

# Porting Manual

---

## 목차

[목차](#)

[개발환경](#)

[서버 인스턴스 사양](#)

[형상관리](#)

[OS](#)

[협업 툴](#)

[UI/UX](#)

[기타 편의 툴](#)

[이슈 관리](#)

[Back-end](#)

[Front-end](#)

[Infra](#)

[DB](#)

[IDE](#)

[API](#)

[배포 과정](#)

[1. SSAFY EC2\(서버\) 접속](#)

[1-1. WSL을 사용하여 EC2에 SSH 연결](#)

[2. EC2 초기 설정](#)

[3. 한국으로 시간 설정 \(안해도 됨\)](#)

[4. EC2 환경 설정](#)

[1. Docker 설치](#)

[5. Docker Compose 설치](#)

[MySQL 설치 \(본 프로젝트는 MySQL로 진행하였음\)](#)

[레디스 설치](#)

[3. Dockerfile로 Jenkins images 받기 \(Docker out of Docker, DooD 방식\)](#)

**[ Jenkins 컨테이너 안에서 직접적으로 Docker를 실행하지 않고 호스트 시스템의 Docker를 활용하여 컨테이너 관리 작업을 수행]**

[Jenkins 초기 세팅 및 테스트 \(호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신하기위함!\)](#)

[Jenkins 플러그인 설정](#)

[Gitlab](#)

[Docker](#)

[Webhook 설정](#)

[7. CI/CD \(빌드 및 배포\) 세팅](#)

[요즘은 코드 기반의 자동화를 선호하며, 파이프라인이 더 널리 사용되고 있습니다.](#)

[파이프라인을 사용하면 코드와 인프라 구성을 함께 관리하며, 배포 파이프라인을 수정하거나 공유하기가 더 간편합니다. 라고합니다.](#)

[배포 환경 구성](#)

[도메인 구매](#)

[SSL 발급 받기](#)

[배포 시작](#)

[빌드 순서는 JenkinsFile → start.prod.sh → docker-compose-prod.yml → Dockerfile](#)

[JenkinsFile \(stages\[큰 묶음\] → stage\[진짜 실행되는 작은 묶음\]\)](#)

[start-prod.sh](#)

[DockerFile - BackEnd](#)

[DockerFile - FrontEnd](#)

[Nginx 설정](#)

[conf.d 폴더의 default.conf file](#)

[Nginx 파일 \(Front-End\) Nginx.conf](#)

[빌드 절차](#)

[외부 서비스 → 구글 로그인](#)

[외부 서비스](#)

[1. Google 로그인](#)

[S3 버킷](#)

[외부 서비스 → application.yml 설정](#)

[1. S3 Bucket](#)

[2. Openvidu](#)

[3. Naver Mail](#)

## 개발환경

### 서버 인스턴스 사양

- CPU 정보 : Intel Core i7-9750H  
CPU @ 2.60GHz 2.59GHz

- 코어개수 : 6

- RAM and Disk :

프로세서 Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

기본 속도: 2.59GHz  
소켓: 1  
코어: 6  
논리 프로세서: 12  
가상화: 사용  
L1 캐시: 384KB  
L2 캐시: 1.5MB  
L3 캐시: 12.0MB

```
ubuntu@ip-172-26-1-121:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        311G  121G  191G   39% /
```

설치된 RAM 16.0GB

### 형상관리

- GitLab

### OS

- Window 10
- Ubuntu 20.04.6 LTS

### 협업 툴

- MatterMost
- Discord

### UI/UX

- Figma

### 기타 편의 툴

- Postman
- Terminus
- Git Window Ver.

### 이슈 관리

- Jira (이슈 관리)

### Back-end

Java	11
Springboot	2.7.14
gradle	8.1.1
Openvidu	2.19.0
FCM	7.1.1
Lombok	1.18.24
websocket	2.7.2

### Front-end

React	18.2.0
Redux Toolkit	1.9.5
react-redux	8.1.1
react-router-dom	6.14.2
Axios	1.4.0
Firebase	10.1.0
react-scripts	5.0.1
sockjs-client	1.6.1
stompjs	2.3.3
openvidu-browser:	2.28.0
react-i18next	13.1.2
Cors	2.8.5
react-google-login	5.2.2
Emotion	11.11.1
Framer Motion	10.15.1
Framer Motion	10.15.1
@react-oauth/google	0.11.1

### Infra

Web Server	nginx
Jenkins	2.416
Docker	24.0.5
Docker-compose	1.25.0

@testing-library/jest-dom	5.17.0
@testing-library/react	13.4.0
@testing-library/user-event	13.5.0
Gapi-script	1.2.0
http-proxy-middleware	2.0.6
i18next	23.4.4
js-cookie	3.0.5
jwt-decode	3.1.2
net	1.0.2
react-icons	4.10.1
sweetalert2	11.7.20
web-vitals	2.1.4
sass	1.64.1
react-bootstrap	2.8.0
MUI (Material-UI)	5.14.3
Bootstrap	5.3.1

## DB

MySQL	8.0.32
Redis	7.0.12

## IDE

IntelliJ	2023.1.3 (Ultimate Edition)
React	7.0.12

## API

STT	Speech recognition
Translation	Google Translation

## 배포 과정

- EC2 서버는 구축된 상태

### 1. SSAFY EC2(서버) 접속

#### 1-1. WSL을 사용하여 EC2에 SSH 연결

- SSH 접속
- 최초 접속 시 권한 요구하면 'yes' 입력

```
# sudo ssh -i [pem키 위치] [접속 계정]@[접속할 도메인]
$ sudo ssh -i I9B204T.pem ubuntu@19b204.p.ssafy.io
```

- EC2에 편하게 접속하는 방법 (TIP)
  - EC2 정보가 담긴 config파일을 만들어 번거롭게 pem와 도메인 경로를 쓰지 않고 접속할 수 있다.
  - ssh 전용 폴더 생성

```
mkdir ~/.ssh
cd ~/.ssh // ssh 폴더 생성 및 이동
cp [로컬 pem 키 위치] ~/.ssh // pem 키 옮기기
vi config // config 파일 생성
```

- config 내용 추가

```
Host ssafy
  HostName [서버 ip 주소]
```

```
User ubuntu
IdentityFile ~/.ssh/[pem키 파일 명].pem
```

- ssafy 계정에 접속

## 2. EC2 초기 설정

```
$ sudo apt update //시스템의 패키지 정보를 업데이트
$ sudo apt upgrade //시스템의 패키지 정보들중 업그레이드
$ sudo apt install build-essential // "build-essential" 패키지를 설치하여서 소프트웨어 개발을 위해 필요한 기본적인 도구와 라이브러리를 포함
//안정성 및 배포에 문제가 생길수있으니 미리 해두는것임.
```

## 3. 한국으로 시간 설정 (안해도 됨)

```
$ sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime

ln: 링크(link)를 생성하는 명령
-sf: 옵션으로, 심볼릭 링크(symLink)를 생성하고 이미 존재하는 경우에 덮어쓰는 것을 의미
/usr/share/zoneinfo/Asia/Seoul: 심볼릭 링크를 만들 대상 파일로, 이 경우 서울 시간대의 정보가 들어 있는 파일을 가리킴
/etc/localtime: 생성될 심볼릭 링크의 경로로, 시스템의 로컬 시간대 설정을 나타냄

쉽게 말해 링크는 원본 파일의 복사본, 심볼릭 링크는 원본 파일이나 디렉토리를 가리키는 별도의 파일
# 시간 확인
$ date
```

## 4. EC2 환경 설정

### 1. Docker 설치

#### 1. 기본 설정, 사전 설치

```
$ sudo apt update //해봤으면 안해도됨~~
$ sudo apt install apt-transport-https ca-certificates curl software-properties-common //쉽게말하자면 HTTPS를 통한 패키지 저장소 접속!
apt-transport-https 는 HTTPS 프로토콜을 통해 패키지 저장소를 접근하기 위해 필요한 패키지
ca-certificates: 인증서 관련 패키지로, SSL/TLS 연결 시 사용되는 인증서 정보를 포함
curl: 커맨드 라인에서 URL을 통해 데이터를 전송하는데 사용되는 도구
software-properties-common: 소프트웨어 저장소 관련 공통 도구 및 라이브러리 패키지
```

- HTTPS를 통한 패키지 저장소를 접속하기위함
- SSL 인증서 관련 문제를 해결한다.
- 파일 다운로드, API 호출에 사용될수있다
- 저장소 추가, 제거, 업데이트 등 CRUD에 용이하다.

#### 2. 자동 설치 스크립트 활용

- 리눅스 배포판 종류를 자동으로 인식하여 Docker 패키지를 설치해주는 스크립트를 제공

```
$ sudo wget -qO- https://get.docker.com/ | sh //결국 https://get.docker.com 에 있는 스크립트를 다운받겠다는 의미다.
wget은 웹에서 파일을 받겠다는 의미다.
-qO-은 다운로드된 내용을 화면에 표시하지 않고 직접 출력하라는 의미
|: 파이프(pipe) 기호로, 한 명령의 출력을 다른 명령의 입력으로 전달하는 역할
sh: 다운로드한 스크립트를 실행하는 명령
```

- 결국 Docker를 설치하기위해서 간편하게 제공되는 공식 스크립트를 이용한 것이다.

#### 3. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```
$ sudo systemctl start docker //이 명령어는 Docker 서비스를 시작하는 명령 start는 Docker 서비스를 수동으로 이용
$ sudo systemctl enable docker //이 명령어는 시스템이 부팅될 때 Docker 서비스가 자동으로 시작되도록 설정, enable 명령은 서비스를 부팅 시 자동으로 시작되도록
```

- 결국은 두 명령어를 순서대로 실행하면 Docker 서비스가 시작되고, 시스템이 부팅될 때마다 Docker 서비스도 함께 자동으로 시작되어 컨테이너 기반의 애플리케이션을 운영할수있다.

- 젠킨스를 할때 자주 Docker가 자동으로 실행되는 것 때문에 오류가 자주났던것같은데, `sudo systemctl enable docker` 이 오류 일 수도있겠다는 생각이 든다.

#### 4. Docker 그룹에 현재 계정 추가

```
$ sudo usermod -aG docker ${USER}
$ sudo systemctl restart docker
```

- sudo를 사용하지 않고 docker를 사용할 수 있다.
- docker 그룹은 root 권한과 동일하므로 꼭 필요한 계정만 포함
- 현재 계정에서 로그아웃한 뒤 다시 로그인

#### 5. Docker 설치 확인

```
$ docker -v
```

## 5.Docker Compose 설치

- 설치하는 이유 ?

**실행 명령어를 일일이 입력하기 복잡해서**

**컨테이너끼리 연결을 편하게 하기 위해서**

**가상 네트워크를 편리하게 연결하기 위해서**

- 최신 버전을 가져오기 위한 jq 라이브러리 설치

최신 버전을 가져오기 위한 jq 라이브러리 설치

- docker-compose 최신 버전 설치

```
$ VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
$ DESTINATION=/usr/bin/docker-compose
$ sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-$(uname -s)-$(uname -m) -o $DESTINATION
$ sudo chmod 755 $DESTINATION

// 터미널 재접속 하기!

$ docker-compose -v
Docker Compose version v2.x.x
```

- `$ VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)`
  - `curl --silent https://api.github.com/repos/docker/compose/releases/latest` : GitHub API를 사용하여 Docker Compose의 가장 최신 릴리스 정보를 가져옵니다.
  - `$ DESTINATION=/usr/bin/docker-compose`
  - `jq .name -r` : `jq`를 사용하여 JSON에서 릴리스 이름을 추출합니다. `-r` 옵션은 결과를 원시 문자열로 출력하도록 합니다.
  - `VERSION=` : 추출한 릴리스 이름을 변수 `VERSION`에 할당합니다.

**버전 확인 및 변수 설정:**

- GitHub API를 통해 Docker Compose의 최신 릴리스 버전을 가져와 변수에 저장

- `$ DESTINATION=/usr/bin/docker-compose`
  - `DESTINATION=` : Docker Compose 실행 파일의 설치 경로를 지정합니다.

**Docker Compose 다운로드 및 설치:**

- Docker Compose 실행 파일을 최신 버전으로 다운로드하여 지정된 경로에 저장합니다.

- `$ sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-$(uname -s)-$(uname -m) -o $DESTINATION`
  - `sudo curl -L ... -o $DESTINATION`: 지정된 경로에 Docker Compose 바이너리 파일을 다운로드하고 저장합니다.
  - `${VERSION}` 은 앞서 가져온 Docker Compose 버전을 사용합니다.
  - `$(uname -s)` 는 현재 운영 체제의 이름 (예: Linux, Darwin)을,
  - `$(uname -m)` 은 현재 시스템 아키텍처 (예: x86\_64)를 나타냅니다.

#### Docker Compose 다운로드 및 설치:

- Docker Compose 실행 파일을 최신 버전으로 다운로드하여 지정된 경로에 저장합니다.

- `$ sudo chmod 755 $DESTINATION`
  - `sudo chmod 755 ...`: 다운로드한 Docker Compose 실행 파일에 실행 권한을 부여합니다.

#### 실행 권한 부여:

- 다운로드한 Docker Compose 파일에 실행 권한을 부여하여 사용할 수 있도록 합니다.

- // 터미널 재접속 하기!
  - 이 부분은 설치 스크립트 실행 후에 새로운 권한을 적용하기 위해 터미널을 재접속하라는 안내
- `$ docker-compose -v`
  - 설치가 올바르게 이루어졌는지 확인하기 위해 Docker Compose의 버전을 출력

## MySQL설치 ( 본 프로젝트는 MySQL로 진행하였음)

### 1. Docker Maria DB 이미지 다운로드 받기

```
$ docker pull mysql
```

### 2. Docker에 Maria DB 컨테이너 만들고 실행하기

```
$ docker run --name mysql -d -p 3306:3306 -v /var/lib/mysql_main:/var/lib/mysql --restart=always -e MYSQL_ROOT_PASSWORD=root mariadb
```

#### ▼ 옵션 설명

- `-v`: 마운트 설정, host 의 `/var/lib/mysql` 과 `mariadb`의 `/var/lib/mysql`의 파일들을 동기화
- `-name`: 만들어서 사용할 컨테이너의 이름을 정의
- `d`: 컨테이너를 백그라운드에서 실행
- `p`: 호스트와 컨테이너 간의 포트를 연결 (`host-port:container-port`) // 호스트에서 3306 포트 연결 시 컨테이너 3306 포트로 포워딩
- `-restart=always`: 도커가 실행되는 경우 항상 컨테이너를 실행
- `e`: 기타 환경설정(Enviornment)
- `MYSQL_ROOT_PASSWORD=root` // `mariadb`의 root 사용자 초기 비밀번호를 설정
- `mariadb`: 컨테이너를 만들 때 사용할 이미지 이름

### 3. Maria DB에 database를 추가하고 user 권한 설정

- Docker - Mariadb 컨테이너 접속하기

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

- MySQL - 루트 계정으로 데이터베이스 접속하기

```
mysql -u root -p
```

비밀번호는 "root"

### MySQL 사용자 추가하기

```
예시) create user 'user_name'@'XXX.XXX.XXX.XXX' identified by 'user_password';

create user 'ssafy601'@'%' identified by 'ssafy601';
```

### MySQL - 사용자 권한 부여하기

```
예시) grant all privileges on db_name.* to 'user_name'@'XXX.XXX.XXX.XXX';
flush privileges;

grant all privileges on *.* to 'ssafy601'@'%';
flush privileges;
```

### MySQL - 데이터 베이스 만들기

```
예시) create database [db_name];

create database ssafy601;
```

## 레디스 설치

- Redis 이미지 받기

```
docker pull redis:alpine
```

- 도커 네트워크 생성 [디폴트값]

```
docker network create redis-network
```

- 도커 네트워크 상세정보 확인

```
docker inspect redis-network
```

- local-redis라는 이름으로 로컬-docker 간 6379 포트 개방

```
docker run --name local-redis -p 6379:6379 --network redis-network -v /redis_temp:/data -d redis:alpine redis-server --appendonly yes
```

- Docker 컨테이너 확인

```
docker ps -a
```

- 컨테이너 진입

```
# 실행 중인 redis 컨테이너에 대해 docker redis-cli 로 직접 진입
docker run -it --network redis-network --rm redis:alpine redis-cli -h local-redis

# bash로도 진입 가능하다.
docker run -it --network redis-network --rm redis:alpine bash
redis-cli
```

- 권한 추가

```
# slaveof no one : 현재 슬레이브(복제)인 자신을 마스터로 만듭니다.
127.0.0.1:6379> slaveof no one
```

- 테스트
  - OK 가 뜨면 성공

```
127.0.0.1:6379> slaveof no one
OK
127.0.0.1:6379> set apple 100
OK
127.0.0.1:6379> get apple
"100"
```

### 3. Dockerfile로 Jenkins images 받기 (Docker out of Docker, DooD 방식)

**[ Jenkins 컨테이너 안에서 직접적으로 Docker를 실행하지 않고 호스트 시스템의 Docker를 활용하여 컨테이너 관리 작업을 수행]**

- Dockerfile 작성

```
# 폴더 생성
mkdir config && cd config

# 아래 내용 작성
$ vi Dockerfile

FROM jenkins/jenkins:jdk17

#도커를 실행하기 위한 root 계정으로 전환
USER root

#Jenkins 컨테이너 내에서 Docker를 사용할 수 있도록 설정하는 부분
COPY docker_install.sh /docker_install.sh
RUN chmod +x /docker_install.sh
RUN /docker_install.sh

#설치 후 도커그룹의 jenkins 계정 생성 후 해당 계정으로 변경
RUN groupadd -f docker
RUN usermod -aG docker jenkins
USER jenkins
```

- docker, Jenkins 설정 shell 파일 (이또한 config 폴더에 vi docker\_install.sh or vi docker.install.sh 파일을 만들어서 넣어야한다.

긴가민가한 내용 ( 뭘 써야하는지 모르겠음)

```
#!/bin/sh
apt-get update && \
apt-get -y install apt-transport-https \
ca-certificates \
curl \
gnupg2 \
zip \
unzip \
software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$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

//GPG 키를 다운로드하고 저장소의 인증을 추가하는 것은 Docker의 소스가 신뢰할 수 있는지 확인
//시스템이 Docker 소프트웨어를 신뢰할 수 있도록 하는 과정
//즉, 해당 스크립트는 Docker 설치를 직접적으로 처리하지 않고, Docker 관련 설정을 수행하기 위한 작업
```

- Docker 이미지 생성

```
docker build -t jenkins/myjenkins . //현재 디렉토리에 있는 Dockerfile을 사용하여 젠킨스 이미지를 빌드하라는 의미
```



- Docker 볼륨 폴더 권한 설정

```
#디렉토리는 Jenkins 컨테이너에서 사용할 데이터 및 설정을 저장하기 위한 목적으로 사용
$ mkdir /var/jenkinsDir/
#Jenkins 컨테이너가 /var/jenkinsDir/ 디렉토리에 쓰기 및 읽기 권한을 가지게 됨
$ sudo chown 1000 /var/jenkinsDir/
```

- Jenkins 컨테이너 생성

```
docker run -d -p 9090:8080 --name=jenkinscid \
-e TZ=Asia/Seoul \
-v /var/jenkinsDir:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/myjenkins
```

- 옵션 설명

**-d** : 백그라운드에서 실행을 의미

**-p** 는 매핑할 포트를 의미합니다. ( p가 port의 단축어가 아니었음 .. )

**:** 기준으로 왼쪽은 로컬포트, 오른쪽은 도커 이미지의 포트를 의미합니다. 도커 이미지에서의 8080 포트를 로컬 포트 9090으로 매핑한다는 뜻입니다.

```
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/myjenkins
```

이 옵션은 로컬의 도커와 젠킨스 내에서 사용할 도커 엔진을 동일한 것으로 사용하겠다는 의미입니다.

**-v** 옵션은 ":"를 기준으로 왼쪽의 로컬 경로를 오른쪽의 컨테이너 경로로 마운트 해줍니다.

즉, 제 컴퓨터의 **사용자경로/jenkinsDir** 을 컨테이너의 **/var/jenkins\_home** 과 바인드 시켜준다는 것입니다. 물론, 양방향으로 연결됩니다. 컨테이너가 종료되거나 알 수 없는 오류로 정지되어도, jenkins\_home에 남아있는 소중한 설정 파일들은 로컬 경로에 남아있게 됩니다.

## Jenkins 초기 세팅 및 테스트 (호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신하기위함!)

- 젠킨스에 접속하기 전에 **/var/run/docker.sock** 에 대한 권한을 설정해주어야 합니다.
- 초기 **/var/run/docker.sock** 의 권한이 **소유자와 그룹 모두 root**였기 때문에 이제 그룹을 root에서 **docker** 로 변경해줄겁니다.
- 먼저, jenkins로 실행했던 컨테이너의 bash를 root 계정으로 로그인 하기전에, 현재 실행되고 있는 컨테이너의 정보들을 확인할 수 있는 명령어를 입력해 아이디를 확인하겠습니다.

### ? 왜 이렇게하는가 ?

- **/var/run/docker.sock** 은 Docker 데몬과 통신하기 위한 소켓 파일입니다. 이 소켓을 통해 호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신합니다. 그런데 이 소켓 파일은 기본적으로 소유자와 그룹 모두 root 계정에 속해 있습니다.
- Jenkins 컨테이너 내에서 Docker를 실행하기 위해서는 이 소켓 파일에 대한 적절한 권한 설정이 필요합니다. Jenkins 컨테이너 내의 Jenkins 프로세스가 Docker 데몬과 통신할 수 있도록 하려면 다음과 같은 이유로 해당 권한 설정을 수행합니다:
- 우리가 방금 생성한 컨테이너의 ID는 **0bcdb8~** 입니다. 도커는 다른 컨테이너 ID와 겹치지 않는 부분까지 입력하면 해당 컨테이너로 알아서 매핑해줍니다.

```
docker exec -it -u root 컨테이너ID /bin/bash
```

**exec** 는 컨테이너에 명령어를 실행시키는 명령어인데, /bin/bash와 옵션 -it를 줌으로써 컨테이너의 셸에 접속할 수 있습니다.

이제 정말로 root 계정으로 컨테이너에 접속하기 위해 컨테이너ID에 0bc를 입력해 실행합니다.

```
> docker exec -it -u root 0bc /bin/bash
root@0bcdb8291015:/#
```

- root 계정으로 로그인에 잘 되었습니다. 이제 그룹을 바꾸기 위해 다음 명령어를 실행해줍니다.

```
chown root:docker /var/run/docker.sock
```

- 그리고 이제 쉘을 exit 명령어로 빠져나온 후 다음 명령어를 실행해 컨테이너를 재실행해줍니다.

```
docker restart [컨테이너 ID]
```

- Jenkins 패스워드 확인

```
docker logs [jenkins 컨테이너 ID]
```

- docker logs 컨테이너 id를 입력해 로그를 출력하면 initialAdminPassword가 출력됩니다. 이 패스워드를 입력해주면 됩니다.

```
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

b99f67a2cf054174a73017e4498ce87d

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****
```

- 보안 그룹 설정을 해야 {public ip}:9090 에 접근할 수 있습니다.

EC2 > 보안 그룹 >

인바운드 규칙 (2)



태그 관리

인바운드 규칙 편집

인바운드 규칙 편집 클릭

인바운드 규칙 정보					
보안 그룹 규칙 ID	유형 정보	프로토콜 정보	포트 범위 정보	소스 정보	설명 - 선택 사항 정보
sgr-04d3d90131b6c0b79	사용자 지정 TCP	TCP	9090	사용자 지정 Q 0.0.0.0/0 X	삭제
sgr-05db5853e7a0e7cbf	SSH	TCP	22	사용자 지정 Q 0.0.0.0/0 X	삭제

규칙 추가

규칙 추가 클릭 → 유형: TCP, 포트 범위: {등록할 포트 번호}

# Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

## Install suggested plugins

Install plugins the Jenkins community finds most useful.

## Select plugins to install

Select and install plugins most suitable for your needs.

- 정상적으로 입력했다면 플러그인 설치가 나오는데, 우리는 Install suggested plugins를 선택합니다.

✓ Folders	OWASP Markup Formatter	Build Timeout	Credentials Binding	** SSH server Folders ** Trilead API
Timestampers	Workspace Cleanup	Ant	Gradle	
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View	
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication	
LDAP	Email Extension	Mailer		

- 설치가 완료되면, 어드민 계정 생성창이 나오고, 본인이 사용하실 정보들을 입력해줍니다.

## Create First Admin User

계정명:	<input type="text"/>
암호:	<input type="password"/>
암호 확인:	<input type="password"/>
이름:	<input type="text"/>
이메일 주소:	<input type="text"/>

# Instance Configuration

Jenkins URL:

http://localhost:9090/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

- 앞으로 이 url로 젠킨스에 접속하시면 됩니다.

## Jenkins 플러그인 설정

- Gitlab, Docker 플러그인을 받습니다. (헛갈리다면 비슷한거라도 플러그인 다운받으세요, 너무 다른거 말고)

### Gitlab

이름 ↓

Generic Webhook Trigger Plugin 1.86.2

Can receive any HTTP request, extract any values from JSON and many more.

[Report an issue with this plugin](#)

GitLab 1.6.0

This plugin allows **GitLab** to trigger Jenkins builds and display build status.

[Report an issue with this plugin](#)

Gitlab API Plugin 5.0.1-78.v47a\_45b\_9f78b\_7

This plugin provides **GitLab API** for other plugins.

[Report an issue with this plugin](#)

GitLab Authentication plugin 1.16

This is the an authentication plugin using gitlab OAuth.

[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers.

### Docker

이름 ↓

Docker API Plugin 3.2.13-37.vf3411c9828b9

This plugin provides **docker-java** API for other plugins.

[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers.

Docker Commons Plugin 1.21

Provides the common shared functionality for various Docker plugins.

[Report an issue with this plugin](#)

Docker Compose Build Step Plugin 1.0

Docker Compose plugin for Jenkins

[Report an issue with this plugin](#)

Docker Pipeline 563.vd5d2e5c4007f

Build and use Docker containers from pipelines.

[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers.

Docker plugin 1.3.0

This plugin integrates Jenkins with **Docker**

[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers.

docker-build-step 2.9

This plugin allows to add various docker commands

[Report an issue with this plugin](#)

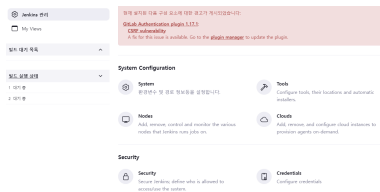
- 여기까지 오셨다면, 젠킨스 설치 및 초기 세팅 완료!

## 6. CI/CD (빌드 및 배포) 초기세팅 (먼저 Plugins에서 GitLab이랑 WebHook을다운받는 다.)

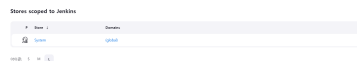
먼저 Jenkins 관리 → Credentials 를 누른다. → 먼저 만드는 이유는?

1. Credentials를 먼저 만들어서 Jenkins 관리에서 관리하는 것은 보안을 강화하고 민감한 정보를 안전하게 저장하며, 효율적인 관리와 재사용성을 가능하게 하는 중요한 단계
2. 즉 Credentials를 중앙화하여 관리함으로써 여러 프로젝트 또는 빌드에서 동일한 인증 정보를 사용할 수 있습니다. 변경이 필요한 경우, 한 곳에서 수정하면 모든 사용처에 즉시 적용됨.

1. 먼저 Jenkins 관리 → Credentials 를 누른다.



2. Add Credentials를 누른다



3. Add Credentials를 누른다



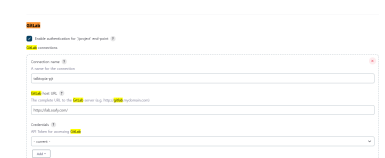
1. Kind는 Username with password 방식
2. UserName은 메일 주소 입력
3. lab.ssafy.com 계정 비밀번호를 입력한다.



2. GitLab Connection을 하기위해서 Jenkins 관리의 System을 누른다.



3. Connection은 자유
4. GitLab Host URL에서 lab.ssafy.com 입력



1. Pipeline에서 구성을 선택한다.

2. Build when a change is pushed to Gitlab~ 체크

3. WebHook 등록하기위한 Secret Token 생성

4. 이걸로 WebHook으로 자동 감지하게 만들어야함.



Status

pipeline

</> Changes

▷ 지금 빌드

⚙ 구성

🗑 Pipeline 삭제

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

Stage View

No data available. This Pipeline has not yet run.

고정링크

☀ Build History

추이 ▼

## Webhook 설정

- Jenkins 트리거 체크

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://i8a6l

- Jenkins 에서 빌드 유발 → Build when ... → 고급 → 하단에 Secret token Generate → 토큰 발급 완료!
- Gitlab Webhook에서 해당 토큰을 등록합니다.
  - lab.ssafy.com Gitlab 프로젝트 접속
  - 좌측 Settings → Webhook 접속
  - Jenkins Project URL 입력 후, Secret Token 입력 그리고 Trigger는 원하는 Event에 대해서만 설정
  - 생성후 Test를 눌렀을 때 200 응답이 return되면 성공

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http:// ssafy.io: /project/shabit-main ?

### URL

http://example.com/trigger-ci.json

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

### Secret token

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

### Trigger

☒ Push events

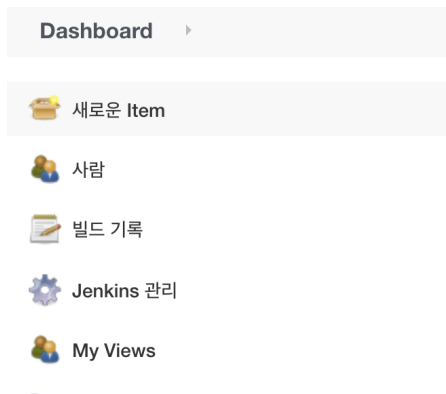
Branch name or wildcard pattern to trigger on (leave blank for all)

## 7. CI/CD (빌드 및 배포) 세팅

# Jenkins is ready!

Your Jenkins setup is complete.

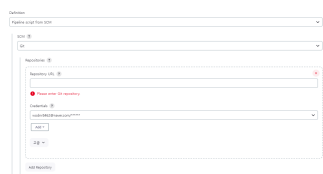
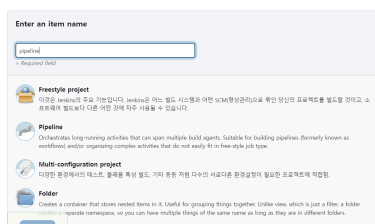
Start using Jenkins



- 먼저, 대쉬보드의 파이프라인 을 클릭합니다.
- FreeStyle Project는 요즘은 많이 안쓴다고 들음... 파이프라인을 많이 쓰는이유는

요즘은 코드 기반의 자동화를 선호하며, 파이프라인이 더 널리 사용되고 있습니다. 파이프라인을 사용하면 코드와 인프라 구성을 함께 관리하며, 배포 파이프라인을 수정하거나 공유하기가 더 간편합니다. 라고합니다.

1. 파이프라인 선택
2. SCM Git 설정 , URL 설정, Credentials 설정
3. 병합할 브랜치 선택, Script Path는 JenkinsFile로 다룰예정



## 배포 환경 구성

### 도메인 구매

- 가비아 접속

웹을 넘어 클라우드로. 가비아

그룹웨어부터 멀티클라우드까지 하나의 클라우드 허브

<https://www.gabia.com/>



- 도메인 선택

!!아래는 예시이고 실제 구매한 도메인은 "talktopia.site"입니다.

전체 1건					≡ 10 20 50 100
<input type="checkbox"/> 도메인	talktopia.site	2023-07-25 ~ 2024-07-25 (D-343) 연장 >	49,000원/년	관리	

- DNS 설정
  - My 가비아 → DNS 관리툴 → DNS 관리에서 호스트 / 값 설정

- 호스트에는 @, 값/위치에는 domain 주소를 작성합니다.

CNAME	@	19b20	600	DNS 설정	수정	삭제
CNAME	www	19b20	600	DNS 설정	수정	삭제
CNAME			600	DNS 설정	수정	삭제

## SSL 발급 받기

- nginx가 설치되어있는 상태에서 시작하겠습니다 (nginx 설치하는 다음장에)
- `sudo certbot certonly --standalone` 실행
  - 80번포트가 개방되어야함 -> 이미 실행중인 서비스, 데몬이 존재시, 이를 중단해야함
  - 자동 갱신 가능
  - 와일드카드 서브도메인 사용불가 (\*.example.com)
  - 도메인이 자신의 서버에 연결되어야함 (A레코드를 자신의 서버로)
  - 와일드카드 서브도메인 인증서가 필요하지 않다면 권장

```
dev@vultr:~$ sudo certbot certonly --standalone
[sudo] password for dev: <root password>
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator standalone, Installer None
Enter email address (used for urgent renewal and security notices)
(Enter 'c' to cancel): vompressor@gmail.com

-----
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf. You must
agree in order to register with the ACME server. Do you agree?
-----
(Y)es/(N)o: Y <- ACME 약관에 동의하는지 N선택시 진행불가

-----
Would you be willing, once your first certificate is successfully issued, to
share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
-----
(Y)es/(N)o: N <- 이메일을 통해 Let's Encrypt 프로젝트 정보를 받아볼지
Please enter in your domain name(s) (comma and/or space separated)
(Enter 'c' to cancel): vompressor.com www.vompressor.com <- {1} 인증서를 발급할 도메인 입력
Requesting a certificate for vompressor.com and www.vompressor.com

IMPORTANT NOTES:
- Congratulations! Your certificate and chain have been saved at:
  /etc/letsencrypt/live/vompressor.com/fullchain.pem <- {2} 발급된 인증서 경로
  Your key file has been saved at:
  /etc/letsencrypt/live/vompressor.com/privkey.pem <- {2} 발급된 인증서 경로
  Your certificate will expire on 2021-05-16. To obtain a new or
  tweaked version of this certificate in the future, simply run
  certbot again. To non-interactively renew *all* of your
  certificates, run "certbot renew"
- If you like Certbot, please consider supporting our work by:

  Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
  Donating to EFF: https://eff.org/donate-le
```

- 이때의 fullchain.pem과 privkey.pem 이존재한다.
  - 나는 fullchain.pem과 privkey.pem 라는 파일을 따로 빼놓았다.
  - 이유 → nginx 를 Docker 내에 삽입시키기위해서 Docker에서 nginx을 올릴때 SSL 을 적용하기위해서 두 파일을 같이 올리기때문
    - 그렇기때문에 프로젝트내에 .nginx/cert 폴더에 넣는다 ( SSL 을 인증하기위해서)
  - BackEnd의 SSL 통신을위해 key.p12 파일을만든다.
  - `openssl pkcs12 -export -in fullchain.pem -inkey privkey.pem -out key.p12`
  - yml or properties의 파일 동일선상에 넣어둔다.그리고 yml or properties 파일에 문장을 넣어준다.



```
server:
  port: 8000
  ssl:
    key-store: classpath:keystore.p12
    key-store-password: let's encrypt password
    key-store-type: PKCS12
```

## 배포 시작

빌드 순서는 JenkinsFile → start.prod.sh → docker-compose-prod.yml → Dockerfile

- 이 모두는 프로젝트내에 존재한다.
- 실행되는 순서는 BackEnd → FrontEnd → nginx 순으로 올라간다.

JenkinsFile (stages[큰 묶음] → stage[진짜 실행되는 작은 묶음])

```
pipeline {
  agent any
  stages {
    # 준비단계 -> Merge가 된 git 파일을 webHook에서 감지하여 갖고온다.
    stage('Prepare') {
      steps {
        sh 'echo "Clonning Repository"'
        git branch: 'master',
            url: 'https://gitlab주소/프로젝트명/프로젝트파일.git',
            credentialsId: 'Jenkins내에있는 credentialsId'
      }
      post {
        success {
          sh 'echo "Successfully Cloned Repository"'
        }
        failure {
          sh 'echo "Fail Cloned Repository"'
        }
      }
    }

    // stage('[BE]Bulid Gradle') {
    //   steps {
    //     sh 'echo "Bulid Gradle Start"'
    //     dir('BE') {
    //
    //     }
    //   }
    //   post {
    //     failure {
    //       sh 'echo "Bulid Gradle Fail"'
    //     }
    //   }
    // }

    #기존에있는 Docker 컨테이너들을 내려야함.
    stage('Docker stop'){
      steps {
        dir('BE'){
          sh 'echo "Docker Container Stop"'
        }
        // 도커 컴포즈 다운
        // sh 'curl -L https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m) -
        // 해당 도커 컴포즈 다운로드 경로로 권한 설정
        // sh 'chmod -R 777 /usr/local/bin/'
        // sh 'chmod +x /usr/local/bin/docker-compose'
        // 기존 백그라운드에 돌아가던 컨테이너 중지
        ## 기존 백그라운드에 돌아가던 컨테이너들을 DooD 방식으로 다운시킴.
        sh 'docker-compose -f /var/jenkins_home/workspace/test3/docker-compose-prod.yml down'
        //sh 'docker-compose -f docker-compose-prod.yml down'

      }

    }

    post {
      failure {
        sh 'echo "Docker Fail"'
      }
    }
  }
}
```

```

}
#정지된 도커 컨테이너를 삭제함
stage('RM Docker'){
  steps {

    sh 'echo "Remove Docker"'

    //정지된 도커 컨테이너 찾아서 컨테이너 ID로 삭제함
    #ttp로 시작하는건 다 삭제함.
    sh '''
      result=$( docker container ls -a --filter "name=ttp*" -q )
      if [ -n "$result" ]
      then
        docker rm $(docker container ls -a --filter "name=ttp*" -q)
      else
        echo "No such containers"
      fi
    '''
    sh '''
      result=$( docker container ls -a --filter "name=ttp*" -q )
      if [ -n "$result" ]
      then
        docker rm $(docker container ls -a --filter "name=ttp*" -q)
      else
        echo "No such containers"
      fi
    '''

    // homesketcher로 시작하는 이미지 찾아서 삭제함
    sh '''
      result=$( docker images -f "reference=ttp*" -q )
      if [ -n "$result" ]
      then
        docker rmi -f $(docker images -f "reference=ttp*" -q)
      else
        echo "No such container images"
      fi
    '''
    sh '''
      result=$( docker images -f "reference=ttp*" -q )
      if [ -n "$result" ]
      then
        docker rmi -f $(docker images -f "reference=ttp*" -q)
      else
        echo "No such container images"
      fi
    '''

    // 안쓰는이미지 -> <none> 태그 이미지 찾아서 삭제함
    sh '''
      result=$(docker images -f "dangling=true" -q)
      if [ -n "$result" ]
      then
        docker rmi -f $(docker images -f "dangling=true" -q)
      else
        echo "No such container images"
      fi
    '''

  }
  post {
    failure {
      sh 'echo "Remove Fail"'
    }
  }
}
# 프로젝트 내에 있는 start-prod.sh 실행
stage('Set Permissions') {
  steps {
    // 스크립트 파일에 실행 권한 추가
    sh 'chmod +x /var/jenkins_home/workspace/test3/start-prod.sh'
  }
}
stage('Execute start-prod.sh Script') {
  steps {
    // start-prod.sh 스크립트 실행
    sh '/var/jenkins_home/workspace/test3/start-prod.sh'
  }
}
}

//
// stage('[FE] prepare') {
//   steps {
//     dir('frontend'){
//       sh 'echo " Frontend Bulid Start"'
//       script {
//         sh 'docker-compose stop'
//       }
//     }
//   }
// }

```

```
//          sh 'docker rm vue'
//          sh 'docker rmi frontend_vue'
//      }
//  }

//  }

//      post {
//          failure {
//              sh 'echo "Frontend Build Fail"'
//          }
//      }
//  }
//  stage('Fronteend Build & Run') {
//      steps {
//          dir('frontend'){
//              sh 'echo " Frontend Build and Start"'
//              script {
//
// //          업데이트된 코드로 빌드 및 실행
//          sh 'docker-compose up -d'
//      }
//  }

//  }

//      post {
//          failure {
//              sh 'echo "Bulid Docker Fail"'
//          }
//      }
//  }
}
```

## start-prod.sh

```
#!/bin/bash

#pem_key_path="/c/Users/SSAFY/Desktop/I9B204T.pem"
#
## 접속할 EC2 인스턴스의 주소
#ec2_instance_address="i9b204.p.ssfy.io"

docker-compose -f docker-compose-prod.yml pull //현재 프로젝트에있는 docker-compose

COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose-prod.yml up --build -d

docker rmi -f $(docker images -f "dangling=true" -q) || true
```

- `docker-compose -f docker-compose-prod.yml pull`
  - `docker-compose-prod.yml` 파일을 사용하여 정의된 서비스의 이미지들을 Docker Hub 또는 지정된 레지스트리로부터 풀(pull)하여 업데이트함.
  - 컨테이너 이미지를 최신 버전으로 업데이트하려는 목적으로 사용함
- `COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose-prod.yml up --build -d`
  - `docker-compose-prod.yml` 파일을 사용하여 정의된 서비스들을 빌드하고 컨테이너를 배포합니다.
  - `COMPOSE_DOCKER_CLI_BUILD` 와 `DOCKER_BUILDKIT` 환경 변수를 설정하여 Docker Compose가 Docker CLI를 사용하여 빌드하고, 빌드 시 BuildKit을 사용하도록 지시합니다.
  - `-build` 옵션은 새 이미지를 빌드하도록 지시합니다.
  - `-d` 옵션은 컨테이너를 백그라운드에서 실행하도록 합니다.
- `docker rmi -f $(docker images -f "dangling=true" -q) || true`:
  - 빌드 과정에서 생성되고 사용되지 않는 "dangling" 이미지를 강제로 삭제합니다.
  - `docker images -f "dangling=true" -q` 명령어를 통해 dangling 이미지들의 ID를 얻고, `-f` 옵션은 필터를 적용하여 dangling 이미지만 선택합니다.

- **q** 옵션은 이미지 ID만 표시하도록 합니다.
- **docker rmi -f** 명령어를 통해 해당 이미지들을 강제로 삭제합니다. (**f**는 강제 삭제를 의미합니다.)
- **|| true** 부분은 명령어 실행 중 에러가 발생해도 종료 코드가 0 (성공)으로 유지되도록 합니다. (에러가 없다면 실행 결과가 true 이므로 무시됩니다.)

## Docker-compose-prod.yml (여러 서비스 및 컨테이너를 정의하여 멀티 컨테이너 어플리케이션을 구축하고 실행하는 데 사용)

```
version: "3" #Compose 파일의 버전
services: # 서비스 정의를 시작합니다. 각 서비스는 별도의 컨테이너로 실행
  server:
    image: ttp-back:latest # 이미지 이름
    container_name: ttp_back # 컨테이너 이름
    build:
      context: ../BE/Talktopia/talktopia # 컨테이너와 호스트 간의 포트 매핑을 설정합니다.
      # 컨테이너 내부의 포트를 호스트의 포트와 연결하여 외부에서 접근할 수 있게 함
    args:
      SERVER_MODE: prod
    ports:
      - 10001:8000 #외부 Port : 내부 Port
    environment: #environment:: 컨테이너 내부의 환경 변수를 설정합니다.
      - TZ=Asia/Seoul
    networks:
      - talktopia_network
  server2:
    image: ttp-back-chat:latest
    container_name: ttp_back_chat
    build:
      context: ../BE/Talktopia/talktopia_chat
    args:
      SERVER_MODE: prod
    ports:
      - 15000:7500
    environment:
      - TZ=Asia/Seoul
    networks:
      - talktopia_network
  client:
    image: ttp-front:latest
    container_name: ttp_front
    build:
      context: ../FE/talktopia
      dockerfile: Dockerfile #굳이안써도됨
    ports:
      - 3000:3000
    depends_on: # depends_on: 이 옵션은 서비스 간의 실행 순서와 의존성을 설정하는 데 사용, depends_on은 단순히 서비스의 실행 순서를 조정하고, 서비스가 실행
      - server
    networks:
      - talktopia_network #networks: 이 옵션은 서비스가 어떤 네트워크에 속하는지 정의하는 데 사용됩니다. 여러 서비스 간의 통신을 위해 사용되는 네트워크를
  nginx:
    image: ttp-nginx:latest
    container_name: ttp_nginx
    build: ../nginx
    depends_on:
      - server
    ports:
      - 80:80 # HTTP
      - 443:443 # HTTPS
    networks:
      - talktopia_network
  # redis:
  #   image: redis
  #   container_name: redis
  #   hostname: talktopia.site
  #   ports:
  #     - 6379:6379
  #   networks:
  #     - talktopia_network
networks:
  talktopia_network:
    driver: bridge
```

- 웹 서버는 웹페이지를 저장하고 보여주는 역할을 함
- 리버스 프록시 서버는 요청을 받아서 어떤 컴퓨터에게 연결시킬지 도와주는 역할

## DockerFile - BackEnd

```
#FROM openjdk:11
#VOLUME /tmp
#EXPOSE 8000
#ARG JAR_FILE=build/libs/talktopia-0.0.1-SNAPSHOT.jar
#COPY ${JAR_FILE} /app.jar
#ENTRYPOINT ["java","-jar","/app.jar"]
#ENV TZ=Asia/Seoul

FROM openjdk:11 as builder

COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

FROM openjdk:11
COPY --from=builder build/libs/*.jar app.jar
EXPOSE 10001

ARG SERVER_MODE
RUN echo "$SERVER_MODE"
ENV SERVER_MODE=$SERVER_MODE

ENTRYPOINT ["java", "-Dspring.profiles.active=${SERVER_MODE}", "-Duser.timezone=Asia/Seoul", "-jar", "/app.jar"]
```

빌더 단계 (as builder):

FROM openjdk:11 as builder: OpenJDK 11 이미지를 사용하여 빌더 단계를 시작  
필요한 파일과 디렉토리를 복사하고 권한을 설정하여 애플리케이션을 빌드  
./gradlew bootJar: Gradle을 사용하여 Spring Boot 애플리케이션의 JAR 파일을 빌드  
실행 단계 (FROM openjdk:11):

FROM openjdk:11: 런타임 단계에서는 또 다른 OpenJDK 11 이미지를 사용  
COPY --from=builder build/libs/\*.jar app.jar: 빌더 단계에서 생성된 JAR 파일을 복사하여 컨테이너 내의 app.jar로 지정된 위치에 배치  
EXPOSE 10001: 컨테이너가 노출하는 포트 번호를 지정.  
환경 설정과 실행:

ARG SERVER\_MODE: 빌드 단계와 런타임 단계에서 사용할 환경 변수를 정의  
ENV SERVER\_MODE=\$SERVER\_MODE: SERVER\_MODE 환경 변수 값을 설정  
ENTRYPOINT: 컨테이너가 시작될 때 실행할 명령을 지정 Spring Boot 애플리케이션을 실행하고 프로파일 및 타임존 설정을 적용  
이 Dockerfile은 멀티스테이지 빌드를 활용하여 애플리케이션을 빌드하고 실행하기 위해 두 개의 단계를 사용하고 있습니다. 빌더 단계에서는 애플리케이션을 빌드하고,  
실행 단계에서는 빌더 단계에서 생성된 JAR 파일을 컨테이너 내에서 실행합니다. 이렇게 하면 최종 이미지의 크기가 줄어들며, 빌드 과정과 런타임 환경을 분리하여 관리할 수 있습니다.

## DockerFile - FrontEnd

```
FROM node:18.13-alpine as builder

# 작업 폴더를 만들고 npm 설치
WORKDIR /usr/src/app
COPY package.json /usr/src/app/package.json

RUN npm install --force

# 소스를 작업폴더로 복사하고 빌드
COPY . /usr/src/app
RUN npm run build

FROM nginx:alpine
# nginx의 기본 설정을 삭제하고 웹에서 설정한 파일을 복사
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d

# 위에서 생성한 앱의 빌드산출물을 nginx의 샘플 앱이 사용하던 폴더로 이동

COPY --from=builder /usr/src/app/build /usr/share/nginx/html

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

Dlr
```

# Nginx 설정

## conf.d 폴더의 default.conf file

- Nginx는 웹 서버 소프트웨어로서 클라이언트의 요청을 받아 정적 파일을 서비스하거나, 리버스 프록시 서버로 동작하여 백엔드 애플리케이션 서버로 요청을 전달하는 역할

```
server {
    listen      80;
    server_name talktopia.site;
    return 301 https://talktopia.site$request_uri;

    location /api {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass http://외부IP주소:10001;
    }

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

server {
    listen 443 ssl;
    server_name talktopia.site;

    ssl_certificate /cert/fullchain1.pem;
    ssl_certificate_key /cert/privkey1.pem;

    location /api/v1 {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass http://외부IP주소:10001;
    }
    location /ws {
        #websocket
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection upgrade;
        proxy_set_header Host $host;

        proxy_intercept_errors on;
        proxy_pass http://외부IP주소:10001;

        error_page 404 404.html;
        error_page 500 502 503 504 50x.html;
    }

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

server {
    listen      80;
    server_name talktopia.site;
    return 301 https://talktopia.site$request_uri;

    location /api {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_pass http://외부IP주소:15000;
    }

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

server {
    listen 443 ssl;
    server_name talktopia.site;
```

```

ssl_certificate /cert/fullchain1.pem;
ssl_certificate_key /cert/privkey1.pem;

location /api/v1 {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_pass http://외부IP주소:15000;
}
location /ws {
    #websocket
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection upgrade;
    proxy_set_header Host $host;

    proxy_intercept_errors on;
    proxy_pass http://외부IP주소:15000;

    error_page 404 404.html;
    error_page 500 502 503 504 50x.html;
}

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

```

HTTP 80 포트 서버 (HTTP 요청 리다이렉션):  
이 서버는 80 포트에서 HTTP 요청을 처리합니다.  
server\_name talktopia.site: 요청이 talktopia.site 도메인으로 온 경우 처리합니다.  
return 301 https://talktopia.site\$request\_uri;: 모든 HTTP 요청을 HTTPS로 리다이렉션합니다.  
/api 경로로 들어오는 요청은 백엔드 서버의 API로 프록시됩니다.  
나머지 요청은 클라이언트 서비스로 프록시됩니다.

HTTPS 443 포트 서버 (HTTPS 요청 처리):  
이 서버는 443 포트에서 HTTPS 요청을 처리합니다.  
ssl\_certificate와 ssl\_certificate\_key 설정을 통해 SSL 인증서 및 개인 키를 설정합니다.  
/api/v1 경로로 들어오는 요청은 백엔드 서버의 API로 프록시됩니다.  
/ws 경로로 들어오는 요청은 WebSocket 요청으로 프록시됩니다.  
나머지 요청은 클라이언트 서비스로 프록시됩니다.

HTTP 80 포트 서버 (HTTP 요청 리다이렉션, Chat 서버 설정):

위와 동일한 패턴으로 HTTP 요청을 처리합니다.  
/api 경로로 들어오는 요청은 다른 백엔드 서버의 API로 프록시됩니다.

## Nginx 파일 (Front-End) Nginx.conf

```

server {
    listen 3000;
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {

        root /usr/share/nginx/html;
    }
}

```

listen 3000;: 이 서버 블록은 3000번 포트에서 들어오는 HTTP 요청을 처리합니다.  
location / { ... }: 이 블록은 루트 경로(/)에 대한 요청을 처리합니다.  
root /usr/share/nginx/html;: 해당 경로에 있는 파일들을 서빙하는 루트 디렉토리를 지정합니다.  
index index.html index.htm;: 해당 디렉토리에서 기본적으로 사용할 인덱스 파일을 설정합니다.  
try\_files \$uri \$uri/ /index.html;: 요청된 URI에 해당하는 파일이나 디렉토리를 찾지 못하면, 항상 /index.html 파일을 반환합니다. 이것은 SPA 라우팅을 위한 설정  
error\_page .... 오류 페이지에 대한 처리를 설정합니다.  
error\_page 500 502 503 504 /50x.html;: 500, 502, 503, 504 오류 발생 시 /50x.html 페이지를 표시합니다.  
location = /50x.html { ... }: 오류 페이지를 처리하는 블록입니다.  
root /usr/share/nginx/html;: 오류 페이지 파일이 위치한 디렉토리를 설정합니다.  
이러한 설정은 Nginx가 3000번 포트에 들어오는 요청을 처리하며, 정적 파일들을 제공하고, SPA 라우팅을 위한 설정을 포함하고 있습니다. 요청된 URI에 해당하는 파일이나 디렉

## 빌드 절차

- Gitalb에서 Jenkins로 Webhooks을 연동한 다음 해당 브랜치에 push, merge 를 진행합니다.
  - start-prod.sh 쉘 파일 실행
  - docker-compose 실행
  - Dockerfile 실행
  - Server → Front 순으로 배포 진행

## 외부 서비스 → 구글 로그인


### 외부 서비스

#### 1. Google 로그인

1. Google Cloud 접속

Cloud Computing Services | Google Cloud

Meet your business challenges head on with cloud computing services from Google, including data management, hybrid & multi-cloud, and AI & ML.

 <https://cloud.google.com/>



2. 최초 가입이라면 카드 정보 등록
3. API 및 서비스
4. 프로젝트 생성
5. 사용자 인증 정보 - OAuth2 클라이언트 ID - 웹 어플리케이션 선택

API	API 및 서비스	사용자 인증 정보
	<ul style="list-style-type: none"><li>사용 설정된 API 및 서비스</li><li>라이브러리</li><li><b>사용자 인증 정보</b></li><li>OAuth 동의 화면</li><li>페이지 사용 동의</li></ul>	<div>사용자 인증 정보 만들기 삭제</div> <p>사용 설정한 API에 액세스하려면 사용자 인증 정보를 만드세요. <a href="#">자세히 알아.</a></p> <div>API 키</div> <div><input type="checkbox"/> 이름 생성일</div> <p>표시할 API 키가 없습니다.</p> <div>OAuth 2.0 클라이언트 ID</div>

1. 서버 URI 추가

- ex ) <https://talktopia.site/login/oauth2/code/google>



## 승인된 리디렉션 URI ②

웹 서버의 요청에 사용

URI 1 \*

https://www.example.com

🗑️

❗ 올바른 리디렉션 URI는 비워 둘 수 없습니다.

+ URI 추가

참고: 설정이 적용되는 데 5분에서 몇 시간이 걸릴 수 있습니다.

만들기 취소

- 생성 후 클라이언트 ID 및 Secret 키 저장

생성일 ↓ 유형

### OAuth 클라이언트 생성됨

API 및 서비스의 사용자 인증 정보에서 언제든지 클라이언트 ID와 보안 비밀에 액세스할 수 있습니다.

❗

OAuth 동의 화면이 확인될 때까지 OAuth에서 [민감한 범위 로그인](#)이 100개로 제한됩니다. 확인 절차가 필요할 수 있으며 며칠 정도 걸립니다.

클라이언트 ID

d.apps.gc 🗑️

클라이언트 보안 비밀번호

GOC 🗑️

⬇️ JSON 다운로드

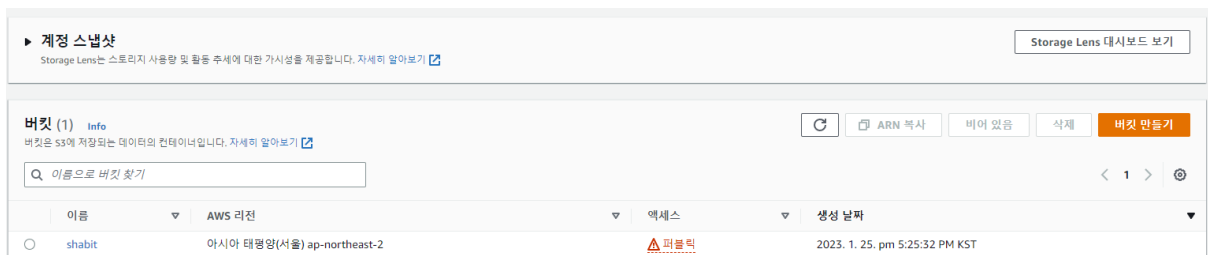
확인

## S3 버킷

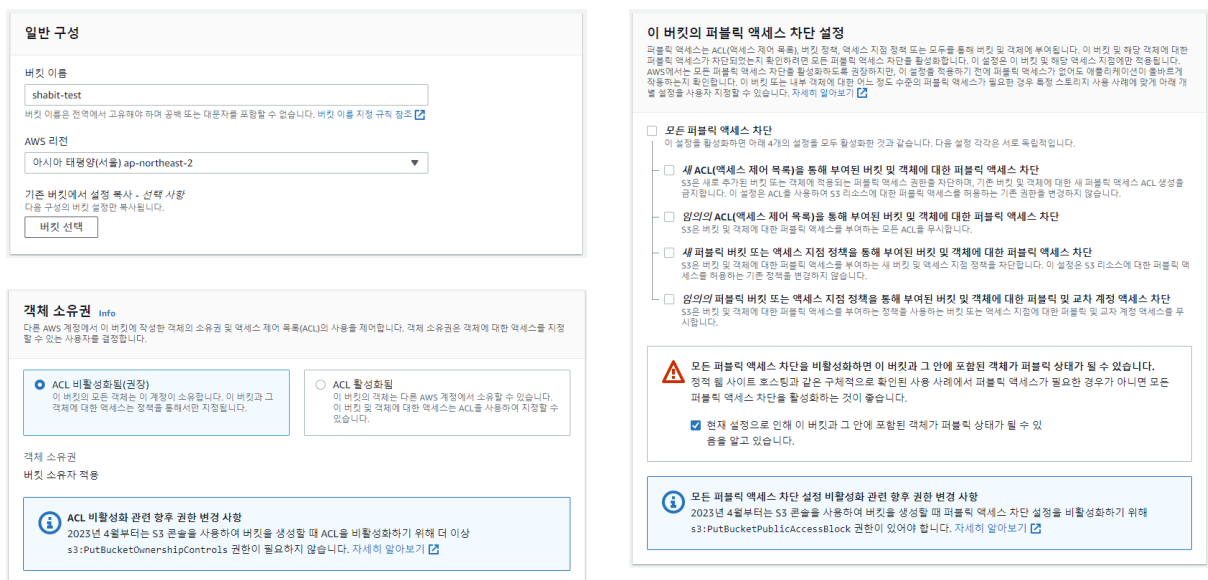
1. AWS 계정 생성
2. AWS 서비스 검색창에 S3 검색 및 이동



### 3. 버킷 클릭



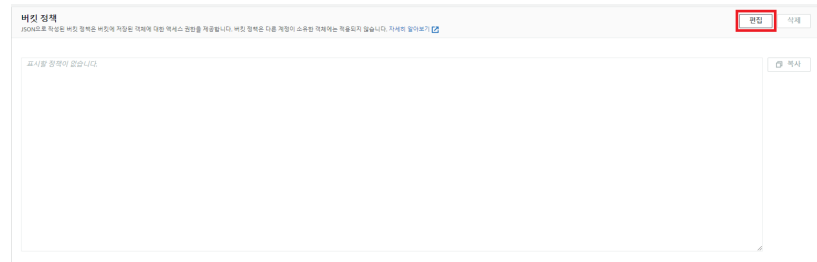
### 4. 버킷 정보 입력



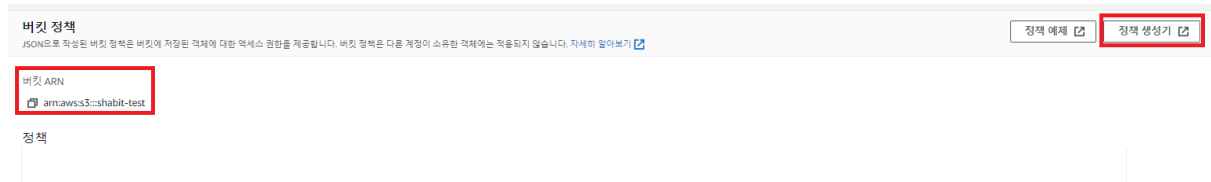
### 5. 버킷 설정 - 권한 설정

- [권한](#) 클릭 → [버킷 정책 편집](#) 클릭





- 권한 클릭 → 버킷 정책 편집 클릭 → 정책 생성기 클릭



- Action에는 `GetObject`, `PutObject`, `DeleteObject` 3개를 체크하고 ARN에는 복사해둔 ARN값을 입력한다.
- ARN값을 입력하되 `/*` 값도 추가 해줘야한다. ARN 값이 `arn:aws:s3:::test`라 가정하면 `arn:aws:s3:::test/*` 라고 입력해주면 된다.

## Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 Bucket Policy](#), an [SNS Queue Policy](#).

Select Type of Policy S3 Bucket Policy

## Step 2: Add Statement(s)

A statement is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal \*

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services (\*\*)

Use multiple statements to add permissions for more than one service.

Actions 3 Action(s) Selected ☐ All Actions (\*\*)

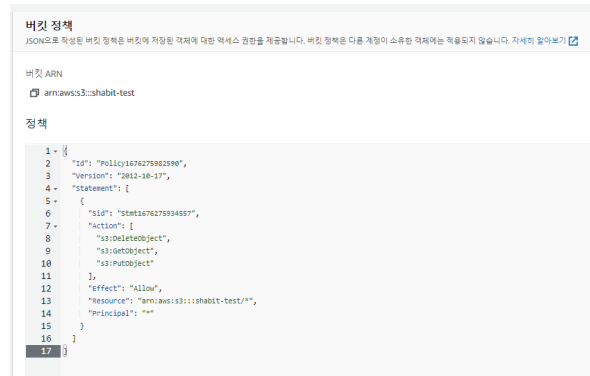
Amazon Resource Name (ARN) arn:aws:s3::shabit-test/\*

ARN should follow the following format: `arn:aws:s3:::${BucketName}/${KeyName}`.  
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

Add Statement

- 스크롤을 아래로 내려 `Generate Policy` 버튼을 클릭한다.
- 생성된 정책을 복사한뒤 정책 편집 부분에 붙여넣은 후 변경 사항 저장 버튼을 눌러 저장한다.



- AWS 서비스 검색창에 IAM 검색 → 사용자 클릭



- **보안 자격 증명** 클릭 → **액세스 키 발급** 클릭

## 요약

ARN  
am:awsiam::120626585696:user/shabit-user

생성됨  
February 13, 2023, 17:33 (UTC+09:00)

콘솔 액세스  
MFA 없이 활성화됨

마지막 콘솔 로그인  
① 안됨

액세스 키 1  
활성화되지 않음

액세스 키 2  
활성화되지 않음

권한

그룹

태그

보안 자격 증명

액세스 관리자

## 권한 정책 (1)

사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.



재가

권한 추가 ▼

## 액세스 키 (0)

액세스 키를 사용하여 AWS CLI, AWS Tools for PowerShell, AWS SDK 또는 직접 AWS API 호출을 통해 AWS에 프로그래밍 방식 호출을 전송합니다. 한 번에 최대 두 개의 액세스 키(활성 또는 비활성)를 가질 수 있습니다. [Learn more](#)

액세스 키 만들기

## 액세스 키 없음

액세스 키와 같은 장기 자격 증명을 사용하지 않는 것이 모범 사례입니다. 대신 단기 자격 증명을 제공하는 도구를 사용하세요. [Learn more](#)

액세스 키 만들기

## 액세스 키 모범 사례 및 대안

보안 개선을 위해 액세스 키와 같은 장기 자격 증명을 사용하지 마세요. 다음과 같은 사용 사례와 대안을 고려하세요.

☐ Command Line Interface(CLI)

AWS CLI를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ 로컬 코드

로컬 개발 환경의 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 컴퓨팅 서비스에서 실행되는 애플리케이션

Amazon EC2, Amazon ECS 또는 AWS Lambda와 같은 AWS 컴퓨팅 서비스에서 실행되는 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ 서드 파티 서비스

AWS 리소스를 모니터링 또는 관리하는 서드 파티 애플리케이션 또는 서비스에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 외부에서 실행되는 애플리케이션

애플리케이션을 온프레미스 호스트에서 실행하거나 로컬 AWS 클라이언트 또는 서드 파티 AWS 클라이언트를 사용할 수 있도록 이 액세스 키를 사용할 것입니다.

☒ 기타

귀하의 사용 사례가 여기에 나열되어 있지 않습니다.

## 설명 태그 설정 - 선택 사항

이 액세스 키에 대한 설명은 이 사용자에게 태그로 연결되고, 액세스 키와 함께 표시됩니다.

## 설명 태그 값

이 액세스 키의 용도와 사용 위치를 설명합니다. 좋은 설명은 나중에 이 액세스 키를 자신있게 교체하는 데 유용합니다.

최대 256자까지 가능합니다. 허용되는 문자는 문자, 숫자, UTF-8로 표현할 수 있는 공백 및 \_ : / = + - @입니다.

취소

이전

액세스 키 만들기

## 외부 서비스 → application.yml 설정

## 1. S3 Bucket

```
cloud:
  aws:
    s3:
      bucket: talktopia
      region:
        static: ap-northeast-2 #Asia Pacific -> seoul
      stack:
        auto: false
      credentials:
        access-key: AAAAAAAAAAAAAAAAAAAAAA
        secret-key: AAAAAAAAAAAAAAAAAAAAAA
```

## 2. Openvidu

```
openvidu:
  url: https://talktopia.site:8443/
  secret: ~~~~~
```

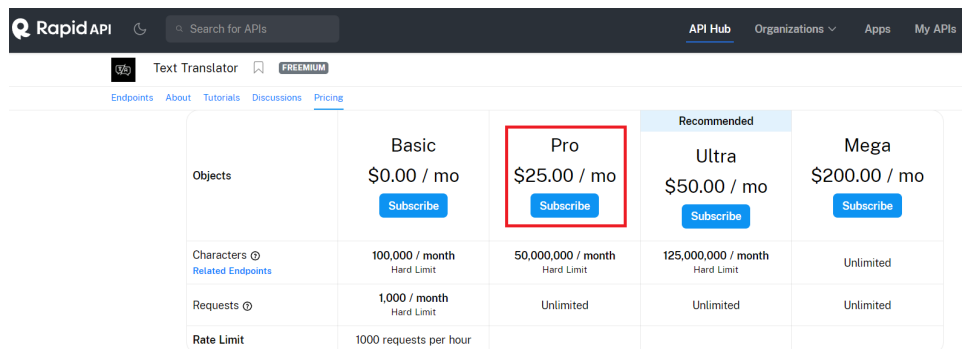
## 3. Naver Mail

```
spring:
  mail:
    username: ~~~~~
    password: ~~~~~
```

## 외부 서비스 → 프론트엔드 STT, Text TransLator

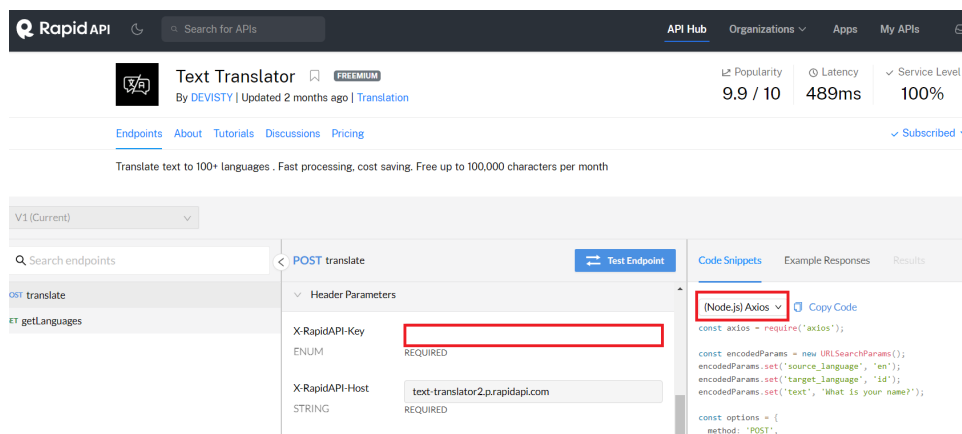
### 1. Text Translator

1. <https://rapidapi.com/dickyagustin/api/text-translator2> 이동후 subscribe



	Basic	Pro	Ultra	Mega
Objects	\$0.00 / mo	\$25.00 / mo	\$50.00 / mo	\$200.00 / mo
Characters	100,000 / month	50,000,000 / month	125,000,000 / month	Unlimited
Requests	1,000 / month	Unlimited	Unlimited	Unlimited
Rate Limit	1000 requests per hour			

2. Endpoints 이동 후, translate의 POST 요청에서 본인의 X-RapidAPI-Key 확인 및 Code Snippets에서 (Node.js)Axios 선택



Text Translator By DEVISTY | Updated 2 months ago | Translation

Popularity: 9.9 / 10 Latency: 489ms Service Level: 100%

Endpoints: About: Tutorials: Discussions: Pricing

Translate text to 100+ languages. Fast processing, cost saving. Free up to 100,000 characters per month

V1 (Current)

Search endpoints

POST translate

Header Parameters

X-RapidAPI-Key: REQUIRED

X-RapidAPI-Host: text-translator2.p.rapidapi.com

Code Snippets

(Node.js) Axios

```
const axios = require('axios');

const encodedParams = new URLSearchParams();
encodedParams.set('source_language', 'en');
encodedParams.set('target_language', 'id');
encodedParams.set('text', 'What is your name?');

const options = {
  method: 'POST',
  url: 'https://text-translator2.p.rapidapi.com/translate',
```

3. Code Snippets의 코드를 사용하여 axios 요청

```
const axios = require('axios');

const encodedParams = new URLSearchParams();
encodedParams.set('source_language', 'en');
encodedParams.set('target_language', 'id');
encodedParams.set('text', 'What is your name?');

const options = {
  method: 'POST',
  url: 'https://text-translator2.p.rapidapi.com/translate',
```

```

headers: {
  'content-type': 'application/x-www-form-urlencoded',
  'X-RapidAPI-Key': '~~~~~',
  'X-RapidAPI-Host': 'text-translator2.p.rapidapi.com'
},
data: encodedParams,
};

try {
  const response = await axios.request(options);
  console.log(response.data);
} catch (error) {
  console.error(error);
}

```

X-RapidAPI-Key는 보안을 위해 .env파일에 저장

## 2. STT

### 1. speech-recognition 공식문서 참조

[https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Speech\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API)

```

const recognitionInstance = new window.webkitSpeechRecognition();
recognitionInstance.continuous = true;
recognitionInstance.lang = user.sttLang;    // 음성 인식되는 언어
recognitionInstance.interimResults = true;
recognitionInstance.maxAlternatives = 1;

// speech recognition 이벤트핸들러 시작
recognitionInstance.onspeechend = function () {
  recognitionInstance.stop();
  console.log("speech end")
}

recognitionInstance.onerror = function (event) {
  console.log('Error occurred in recognition: ' + event.error);
}

recognitionInstance.onaudiostart = function (event) {
  //Fired when the user agent has started to capture audio.
  console.log('onaudiostart');
}

recognitionInstance.onaudioend = function (event) {
  //Fired when the user agent has finished capturing audio.
  console.log('onaudioend');
}

recognitionInstance.onend = function (event) {
  //Fired when the speech recognition service has disconnected.
  if (recognitionEnableRef.current) {
    recognitionInstance.start();
    console.log('재연결 되었습니다.')
  } else {
    console.log('한 문장이 인식이 끝났습니다. ');
  }
  // console.log('한 문장이 인식이 끝났습니다. ');
}

recognitionInstance.onnomatch = function (event) {
  //Fired when the speech recognition service returns a final result with no significant recognition. This may involve s
  console.log('nomatch');
}

recognitionInstance.onsoundstart = function (event) {
  //Fired when any sound – recognisable speech or not – has been detected.
  console.log('on sound start');
}

recognitionInstance.onsoundend = function (event) {
  //Fired when any sound – recognisable speech or not – has stopped being detected.
  console.log('on sound end');
}

recognitionInstance.onspeechstart = function (event) {
  //Fired when sound that is recognised by the speech recognition service as speech has been detected.
  console.log('onspeechstart');
}

recognitionInstance.onstart = function (event) {
  //Fired when the speech recognition service has begun listening to incoming audio with intent to recognize grammars as

```

```

        console.log('onstart');
    }

    // 음성 인식 결과 반환 시
    recognitionInstance.onresult = (e) => {
        // e.results 배열의 마지막 인덱스를 가져와서 처리합니다.
        const lastResult = e.results[e.results.length - 1];
        setTranscript(lastResult[0].transcript); // 내가 말한 내용

        // isFinal 프로퍼티를 통해 해당 결과가 최종 결과인지 확인합니다.
        if (lastResult.isFinal) { // true이면 말 다했기에 전달함수 호출
            sendMessage(lastResult[0].transcript); // 메시지 전달 함수 호출
        }
    };
};

```