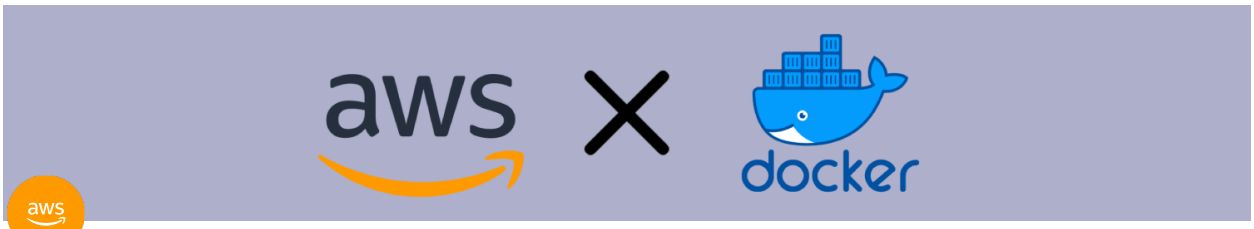


포팅 매뉴얼

프로젝트 기술 스택

Front-end

- React 17.0.2
- JavaScript (ES2017 == ES8)
- SCSS / HTML5
- Chrome / Edge
- Axios
- Node JS LTS
- Npm 8.11.0
- Redux
- ESLint
- Webpack
- 오픈소스 Editor
- Electron

Back-end

- JAVA 1.8
- Spring Boot 2.7.1
- Spring Security
- MariaDB 5.6.47.0
- Hibernate
- Docker
- NginX
- Swagger 3.0.0
- Gradle
- OpenVidu 2.22.0
- Kurento-media-server 6.18.0
- Jenkins

---

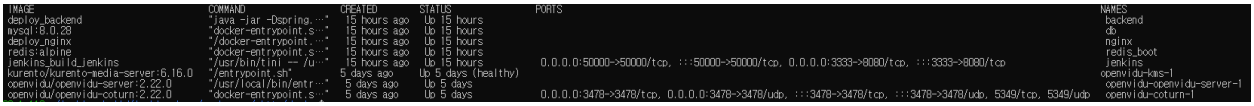
0. 서버 배포 준비

0.0 배포 구조

도커와 젠킨스를 이용한 자동 빌드 및 배포를 수행한다. 단, 프로젝트가 OpenVidu에 종속적이기 때문에 OpenVidu 컨테이너를 사전에 미리 설치해야 한다.

배포가 완료되었을 때, 컨테이너 환경은 다음과 같다.

- MySQL
- Backend (스프링 부트 컨테이너)
- NginX
- Redis
- Jenkins
- OpenVidu 서버 컨테이너
- OpenVidu Coturn 컨테이너
- Kurento Media Server 컨테이너

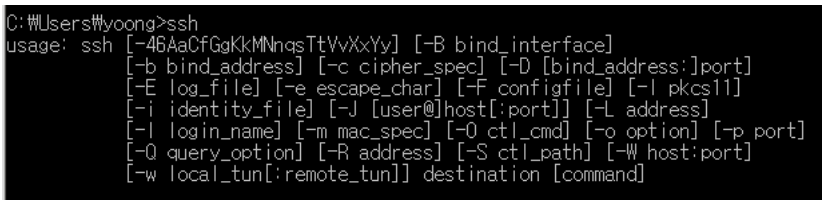


위 컨테이너 중 이미지 제작이 필요한 컨테이너는 Backend, Nginx 2개이다. 나머지는 이미 이미지파일이 존재하므로 다운받아서 사용하면 된다.

0.1 서버 접속하기

AWS 인스턴스에 접속할 수 있는 방법은 여러가지가 있다. 편리한 방법도 존재하지만, 이 메뉴얼에서는 SSH를 이용한, 명령어 위주의 방법으로 접속한다.

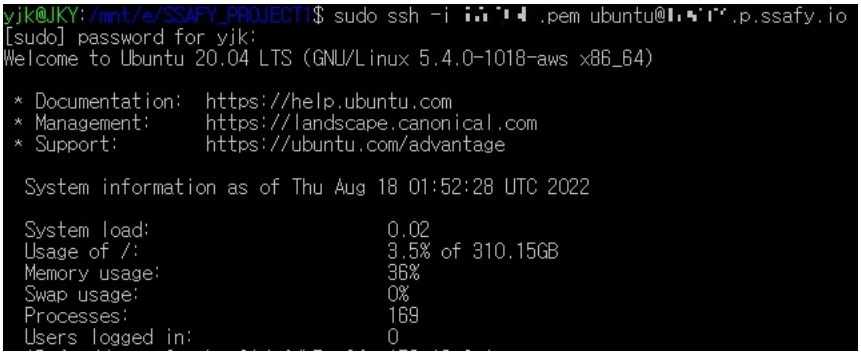
일단, 본인 컴퓨터에 SSH를 설치한다. 설치 방법은 따로 기술하지는 않는다. 설치가 완료되면, CMD에서 SSH 명령어를 입력해서 설치가 되었는지 체크한다.



그리고 지급받은 `키이름.pem` 파일이 있는 경로로 이동해서 다음과 같은 명령어를 입력한다.

```
ssh -i 키이름.pem ubuntu@(도메인 혹은 IP)
```

(위 명령어를 통해서 접속하기 위해서는 키 접근 권한을 줄여줘야 한다. 그래서 윈도우에서 진행하는 것 보다 우분투 등 WSL 기반 시스템을 설치해서 사용하는 것을 권장한다.)



성공적으로 접속한 화면

1. EC2에서 도커설치

프로젝트를 도커를 통한 컨테이너화 할 예정이기 때문에 EC2에는 도커와 도커 컴포즈 이외에 설치할 것은 없다. 다음과 같은 과정을 통해 도커를 설치한다.

먼저 설치에 필요한 사전 업데이트 및 설치를 진행한다.

```
sudo apt-get update
```

```
sudo apt install git
```

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

아래 명령어를 이후 콘솔에 출력되는 결과값이 'OK' 이면 성공이다.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

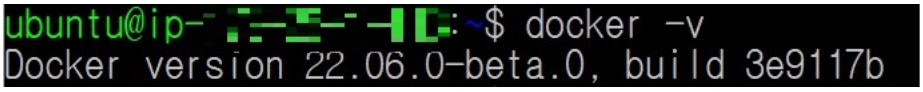
아래 명령어의 `arch=아키텍쳐` 에는 자신의 환경에 맞는 아키텍쳐를 적어줘야한다.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

도커 설치를 진행한다.

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

성공적으로 설치되었는지 확인한다.



도커를 실행한다.

```
sudo systemctl enable docker && service docker start
```

1-1 도커 On Off

- 도커 종료

```
sudo systemctl stop docker.socket
```

- 도커 실행

```
sudo systemctl start docker.socket
sudo systemctl enable docker && sudo service docker start
```

- 도커 상태 확인

```
service docker status
```

2. Docker-Compose 설치

도커 컴포즈는 여러개의 컨테이너의 실행과 관리를 할 수 있게 해준다. 오픈비두 설치를 위해서는 필수로 설치해야 한다.

```
sudo curl -L “https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)” -o /usr/local/bin/docker-c
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

2. OpenVidu 설치

배포 환경에서 OpenVidu 설치

포트 충돌을 피하기 위해서, 다른 컨테이너들을 실행시키기 전, OpenVidu를 먼저 EC2에 설치하는 것을 권장한다. 다음 내용은 위 링크의 내용을 의역한 것이다.

OpenVidu가 기본적으로 제공해주는 컨테이너는 총 5개이다.

- openvidu-openvidu-server-1
- openvidu-kms-1
- openvidu-coturn-1
- openvidu-nginx-1

- openvidu-app-1

이 중 nginx와 app은 사용하지 않는다. 그래서 맨 처음 설치 확인 용으로 한번 테스트 이후 컨테이너를 종료시킬 것이다,

2-1 전제조건

- docker 및 docker-compose 설치
- 도메인 이름 : HTTPS 사용해야 한다.

열어야 하는 포트

- **22 TCP:** SSH를 사용하여 관리자 OpenVidu에 연결
- **80 TCP:** NginX HTTP 포트
- **443 TCP:** NginX HTTPS 포트
- **3478 TCP+UDP:** 클라이언트 IP를 확인하기 위해 STUN/TURN 서버에서 사용합니다.
- **40000 - 57000 TCP+UDP:** Kurento Media Server에서 미디어 연결을 설정하는 데 사용합니다.
- **57001 - 65535 TCP+UDP :** 중계된 미디어 연결을 설정하기 위해 TURN 서버에서 사용합니다.

2-2 배포

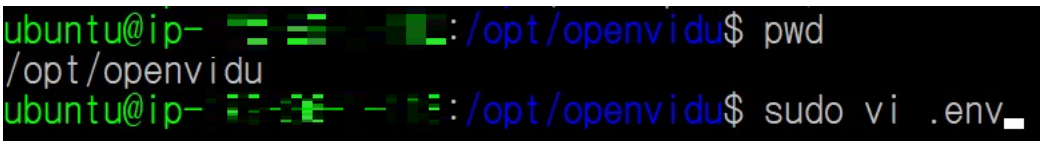
오픈비두 설치 권장 경로로 이동한다.

```
cd /opt
```

오픈비두 설치 파일을 다운받는다.

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

성공적으로 설치되었다면, OpenVidu 폴더로 이동해서 .env 파일을 수정한다.



필수적으로 설정해야 할 부분은 아래와 같다.

```
# Domain name. If you do not have one, the public IP of the machine.
# For example: 198.51.100.1, or openvidu.example.com
DOMAIN_OR_PUBLIC_IP=

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

프로젝트 실행에 필요한 설정은 다음과 같다.

▼ .env

```
# OpenVidu configuration
# ----------------------
# Documentation: https://docs.openvidu.io/en/stable/reference-docs/openvidu-config/

# NOTE: This file doesn't need to quote assignment values, like most shells do.
# All values are stored as-is, even if they contain spaces, so don't quote them.

# Domain name. If you do not have one, the public IP of the machine.
# For example: 198.51.100.1, or openvidu.example.com
DOMAIN_OR_PUBLIC_IP=도메인키

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

# 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

# If CERTIFICATE_TYPE=letsencrypt, you need to configure a valid email for notifications
LETSENCRYPT_EMAIL=user@example.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

# Changes the port of all services exposed by OpenVidu.
# SDKs, REST clients and browsers will have to connect to this port
HTTPS_PORT=443

# Old paths are considered now deprecated, but still supported by default.
# OpenVidu Server will log a WARN message every time a deprecated path is called, indicating
# the new path that should be used instead. You can set property SUPPORT_DEPRECATED_API=false
# to stop allowing the use of old paths.
# Default value is true
# SUPPORT_DEPRECATED_API=true

# If true request to with www will be redirected to non-www requests
# Default value is false
# REDIRECT_WWW=false

# How many workers to configure in nginx proxy.
# The more workers, the more requests will be handled
# Default value is 10240
# WORKER_CONNECTIONS=10240

# Access restrictions
# In this section you will be able to restrict the IPs from which you can access to
# Openvidu API and the Administration Panel
# WARNING! If you touch this configuration you can lose access to the platform from some IPs.
# Use it carefully.

# This section limits access to the /dashboard (OpenVidu CE) and /inspector (OpenVidu Pro) pages.
# The form for a single IP or an IP range is:
# ALLOWED_ACCESS_TO_DASHBOARD=198.51.100.1 and ALLOWED_ACCESS_TO_DASHBOARD=198.51.100.0/24
# To limit multiple IPs or IP ranges, separate by commas like this:
# ALLOWED_ACCESS_TO_DASHBOARD=198.51.100.1, 198.51.100.0/24
# ALLOWED_ACCESS_TO_DASHBOARD=

# This section limits access to the Openvidu REST API.
# The form for a single IP or an IP range is:
# ALLOWED_ACCESS_TO_RESTAPI=198.51.100.1 and ALLOWED_ACCESS_TO_RESTAPI=198.51.100.0/24
```

```
# To limit multiple IPs or or IP ranges, separate by commas like this:
# ALLOWED_ACCESS_TO_RESTAPI=198.51.100.1, 198.51.100.0/24
# ALLOWED_ACCESS_TO_RESTAPI=

# 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

# System path where OpenVidu Server should look for custom recording layouts
OPENVIDU_RECORDING_CUSTOM_LAYOUT=/opt/openvidu/custom-layout

# if true any client can connect to
# https://OPENVIDU_SERVER_IP:OPENVIDU_PORT/recordings/any_session_file.mp4
# and access any recorded video file. If false this path will be secured with
# OPENVIDU_SECRET param just as OpenVidu Server dashboard at
# https://OPENVIDU_SERVER_IP:OPENVIDU_PORT
# Values: true | false
OPENVIDU_RECORDING_PUBLIC_ACCESS=true

# Which users should receive the recording events in the client side
# (recordingStarted, recordingStopped). Can be all (every user connected to
# the session), publisher_moderator (users with role 'PUBLISHER' or
# 'MODERATOR'), moderator (only users with role 'MODERATOR') or none
# (no user will receive these events)
OPENVIDU_RECORDING_NOTIFICATION=publisher_moderator

# Timeout in seconds for recordings to automatically stop (and the session involved to be closed)
# when conditions are met: a session recording is started but no user is publishing to it or a session
# is being recorded and last user disconnects. If a user publishes within the timeout in either case,
# the automatic stop of the recording is cancelled
# 0 means no timeout
OPENVIDU_RECORDING_AUTOSTOP_TIMEOUT=10

# Maximum video bandwidth sent from clients to OpenVidu Server, in kbps.
# 0 means unconstrained
OPENVIDU_STREAMS_VIDEO_MAX_RECV_BANDWIDTH=1000

# Minimum video bandwidth sent from clients to OpenVidu Server, in kbps.
# 0 means unconstrained
OPENVIDU_STREAMS_VIDEO_MIN_RECV_BANDWIDTH=300

# Maximum video bandwidth sent from OpenVidu Server to clients, in kbps.
# 0 means unconstrained
OPENVIDU_STREAMS_VIDEO_MAX_SEND_BANDWIDTH=1000

# Minimum video bandwidth sent from OpenVidu Server to clients, in kbps.
# 0 means unconstrained
OPENVIDU_STREAMS_VIDEO_MIN_SEND_BANDWIDTH=300

# All sessions of OpenVidu will try to force this codec. If OPENVIDU_STREAMS_ALLOW_TRANSCODING=true
# when a codec can not be forced, transcoding will be allowed
# Values: MEDIA_SERVER_PREFERRED, NONE, VP8, VP9, H264
# Default value is MEDIA_SERVER_PREFERRED
# OPENVIDU_STREAMS_FORCED_VIDEO_CODEC=MEDIA_SERVER_PREFERRED

# Allow transcoding if codec specified in OPENVIDU_STREAMS_FORCED_VIDEO_CODEC can not be applied
# Values: true | false
# Default value is false
# OPENVIDU_STREAMS_ALLOW_TRANSCODING=false

# true to enable OpenVidu Webhook service. false' otherwise
# Values: true | false
OPENVIDU_WEBHOOK=true

# HTTP endpoint where OpenVidu Server will send Webhook HTTP POST messages
# Must be a valid URL: http(s)://ENDPOINT
OPENVIDU_WEBHOOK_ENDPOINT=웹훅 엔드포인트

# List of headers that OpenVidu Webhook service will attach to HTTP POST messages
#OPENVIDU_WEBHOOK_HEADERS=

# List of events that will be sent by OpenVidu Webhook service
# Default value is all available events
OPENVIDU_WEBHOOK_EVENTS=[sessionCreated,sessionDestroyed,participantJoined,participantLeft,webrtcConnectionCreated,webrtcConnectionDest
```

```
# How often the garbage collector of non active sessions runs.
# This helps cleaning up sessions that have been initialized through
# REST API (and maybe tokens have been created for them) but have had no users connected.
# Default to 900s (15 mins). 0 to disable non active sessions garbage collector
OPENVIDU_SESSIONS_GARBAGE_INTERVAL=900

# Minimum time in seconds that a non active session must have been in existence
# for the garbage collector of non active sessions to remove it. Default to 3600s (1 hour).
# If non active sessions garbage collector is disabled
# (property 'OPENVIDU_SESSIONS_GARBAGE_INTERVAL' to 0) this property is ignored
OPENVIDU_SESSIONS_GARBAGE_THRESHOLD=3600

# Call Detail Record enabled
# Whether to enable Call Detail Record or not
# Values: true | false
OPENVIDU_CDR=false

# Path where the cdr log files are hosted
OPENVIDU_CDR_PATH=/opt/openvidu/cdr

# Kurento Media Server image
# --------------------------
# Docker hub kurento media server: https://hub.docker.com/r/kurento/kurento-media-server
# Uncomment the next line and define this variable with KMS image that you want use
# KMS_IMAGE=kurento/kurento-media-server:6.16.0

# Kurento Media Server Level logs
# -------------------------------
# Uncomment the next line and define this variable to change
# the verbosity level of the logs of KMS
# Documentation: https://doc-kurento.readthedocs.io/en/stable/features/logging.html
# KMS_DOCKER_ENV_GST_DEBUG=

# Openvidu Server Level logs
# --------------------------
# Uncomment the next line and define this variable to change
# the verbosity level of the logs of Openvidu Service
# RECOMENDED VALUES: INFO for normal logs DEBUG for more verbose logs
# OV_CE_DEBUG_LEVEL=INFO

# Java Options
# ------------
# Uncomment the next line and define this to add
# options to java command
# Documentation: https://docs.oracle.com/cd/E37116_01/install.111210/e23737/configuring_jvm.htm#OUDIG00058
# JAVA_OPTIONS=-Xms2048m -Xmx4096m -Duser.timezone=UTC
```

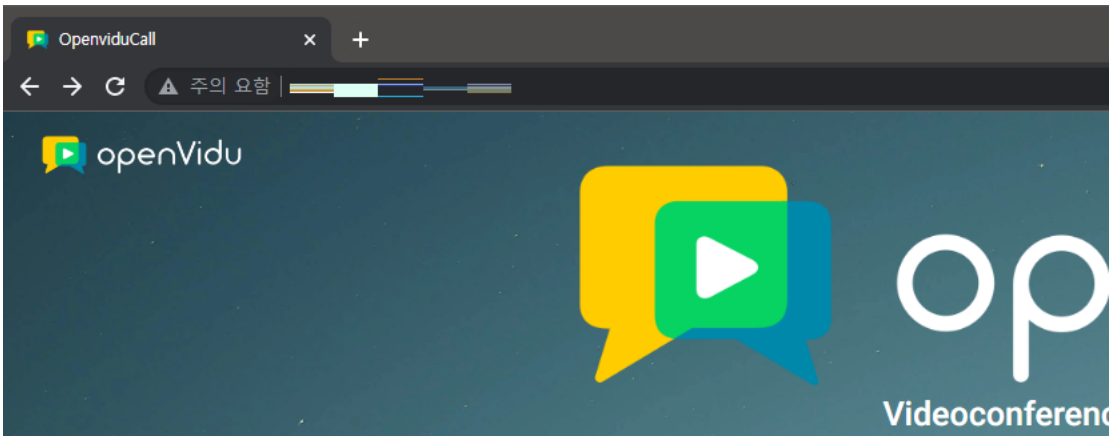
설정이 완료되었다면 오픈비두를 실행시킨다.

```
./openvidu start
```

그리고, 컨테이너 5개가 다 켜진 것을 확인한다.

```
ubuntu@ip-아이피:~$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS                  PORTS
f0714d619357   openvidu/openvidu-coturn:2.22.0      "docker-entrypoint.s…"   38 hours ago   Up 38 hours             0.0.0.0:3478->3478/tcp
188d92eb89ab   openvidu/openvidu-proxy:2.22.0       "/docker-entrypoint.…"   38 hours ago   Up 38 hours
23196c96d6df   kurento/kurento-media-server:6.16.0   "/entrypoint.sh"         38 hours ago   Up 38 hours (healthy)
1e037202b91f   openvidu/openvidu-call:2.22.0        "docker-entrypoint.s…"   38 hours ago   Up 38 hours
07e382e7900c   openvidu/openvidu-server:2.22.0      "/usr/local/bin/entr…"   38 hours ago   Up 38 hours
```

도메인 주소에 접근해서 오픈비두 작동을 테스트한다.



테스트가 완료되면 nginx 컨테이너, app 컨테이너를 종료시켜야 한다. 각 컨테이너의 이미지 이름은 다음과 같다.

- openvidu/openvidu-proxy:2.22.0
- openvidu/openvidu-proxy:2.22.0

3. 빌드

3-1 수동 빌드

자동 배포 시스템을 구축하기 위해서는, 프로젝트가 어떤 식으로 빌드되고, 실행되는 지를 알아야한다. 자동 빌드의 결과도 수동 빌드와 같기 때문에 수동 빌드 과정은 최소 1번은 필요하다.

프론트엔드 빌드

0. 빌드 환경 및 버전 체크

```
node --version
v16.16.0

npm -v
8.11.0
```

1. 사용 라이브러리

```
"dependencies": {
    "@ckeditor/ckeditor5-build-classic": "^35.0.1",
    "@ckeditor/ckeditor5-react": "^3.0.3",
    "@emotion/react": "^11.9.3",
    "@emotion/styled": "^11.9.3",
    "@fortawesome/fontawesome-svg-core": "^6.1.2",
    "@fortawesome/free-brands-svg-icons": "^6.1.2",
    "@fortawesome/free-regular-svg-icons": "^6.1.2",
    "@fortawesome/free-solid-svg-icons": "^6.1.2",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@mui/icons-material": "^5.8.4",
    "@mui/material": "^5.9.1",
    "@reduxjs/toolkit": "^1.8.3",
    "apexcharts": "^3.35.4",
    "axios": "^0.27.2",
    "dayjs": "^1.11.5",
```

```
    "dotenv": "^16.0.1",
    "jwt-decode": "^3.1.2",
    "lodash": "^4.17.21",
    "openvidu-browser": "^2.22.0",
    "openvidu-react": "^2.22.0",
    "react": "^17.0.2",
    "react-apexcharts": "^1.4.0",
    "react-dom": "^17.0.2",
    "react-redux": "^8.0.2",
    "react-router": "^6.3.0",
    "react-router-dom": "^6.3.0",
    "react-time-sync": "^5.2.1",
    "redux": "^4.2.0",
    "redux-persist": "^6.0.0",
    "redux-promise": "^0.6.0",
    "redux-thunk": "^2.4.1"
  },
  "devDependencies": {
    "@babel/core": "^7.18.9",
    "@babel/preset-env": "^7.18.9",
    "@babel/preset-react": "^7.18.6",
    "@pmmmwh/react-refresh-webpack-plugin": "^0.5.7",
    "babel-loader": "^8.2.5",
    "css-loader": "^6.7.1",
    "dotenv-webpack": "^8.0.0",
    "eslint": "^8.20.0",
    "eslint-config-airbnb": "^19.0.4",
    "eslint-plugin-import": "^2.26.0",
    "eslint-plugin-jsx-a11y": "^6.6.0",
    "eslint-plugin-react": "^7.30.1",
    "eslint-plugin-react-hooks": "^4.6.0",
    "file-loader": "^6.2.0",
    "html-webpack-plugin": "^5.5.0",
    "react-refresh": "^0.14.0",
    "sass": "^1.53.0",
    "sass-loader": "^13.0.2",
    "style-loader": "^3.3.1",
    "webpack": "^5.73.0",
    "webpack-cli": "^4.10.0",
    "webpack-dev-server": "^4.9.3"
  }
```

.gitignore

```
# Logs
logs
*.log

# Runtime data
pids
*.pid
*.seed

# Coverage directory used by tools like istanbul
coverage
.eslintcache

# Dependency directory
# https://www.npmjs.org/doc/misc/npm-faq.html#should-i-check-my-node_modules-folder-into-git
node_modules

# OSX
.DS_Store

release/app/dist
release/build
.erb/dll

.idea
npm-debug.log.*
*.css.d.ts
*.sass.d.ts
*.scss.d.ts

.env
```

```
.editorconfig
.erb/
.eslintignore
.eslintrc.js
.gitattributes
.github/
.husky/
.vscode/
release/
```

2. 원리 이해하기

React는 서버 사이드 렌더링이 아닌, 클라이언트 사이드 렌더링이다. 또한 SPA이기 때문에 HTML파일 하나와 JS파일 하나 이상 필요하다.

개발 환경에서 수 많은 JS파일을 작성하고, 로컬 환경에서 테스트했을 것이다. Webpack은 우리가 작성한 파일들을 한 곳에 모아 하나 혹은 여러개의 JS 파일로 바꾸어 웹 브라우저에서 실행되도록 만들어준다.

webpack.config.js

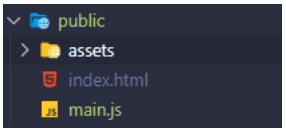
```
output: {
    filename: 'main.js', // 빌드 후의 파일 이름
    path: path.join(__dirname, '/dist'), // 빌드 파일이 저장될 곳
},
```

빌드가 된 main.js 파일은 public 폴더의 index.html 에서 사용한다.

index.html

```
<body>
  <div id="root"></div>
  <script src="/main.js"></script>
</body>
```

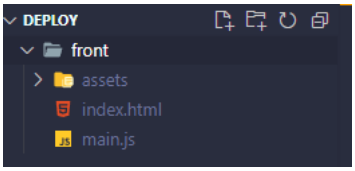
그래서 main.js 파일을 public 폴더로 이동시켜 줘야한다. 최종적으로 사용할 파일 폴더는 public 폴더이다.



- main.js : 빌드 된 리액트
- index.html: main.js 를 보여줄 html 파일
- assets: 정적 파일

3. 폴더 이동하기

로컬의 아무 곳에나 deploy 라는 폴더를 만든다. 그 안에 front 라는 폴더를 만든 후, public 폴더 안의 내용을 front 폴더 안으로 이동시킨다.



백엔드 빌드

0. 빌드 환경 및 버전 체크하기

개발도구는 IntelliJ 2021.3.1 을 이용했으며 JDK는 1.8 (Zulu 8.0) 을 사용했다.

빌드 도구는 Gradle을 사용했다.

Spring Boot 버전은 2.7.1을 사용했다.

실행에 필요한 properties를 사전에 설정해줘야한다. 구글, 카카오, 네이버 소셜로그인을 사용하므로 각 서비스에서 앱을 등록해서 클라이언트 아이디와 시크릿을 발급받아야 한다.

application.properties

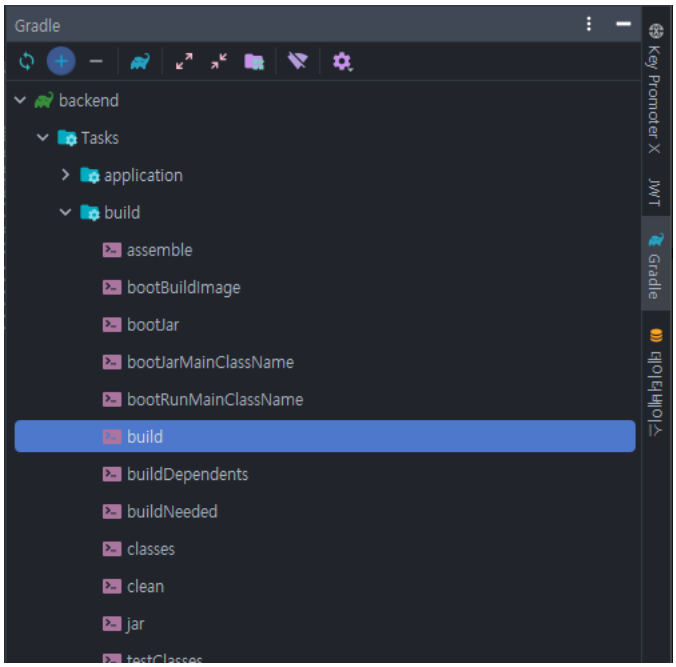
```
build.date=@build.date@
server.port=8080
server.address=localhost
server.servlet.contextPath=/api/v1

server.servlet.encoding.charset=UTF-8
server.servlet.encoding.enabled=true
server.servlet.encoding.force=true

## JPA
spring.jpa.hibernate.naming.implicit-strategy=org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.jdbc.time_zone=UTC

## DB Setting
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/webdb?serverTimezone=UTC&useLegacyDatetimeCode=false&useUnicode=true&characterEncoding=
utf8
spring.datasource.username=ssafy104
spring.datasource.password=ssafy1357

## Redis Setting
spring.redis.host=localhost
spring.redis.port=6379


## JWT Setting
jwt.secret-key=sasdKAnzkaAlfmFimmAzsQkgYodiuqlxw
# 0.5h
jwt.access-token-expire-time=1800000
jwt.refresh-token-expire-time=1209600000

password.reset.expire-time=300000

## response Setting
response.success=success
response.fail=fail

## Swagger Setting
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

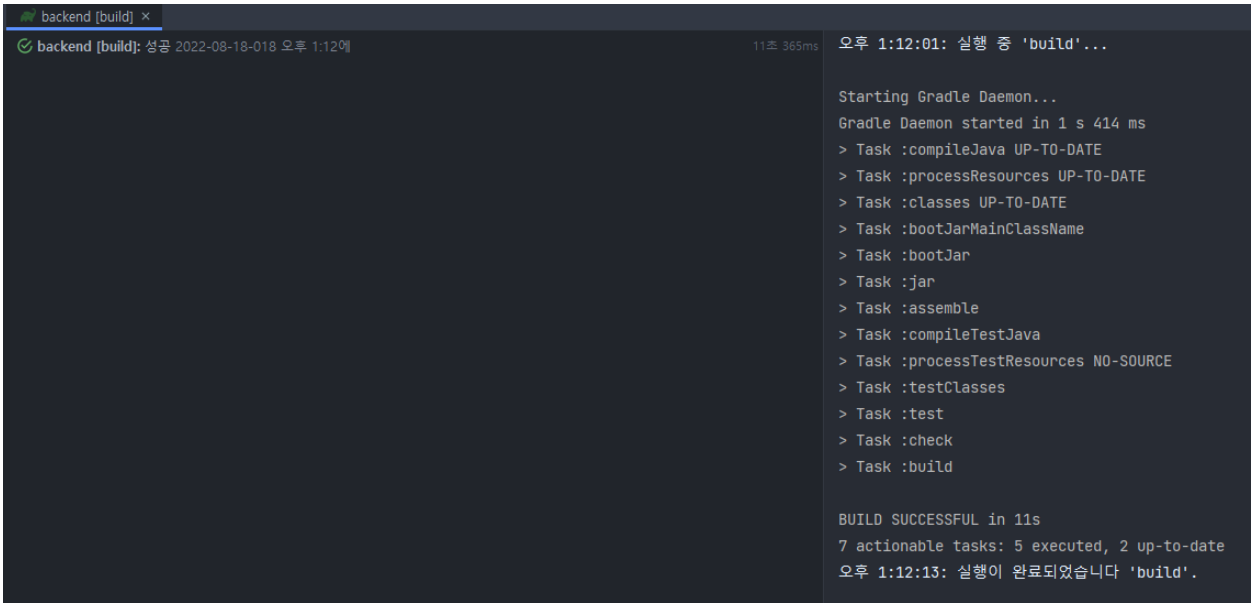
```
## SMTP Setting
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=Gmail계정
spring.mail.password=비밀번호
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
spring.mail.properties.mail.smtp.starttls.enable=true

logging.level.org.hibernate.type.descriptor.sql=trace
logging.level.root=WARN
logging.level.com.ssafy.backend=debug

#email variable
email.expire-day=1
email.auth-key-size=20

#Frontend
client.url=http://localhost:3000/user/auth

#OpenVidu
openvidu.url=https://localhost:4443/
openvidu.secret=MY_SECRET

#Riot Api
riot.url=https://kr.api.riotgames.com
riot.asia-url=https://asia.api.riotgames.com
riot.api-key=RGAPI-8b25c0de-10fc-4881-af61-f0f3a41b4a66
riot.game-count=10

##Aws S3
cloud.aws.credentials.access-key=AKIAT47G4FGKUGVQ6ZRA
cloud.aws.credentials.secret-key=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
cloud.aws.stack.auto=false

#cloud.aws.s3.bucket=okttа
cloud.aws.region.static=us-east-2

# multipart
spring.servlet.multipart.max-file-size=5MB
spring.servlet.multipart.max-request-size=5MB

# google oauth
spring.security.oauth2.client.registration.google.client-id=클라이언트 아이디
spring.security.oauth2.client.registration.google.client-secret=클라이언트 시크릿
spring.security.oauth2.client.registration.google.scope=profile, email

# naver oauth
spring.security.oauth2.client.registration.naver.client-id=클라이언트 아이디
spring.security.oauth2.client.registration.naver.client-secret=클라이언트 시크릿
spring.security.oauth2.client.registration.naver.client-authentication-method=post
spring.security.oauth2.client.registration.naver.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.naver.redirect-uri=http://localhost:8080/api/v1/login/oauth2/code/naver
spring.security.oauth2.client.registration.naver.scope=email
spring.security.oauth2.client.registration.naver.client-name=Naver
spring.security.oauth2.client.provider.naver.authorization-uri=https://nid.naver.com/oauth2.0/authorize
spring.security.oauth2.client.provider.naver.token-uri=https://nid.naver.com/oauth2.0/token
spring.security.oauth2.client.provider.naver.user-info-uri=https://openapi.naver.com/v1/nid/me
spring.security.oauth2.client.provider.naver.user-name-attribute=response

# kakao oauth
spring.security.oauth2.client.registration.kakao.client-id=클라이언트 아이디
spring.security.oauth2.client.registration.kakao.client-secret=클라이언트 시크릿
spring.security.oauth2.client.registration.kakao.client-authentication-method=post
spring.security.oauth2.client.registration.kakao.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.kakao.redirect-uri=http://localhost:8080/api/v1/login/oauth2/code/kakao
spring.security.oauth2.client.registration.kakao.scope=account_email
spring.security.oauth2.client.registration.kakao.client-name=Kakao
spring.security.oauth2.client.provider.kakao.authorization-uri=https://kauth.kakao.com/oauth/authorize
spring.security.oauth2.client.provider.kakao.token-uri=https://kauth.kakao.com/oauth/token
spring.security.oauth2.client.provider.kakao.user-info-uri=https://kapi.kakao.com/v2/user/me
spring.security.oauth2.client.provider.kakao.user-name-attribute=id

# oauth
app.oauth2.authorizedUris=http://localhost:3000/oauth/redirect

# Paging Limit
pagingLimit = 10
myLimit = 5
```

```
frontend = http://localhost:3000

# default image url
default-profile-image-url=https://oktta.s3.us-east-2.amazonaws.com/defaultProfile.png
```

1. 빌드 하기

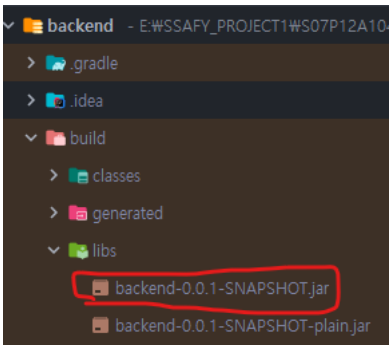
사진에서 클릭된 build 아이콘을 더블클릭하면 빌드가 실행된다.





빌드 성공

빌드 폴더 위치는 다음과 같다.

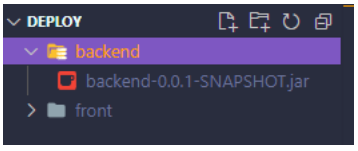


다음과 같은 명령어로 jar 파일을 실행시킬 수 있다.

```
java -jar backend-0.0.1-SNAPSHOT.jar
```

2. 폴더 이동하기

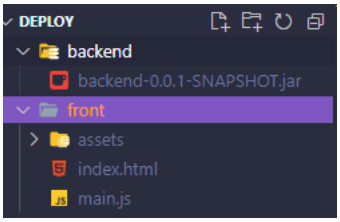
프론트엔드 빌드 파일을 이동시킨 것처럼 jar 파일을 이동시킨다. 폴더 명 backend 에 jar 파일을 넣어서 다음과 같은 구조가 되게 만든다.



3-2 자동 빌드

젠킨스를 통해 진행하기 때문에 뒤에서 설명한다.

프론트엔드와 백엔드 빌드가 완료되었다면, 다음과 같은 구조를 지닌 폴더가 존재해야 한다.



---

4. 배포

4-1 도커 파일 작성

컨테이너를 만들기 위한 필요한 이미지 파일을 제작하기 위해서는 Dockerfile 작성이 필요하다. Nginx와 백엔드 컨테이너 만 이미지 파일 제작이 필요하다.

백엔드

backend 폴더 내부에 Dockerfile 을 생성하고 내용을 적어준다.

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080/tcp
ENTRYPOINT ["java","-jar","-Dspring.profiles.active=prod","/app.jar"]
```

Nginx

nginx 폴더를 만들고 Dockerfile 을 생성한다.

```
FROM nginx:stable-alpine
RUN mkdir /front
COPY ./default.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
```

2번째 줄에서 만든 front 폴더는 프론트엔드 빌드 파일이 저장되는 폴더이다.

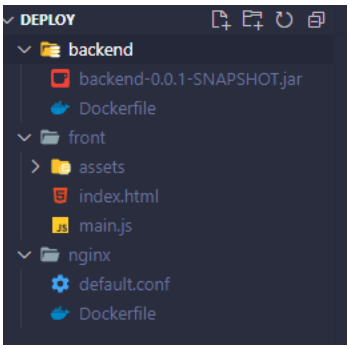
그리고, Nginx는 설정파일이 필요하기 때문에 같은 경로에 설정파일을 default.conf라는 이름으로 작성한다.

```
upstream backend {
    server localhost:8080;
}

upstream openviduserver {
```

```
    server localhost:5443;
}

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

    server_name i7a104.p.ssafy.io;

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

server {
    listen 443 ssl default_server;
  listen [::]:443 ssl default_server;


    server_name  i7a104.p.ssafy.io;
    access_log   /var/log/nginx/nginx.vhost.access.log;
    error_log    /var/log/nginx/nginx.vhost.error.log;

    ssl                  on;
    ssl_certificate      /cert/certificate.crt;
    ssl_certificate_key  /cert/private.key;

    # Proxy
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Proto https;
    proxy_headers_hash_bucket_size 512;
    proxy_redirect off;

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    location ~ /openvidu$ {
        proxy_pass http://openviduserver;
    }

    location / {
        root /front;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
  }

    location /api {
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
        proxy_pass http://backend;
    }

     location /dashboard {
        allow all;
        # deny all;
        proxy_pass http://openviduserver;
    }

    location ~/.well-known/pki-validation {
    default_type "text/plain";
    index .txt;
  }

    location /layouts/custom {
        rewrite ^/layouts/custom/(.*)$ /custom-layout/$1 break;
        root /opt/openvidu;
    }

    location /recordings {
        proxy_pass http://openviduserver;
    }

    location /openvidu/layouts {
        rewrite ^/openvidu/layouts/(.*)$ /custom-layout/$1 break;
        root /opt/openvidu;
    }

    location /openvidu/recordings {
        proxy_pass http://openviduserver;
```

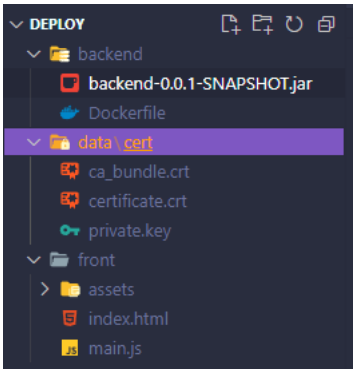
```
    }
}
```

2개의 파일을 작성하고 나면 폴더 구조는 다음과 같다.



4-2 SSL 인증서 발급

지급받은 crt 파일과 key 파일을 아래 사진처럼 폴더를 만들어서 넣어준다.



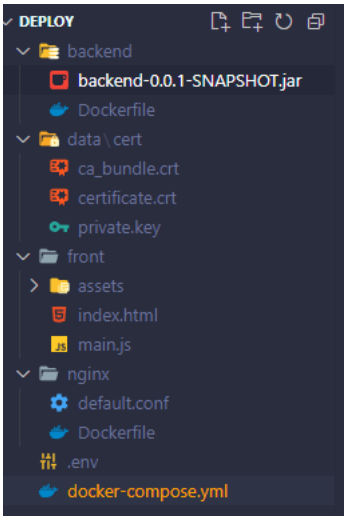
4-3 Docker-compose.yml 작성

```
version: '3.7'
services:
  nginx:
    network_mode: host
    container_name: nginx
    build:
      dockerfile: Dockerfile
      context: ./nginx
    expose:
      - "80"
    volumes:
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/oktta/deploy/front:/front
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/oktta/deploy/data/cert:/cert
      - /opt/openvidu/custom-layout:/opt/openvidu/custom-layout
  db:
    image: mysql:8.0.28
    network_mode: host
    expose:
      - "3306"
    container_name: db
```

```
    volumes:
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/oktta/deploy/db/conf.d:/etc/mysql/conf.d
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/oktta/deploy/db/data:/var/lib/mysql
      - /home/ubuntu/jenkins_build/jenkins_home/workspace/oktta/deploy/db/initdb.d:/docker-entrypoint-initdb.d
    environment:
      MYSQL_DATABASE: webdb
      MYSQL_ROOT_PASSWORD: "eq4jafgo!23QzG05^^2a"
  backend:
    container_name: backend
    network_mode: host
    restart: on-failure
    build:
      dockerfile: Dockerfile
      context: ./backend
    expose:
      - "8080"
    environment:
      SERVER_PORT: 8080
      SPRING_DATASOURCE_URL: jdbc:mysql://localhost:3306/webdb?serverTimezone=UTC&useLegacyDatetimeCode=false&useUnicode=true&characterEnco
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: "eq4jafgo!23QzG05^^2a"
      SPRING_REDIS_HOST: localhost
      OPENVIDU_URL: http://localhost:5443/
      OPENVIDU_SECRET: "MY_SECRET"
    depends_on:
      - db
      - redis_boot
  redis_boot:
    image: redis:alpine
    network_mode: host
    command: redis-server --port 9999
    container_name: redis_boot
    hostname: redis_boot
    labels:
      - "name=redis"
      - "mode=standalone"
    expose:
      - "9999"
```

순서대로 Nginx, Mysql, Spring Boot, Redis 컨테이너이다. Volumes 같은 경우는 Jenkins에서 이 파일을 실행시키기 때문에 `절대경로`로 적어줘야 한다.

4.4 최종 결과



이 폴더들을 Jenkins에 올려서 자동 CI/CD 할 예정이다. 위에서 설명안한 .env 파일이 있는데, 프론트엔드 빌드에 사용된다.

5. 젠킨스

EC2에는 도커와 도커 컴포즈 이외에는 아무것도 설치하지 않았다. 그래서 젠킨스도 도커 컨테이너를 통해 실행한다.

5-1 젠킨스 설치

이미지 파일을 내려받는다

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

Dockerfile 을 작성한다.

```
FROM jenkins/jenkins:lts

USER root

# install docker
RUN 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
```

docker-compose.yml 을 작성한다.

```
version: '3.7'
services:
  jenkins:
    build:
      context: .
    container_name: jenkins
    user: root
    privileged: true
    ports:
      - 3333:8080
      - 50000:50000
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
```

젠킨스 포트번호를 3333으로 정했다.

젠킨스 폴더 구조는 다음과 같다.

```
total 20
drwxrwxr-x  3 ubuntu ubuntu 4096 Aug 15 06:09 .
drwxr-xr-x 11 ubuntu ubuntu 4096 Aug 18 14:31 ..
-rwxrwxr-x  1 ubuntu ubuntu  610 Aug 15 02:15 Dockerfile
-rwxrwxr-x  1 ubuntu ubuntu  293 Aug 15 02:15 docker-compose.yml
```

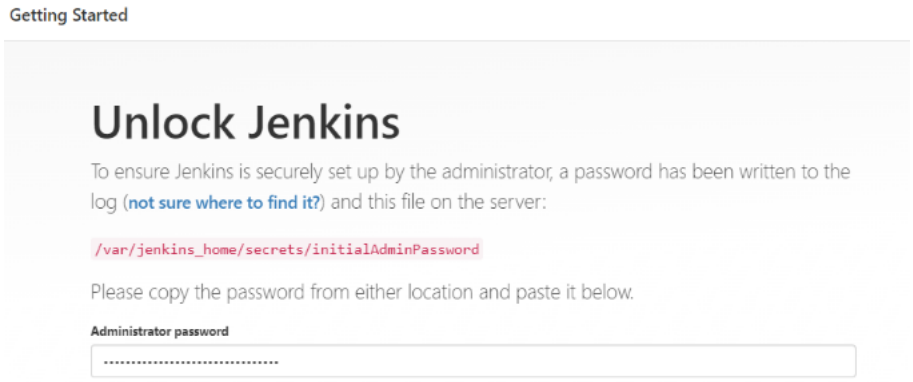
작성이 끝났다면 도커 컴포즈를 실행시킨다.

```
sudo docker-compose up
```

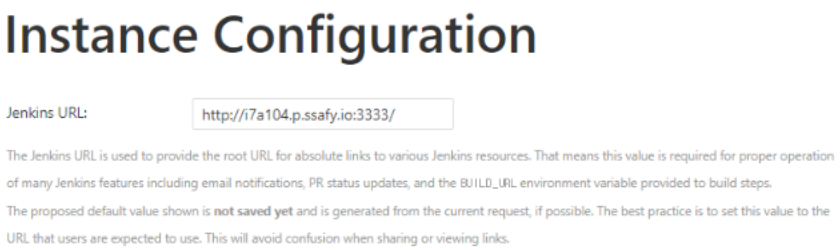
젠킨스 설정을 위해 비밀번호가 필요하므로, 로그 중에 Password 를 찾는다.

```
jenkins    |
jenkins    | *************************************************************
jenkins    | *************************************************************
jenkins    | *************************************************************
jenkins    |
jenkins    | Jenkins initial setup is required. An admin user has been created and a password generated.
jenkins    | Please use the following password to proceed to installation:
jenkins    |
jenkins    | 49f0ef37765147238d610bd2ccd7ceb2
jenkins    |
jenkins    | This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
jenkins    |
jenkins    | *************************************************************
jenkins    | *************************************************************
jenkins    | *************************************************************
jenkins    |
```

도메인:포트번호 로 접속해서 로그인 한다.

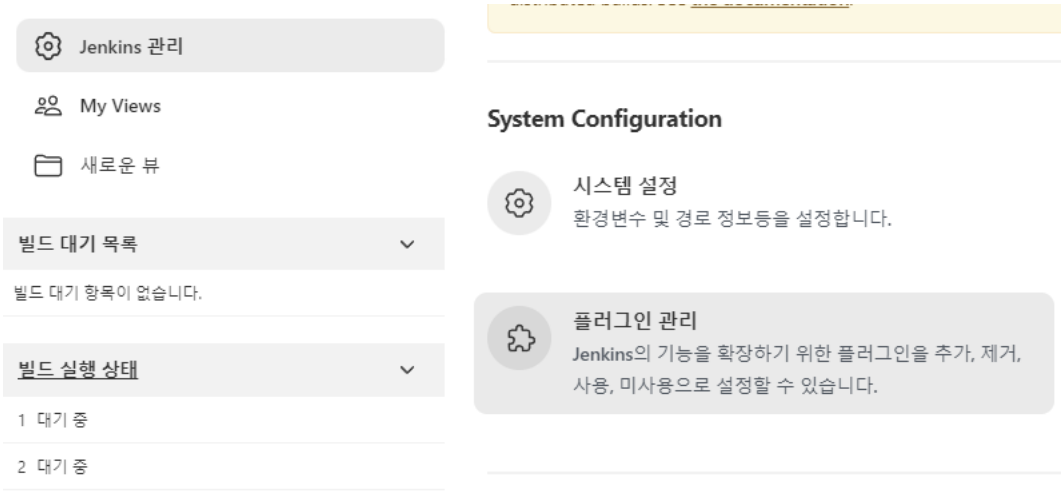


잘 설치되었는지 확인한다.

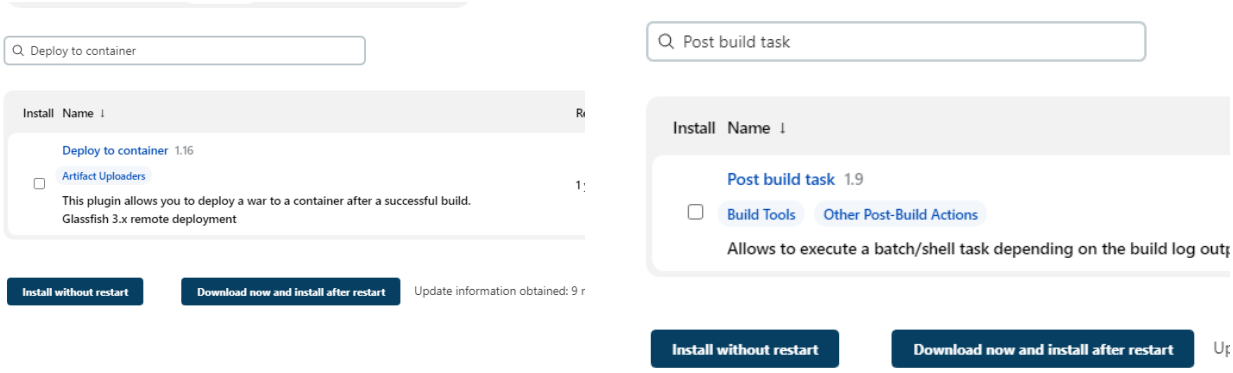


5-2 젠킨스 세팅

Jenkins 관리에 들어가서 플러그인 관리를 선택한다

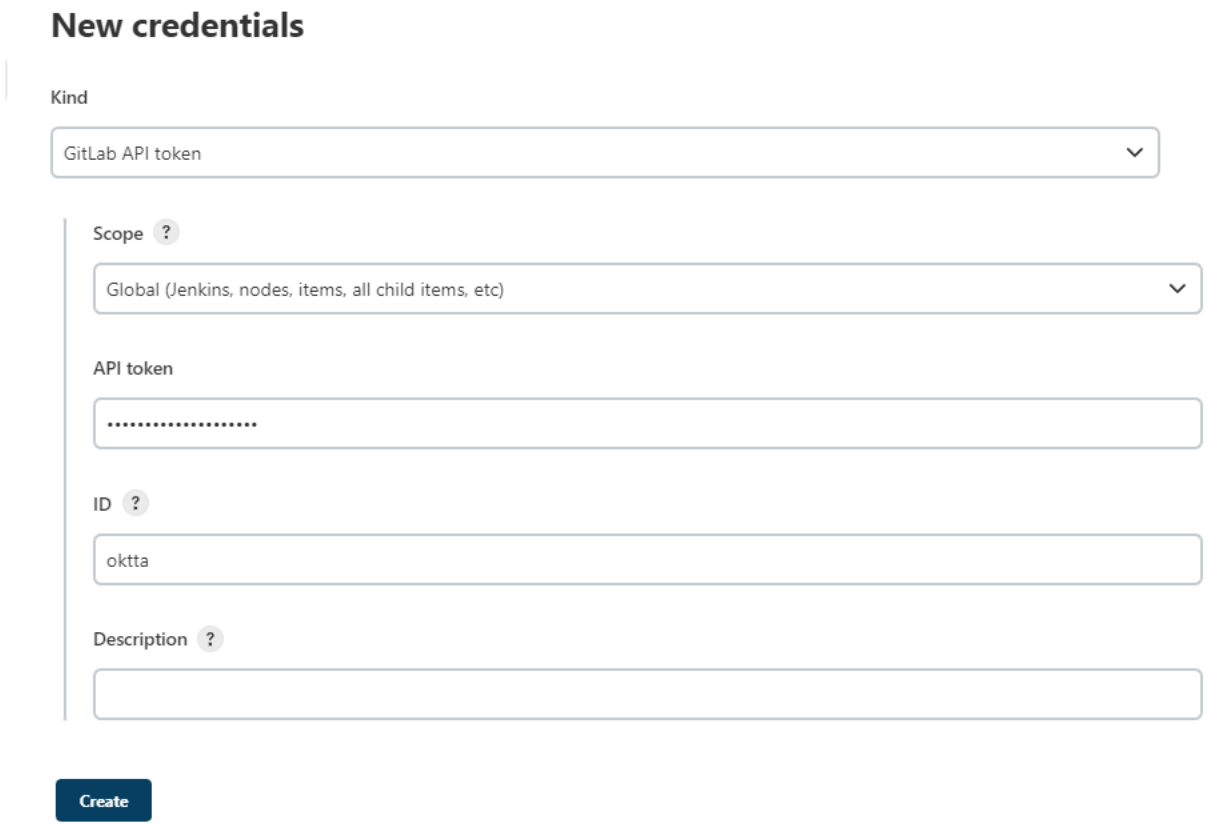


'**Deploy to container**'와 '**Post build task**' 플러그인을 **재시작 없이** 설치한다.

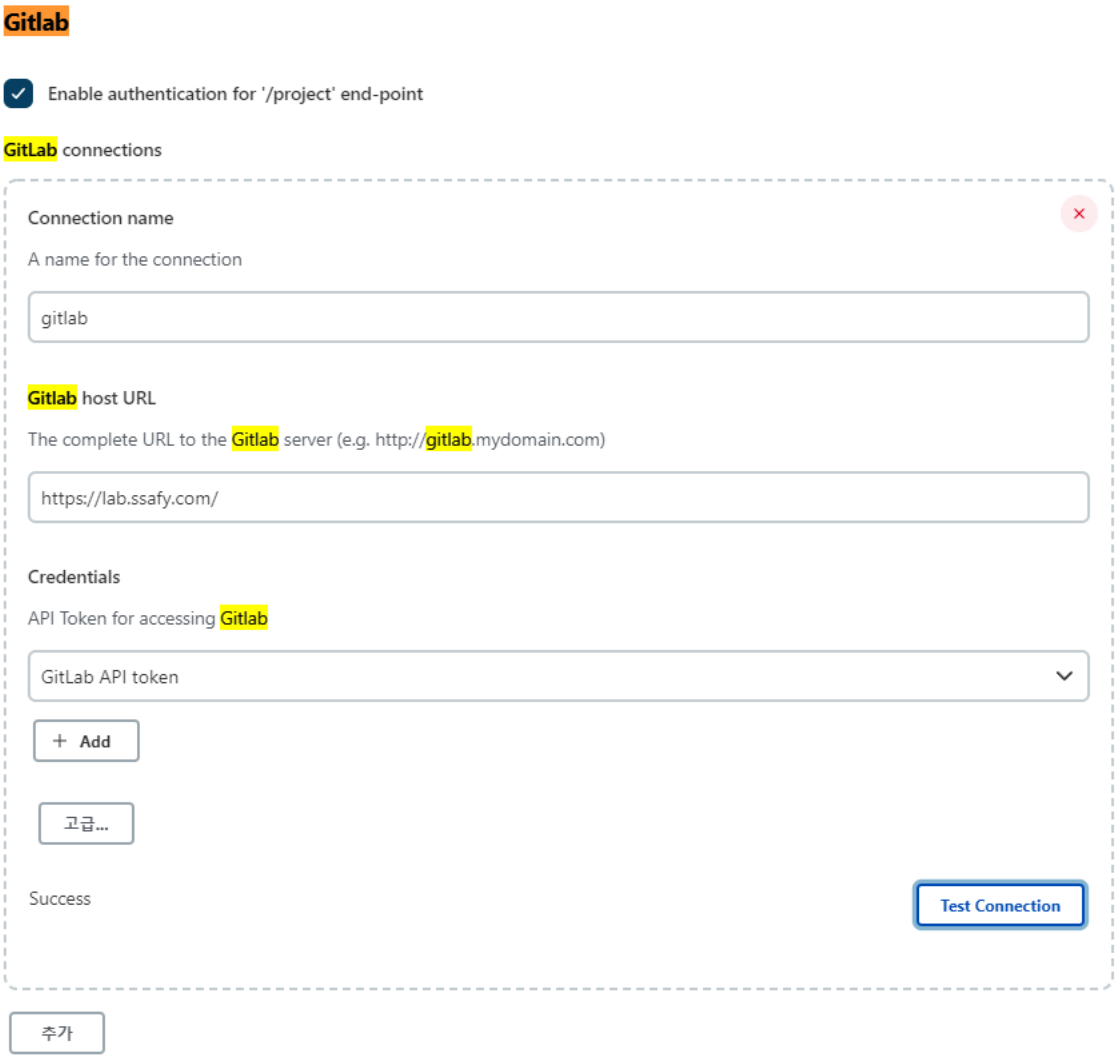


5-3 새 프로젝트 생성

새 프로젝트를 만들기 전에, GItLab 연동을 위해 API Token을 발급받는다.



그리고 프로젝트 설정에서, GitLab 연결을 마무리한다.



백엔드 빌드를 위한 Gradle 설정과 WebHook 설정을 한다.

빌드 순서는 다음과 같다.

1. Gradle 빌드
2. Node JS를 이용한 프론트엔드 빌드
3. Shell 명령어 실행

```
\cp ${WORKSPACE}/deploy/.env ${WORKSPACE}/frontend/.env &&
cd ${WORKSPACE}/frontend &&
npm install &&
npm run build &&
\cp public/index.html dist/index.html &&
\cp -r public/assets dist/assets &&
cd ${WORKSPACE} &&
rm -r deploy/dist || true;
\mv frontend/dist deploy/ &&
rm -r deploy/front || true;
\mv deploy/dist deploy/front
```

빌드 후 명령어에는 아래 처럼 작성한다.

```
\mv /var/jenkins_home/workspace/oktta/backend/build/libs/backend-0.0.1-SNAPSHOT.jar /var/jenkins_home/workspace/oktta/deploy/backend &&
cd ${WORKSPACE}/deploy &&
/usr/local/bin/docker-compose down -v &&
/usr/local/bin/docker-compose up -d
```

5-4 도커 설정

젠킨스와 도커는 연결되었지만, Docker Compose 는 아니다. 그래서 따로 설치해댜 한다.

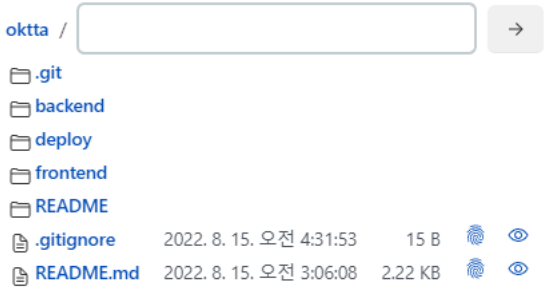
Jenkins 컨테이너 내부에서 아래 명령어들을 순서대로 입력한다.

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compos
```

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

5-5 작업공간으로 배포 파일 이동시키기

4번 과정에서 작성한 폴더를 작업공간 가장 위로 이동시킨다. 그러면 작업 공간은 다음과 같이 나온다.



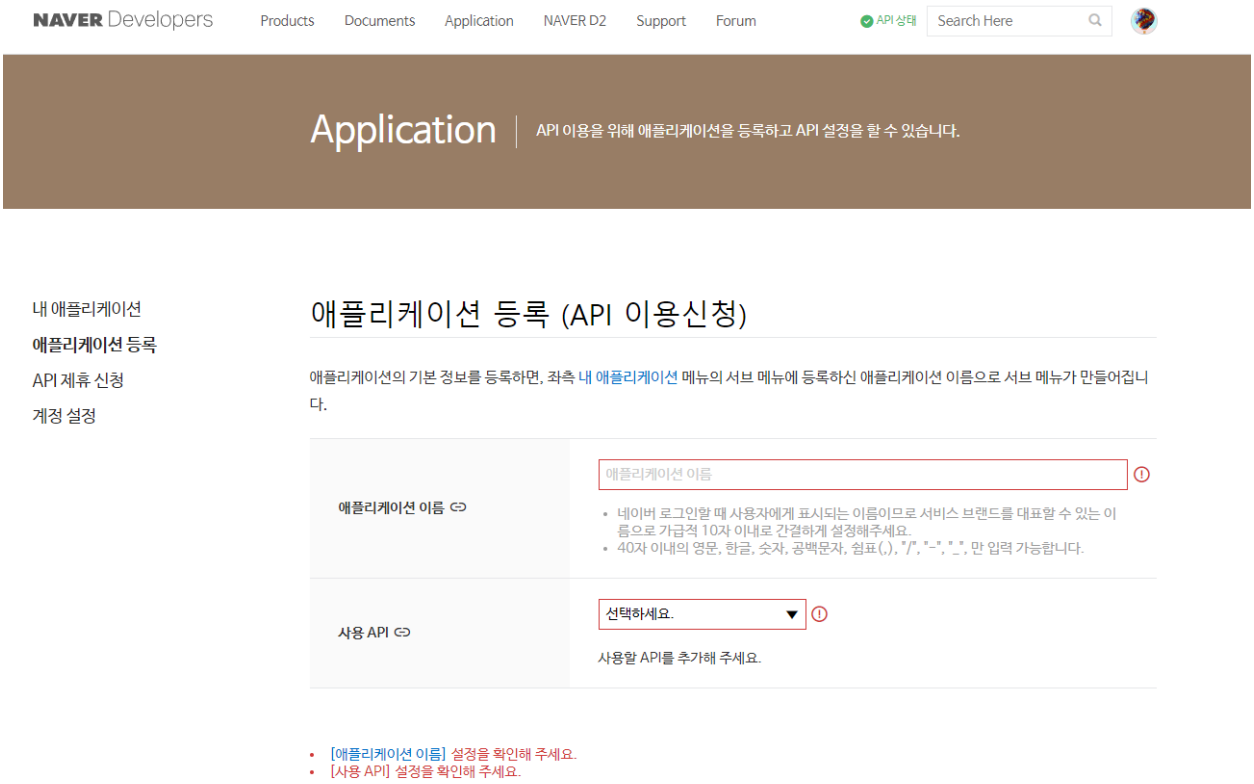
5-6 빌드하기

WebHook을 연결했으므로, Master 브랜치로 Merge 하면 자동으로 빌드 및 배포가 된다. 혹은, 젠킨스에서 빌드 버튼을 통해 빌드 할 수 있다.

외부 서비스 설정하기

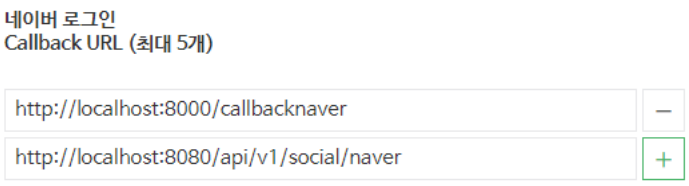
네이버 소셜 로그인

네이버 개발자 사이트에서 애플리케이션을 등록해준다. 사용 API는 네이버 로그인을 선택한다.



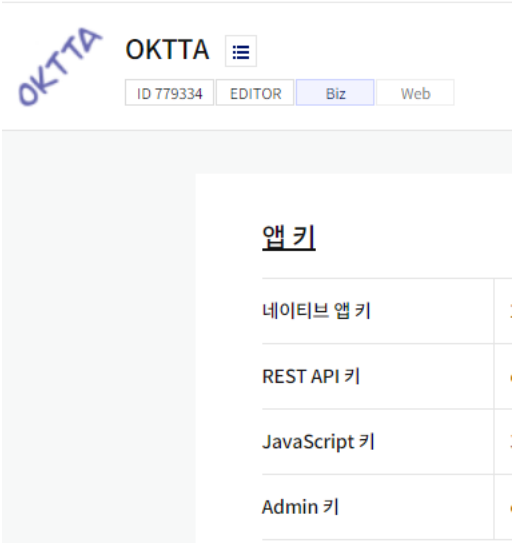
등록한 애플릭이션을 클릭해서 클라이언트 아이디와 시크릿을 확인한다.

콜백 URL을 설정하고 저장한다.

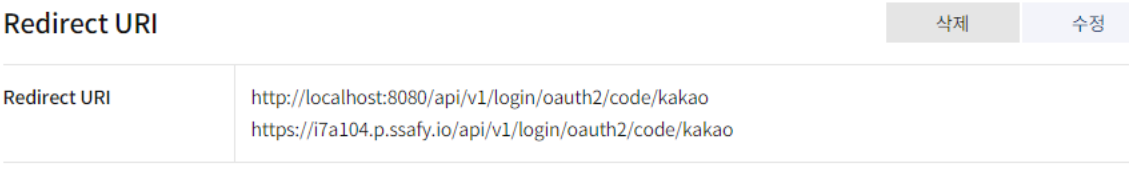


카카오 소셜 로그인

네이버와 마찬가지로 애플리케이션을 등록하고 발급받은 키를 확인한다.



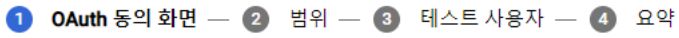
카카오 로그인을 활성화 시키고 Redirect URI를 설정한다.



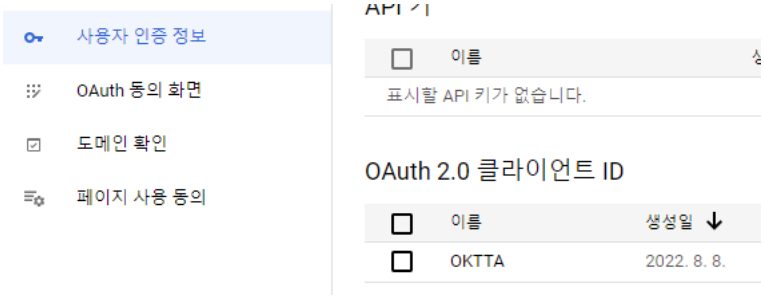
구글 소셜 로그인

1. Google API 콘솔에 접속한다.
2. 새 프로젝트 만들기
3. Google+ API를 사용한다
4. OAuth 동의화면을 구성한다.

5. 순서대로 설정한다.



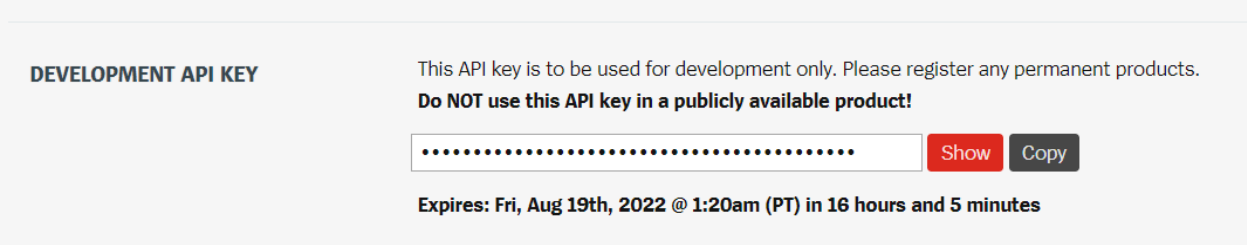
6. 사용자 인증 정보에서 클라이언트 ID를 생성한다.



7. 클라이언트 ID설정에서 리디렉션 URI도 설정해준다.

RIOT API

라이엇 개발자 사이트에서 회원 가입 후, 개발용 API키를 발급받는다.



개발용 키는 24시간 후 만료되기 때문에 매일 갱신시켜줘야 한다.