



포팅 매뉴얼

외부 서비스 정보

Kakao API : <https://developers.kakao.com/>

Clarifai API : <https://www.clarifai.com/>

OpenVidu : <https://openvidu.io/>

크롤링 사이트

만개의 레시피 : <https://www.10000recipe.com/>

Web Server : NginX Web Server

이미지 서버 : AWS S3

DB

- MySql Workbench : 8.0

Version

- Front - End

Node.js	18.13.0
React	18.2.0
VS Code	1.75.1
Create-react-app	5.0.1
React-router-dom	5.2.0
npm	8.19.3
redux	1.9.2
CSS	

- Back - End

IntelliJ	2022.3.1
JDK	11.0.13

SpringBoot	2.7.8
dependency	1.0.15
gradle	7.6
jwt	0.9.1
JPA	

DB 접속 정보

Host	<u>i8b304.p.ssafy.io</u>
Username	root
Password	root
Database	bobs2
Port	3306

Ignore 파일

application.properties.yml

```
# MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# DB Source URL (Local)
spring.datasource.url=jdbc:mysql://localhost:3306/bobs2?

# DB Source URL (Server)
# spring.datasource.url=jdbc:mysql://{도메인}:3306/bobs2?

# DB username
spring.datasource.username={username}

# DB password
spring.datasource.password={userpw}

# JPA
spring.jpa.show-sql=true

# DDL(create, alter, drop) setting
spring.jpa.hibernate.ddl-auto=none

# JPA Hibernate SQL
spring.jpa.properties.hibernate.format_sql=true

# Redis setting
spring.cache.type=redis
spring.redis.host={도메인}
spring.redis.port={****}
```

application.yml

```
spring:
  profiles:
    include: oauth
  servlet:
    multipart:
      max-file-size: 5MB
      max-request-size: 5MB

cloud:
  aws:
    credentials:
      accessKey: {accessKey}
      secretKey: {secretKey}
    s3:
      bucket: bobsimg
      region:
```

```

    static: ap-northeast-2
    stack:
      auto: false

#logging:
#  level:
#    root: debug1

```

application-oauth.yml

```

spring:
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: {client-id}
            client-secret: {client-secret}
            # Local uri
            redirect-uri: http://localhost:8080/login/oauth2/code/kakao
            # Server uri
            redirect-uri: https://{도메인}/api/login/oauth2/code/kakao
            client-authentication-method: POST
            authorization-grant-type: authorization_code
            scope: profile_nickname, account_email
            client-name: Kakao
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id

  app:
    auth:
      jwt:
        secret-key: {jwt-secret-key}

```

Kakao Dev 설정

개인정보 동의 항목

동의항목

카카오 로그인으로 서비스를 시작할 때 동의 받는 항목을 설정합니다. 미리 보기를 통해 사용자에게 보여질 화면을 확인할 수 있습니다. 사업자 정보를 등록하여 비즈 앱으로 전환하고 비즈니스 채널을 연결하면 권한이 없는 동의 항목에 대한 검수 신청을 할 수 있습니다.

[비즈니스 설정 바로가기](#)

개인정보

항목 이름	ID	상태	
닉네임	profile_nickname	● 필수 동의	설정
프로필 사진	profile_image	● 필수 동의	설정
카카오계정(이메일)	account_email	● 선택 동의 [수집]	설정

내 애플리케이션 > 앱 설정 > 플랫폼

Web

삭제

수정

사이트 도메인	http://localhost:8080 https://localhost:8080 https://localhost:3000 https://i8b304.p.ssafy.io
---------	--

• 카카오 로그인 사용 시 Redirect URI를 등록해야 합니다. [등록하러 가기](#)

Redirect URI

Redirect URI

[삭제](#)[수정](#)

Redirect URI	http://localhost:8080/login/oauth2/code/kakao http://i8b304.p.ssafy.io:8080/login/oauth2/code/kakao https://i8b304.p.ssafy.io:8080/login/oauth2/code/kakao http://i8b304.p.ssafy.io:8080/log http://i8b304.p.ssafy.io/login/oauth2/code/kakao https://i8b304.p.ssafy.io/login/oauth2/code/kakao http://i8b304.p.ssafy.io/api/login/oauth2/code/kakao https://i8b304.p.ssafy.io/api/login/oauth2/code/kakao https://i8b304.p.ssafy.io/oauth2/authorization/kakao
--------------	---

- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

EC2 개발 환경

Ubuntu 운영체제

version : 20.04 LTS

310 GB

Docker

version : 20.10.23

Docker Compose

version : 2.15.1

NginX (EC2)

Openvidu 먼저 설치 후에 Nginx 설치!

Docker

Jenkins (container)

version : 2.375.2

- 0.0.0.0 : 9090 → 8080
- -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -u root

MySQL (container)

- Version : 8.0.32 MySQL Community Server - GPL
- image name : mysql:latest
- volume name : mysql-volume
- 0.0.0.0 : 3306 → 3306
- -v mysql-volume:/var/lib/mysql

Nginx & React (container)

- 0.0.0.0 : 3000 → 80

SpringBoot (container)

- 0.0.0.0 : 8080 → 8080

OpenVidu (docker-compose container)

Redis (container)

- 0.0.0.0 : 8379 → 6379

배포 과정

Docker 설치

1. apt 업데이트 및 관련 패키지 install

```
sudo apt-get update
sudo apt-get install ca-certificates
sudo apt-get install curl
sudo apt-get install gnupg
sudo apt-get install lsb-release
```

2. GPG 키 추가

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

3. 저장소 setting

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. apt 패키지 업데이트

```
sudo apt-get update
```

5. if, Receiving a GPG error when running `apt-get update` ? 위에 명령에서 에러 안나면 스킵!!

```
sudo chmod a+r /etc/apt/keyrings/docker.gpg
sudo apt-get update
```

6. Docker Engine, Containerd, Docker Compose 설치

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

7. 잘 설치되었는지 test

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

Openvidu 설치 과정

- **Opnvidu를 설치 하기전 확인해주어야 할 사항!** (정답이 아닐 수 있지만 오류는 해결했습니다)
 - Nginx가 설치되어있다면 제거
 - 80, 443, 3478, 5442, 5443, 6379, 8888 포트들은 Opnvidu가 사용하는 포트이므로 겹치지 않게 설정해주어야 함
 - 만약 겹치는 docker 포트가 있다면 포트 변경을 해주면 됨. 해당 주소 참고(<https://buildabetterworld.tistory.com/176>)

Opnvidu 설치 참고 링크(<https://docs.opnvidu.io/en/2.22.0/deployment/ce/on-premises/>)

1. root 관리자로 이동

```
sudo su
```

2. /opt 경로로 이동

```
cd opt
```

3. opnvidu 설치 (curl 패키지 없으면 curl install 후 해당 내용 진행)

```
curl https://s3-eu-west-1.amazonaws.com/aws.opnvidu.io/install_opnvidu_latest.sh | bash
```

4. 설치된 Opnvidu 경로로 이동

```
cd opt/opnvidu
```

5. vi 편집기로 .env 파일 수정

```
vi .env

# OpnVidu configuration
# -----
# 도메인 또는 퍼블릭IP 주소
DOMAIN_OR_PUBLIC_IP={도메인}

# 오픈비두 서버와 통신을 위한 시크릿
OPNVIDU_SECRET=MYSECRET

# Certificate type
CERTIFICATE_TYPE=letsencrypt

# 인증서 타입이 letsencrypt일 경우 이메일 설정 (ec2 letsencrypt와 같게 꼭 설정해줘야함)
LESENCRYPT_EMAIL=*****@naver.com

# HTTP port
HTTP_PORT=8442

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

6. Opnvidu 실행 (현재 /opt/opnvidu 경로)

```
./opnvidu start
```

7. Ctrl + c 눌러서 백그라운드로 동작하게 하고 나온다.

8. docker container 전부 잘 동작하는지 확인

```
docker ps
```

Openvidu 포트 충돌 날 시 대처

- openvidu에서 사용하는 포트들 : 80, 443, 3478, 5442, 5443, 6379 and 8888.

1. 겹치는 포트 있을 시, 이미 실행한 컨테이너 포트포워딩 재설정하기

- Stop the container (docker stop <container_name>)

```
docker stop <container_name>
```

- Stop docker service (per Tacsiazuma's comment)

```
sudo systemctl stop docker
```

- Change the file -> config.v2.json 에서는 컨테이너 내부에서 외부로 공개되는 포트를 추가,수정할 수 있다. 여기서는 hostconfig.json 을 사용한다.

```
nano /var/lib/docker/containers/<container id(hash)>/hostconfig.json  
> portbinding 찾아서 포트번호 변경
```

- Restart your docker engine (to flush/clear config caches).

```
sudo systemctl start docker
```

- Start the container (docker start <container_name>)

```
docker restart <container_name>
```

Nginx 충돌 시 대처

1. 먼저, nginx가 포함된 docker, image 전부 삭제 해준다.
2. 기존 openvidu 정지 및 해당 docker, image 전부 삭제
3. /opt 에서 /openvidu 폴더 삭제
4. 혹시나 남아있는 docker와 연결안된 image 쓰레기 삭제

```
docker image prune -a
```

5. ec2 자체 설치된 nginx 삭제

```
# stop nginx  
sudo systemctl stop nginx  
  
# remove nginx  
sudo apt-get remove nginx
```

6. openvidu 재설치 후 openvidu 잘 동작하는지 확인
7. nginx 재설치

```
sudo apt-get install nginx
```

8. nginx 확인


```
# 다음 명령어에서 successful이 뜨면 nginx를 실행할 수 있다.
sudo nginx -t

sudo systemctl restart nginx
```

Nginx 설치, SSL 인증 과정, nginx.conf 설정

1. Nginx 설치

```
sudo apt-get install nginx
```

2. 버전 확인

```
nginx -v
```

3. letsencrypt 설치

```
sudo apt-get install letsencrypt

sudo systemctl stop nginx

sudo letsencrypt certonly --standalone -d {*****.p.ssafy.io}
```

4. 인증서 발급 성공 확인

```
/etc/letsencrypt/live/{도메인} # 해당 경로에 잘 저장됐는지 확인
```

5. cd /etc/nginx/sites-available 이동하고 nginxEC2.conf 파일 만들고 작성(자유로운 파일이름)

```
vi nginxEC2.conf
```

```
server {

    location /{
        proxy_pass http://localhost:3000;
    }

    location /api/ {
        proxy_pass http://localhost:8080/;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

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

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

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

6. 심볼릭 링크 생성해주고 nginx restart

```
sudo ln -s /etc/nginx/sites-available/nginxEc2.conf /etc/nginx/sites-enabled/nginxEc2.conf

#success 확인
sudo nginx -t

sudo systemctl restart nginx
```

MySQL 배포 과정

1. mysql image pull

```
docker pull mysql:latest
```

2. Docker 컨테이너 볼륨 설정 (백업)

```
docker volume create mysql-volume
```

```
# volume 확인
docker volume ls
```

3. volume을 mysql 컨테이너와 바인딩. root password 설정.

```
docker run -d --name mysql-container -p 3306:3306 -v mysql-volume:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=1234 mysql:latest
```

4. 컨테이너 접속

```
docker exec -it mysql-container /bin/bash
```

5. MySQL 서버에 접속

```
mysql -u root -p

Enter password:
...
mysql>
```

6. user 생성

```
# USER 생성, '%'는 모든 IP에서 접속 가능
mysql> CREATE USER test01@'%' identified by '1234';
# 생성한 USER에 모든 권한 부여
mysql> GRANT ALL PRIVILEGES ON *.* to test01@'%';
# 변경 사항 적용
mysql> FLUSH PRIVILEGES;
mysql> exit;
```

7. 생성한 user로 MySQL 서버에 접속

```
mysql -u test01 -p

Enter password:
...
mysql>
```

8. 데이터베이스 생성

```
mysql> CREATE DATABASE test;
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| test |
+-----+
5 rows in set (0.00 sec)
```

- MySQL workbench에서 외부 접속
Hostname에 서버의 IP or 도메인을 입력하고, 사용자명과 비밀번호를 입력 후 TestConnection 성공하면 진입

Redis 배포 과정

1. docker image pull

```
docker pull redis
```

2. docker run container

```
docker run -p 8379:6379 --name redis_db -d redis #openvidu에서 6379포트를 써서 8379로 설정
```

3. cli 접속 방법

```
docker exec -it redis_db redis-cli
```

Jenkins 설정 과정

1. jenkins image pull

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

2. jenkins 컨테이너 실행

```
sudo docker run -d --name jenkins -p 9090:8080 -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock
```

- -v /jenkins:/var/jenkins_home
 - jenkins와 host가 volume을 공유할 수 있도록 바인딩 한다.
- -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock
 - jenkins 내부에서 도커 명령어를 실행해야하므로 jenkins container 내부에 도커를 또 설치해야한다. 이 방법은 docker in docker 이고 docker에서는 권장하지 않는 방법이다.
따라서 위 바인딩으로 host의 docker를 사용할 수 있도록 해준다(docker out docker 방법)

3. Jenkin 서버 접속 (도메인주소:9090)

- 접속하면 키값을 요구할 것이다.

```
docker logs jenkins # 실행하여 키값 확인 및 복사 후 붙여 넣어준다.
```

- Admin User로 회원가입을 한다.
- 플러그인 관리에서 Nodejs 설치 해준다. **Gloval Tool Configuration**에 들어가서 리액트 버전에 맞게 선택 후 ADD 해준다.
- 플러그인 관리에서 Gitlab 관련 파일을 설치한다.
- 깃랩 프로젝트 access tokens을 생성한다.(생성 직후만 확인할 수 있으므로 복사 해둔다.)
- 젠킨스 Credentials 등록한다. username with password(kind)로 등록하고, password에 복사한 토큰을 붙여넣는다.

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

gitlab_connection_bobs

Gitlab host URL

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

https://lab.ssafy.com

Credentials

API Token for accessing Gitlab

GitLab API token

+ Add

고급...

Test Connection

Frontend, Backend 배포를 위한 Pipeline 작성

bobs-front pipeline

```
pipeline {
  agent any

  environment {
    GIT_URL = "https://lab.ssafy.com/s08-webmobile1-sub2/S*****"
  }

  tools {
    nodejs "nodejs-blog"
  }

  stages {
    stage('Pull') {
```

```

    steps {
        git url: "${GIT_URL}", branch: "develop", credentialsId: "jenkins_gitlab"
    }
}

stage('React Build') {
    steps {
        dir('frontend/pjt01') {
            sh 'pwd'
            sh 'npm i --legacy-peer-deps'
            sh 'CI=false npm run build'
        }
    }
}

stage('Build') {
    steps {
        dir('frontend/pjt01') {
            sh 'docker build -t nginx-react:0.1 .'
        }
    }
}

stage('Deploy') {
    steps {
        script {
            try {
                sh 'docker ps -q -f name=nginx-react | grep . && docker stop nginx-react && docker rm nginx-react'
            } catch (e) {
                sh 'exit 0'
                sh 'echo docker container stop and remove Fail!!'
            }
        }
        sh 'docker run --name nginx-react -d -p 3000:80 nginx-react:0.1'
    }
}

stage('Finish') {
    steps {
        sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
    }
}
}

```

- stage('Pull')
 - 깃랩의 develop branch를 Pull 해온다.
- stage('React Build')
 - npm install 명령을 수행 후 npm run build 명령어 실행한다.
- stage('Build')
 - Dockerfile을 bulid 하여 image를 생성한다.
- stage('Deploy')
 - 현재 실행 중인 같은 이름의 배포 container가 있다면 정지 시킨 후 제거한다
 - 실행되고 있는 container가 없다면 에러가 발생하므로 try catch로 에러를 잡았다.
- stage('Finish')
 - 현재 컨테이너의 쓰이고 있지 않는 이미지나 이름이 없는 이미지를 다 지워준다.
- Dockerfile

```

# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# root 에 app 폴더를 생성
RUN mkdir /app

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 80 포트 오픈
EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]

```

Backend Pipeline

```

pipeline {
    agent any

    environment {
        GIT_URL = "https://lab.ssafy.com/s08-webmobile1-sub2/S*****"
    }

    stages {
        stage('Pull') {
            steps {
                git url: "${GIT_URL}", branch: "develop", credentialsId: "jenkins_gitlab"
            }
        }

        stage('copy file') {
            steps {
                sh 'cp /****/****/application-oauth.yml /var/jenkins_home/workspace/bobs-springboot/backend/Project/bobs/src/main/re'
                sh 'cp /****/****/application.properties /var/jenkins_home/workspace/bobs-springboot/backend/Project/bobs/src/main/r'
                sh 'cp /****/****/application.yml /var/jenkins_home/workspace/bobs-springboot/backend/Project/bobs/src/main/resource'
            }
        }

        stage('Build') {
            steps {
                dir('backend/Project/bobs') {
                    sh 'javac -version'
                    sh 'chmod +x gradlew'
                    sh './gradlew clean build -x test'
                    sh 'docker build -t bobs_backend:0.1 .'
                }
            }
        }

        stage('Deploy') {
            steps{
                script {
                    try {
                        sh 'docker ps -q -f name=bobs_backend | grep . && docker stop bobs_backend && docker rm bobs_backend'
                    } catch (e) {
                        sh 'exit 0'
                        sh 'echo bobs_backend docker container stop and remove Skip!!'
                    }
                }
                sh 'docker run --name bobs_backend -d -p 8080:8080 bobs_backend:0.1'
            }
        }

        stage('Finish') {
            steps{

```

```

    sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
  }
}
}

```

- stage('Pull')
 - 깃랩의 develop branch를 Pull 해온다.
- stage('copy file')
 - 설정파일들은 pull을 받아오지 못하므로 EC2내에서 직접 작성하고 바인딩하여 해당 위치로 복사한다.
- stage('Build')
 - Dockerfile을 bulid 하여 image를 생성한다.
- stage('Deploy')
 - 현재 실행 중인 같은 이름의 배포 container가 있다면 정지 시킨 후 제거한다
 - 실행되고 있는 container가 없다면 에러가 발생하므로 try catch로 에러를 잡았다.
- stage('Finish')
 - 현재 컨테이너의 쓰이고 있지 않는 이미지나 이름이 없는 이미지를 다 지워준다.
- Dockerfile

```
FROM openjdk:11
VOLUME /tmp
EXPOSE 8080
ARG JAR_FILE=build/libs/bobs-1.0-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Asia/Seoul
RUN apt-get install -y tzdata
```