

# TonnyBunny's Porting Manual

[기술 스택](#)

[배포 방법](#)

[EC2 설정](#)

[EC2 ssh로 접속](#)

[패키지 업데이트 및 도커 설치](#)

[MySQL 도커 컨테이너 실행](#)

[Redis 도커 컨테이너 실행](#)

[Jenkins 배포](#)

[Jenkins 도커 컨테이너 실행 및 계정 생성](#)

[Jenkins 프로젝트 설정](#)

[Jenkins 프로젝트와 Gitlab 연동 및 Webhook 설정](#)

[Gitlab Webhook 설정](#)

[Openvidu 배포](#)

[Openvidu Server Application 빌드 준비](#)

[Openvidu Server 도커 컨테이너 실행](#)

[Vuejs 프로젝트 배포 준비](#)

[환경 변수 설정](#)

[Nginx 설정 파일](#)

[Vuejs 프로젝트 실행 준비](#)

[Spring Boot 프로젝트 배포 준비](#)

[환경 변수 설정](#)

[Spring Boot 프로젝트 실행 준비](#)

[Jenkins 프로젝트 자동 배포](#)

[빌드 쉘 스크립트 작성](#)

[배포](#)

[Nginx 리버스 프록시 설정](#)

[Nginx 설치](#)

[Nginx로 리버스 프록시 설정](#)

[SSL 인증](#)

[방화벽 설정](#)

[배포 시 주의사항](#)

[데이터 베이스 접속 정보](#)

[Redis 접속 정보](#)

[MySQL 접속 정보](#)

[외부 서비스 정보](#)

[Naver SMS](#)

[아키텍처 구조](#)

## 기술 스택

### Deploy Server

- [AWS EC2](#)
- [Ubuntu](#) 20.04
- [Docker](#)
- [Docker-compose](#)
- [Nginx](#)

### API Docs

### Front-End

- [Nodejs](#) 18.12.1
- [Vue](#) 3.2.13
- [Vue router](#) 4.1.6
- [Vuex](#) 4.0.2
- [Vuetify](#) 3.1.3
- [Vue3-lottie](#) 2.4.0
- [Axios](#) 1.2.2

### IDE

- [IntelliJ](#) 2022.3.1
- [Visual Studio Code](#) 1.75.1

### Communication

- [Notion](#)
- [Discord](#)
- [Mattermost](#)

- `Swagger`

- `Sockjs-client` 1.6.1
- `Toastify-js` 1.12.0
- `Webstomp-client` 1.2.6
- `Bootstrap` 5

## | 형상 관리

- `Gitlab`
- `SourceTree`

## | Back-End

- `Spring Boot` 3.0.0
- `Lombok` 1.18.24
- `Spring Security` 2.7.7
- `Spring Data JPA` 2.7.7
- `Spring Data Redis` 2.7.7
- `Spring Boot Websocket` 2.7.7
- `Swagger-ui` 3.0.0
- `Stomp-websocket` 2.3.4
- `JWT` 0.9.1

## | 이슈 관리

- `Jira`

## | 기획

- `Figma`
- `ERD Cloud`

## | 그 외 툴

- `MobaXTerm`

## | DB

- `MySQL` 8.0.31
- `MySQL Workbench`
- `Redis` 5.0.3

# 배포 방법

## EC2 설정

### EC2 ssh로 접속

1. ssafy에서 제공해준 ec2 private key를 받는다.
2. private key가 있는 디렉토리에서 cmd창을 연다.
3. `ssh -i I8E105T.pem ubuntu@[도메인 주소]` 로 ec2에 접속하고, 'yes'를 입력한다.

```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22621.1105]
(c) Microsoft Corporation. All rights reserved.

C:\Users\JCW>ssh -i I8E105T.pem ubuntu@i8E105.p.ssafy.io|
```

4. 로그인 아이디를 입력하라는 문구가 나오면 `ubuntu`를 입력한다.

```
ubuntu@ip-172-26-0-190: ~
Swap usage: 0%
Processes: 162
Users logged in: 1
IPv4 address for br-16d8bdd0edd8: 172.18.0.1
IPv4 address for br-6e4742e587a1: 172.19.0.1
IPv4 address for docker0: 172.17.0.1
IPv4 address for eth0: 172.26.0.190

* Ubuntu Pro delivers the most comprehensive open source security and
compliance features.

https://ubuntu.com/aws/pro

136 updates can be installed immediately.
9 of these updates are security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

3 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log

*** System restart required ***
Last login: Thu Jan 26 12:19:02 2023 from 223.62.222.142
ubuntu@ip-172-26-0-190:~$
```

## 패키지 업데이트 및 도커 설치

- ubuntu 패키지를 업데이트하고, 사전 패키지를 설치한다.

```
sudo apt update
sudo apt-get install -y ca-certificates curl software-properties-common apt-transport-https gnupg lsb-release
```

- 도커 설치를 위해 linux 용 gpg key를 다운로드 받는다.

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

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release
```

- ubuntu에 도커를 설치한다.

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

## MySQL 도커 컨테이너 실행

- docker-compose.yml 파일을 생성한다.

```
version: "3.5"

services:
  tb_mysql:
    image: mysql:8.0.31
    container_name: tb_mysql
    volumes:
      - /home/ubuntu/docker/mysql/mysql:/var/lib/mysql
      - /home/ubuntu/docker/mysql/mysql_custom.cnf:/etc/mysql/conf.d/custom.cnf
    restart: always
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    environment:
      - MYSQL_ROOT_PASSWORD={비밀번호}
      - MYSQL_DATABASE=tonnybunny
    ports:
      - "3306:3306"
    expose:
      - "3306"
    networks:
      - tb_network

networks:
  tb_network:
    driver: bridge
    external: true
```

- mysqld-custom.cnf 파일을 생성한다.
  - 문자 인코딩 방식 설정
  - 타임존을 UTC에서 KST(UTC+9)로 변경

```
[mysqld]
default_time_zone = '+09:00'
character-set-server = utf8
collation-server = utf8_unicode_ci
skip-character-set-client-handshake
```

- mysql 도커를 실행한다.

```
> sudo docker-compose -f {yml 파일 이름} up -d
```

## Redis 도커 컨테이너 실행

- docker-compose.yml 파일을 생성한다.
  - 비밀번호를 설정한다.

```
version: '3.5'
services:
  tb_redis:
    image: redis:5.0.3
    container_name: tb_redis
    hostname: tb_redis
    labels:
      - "name=redis"
```

```

- "mode=standalone"
#network_mode: host
ports:
- 6379:6379
volumes:
- /etc/docker/redis-5.0.3/6001:/data
command: redis-server --requirepass {비밀번호} --port 6379
networks:
- tb_network
networks:
tb_network:
external: true

```

- redis를 실행한다.

```
> sudo docker-compose -f {yaml 파일 이름} up -d
```

## Jenkins 배포

### Jenkins 도커 컨테이너 실행 및 계정 생성

- docker-compose.yml 파일을 생성한다.

```

version: '3.5'

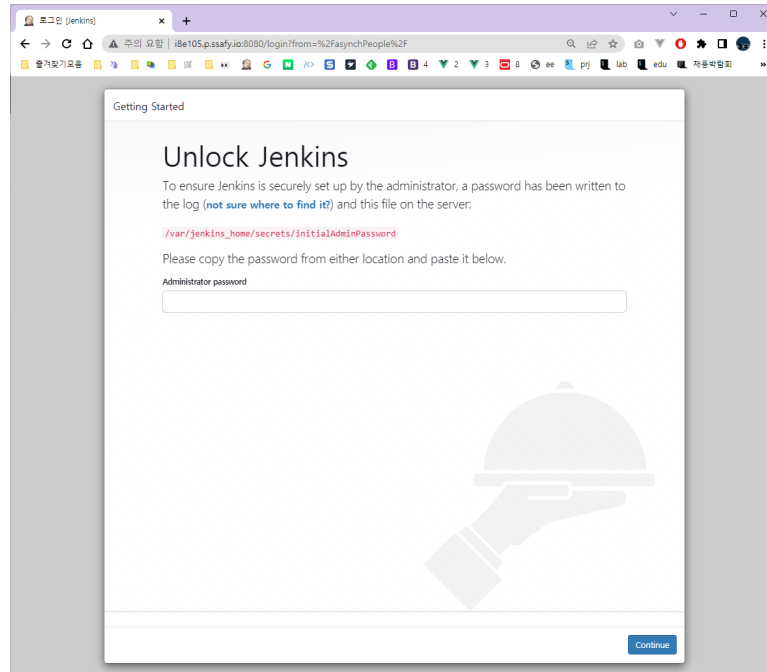
services:
# 서비스 명
jenkins:
# image: jenkins/jenkins:lts
container_name: jenkins
build: .
environment:
- JENKINS_OPTS=--httpPort=8080
volumes:
- /var/run/docker.sock:/var/run/docker.sock
- /home/ubuntu/tb/dev/tonnybunny/jenkins:/var/jenkins_home
- /home/ubuntu/tb/dev/config:/var/conf
- /home/ubuntu/tb/dev/tonnybunny/app:/app
ports:
- "8080:8080"
privileged: true
user: root

```

- Jenkins를 컨테이너를 실행한다.

```
> sudo docker-compose -f {yaml 파일 이름} up -d
```

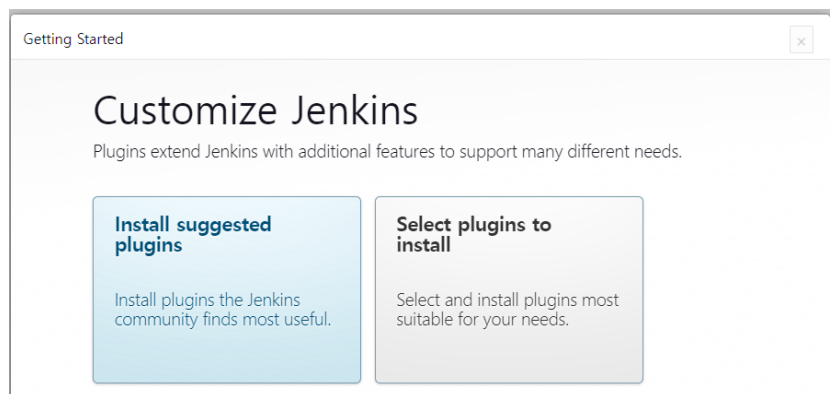
- [서버주소]:8080 ( [i8e105.p.ssafy.io:8080/](https://i8e105.p.ssafy.io:8080/) ) 으로 이동하면 Jenkins 로그인창이 나온다.



- `sudo docker logs jenkins` 명령어로 관리자 비밀번호를 찾아서 창에 입력한다.

```
ubuntu@ip-172-26-0-190: ~$ sudo docker logs jenkins
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
567bca4913134bd680acab0015bdbb31
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
*****
*****
2023-01-26 13:26:34.172+0000 [id=33] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
2023-01-26 13:26:34.193+0000 [id=23] INFO hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
2023-01-26 13:26:34.951+0000 [id=49] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
2023-01-26 13:26:34.952+0000 [id=49] INFO hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1
ubuntu@ip-172-26-0-190:~$
```

- `Install Suggested plugins` 를 선택한다.



- 플러그인들이 설치되고 나면, 관리자 계정을 만들고 `save and continue` > `save and finish` > `start using jenkins` 를 누른다.

Getting Started

## Create First Admin User

계정명  
tb

암호  
\*\*\*\*\*

암호 확인  
\*\*\*\*\*

이름  
jcw

이메일 주소  
codnjs0221@naver.com

Jenkins 2.375.2

Skip and continue as admin

Save and Continue

## Jenkins 프로젝트 설정

1. Jenkins 관리 > 플러그인 관리를 누른다.

+ 새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중

2 대기 중

### Jenkins 관리

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

Set up agent Set up cloud Dismiss

#### System Configuration

시스템 설정

환경변수 및 경로 정보등을 설정합니다.

플러그인 관리

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

Global Tool Configuration

Configure tools, their locations and automatic installers.

노드 관리

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

2. Gitlab 관련 플러그인을 설치한다. > **Install without restart** 를 누른다.

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

## Plugins

Q gitlab /

Install	Name ↓	Released
✓	<b>GitLab</b> 1.6.0 <a href="#">Build Triggers</a> This plugin allows <b>GitLab</b> to trigger Jenkins builds and display their results in the GitLab UI. This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	2 mo 15 days ago
✓	<b>Generic Webhook Trigger</b> 1.86.2 <a href="#">notification</a> <a href="#">github</a> <a href="#">webhook</a> <a href="#">Build Parameters</a> <a href="#">gitlab</a> <a href="#">Build Triggers</a> Can receive any HTTP request, extract any values from JSON or XML and trigger a job with those values available as variables. Works with GitHub, GitLab, Bitbucket, Jira and many more.	21 days ago
✓	<b>GitLab API</b> 5.0.1-78.v47a_45b_9f78bb_7 <a href="#">Library plugins (for use by other plugins)</a> This plugin provides <b>GitLab API</b> for other plugins.	6 mo 5 days ago
✓	<b>GitLab Authentication</b> 1.16 <a href="#">Authentication and User Management</a> This is the an authentication plugin using gitlab OAuth. This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	9 mo 12 days ago

[Install without restart](#)
[Download now and install after restart](#)
 Update information obtained: 20 min ago

3. 마찬가지로 Docker 관련 플러그인을 설치한다.

## Plugins

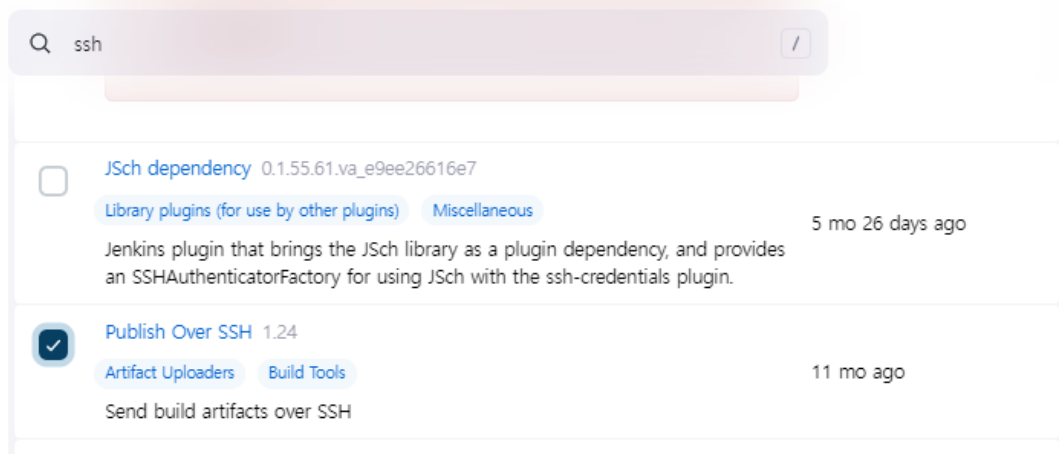
Q docker /

Install	Name ↓	Released
✓	<b>Docker</b> 1.3.0 <a href="#">Cloud Providers</a> <a href="#">Cluster Management</a> <a href="#">docker</a> This plugin integrates Jenkins with <b>Docker</b> This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	1 day 14 hr ago
✓	<b>Docker Commons</b> 1.21 <a href="#">Library plugins (for use by other plugins)</a> <a href="#">docker</a> Provides the common shared functionality for various Docker-related plugins.	5 mo 3 days ago
✓	<b>Docker Pipeline</b> 563.vd5d2e5c4007f <a href="#">pipeline</a> <a href="#">DevOps</a> <a href="#">Deployment</a> <a href="#">docker</a> Build and use Docker containers from pipelines. This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	1 mo 25 days ago
✓	<b>Docker API</b> 3.2.13-37.vf3411c9828b9 <a href="#">Library plugins (for use by other plugins)</a> <a href="#">docker</a> This plugin provides <b>docker-java</b> API for other plugins. This plugin is up for adoption! We are looking for new maintainers. Visit our <a href="#">Adopt a Plugin</a> initiative for more information.	9 mo 12 days ago

[Install without restart](#)
[Download now and install after restart](#)
 Update information obtained: 21 min ago

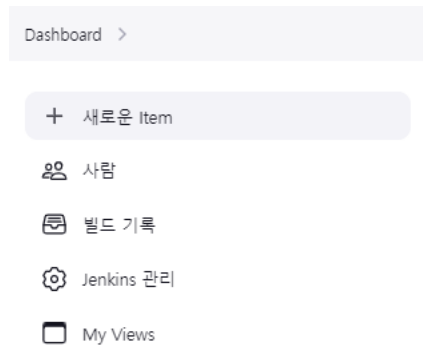
4. 마찬가지로 SSH 관련 플러그인을 설치한다.





## Jenkins 프로젝트와 Gitlab 연동 및 Webhook 설정

1. 새로운 item 을 클릭한다.




2. 생성할 item의 이름을 입력하고, Freestyle project 를 클릭한다.


Enter an item name

tonnybunny


» Required field


**Freestyle project**


이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.


**Pipeline**


Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


**Multi-configuration project**


다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.


**Folder**

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.


**Multibranch Pipeline**

Creates a set of Pipeline projects according to detected branches in one SCM repository.


**Organization Folder**

Creates a set of multibranch project subfolders by scanning for repositories.

OK

3. Git 탭에 만들어둔 gitlab 레포지토리의 url을 복사해서 붙여넣기, 빌드할 브랜치를 적는다.

(아직 Credential을 등록하지 않았으므로 에러 메시지가 나타나면 정상이다.)

Git ?

Repositories ?

Repository URL ?

`https://lab.ssafy.com/s08-webmobile1-sub2/S08P12E105.git`

Failed to connect to repository : Command "git ls-remote -h -- https://lab.ssafy.com/s08-webmobile1-sub2/S08P12E105.git HEAD" returned status code 128:  
stdout:  
stderr: remote: HTTP Basic: Access denied. The provided password or token is incorrect or your account has 2FA enabled and you must use a personal access token instead of a password. See [https://lab.ssafy.com/help/topics/git/troubleshooting\\_git#error-on-git-fetch-http-basic-access-denied](https://lab.ssafy.com/help/topics/git/troubleshooting_git#error-on-git-fetch-http-basic-access-denied)  
fatal: Authentication failed for 'https://lab.ssafy.com/s08-webmobile1-sub2/S08P12E105.git/'

Credentials ?

- none -

+ Add

고급...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

`*/develop`

Add Branch

4. Credentials의 Add 버튼 > Jenkins를 클릭한다.
5. 계정을 입력한다. (ID는 Credential 을 구분하기 위한 것이므로 아무거나 적으면 된다.)

## Jenkins Credentials Provider: Jenkins

### Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope ?

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

Username ?

jcw

☐ Treat username as secret ?

Password ?

\*\*\*\*\*

ID ?

jcw

- Username : 싸피깃 아이디
- Password : 싸피깃 비밀번호

6. 추가한 Credential을 선택하고, 에러 메시지가 없어지는 걸 확인한다.

Repositories ?

Repository URL ?

https://lab.ssafy.com/s08-webmobile1-sub2/S08P12E105.git

Credentials ?

jcw/\*\*\*\*\*

- none -

jcw/\*\*\*\*\*

7. 빌드 유발 탭에서 아래와 같이 선택한다. 언제 빌드를 할지 트리거 이벤트를 설정한다.

## 빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://i8e105.p.ssafy.io:8080/project/tonnybunny> ?

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request ?

☐ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

☒ Approved Merge Requests (EE-only)

☒ Comments

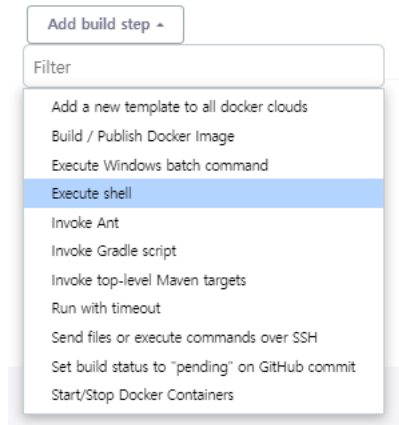
8. 빌드 유발 탭 > 고급 버튼을 누른 후, Secret Token을 받아서 저장해둔다.

Secret token ?

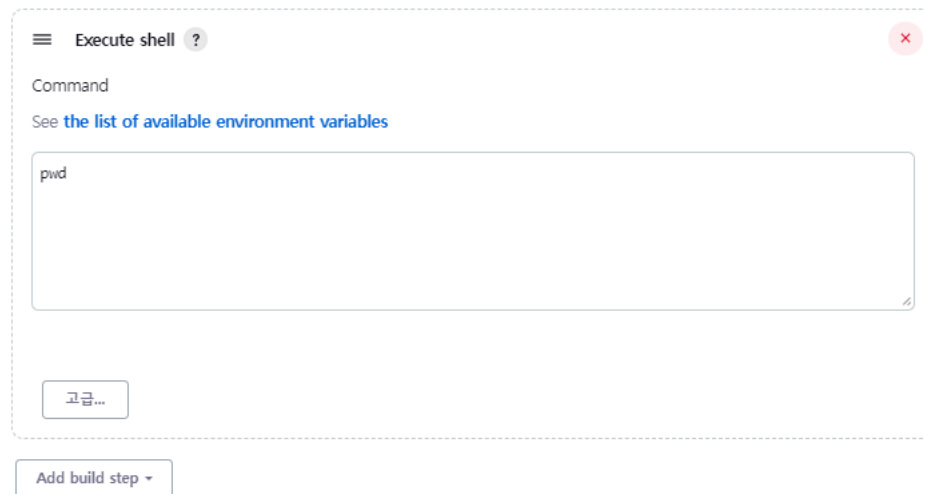
e63a0a1e9b06f418f1005ac89f71949e

Generate

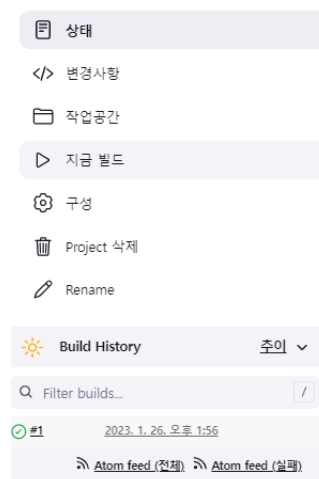
9. Build Steps > Add build step > Execute shell을 누른 후, `pwd` 를 입력하고 저장한다.



## Build Steps



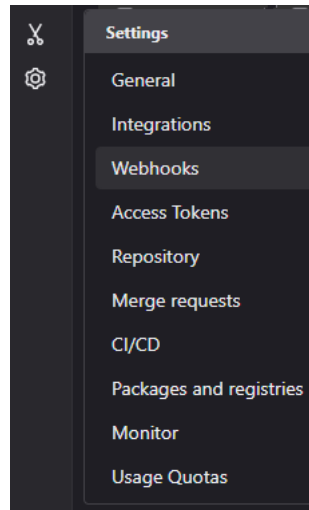
10. 지금 빌드 버튼을 누르고, 젠킨스 프로젝트 빌드가 잘 동작하는지 확인한다.



```
[tonnybunny] $ /bin/sh -xe /tmp/jenkins12331814307976433023.sh
+ pwd
/var/jenkins_home/workspace/tonnybunny
Finished: SUCCESS
```

## Gitlab Webhook 설정

1. Gitlab의 Webhook을 선택한다.



2. 아까 얻은 secret token을 입력하고, trigger를 설정한 후, **Add webhook** 을 클릭한다.

Search page

## Webhooks

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://배표공인ip:8090/project/deploytest/

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

**Secret token**

711253291c073f110c4b6b59cdb5803c

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

**Trigger**

☒ **Push events**

master

Push to the repository.

☐ Tag push events  
A new tag is pushed to the repository.

☐ Comments  
A comment is added to an issue or merge request.

☐ Confidential comments  
A comment is added to 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.

3. 생성된 Webhook에서 Test > Push Events를 선택한다.

☐ Releases events  
A release is created or updated.

**SSL verification**

☒ Enable SSL verification

Add webhook

Project Hooks (2)

http://i8e105.p.ssafy.io:8080/project/tonnybunny/

Push Events Merge Requests Events

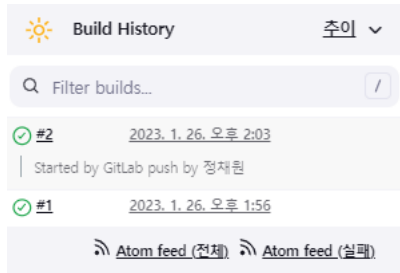
SSL Verification: enabled

Test Edit Delete

Hook executed successfully: HTTP 200

4. Jenkins에서 push event test가 잘 완료된 것을 알 수 있다.





## Openvidu 배포

### Openvidu Server Application 빌드 준비

- Dockerfile 을 생성한다.

```
FROM node:18.12.1
COPY package*.json ./
RUN mkdir ./OPENVIDU_SERVER/
COPY . ./OPENVIDU_SERVER/
RUN npm install

EXPOSE 5000
```

- 빌드와 실행에 필요한 파일을 로컬에서 jenkins 컨테이너로 복사한다.
  - /home 안으로 필요한 파일을 미리 복사해두고, gitlab에서 pull한 코드 안으로 필요한 파일을 이동시킨다.

```
# fe - openvidu
sudo docker cp {Dockerfile 경로} jenkins:/home/fe/
```

- 도커 이미지 빌드 및 실행은 jenkins 컨테이너 내에서 수행한다.

### Openvidu Server 도커 컨테이너 실행

- Openvidu 설치 파일 다운로드

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

- Openvidu docker 설정 파일 수정
  - /opt/openvidu 에서 .env 파일 수정

```
# Whether to enable recording module or not
OPENVIDU_RECORDING=true

# Use recording module with debug mode.
OPENVIDU_RECORDING_DEBUG=false

# Openvidu Folder Record used for save the openvidu recording videos. Change it
# with the folder you want to use from your host.
OPENVIDU_RECORDING_PATH=/opt/openvidu/recordings
OPENVIDU_RECORDING_PATH=/home/ubuntu/resources/videos
```

```
# Domain name. If you do not have one, the public IP of the machine.
# For example: 199.51.100.1, or openvidu.example.com
DOMAIN_OR_PUBLIC_IP=i8e105.p.ssafy.io

# OpenVidu SECRET used for apps to connect to OpenVidu server and users to access to OpenVidu Dashboard
OPENVIDU_SECRET=TonyBunny555

# Certificate type:
# - selfsigned: Self signed certificate. Not recommended for production use.
#               Users will see an ERROR when connected to web page.
# - owncert:    Valid certificate purchased in a Internet services company.
#               Please put the certificates files inside folder ./owncert
#               with names certificate.key and certificate.cert
# - letsencrypt: Generate a new certificate using letsencrypt. Please set the
#               required contact email for Let's Encrypt in LETSENCRYPT_EMAIL
#               variable.

#CERTIFICATE_TYPE=selfsigned
#CERTIFICATE_TYPE=letsencrypt

# If CERTIFICATE_TYPE=letsencrypt, you need to configure a valid email for notifications
#LETSENCRYPT_EMAIL=test@example.com
#LETSENCRYPT_EMAIL=lalae022@gmail.com

# Proxy configuration
# If you want to change the ports on which openvidu listens, uncomment the following lines

# Allows any request to http://DOMAIN_OR_PUBLIC_IP:HTTP_PORT/ to be automatically
# redirected to https://DOMAIN_OR_PUBLIC_IP:HTTPS_PORT/
# WARNING: the default port 80 cannot be changed during the first boot
# if you have chosen to deploy with the option CERTIFICATE_TYPE=letsencrypt
#HTTP_PORT=80
#HTTPS_PORT=443

# Changes the port of all services exposed by OpenVidu.
# SDKs, REST clients and browsers will have to connect to this port
#HTTP_PORT=30000
#HTTPS_PORT=30001
```

- 실시간 화면 녹화 기능을 사용할 것이므로, OPENVIDU\_RECORDING을 true로 바꿔준다.

- letsencrypt 방식으로 ssl 인증을 해준다.
- PUBLIC\_IP에 도메인 주소를 적어준다.
- ec2 로컬에 설치된 nginx에서 80번과 443번 포트를 사용할 것이기 때문에, HTTP\_PORT와 HTTPS\_PORT의 주소를 30000, 30001번으로 변경하였다.
- Openvidu 도커 컨테이너 실행

```
sudo ./openvidu start
```

- 실행 결과 모습

```
openvidu-server_1 | -----
openvidu-server_1 |
openvidu-server_1 | OpenVidu is ready!
openvidu-server_1 | -----
openvidu-server_1 |
openvidu-server_1 | * OpenVidu Server URL: https://i8e105.p.ssafy.io:30001/
openvidu-server_1 |
openvidu-server_1 | * OpenVidu Dashboard: https://i8e105.p.ssafy.io:30001/dashboard
openvidu-server_1 | -----
openvidu-server_1 |
```

이렇게 콘솔창에 로그가 뜨면 ctrl+c를 눌러서 백그라운드에서 실행시킨다.

## Vuejs 프로젝트 배포 준비

### 환경 변수 설정

- `.env` 파일을 생성한다.
  - `VUE_APP_OPENVIDU_URL` 경로는 nginx reverse proxy로 설정(참고)한 openvidu server application의 경로이다.

```
VUE_APP_NODE_ENV="production"
VUE_APP_OPENVIDU_URL="https://i8e105.p.ssafy.io/opvclient/"
VUE_APP_SERVER_URL="https://i8e105.p.ssafy.io"
```

## Nginx 설정 파일

- Vuejs 프로젝트를 실행시킬 웹서버용 Nginx이다.
- nginx.conf 파일을 작성한다.

```
server {
    listen      5080 default_server;
    listen  [::]:5080 default_server;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html; # when 404 error
        add_header 'Access-Control-Allow-Origin' '*';
    }
}
```

## Vuejs 프로젝트 실행 준비

- Vuejs 프로젝트를 빌드할 Dockerfile을 만든다.

```
FROM node:18.12.1 as build-stage
WORKDIR /var/jenkins_home/workspace/tonnybunny/FE/tonny-bunny
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /var/jenkins_home/workspace/tonnybunny/FE/tonny-bunny/dist /usr/share/nginx/html
COPY --from=build-stage /var/jenkins_home/workspace/tonnybunny/FE/tonny-bunny/nginx.conf /etc/nginx/conf.d/default.conf
# For SSL certification
#RUN mkdir /etc/sslkey
#COPY --from=build-stage /var/jenkins_home/workspace/tonnybunny/FE/tonny-bunny/sslkey/fullchain.pem /etc/sslkey/fullchain.pem
#COPY --from=build-stage /var/jenkins_home/workspace/tonnybunny/FE/tonny-bunny/sslkey/privkey.pem /etc/sslkey/privkey.pem
# COPY /var/conf/nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 5080
CMD ["nginx", "-g","daemon off;"]
```

- 빌드와 실행에 필요한 파일을 로컬에서 jenkins 컨테이너로 복사한다.
  - /home 안으로 필요한 파일을 미리 복사해두고, gitlab에서 pull한 코드 안으로 필요한 파일을 이동시킨다.

```
sudo docker exec jenkins mkdir -p /home/fe/

# fe - vue
sudo docker cp {.env 파일} jenkins:/home/fe/.env
```

## Spring Boot 프로젝트 배포 준비

### 환경 변수 설정

- `env.properties` 파일을 생성한다.

```
# Preference - File Encodings - Default encoding for properties files -> UTF-8 (checkbox check)
# server Settings
env.dev.port=10000

# Mysql DataBase Settings
dev.mysql.driver=com.mysql.cj.jdbc.Driver
dev.mysql.url=jdbc:mysql://i8e105.p.ssafy.io:3306/tonnybunny?autoReconnect=true&useUnicode=true&characterEncoding=utf-8
dev.mysql.username=root
```

```

dev.mysql.password={mysql 비밀번호}

# tonnybunnytonnybunnycarrotcarrot base64 key
auth.accesskey=dG9ubnlldW5ueXRvbm55YnVubnljYXJyb3RjYXJyb3QK

# chunghyeonheijeseungtaejeongachawonkwonmin key
auth.refreshkey=Y2h1bmdoewVvbmhlaWp lc2V1bmd0YWVqZW9uZ2FjaGF3b25rd29ubWLuCg
auth.datakey=seq

# Image Path Setting
app.file.path=/var/resources/images

# Naver SMS Service
naver-cloud-sms.accesskey={발급받은 access key}
naver-cloud-sms.secretkey={발급받은 secret key}
naver-cloud-sms.serviceid={발급받은 service id}
naver-cloud-sms.senderphone={인증 문자 보낼 전화번호}!

# JPA
dev.ddl.option=none
sql.init.mode=always
jpa.show.sql=false

# Redis
db.redis.host=tb_redis
db.redis.port=6379
db.redis.password={redis 비밀번호}

```

- `application.yml` 파일은 로컬 환경(local), 개발 환경(dev), 배포 환경(prod)에 따라 다르게 설정한다.
  - `application-dev.yml` 를 설정한다.

```

server:
  port: ${env.dev.port}

# java -jar -Dspring.profiles.active=dev *.jar
spring:
  config:
    activate:
      on-profile: dev
  # MySQL
  datasource:
    driver-class-name: ${dev.mysql.driver}
    url: ${dev.mysql.url}
    username: ${dev.mysql.username}
    password: ${dev.mysql.password}
  # Jpa
  jpa:
    hibernate.ddl-auto: ${dev.ddl.option}
  sql:
    init:
      mode: ${sql.init.mode}
      schema-locations: classpath:schema.sql
      data-locations: classpath:data.sql
  redis:
    password: ${db.redis.password}

```

## Spring Boot 프로젝트 실행 준비

- Spring Boot 프로젝트를 빌드할 Dockerfile을 생성한다.

```

FROM adoptopenjdk/openjdk11 AS builder
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
# COPY /var/conf/env.properties src/main/resources/properties/
RUN chmod +x ./gradlew
RUN ./gradlew bootJAR

```

```
FROM adoptopenjdk/openjdk11
COPY --from=builder build/libs/*.jar app.jar
EXPOSE 10000
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=dev", "/app.jar"]
```

- 빌드와 실행에 필요한 파일을 로컬에서 jenkins 컨테이너로 복사한다.
  - /home 안으로 필요한 파일을 미리 복사해두고, gitlab에서 pull한 코드 안으로 필요한 파일을 이동시킨다.

```
sudo docker exec jenkins mkdir -p /home/be/

# be - spring
sudo docker cp ~/dockers/be/env.properties jenkins:/home/be/env.properties
```

## Jenkins 프로젝트 자동 배포

- Jenkins GUI에서 프로젝트 빌드를 위한 쉘 스크립트를 작성한다.
  - 빌드와 실행을 위해 필요한 파일을 jenkins 프로젝트 안으로 복사한다.
  - Vuejs, Spring boot, Openvidu Server Application 도커 이미지를 빌드하고 컨테이너를 실행한다.

### 빌드 쉘 스크립트 작성

- Vuejs 프로젝트 빌드

```
# fe
cp /home/fe/.env ./FE/tonny-bunny/

docker build -t frontimg -f ./FE/tonny-bunny/Dockerfile ./FE/tonny-bunny/
if (docker ps | grep "frontimg"); then docker stop frontimg; fi
if (docker ps -a | grep 'frontimg'); then docker rm frontimg; fi
docker run -itd -p 5080:5080 --name frontimg --net=tb_network frontimg
echo "Run testproject_react"
```

- Spring Boot 프로젝트 빌드

```
# be
cp /home/be/env.properties ./BE/src/main/resources/properties/

docker build -t backimg -f ./BE/Dockerfile ./BE
if (docker ps | grep "backimg"); then docker stop backimg; fi
if (docker ps -a | grep 'backimg'); then docker rm backimg; fi
docker run -itd -p 10000:10000 -v /home/ubuntu/resources:/var/resources --name backimg --net=tb_network backimg
echo "Run testproject"
```

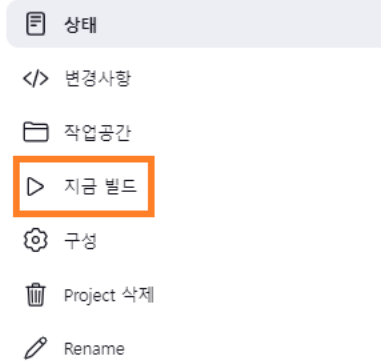
- Openvidu Server Application 빌드

```
# fe
cp /home/sslkey/fullchain.pem ./OPENVIDU_SERVER/opencvert.pem
cp /home/sslkey/privkey.pem ./OPENVIDU_SERVER/opencvdukey.pem

docker build -t ovimg -f ./OPENVIDU_SERVER/Dockerfile ./OPENVIDU_SERVER
if (docker ps | grep "ovimg"); then docker stop ovimg; fi
if (docker ps -a | grep 'ovimg'); then docker rm ovimg; fi
docker run -itd -p 5000:5000 --name ovimg --net=tb_network ovimg
echo "Run openvidu_server"
```

## 배포

- 수동으로 배포를 할 경우 아래의 버튼을 눌러준다.



- 앞서 Gitlab과 연동하여 Webhook을 설정했기 때문에, gitlab에서 Push나 MR 이벤트가 들어오면 위의 쉘 스크립트가 실행돼서 자동으로 서버가 배포된다.

## Nginx 리버스 프록시 설정

### Nginx 설치

- ec2 로컬에 nginx를 설치한다.

```
sudo apt install nginx
```

- nginx를 실행하기 전에 중복되는 포트를 가진 openvidu는 잠시 멈춰준다.

```
cd /etc/openvidu
sudo ./openvidu stop
```

- nginx 서비스를 실행시킨다.

```
sudo service nginx start
```

### Nginx로 리버스 프록시 설정

- `/etc/nginx/site-available` 에서 reverse-nginx.conf 파일을 생성한다. 아래와 같이 작성한다.
  - 정적 리소스 접근 경로 설정함
  - Vuejs, Spring boot server, Openvidu server로 접근하는 경로를 설정함

```
server {
    listen 80;
    listen [::]:80;

    server_name i8e105.p.ssafy.io;

    return 301 https://$server_name$request_uri;
}

server {
```

```

listen 443 ssl;
listen [::]:443 ssl;
access_log /home/ubuntu/nginx/reverse-access.log;
error_log /home/ubuntu/nginx/reverse-error.log;

ssl_certificate      /etc/nginx/sslkey/fullchain.pem;
ssl_certificate_key  /etc/nginx/sslkey/privkey.pem;

location / {
    proxy_pass http://localhost:5080/;
    proxy_redirect default;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';

    # wss setting
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Origin "";
}

location /images {
    #alias /home/ubuntu/resources/images;
    root /home/ubuntu/resources;
}

location /videos {
    #alias /home/ubuntu/resources/videos;
    root /home/ubuntu/resources;
}

location /api {
    proxy_pass http://localhost:10000/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;
    add_header 'Access-Control-Allow-Origin' '*';
    # wss setting
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Origin "";
}

location /opvclient/ {
    proxy_pass http://localhost:5000/;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';
}
}

```

- nginx 서비스를 reload 또는 restart 를 한다.

```

> sudo service nginx reload
or
> sudo service nginx restart

```

- nginx가 정상 동작하는지 테스트한다.

```

> sudo nginx -t

```

```

ubuntu@ip-172-26-0-190:/etc/nginx/sites-available$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

```

## SSL 인증

- letsencrypt 설치

```
# 인증서 발급
sudo letsencrypt certonly --standalone -d {도메인}
# 이메일 쓰고 Agree
# 뉴스레터 no
# 이제 인증서가 발급된다. 이 인증서를 잘보관하자
```

- 인증키 발급

- 인증키를 적절한 위치로 이동

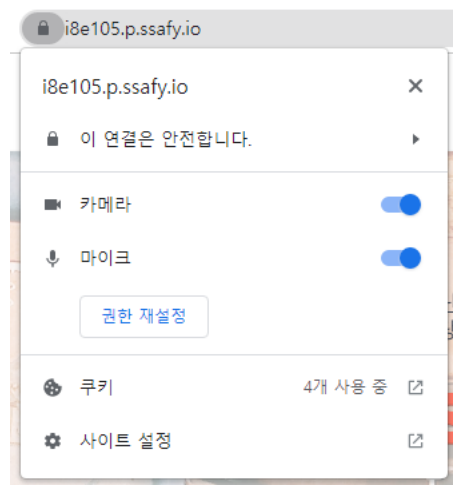
```
# 2가지 키가 발급되는데 이 두가지를 써야한다. 밑의 경로에 각각 하나씩 있다.
ssl_certificate /etc/letsencrypt/live/{도메인}/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/{도메인}/privkey.pem;
```

- Nginx의 conf파일에 ssl 인증 설정을 추가
- Nginx 서비스 reload해서 적용

```
# ...
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    access_log /home/ubuntu/nginx/reverse-access.log;
    error_log /home/ubuntu/nginx/reverse-error.log;

    ssl_certificate      /etc/nginx/sslkey/fullchain.pem;
    ssl_certificate_key  /etc/nginx/sslkey/privkey.pem;
    # ...
}
```

- https 연결 인증된 모습



## 방화벽 설정

- ufw 설치



```
sudo apt install ufw
```

- ec2 원격 접속을 위한 22(ssh) 포트 허용
  - 원격을 접속을 하기 위해서는 ufw 방화벽을 enable하기 전에 반드시 ssh포트를 먼저 허용해야 한다.

```
sudo ufw allow 22
```

- http(80), https(443), Openvidu Server Application(5000), Openvidu Server(30001) 통신을 위한 포트 허용

```
sudo ufw allow 80
sudo ufw allow 443
sudo ufw allow 5000
sudo ufw allow 30001
```

- 방화벽 켜기

```
sudo ufw enable
```

- 방화벽 상태 확인

```
sudo ufw status
```

```
ubuntu@ip-172-26-0-190:~$ sudo ufw status
Status: active

To Action From
--
22 ALLOW Anywhere
80 ALLOW Anywhere
443 ALLOW Anywhere
8080 ALLOW Anywhere
5000 ALLOW Anywhere
30001 ALLOW Anywhere
22 (v6) ALLOW Anywhere (v6)
80 (v6) ALLOW Anywhere (v6)
443 (v6) ALLOW Anywhere (v6)
8080 (v6) ALLOW Anywhere (v6)
5000 (v6) ALLOW Anywhere (v6)
30001 (v6) ALLOW Anywhere (v6)
```

## 배포 시 주의사항

1. Vuejs, Spring boot, Openvidu Server Application의 경우 Jenkins 프로젝트 안으로 Dockerfile이나 기타 설정파일들을 복사한 후, Jenkins 프로젝트 상에서 도커를 실행해야 하는 구조이다.

그래서 로컬에서 Jenkins로의 필요한 파일 복사 → Jenkins 프로젝트 내의 gitlab 소스코드 안으로 필요한 파일 복사 → Docker 빌드 및 컨테이너 실행 이라는 복잡한 과정을 거쳐서 배포되고 있다.

추후 구조의 개선이 필요한 부분이다.

2. Https 연결해야 한다.

Openvidu를 사용할 경우 Https 통신을 하기 때문에 반드시 ssl 인증을 해야 한다.

## 데이터 베이스 접속 정보

### Redis 접속 정보

- host : 서비스 이름 `tb_redis`
- port : 6379
- password : tonnybunny555

## MySQL 접속 정보

- host : 서비스 이름 `tb_mysql`
- port : 3306
- user : root
- password : tonnybunny555
- scheme : tonnybunny

## 외부 서비스 정보

### Naver SMS

- 회원가입 시 문자 인증 서비스를 제공한다.
- Spring boot 프로젝트의 env.properties에서 설정([참고](#))한다.

```
# Naver SMS Service
naver-cloud-sms.accesskey={발급받은 access key}
naver-cloud-sms.secretkey={발급받은 secret key}
naver-cloud-sms.serviceid={발급받은 service id}
naver-cloud-sms.senderphone={인증 문자 보낼 전화번호}!
```

## 아키텍처 구조

