



# 포팅 메뉴얼

[키즈링크 개발환경](#)

[포트번호](#)

[EC2 서버 기본설정](#)

[Certbot으로SSL/TLS 인증서를 발급받기](#)

1. Certbot 설치 (sudo 권한있는 ubuntu에서 실행)
2. Certbot을 사용하여 SSL/TLS 인증서 발급받기
2. EC2에 인증서 minsun계정 하위에 복사.

[수동배포](#)

[배포 기본 원리](#)

[Frontend, Backend 배포 원리](#)

[Docker와 Docker-compose](#)

[EC2에 Docker, Docker-compose 설치](#)

[Docker](#)

[Docker-compose](#)

[Nginx.conf](#)

[첫 번째 서버 블록 \(HTTP → HTTPS 리디렉션\)](#)

[두 번째 서버 블록 \(HTTPS\)](#)

[Openvidu](#)

[docker-compose.yml](#)

[Openvidu-nginx.conf](#)

[AI 서버\( Face-Recognition, 수동배포\)](#)

[Dockerfile](#)

[자동배포 jenkins CI/CD](#)

[Jenkins 동작원리](#)

[Jenkins 초기 설정\(ubuntu계정\)](#)

[Jenkins Job 구성](#)

[GitLab Webhook 구성](#)

[Jenkinsfile](#)

## 키즈링크 개발환경

<http://52.78.118.241:8081/>

jenkins 서버

EC2 서버 정보

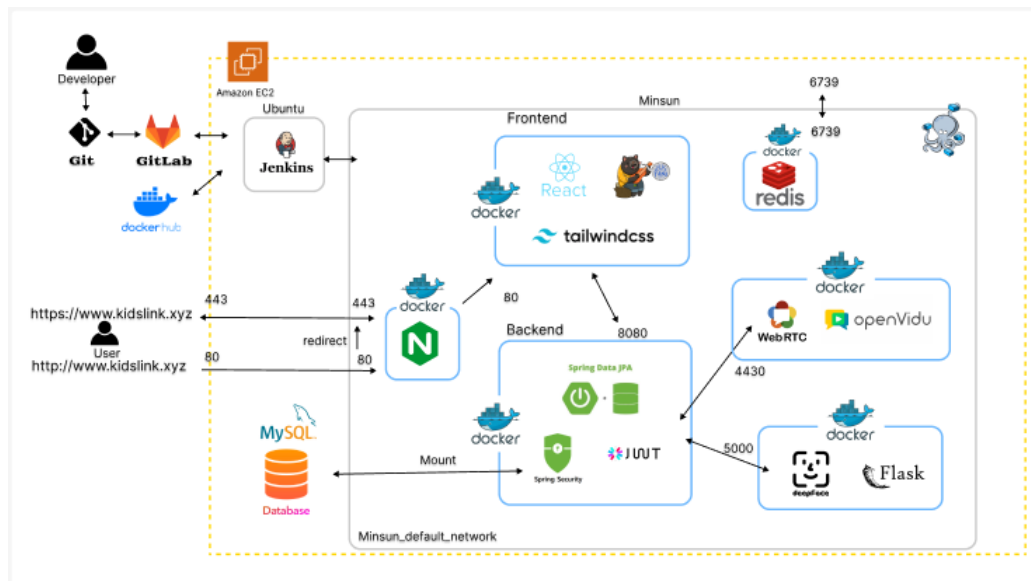
ec2서버 공인 ip 주소: 52.78.118.241  
내부 ip주소: 172.26.15.83

서버민선계정: minsun  
비밀번호: alstjs123

서버ubuntu계정: ubuntu  
비밀번호: alstjs123

서버 mysql 민선계정: minsun  
비밀번호 : alstjs123

서버 mysql root 계정: root  
비밀번호 : ssafy



## 포트번호

### ▼ 포트

**kidslink:** 443, 80

**nginx:** 443, 80

**backend:** 8080

**frontend:** 80

**ec2 mysql:** 3306

**redis:** 6378

**flask:** 5000

**젠킨스:** 8081

### Openvidu

- **22 TCP:** SSH to admin OpenVidu.
- **80 TCP:** SSL certificate this port
- **443 TCP:** OpenVidu server https port.
- **3478 TCP+UDP:** used by STUN/TURN server to resolve clients IPs.
- **40000 - 57000 TCP+UDP:** used by Kurento Media Server.
- **57001 - 65535 TCP+UDP:** used by TURN server.
- **5442 TCP: openvidu server**
- **5443 TCP: openvidu call server**
- **6379 TCP:** openvidu redis
- **8888 TCP:**

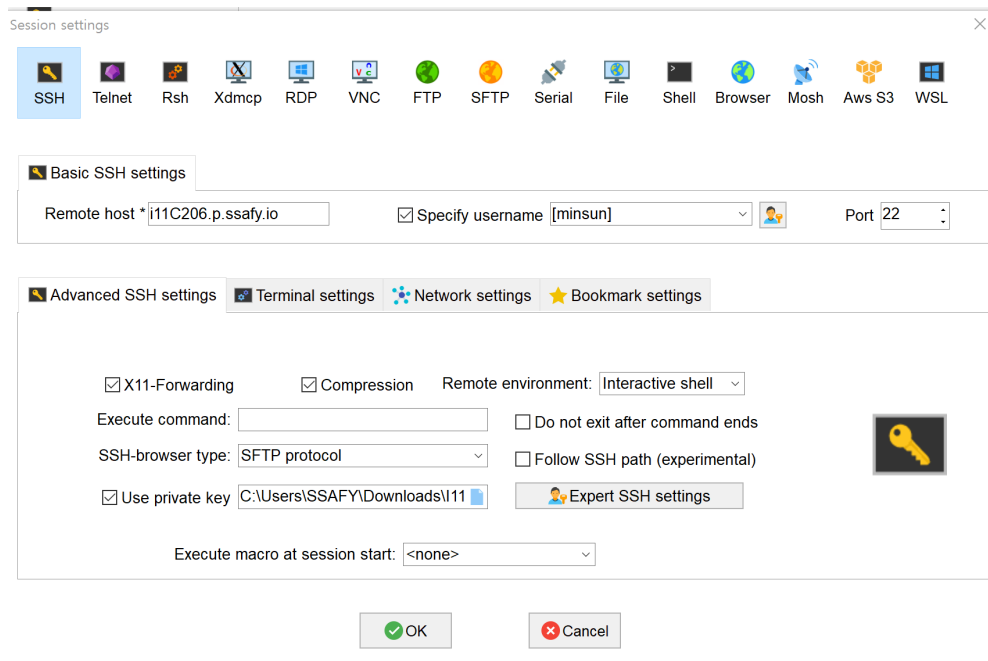
To	Action	From
--	----	----
22	ALLOW	Anywhere
8989	ALLOW	Anywhere
443	ALLOW	Anywhere
8080	ALLOW	Anywhere
3306	ALLOW	Anywhere
80/tcp	ALLOW	Anywhere
8081	ALLOW	Anywhere
4443/tcp	ALLOW	Anywhere
800	ALLOW	Anywhere
4430	ALLOW	Anywhere
5442	ALLOW	Anywhere
5443	ALLOW	Anywhere
3478	ALLOW	Anywhere
5439	ALLOW	Anywhere
8888	ALLOW	Anywhere
6378	ALLOW	Anywhere
5000	ALLOW	Anywhere
22 (v6)	ALLOW	Anywhere (v6)
8989 (v6)	ALLOW	Anywhere (v6)
443 (v6)	ALLOW	Anywhere (v6)
8080 (v6)	ALLOW	Anywhere (v6)
3306 (v6)	ALLOW	Anywhere (v6)
80/tcp (v6)	ALLOW	Anywhere (v6)
8081 (v6)	ALLOW	Anywhere (v6)
4443/tcp (v6)	ALLOW	Anywhere (v6)
800 (v6)	ALLOW	Anywhere (v6)
4430 (v6)	ALLOW	Anywhere (v6)
5442 (v6)	ALLOW	Anywhere (v6)
5443 (v6)	ALLOW	Anywhere (v6)
3478 (v6)	ALLOW	Anywhere (v6)
5439 (v6)	ALLOW	Anywhere (v6)
8888 (v6)	ALLOW	Anywhere (v6)
6378 (v6)	ALLOW	Anywhere (v6)
5000 (v6)	ALLOW	Anywhere (v6)

## EC2 서버 기본설정

EC2 접속

✓ MobaXterm 사용 (추천!)

- <https://mobaxterm.mobatek.net/download.html> 에서 MobaXterm 다운
- Session 생성



👉 Use private key에는 싸피에서 받은 ec2키파일 넣으면 됩니다.

👉 기본 사용자는 ubuntu. minsun계정은 내가 임의로 생성.

`sudo adduser minsun`

💡 "No supported authentication methods available (server sent: publickey)"

→ minsun 계정 새로 생기면 생기는 오류다. ubuntu의 `authorized_keys` 가 minsun에는 없어서 생기는 문제다.

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
nano ~/.ssh/authorized_keys
nano 편집기가 열리면, ubuntu 사용자에서 복사한 공개 키를 붙여넣는다.
chmod 600 ~/.ssh/authorized_keys
```

(sudo 권한있는 ubuntu에서 실행)



## EC2 환경 세팅

### ✓ 패키지 관리자 업데이트

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

### ✓ JDK 설치

- `sudo apt-get install openjdk-8-jdk` #11버전의 경우 8->11
- `java -version`

📌 우리는 java 17 version으로 설치했음



## EC2 환경 세팅



### ✓ DB 설치

- `sudo apt-get install mysql-server`
- Mysql 접속 : `sudo mysql -uroot`
- 유저 생성 : `create user 'id'@'%' identified by '비밀번호';`
- 권한 설정 : `grant all privileges on *.* to 'id'@'%';`
- 외부 접속 허용 : `cd /etc/mysql/mysql.conf.d/`

`sudo vi mysql.cnf`

bind-address를 0.0.0.0으로 변경

```
bind-address            = 0.0.0.0
mysqlx-bind-address     = 127.0.0.1
```

📌 Mysql

userId : minsun

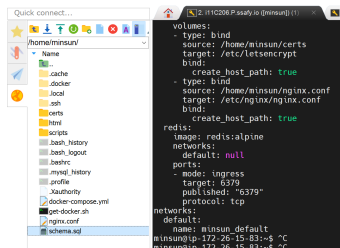
password : alstjs123

```
GRANT ALL PRIVILEGES ON kidslink.* TO 'minsun'@'%';
FLUSH PRIVILEGES;
```

👉 mysql root 사용자가 minsun 사용자에게 CRUD 권한부여.

📌 우리는 mysql은 docker에 안올리고 EC2의 mysql을 사용한다.

이유는 매번 CI/CD마다 스키마를 생성하고 다 날라가기때문에 CICD와 상관없는 mysql을 구성했다.



schema.sql을 ec2서버에 옮겨놓고 실행.

```
mysql -u minsun -p
CREATE DATABASE kidslink;
USE kidslink;
source ~/schema.sql;
```

## Certbot으로SSL/TLS 인증서를 발급받기

### 1. Certbot 설치 (sudo 권한있는 ubuntu에서 실행)

Ubuntu에 Certbot 설치

```
sudo apt-get update
sudo apt-get install -y certbot
```

### 2. Certbot을 사용하여 SSL/TLS 인증서 발급받기

Certbot을 사용하여 SSL 인증서 발급

```
sudo certbot certonly --standalone -d kidslink.xyz -d www.kidslink.xyz
```

명령어를 실행하면 Certbot이 자동으로 HTTP 서버를 시작하여 도메인 소유권을 확인한 후, SSL 인증서를 발급받습니다. 인증서가 성공적으로 발급되면 다음 경로에 저장됩니다:

- 인증서: `/etc/letsencrypt/live/kidslink.xyz/fullchain.pem`
- 개인 키: `/etc/letsencrypt/live/kidslink.xyz/privkey.pem` ``

📌여기서 kidslink.xyz라는 도메인은 가비아에서 도메인 구입했음. 도메인이 있어야 certbot인증서 발급가능! ip로는 발급안됨.

### 2. EC2에 인증서 minsun계정 하위에 복사.

```
sudo mkdir -p /home/minsun/certs/live/kidslink.xyz
sudo cp /etc/letsencrypt/live/kidslink.xyz/fullchain.pem /home/minsun/certs/live/kidslink.xyz/
sudo cp /etc/letsencrypt/live/kidslink.xyz/privkey.pem /home/minsun/certs/live/kidslink.xyz/
```

📌이후 이 인증서는 docker에서 접근해서 사용할 예정.

## 수동배포

### 배포 기본 원리

#### Frontend, Backend 배포 원리

## Frontend 배포

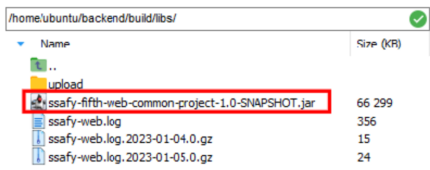
- Frontend build 결과물을 서버에 넣기
  - `cd front`
  - `npm run build`



☞ Frontend는 npm run build 하면 dist폴더 생성된다. 해당 dist 폴더를 배포하는 것.

## Backend 배포

- Backend build 결과물 서버에 넣기
  - `cd Backend`
  - `./gradlew build`



☞ `./gradlew clean build -x test` 이걸로 빌드하는게 좋다.

Backend는 .jar파일 생성. 해당 jar파일 배포하는것.

## Docker와 Docker-compose

### EC2에 Docker, Docker-compose 설치

Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

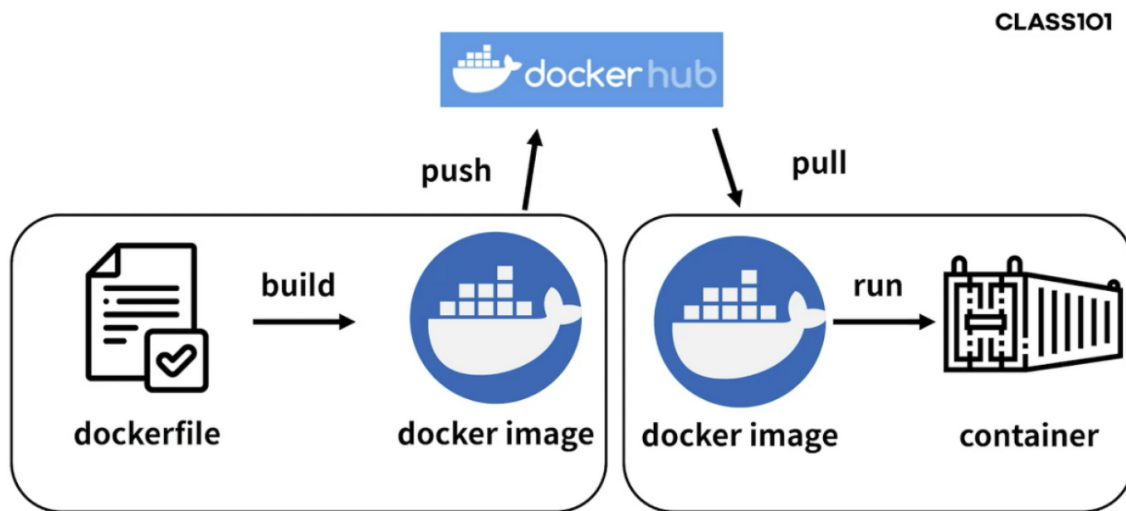
Docker-compose

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
docker-compose --version
```

```
sudo usermod -aG docker minsun
newgrp docker
```

🔥 docker 그룹에 minsun추가해줘서 sudo없이 사용

## Docker



<https://class101.net/products/5fc4a3b4fc231b000d85661b>

1. 로컬에서 dockerfile로 프로젝트를 빌드하면 도커 이미지로 변환한다.

```
docker build --no-cache -t {Docker-hub-Id}/이미지명 .
docker build --no-cache -t sangmin0806/frontend .
docker build --no-cache -t sangmin0806/backend .
```

2. Docker Hub 에 이미지를 push하면, EC2서버에서 pull하여 이미지를 내려받을 수 있다.

```
docker push {Docker-hub-Id}/이미지명
docker push sangmin0806/frontend
docker push sangmin0806/backend
```

3. 이후 이미지를 서버에서 run 하여 컨테이너 형태로 실행한다.(우리 이미지 run은 docker-compose가 할꺼다.)

## BackEnd Dockerfile

```
FROM openjdk:17-jdk-buster
# ffmpeg 설치
RUN apt-get update && apt-get install -y ffmpeg
```



```
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
COPY src/main/resources/static/profiles /app/src/main/resources/static/profiles
ENTRYPOINT ["java", "-Dspring.profiles.active=prod", "-jar", "app.jar"]
```

👉 빌드하는 jar파일을 이미지화 시킨다는 뜻

## FrontEnd Dockerfile

```
# 1단계: 빌드 환경 설정
FROM node:18 AS build

WORKDIR /app

# 앱 소스 코드 복사
COPY package*.json ./
COPY . .

# 종속성 설치 및 빌드
RUN npm install
RUN npm run build

# 2단계: 실행 환경 설정
FROM nginx:alpine

# 빌드된 파일 복사
COPY --from=build /app/dist /usr/share/nginx/html

# Nginx 실행
CMD ["nginx", "-g", "daemon off;"]

# 포트 노출
EXPOSE 80
EXPOSE 443
```

👉 1단계에서 front 소스코드 build, 2단계에서는 1단계에서 만든 build산출물인 dist파일을 nginx/html로 복사 후 실행.

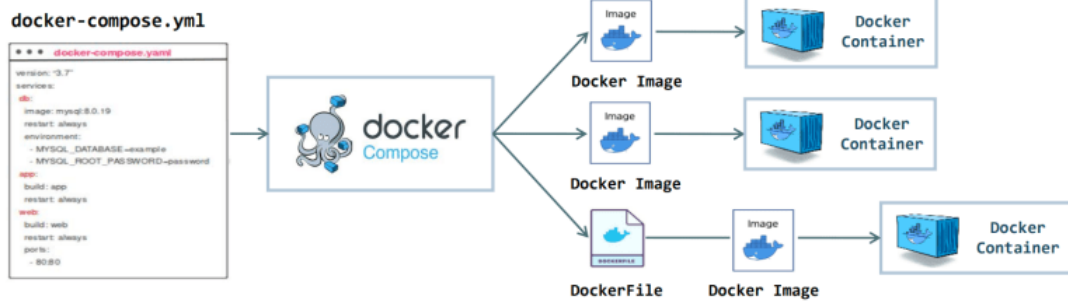
80포트는 http포트, 443은 https 포트 열어준다.

즉, 프론트는 프론트 이미지, nginx이미지 두가지를 빌드하는 셈

📌 각 Dockerfile은 로컬 프로젝트의 루트디렉토리에 위치시킨다.

로컬의 Docker Demon(도커엔진)이 Dockerfile을 찾아서 실행시킨다.

## Docker-compose



☞서버의 여러개의 도커이미지들을 한번에 실행.

이미지간 통신설정, 연관관계, 실행방식등을 docker-compose.yml에 작성

📌docker-compose.yml은 이미지들의 실행환경에 위치해야 한다. 즉, EC2 서버에 위치시킨다.

docker-compose.yml

```

version: '3.8'

services:
  frontend:
    image: sangmin0806/frontend
    env_file:
      - /home/minsun/.env.production
    volumes:
      - /home/minsun/certs:/etc/letsencrypt
    depends_on:
      - backend

  backend:
    image: sangmin0806/backend
    env_file:
      - .env
    volumes:
      - ./keystore.p12:/app/keystore.p12
      - /opt/openvidu/recordings:/opt/openvidu/recordings
    depends_on:
      - redis
    ports:
      - "8080:8080"

  redis:
    image: redis:alpine
    ports:
      - "6378:6379"

  flask_backend:
    image: sangmin0806/flask-backend
    ports:
      - "5000:5000"
    depends_on:
      - redis
  
```

```

nginx:
  image: sangmin0806/frontend
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - /home/minsun/certs:/etc/letsencrypt
    - ./nginx.conf:/etc/nginx/nginx.conf
  depends_on:
    - frontend
    - backend
    - flask_backend

```

- **frontend:**

- **image:** `sangmin0806/frontend` - 프론트엔드 애플리케이션을 실행하기 위한 도커 이미지입니다.
- **volumes:** `/home/minsun/certs:/etc/letsencrypt` - 호스트의 `/home/minsun/certs` 디렉토리를 컨테이너의 `/etc/letsencrypt` 디렉토리로 마운트합니다.  
https 인증을 처리하기 위해 ec2서버의 인증서를 가져온다.
- **depends\_on:** `backend` - `backend` 서비스가 시작된 후에 이 서비스가 시작됩니다.
- `.env`파일에서 백엔드 uri 변수 지정한다.

```

VITE_API_KEY="https://www.kidslink.xyz/api"
VITE_API="https://www.kidslink.xyz"
VITE_WEBSOCKET_URL="wss://www.kidslink.xyz/ws/bus"
VITE_KAKAO_API_KEY="137497cddc91726fbd7601efe1bb0072"
VITE_OPENVIDU_SECRET="MY_SECRET"
VITE_OPENVIDU_URL="https://www.kidslink.xyz/api/video"
VITE_PUBLIC_URL="https://www.kidslink.xyz"

```

- **backend:**

- **image:** `sangmin0806/backend` - 백엔드 애플리케이션을 실행하기 위한 도커 이미지입니다.
- **ports:** `"8080:8080"` - 호스트의 포트 8080을 컨테이너의 포트 8080에 매핑합니다.
- **environment:** 여러 환경 변수를 설정하여 데이터베이스와 Redis 서버에 연결합니다.  
Spring application.properties에서 해당 변수들 사용된다.
  - `SPRING_DATASOURCE_URL`: MySQL 데이터베이스 URL
  - `SPRING_DATASOURCE_USERNAME`: 데이터베이스 사용자 이름
  - `SPRING_DATASOURCE_PASSWORD`: 데이터베이스 비밀번호
  - `SPRING_DATA_REDIS_HOST`: Redis 호스트
  - `SPRING_DATA_REDIS_PORT`: Redis 포트
  - `FRONTEND_URL`: `http://frontend:80`
- **depends\_on:** `redis` - `redis` 서비스가 시작된 후에 이 서비스가 시작됩니다.
- **backend .env파일**

```

SPRING_PROFILES_ACTIVE=prod
SPRING_DATASOURCE_URL=jdbc:mysql://52.78.118.241:3306/kidslink?useSSL=false&useUni
SPRING_DATASOURCE_USERNAME=minsun
SPRING_DATASOURCE_PASSWORD=alstjs123
SPRING_DATA_REDIS_HOST=redis
SPRING_DATA_REDIS_PORT=6379

```

```

CLOUD_AWS_S3_BUCKET=kidslink-s3-c206
AWS_REGION=ap-northeast-2
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
FRONTEND_URL=https://www.kidslink.xyz
USE_S3=true
SSL_KEY_STORE_PASSWORD=alstjs123
OAUTH2_NAVER_CLIENT_NAME=naver
OAUTH2_NAVER_CLIENT_ID=BeugClrniZApZo9fKSNQ
OAUTH2_NAVER_CLIENT_SECRET=jCYFckmDn7
OAUTH2_NAVER_REDIRECT_URI=https://www.kidslink.xyz/login/oauth2/code/naver
OAUTH2_GOOGLE_CLIENT_NAME=google
OAUTH2_GOOGLE_CLIENT_ID=433862441512-iojcnlbnk2l1l164tjal9dlesjetcecm3.apps.googleu
OAUTH2_GOOGLE_CLIENT_SECRET=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
OAUTH2_GOOGLE_REDIRECT_URI=https://www.kidslink.xyz/login/oauth2/code/google
OAUTH2_KAKAO_CLIENT_NAME=kakao
OAUTH2_KAKAO_CLIENT_ID=c21638cffbecf8108bda87b1a1ac84cc
OAUTH2_KAKAO_CLIENT_SECRET=HWpKvOE640wfZpe2NAyA7XuDCiK9Ynfw
OAUTH2_KAKAO_REDIRECT_URI=https://www.kidslink.xyz/login/oauth2/code/kakao

```

- application-prod.properties

```

ai.server.url=http://flask_backend:5000
frontend.server.url=https://www.kidslink.xyz
server.url=https://www.kidslink.xyz:8080/api
file.profile-dir=/app/src/main/resources/static/profiles

server.port=8080
server.ssl.enabled=true
server.ssl.key-store=/app/keystore.p12
server.ssl.key-store-password=${SSL_KEY_STORE_PASSWORD}
server.ssl.keyStoreType=PKCS12
server.ssl.keyAlias=tomcat

spring.security.oauth2.client.registration.naver.redirect-uri=${OAUTH2_NAVER_REDIRECT
spring.security.oauth2.client.registration.google.redirect-uri=${OAUTH2_GOOGLE_REDIR
spring.security.oauth2.client.registration.kakao.redirect-uri=${OAUTH2_KAKAO_REDIRECT

openvidu.url=https://www.kidslink.xyz:4430
openvidu.secret=MY_SECRET

```

- redis:

- **image:** `redis:alpine` - Redis 서버를 실행하기 위한 도커 이미지입니다. ( Docker Hub에서 제공하는 공식 이미지를 사용)
- **ports:** `"6379:6379"` - 호스트의 포트 6379를 컨테이너의 포트 6379에 매핑합니다.

- nginx:

- **image:** `sangmin0806/frontend` - 프론트(nginx)를 실행하기 위한 도커 이미지입니다 (프론트에서 nginx까지 빌드했기때문에 frontend의 이미지를 사용한다.).
- **ports:**
  - `"80:80"` - 호스트의 포트 80을 컨테이너의 포트 80에 매핑합니다.
  - `"443:443"` - 호스트의 포트 443을 컨테이너의 포트 443에 매핑합니다.

- **volumes:**
  - `/home/minsun/certs:/etc/letsencrypt` - 호스트의 `/home/minsun/certs` 디렉토리를 컨테이너의 `/etc/letsencrypt` 디렉토리로 마운트합니다.
  - `./nginx.conf:/etc/nginx/nginx.conf` - 현재 디렉토리의 `nginx.conf` 파일을 컨테이너의 `/etc/nginx/nginx.conf` 파일로 마운트합니다.
- **depends\_on:** `frontend` - `frontend` 서비스가 시작된 후에 이 서비스가 시작됩니다.

## Nginx.conf

```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log debug;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

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

    access_log /var/log/nginx/access.log combined;

    gzip on;
    client_max_body_size 200M;

    server {
        listen 80;
        server_name www.kidslink.xyz;

        location / {
            return 301 https://$host$request_uri;
        }
    }

    server {
        listen 443 ssl;
        server_name www.kidslink.xyz;

        ssl_certificate /etc/letsencrypt/live/www.kidslink.xyz/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/www.kidslink.xyz/privkey.pem;

        location /api/ {
            proxy_pass https://minsun-backend-1:8080/api/;
            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;
        }

        location /oauth2/authorization/ {
```

```

        proxy_pass https://minsun-backend-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;
    }

    location /login/oauth2/ {
        proxy_pass https://minsun-backend-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;
    }

    location /ws/ {
        proxy_pass https://minsun-backend-1:8080/ws/;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location / {
        root /usr/share/nginx/html;
        try_files $uri $uri/ /index.html;
        proxy_pass http://frontend:80;
        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;
    }

}

}

```

### 첫 번째 서버 블록 (HTTP -> HTTPS 리디렉션)

- 목적: HTTP로 들어오는 요청을 HTTPS로 리디렉션.
- `listen 80;`: 포트 80에서 HTTP 요청 수신.
- `server_name www.kidslink.xyz;`: 서버 이름 설정.
- `location /`: 모든 요청을 받아서 HTTPS로 리디렉션.
  - `return 301 https://$host$request_uri;`: 301 Moved Permanently 응답을 보내면서 요청된 URL을 HTTPS로 리디렉션.

📌 `http://www.kidslink.xyz`로 들어온 모든 요청을 `https://www.kidslink.xyz`로 리다이렉트

### 두 번째 서버 블록 (HTTPS)

- 목적: HTTPS 요청 처리 및 프록시 설정.

- `listen 443 ssl;` : 포트 443에서 HTTPS 요청 수신.
- `server_name www.kidslink.xyz;` : 서버 이름 설정.
- `ssl_certificate` 및 `ssl_certificate_key` : SSL 인증서와 개인 키 파일의 경로 설정.
- `location = /sw.js` 및 `location = /vite.svg` `location /assets/` : 특정 파일 요청을 처리.
  - `root /usr/share/nginx/html;` : 해당 파일의 루트 디렉토리 설정.  
👉 위치 명시안하면 무한해당 경로에서 리다이렉트 무한루프에 빠짐.(이유는 모르겠음)
  - `try_files $uri =404;` : 파일이 없으면 404 오류 반환.
- `location /` : 모든 기타 요청을 프록시로 전달.
  - `proxy_pass http://frontend:80;` : 프록시 대상으로 `frontend` 컨테이너의 포트 80 설정.
  - `proxy_set_header` 지시어들: 프록시 요청 시 추가 헤더 설정.
    - `Host $host` : 원래의 호스트 헤더를 전달.
    - `X-Real-IP $remote_addr` : 클라이언트의 실제 IP 주소를 전달.
    - `X-Forwarded-For $proxy_add_x_forwarded_for` : 원래의 클라이언트 IP 주소를 포함한 X-Forwarded-For 헤더 추가.
    - `X-Forwarded-Proto $scheme` : 요청의 프로토콜(HTTP 또는 HTTPS)을 전달.

## Openvidu

### docker-compose.yml

```
version: '3.1'

services:

  openvidu-server:
    image: openvidu/openvidu-server:2.30.0
    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}
    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
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:2.30.0
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:2.30.0
restart: always
network_mode: host
volumes:
  - /etc/letsencrypt/live:/etc/letsencrypt/live
  - /etc/letsencrypt/archive:/etc/letsencrypt/archive
  - /etc/letsencrypt/renewal:/etc/letsencrypt/renewal
  - ./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-nginx.conf

```

server {
    listen 800;
    server_name www.kidslink.xyz;

    location / {
        return 301 http://$host:5443$request_uri;
    }
}

server {
    listen 4430 ssl;
    server_name www.kidslink.xyz;

    ssl_certificate /etc/letsencrypt/live/www.kidslink.xyz/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/www.kidslink.xyz/privkey.pem;
    location / {
        proxy_pass http://localhost:5443;
        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;
    }
}

```

👉 openvidu 서버 포트를 800, 4430으로 Open

## AI 서버( Face-Recognition, 수동배포)

### Dockerfile

```
FROM python:3.10-slim
```

```

WORKDIR /app

RUN apt-get update && apt-get install -y \
    cmake \
    g++ \
    libgl1-mesa-glx \
    libglib2.0-0 && \
    apt-get clean

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .


CMD ["python", "app.py"]

```


🔍여기까지가 수동배포. 이 과정을 자동화하는게 jenkins.

## 자동배포 jenkins CI/CD

### Jenkins 동작원리


**S11P12C206**

develop
S11P12C206 /
+
History
Find file
Edit
Code


**Merge branch 'feature/S11P12C206-341-ai-post' into 'develop'**
김범수 authored 43 minutes ago
436889dc

Name	Last commit	Last update
backend	feat: AI 서버 POST 요청 후 앨범 분...	44 minutes ago
docs	project init	1 week ago
exec	feat: s3 버킷 연결 #S11P12C206-...	1 day ago
frontend	feat: 알림장 등록 API 연동 #S11P1...	7 hours ago
.gitignore	feat: DEV환경 개발 시 In-Memory...	1 hour ago
Jenkinsfile	jenkinsfile 수정	7 hours ago

1. GitLab에 (push,merge)등의 요청이 오면 GitLab webhook 발동.
2. Jenkins 서버가 webhook을 확인시 프로젝트 루트의 Jenkinsfile 읽음.
3. Jenkinsfile 내용을 바탕으로 CI/CD 수행

### Jenkins 초기 설정(ubuntu계정)

```

docker pull jenkins/jenkins:lts
mkdir -p /var/jenkins_home

```

```
sudo docker run -d \
  --name jenkins \
  -p 8081:8080 -p 50000:50000 \
  -v /var/jenkins_home:/var/jenkins_home \
  jenkins/jenkins:lts
```

📌 jenkins를 docker위에서 사용하고 있기때문에 호스트의 도커엔진을 jenkins 컨테이너에 마운트 해야한다. 그래야 스크립트의 도커명령어 사용가능.

마운트 → 권한부여

```
docker run --name jenkins -d -p 8081:8080 -p 50000:50000 -v /var/run/docker.sock:/var/run
```

jenkins 컨테이너에도 도커명령어 사용권한 부여

```
usermod -aG docker jenkins
```

<http://52.78.118.241:8081> 접속

```
sudo docker logs jenkins //jenkins 초기 비밀번호 확인 출력된 로그에서
Please use the following password to proceed to installation 부분을 찾아 비밀번호를 복사
```

웹 브라우저에서 복사한 초기 관리자 비밀번호를 입력하고 설정 마법사를 진행

💡 jenkins/user

사용자 계정생성

userId: minsun

password:alstjs123

플러그 인 설치

<b>Docker Commons Plugin</b> 439.va_3cb_0a_6a_fb_29 Provides the common shared functionality for various Docker-related plugins. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>Docker Pipeline</b> 580.vc0c340686b_54 Build and use Docker containers from pipelines. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>Docker plugin</b> 1.6.2 This plugin integrates Jenkins with <a href="#">Docker</a> <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>GitLab Plugin</b> 1.8.1 This plugin allows <a href="#">GitLab</a> to trigger Jenkins builds and display their results in the GitLab UI. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>Gradle Plugin</b> 2.12 This plugin allows Jenkins to invoke <a href="#">Gradle</a> build scripts directly. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>SSH Agent Plugin</b> 367.vf9076cd4ee21 This plugin allows you to provide SSH credentials to builds via a ssh-agent in Jenkins. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<b>Strict Crumb Issuer Plugin</b> 2.1.1 A strict crumb issuer with capacities such session ID check, time-dependent validity or protection against BREACH. <a href="#">Report an issue with this plugin</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

## 💡jenkins관리/system

### Jenkins Location

Jenkins URL ?

http://52.78.118.241:8081/

System Admin e-mail address ?

address not configured yet <nobody@nowhere>

### Global properties

☐ Disable deferred wipeout on this node ?

☐ Disk Space Monitoring Thresholds

☒ Environment variables

키-값 목록 ?

이름

EC2\_IP

값

52.78.118.241

추가

☐ Tool Locations

### GitLab

☒ Enable authentication for '/project' end-point ?

GitLab connections

Connection name ?

A name for the connection

GitLab

GitLab host URL ?

The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)

https://lab.ssafy.com

Credentials ?

API Token for accessing GitLab

GitLab API token

+ Add

고급

Test Connection

👉credentials 에서는 gitlab의 personal access token 생성해서 추가해준다.

## 💡jenkins관리/Security

**CSRF Protection**

Crumb Issuer

Strict Crumb Issuer

Expiration (in hours) ?

2

⌵ ⌵ Edited

- ☐ Check the session ID
- ☐ Prevent BREACH attack
- ☐ Check the client IP (potential proxy incompatibility)
- ☐ Check HTTP referer header
- ☐ Limit the referer check to the local path

👉 **[Jenkins] 403 No valid crumb was included in the request** 에러 해결을 위해 설정.

CSRF와 관련된 보안문제로 gitLab에서 webhook 보내면 발생하는 문제.

💡 jenkins 관리/Credentials

## Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	docker-hub-credentials	sangmin0806/***** (Docker Hub credentials)
		System	(global)	ssh-key	jenkins
		System	(global)	GitLab_API_token	GitLab API token
		System	(global)	GitLab_Username_Credential	sangmin0806/*****
		System	(global)	EC2_IP_SECRET	EC2_IP_SECRET

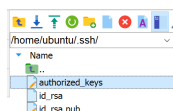
### 1. docker-hub-credentials

→ docker 허브 id 비밀번호 추가.

### 2. ssh-key설정

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com" //ssh-key 생성
```

~/ssh/id\_rsa 와 ~/ssh/id\_rsa.pub 에 저장됩니다.(개인키, 공개키(.pub))



authorized\_keys에 id\_rsa.pub 추가로 복붙해서 공개키 등록.

```
docker-compose.y... nginx.conf * authorized_keys
1 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACjPyjEXWC/GYqBxcz0vNrC0wx3Gxgkh0SD4WH7KDUyaJh46WkIj6Ld8ngdmIrQpHp
  nz63JVfUzWKMqtee5Eh/3yCzLPRNwxQeBxUcZ9086z89wDRfrr8KJBj/X3FHMOfmFBrEEMITp2moWnjkyUdy08wQ6VrXb0AZVoQ0yfnC
  A2Nh0HNrzL7LJjMe3VY2M6d731H1wCjkd/AVxzLHC9pvvbj50CGdb6YNYAhDS7S7z55glIBGeNHNhPsVqmHHMpVwnc6UzjPvfYz2gvIS1
  TKNg8lkobM07XCAVwChoZxMe2CWrXRhr21Zn2oJybcS5tvcCIjhZYqUSMSVwnc5crApH I11C206T
2 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACuxNa5etLBTV9NH7nXeT35efv6cYD7E6PY4rZ1q
  /9JrrXRXCsuRL29pbQNVc6yYYFph4bp7Wai/QwwjoBB1lju7lOVw1H6K8LbZwte1rnbbtHB4xCfeB9+sBXhVu8z7Rga3x5X
  +MPBRCS8HpTLzNBxKdHc31Ix7qmOCCfBTxyg99xups0P+7+TLns2BTLVZhfQOMBLJ3yOvzNi2Vi4gq
  /WBH6RzQpFVYh0pYGBd3Q4wzZEZQ42xZtvjHpfDhnyKdNI4LyQ7IYyRyB933dTrJi1h9ABlNyFgPGMA+81sI+AtUrSP
  +DQU8RK0yk50PnPtL0Eov2h4N0dFk1439y71mNW2LwZzkxCbpyPMrneqZFOmo18vwoI7RE/qkpxfkoH+gINX+Q2EF3XPB651ohmPIZ6
  +5paFwd1p4fXo5uY7bUChanUgU075vePiCOnvMPCL++nrUj1utm40LKLCZDwOCNgLh+eraUt
  /TIRBkb47bWwFguGS0kEv9KeLibZohtrJBUESuEjpyAmPIjsid+Ryes0SX03PEVZLK2rjKpQXD0pWouXvSet0oLdXfs2esc
  /8dnrY4HqeYZB4DV48P6Si/tQPHJSxY9WsAskjMmrco7ZegDt8fyOp1+porxiSsxfgbtXASBPq3RL
  /X07bHhristpFmdnwIAkKCKkWPf52Tw=-cistkd2866@naver.com
```

👉이건 근데 EC2에 위치한거. jenkins는 도커컨테이너 환경에서 돌아가기 때문에 개인키를 jenkins 컨테이너 안에 넣어야한다.

```

docker exec -u root -it jenkins bash
mkdir -p /var/jenkins_home/.ssh
exit

docker cp ~/.ssh/id_rsa jenkins:/var/jenkins_home/.ssh/id_rsa
docker cp ~/.ssh/id_rsa.pub jenkins:/var/jenkins_home/.ssh/id_rsa.pub

docker exec -u root -it jenkins bash
chown jenkins:jenkins /var/jenkins_home/.ssh/id_rsa /var/jenkins_home/.ssh/id_rsa.pub
chmod 600 /var/jenkins_home/.ssh/id_rsa
chmod 644 /var/jenkins_home/.ssh/id_rsa.pub
exit

```

jenkins 화면에서 credential 추가시 개인키 id\_rsa 복붙.

3. GitLab-API-Token  
→ GitLab personal Access Token 받아서 추가
4. GitLab-Username-credential  
→ GitLab personal Access Token 받아서 추가
5. EC2 IP 추가

## Jenkins Job 구성

메인페이지에서 new Item 누름

JobName :

**S11P12C206\_Pipeline**

- ☐ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☒ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

Authentication Token

pipeline-token

다음 URL을 사용하여 원격 빌드 유발: JENKINS\_URL/job/S11P12C206\_Pipeline/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME  
Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

## 👉 merge request 시에만 build trigger 설정

Authentication Token 이름 설정.(git lab webhook에서 사용)

### Pipeline

Definition

Pipeline script from SCM

Pipeline script

Pipeline script from SCM

Git

Repositories ?

Repository URL ?

https://lab.ssfy.com/s11-webmobile1-sub2/S11P12C206.git

Credentials ?

sangmin0806/\*\*\*\*\*

+ Add

고급

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/develop

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

## GitLab Webhook 구성

Q Search page

## 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

`http://minsun:11d6ba127934b532b01f18c657ff86634c@52.78.118.241:8081/job/S1`

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.

```
http://{id}:{secretKey}@{JenkinsURL}/job/{jobName}/build?token=Authentication Token Name
// jenkins 서버의 Jenkins Job 확인.
//secretKey : Dashboard > 사용자 > 대표계정 선택 > 설정 > API Token 탭으로 이동하여
새로운 토큰 생성
http://minsun:11d6ba127934b532b01f18c657ff86634c@52.78.118.241:8081/job/S11P12C206_Pipe1
```

## Webhook 트리거 설정

### Trigger

- ☐ Push events
- ☐ Tag push events  
A new tag is pushed to the repository.
- ☐ Comments  
A comment is made or edited on an issue or merge request.
- ☐ Confidential comments  
A comment is made or edited on a confidential issue.
- ☐ Issues events  
An issue is created, updated, closed, or reopened.
- ☐ Confidential issues events  
A confidential issue is created, updated, closed, or reopened.
- ☒ Merge request events  
A merge request is created, updated, or merged.
- ☐ Job events  
A job's status changes.
- ☐ Pipeline events  
A pipeline's status changes.
- ☐ Wiki page events  
A wiki page is created or updated.
- ☐ Deployment events  
A deployment starts, finishes, fails, or is canceled.
- ☐ Feature flag events  
A feature flag is turned on or off.

### SSL verification

- ☐ Enable SSL verification

Save changes

Test ▾

SSL verification 해제. jenkins는 http이므로.

## Jenkinsfile

```
pipeline {
    agent any
```



```

environment {
    SSH_KEY_ID = 'ssh-key'
    DOCKER_HUB_REPO = 'sangmin0806'
}

stages {
    stage('Build Backend') {
        steps {
            script {
                dir('backend') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean build -x test'

                    withCredentials([usernamePassword(credentialsId: 'docker-hub-cre
                        sh "docker build -t ${DOCKER_HUB_REPO}/backend ."
                        sh "echo \${DOCKER_HUB_PASS} | docker login -u \${DOCKER_HUB_US
                        sh "docker push ${DOCKER_HUB_REPO}/backend"
                    }
                    sh "docker image prune -f"
                    sh "docker builder prune -f"
                }
            }
        }
    }
    stage('Build Frontend') {
        steps {
            script {
                dir('frontend') {

                    withCredentials([usernamePassword(credentialsId: 'docker-hub-cre
                        sh "docker build -t ${DOCKER_HUB_REPO}/frontend ."
                        sh "echo \${DOCKER_HUB_PASS} | docker login -u \${DOCKER_HUB_US
                        sh "docker push ${DOCKER_HUB_REPO}/frontend"
                    }
                    sh "docker image prune -f"
                    sh "docker builder prune -f"
                }
            }
        }
    }
    stage('Deploy') {
        steps {
            script {
                withCredentials([string(credentialsId: 'EC2_IP_SECRET', variable: 'E
                    sshagent([SSH_KEY_ID]) {
                        sh 'export EC2_IP=${EC2_IP} && ssh -o StrictHostKeyChecking=no
                    }
                }
            }
        }
    }
}

```

🔍수동배포 명령어를 스크립트를 통해 설정한것뿐 과정은 동일하다.

ssh -o StrictHostKeyChecking=no //ssh 키 연결설정할때 처음연결시에도 승인하도록 처리한다.

merge 요청시 webhook 발동, jenkins 파이프라인 동작 확인.

The screenshot shows the Jenkins web interface. The left sidebar contains navigation links: Status, Changes, Console Output (selected), View as plain text, Edit Build Information, Delete build #30, Timings, Git Build Data, Pipeline Overview, Pipeline Console, Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main area displays the console output for build #30 of the S11P12C206\_Pipeline. The output shows the pipeline starting, checking out the code from a Git repository, and performing a checkout. The commit message is "Merge branch 'feature/S11P12C206-302-ki-out' into 'develop'". The pipeline is currently in a 'waiting' state.

```
Started by remote host 52.231.25.158
Started by remote host 52.231.25.158
Obtained Jenkinsfile from git https://lab.ssafty.com/s11-webmobile1-sub2/S11P12C206.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/S11P12C206_Pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
using credential GitLab_OwnerName_Credential
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/S11P12C206_Pipeline/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://lab.ssafty.com/s11-webmobile1-sub2/S11P12C206.git # timeout=10
Fetching upstream changes from https://lab.ssafty.com/s11-webmobile1-sub2/S11P12C206.git
> git --version # timeout=10
> git --version # 'git version 2.39.2'
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- https://lab.ssafty.com/s11-webmobile1-sub2/S11P12C206.git --refs/heads/:refs/remotes/origin/ # timeout=10
> git rev-parse refs/remotes/origin/develop^{commit} # timeout=10
Checking out Revision 4368896c8f6b3c9d17e6c4c5e999238f91144 (refs/remotes/origin/develop)
> git config core.sparsecheckout # timeout=10
> git checkout -f 4368896c8f6b3c9d17e6c4c5e999238f91144 # timeout=10
Commit message: "Merge branch 'feature/S11P12C206-302-ki-out' into 'develop'"
> git rev-list --no-walk f2b07c9ecf9731c7166ac9f1f665683c27de9f # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withEnv
[Pipeline] }
```