conf

/home/ubuntu/nginx/conf.d

```

server {
    listen 80;
    server_name secretzoo.site;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name secretzoo.site;

```

```

ssl_certificate /etc/cert2/live/openvidu.secretzoo.site/fullchain.pem;
ssl_certificate_key /etc/cert2/live/openvidu.secretzoo.site/privatekey.pem;

## 프론트 서버
location / {
    # React 애플리케이션 컨테이너의 이름과 포트
    proxy_pass http://react-app:3000;
    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;
    add_header 'X-SSH-Endpoint' 'secretzoo.site/' always;
}

## 서버 점검 페이지
error_page 403 404 405 406 500 501 502 503 504 505 /index.html;
location =/index.html {
    root /etc/nginx/secretzoo/maintenance/;
    index index.html index.htm;
    try_files $uri $uri/ /index.html?path=$uri&$args;
}
location /images {
    rewrite ^/images$ $1 break;
    root /etc/secretzoo/maintenance/css/image/;
}

## 서버 점검 페이지 직접 접근
location /server {
    rewrite ^/server(/.*)$ $1 break;
    root /etc/nginx/secretzoo/maintenance/;
    index index.html index.htm;
    try_files $uri $uri/ /index.html?path=$uri&$args;
}

## 웹 소켓 서버 통신
location /ws {
    proxy_pass http://react-app:3000;
    # rewrite ^/ws(/.*)$ $1 break;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
}

```

```

        proxy_set_header Origin "";

    }

    ## 웹 소켓 서버
    location /socket.io/ {
        proxy_pass http://node-server:3001;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;

    }

}

## 백엔드 API 서버
server {
    listen 80;
    server_name spring.secretzoo.site;

    return 301 https://$host$request_uri;

}

server {
    listen 443 ssl;
    server_name spring.secretzoo.site;

    ssl_certificate /etc/cert2/live/openvidu.secretzoo.site/fullchain.pem;
    ssl_certificate_key /etc/cert2/live/openvidu.secretzoo.site/private.key;

    location / {
        # Spring 애플리케이션 컨테이너의 이름과 포트
        proxy_pass http://spring-app: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;
    }
}

```

```

        # Required for new HTTP-based CLI
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off; # Required for HTTP-based CLI to work
        add_header 'X-SSH-Endpoint' 'spring.secretzoo.site/' always;
    }
}

## 젠킨스 서버
server {
    listen 80;
    server_name i10a406.p.ssafy.io;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name i10a406.p.ssafy.io;

    ssl_certificate /etc/cert/cert.pem; # SSL 인증서 파일
    ssl_certificate_key /etc/cert/privkey.pem; # SSL 키 파일
    ssl_trusted_certificate /etc/cert/chain.pem;

    location / {

        proxy_pass http://jenkins:8080;
        # proxy_redirect http://localhost:8080 https://$host;
        proxy_set_header Host $host:$server_port;
        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;

        # Required for new HTTP-based CLI
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off; # Required for HTTP-based CLI to
        add_header 'X-SSH-Endpoint' 'i10a406.p.ssafy.io/' always;
    }
}

```

```

## openvidu 시그널링 서버
server {
    listen 80;
    server_name openvidu.secretzoo.site;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name openvidu.secretzoo.site;

    ssl_certificate /etc/cert2/live/openvidu.secretzoo.site/fullchain.pem;
    ssl_certificate_key /etc/cert2/live/openvidu.secretzoo.site/privatekey.pem;

    location / {
        proxy_pass http://openvidu-app:5442;
        proxy_set_header Host $host:$server_port;
        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;

        # Required for new HTTP-based CLI
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off; # Required for HTTP-based CLI to work
        add_header 'X-SSH-Endpoint' 'openvidu.secretzoo.site/' always;
    }
}

```

## ▼ 10 CI/CD

### (1) Gitlab Webbook

🔑 GitLab Credential 등록 (Username with password)

Jenkins

검색 (CTRL+K)

admin

로그아웃

Dashboard > Jenkins 관리

새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중

2 대기 중

Jenkins 관리

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#).

Set up agent

Set up cloud

Dismiss

현재 설치된 다음 구성 요소에 대한 경고가 게시되었습니다.

플러그인 메커니즘으로 이동

다음 경고 중 표시할 경고 구성

SSH plugin 2.6.1

CSRF vulnerability and missing permission checks allow capturing credentials (no fix available)

Missing permission check allows enumerating credentials IDs (no fix available)

No fixes for these issues are available. It is recommended that you review the security advisory and apply mitigations if possible, or uninstall this plugin.

System Configuration

System

환경 변수 및 경로 정보들을 설정합니다.

Tools

Configure tools, their locations and automatic installers.

Plugins

Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.

Nodes and Clouds

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Managed files

e.g. settings.xml for maven, central managed scripts, custom files, ...

Security

Security

Secure Jenkins; define who is allowed to access/use the system.

Manage Credentials

Configure credentials

Configure Credential Providers

Configure the credential providers and types

Jenkins

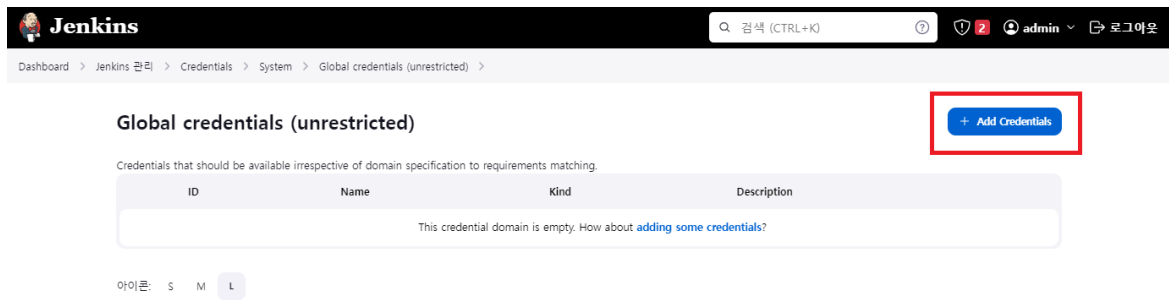
검색 (CTRL+K)

Dashboard > Jenkins 관리 > Credentials

Credentials

T	P	Store ↓	Domain	ID	Name
Stores scoped to Jenkins					
	P	Store ↓	Domains		
		System	global		
			Add credentials		

아이콘: S M L



- Kind : Username with password 선택
- Username : Gitlab 계정 아이디 입력
- Password : Gitlab 계정 비밀번호 입력 (토큰 발행시, API 토큰 입력)
- ID : Credential에 대한 별칭

## GitLab API token 등록

**New credentials**

Kind: GitLab API token

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

API token: .....

ID: gitlab-hyuns

Description:

Create

REST API Jenkins 2.426.3

## GitLab과 연결

**GitLab**

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name: fiveguys

GitLab host URL: https://lab.ssafy.com

Credentials: GitLab API token

+ Add

고급

Success

Test Connection

## 📌 Gitlab Webhook 등록

Gitlab에 특정 브랜치에 merge request / push 가 된 경우 Webhook을 통해 빌드 후 배포 프로세스 자동화

- URL : Jenkins의 Item URL 입력 (양식 : `http://[Jenkins Host]:[Jenkins Port]/project/[파인 프라이빗 아이템명]` )
- Secret token : Jenkins의 Gitlab trigger 고급 설정 중 Secret token Generate 을 이용해 만든 토큰 입력



- Trigger : Push events 체크, merge request / push 이벤트를 감지해서 젠킨스 파이프라인으로 빌드 배포 프로세스가 자동하게 할 브랜치 입력

## dev/backend 브랜치

### Webhook

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  
https://10a406.p.ssafy.io/project/spring

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

☒ Show full URL

☐ Mask portions of URL

Do not show sensitive data such as tokens in the UI.

### Secret token

.....

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

### Trigger

☒ Push events

☐ All branches

☐ Wildcard pattern

☒ Regular expression

dev/backend

Regular expressions such as `^(feature|hotfix)/` are supported.

## dev/frontend 브랜치

### Webhook

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  
https://10a406.p.ssafy.io/project/react

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

☒ Show full URL

☐ Mask portions of URL

Do not show sensitive data such as tokens in the UI.

### Secret token

.....

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

### Trigger

☒ Push events

☐ All branches

☐ Wildcard pattern

☒ Regular expression

dev/frontend

Regular expressions such as `^(feature|hotfix)/` are supported.

## (2) Jenkins pipeline

spring / react 동일한 파이프라인

### 개요

- 1) git clone - gitlab push 이벤트 발생하면 jenkins workspace로 git clone
- 2) spring → jar 파일 빌드 / react → npm install 후 빌드

- 3) 실행되고 있는 컨테이너 stop - remove
- 4) 도커 이미지 삭제
- 5) docker-compose up 명령어로 컨테이너 실행
- 6) nginx 서버 reload

## spring

```
pipeline {
    agent any
    environment {
        imageName = "secretzoo/backend-develop"
        registryCredential = 'hyunsoo4885'
        dockerImage = ''

        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'i10a406.p.ssafy.io'
        releasePort = '8080'
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: 'dev/backend',
                    credentialsId: 'hyun',
                    url: 'https://lab.ssafy.com/s10-webmobile1-sub2/S10P12A406'
            }
            post {
                success {
                    echo 'Successfully Cloned Repository'
                }
                failure {
                    error 'Git Clone is failed'
                }
            }
        }

        stage('Jar Build') {
            steps {
                dir ('backend') {
                    sh 'chmod +x ./gradlew'
                }
            }
        }
    }
}
```

```

        sh './gradlew clean bootJar'
    }
}
post {
    success {
        echo 'Successfully Jar build'
    }
    failure {
        error 'Jar build is failed'
    }
}
}

stage('Stop and remove Docker Container') {
    steps {

        script {
            // 도커 컨테이너 이름 지정
            def containerName = "spring-app"

            // 도커 컨테이너 중지 명령어
            def dockerStopCmd = "docker stop ${containerName}"

            // 도커 컨테이너 중지
            sh """
                ${dockerStopCmd}
            """

            // 도커 컨테이너 삭제 명령어
            def dockerRmCmd = "docker rm ${containerName}"

            // 도커 컨테이너 삭제
            sh """
                ${dockerRmCmd}
            """
        }

        post {
            success {

```

```

        echo 'Successfully Stopped and removed Docker Container'
    }
    failure {
        error 'Failed to Stop Docker Container'
    }
}
}

```

```

stage('Delete Docker Image') {
    steps {
        script {
            // 도커 이미지 이름 및 태그 지정
            def dockerImageName = "backend-spring-app "

            // 도커 이미지 삭제 명령어
            def dockerRmiCmd = "docker rmi ${dockerImageName}"

            // 도커 이미지 삭제
            sh """
                ${dockerRmiCmd}
            """
        }
    }
    post {
        success {
            echo 'Successfully Deleted Docker Image'
        }
        failure {
            error 'Failed to Delete Docker Image'
        }
    }
}
}

```

```

stage('Build and Run Docker Compose') {
    steps {
        dir ('backend'){
            script {

```

```

// 도커 컴포즈 파일 경로 지정
def dockerComposeFile = 'docker-compos

e.yml'

// 도커 컴포즈 실행 명령어
def dockerComposeCmd = " docker-compose

up -d"

// 도커 컴포즈 실행
sh """
    ${dockerComposeCmd}
    """
}
}
}

stage('Nginx reload') {
    steps{
        script {
            def containerName = "ubuntu-nginx-1"

            // 도커 컨테이너 내에서 Nginx 다시 로드
            sh "docker exec ${containerName} /bin/sh -c
'nginx -s reload'"

        }
    }
    post {
        success {
            echo 'Successfully nginx reloaded'
        }
        failure {
            error 'Failed to reload nginx'
        }
    }
}
}
}
}

```

**react**

```

pipeline {
    agent any

    stages {
        stage('Git Clone') {
            steps {
                git branch: 'dev/frontend',
                    credentialsId: 'hyun',
                    url: 'https://lab.ssafy.com/s10-webmobile1-sub
            }
            post {
                success {
                    echo 'Successfully Cloned Repository'
                }
                failure {
                    error 'Git Clone is failed'
                }
            }
        }
    }
}

```

```

stage('Stop Docker Container') {
    steps {
        script {
            // 도커 컨테이너 이름 지정
            def containerName1 = "react-app"
            def containerName2 = "node-server"

            // 도커 컨테이너 중지 명령어
            def dockerStopCmd1 = "docker stop ${containerName1}"
            def dockerStopCmd2 = "docker stop ${containerName2}"

            // 도커 컨테이너 중지
            sh """
                ${dockerStopCmd1}
            """
            sh """
                ${dockerStopCmd2}
            """
        }
    }
}

```

```

    }
    post {
        success {
            echo 'Successfully Stopped Docker Container'
        }
        failure {
            error 'Failed to Stop Docker Container'
        }
    }
}

stage('Remove Docker Container') {
    steps {
        script {
            // 도커 컨테이너 이름 지정
            def containerName1 = "react-app"
            def containerName2 = "node-server"

            // 도커 컨테이너 삭제 명령어
            def dockerRmCmd1 = "docker rm ${containerName1}"
            def dockerRmCmd2 = "docker rm ${containerName2}"
            // 도커 컨테이너 삭제
            sh """
                ${dockerRmCmd1}
            """
            sh """
                ${dockerRmCmd2}
            """
        }
    }
    post {
        success {
            echo 'Successfully Removed Docker Container'
        }
        failure {
            error 'Failed to Remove Docker Container'
        }
    }
}

stage('Delete Docker Image') {
    steps {
        script {

```

```

// 도커 이미지 이름 및 태그 지정
def dockerImageName1 = "frontend-react-app"
def dockerImageName2 = "frontend-node-server"

// 도커 이미지 삭제 명령어
def dockerRmiCmd1 = "docker rmi ${dockerImageName1}"
def dockerRmiCmd2 = "docker rmi ${dockerImageName2}"

// 도커 이미지 삭제
sh """
    ${dockerRmiCmd1}
"""
sh """
    ${dockerRmiCmd2}
"""
    }
}
post {
    success {
        echo 'Successfully Deleted Docker Image'
    }
    failure {
        error 'Failed to Delete Docker Image'
    }
}
}
}

```

```

stage('Build and Run Docker Compose') {
    steps {
        dir ('frontend/'){
            script {
                // 도커 컴포즈 파일 경로 지정
                def dockerComposeFile = 'docker-compose.yml'

                // 도커 컴포즈 실행 명령어
                def dockerComposeCmd = " docker-compose up"

                // 도커 컴포즈 실행
                sh """
                    ${dockerComposeCmd}
                """
            }
        }
    }
}

```





